МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри, канд. техн. наук,
доцент _____ Є. О. Давиденко
підпис
«___» _____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

ВЕБЗАСТОСУНОК ОЦІНЮВАННЯ НАВЧАЛЬНИХ ДОСЯГНЕНЬ УЧНІВ ЗАКЛАДІВ ЗАГАЛЬНОЇ СЕРЕДНЬОЇ ОСВІТИ

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 409.22010908

Здобувач _____ М. В. Даниленко
підпис
«__» _____ 2024 р.

Керівник канд. техн. наук, доцент _____ Є. О. Давиденко
підпис
«__» _____ 2024 р.

Консультант канд. техн. наук, доцент _____ А. О. Алексєєва
підпис
«__» _____ 2024 р.

Миколаїв 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ

Зав. кафедри _____ Є. О. Давиденко

«_____» _____ 2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи бакалавра

Видано здобувачці групи 409 факультету комп'ютерних наук

Даниленко Марії Вадимівні

(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи

Вебзастосунок оцінювання навчальних досягнень учнів закладів загальної середньої освіти.

Затверджена наказом по ЧНУ від «22» грудня 2023 р. № 269

2. Строк представлення кваліфікаційної роботи «__._» _____ 20_ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні Очікуваним результатом є вебзастосунок оцінювання навчальних досягнень учнів закладів загальної, який буде надавати можливість для створення та проходження тестів.

4. Перелік питань, що підлягають розробці:

- проведення аналізу предметної галузі та систем із подібним функціоналом;
- моделювання необхідних для системи діаграм;
- обрання технологій реалізації програмного коду;
- розробка серверної частини проєкту;
- реалізування користувацького інтерфейсу;

5. Перелік графічних матеріалів

Презентація.

6. Завдання до спеціальної частини

Охорона праці вчителів та учнів в робочих приміщеннях закладів загальної середньої освіти.

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Алексєєва А. О	Кафедра екології	Спеціальна частина з охорони праці

Керівник роботи канд. техн. наук., доцент Давиденко Євген Олександрович
(посада, прізвище, ім'я, по батькові)

_____
(підпис)

Завдання прийнято до виконання

Даниленко Марія Вадимівна
(прізвище, ім'я, по батькові здобувача)

_____
(підпис)

Дата видачі завдання «____» _____ 20__ р

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: Вебзастосунок оцінювання навчальних досягнень учнів закладів загальної середньої освіти

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КРБ	15.01.2024 р.	19.01.2024 р.	Виконано
2.	Огляд літератури за темою роботи	22.01.2024 р.	26.01.2024 р.	Виконано
3.	Складання календарного плану КРБ	29.01.2024 р.	02.02.2024 р.	Виконано
4.	Аналіз предметної області	05.02.2024 р.	09.02.2024 р.	Виконано
5.	Розробка проєктних рішень	12.02.2024 р.	16.02.2024 р.	Виконано
6.	Моделювання та конструювання ПЗ	19.02.2024 р.	08.03.2024 р.	Виконано
7.	Кодування ПЗ	11.03.2024 р.	03.05.2024 р.	Виконано
8.	Тестування та апробація розробленого ПЗ	06.05.2024 р.	07.05.2024 р.	Виконано
9.	Розробка керівництва користувача	08.05.2024 р.	10.05.2024 р.	Виконано
10.	Розробка спеціальної частини з охорони праці	13.05.2024 р.	14.05.2024 р.	Виконано
11.	Оформлення КРБ та презентації	15.05.2024 р.	29.05.2024 р.	Виконано
12.	Відгук керівника КРБ	30.05.2024 р.	24.05.2024 р.	Виконано
13.	Попередній захист	03.06.2024 р.	03.06.2024 р.	Виконано
14.	Завершення оформлення КРБ та презентації	03.06.2024 р.	07.06.2024 р.	Виконано
15.	Рецензування	10.06.2024 р	11.06.2024 р.	Виконано
16.	Захист кваліфікаційної роботи	26.06.2024 р.	20.06.2024 р.	Виконано

Розробила здобувачка Даниленко М. В. _____
(прізвище, ім'я, по батькові) *(підпис)*

«___»_____20__ р.

Керівник роботи канд. техн. наук, доцент Давиденко Є. О. _____
(посада, прізвище, ім'я, по батькові) *(підпис)*

«___»_____20__ р.

АНОТАЦІЯ

до кваліфікаційної роботи бакалавра

«Вебзастосунок оцінювання навчальних досягнень учнів закладів загальної середньої освіти»

Здобувачка 409 гр.: Даниленко Марія Вадимівна

Керівник: канд. техн. наук, доцент Давиденко Є. О.

Кваліфікаційна робота бакалавра присвячена розробці вебзастосунку оцінювання навчальних досягнень учнів закладів загальної середньої освіти. Вона відповідає актуальній потребі викладачів у простому в користуванні та безкоштовному інструменту оцінювання знань учнів.

Об’єкт дослідження − процес оцінювання навчальних досягнень учнів закладів загальної середньої освіти.

Предмет дослідження − технології для організації підсумкового оцінювання навчальних досягнень учнів закладів загальної середньої освіти.

Метою кваліфікаційної роботи є розробка вебзастосунку оцінювання навчальних досягнень учнів для удосконалення внутрішньої системи забезпечення якості освіти в закладах загальної середньої освіти.

Кваліфікаційна робота бакалавра складається з вступу, 4 розділів, висновків та додатків.

У вступі визначається актуальність теми, предмет та об’єкт кваліфікаційної роботи та проводиться короткий огляд поставленої задачі.

У першому розділі проводиться аналіз предметної області, а саме огляд існуючих аналогів системи, визначення їх переваг та недоліків, на базі яких було сформовано специфікації вимог.

У другому розділі наведено етапи моделювання майбутнього застосунку створенням UML діаграм для візуалізації умов та вимог.

У третьому розділі описується процес проєктування вебзастосунку з використанням діаграм та детальним оглядом стеку застосованих технологій.

У четвертому розділі наведено опис користувацького інтерфейсу та програмної реалізації розробленої системи.

У висновках проводиться проведеної аналіз роботи та отриманих результатів.

Кваліфікаційна робота бакалавра викладена на 68 сторінок, вона містить 4 розділи, 43 ілюстрації, 5 таблиць, 36 джерел в переліку посилань.

Ключові слова: вебзастосунок, оцінювання навчальних досягнень, дистанційне навчання, загальна середня освіта, розробка програмного застосунку, база даних, користувацький інтерфейс.

ABSTRACT

of the Bachelor's Thesis

"Web application for evaluating educational achievements of students of general secondary education institutions"

Student of group 409: Danylenko Mariia Vadymivna

Supervisor: Candidate of Technical Sciences (Ph. D.), Associate Professor, Davydenko Y. O.

The qualification work of the bachelor is devoted to the development of a web application for evaluating educational achievements of students of general secondary education institutions. It meets the current need of teachers for an easy-to-use and free tool for assessing students' knowledge.

The object of the study is the process of assessment of academic achievements of students of general secondary education institutions.

The subject of the study is technologies for organising the final assessment of students' learning achievements in general secondary education institutions.

The purpose of the qualification work is the development of a web application for evaluating the educational achievements of students for the improvement of the internal system of ensuring the quality of education in general secondary education institutions.

The qualification work of the bachelor consists of an introduction, 4 chapters, conclusions and appendices.

The introduction defines the relevance of the topic, the subject and object of the qualification work, and provides a brief overview of the task.

The first section analyses the subject area, namely, a review of existing analogues of the system, identifying their advantages and disadvantages, on the basis of which the requirements specifications were formed.

The second chapter provides the steps for modeling a future application by creating UML diagrams to visualize conditions and requirements.

The third section describes the process of designing a web application using diagrams and a detailed overview of the stack of applied technologies.

The fourth section describes the user interface and software implementation of the developed system.

The conclusions provide an analysis of the work and the results obtained.

The qualification work of the bachelor is presented on 68 pages, it contains 4 sections, 43 illustrations, 5 tables, 36 sources in the list of references.

Keywords: web application, academic achievement assessment, distance learning, general secondary education, software application development, database, user interface.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД	–	база даних
ЗЗСО	−	заклад загальної середньої освіти
ІТ	−	інформаційні технології
МОНУ	−	Міністерство освіти і науки України
ОС	–	операційна система
ПЗ	–	програмне забезпечення
ТОВ	−	товариство з обмеженою відповідальністю
API	–	application programming interface
CRUD	–	create, read, update, delete
DOM	−	document object model
LINQ	−	language-integrated query
MS	−	Microsoft
REST	−	Representational State Transfer
JSON	−	JavaScript object motation
JWT	−	JSON Web token

ВСТУП

Сьогоденне навчання кардинально відрізняється від минулого століття. Швидкий розвиток технологій, зокрема ІТ-сфери, надав здобувачам освіти можливість набуття знань в інтерактивній формі, знаходячись в іншій частині світу, далеко від альма-матер. Це не лише відкриває ширші можливості учням, а й вчителям, полегшавши їм пошук інформації та підготовку до занять.

Імпульсом до стрімкого розвитку та всесвітнього поширення дистанційного навчання став 2020 рік через спалах пандемії коронавірусної хвороби. Велика кількість закладів освіти була вимушена перейти на онлайн формат навчання, покинувши рідні стіни на невизначений термін. І до сьогодні, через повномасштабне вторгнення російської федерації в Україну, більшість українських закладів загальної середньої освіти досі вимушена працювати у дистанційному та змішаному форматах.

В Україні відсутня вимога до використання єдиної платформи та інструменту дистанційного навчання, адже Законом України «Про освіту», держава гарантує академічну й організаційну автономію закладів освіти (стаття 23), зокрема, академічну свободу педагогічним працівникам (стаття 53). У закладах загальної середньої освіти (далі ЗЗСО) кожен учитель працює відповідно до своїх можливостей та уподобань. Зважаючи на фінансову спроможність ЗЗСО та середній вік педагогічних працівників, який складає 46 років (за даними дашборду МОНУ [1]), багатофункціональні системи дистанційного навчання використовуються досить рідко. Педагоги надають перевагу окремим інструментам.

Об’єкт дослідження: процес оцінювання навчальних досягнень учнів закладів загальної середньої освіти.

Предмет дослідження: технології для організації підсумкового оцінювання навчальних досягнень учнів закладів загальної середньої освіти.

Мета роботи: розробка вебзастосунку оцінювання навчальних досягнень учнів для удосконалення внутрішньої системи забезпечення якості освіти в закладах загальної середньої освіти.

Для досягнення зазначеної мети необхідно виконати наступні **завдання**:

- провести аналіз предметної області та систем із подібним функціоналом;
- змоделювати необхідні для системи діаграми;
- обрати технології реалізації програмного коду;
- розробити серверну частину проєкту;
- реалізувати користувацький інтерфейс;

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд існуючих аналогів

Для розробки специфікацій вимог до програмного забезпечення, що проєктується, важливо провести аналіз існуючих рішень у галузі створення та проходження тестів для оцінювання навчальних досягнень, виділивши їх переваги та недоліки. Серед обраних аналогів − вебзастосунки «На Урок» (розділ «Тести») [2], «Всеосвіта» (розділ «Конструктор тестів») [3] та Google Forms [4], які вирізняються високою популярністю та функціональністю.

«На Урок» (розділ «Тести»)

Сайт «На Урок» має багато різноманітного функціоналу не лише для здобувачів освіти, але й для викладачів. Через його широконаправленість в рамках теми розглядається розділ, пов’язаний зі створенням та проходженням тестових завдань «Тести». Вебзастосунок надає можливість переглядати, створювати та проходити тести та зберігати для подальшого створення особистого тесту на базі збереженого.

Таблиця 1.1 – Характеристики вебзастосунку «На Урок» (розділ «Тести»)

Назва	«На Урок» (розділ «Тести»)
Виробник	ТОВ «На Урок»
Архітектура	Web application
Мова реалізації	HTML, CSS, JavaScript
Функції	1) створення тестів 2) проходження тестів; 3) збереження тестів; 4) автоматичне оцінювання тесту; 5) збереження оцінок в особистому профілі; 6) збирання зворотного зв’язку.

Кінець таблиці 1.1

Переваги	1) безкоштовне використання; 2) активна підтримка розробником; 3) лаконічний дизайн; 4) проходження публічно доступних тестів.
Недоліки	1) два типи тестових питань; 2) довгі анімації перевірки відповіді; 3) некоректне збереження результатів тестувань в особистий профіль; 4) проходження тестів неавторизованим користувачем; 5) перегляд питань тесту до його проходження.
Джерело інформації	https://naurok.com.ua/test

«Всеосвіта» (розділ «Конструктор тестів»)

Як і попередньо проаналізований сайт, «Всеосвіта» має широкий функціонал, тому розглядається розділ схожий у функціоналі розділ «Конструктор тестів». Вебзастосунок надає інструменти для перегляду, створення та проходження тести та створення тесту на основі вже існуючого. Головною особливістю сайту є можливість монетизації тестових завдань.

Таблиця 1.2 – Характеристики вебзастосунку «Всеосвіта» (розділ «Конструктор тестів»)

Назва	«Всеосвіта» (розділ «Конструктор тестів»)
Виробник	ТОВ «Всеосвіта»
Архітектура	Web application
Мова реалізації	HTML, CSS, JavaScript

Кінець таблиці 1.2

Функції	1) створення тестів; 2) проходження тестів; 3) збереження тестів; 4) автоматичне оцінювання тесту; 5) задання часового обмеження тестування.
Переваги	1) безкоштовне використання; 2) підтримка розробником; 3) монетизація авторських тестів; 4) пошук власних тестів за фільтрами.
Недоліки	1) складність в орієнтації по сторінці; 2) проходження тестів неавторизованим користувачем; 3) перегляд питань тесту до його проходження; 4) перехід браузеру у повноекранний режим під час тестування.
Джерело інформації	https://vseosvita.ua/test

«Google Forms»

Google Forms це вебзастосунок, головною відмінністю якого від попередніх аналогів є його вузьконаправленість для роботи з тестами або опитуваннями. Інструмент від Google необхідний для створення та проходження тестів.

Таблиця 1.3 – Характеристики вебзастосунку «Google Forms»

Назва	Google Forms
Виробник	Google
Архітектура	Web application
Мова реалізації	HTML, CSS, JavaScript

Кінець таблиці 1.3

Функції	1) створення тестів; 2) проходження тестів; 3) автоматичне оцінювання тесту; 4) надсилання копії відповідей тестування на пошту; 5) створення діаграм відповідей.
Переваги	1) безкоштовне використання; 2) підтримка розробником; 3) підключення додаткових JS скриптів; 4) налаштування доступу до тесту авторизованим та неавторизованим користувачам.
Недоліки	1) зберігання тестів на власному Google Диску; 2) відсутність обмеження в часі проходження без додаткових скриптів; 3) відсутність профілів для збереження оцінок.
Джерело інформації	https://www.google.com/intl/uk/forms/about/

Кожен з розглянутих аналогів володіє перевагами, такими як розширений функціонал і зручність користування, що забезпечують високу користувацьку цінність. Проте, існують і недоліки, такі як перегляд питань тесту до його проходження та проблеми зі збереженням результатів, що вимагають уваги при розробці нових рішень. Ці висновки будуть враховані для уникнення подібних проблем у майбутньому проєкті та для покращення якості користувацького досвіду.

1.2 Аналіз системи, що розробляється

Вебзастосунок оцінювання навчальних досягнень учнів повинен облегшити проведення залікових та інших робіт, надаючи зручний та доступний інструмент як для здобувачів освіти, так і для викладачів.

Створення та проходження тестів має бути простим та швидким, а профілі користувачів мають надавати можливості призначення та оцінення тестів без додаткової комунікації іншими застосунками.

Таблиця 1.4 – Аналіз системи, що розробляється

Основні задачі	1) реєстрація користувачів; 2) авторизація існуючих користувачів; 3) додавання учнів одного класу до вчителя; 4) створення тестів; 5) редагування створених тестів; 6) призначення створених тестів; 7) проходження тестів; 8) перевірка відкритого питання в тесті; 9) зберігання результатів тестування в профілі учня; 10) сповіщення у профілі.
Користувачі системи	1) учень; 2) вчитель.
Сценарії роботи системи	1) вчитель створює тест; 2) вчитель призначає створений тест. Учням додається тест для проходження та надсилається повідомлення про необхідність проходження тесту в їх профілі; 3) вчитель перевіряє відкритті питання тесту. Після оцінення питання оцінка оновлюється в профілі учня; 4) учень проходить тест. Після проходження тесту на сторінці з оцінками з’являється новий запис з отриманою оцінкою. Якщо тест мав відкриті питання, на сторінці з відповідями на даний тест з’являється відповідь учня на питання для подальшого оцінення, а також повідомлення про необхідність оцінення відкритого питання в профілі.

Кінець таблиці 1.4

Засоби апаратної та програмної реалізації	1) back-end: C#, .NET, AspNetCore, EntityFrameworkCore; 2) front-end: JavaScript, React; 3) database: Microsoft SQL Server.

Майбутній вебзастосунок має значно покращити процес оцінювання навчальних досягнень, забезпечуючи здобувачів освіти та викладачів ефективними засобами для створення, проходження та оцінювання тестів.

1.3 Специфікації вимог до програмного забезпечення

ПРИЗНАЧЕННЯ ТА МЕЖІ ПРОЄКТУ

Призначення системи (застосунку), для якої розробляється програмне забезпечення

Призначенням застосунку є надання закладам загальної середні освіти вебзастосунок для оцінення навчальних досягнень учнів.

Погодження, що ухвалені в програмній документації

Було погоджено, що для створення загального ПЗ та його стабільної роботи будуть використовуватися фреймворки ASP.NET та EntityFrameworkCore.

Межі проєкту ПЗ

Крайня дата завершення роботи над ПЗ – 31.05.2024р.

ЗАГАЛЬНИЙ ОПИС

Сфера застосування

ПЗ призначене для оцінення навчальних досягнень учнів закладів загальної середньої освіти шляхом створення та проходження тестових завдань.

Характеристики користувачів

Основні характеристики користувачів: наявність ПК та підключення до мережі Інтернет.

Загальна структура і склад системи

Програмне забезпечення складається з клієнтського інтерфейсу на React, серверної частини на ASP.NET Core для обробки запитів і логіки, MS SQL Server для зберігання даних через Entity Framework Core, і REST API для зв'язку між клієнтською і серверною частинами.

ФУНКЦІЇ СИСТЕМИ ОЦІНЮВАННЯ НАВЧАЛЬНИХ ДОСЯГНЕНЬ УЧНІВ ЗАКЛАДІВ ЗАГАЛЬНОЇ СЕРЕДНЬОЇ ОСВІТИ

Функція створення тестів

Опис функції

Функція створення тестів дозволяє вчителям формувати індивідуальні тестові завдання для оцінювання навчальних досягнень учнів.

Вхідна і вихідна інформація

Вхідна інформація – назва та опис тесту, опціональне обмеження в часі, перелік питань, типи відповідей, бали за кожне питання.

Вихідна інформація – структурований тест готовий до розміщення на сайті.

Функціональні вимоги

Доступ до таблиць БД зі створеними тестами, доступ до функціоналу для їх редагування та доступ до мережі Інтернет.

Функція проходження тестів

Опис функції

Функція дозволяє учням виконувати тести в онлайн-режимі, відповідно до встановлених вимог викладача.

Вхідна і вихідна інформація

Вхідна інформація – доступ до обраного тесту, відповіді учня.

Вихідна інформація – результати та оцінка тесту.

Функціональні вимоги

Доступ до таблиць БД з записами інформації про тест, опціональні часові обмеження для виконання тесту та доступ до мережі Інтернет.

Функція зміни тестів

Опис функції

Функція зміни тестів дозволяє вчителям оновлювати або модифікувати існуючі тестові завдання, змінюючи назву та опис тесту, часове обмеження, питання, варіанти відповідей, а також налаштування оцінювання.

Вхідна і вихідна інформація

Вхідна інформація – ідентифікатор тесту, змінені питання та параметри.

Вихідна інформація – оновлений тест з новими даними.

Функціональні вимоги

Доступ до таблиць БД з інформацією про тест, доступ до БД для збереження змін та доступ до мережі Інтернет.

Функція оцінення відкритих питань

Опис функції

Ця функція надає вчителям можливість оцінювання відповідей на відкриті питання, які не можуть бути автоматично перевірені системою. Відповіді на відкриті питання зберігаються окремо і доступні для ручного перегляду та оцінювання вчителем.

Вхідна і вихідна інформація

Вхідна інформація – відповіді учнів на відкриті питання.

Вихідна інформація – оцінки за відкриті питання, внесені вчителем.

Функціональні вимоги

Доступ до таблиць БД з результатами тестування та доступ до мережі Інтернет.

ВИМОГИ ДО ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

Джерела і зміст вхідної інформації

Джерелами інформації виступають вчителі, учні, адміністрація школи. Змістом інформації є особисті дані користувачів, питання тестів, відповіді учнів, оцінки, адміністративні дані.

Нормативно-довідкова інформація

Довідниками є перелік шкільних предметів, критерії оцінювання та навчальні плани.

Вимоги до способів організації, збереження та ведення інформації

Обмін даними між клієнтською та серверними частинами відбувається за допомогою REST API. БД для збереження інформації було обрано MS SQL Server.

ВИМОГИ ДО ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

Жорстких вимог до технічного забезпечення немає. Користувач повинен мати комп’ютер чи ноутбук з будь якою встановлена ОС та підключення до мережі Інтернет.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Архітектура програмної системи

Архітектура складається з клієнтської частини, серверної частини та бази даних.

Системне програмне забезпечення

React обрано для розробки клієнтського застосунку. Серверна частина буде написана з використанням ASP.NET Core. MS SQL Server обрано для зберігання і обробки даних з використанням Entity Framework Core для ведення бази даних.

Мережне програмне забезпечення

Для створення ПЗ використовується ОС Windows, редактори коду Visual Studio та Visual Studio Code та Google Chrome у якості браузеру.

Програмне забезпечення ведення інформаційної бази

Всі взаємодії з базою даних відбуваються через Entity Framework Core для виконання CRUD-операцій.

Мова і технологія розробки ПЗ

Для розробки програмного забезпечення було обрано використання C# для серверної частини та JavaScript для клієнтської частини, а також фреймворки ASP.NET Core для бекенду та React для фронтенду.

ВИМОГИ ДО ЗОВНІШНІХ ІНТЕРФЕЙСІВ

Інтерфейс користувача

Інтерфейс має мати інтуїтивно зрозумілий та адаптивний дизайн для легкого користування застосунком.

Апаратний інтерфейс

Апаратним інтерфейсом є ПК або ноутбук, який буде використовуватися для взаємодії з вебзастосунком.

Програмний інтерфейс

ASP.NET Core – фреймворк для створення вебзастосунків і сервісів на C# з використанням .NET. React – JavaScript-фреймворк для розробки інтерфейсів користувача, які можуть оновлювати дані без перезавантаження сторінки.

Комунікаційний протокол

Вебзастосунок буде використовувати мережеві протоколи HTTP, HTTPS та протокол бездротової передачі даних TCP/IP.

ВЛАСТИВОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Доступність

Застосунок має бути доступним для всіх користувачів, що мають ПК або ноутбук та доступ до мережі Інтернет.

Супроводжуваність

Супроводження не передбачається.

Переносимість

Програмне забезпечення має працювати на будь-яких операційних системах, що підтримують сучасні браузери.

Продуктивність

Продуктивність роботи ПЗ залежить від швидкості мережевого з’єднання користувача та навантаження на БД. Час виконання запиту не має перевищувати 5 секунди.

Надійність

Програмне забезпечення має бути надійним і виключати можливості зловживанням рівнем доступу шляхом обмеження доступу до приватних даних користувачів, які не використовуються для здійснення оцінення та надання користувачам виключно їх особисті дані у їх профілях, які доступні лише після авторизації.

Безпека

Наявність процедури авторизації та автентифікації використовуючи зашифровані токени.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи бакалавра проведено аналіз існуючих вебзастосунків для оцінювання навчальних досягнень шляхом тестування, що дозволило виявити ключові переваги та недоліки кожного з них. Аналізовані застосунки включають «На Урок», «Всеосвіта» та Google Forms, кожен з яких має унікальні характеристики в контексті створення та проходження тестів.

В розділі міститься аналіз системи, що розробляється, яка має на меті поліпшення процесу оцінювання у закладах загальної середньої освіти. Визначено основні задачі системи, користувачів та сценарії її використання, що включають реєстрацію та авторизацію користувачів, створення та проходження тестів, а також оцінювання відповідей. Розроблено технічну специфікацію програмного забезпечення, що включає вимоги до клієнтського інтерфейсу, серверної частини, бази даних та зовнішніх інтерфейсів. Зазначено, що вебзастосунок розробляється на базі ASP.NET і React, що забезпечує надійність та ефективність системи.

2 МОДЕЛЮВАННЯ ВЕБЗАСТОСУНКУ

2.1 Створення use case

Розробка нового вебзастосунку вимагає як розуміння технічних аспектів, так і чіткого уявлення про поведінку користувачів та їхні взаємодії з системою. Для цього було проведено аналіз використання, що включає Use case діаграми та текстові сценарії. Ці інструменти дозволяють візуалізувати та структурувати вимоги до застосунку, що розробляється.

Use case діаграма, або діаграма використання – це графічне представлення взаємодії користувача з системою та опис специфікацій варіантів використання [5]. Діаграма відображає різні типи користувачів системи та різні способи їх взаємодії з системою. Це дозволяє виявити і оптимізувати всі потенційні шляхи взаємодії зі створеним продуктом, забезпечуючи зручність та інтуїтивність використання.

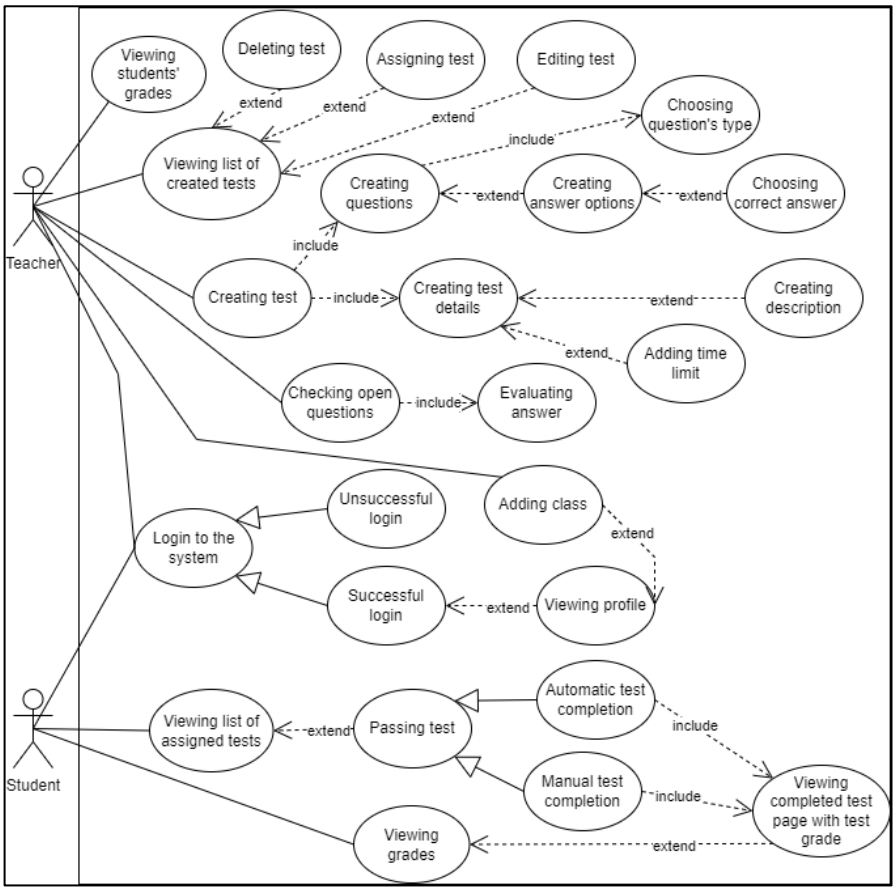

Рисунок 2.1 − Діаграма використання вебзастосунку оцінювання навчальних досягнень учнів

Текстові сценарії використання, що доповнюють діаграму, надають детальний опис кожної взаємодії, включаючи послідовність дій, альтернативні шляхи та виключення, що зустрічаються під час роботи з системою. Існує три форми написання сценаріїв використання:

1) коротка форма – найбільш стисла форма і, як правило, використовується на ранніх стадіях розробки для швидкого огляду потенційних функціональностей системи. Містить лише кілька речень або абзац, що описує основну мету сценарію та його основні дії без детального опису взаємодій або альтернативних шляхів;

2) поверхнева – містить більше деталей, ніж коротка форма, але є досить лаконічним варіантом. Включає кілька абзаців, що описують основні дії та основні альтернативні шляхи, які можуть виникнути під час використання;

3) повна − найбільш детальна форма, яка містить всі можливі аспекти сценарію використання, включаючи передумови, основні та альтернативні потоки подій, постумови, спеціальні вимоги, ідентифікацію учасників та їхні ролі.

Коротка форма use case: входження в систему

Користувач заходить на сайт. У поля для вводу вводить свої електронну пошту та пароль, після чого натискає на кнопку «Login». Система перевіряє правильність введених даних. Після успішної перевірки система відкриває сторінку з профілем користувача.

Поверхнева форма use case: перевірка відкритого питання в тесті

Головний сценарій (успішний):

Вчитель успішно заходить в систему. В хедері натискає на кнопку «Tests List» та переходить на сторінку зі створеними тестами. Обирає необхідний тест та натискає на його назву. На новій сторінці, що відкрилась, обирає учня, якому необхідно перевірити відкрите питання. Під іменем учня відображається його відповідь, текст та оцінка відкритого питання. Вчитель

виставляє оцінку за відповідь та натискає кнопку «Submit». Система додає до оцінки за автоматично-перевірені тести оцінку за відкрите питання та оновлює загальну оцінку за тест.

Альтернативні сценарії:

1) вчитель ввів не правильні дані для входу, система не може відкрити профіль користувача;

2) вчитель не надав в тесті питання з відкритою відповіддю, на сторінці тесту немає можливості перевірити відповідь учня за відсутності відкритого питання;

3) вчитель не ввів оцінку за відповідь на питання та натиснув кнопку «Submit», система виділяє червоним кольором поле для вводу оцінки та виводить повідомлення про помилку;

4) вчитель оновив сторінку, не натиснувши кнопку для збереження «Submit», система не зберігає оцінки, заповнені поля з оцінками очищаються.

Повна форма use case: проходження тесту

Таблиця 2.1 – Повний use case: проходження тесту

Use Case Name	Проходження тесту
Scope	Вебзастосунок оцінювання навчальних досягнень учнів ЗЗСО
Level	Успішно почати та завершити тест
Primary Actor	Учень
Stakeholders and interests	1) Учень: зацікавлений в проходженні тесту для отримання оцінки в поточному семестрі. 2) Вчитель: зацікавлений в швидкому оцінюванні учня за його навчальні досягнення без необхідності довгої перевірки. 3) Шкільна адміністрація: зацікавлена в оцінені учня у поточному семестрі за його навчальні досягнення. 4) Батьки учня: зацікавлені в інформації щодо знань учня методом надання їм його оцінки за навчальні досягнення.

Кінець таблиці 2.1

Preconditions	Учень має бути зареєстрованим та авторизованим в системі
Success guarantee	1) Учень використовує браузер, що підтримує HTML5 та JavaScript. 2) Учень має стабільне підключення до мережі Інтернет.
Main Success Scenario	1) Учень відкрив тест для проходження. 2) Учень відповів на всі питання. 3) Учень завершив тест натисканням на кнопку «Submit Test».
Extensions	1) Учень відповів не на всі питання: а) учень відкрив тест для проходження; б) учень відповів не на всі питання; в) учень натиснув кнопку «Submit Test»; г) питання, на які не було надано відповіді підсвічуються червоним кольором та виводиться повідомлення про помилку. 2) Учень не натиснув на кнопку «Submit Test», тест має часові обмеження: а) учень відкрив тест для проходження; б) учень відповів на всі питання; в) учень не натиснув кнопку «Завершити»; г) після закінчення таймеру, система самостійно завершує тест учня. 3) Учень не натиснув на кнопку «Submit Test», тест не має часових обмежень: а) учень відкрив тест для проходження; б) учень відповів на всі питання; в) учень не натиснув кнопку «Завершити»; г) система нічого не робить, очікуючи дій учня.
Special Requirements	Система має виконувати запити не довше 3 секунд
Technology and Data Variations List	Учень не має відволікатися на зовнішні фактори під час проходження тесту.
Frequency of Occurrence	≈ 75%

2.2 Побудова діаграм діяльності

Діаграма діяльності − це інструмент, який допомагає візуалізувати кроки та процеси всередині системи. Вона показує, як різні завдання взаємодіють між собою і які дії відбуваються для досягнення кінцевої мети. Діаграми діяльностей є важливим інструментом для візуалізації та аналізу бізнес-процесів вебзастосунку, що розробляється. Через детальне зображення кроків, умовних розгалуджень та паралельних процесів, діаграми дозволяють чітко розуміти потоки даних та управлінські взаємодії всередині системи.

Для вебзастосунку оцінювання навчальних досягнень учнів в ЗЗСО було створено три діаграми діяльності: авторизація зареєстрованого користувача (рис. 2.2), створення тесту (рис. 2.3) та призначення створеного тесту для учнів одного класу (рис. 2.4).

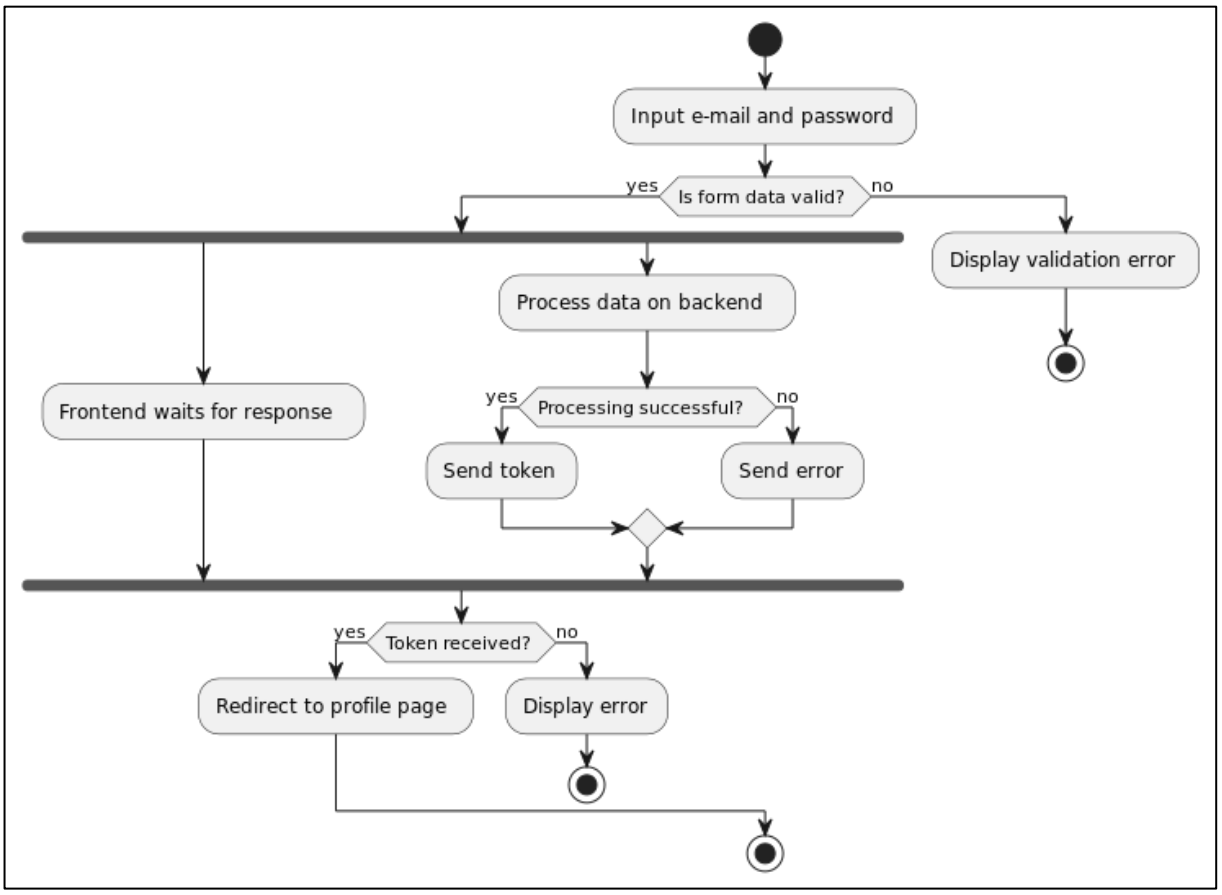

Рисунок 2.2 − Діаграма діяльності авторизації користувача

Ця діаграма діяльності ілюструє процес авторизації зареєстрованих користувачів системи. Незалежно від ролі користувача, процес

розпочинається з введення електронної пошти та пароля. Подальші кроки включають валідацію форми на предмет Перевірка наявності та формату введених даних. Якщо дані виявляються неповними або неправильно введеними, користувачу відображається повідомлення про помилку. У разі успішної валідації, система обробляє дані на сервері, шукаючи відповідності в базі даних. Якщо авторизація пройшла успішно, користувач отримує токен, що дозволяє перейти на сторінку свого профілю. У випадку помилки під час обробки, користувачу також відображається повідомлення про помилку.

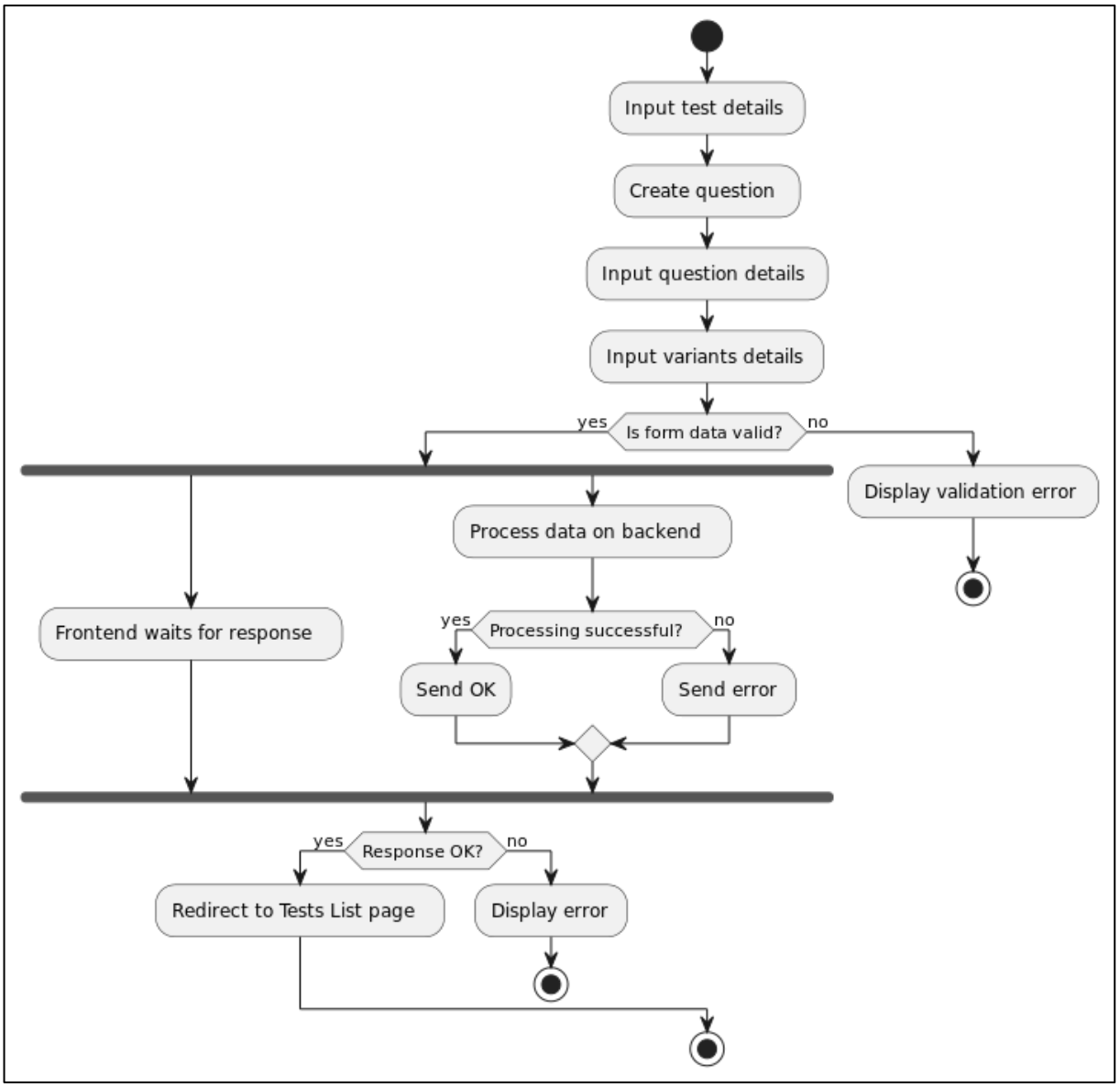

Рисунок 2.3 − Діаграма діяльності створення тесту

Діаграма діяльності демонструє процес створення тесту. Спершу користувач вводить всі необхідні деталі тесту, включаючи деталі питань та

варіанти відповідей. Після введення даних система виконує їх валідацію. У разі виявлення помилок, відображається відповідне повідомлення про помилку. Якщо ж дані валідні, вони обробляються на сервері, де відбувається їх збереження в базі даних. Після успішної обробки, з серверу до клієнта відправляється повідомлення «ОК», і система перенаправляє користувача на сторінку зі списком усіх створених тестів, де буде знаходитись новостворений тест разом з попередньо створеними.

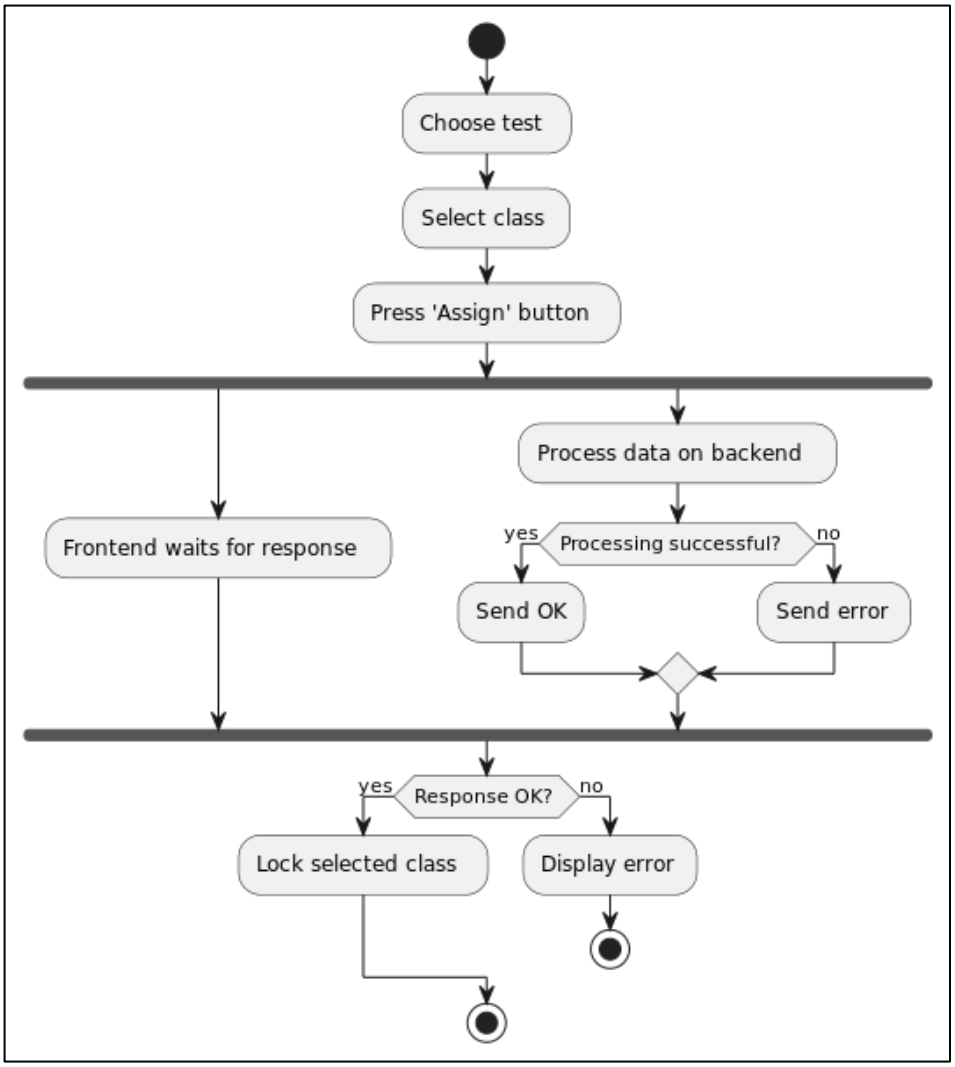

Рисунок 2.4 − Діаграма діяльності призначення тесту

На діаграмі діяльності призначення тесту зображено послідовність дій для надання учням доступу до створеного тесту для його подальшого проходження. Користувач обирає бажаний тест із доступного переліку, та обирає клас, якому буде призначено тест, з випадаючого списку класів. Цей

список генерується автоматично з даних, які надсилаються з бази даних, тому перевірка на існування класу в БД відсутня. Після вибору класу, користувач натискає кнопку «Assign», ініціюючи на сервері процес перевірки коректності даних та виявлення чи клас вже має цей тест в призначених. Якщо дані валідні та тест ще не призначений для класу, в клієнтському інтерфейсі клас блокується від подальшого призначення цього ж тесту. У випадку помилки користувачу відображається повідомлення про помилку.

2.3 Розробка діаграм взаємодії

Діаграми взаємодії використовується для візуалізації та документування взаємодій між об’єктами в системі. Вони дозволяють зрозуміти, як різні частини системи комунікують між собою під час виконання певних процесів або сценаріїв.

Діаграми послідовностей, які є одним з двох типів діаграм взаємодії, показують, як об’єкти взаємодіють згідно з часовою послідовністю. Кожен об’єкт представлений вертикальною лінією, а взаємодії між ними зображені горизонтальними стрілками, що вказують на послідовність повідомлень між ними.

Для забезпечення кращого розуміння того, як компоненти системи взаємодіють та ідентифікування потенційний проблем було створено діаграму взаємодії додавання нового класу вчителем до свого профілю (рис. 2.5) для подальшого призначення доданим класам тестів та перевірки відповіді на відкрите питання в тесті (рис. 2.6).

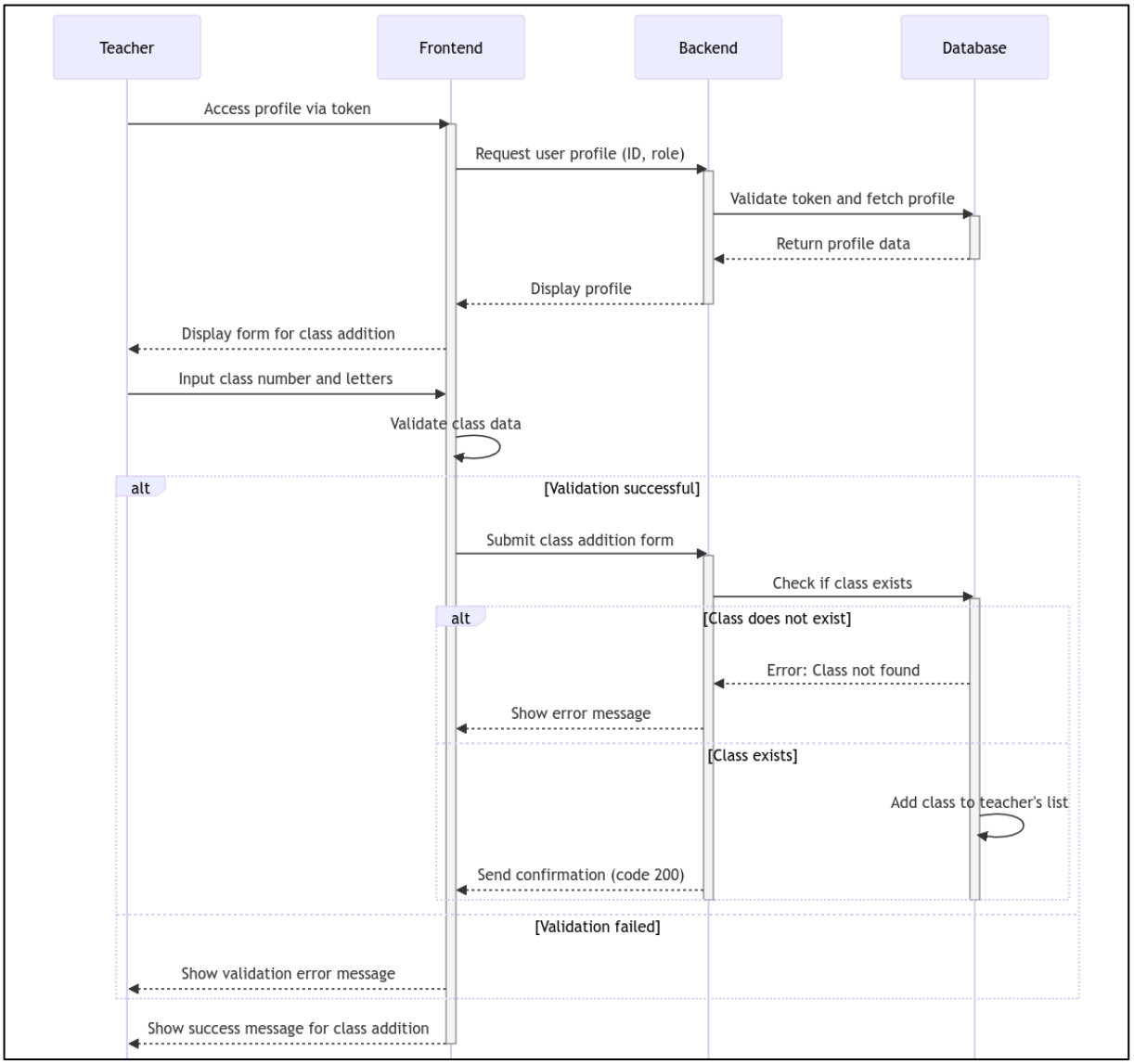

Рисунок 2.5 – Діаграма взаємодії додавання нового класу вчителем

На діаграмі зображено детальний процес додавання нового класу до профілю вчителя в системі оцінювання. Всі дії починаються з аутентифікації вчителя, де його ID та роль передаються через токен на бекенд для валідації та отримання деталей профілю. Після успішного доступу до профілю, відображається форма для додавання нового класу, де вчитель вводить дані про клас: цифру та букву. Ці дані проходять валідацію для перевірки їх коректності. У випадку, коли всі введені дані валідні, система перевіряє існування класу в базі даних. Якщо клас вже існує, у базі даних робиться запис про призначення цього класу вчителю, після чого відправляється підтвердження успішної операції з бекенду на фронтенд, і користувачу

відображається повідомлення про успіх. Якщо клас не знайдено або виникають інші помилки під час перевірки даних, вчителю відображається відповідне повідомлення з описом помилки.

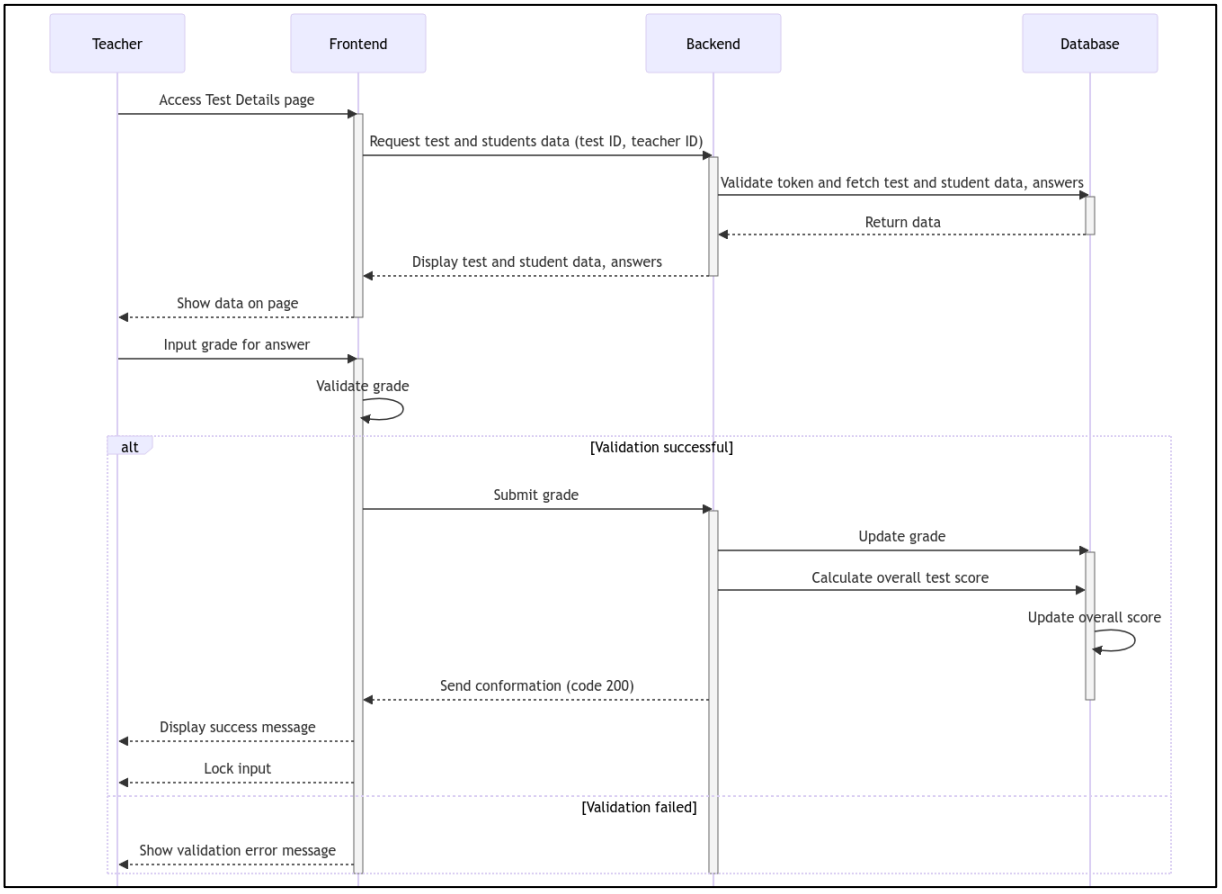

Рисунок 2.6 – Діаграма взаємодії оцінювання відповіді на відкрите питання

Ця діаграма взаємодії детально описує процес оцінювання відповідей учнів на відкриті питання тесту вчителем. Після отримання доступу до сторінки, використовуючи ідентифікатор вчителя з його токену та ідентифікатор тесту, відображені всі учні, які брали участь у тесті, разом з їх відповідями на відкриті питання. Вчитель вводить оцінки за кожну відповідь, які після валідації відправляються на сервер. На сервері дані перевіряються на коректність, після чого оновлюються у базі даних. Система також автоматично перераховує загальну оцінку за тест, оновлюючи її в базі. У випадку успішного оновлення оцінок, вчитель отримує підтвердження успішного виставлення оцінки, і поле для вводу оцінки для даної відповіді

блокується, що запобігає подальшій модифікації. Якщо в процесі виникають помилки, вчитель отримує відповідне повідомлення про помилку.

2.4 Діаграма розгортання

Діаграма розгортання (рис. 2.7) є важливим інструментом в архітектурі будь-якої інформаційної системи, яка візуалізує фізичне розміщення компонентів системи на апаратних ресурсах. Це допомагає забезпечити чітке розуміння того, як програмне забезпечення взаємодіє з обладнанням, на якому воно виконується, включаючи сервери, мережі та інші фізичні ресурси. Наявність цієї діаграми є необхідною для планування адекватної інфраструктури, ефективного розподілу ресурсів та забезпечення масштабованості та безпеки системи.

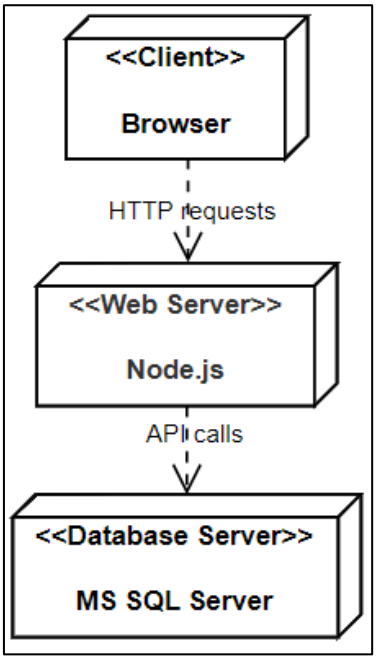

Рисунок 2.7 − Діаграма розгортання

Діаграма розгортання демонструє ключові компоненти системи оцінювання навчальних досягнень учнів та їх взаємодію. Клієнтська сторона представлена веб-браузером користувача, який взаємодіє з системою через HTTP запити. Веб-сервер, реалізований на платформі Node.js [6], обробляє ці запити, забезпечуючи динамічну генерацію контенту та логіку взаємодій користувача з інтерфейсом. API-виклики з веб-сервера забезпечують

комунікацію з сервером бази даних MS SQL Server [7], де зберігаються всі дані, необхідні для функціонування системи, включно з даними тестів, відповідями учнів, та результатами оцінювань.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи бакалавра детально розглянуто процес моделювання вебзастосунку для оцінювання навчальних досягнень учнів. Приділена увага використанню Use Case діаграм та сценаріїв використання, які покращують розуміння взаємодії користувачів з системою та оптимізують їх досвід. Продемонстровано застосування діаграм діяльності та взаємодії для наочного представлення робочих процесів та аналізу потенційних точок збою.

Важливою частиною розділу є діаграма розгортання, яка надає інформацію щодо розподілу компонентів системи на апаратних ресурсах, що допомагає плануванню інфраструктури та забезпеченню масштабованості та безпеки.

3 ПРОЄКТУВАННЯ ЗАСТОСУНКУ ТА ОГЛЯД СТЕКУ ТЕХНОЛОГІЙ

3.1 Розробка UML-діаграм

UML (Unified Modeling Language) є стандартною мовою моделювання, яка допомагає розробникам у візуалізації, проєктуванні, та документації архітектури програмних систем. Використання UML в архітектурі вебзастосунків дозволяє чітко представити структурні та поведінкові аспекти системи, спрощує передачу знань між учасниками проєкту та допомагає в ітеративному та інкрементальному розвитку застосунків [8]. Використання цих діаграм в процесі розробки дозволяє оптимізувати процес створення вебзастосунків, знижуючи ризики і забезпечуючи більш високу якість кінцевого продукту. В цьому розділі будуть розроблені такі UML-діаграми, як:

1) діаграма класів − описує структуру системи у вигляді класів і відносин між ними. Діаграма надзвичайно корисна для визначення структур даних, що є критично важливим у розробці вебзастосунків для забезпечення їх продуктивності та ефективності. Діаграма класів допомагає розробникам уникнути непорозумінь і помилок на етапі реалізації моделей даних;

2) діаграма компонентів − визначає взаємодію та залежності між різними компонентами програми, включаючи середовище виконання, бібліотеки та модулі. Діаграма компонентів є ключовою для проєктування модульної структури вебзастосунку, що дозволяє легше управляти змінами і масштабуванням системи. Це допомагає забезпечити, що архітектурні рішення узгоджені та всі компоненти інтегровані правильно;

3) діаграма пакетів − показує вищий рівень структурної організації системи, розділяючи компоненти на пакети, що дозволяє групувати пов’язані класи та компоненти в логічні модулі. Ця діаграма допомагає зрозуміти

загальну організацію системи і спростити навігацію по складних вебзастосунках.

3.1.1 Діаграма класів серверної частини застосунку

Діаграми класів серверної частини вебзастосунку є ключовим інструментом для проектування структури API застосунку, що надається фронтенду. Ці діаграми відіграють важливу роль у визначенні взаємодії між різними компонентами системи, такими як сервіси, контролери, моделі та контекст бази даних, що дозволяє ефективно керувати логікою та даними [8].

Логіку класів можна розділити на роботу з даними користувача та даними тесту. Так як моделі є сутностями, що пов’язані безпосередньо з таблицями бази даних, після проведення нормалізації останніх [9] кількість класів збільшилася, а отже через об’ємність кінцевої діаграми класів, її було розділено на дві за логікою роботи компонентів.

Діаграма класів (рис. 3.1) демонструє обробку даних користувачів у вебзастосунку. В центрі цієї архітектури знаходяться AppDbContext та контролери. AppDbContext використовується для інтеграції бізнес-логіки з базою даних через Entity Framework [10], що дозволяє ефективно управляти даними з високим рівнем абстракції та низькою залежністю від конкретної бази даних. У системі присутній інтерфейс IUserAuthService, що декларує ключові методи для аутентифікації та реєстрації користувачів. Його застосування дозволяє легко інтегрувати вебзастосунок, що розробляється, в різні системи, забезпечуючи адаптацію до різноманітних механізмів управління користувачами. Цей інтерфейс реалізує сервіс UserAuthService, що надає функціональність аутентифікації, реєстрації нових користувачів, перевірки існування користувача, та генерації токенів. EncryptionService використовується для шифрування паролів, забезпечуючи додатковий рівень захисту даних користувачів. Цей сервіс є відокремленим, що дозволяє його легко оновлювати або модифікувати в міру зміни вимог до безпеки.

Контролери, такі як UserAuthController, відіграють роль мосту між зовнішнім API і внутрішньою логікою застосунку, обробляючи запити від користувачів та викликаючи відповідні методи сервісів.

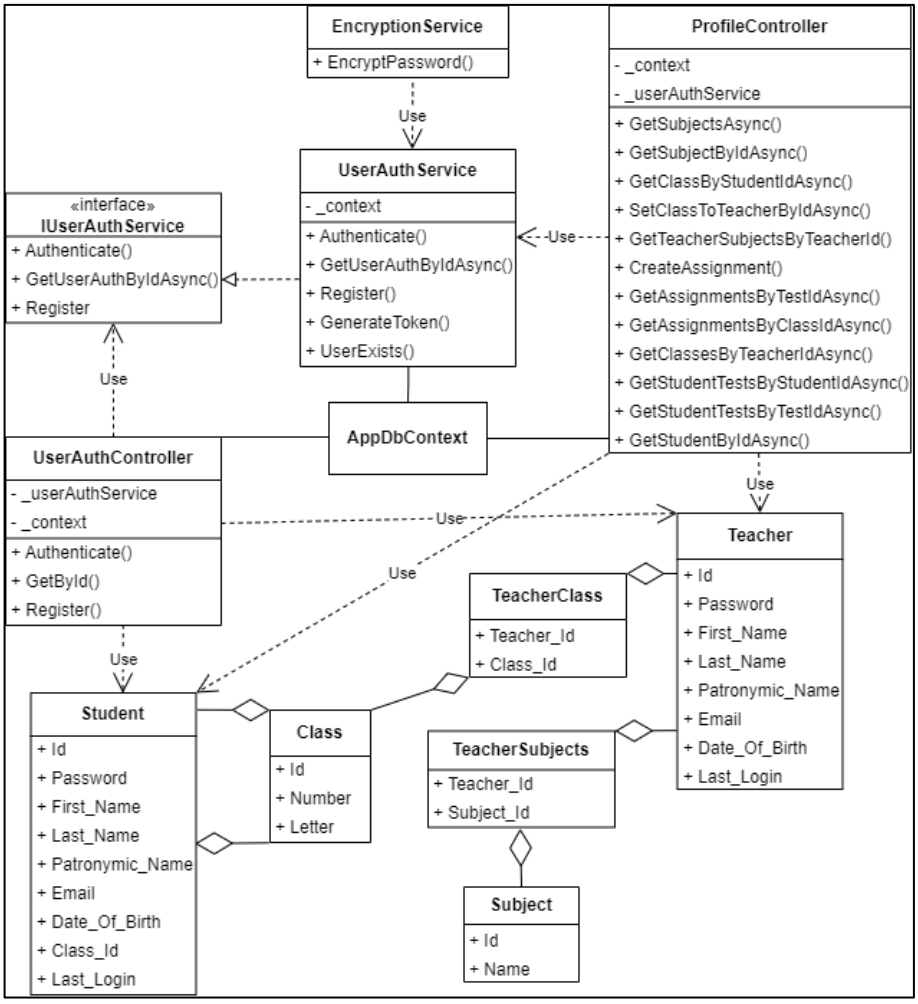

Рисунок 3.1 – Діаграма класів роботи з даними користувачів

Діаграма класів (рис. 3.2) представляє архітектуру серверної частини вебзастосунку, що керує даними тестів. Ключовий компонент цієї діаграми − TestsController, який відповідає за управління тестами через різноманітні методи, такі як створення, отримання, оновлення та видалення тестів. Цей контролер здійснює взаємодію з AppDbContext для роботи з базою даних через Entity Framework, забезпечуючи високий рівень абстракції у роботі з даними. Всі класи на діаграмі відображають таблиці бази даних і служать для зручності роботи з цими даними.

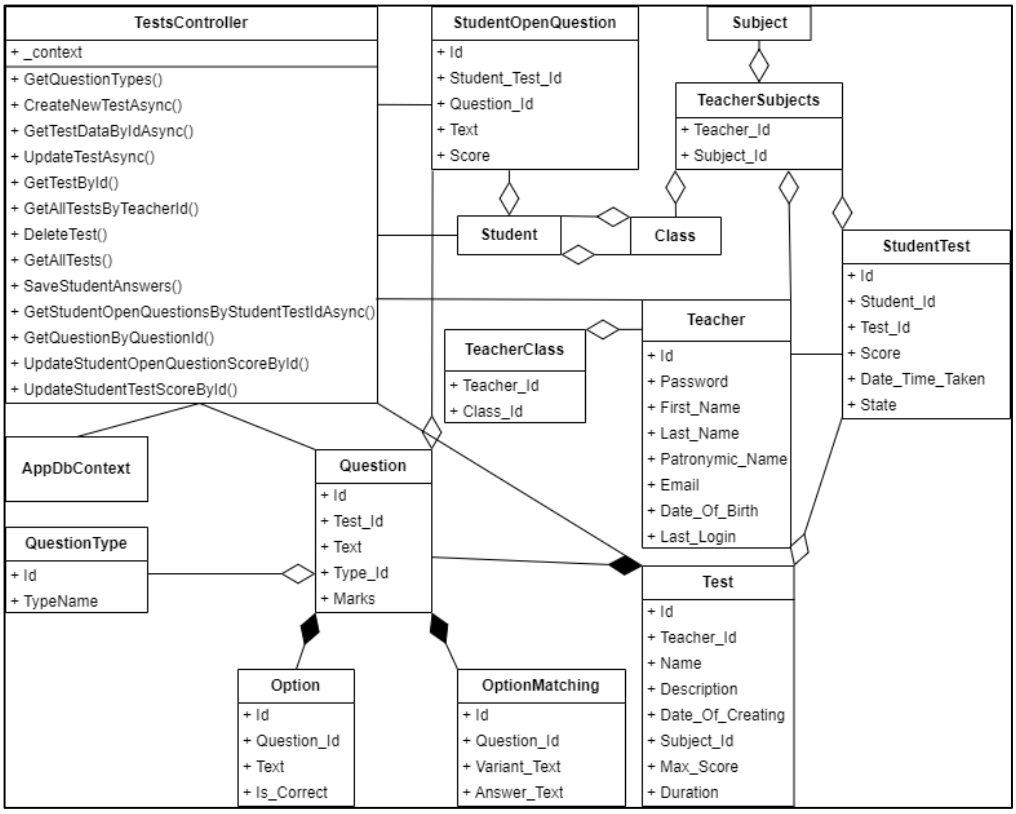

Рисунок 3.2 – Діаграма класів роботи з даними тестів

Використання діаграм класів ефективне для детального проектування архітектури серверної частини вебзастосунку. Завдяки чіткій структуризації та визначенню взаємодії між компонентами системи, забезпечено високу ефективність управління даними та логікою. Також, інтеграція сервісів для безпеки даних значно підвищила рівень захисту інформації у системі.

3.1.2 Діаграма компонентів

Діаграма компонентів відображає архітектуру системи, яка включає взаємозв’язки між основними частинами системи: презентаційним шаром, бізнес-логікою та базою даних. Це дозволяє ілюструвати, як взаємодіють різні ланки системи, підкреслюючи їх ролі та відповідальності в загальній структурі програми. Кожен компонент на діаграмі представляє фізичний модуль коду, що часто збігається з пакетом [8]. На діаграмі виділені основні блоки системи, їх відповідальності та способи взаємодії, що допомагає швидше зрозуміти основну архітектурну концепцію.

Діаграма компонентів (рис. 3.3) відображає триланкову архітектуру системи, що розробляється. MainComponent є центральним компонентом, який слугує точкою входу для користувацького інтерфейсу містить всю логіку презентаційної частини вебзастосунку, що розробляється. PresentationLayer складається з різних компонентів React, що відповідають за відображення користувацького інтерфейсу, маршрутизацію, управління станами, передачу даних через властивості і використання хуків для збереження функціональності між компонентами. BusinessLayer включає контролери, які управляють логікою обробки даних, сервіси для виконання бізнес-операцій, моделі, які представляють структуру даних, і контекст доступу до бази даних, що взаємодіє з Entity Framework для організації запитів до бази даних. DataLayer містить базу даних MS SQL Server, яка зберігає всі дані застосунку і є основним сховищем для збереження та відновлення інформації, яка обробляється вищезазначеними шарами.

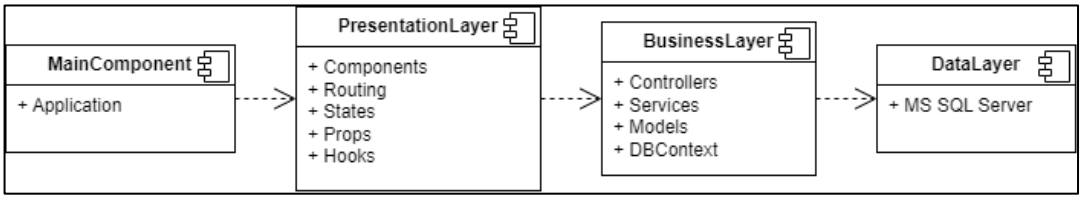

Рисунок 3.3 – Діаграма компонентів

Використання діаграми компонентів детально ілюструє архітектуру вебзастосунку. Діаграма наглядно відображає взаємозв’язки між презентаційним шаром, бізнес-логікою та базою даних, забезпечуючи зрозумілість структури та взаємодії компонентів системи. Розділення архітектури на логічні шари дозволяє ефективно управляти розробкою та забезпечувати високий рівень модульності та гнучкості у майбутньому.

3.1.2 Діаграма пакетів

Діаграми пакетів використовуються для візуалізації та структуризації компонентів системи. Така діаграма є одним із видів діаграм класів, яка акцентує увагу на пакетах та їхніх залежностях. Такі діаграми є корисними

для зображення модульності системи та визначення взаємозв’язків між різними модулями або компонентами. Визначення залежностей між пакетами допомагає розробникам розуміти, як зміни у одному сегменті програми можуть вплинути на інші. Це особливо важливо при плануванні та впровадженні модифікацій, оскільки дозволяє прогнозувати потенційні проблеми та мінімізувати ризики, пов’язані з взаємозалежністю компонентів [8].

Діаграма пакетів (рис. 3.4) ілюструє архітектуру системи, організовану у вигляді трьох рівнів, що відображають триланкову архітектуру. На рівні Presentation Layer розміщений основний компонент React Application, відповідальний за інтерфейс та взаємодію з користувачем. Business Layer включає контролери, що обробляють запити від користувачів, сервіси для авторизації та забезпечення, моделі, які представляють структуру даних в базі даних, та контекст, який забезпечує інтеграцію цих моделей з базою даних через Entity Framework. Data Layer, представлений MS SQL Server, зберігає всі дані системи. Компоненти кожного рівня взаємодіють, забезпечуючи чітке розділення відповідальностей, що спрощує подальше масштабування та технічне обслуговування системи.

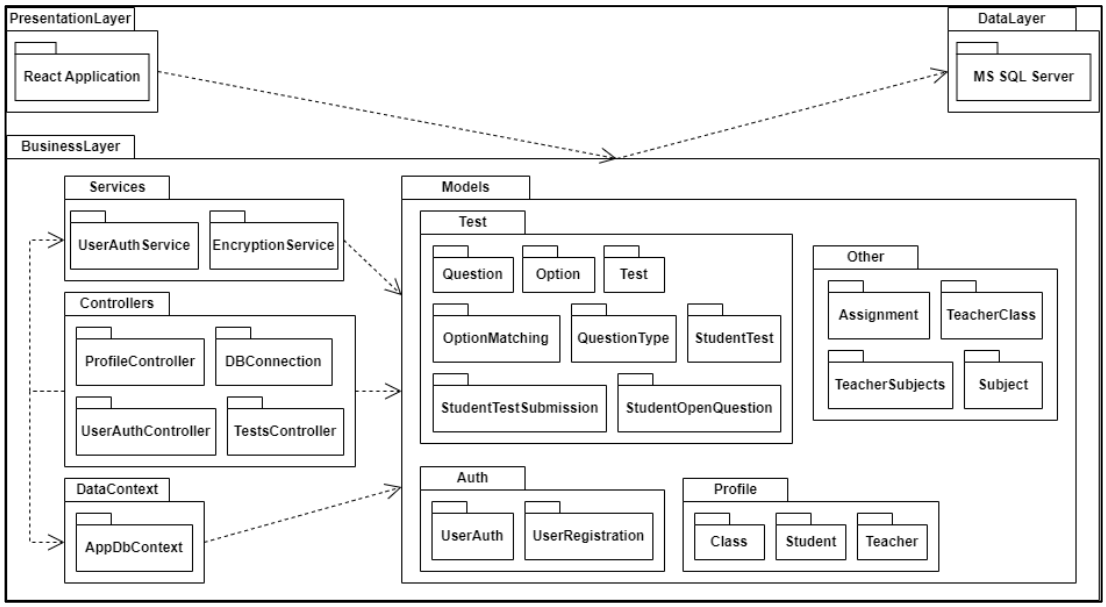

Рисунок 3.4 – Діаграма пакетів

Діаграми пакетів візуалізує архітектурну організацію та модульну структуру системи. Діаграма демонструє, як окремі компоненти взаємопов’язані та взаємозалежні, що критично для планування змін та оптимізації робочих процесів. Розділення системи на чітко визначені шари сприяє гнучкості управління проєктом та знижує ризики, пов'язані з масштабуванням та оновленням компонентів.

3.2 Огляд стеку технологій

Опис стеку технологій є ключовим для розуміння які технології застосовуються для досягнення бізнес-цілей проєкту, допомагає визначити оптимальні рішення для масштабування, підтримки та розширення вебзастосунку, враховуючи сучасні вимоги до продуктивності, безпеки та інтеграції. Основними технологіями, що були використані при проєктуванні системи є:

- HTML, JS, CSS;
- React;
- C#;
- ASP.NET Core;
- Entity Framework Core;
- MS SQL Server.

Детальний розбір технологій, а також не зазначених утиліт та бібліотек, надасть більш сформовану картину кінцевого продукту та його актуальність з технічної точки зору.

3.2.1 Мови програмування

Мова програмування C# була обрана для реалізації серверної частини вебзастосунку через її сучасність та широку підтримку в галузі. C# є найпопулярнішою мовою на платформі .NET, яка є безкоштовною, крос-платформною і має відкритий код. C# підтримує багатофункціональне

програмування та забезпечує високу продуктивність, маючи мільйони розробників по всьому світу [11].

C# є мовою з сильною типізацією, що сприяє зменшенню помилок пов’язаних із неправильним використанням типів даних. Вона включає автоматичне управління пам’яттю через збирач сміття, що забезпечує автоматичне звільнення пам’яті, використаної програмою. Також, C# підтримує обробку винятків, що дозволяє ефективно управляти помилками у програмі. Як повністю об’єктно-орієнтована мова, C# дозволяє використовувати класи, спадковість, поліморфізм і інкапсуляцію, що робить її ідеальною для створення масштабованих та легко підтримуваних застосунків [12].

Мова C++ також широко використовується для розробки бекенду завдяки своїй ефективності та низькому рівню абстракції, який надає тісний контроль над апаратними ресурсами. Однак, для реалізації серверної частини цього вебзастосунку було обрано C# через ряд переваг, які відповідають сучасним вимогам програмної інженерії. По-перше, сильна типізація і управління пам’яттю збирачем сміття знижують ризик помилок і витоків пам’яті, зустрічаних у мові C++. По-друге, об’єктно-орієнтовані можливості C# сприяють кращій організації коду і полегшують його підтримку та масштабування [13].

Ці особливості роблять C# вибором, який сприяє розробці продуктивного, безпечного та легко підтримуваного програмного забезпечення для сучасних вебзастосунків.

JavaScript − це високорівнева, динамічна та інтерпретована мова програмування, що добре підходить для веброзробки. Вона необхідна для додавання інтерактивної поведінки до вебсторінок, що робить її поширеною на вебсайтах по всьому світу. JavaScript дозволяє розробникам маніпулювати вмістом сторінок і реагувати на події користувача, забезпечуючи багатий користувацький досвід. На відміну від мов, які потребують компіляції, код

JavaScript можна писати в текстовому редакторі і запускати безпосередньо у браузері, що полегшує швидку розробку і тестування. Його неблокована, керована подіями природа робить його особливо ефективним для завдань, які не потребують складної обробки, а здатність легко взаємодіяти з HTML і CSS підвищує його корисність у розробці інтерфейсів [14].

Вибір JavaScript замість PHP, незважаючи на популярність останнього, є вигідним для для завдань на стороні сервера. Його здатність працювати як на стороні клієнта, так і на сервері з такими технологіями, як Node.js, що використовується у проєкті, забезпечує більш уніфікований досвід розробки. Це робить JavaScript особливо придатним для застосунків, які отримують переваги від функцій у реальному часі та безперебійної взаємодії з користувачем без перезавантаження сторінок, використовуючи його розгалужену екосистему та сучасні фреймворки, такі як React, для надійних, масштабованих програм [15].

3.2.2 Технології розробки front-end

Ні одна вебсторінка не може бути написаною без використання HTML. Щодо додаткових технологій, основною є бібліотека React для побудови користувацьких інтерфейсів [16].

React є популярною бібліотекою JavaScript, розробленою Facebook для створення користувальницьких інтерфейсів, зокрема для односторінкових програм, де потрібна швидка взаємодія з користувачем. Це дозволяє розробникам створювати великі веб-додатки, які можуть динамічно оновлювати дані без перезавантаження сторінки. React відомий своєю ефективністю та гнучкістю, використовуючи багаторазові компоненти, які керують власним станом і ефективно оновлюють і відтворюють потрібні компоненти, коли змінюються дані. Цей модульний підхід робить React високомасштабованим і зручним для обслуговування, ідеальним для складних програм [17].

React надає багато елементів для гнучкого програмування, які використовувались в проєкті, такі як компоненти, маршрутизація, стани, властивості та хуки. Компоненти − це будівельні блоки React-застосунку, невеликі багаторазові фрагменти коду, які повертають елемент React для відтворення в DOM (структуроване представлення елементів HTML, які присутні на вебсторінці). Маршрутизація керується такими бібліотеками, як React Router, і керує навігацією між різними частинами вебзастосунку, змінюючи URL-адресу браузера без перезавантаження сторінки. Стан є внутрішнім сховищем даних компонента, яке визначає його поведінку та спосіб відтворення. Властивості − це параметри, які передаються в компоненти для їх конфігурації та керування їх поведінкою, наприклад, айді вчителя чи тесту. Хуки є досить новим елементом, введеним в React версії 16.8 в 2019 році. Вони дозволяють використовувати стан та інші функції React у функціональних компонентах, забезпечуючи такі можливості, як керування станом, ефектами тощо, без необхідності писати компоненти на основі класів [17].

Як було зазначено вище, для реалізації маршрутизації використовується бібліотека React Router [18]. Маршрутизація − це процес визначення кінцевих точок для запитів вашого клієнта. Ці кінцеві точки працюють у поєднанні з об’єктами розташування та історії браузера. Вони використовуються для ідентифікації запитуваного вмісту, щоб JavaScript міг завантажити та відобразити відповідний інтерфейс користувача [17]. На відміну від Angular, Ember або Backbone, React не постачається зі стандартним маршрутизатором, тому в проєкті використовується бібліотека React Router для обробки маршрутизації в програмах React. Це дозволяє реалізувати динамічну маршрутизацію, процес, у якому навігація між різними частинами програми обробляється без необхідності перезавантажувати всю сторінку.

Axios − це популярна бібліотека JavaScript, яка використовується для створення HTTP-запитів, ідеально підходить для спілкування з API, розробленими на сервері. Він спрощує взаємодію з даними сервера, автоматично обробляючи дані JSON і забезпечуючи надійну підтримку асинхронних операцій через Promises [19]. Axios особливо корисний для вебзастосунків, які вимагають частої взаємодії з сервером, оскільки він ефективно керує отриманням даних і підтримує широкий спектр операцій запитів HTTP. Це робить його практичним вибором для проєкту, що розробляється, який передбачають інтенсивний обмін даними, використовуючи API.

React Query − це потужна бібліотека для керування станом сервера в React-застосунках. Вона покращує вибірку, кешування та синхронізацію даних, дозволяючи ефективно та швидко взаємодіяти з бекендом [20]. React Query автоматично оновлює інтерфейс користувача останнім станом сервера, що значно прискорює завантаження сторінки, зменшуючи потребу в повторній вибірці даних. Ця бібліотека оптимізує продуктивність рендерингу та управління даними.

Moment.js − це бібліотека, яка використовується у проєкті для форматування дат і часу, отриманих від API [21]. Це спрощує процес відображення цих значень у зручному для користувача форматі на інтерфейсі, забезпечуючи чітке й узгоджене представлення даних про дату й час.

jwtDecode − це утиліта, яка декодує вебтокени JSON (JWT) у програмах JavaScript [22]. Вона використовується для отримання інформації про користувача, вбудованої в токен, такої як ідентифікатори користувачів та роль, без запиту на сервер. Це робить утиліту ефективним інструментом для керування станами автентифікації та дозволами у вебзастосунку, допомагаючи гарантувати, що елементи інтерфейсу користувача та маршрути доступні відповідно до облікових даних користувача.

Стилізація вебзостунку була виконана фреймворком Bootstrap та бібліотекою Ant Design.

Bootstrap − це широко використовуваний фреймворк, який спрощує розробку адаптивних вебсторінок. Він надає повний набір інструментів CSS і JavaScript для створення компонентів інтерфейсу користувача, таких як кнопки, моди та панелі навігації. Система сітки Bootstrap і попередньо розроблені компоненти дозволяють швидко розробляти естетично привабливі та узгоджені інтерфейси для різних пристроїв і браузерів, покращуючи взаємодію з користувачем, не вимагаючи значного кодування CSS з нуля [23].

Ant Design − це бібліотека React UI, яка надає низку високоякісних компонентів та інструментів для покращення розробки масштабний застосунків. Бібліотека містить набір із понад 50 багаторазових компонентів, які задовольняють потреби більшості додатків, від простих кнопок і вводів до складних систем навігації. Ant Design також містить набір принципів дизайну та найкращих практик, які забезпечують узгодженість і зручність використання продуктів, що робить його популярним вибором для розробників [24].

Node.js відіграє важливу роль у проєкті, забезпечуючи можливість розробки серверної частини за допомогою JavaScript. Це кросплатформне середовище виконання, яке дозволяє JavaScript працювати поза браузером, що відкриває можливості для створення швидких і масштабованих мережевих застосунків. Node.js використовується для розробки бекенду вебзастосунків і API, а також для взаємодії з базами даних, обробки запитів та управління серверними процесами, роблячи його незамінним інструментом для сучасних вебзастосунків. Його неблокуюча модель вводу/виводу робить його особливо ефективним для операцій, що вимагають високої продуктивності [6].

3.2.2 Технології розробки back-end

Розробка бекенд-частини відбувалася мовою C# з використанням двох фреймворків, які відіграли важливу роль в розробці: ASP.NET Core та Entity Framework Core.

ASP.NET Core − це кросплатформний, високопродуктивний фреймворк з відкритим вихідним кодом для створення сучасних хмарних вебзастосунків, підключених до Інтернету. Він підтримує асинхронну обробку запитів, що підвищує продуктивність застосунків. Фреймворк використовує модульне та розширюване середовище, що дозволяє розробникам вибрати потрібні бібліотеки без залучення зайвого функціоналу. Також він інтегрований з Kestrel, високопродуктивним вебсервером, і вбудованою підтримкою контейнеризації, що сприяє легкості розгортання та масштабування застосунків [25].

ASP.NET Core дуже ефективний для побудови API, оскільки він розроблений для обробки HTTP-запитів, зіставляючи їх безпосередньо з методами, які повертають дані JSON або XML. Цей дизайн спрощує створення інтерфейсів RESTful, де використовуються різні методи HTTP, такі як GET, POST, PUT і DELETE, які ефективно взаємодіють із ресурсами даних. ASP.NET Core гарантує, що ці взаємодії не мають стану, узгоджені з архітектурним стилем REST, який є оптимальним для продуктивності вебслужб [26]. У проєкті, що розробляється, ASP.NET Core служить надійною платформою для розробки та реалізації логіки на стороні сервера та ефективної обробки запитів API. Він підтримує асинхронне програмування з async/await, підвищуючи продуктивність, дозволяючи вашим веб-службам обробляти кілька запитів одночасно, не чекаючи, поки кожне завдання завершиться синхронно.

Крім того, було використано спеціальні пакети з ASP.NET Core, такі як: Microsoft.AspNetCore.Authentication.JwtBearer [27] для обробки автентифікації JWT, Microsoft.AspNetCore.Mvc.NewtonsoftJson [28] для

операцій JSON і Swashbuckle.AspNetCore [29] для інтеграції інтерфейс Swagger [30], який допомагає документувати кінцеві точки API. Ці компоненти були інтегровані за допомогою диспетчера пакетів NuGet [31].

Entity Framework Core − це сучасна, полегшена та розширювана версія Entity Framework, яка є інфраструктурою об’єктно-реляційного відображення Microsoft для .NET [10]. Фреймворк діє як посередник між моделями даних програми та базою даних, дозволяючи маніпулювати базою даних за допомогою об’єктів .NET замість написання великого коду SQL. Це спрощує рівень доступу до даних шляхом автоматизації операцій бази даних, таких як дії CRUD, що дозволяє більше зосередитися на бізнес-логіці.

Entity Framework Core має велике значення для ефективного керування взаємодією між програмою та базою даних. Він спрощує операції, дозволяючи використовувати високорівневі конструкції програмування для обробки взаємодії з базою даних, що підвищує продуктивність і зменшує ризик помилок. Крім того, він підтримує запити LINQ, що дозволяє писати типобезпечні запити на C#, підвищуючи зручність обслуговування та безпеку [32]. В проєкті фреймворк автоматично обробляє складні запити SQL і оновлює схему бази даних, спрощуючи завдання керування даними.

Також у проєкті використано деякі компоненти Entity Framework Core, які були інтегровані за допомогою диспетчера пакетів NuGet: Microsoft.EntityFrameworkCore.SqlServer [33], який з’єднує феймворк з Microsoft SQL Server, надаючи необхідні інструменти для перетворення операцій .NET у запити SQL, які може виконувати сервер, та Microsoft.EntityFrameworkCore.Tools [34], що надає низку команд, які допомагають виконувати такі завдання, як створення коду для моделей бази даних, керування міграціями бази даних тощо, безпосередньо з середовища розробки.

MS SQL Server − це реляційна система управління базами даних, розроблена корпорацією Microsoft [35]. Це комплексне рішення для

зберігання, управління та аналізу даних, що підтримує широкий спектр транзакційних та аналітичних операцій у бізнес-застосунках. У проєкті, що розроблюється, MS SQL Server використовується як основна база даних для зберігання та управління всіма даними вебзастосунку. Завдяки його високій продуктивності, надійності та безпеці, MS SQL Server ідеально підходить для роботи з об’ємними та критично важливими даними [36]. Ця БД забезпечує зберігання інформації про користувачів та тести, що вимагає швидкого доступу та високої готовності.

Важливо, що використання MS SQL Server дозволяє використовувати складні запити SQL для CRUD операцій, що значно спрощує інтеграцію з back-end-частиною застосунку, розробленою на ASP.NET Core, де Entity Framework Core слугує як ORM-інструмент, що спрощує роботу з даними.

Висновки до розділу 3

В третьому розділі кваліфікаційної роботи бакалавра було описано кроки у процесі проєктування структури вебзастосунку для оцінювання навчальних досягнень учнів в ЗЗСО. Розглянуто підхід до моделювання архітектури застосунку використовуючи UML-діаграми, які включають діаграми класів, компонентів та пакетів, що дозволяють глибше зрозуміти взаємозв’язки та структуру системи. Було наведено необхідність використання діаграм та описано створені діаграми та їх компоненти.

Детальний огляд стеку технологій висвітлює використання мов програмування та фреймворків, зокрема C#, JavaScript, ASP.NET Core, та Entity Framework Core, які лежать в основі реалізації серверної та клієнтської частин вебзастосунку. Значна увага приділяється інтеграції різних технологій для забезпечення ефективної взаємодії між бекендом та фронтендом. Крім того, розділ включає аналіз використання бази даних MS SQL Server та ролі Node.js в розробці, підкреслюючи їх важливість для забезпечення стабільної та ефективної роботи вебзастосунку.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

4.1 Огляд дизайну вебзастосунку

При вході на сайт користувача вітає сторінка авторизації (рис. 4.1). У верхній лівій частині розміщено назву системи оцінювання, EduTest, та справа розташовані кнопки для входу та реєстрації. Ці кнопки змінюють свій зовнішній вигляд після того, як користувач автентифікується в системі.

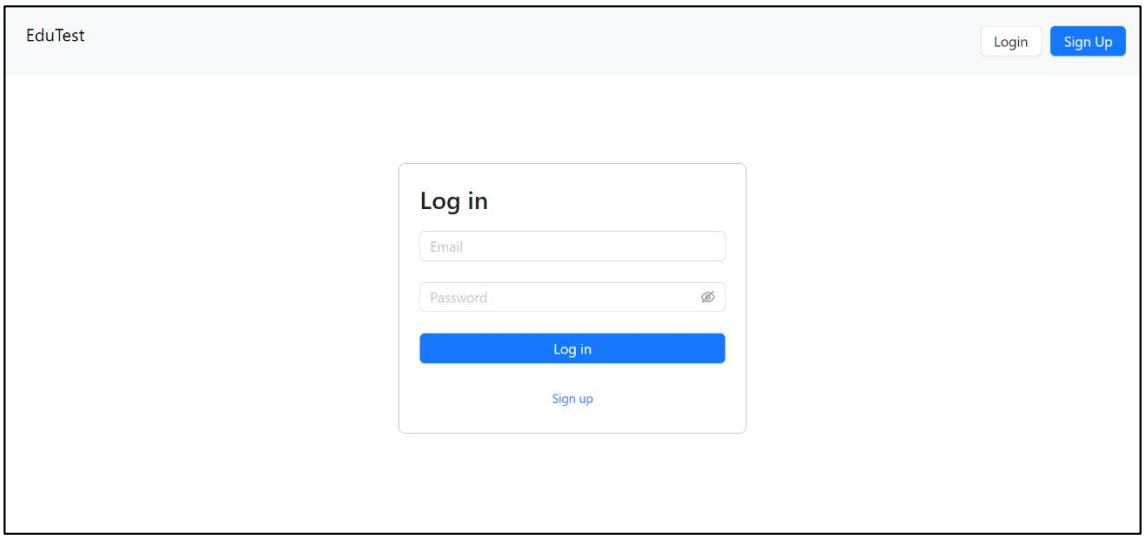

Рисунок 4.1 – Вигляд сторінки логіну

Центральний елемент сторінки − форма для введення електронної пошти та паролю. Для зручності користувачів, пароль може бути відображений у вигляді точок для збереження конфіденційності. Нижче поля для введення паролю розміщена велика синя кнопка для входу, яка яскраво виділяється та привертає увагу користувача. Під кнопкою входу знаходиться гіперпосилання для реєстрації нового акаунту, яке надає можливість користувачам швидко перейти до створення нового облікового запису.

Після переходу на сторінку реєстрації (рис. 4.2), користувачі зустрічаються з формою, яка включає поля для вводу імені, прізвища, необов’язкового по батькові, дати народження та електронної пошти. Форма також містить два поля для пароля − для введення та підтвердження, обидва з опцією приховування символів.

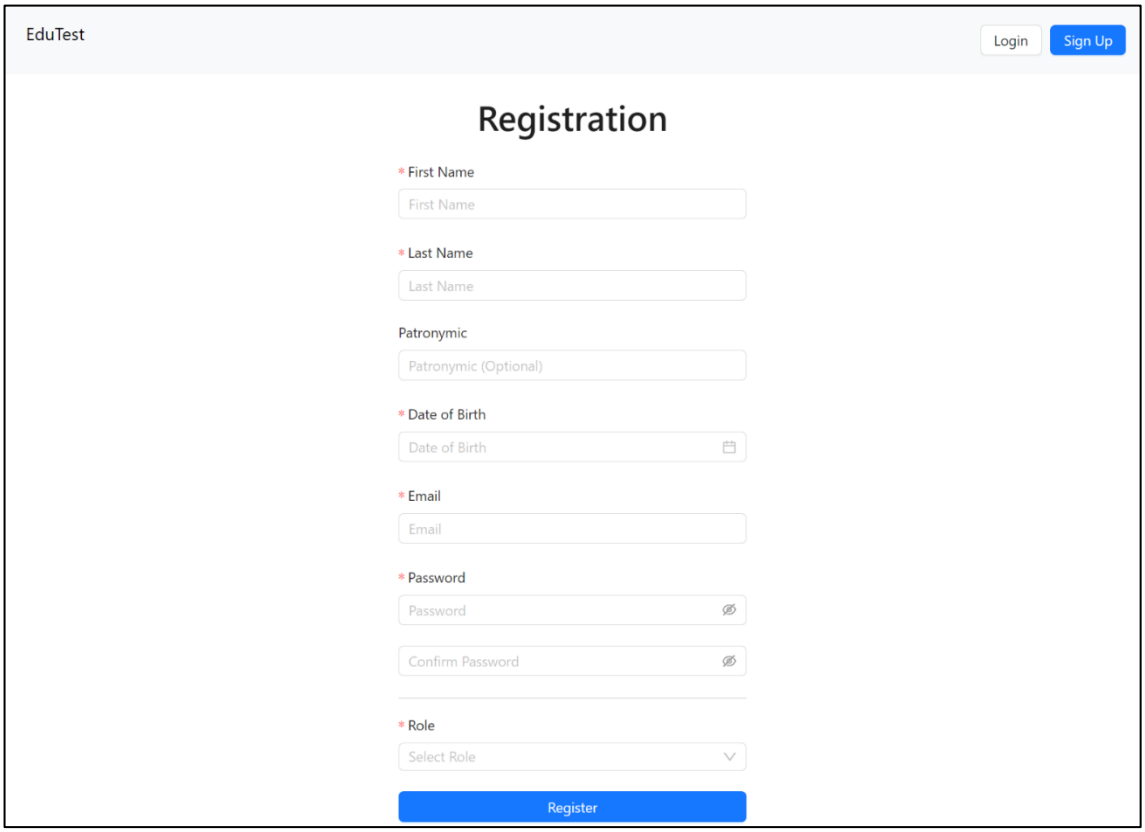

Рисунок 4.2 – Вигляд сторінки реєстрації

Додатково, форма дозволяє обрати роль користувача в системі (учень або вчитель), що впливає на подальший досвід користування сайтом.

Після обрання ролі, вчитель або учень, з'являються додаткові поля для реєстрації (рис. 4.3). Учень у нових полях має ввести свій клас, в якому він навчається, а вчитель має обрати предмети, які він викладає, з випадаючого списку. Під формою розташована кнопка «Реєстрація», що завершує процес реєстрації.

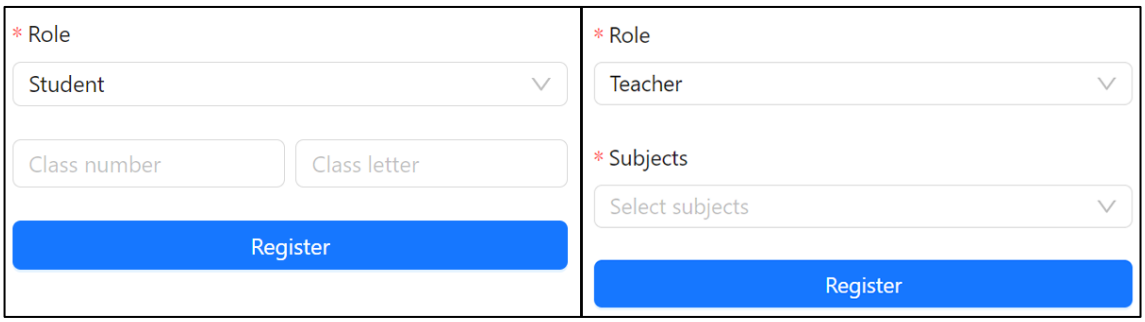

Рисунок 4.3 – Різниця додаткових полів форми реєстрації в залежності від ролі

Після авторизації, користувач-учень переходить на сторінку свого профілю (рис. 4.4), де відображаються його повне ім’я, роль у системі, та основна інформація, вказана при реєстрації: клас, дата народження, та електронна адреса. У верхній частині сторінки розміщені посилання на інші важливі розділи: оцінки, тести та особистий профіль. В правій частині хедера тепер знаходиться кнопка виходу з системи. Нижче, у правому куті екрану, з’являється спливаюче повідомлення про необхідність пройти призначені тести, якщо такі є.

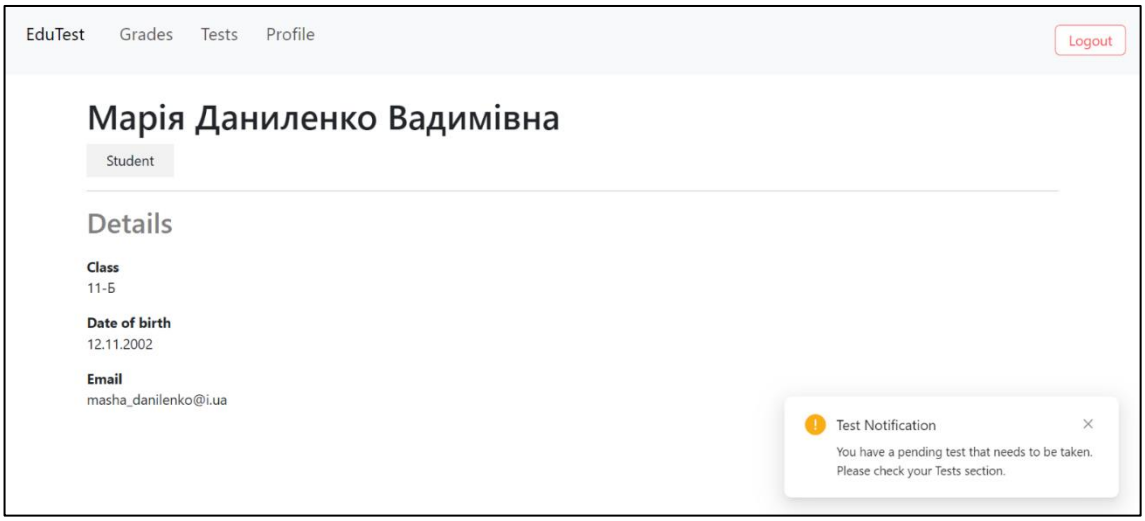

Рисунок 4.4 – Вигляд профілю учня з повідомленням про наявність тестів для проходження

На сторінці профілю вчителя (рис. 4.5) відображається інформація, внесена при реєстрації, включно з предметами, які він викладає, датою народження та електронною адресою. Логін та реєстраційні кнопки замінені на кнопку для виходу з системи. Навігаційний рядок містить посилання на сторінки оцінок учнів, список створених тестів, сторінку для створення нових тестів та профіль. У правому нижньому куті є сповіщення для вчителя про необхідність перевірки відповідей на відкриті питання у тестах, якщо вчитель попередньо вже створював та призначав тести з відкритими відповідями, які учні пройшли. Справа від імені вчителя є опція для додавання класів, що дозволяє призначати тести цілому класу, замість окремих учнів.

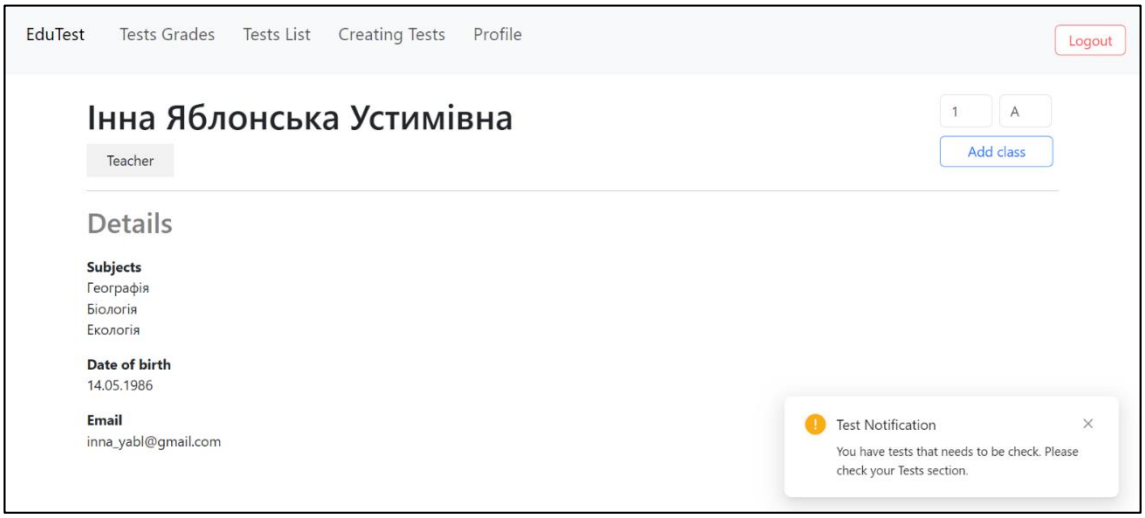

Рисунок 4.5 – Вигляд профілю вчителя з повідомленням про наявність тестів для перевірки

Після введення цифри та букви класу, в якому вчитель викладає, якщо всі дані введено правильно, у нижньому правому кутку з'являється повідомлення про успішне додавання класу до вчителя (рис. 4.6).

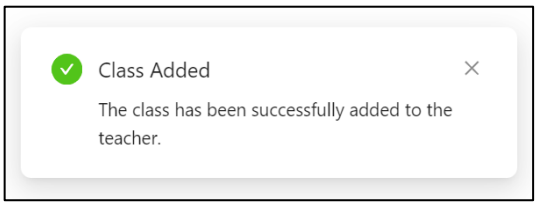

Рисунок 4.6 – Вигляд повідомлення про успішне додавання класу до вчителя

На сторінці «Creating Tests» (рис. 4.7) вказується назва тесту та, за бажанням, його опис. Використання таймера дозволяє встановити часове обмеження для тесту. У розділі вибору предмета відбувається вибір категорії тесту. Для додавання питань натискається на кнопку «Add Question», де можна вводити текст питання та обрати тип питання: одна правильна відповідь, декілька правильних, відкрите питання, або питання на відповідність. Додавання варіантів відповідей вимагає натискання на «Add Variant», де вказуються текст варіанту та його коректність. Варіанти на відповідність вимагають введення тексту варіанту і правильної відповіді. Кнопки «Delete» дозволяють видаляти питання чи варіанти. Завершення процесу створення тесту відбувається через кнопку «Save Test», після чого,

якщо дані правильно валідовано, користувача перенаправляють до списку створених тестів (рис. 4.8).

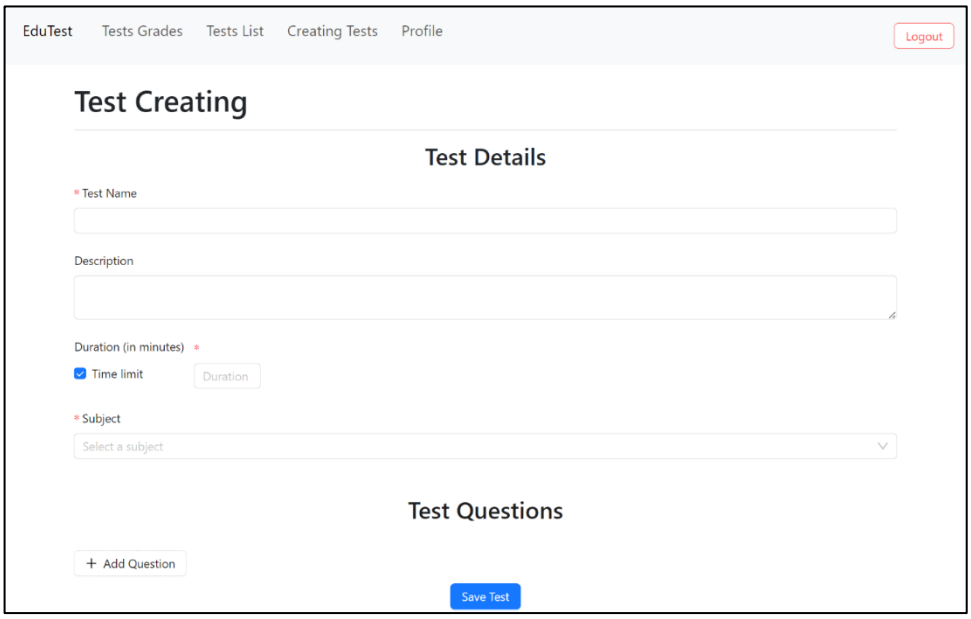

Рисунок 4.7 – Вигляд сторінки створення тесту без питань

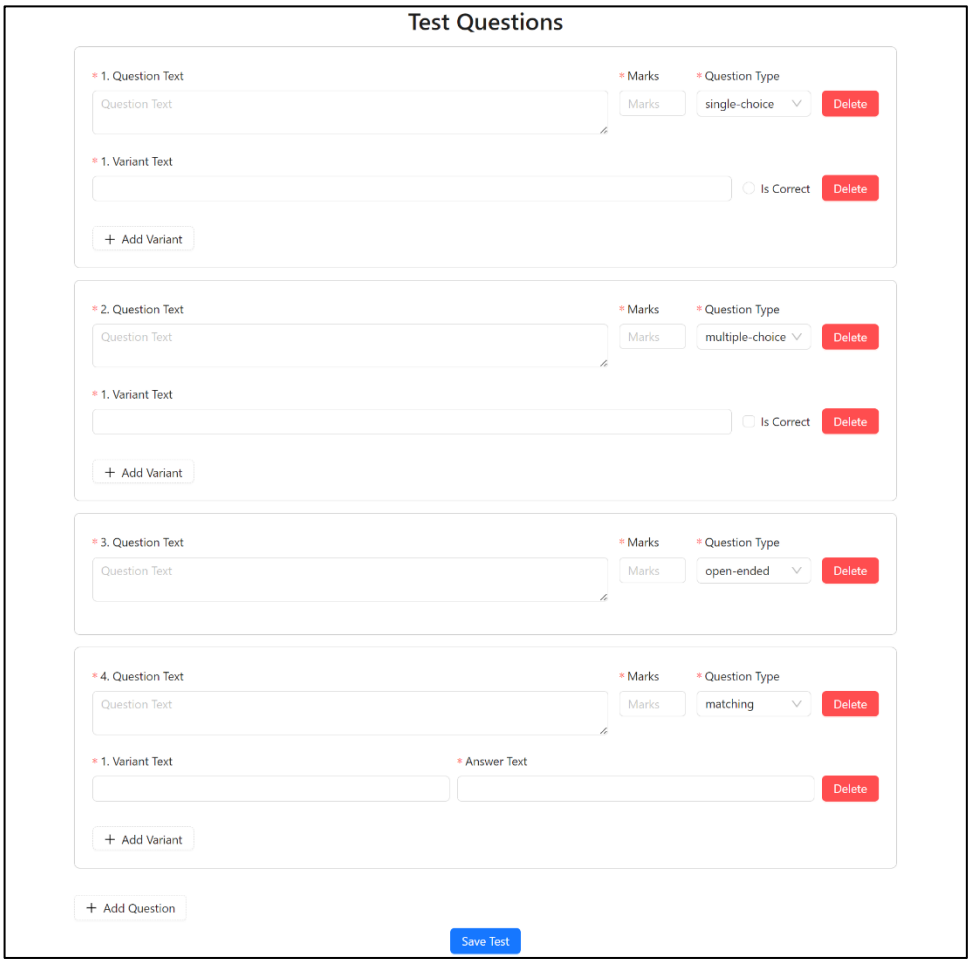

Рисунок 4.8 – Вигляд сторінки створення тесту (секція питань)

На сторінці «Tests List» (рис. 4.9) відображаються назви тестів, предмети та дати їх створення. Для кожного тесту доступні кнопки «Assign», «Edit» та «Delete». Кнопка «Assign» активується тільки після вибору класу з випадаючого списку. У цьому списку доступні тільки ті класи, які вчитель додав у свій профіль. Класи, яким вже призначено тест, блокуються в списку. Після успішного призначення тесту з'являється повідомлення, і клас автоматично блокується в списку (рис. 4.10).

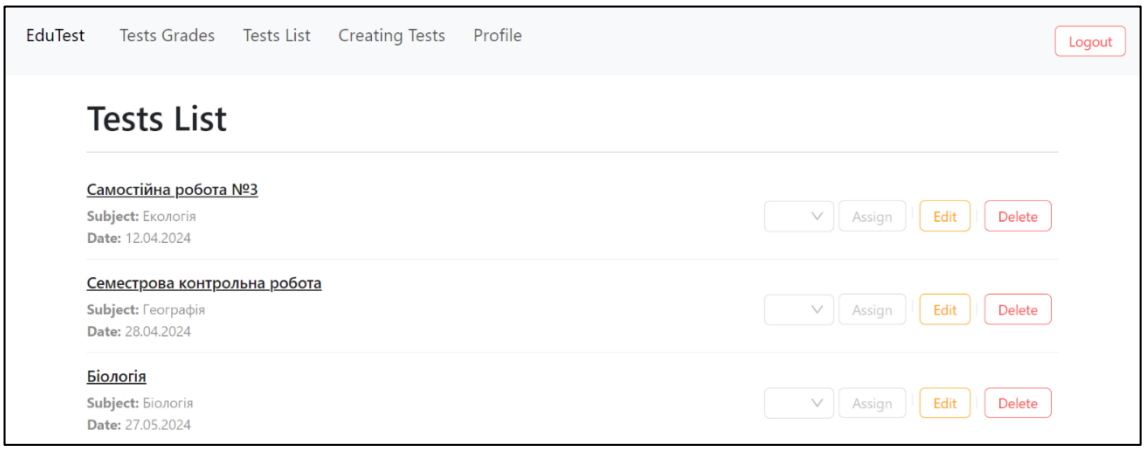

Рисунок 4.9 – Вигляд сторінки зі списком тестів

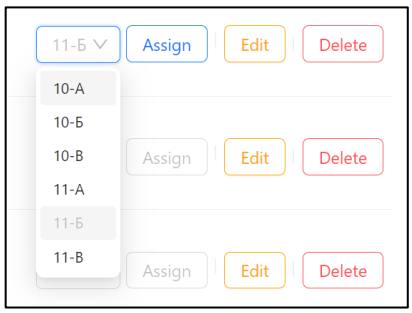

Рисунок 4.10 – Вигляд випадаючого списку з заблокованим елементом

При натисканні на назву тесту, яке підсвічується синім кольором при наводженні на нього курсору мишки, відкривається сторінка з відкритими питаннями до тесту (рис. 4.11). Якщо тест не мав відкритих питань, або учні ще не надали на нього відповіді, на сторінці буде повідомлення про відсутність відповідей. Якщо відповіді наявні, користувач бачить список усіх відповідей від усіх учнів, які їх надали. Кожний запис на сторінці містить повне ім'я учня, максимальну оцінку, яку вчитель поставив при створенні

тесту за це питання, текст питання та саму відповідь на нього. Після введення оцінки та натискання кнопки для її збереження поле і кнопка блокуються, залишаючи видимим поставлену оцінку.

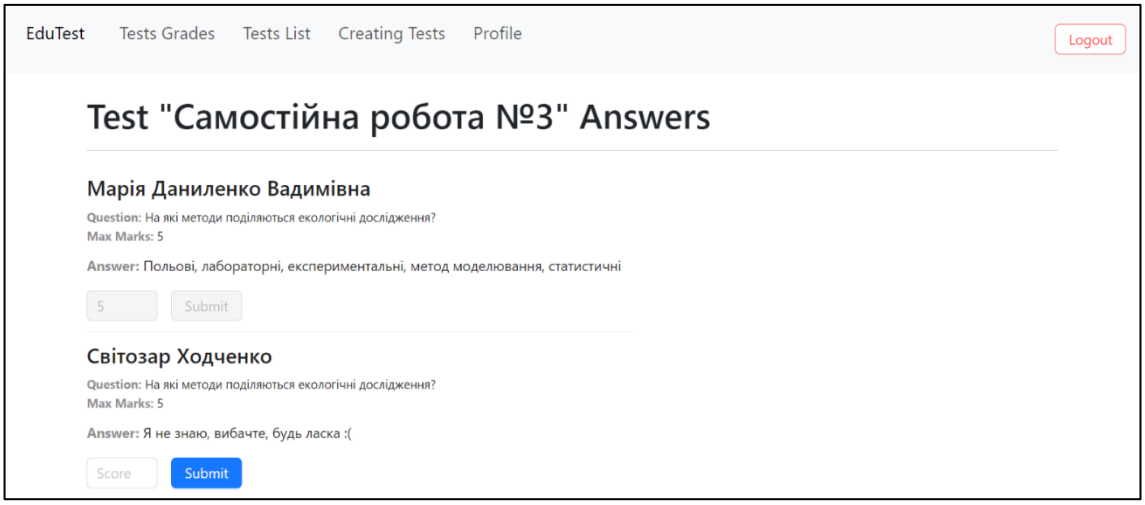

Рисунок 4.11 – Вигляд сторінки з відкритими відповідями до тесту

Якщо на сторінці зі списком тестів натиснути на кнопку для редагування, відкриється сторінка, ідентична на сторінку для створення не лише візуально, а й функціонально, однак заповнена усіма даними створеного тесту (рис. 4.12).

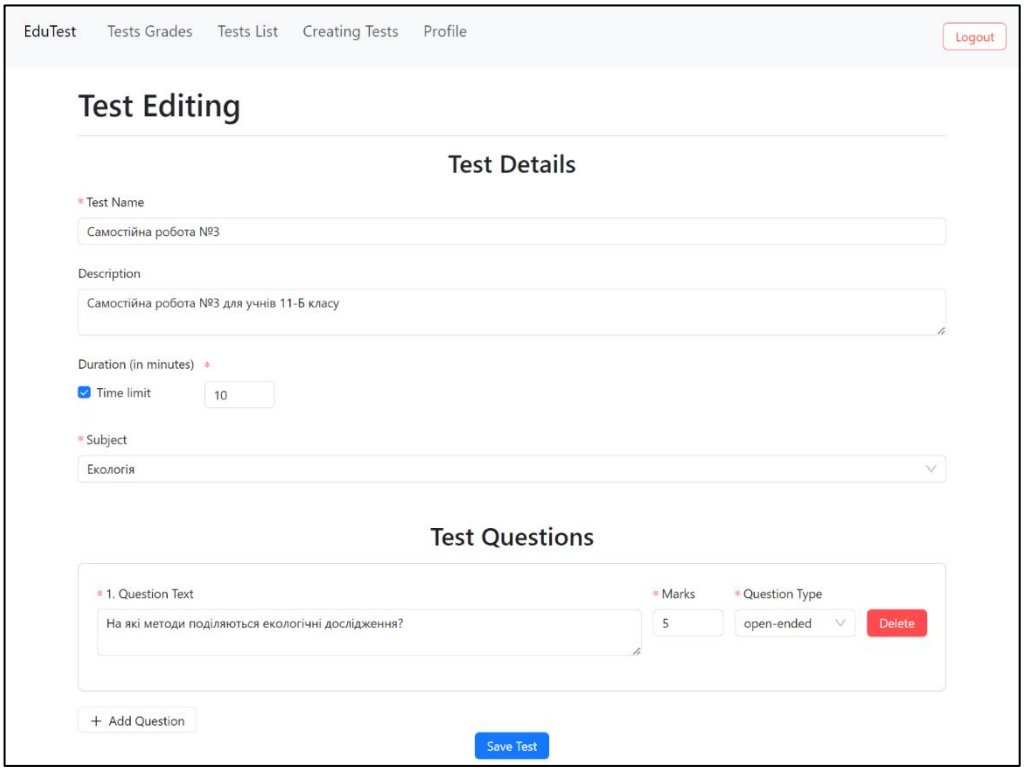

Рисунок 4.12 – Вигляд сторінки редагування тесту

На сторінці «Tests Grades» (рис. 4.13) відображаються всі тести, які були призначені учням. Користувач може розгорнути деталі кожного тесту, натиснувши на плюс поруч з його назвою, що відкриє таблицю з результатами всіх учнів, яким тест був призначений. Якщо тест ще не пройдено, в стовпці «State» відображається відповідне повідомлення. Кожен стовпець таблиці можна відсортувати для кращого перегляду результатів.

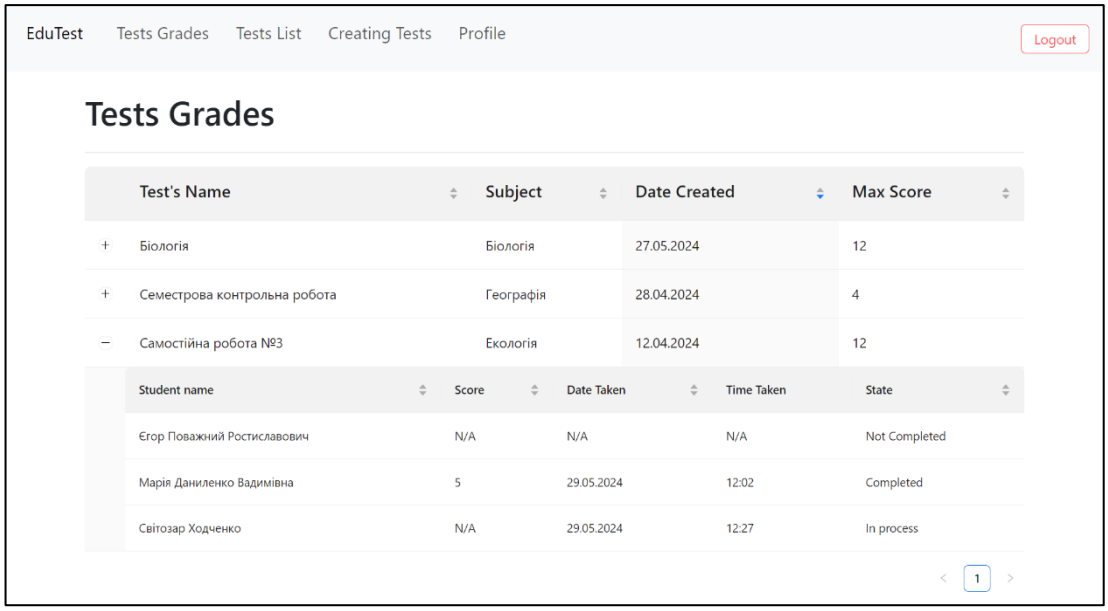

Рисунок 4.13 – Вигляд сторінки з оцінками за тести вчителя

Повернемося до профілю учня. На сторінці «Tests» (рис. 4.14) відображаються доступні для проходження тести. Кожен тест містить інформацію про предмет і дату. Справа від інформації про кожен тест розташована кнопка «Take Test». При натисканні на неї з’являється діалогове вікно, для підтвердження наміру користувача розпочати тест, щоб уникнути випадкового старту.

Рисунок 4.14 – Вигляд сторінки з тестами для проходження

Після підтвердження наміру пройти тест, відкривається сторінка для його проходження «Test Passing» (рис. 4.15). Вгорі відображено назву та опис тесту. Сторінка включає питання та поля для введення відповідей або вибору варіанту. При наявності часового обмеження, таймер відліку часу знаходиться в нижньому правому кутку та є видимим під час прокрутки. Після завершення тесту, натиснувши «Submit Test», з'являється діалогове вікно для підтвердження завершення. У разі закінчення відведеного часу, тест автоматично завершується, результати фіксуються, і користувач бачить оцінку на відповідній сторінці (рис. 4.16).

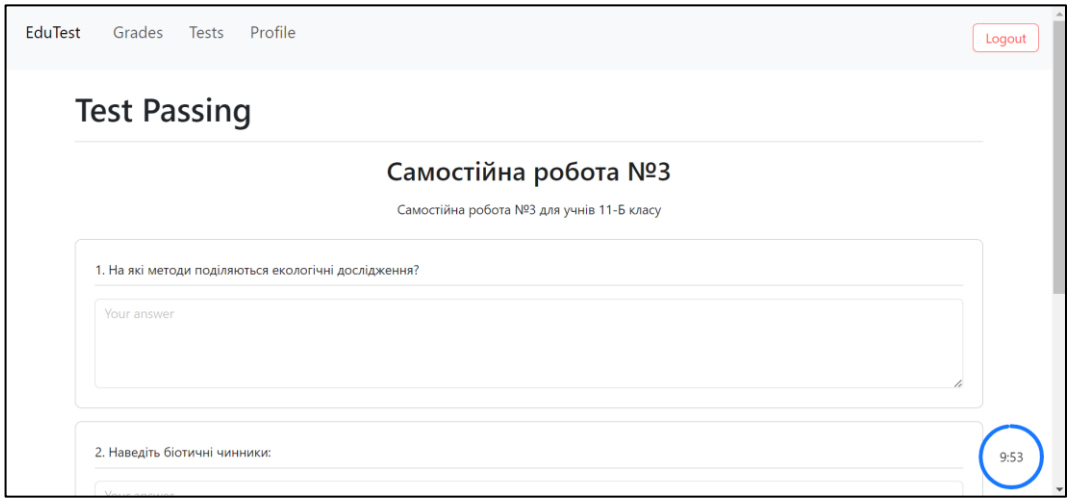

Рисунок 4.15 – Вигляд сторінки під час проходження тесту

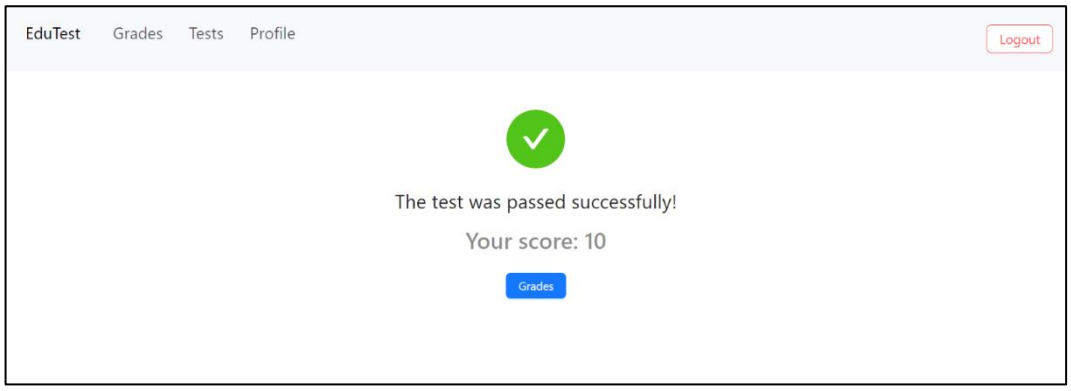

Рисунок 4.16 – Вигляд сторінки після завершення тесту

На сторінці «Grades» (рис. 4.17) користувач може переглянути оцінки за пройдені тести. Для тестів із відкритими питаннями, оцінки відображаються після перевірки вчителем. Сторінка включає таблицю із

назвами тестів, предметами, датами проходження та оцінками. Усі колонки таблиці можна сортувати для зручного перегляду.

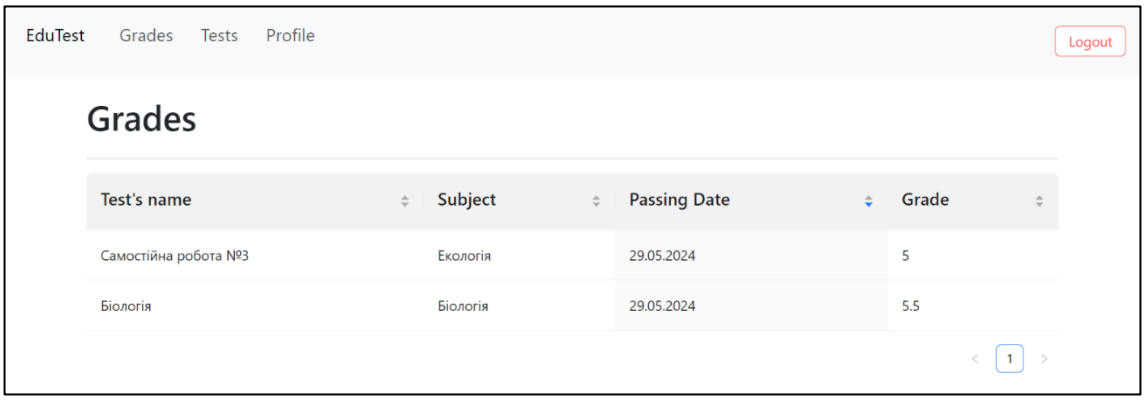

Рисунок 4.17 – Вигляд сторінки з оцінками учня

Як можна побачити на скріншотах з вебзастосунку, дизайн є досить простим, він має два основних кольори в дизайні: сірий та білий, а також три кольори для інтерактивних елементів: червоний, помаранчевий та синій. Використання цих кольорів дозволяє легко орієнтуватися на сторінці не напрягаючи зорові нерви непотрібними деталями.

Дизайн вебзастосунку для оцінювання навчальних досягнень учнів «EduTest» брав за основу чистий і мінімалістичний підхід, який зосереджує увагу користувача на функціональності без зайвих відволікаючих елементів. Використання двох основних кольорів, сірого та білого, створює спокійний візуальний фон, який не втомлює під час довготривалої роботи. Червоний, помаранчевий та синій кольори застосовуються для інтерактивних елементів, таких як кнопки та посилання, допомагаючи користувачам легко ідентифікувати можливості для взаємодії. Це ефективно покращує навігацію та взаємодію зі сторінками, роблячи досвід користувача більш інтуїтивно зрозумілим.

4.2 Програмна реалізація back-end частини

Для зберігання даних було використано MS SQL Server через його відмінну інтеграцію з іншими інструментами Microsoft, що спростило його впровадження в серверну частину вебзастосунку. Структура бази даних

включає багато таблиць для оптимального доступу до даних, вирішуючи завдання нормалізації (рис. 4.18).

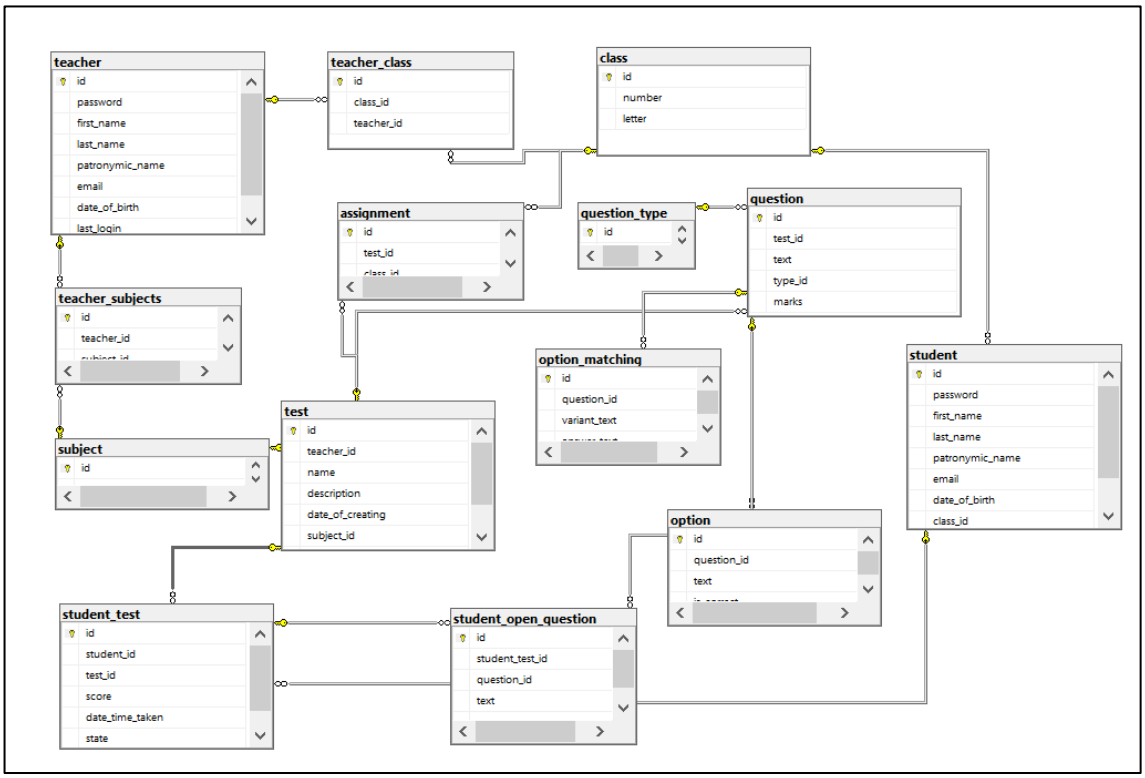

Рисунок 4.18 – Діаграма бази даних

Основними компонентами програмної реалізації back-end є вже зазначені Entity Framework Core, який використовується для взаємодії з базою даних через моделі, та ASP.NET Core, що застосовано для створення API з контролерами, які обробляють запити до сервера. Ці бібліотеки та їх компоненти були підключені та налаштовані в головному виконавчому файлі програми Program.cs.

Підключення до бази даних реалізовано в класі DBConnection (рис. 4.19). Метод GetSqlConnection() інкапсулює створення об’єкта SqlConnection. Метод ExecuteQuery() використовується для виконання SQL-запитів до бази даних. Він приймає як параметри текст команди та словник параметрів для запобігання SQL-ін’єкції, відкриває з’єднання, виконує команду і читає результати. Результати серіалізуються у JSON формат перед поверненням, що забезпечує універсальність даних для клієнтської частини застосунку.

```
public class DBConnection
{
    1 reference
    protected static SqlConnection GetSqlConnection()
    {
        return new SqlConnection("Server=DESKTOP-GF8REUK\\SQLEXPRESS;Database=EduTestDB;Trusted_Connection=true;Encrypt=False");
    }
    1 reference
    public static string ExecuteQuery(string commandText, Dictionary<string, object> parameters = null)
    {
        using (SqlConnection conn = GetSqlConnection())
        {
            conn.Open();
            using (SqlCommand cmd = new SqlCommand(commandText, conn))
            {
                cmd.CommandType = System.Data.CommandType.Text;

                if (parameters != null)
                {
                    foreach (var parameter in parameters)
                    {
                        cmd.Parameters.AddWithValue(parameter.Key, parameter.Value);
                    }
                }

                using (SqlDataReader reader = cmd.ExecuteReader())
                {
                    var results = new List<string>();
                    while (reader.Read())
                    {
                        results.Add(reader.GetString(0));
                    }

                    return JsonConvert.SerializeObject(results);
                }
```

Рисунок 4.19 – Фрагмент коду контролеру DBConnection

У бекенді застосунку важливою складовою є AppDbContext, клас, що наслідується від DbContext з Entity Framework Core (рис. 4.20). AppDbContext включає властивості типу DbSet для кожного з класів моделей, що представляють таблиці у базі даних. Кожен DbSet дає можливість здійснювати запити до відповідної таблиці, додавати нові записи, оновлювати існуючі та видаляти їх. Ця структура забезпечує централізоване керування схемою даних застосунку.

```
public class AppDbContext : DbContext
{
    0 references
    public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)
    {
    }

    4 references
    public DbSet<Teacher> Teacher { get; set; }
    6 references
    public DbSet<Student> Student { get; set; }
    4 references
    public DbSet<Class> Class { get; set; }
    2 references
    public DbSet<Subject> Subject { get; set; }
    1 reference
    public DbSet<TeacherSubjects> Teacher_Subjects { get; set; }
    3 references
    public DbSet<TeacherClass> Teacher_Class { get; set; }
    13 references
    public DbSet<Test> Test { get; set; }
    7 references
    public DbSet<Question> Question { get; set; }
    8 references
    public DbSet<Option> Option { get; set; }
```

Рисунок 4.20 – Код класу-контексту AppDbContext

Як зазначено вище, для забезпечення легкої інтеграції розроблюваного вебзастосунку з іншими системами автентифікації, розроблено інтерфейс IUserAuthService. Цей інтерфейс реалізовується сервісом UserAuthService, екземпляр якого використовується в UserAuthController (рис. 4.21). Це дозволяє централізовано управляти процесами автентифікації та забезпечує високий рівень абстракції та гнучкості в роботі з різними автентифікаційними системами. Використання цього сервісу в контролері, який відповідає за автентифікацію користувачів, спрощує процес верифікації користувачів та забезпечує безпечний обмін даними.

```
[ApiController]
[Route("api/[controller]")]
1 reference
public class UserAuthController : ControllerBase
{
    private readonly IUserAuthService _userAuthService;
    private readonly AppDbContext _context;

    0 references
    public UserAuthController(IUserAuthService userAuthService, AppDbContext context)
    {
        _userAuthService = userAuthService;
        _context = context;
    }

    [AllowAnonymous]
    [HttpPost("login")]
    0 references
    public async Task<IActionResult> Authenticate([FromBody] UserAuth model)
    {
        var token = await _userAuthService.Authenticate(model.Email, model.Password);

        if (token == null)
        {
            return BadRequest(new { message = "Invalid email or password" });
        }

        return Ok(new { token });
    }
```

Рисунок 4.21 – Фрагмент коду контролеру UserAuthController

Після авторизації для надання користувачам доступу до обмежених функцій системи використовується генерація JWT. Для цього в системі впроваджено механізм, який включає використання SecurityKey, алгоритму шифрування HmacSha256, та класу JwtSecurityToken для збереження ідентифікатора користувача та його ролі. Токен генерується із використанням секретного ключа, який забезпечує захист від несанкціонованого доступу, і передається користувачу для подальшого використання в системі (рис. 4.22).

```
private string GenerateToken(int userId, string role)
{
    var key = new SymmetricSecurityKey(Encoding.ASCII.GetBytes("edu_test_123_very_long_secret_key_123"));
    var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

    var claims = new[]
    {
        new Claim(ClaimTypes.NameIdentifier, userId.ToString()),
        new Claim(ClaimTypes.Role, role),
    };

    var token = new JwtSecurityToken(
        claims: claims,
        expires: DateTime.UtcNow.AddDays(7),
        signingCredentials: credentials
    );

    return new JwtSecurityTokenHandler().WriteToken(token);
}
```

Рисунок 4.22 – Фрагмент коду сервісу UserAuthService

Для безпечного зберігання введених паролів використовується автоматичне кодування в моделі під час реєстрації, яке забезпечується методом EncryptPassword класу EncryptionService. Цей же пароль розкодовується для перевірки при авторизації користувача.

Після успішної авторизації, всі дані, які відображаються на сторінках користувача, завантажуються з бази даних через API. У ProfileController описано методи, які взаємодіють з профілем користувача. Для запобігання несанкціонованому доступу, контролер використовує атрибут [Authorize] для захисту функцій. Також у методах контролера активно використовуються контекст та моделі для зв’язку з базою даних (рис. 4.23).

```
[Authorize]
[HttpGet("student/{studentId}")]
0 references
public async Task<Student> GetStudentByIdAsync(int studentId)
{
    var student = await _context.Student
                        .Where(s => s.Id == studentId)
                        .FirstOrDefaultAsync();

    return student;
}
```

Рисунок 4.23 – Фрагмент коду контролеру ProfileController

Для управління тестами та даними, пов’язаними з тестуванням, використовується контролер TestsController, схожий на ProfileController, але з більшою кількістю методів. Він займається обробкою численних таблиць бази даних одночасно, зокрема методом CreateNewTestAsync (рис. 4.24), який є складним через множинні перевірки та обсяг даних, що потрібно зберегти.

Для оптимізації коду і спрощення роботи з базою даних використано підхід Extract Method, розділивши логіку на декілька дрібніших методів для збереження даних у кожну таблицю, але, незважаючи на це, велика кількість полів у моделях зберігає метод об’ємним (рис. 4.25).

```
[Authorize]
[HttpPost("create_new_test")]
0 references
public async Task<IActionResult> CreateNewTestAsync(TestData testData)
{
    var test = new Test
    {
        Teacher_Id = testData.TeacherId,
        Name = testData.TestName,
        Description = testData.Description,
        Duration = testData.IsTimeLimitEnabled ? testData.Duration : null,
        Max_Score = testData.MaxScore,
        Subject_Id = testData.SubjectId,
        Date_Of_Creating = DateTime.Now,
    };

    await PostTest(test);

    foreach (var questionData in testData.Questions)
    {
        var question = new Question
        {
            Test_Id = test.Id,
            Text = questionData.QuestionText,
            Type_Id = questionData.QuestionTypeId,
            Marks = questionData.Marks
        };

        await PostQuestion(question);
```

Рисунок 4.24 – Фрагменту коду функції для створення тесту

```
private async Task<IActionResult> PostTest(Test test)
{
    if (_context.Test == null)
    {
        return BadRequest(new { message = "Entity is null" });
    }
    _context.Test.Add(test);
    await _context.SaveChangesAsync();

    return Ok();
}
```

Рисунок 4.25 – Код методу, що виділений в окрему функцію

Метод рефакторингу Extract Method було застосовано для спрощення складних функцій шляхом розділення великих блоків коду на менші методи. Це значно підвищило читабельність і підтримуваність коду. Всі файли проєкту систематизовані в чітко структуровані папки, забезпечуючи ефективну навігацію та доступ до коду (рис. 4.26).

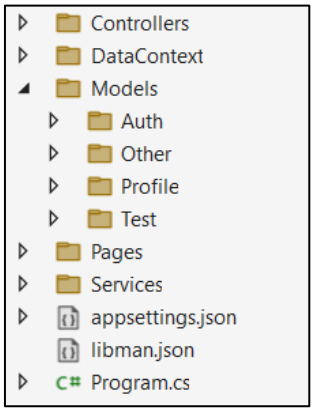

Рисунок 4.26 – Структура файлів серверної частини

Завдяки інтеграції бібліотек Entity Framework Core та ASP.NET Core, а також підтримці MS SQL Server, розробка відбувалася ефективно без потреби написання об’ємних SQL-запитів, значно спрощуючи взаємодію з базою даних.

4.3 Програмна реалізація front-end частини

Розробка фронтенду застосунку, реалізованого за допомогою JavaScript та бібліотеки React, починається із файлу App.js, який є вхідною точкою проєкту (рис. 4.27). У цьому файлі ініціюється основна конфігурація додатку: підключення необхідних бібліотек, компонентів та налаштування маршрутизації через компонент Router. Також, для оптимізації взаємодії з серверною частиною, створюється і налаштовується екземпляр QueryClient.

```
const queryClient = new QueryClient(); // Create a client

function App() {
  return (
    <QueryClientProvider client={queryClient}>
      <Router>
        <div className="App">
          <Header />
        </div>
        <Routes>
          <Route path="/" element={<AuthRedirect />} />
          <Route path="/create-test" element={<TestCreatingPage />} />
          <Route path="/take-test/:testId" element={<TestTakingPage />} />
          <Route path="/edit-test/:testId" element={<TestEditPage />} />
          <Route path="/grades" element={<StudentGradesPage />} />
          <Route path="/tests-list" element={<TestsListPage />} />
          <Route path="/student-profile/:id" element={<StudentProfilePage />} />
          <Route path="/teacher-profile/:id" element={<TeacherProfilePage />} />
```

Рисунок 4.27 – Фрагмент коду файлу App.js

Для забезпечення зв’язку з API, реалізованим на бекенді, фронтенд частина використовує бібліотеку axios (рис 4.28). Axios дозволяє легко відправляти асинхронні HTTP-запити до бекенду, отримуючи або надсилаючи дані. Результати запиту обробляються у хуках, таких як useEffect(), для асинхронного завантаження даних під час ініціалізації компоненту. В разі успішного отримання відповіді, дані зберігаються в стані компоненту за допомогою функції set. Додатково, для забезпечення правильного відображення формату дати і часу, використовується бібліотека moment, яка дозволяє адаптувати часові мітки до зручного для користувача формату.

```
axios
  .get("/api/Profile/subject")
  .then((response) => {
    setSubjects(response.data);
  })
  .catch((error) => {
    console.error("Failed to fetch subjects:", error);
  });
```

Рисунок 4.28 – Фрагмент коду використання axios для отримання списку шкільних предметів

Для управління автентифікацією та доступом до ресурсів застосунку, використовується контекст AuthContext (рис. 4.29). Цей контекст відповідає за декодування JWT-токена, збереженого в localStorage, де закодовано ідентифікатор та роль користувача. З отриманих даних формуються стани користувача, що включають його ідентифікатор, роль та статус автентифікації. Це дозволяє динамічно змінювати відображення компонентів відповідно до статусу автентифікації користувача, забезпечуючи таким чином контроль доступу до різних частин застосунку.

```
const AuthContext = createContext();

export function useAuth() {
  return useContext(AuthContext);
}

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [isAuthenticated, setIsAuthenticated] = useState(false);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const token = localStorage.getItem('token');
    if (token) {
      try {
        const decoded = jwtDecode(token);
        setUser({
          id: decoded["http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier"],
          role: decoded["http://schemas.microsoft.com/ws/2008/06/identity/claims/role"],
        });
        setIsAuthenticated(true);
      } catch (error) {
        console.error("Token decoding failed", error);
      }
      axios.defaults.headers.common['Authorization'] = `Bearer ${token}`;
    }
    setLoading(false);
```

Рисунок 4.29 – Фрагмент коду контексту AuthContext

Дані для компонентів тестів завантажуються через API та управляються за допомогою хуків useState, які зберігають стан компонентів. На основі цих даних, компоненти динамічно відтворюють вміст сторінки, використовуючи JavaScript методи map та forEach для ітерації по масивах отриманих даних. Це забезпечує адаптивне відображення змінних вмісту, відповідно до дій користувача (рис. 4.30).

```
{testData.questions.map((question, qIndex) => (
  <Card
    key={qIndex}
    style={{ borderColor: "#c4c4c4", marginBottom: "16px" }}
  >
    <p style={{ marginBottom: "0px" }}>
      {qIndex + 1}. {question.questionText}
    </p>
    <hr style={{ marginTop: "7px" }} />
    {question.questionTypeId === 1 && (
      <Form.Item name={`question-${qIndex}`} style={{ margin: "0px" }}>
        <Radio.Group
          style={{ display: "flex", flexDirection: "column" }}
        >
          {question.variants.map((variant, vIndex) => (
            <Radio
```

Рисунок 4.30 – Фрагмент коду динамічного створення сторінки для проходження тесту

Для забезпечення кращого користувацького досвіду під час завантаження даних, замість стандартного білого екрану відображається анімаційне коло завантаження від бібліотеки AntD. Проте, під час розробки виявилося, що використання лише axios для взаємодії з API призводить до довгих часів завантаження через велику кількість запитів та оброблення даних, що іноді досягало 8 секунд, що супроводжувалось анімацією завантаження. Щоб вирішити цю проблему, було застосовано бібліотеку React Query. Її можливості кешування та синхронізації даних дозволили зменшити час завантаження до прийнятних у вимогах 5 секунд. Цей крок значно покращив ефективність завантаження сторінок, що вимагають багато API взаємодій (рис. 4.31).

```
const { data, isLoading, isError, error } = useQuery(
  ["testsData", user.id],
  () => fetchTestsAndClasses(user.id, token)
);

const { tests, classes } = data || { tests: [], classes: [] };

const deleteMutation = useMutation(
  (testId) =>
    axios.delete(`/api/Tests/${testId}`, {
      headers: { Authorization: `Bearer ${token}` },
    }),
  {
    onSuccess: () => {
      queryClient.invalidateQueries(["testsData", user.id]);
      notification.success({
        message: "Test Deleted",
        description: "The test has been successfully deleted.",
      });
    },
```

Рисунок 4.31 –Приклад використання React Query на стоырнці з тестами викладача

Структурування файлів фронтенду в різні папки значно полегшує навігацію по проєкту та підтримку коду (рис. 4.32). Це організаційний підхід дозволяє ефективно управляти великими об’ємами коду та компонентів.

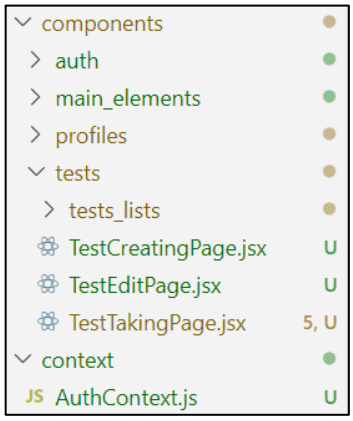

Рисунок 4.32 – Структура файлів клієнтської частини

Використання API для взаємодії між бекендом і фронтендом значно спростило розробку, усунувши необхідність безпосереднього з'єднання з серверною частиною. Це дозволяє зосередитися на користувацькому досвіді та інтерфейсі без зайвого клопоту з серверною логікою.

Бібліотеки AntD та Bootstrap надали потужні інструменти для дизайну, що не потребують написання великих CSS класів. Це зробило процес розробки швидшим і більш ефективним, дозволяючи створювати візуально привабливий і зручний в користуванні вебзастосунок.

Висновки до розділу 4

В четвертому розділі кваліфікаційної роботи бакалавра було описано реалізацію вебзастосунку для оцінювання навчальних досягнень учнів в ЗЗСО. Проведено детальний огляд користувацького інтерфейсу, який забезпечує високий рівень користувацького досвіду та зручності у використанні системи.

Також розглянуто реалізацію бекенду та фронтенду, технологіями C#, JavaScript, ASP.NET Core, та Entity Framework Core, що сприяють інтеграції серверної та клієнтської частин застосунку. Окреслено використання бібліотеки React Query для оптимізації запитів до API, значно знижуючи час завантаження сторінок. Додаткова увага приділена модульній організації проєкту, що забезпечує ефективне управління кодом та розвиток системи.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи бакалавра розроблено вебзастосунок оцінювання навчальних досягнень учнів для удосконалення внутрішньої системи забезпечення якості освіти в закладах загальної середньої освіти.

Для досягнення поставленої мети виконано поставлені завдання:

- проведено аналіз предметної області та систем із подібним функціоналом;
- змодельовано необхідні для системи діаграми;
- обрано технології реалізації програмного коду;
- розроблено серверну частину проєкту;
- реалізовано користувацький інтерфейс.

Виконано порівняння існуючих аналогів, зокрема на українському ринку, результатом якого стало виявлення переваг і недоліків кожного, виявлено їх архітектуру та мови реалізації, досліджено їх функціонал.

Сформовано вимоги до програмного забезпечення та проведено моделювання застосунку за допомогою створення UML-діаграм, що відповідають потребам системи.

Вибрано технології для реалізації проєкту з детальним аналізом кожної. Використовуючи обрані інструменти, розроблено серверну та клієнтську частину, описуючи їх основні складові та труднощі, які були вирішені при реалізації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Освітні індикатори національного та місцевого рівнів. *Power BI*. URL: https://app.powerbi.com/view?r=eyJrIjoiYWZiMDA1YjItMmY2MC00NzVmLWEyN2ItNTZkMWEwNmQ1OTJjIiwidCI6IjcxNmVkNjRjLTI2ZTItNGI5ZS04Yjg4LTcyZjllZDlhNWU4MyIsImMiOjl9&pageName=ReportSection (дата звернення: 24.03.2024).

2. Онлайн-тести «На Урок». *Освітній проект «На Урок» для вчителів*. URL: https://naurok.com.ua/test (дата звернення: 02.05.2024).

3. Конструктор тестів. *Всеосвіта - Національна освітня платформа*. URL: https://vseosvita.ua/test (дата звернення: 02.05.2024).

4. Google Форми. *Google Forms*. URL: https://docs.google.com/forms (дата звернення: 02.05.2024).

5. Kulak D., Guiney E. Use Cases: Requirements in Context, Second Edition. Addison-Wesley Professional, 2003. 272 p.

6. Node.js v21.7.1 Documentation. *Node.js – Run JavaScript Everywhere*. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 24.03.2024).

7. What's new in SQL Server 2019 - SQL Server. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/sql/sql-server/what-s-new-in-sql-server-2019?view=sql-server-ver16 (дата звернення: 24.03.2024).

8. Fowler M. UML distilled: A brief guide to the standard object modeling language. 3rd Edition. Boston, MA : Addison-Wesley, 2004. 175 p.

9. Kumar K., Azad S. K. Database normalization design pattern. 2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics (UPCON), Mathura, 26–28 October. 2017. URL: https://doi.org/10.1109/upcon.2017.8251067 (дата звернення: 26.03.2024).

10. Overview of Entity Framework Core - EF Core. *Microsoft Learn*. URL: https://learn.microsoft.com/en-us/ef/core/ (date of access: 22.05.2024).

11. Overview - A tour of C#. *Microsoft Learn*. URL: https://learn.microsoft.com/uk-ua/dotnet/csharp/tour-of-csharp/overview (date of access: 21.05.2024).

12. Strauss D. Exploring Advanced Features in C#: Enhance Your Code and Productivity. Apress, 2019. 308 p.

13. Comparative Studies of Six Programming Languages / Z. Alomari et al. 2015. 71 p. (Preprint. arXiv:1504.00693). URL: https://doi.org/10.48550/arXiv.1504.00693 (date of access: 21.05.2024).

14. Svekis L. L., Putten M. v., Percival R. JavaScript from Beginner to Professional: Learn JavaScript Quickly by Building Fun, Interactive, and Dynamic Web Apps, Games, and Pages. Packt Publishing, Limited, 2021.

15. PHP vs. JavaScript: Choose the Right Tech for Your Project. *Cloudways*. URL: https://www.cloudways.com/blog/php-vs-javascript/ (date of access: 21.05.2024).

16. Quick Start – React. *React*. URL: https://react.dev/learn (дата звернення: 24.03.2024).

17. Banks A., Porcello E. Learning React: Functional Web Development with React and Redux. O'Reilly Media, 2017. 350 p.

18. Feature Overview v6.23.1. *React Router*. URL: https://reactrouter.com/en/main/start/overview (date of access: 21.05.2024).

19. Getting Started | Axios Docs. *Axios*. URL: https://axios-http.com/docs/intro (date of access: 21.05.2024).

20. Overview | TanStack Query React Docs. *TanStack*. URL: https://tanstack.com/query/latest/docs/framework/react/overview (date of access: 21.05.2024).

21. Moment.js | Docs. *Moment.js*. URL: https://momentjs.com/docs/ (date of access: 21.05.2024).

22. jwt-decode. *npm*. URL: https://www.npmjs.com/package/jwt-decode (date of access: 21.05.2024).

23. Get started with Bootstrap. *Bootstrap*. URL: https://getbootstrap.com/docs/5.2/getting-started/introduction/ (дата звернення: 24.03.2024).

24. Components Overview - Ant Design. *Ant Design - The world's second most popular React UI framework*. URL: https://ant.design/components/overview (дата звернення: 24.03.2024).

25. Overview of ASP.NET Core. *Microsoft Learn*. URL: https://learn.microsoft.com/uk-ua/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0 (date of access: 22.05.2024).

26. Esposito D. Programming ASP. NET Core. Microsoft Press, 2018. 416 p.

27. Authentication and authorization in minimal APIs. *Microsoft Learn*. URL: https://learn.microsoft.com/uk-ua/aspnet/core/fundamentals/minimal-apis/security?view=aspnetcore-8.0 (date of access: 22.05.2024).

28. Microsoft.AspNetCore.Mvc.NewtonsoftJson Namespace. *Microsoft Learn*. URL: https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.newtonsoftjson?view=aspnetcore-8.0 (date of access: 22.05.2024).

29. Get started with Swashbuckle and ASP.NET Core. *Microsoft Learn*. URL: https://learn.microsoft.com/uk-ua/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-8.0&tabs=visual-studio (date of access: 22.05.2024).

30. API Documentation Tools. *Swagger*. URL: https://swagger.io/solutions/api-documentation (date of access: 22.05.2024).

31. NuGet Gallery | Home. *NuGet*. URL: https://www.nuget.org/ (date of access: 22.05.2024).

32. Smith J. P. Entity Framework Core in Action, Second Edition. Manning Publications Co. LLC, 2021. 624 p.

33. Microsoft SQL Server Database Provider - EF Core. *Microsoft Learn*. URL: https://learn.microsoft.com/uk-ua/ef/core/providers/sql-server/?tabs=dotnet-core-cli (date of access: 22.05.2024).

34. Entity Framework Core tools reference - EF Core. *Microsoft Learn*. URL: https://learn.microsoft.com/en-us/ef/core/cli/ (date of access: 22.05.2024).

35. SQL Server 2019. *Microsoft*. URL: https://www.microsoft.com/uk-ua/sql-server/sql-server-2019 (date of access: 22.05.2024).

36. Introducing Microsoft SQL Server 2019 / K. Gorman et al. Packt Publishing Ltd., 2019. 489 p.