Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка вебсистеми для проведення тестувань з використанням Node.js, React.js та PostgreSQL

Виконав: студент IV курсу, групи СНс-41
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Чекановський А.Б.
(підпис) (прізвище та ініціали)

Керівник _____ Гром’як Р.С.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Литвиненко Я.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Кульчицький Т.Р.
(підпис) (прізвище та ініціали)

Тернопіль
2023

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Чекановському Андрію Богдановичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка вебсистеми для проведення тестувань з використанням Node.js, React.js та PostgreSQL

Керівник роботи Гром'як Роман Сильвестрович, кандидат фізико-математичних наук, доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «7» лютого 2023 року № 4/7-133 .

2. Термін подання студентом завершеної роботи 2023р.

3. Вихідні дані до роботи Літературні та інтернет джерела інформації про технології розробки вебсистеми для проведення тестувань з використанням Node.js, React.js та PostgreSQL

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Постановка задачі та формування вимог до вебсистеми для проведення тестувань. Аналіз предметної області. Формування вимог до вебсистеми для проведення тестувань. Пошук актантів та варіантів використання. Вибір оптимального методу вирішення задачі. Вибір середовища розробки. Обґрунтування використовуваних технологій. Висновок до першого розділу. 2. Проєктування та реалізація вебсистеми для проведення тестувань. Структурна модель вебсистеми для проведення тестувань. Проєктування поведінки вебсистеми для проведення тестувань. Розробка моделей даних вебсистеми для проведення тестувань. Перелік інформаційних сутностей та способів їх зберігання. Проєктування концептуальної моделі даних. Проєктування логічної та фізичної моделей даних. Проєктування інтерфейсу вебсистеми для проведення тестувань. Розробка основних модулів вебсистеми для проведення тестувань. Тестування вебсистеми для проведення тестувань. Висновок до другого розділу. 3. Безпека життєдіяльності, основи охорони праці. Соціальні та психологічні фактори ризику при користуванні вебсистемою для проведення тестувань. Психофізіологічне розвантаження для користувачів вебсистеми. Висновок до третього розділу Висновки. Перелік джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Титульний слайд. 2. Мета, об'єкт дослідження. 3. Аналіз предметної області. 4. Вимоги до вебсистеми. 5. Варіанти використання. 6. Використані технології та програмне забезпечення. 7. Архітектура та структура вебсистеми. 8. Сутності БД. 9. Користувацька частина вебсистеми. 10. Серверна частина вебсистеми. 11. Інтернаціоналізація. 12. Інтерфейс. 13. Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			

7. Дата видачі завдання 23 січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	23.01.2023	*Виконано*
2.	Підбір джерел про технології розробки вебсистеми для	24.01.2023-26.01.2023	*Виконано*
	проведення тестувань з використанням Node.js,		
	React.js, PostgreSQL		
3.	Опрацювання джерел по темі кваліфікаційної роботи	27.01.2023-31.01.2023	*Виконано*
4.	Виконання дослідження щодо розробки вебсистеми для проведення тестувань	01.02.2023-07.02.2023	*Виконано*
	Розроблення функціоналу вебсистеми		
5.	Оформлення розділу «Постановка задачі та	08.02.2023-09.02.2023	*Виконано*
	формування вимог до вебсистеми для проведення		
	тестувань»		
6.	Оформлення розділу «Проєктування та реалізація	10.02.2023-12.02.2023	*Виконано*
	вебсистеми для проведення тестувань»		
7.	Виконання завдання до підрозділу «Соціальні та	05.06.2023-06.06.2023	*Виконано*
	психологічні фактори ризику при користуванні		
	вебсистемою для проведення тестувань»		
8.	Виконання завдання до підрозділу «Психофізіологічне	07.06.2023-08.06.2023	*Виконано*
	розвантаження для користувачів вебсистеми»		
9.	Оформлення кваліфікаційної роботи	09.06.2023-11.06.2023	*Виконано*
10.	Нормоконтроль	12.06.2023-13.06.2023	*Виконано*
11.	Перевірка на плагіат		*Виконано*
12.	Попередній захист кваліфікаційної роботи		*Виконано*
13.	Захист кваліфікаційної роботи		

Студент _____ Чекановський А.Б.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Гром'як Р.С.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка вебсистеми для проведення тестувань з використанням Node.js, React.js та PostgreSQL // Кваліфікаційна робота освітнього рівня «Бакалавр» // Чекановський Андрій Богданович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп’ютерних наук, група СНс-41 // Тернопіль, 2023 // С. – 63, рис. – 28, табл. – 2, кресл. – 0, додат. – 7, бібліогр. – 34.

Ключові слова: система тестувань, розробка, інтерфейс, сервер, база даних, javascript, react, nodejs, express, postgresql.

Кваліфікаційна робота присвячена розробці вебсистеми для проведення тестувань з використанням Node.js, React.js та PostgreSQL.

Метою даної роботи є розробка системи для проведення тестувань.

У першому розділі кваліфікаційної роботи було проаналізовано предметну область, сформовано вимоги до застосунку, здійснено оцінку методів розв’язання поставленої задачі та обґрунтовано вибір середовища розробки та використовуваних технологій.

У другому розділі було здійснено проєктування та реалізація застосунку, а саме: побудовано структурну модель вебзастосунку, змодельовано його архітектуру, спроєктовано структуру та поведінку, розроблено моделі даних, описано розробку основних елементів системи.

В третьому розділі кваліфікаційної роботи розглянуто питання з безпеки життєдіяльності та охорони праці. Було проаналізовано соціальні та психологічні фактори ризику, а також психофізіологічне розвантаження для користувачів розробленої вебсистеми.

ANNOTATION

Web-based system development for testing using Node.js, React.js and PostgreSQL // Bachelor's degree qualification work // Chekanovskyi Andrii Bogdanovich // Ternopil Ivan Pul'uj National Technical University, Faculty of Computer Information Systems and Software Engineering, Computer Science Department, group SNs–41 // Ternopil, 2023 // P. – 63, fig. – 28, tbl. – 2, drawing – 0, add. – 7, ref. – 34.

Keywords: testing system, development, interface, server, database, javascript, react, nodejs, express, postgresql.

The qualification work is devoted to the development of a web system for conducting tests using Node.js, React.js and PostgreSQL.

The purpose of this work is to develop a system for conducting testing.

In the first section of the qualification work, the subject area was analyzed, application requirements were formed, methods for solving the problem were evaluated, and the choice of the development environment and technologies used was justified.

In the second chapter, the application was designed and implemented, namely: the structural model of the web application was built, its architecture was modeled, the structure and behavior were designed, data models were developed, and the development of the main elements of the system was described.

In the third chapter of the qualification work, issues related to life safety and labor protection are considered. Namely, social and psychological risk factors were analyzed, as well as psychophysiological unloading for users of the developed web system.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних.

ІС – інформаційна система.

СУБД – система управління базами даних.

API – application programming interface.

CORS – cross-origin resource sharing.

CSS – cascading style sheets.

DFD – data-flow diagram.

HTML – hypertext markup language.

HTTP – hypertext transfer protocol.

MVC – model-view-controller.

JS – javascript.

JSX – javascript extensible markup language.

ORM – object-relational mapping.

SCSS – sassy cascading style sheets.

TS – typescript.

UML – unified modeling language.

ЗМІСТ

ВСТУП

Актуальність теми. Процес навчання та засвоєння інформації часто супроводжується проведеннями проміжних і фінальної перевірок знань. Існують різні способи, щоб визначити рівень оволодіння матеріалом чи розуміння теми. У сучасному світі, одним із найпопулярніших методів, щоб з’ясувати це є онлайн-тестування. Це зручно, швидко та ефективно. До того ж, за наявності історії проходжень у таких системах та, аналізуючи результати проходження тестів, можна відслідковувати прогрес та визначати, в яких областях ще потрібно докласти зусиль для вивчення матеріалу.

Мета і задачі дослідження. Метою і задачею дослідження є розробити вебсистему для проведення тестувань з функціями створення тестів, проходження та відслідковування результатів, що дасть можливість вимірювати якість знань, здібностей та навичок.

Для досягнення поставленої мети було сформовано ряд наступних завдань:

- здійснити аналіз предметної області;
- сформувати вимоги до розроблюваної системи;
- виконати проєктування;
- розробити систему;
- провести тестування вебсистеми.

Об’єктом дослідження є система для проведення онлайн-тестувань.

Предметом дослідження є принципи розробки вебсистем та засоби їх практичної реалізації.

Практичне значення одержаних результатів. Основною ідеєю реалізованого вебзастосунку є надання інструменту, що використовуватиметься його користувачами у навчальних цілях. Проте, практичне використання даного вебзастосунку може спрямовуватись на досягнення різних цілей.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ ТА ФОРМУВАННЯ ВИМОГ ДО ВЕБСИСТЕМИ ДЛЯ ПРОВЕДЕННЯ ТЕСТУВАНЬ

1.1 Аналіз предметної області

Тестування – це форма оцінювання, в якій учасники намагаються правильно відповісти на запитання на одну або кілька конкретних тем. Тести можна використовувати як форму оцінювання в освіті та подібних сферах для вимірювання рівня знань, здібностей та навичок [1].

В освітньому контексті тест може бути замінений на вікторини, що зазвичай є також формою оцінювання студентів, але часто містить менше запитань меншої складності та вимагає менше часу для виконання, ніж тест. Сучасні заклади освіти, в більшості випадків, проводять тестування саме у онлайн форматі. Наприклад, на заняттях з математики онлайн-тестування може перевірити розуміння певної теми цього предмету.

Тестування може бути організовано у формі серії запитань, де кожне запитання має кілька варіантів відповідей для вибору. Ці варіанти можуть мати одну правильну відповідь або не мати її взагалі, залежно від типу тесту та постановки запитань. Також, у деяких випадках тестування може включати відкриті питання, де відповідь не обмежується варіантами, і не має конкретної правильної або неправильної відповіді. Результати тестування можуть показати слабкі чи сильні сторони особи, а після отримання результатів, стати плацдармом для удосконалення.

1.2 Формування вимог до вебсистеми для проведення тестувань

Найшвидшим способом оцінити та дізнатись свій рівень знань у певній галузі є проходження короткого онлайн-тестування. Отримавши результат, ми можемо зробити висновки та визначити, який матеріал було засвоєно, а над яким необхідно ще провести роботу. Тому, необхідно розробити вебзастосунок, що дозволятиме створювати тести за допомогою внутрішнього конструктора,

даватиме можливість публікувати, складати їх та відслідковувати успішність виконаних тестів.

Вебсистема для проведення тестувань повинна задовольняти ряд вимог, серед яких:

– дозволяти здійснювати реєстрацію та авторизацію;

– давати можливість складання тесту;

– дозволяти створення, модифікацію та видалення тесту зареєстрованим користувачем;

– давати можливість переглянути інформацію з описом тесту;

– дозволяти здійснювати пошук та фільтрування тестів;

– давати можливість перегляду та модифікації профілю користувача;

– після проходження тестування відображати результат, а для випадків, коли це зареєстрований користувач, зберігати результати у системі;

– надавати зареєстрованим користувачам статистичні дані з результатами їх тестувань та здійснених спроб;

– забезпечувати інтернаціоналізацію.

Вимогами до структури вебсистеми є:

– розділення фронтенд і бекенд частин як окремих модулів системи;

– використання компонентного підходу до реалізації інтерфейсу користувача;

– виокремлення модуля із стилями клієнтської частини застосунку;

– виокремлення модуля для роботи із HTTP-запитами на стороні клієнта;

– файли, компоненти та інші елементи системи, які можуть бути перевикористані поміщаються в окрему папку, яка повинна бути легкодоступною;

– логіка, яка пов’язана із конкретною сутністю повинна знаходитись поруч із іншою її логікою;

– файли конфігурацій, допоміжних функцій, проміжних обробників мають мати власний каталог із відповідною назвою.

Одним із ключових аспектів у розробці вебзастосунку є забезпечення цілісності та конфіденційності даних [2]. Враховуючи це, було сформульовано вимоги до захисту та доступу:

– дані, які вводить користувач повинні проходити валідацію для перевірки на відповідність введених значень правилам, передбаченим розробником та системою в цілому;

– доступ до непублічних даних має бути закритий для користувача, а можливість використання цих даних, повинна бути надана після проходження автентифікації;

– дані підключення до БД, секретні ключі та інші не публічні дані повинні зберігатись як змінні середовища;

– на стороні сервера, а саме у базі даних, пароль не повинен зберігатись у чистому вигляді, а повинен пройти хешування;

– під час побудови запитів дані введені користувачем повинні проходити очищення, щоб зменшити ризик впровадження зловмисного коду.

Нижче подано перелік вимог до інтерфейсу розроблюваної системи:

– сумісність із різними браузерами;

– інтуїтивність та передбачуваність елементів інтерфейсу;

– інтерфейс має бути лояльним до помилок користувачів;

– не повинен завдавати труднощів у використання для користувачів;

– впроваджує прості принципи до заповнення полів вводу;

– реалізований з використанням семантичних тегів.

1.3 Пошук актантів та варіантів використання

Для демонстрації того, яким чином система взаємодіє з користувачем або іншою системою існує діаграма прецедентів. Варіанти використання (прецеденти) та дійові особи на діаграмах варіантів використання описують, що робить система і як учасники її використовують, але не те, як система працює всередині [3].

В розроблюваній вебсистемі для тестувань наявні такі актори як:

– незареєстрований користувач (відвідувач);

– зареєстрований користувач.

Діаграма акторів системи подана на рисунку 1.1.

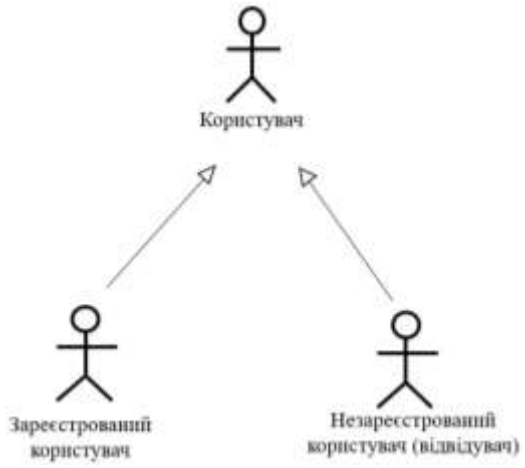

Рисунок 1.1 – Актори системи

Визначивши акторів застосунку нам необхідно зрозуміти, які функції може виконувати кожен з них. Відповідно до предметної області та з враховуванням обов’язків акторів, формуємо варіанти використання системи для онлайн-тестувань. Найменування та формулювання варіантів використання подано у додатку А.

Діаграми варіантів використання часто використовуються в контексті аналізу вимог або в процесі розробки вимог [4]. Їх можна використовувати для запису вимог до системи, яка складається з програмного або апаратного забезпечення. Вони добре підходять для аналізу поведінки системи, оскільки їх можна використовувати для визначення функціональності системи з точки зору користувача.

Таким чином, діаграма варіантів використання представляє ескіз системи, який вказує на призначення запланованої системи, включаючи межі та інтерфейси. Діаграма прецедентів зареєстрованого користувача зображена на рисунку 1.2.

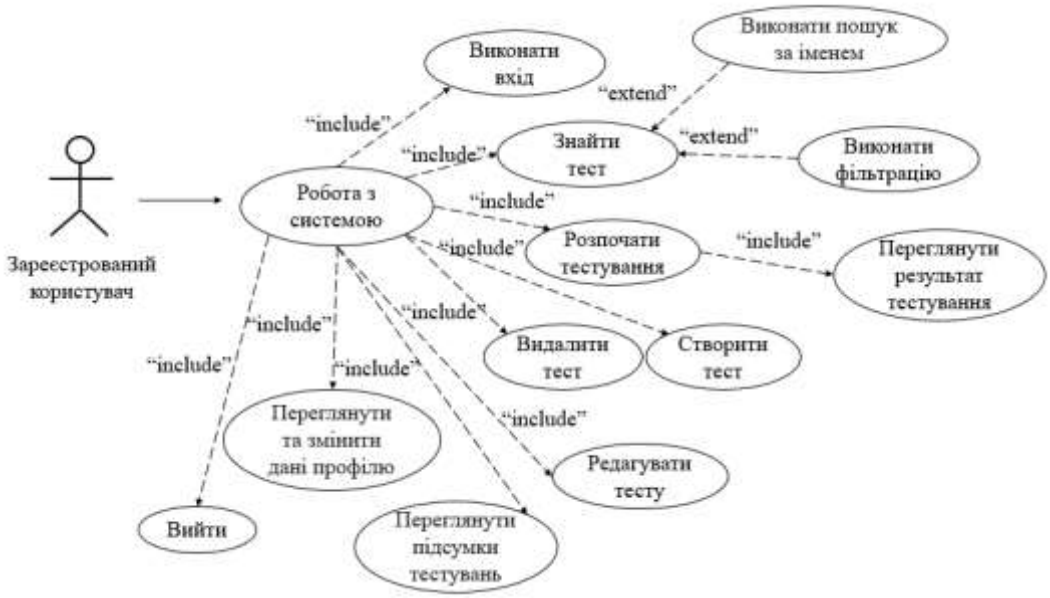

Рисунок 1.2 – Діаграма прецедентів зареєстрованого користувача

Діаграма прецедентів незареєстрованого користувача подана на рисунку 1.3.

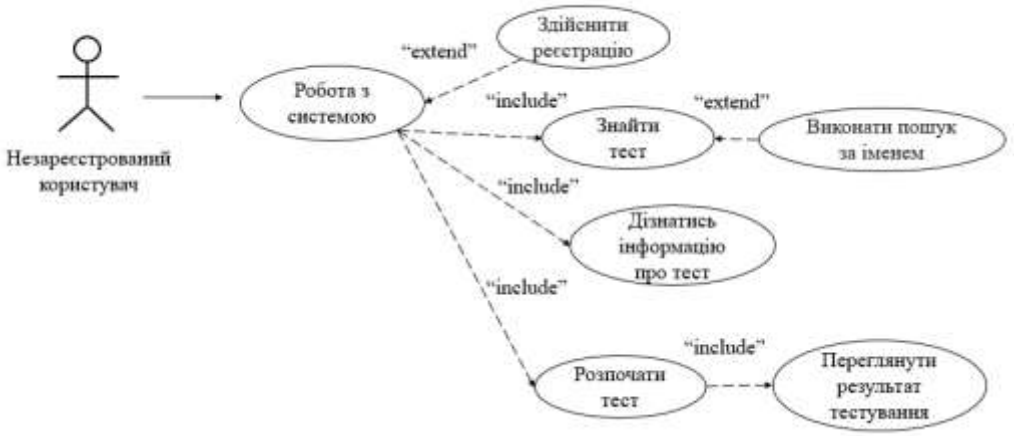

Рисунок 1.3 – Діаграма прецедентів незареєстрованого користувача

1.4 Вибір оптимального методу вирішення задачі

Для розробки вебзастосунків існує три базових інструменти розробки – це HTML, CSS, JavaScript [5]. Проте, сучасні вимоги до якості та швидкості розробки вимагають більш гнучких та ефективних рішень у галузі розробки. До

таких ми можемо віднести бібліотеки та фреймворки. Існують як фронтенд, так і бекенд фреймворки, які спрощують процес розробки.

Фреймворки веброзробки – це інструменти, які розробники використовують, щоб зробити розробку простішою та ефективнішою. Вони надають інтерфейси для доступу до часто використовуваних функцій, а також абстракцій, які полегшують розуміння та обробку складних речей. [6]

Бібліотеки – це набори файлів, програм, процедур, сценаріїв або функцій, які можна інтегрувати під час написання коду. Бібліотеки працюють, групуючи фрагменти коду разом, щоб інтегрувати функціональність, тож вам не потрібно писати код самостійно. Подібно до фреймворків, вони зменшують ризик неправильного кодування, роблять процес розробки більш ефективним і економлять гроші. [7]

До найбільш часто використовуваних технологій у сучасній веброзробці є [8]:

– для клієнтської частини: бібліотека React, фреймворки Angular та Vue;

– для серверної частини: Node.js (фреймворки Express.js, Nest.js), PHP (фреймворки Laravel, Symfony), Python (Django фреймворк), Ruby (Ruby on rails фреймворк), Java (фреймворк Spring Boot);

– СУБД: MongoDB, PostgreSQL, MySQL.

Було прийнято рішення реалізовувати серверну частину вебсистеми для проведення тестувань, використовуючи відомий фреймворк – Express.js. У порівнянні із написанням систем на чистому Node.js, швидкість та ефективність розробки за допомогою Express.js є більш високою.

Для реалізації клієнтської частина було обрано бібліотеку React.js, яка є однією з найпопулярніших інтерфейсних бібліотек, з великою екосистему інструментів та плагінів. [9]. Можливості React дозволяють підвищити продуктивність і заощадити час розробки.

1.5 Вибір середовища розробки

Для реалізації вебсистеми для тестувань було обрано інтегроване середовище розробки WebStorm. Середовище є багатофункціональним, автоматизує рутинні процеси в ході розробки та підтримує широкий спектр мов програмування та синтаксисів. В основному, WebStorm зосереджений на веброзробку. Сюди входить підтримка таких технологій: TypeScript, Vue, Angular, React, Node.js, HTML, CSS та багато іншого [10].

Також сильними сторонами WebStorm є автозаповнення коду, визначення помилок у коді чи навіть його дублювання. Середовище глибоко аналізує структуру проєкту, що дозволяє з легкістю застосовувати автоімпорт, навігацію по файлах, класах, здійснювати пошук необхідних функцій, методів, компонентів і тд [11].

Іншими ключовими можливостями даного середовища є: вбудовані інструменти розробки та відстежування помилок, до прикладу, у WebStorm можна використовувати інтегрований HTTP-клієнт; достатньо велика кількість безкоштовних плагінів; ефективність роботи в команді, що супроводжується наданням різноманітної функціональності інструментів: ефективна робота з Git та GitHub, спільна конфігурація, кодування в режимі реального часу та навіть дзвінки; можливості до рефакторингу.

1.6 Обґрунтування використовуваних технологій

Розробка вебсистеми відбуватиметься із використанням наступних технологій: React.js – для створення інтерфейсу користувача, для реалізації серверної частини було обрано фреймворк Node.js, що має назву Express.js. Системою управління базою даних було обрано PostgreSQL.

React.js – це інтерфейсна бібліотека JavaScript на основі компонентів з відкритим кодом, яку розробники використовують для створення інтерфейсів користувача для односторінкових програм. Вона допомагає створювати простий та інтуїтивно зрозумілий код. Бібліотека пропонує потужне керування станом, діями та подіями. Також, перевагою React є гнучкість. Модульна структура робить його одним із найкращих інструментів [12]. Завдяки реалізації Virtual

DOM, що забезпечує оновлення лише необхідної частини програми, а не всієї, зростає також продуктивність застосунків [13].

Node.js постійно зростає та набуває популярності для розробки вебдодатків, але тепер його можливості зросли ще більше, оскільки він пропонує різноманітні переваги для бекенд-розробки. Express.js – це бекенд фреймворк із відкритим кодом для Node.js [14]. Оскільки для Express.js потрібен лише JavaScript, програмістам легше створювати вебдодатки та API [15].

Express.js є популярним інструментом завдяки таким перевагами: веброзробник може використовувати JavaScript як єдину мову як для клієнтської, так і для серверної розробки; код JavaScript інтерпретується через Google V8 JavaScript Engine за допомогою Node.js, таким чином, код впроваджується швидко та легко в ефективний спосіб; Express.js простий у налаштуванні та використанні відповідно до потреб; надає гнучку систему мідлварів, що в основному корисно для виконання додаткових завдань над об'єктами відповіді та запиту.

PostgreSQL – одна з найстаріших, але й найдосконаліших систем керування базами даних з відкритим кодом. Вона керується та підтримується спільнотою, що активно допомагає вирішити проблеми під час роботи з PostgreSQL. Його можна запускати на різних ОС – Windows, Linux і MacOS. Воно надаєш широкий функціонал, в тому числі дозволяє визначати власні типи даних, типи індексів [16]. Дане СУБД вимагає мінімального обслуговування через свою стабільність.

1.7 Висновок до першого розділу

У першому розділі кваліфікаційної роботи було проаналізовано предметну область вебсистеми для проведення тестувань, сформовано ряд вимог до вебсистеми, її структури, безпеки та інтерфейсу користувача. Також, було продемонстровано діаграми варіантів використання та показано, яким чином актори можуть взаємодіяти із системою. Обґрунтовано вибір методу вирішення задачі та описано технології, що використовуватимуться в ході реалізації системи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ВЕБСИСТЕМИ ДЛЯ ПРОВЕДЕННЯ ТЕСТУВАНЬ

2.1 Структурна модель вебсистеми для проведення тестувань

Структурну модель вебсистеми для проведення тестувань можна поділити на дві частина: перша частина – клієнтська сторона, друга – серверна.

Для розробки інтерфейсу користувача було виконано розділення логіки системи на наступні структурні елементи:

– components – містить компоненти інтерфейсу, що використовуються чи можуть бути використані в декількох місцях системи. Компоненти – є будівельними блоками будь-якого проєкту React [17]. Кожен компонент представляє собою файл із функцією, яка повертає JSX;

– hooks – складається із власноруч створених хуків (функції, що дозволяють ізолювати логіку коду, який керує поведінкою стану функціонального компонента [18]);

– common – містить інтерфейси, перелічення чи оголошені типи, що використовують в декількох частинах програмного коду;

– pages – містить реалізацію та логіку роботи сторінок вебзастосунку. Кожен файл у цій папці також являється окремим компонентом, що, зазвичай, складається із інших компонентів та не передбачає його повторне використання;

– routes – складається із файлів, що відповідають за усі маршрути програми. Вони складається як і з приватних, так і з публічних типів маршрутів.

– scss – тут розташовуються файли стилізації сторінок та компонентів, визначені змінні стилів та міксіни;

– services – служби, які відповідають за відправку запитів на сервер, роботою з локальними сховищами даних та іншими;

– store – модуль, який містить підмодулі управління станами застосунку та виступає як єдине джерело даних, які необхідні у різних частинах системи.

Використовує бібліотеку Redux та впроваджує її підходи до управління станом. Зазвичай store складається з трьох основних частин: actions, reducers та state [19];

– utils – містить допоміжні функції, які часто використовуються в проєкті. Тут мають бути лише звичайні функції та об’єкти, як-от параметри спадного меню, умови регулярного виразу, форматування даних тощо.

– index.tsx – кореневий файл реалізації клієнтської логіки.

Схема рівнів абстракції клієнтської частини наведена на рисунку 2.1.

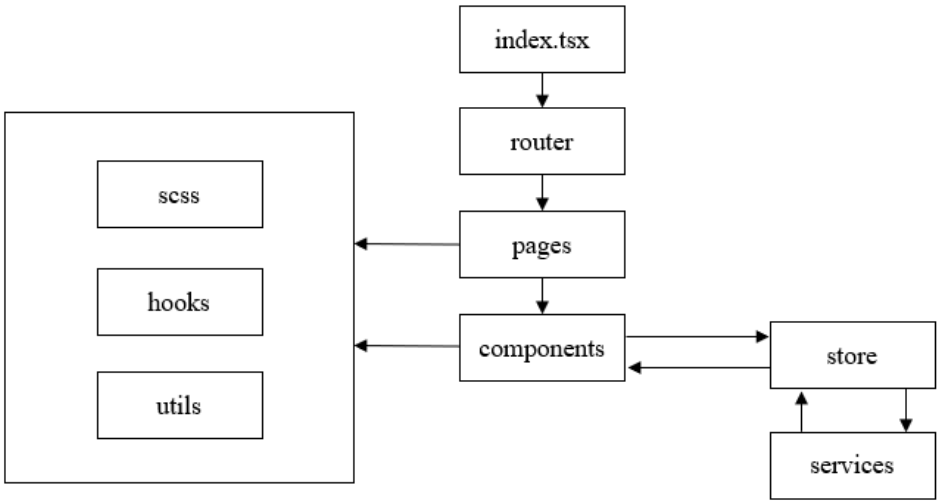

Рисунок 2.1 – Структурна схема клієнтської реалізації

Для розробки серверної частини було визначено наступну структуру системи:

– entities – сутності бази даних;

– migrations – сховище збереження запитів міграцій. Система міграцій схожа на систему створення та контролю версій бази даних. Міграції використовується для зміни та спільного використання схеми бази даних програми [20];

– repositories – складається із власних оголошених репозиторіїв, для управління конкретними сутностями та містить методи для роботи з ними;

– ormconfig.ts – файл з конфігурацією підключення до бази даних, а також визначає місце розташування міграцій;

– common – містить інтерфейси, перелічення чи оголошені типи, що використовують в декількох частинах програмного коду;

– exceptions – являє собою модуль, що містить часто використовувані виключні ситуації;

– middlewares – складається із проміжних обробників, через які проходить об’єкт запиту. Тут можуть бути мідлвари для автентифікації, логування або будь-якої іншої мети;

– controllers – презентаційний шар системи, що являє собою контролери, які отримують запити за певними маршрутами, використовують параметри та інші дані запиту, викликають сервіси для виконання бізнес-логіки та повертають HTTP-відповідь на сторону клієнта [21];

– services – шар бізнес-логіки, який відповідає за обробку та маніпулювання даними перед тим, як вони будуть представлені користувачеві або збережені в базі даних. Представлений у вигляді служб, що також керують робочими процесами, сценаріями використання програми, призначені для повторного використання та не залежить від інтерфейсу користувача та реалізації зберігання даних [22];

– utils – модуль із допоміжними функціями, які можуть стати в нагоді в процесі написання певної логіки системи;

– app.ts – являє собою клас застосунку, в якому створюється сервер, ініціалізуються контролери та мідлвари;

– server.ts – точка входу реалізації серверної логіки, де встановлюється підключення до бази даних та створює екземпляр класу застосунку.

Схема рівнів абстракції серверної частина частини наведена на рисунку 2.2.

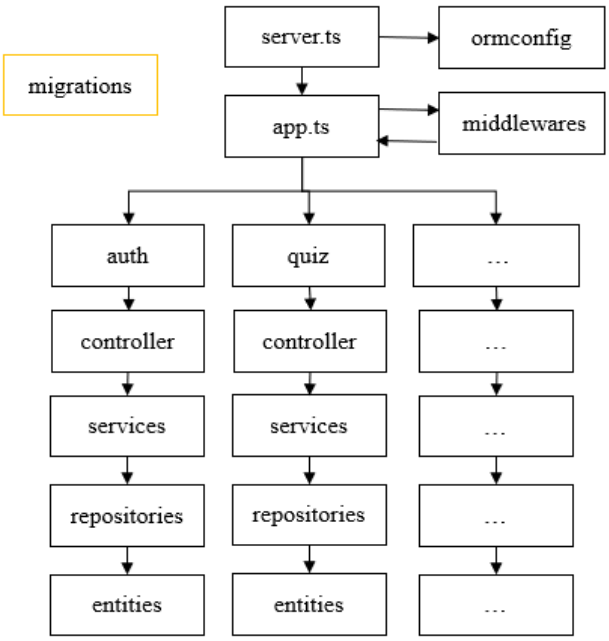

Рисунок 2.2 – Структурна схема серверної реалізації

2.2 Проєктування поведінки вебсистеми для проведення тестувань

Діаграма діяльності – це тип блок-схеми уніфікованої мови моделювання (UML), яка показує потік від однієї діяльності до іншої в системі чи процесі. Вона використовується для опису різних динамічних аспектів системи і називається «діаграмою поведінки», оскільки вона описує, що має статися в змодельованій системі.

Навіть дуже складні системи можна візуалізувати за допомогою діаграм діяльності. У результаті діаграми діяльності часто використовуються в моделюванні бізнес-процесів або для опису етапів діаграми варіантів використання в організаціях. Вони показують окремі кроки в діяльності та порядок, у якому вони представлені. Вони також можуть показувати потік даних між видами діяльності.

Діаграми діяльності показують процес від початку (початковий стан) до кінця (кінцевий стан). Кожна діаграма діяльності включає дію, вузол прийняття рішень, потоки керування, початковий вузол і кінцевий вузол [23]. На рисунку 2.3 продемонстровано діаграму діяльності розроблюваної вебсистеми для проведення тестувань.

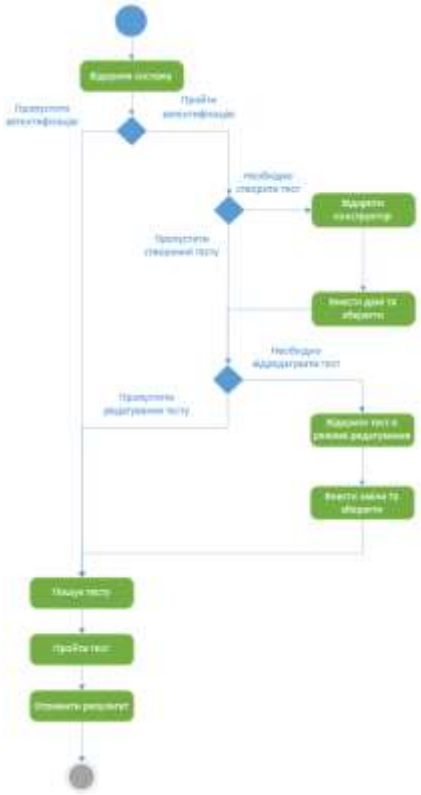

Рисунок 2.3 – Діаграма діяльності вебсистеми

Діаграма станів, також відома як діаграма кінцевого автомата, є ілюстрацією станів, яких може досягати об’єкт, а також переходів між цими станами в уніфікованій мові моделювання . У цьому контексті стан визначає етап в еволюції або поведінці об’єкта, який є певною сутністю в програмі або одиницею коду, що представляє цю сутність [24]. На рисунку 2.4 зображено діаграму станів вебсистеми.

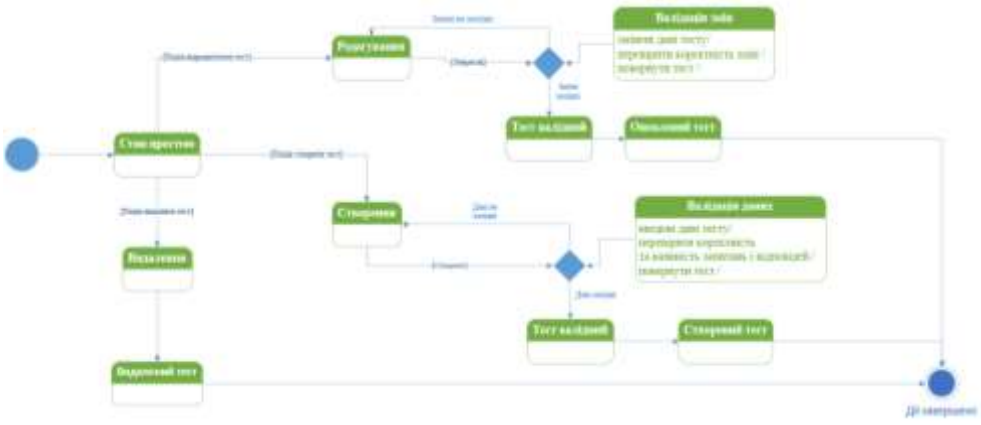

Рисунок 2.4 – Діаграма станів

Діаграма стану за своєю природою нагадує блок-схему, однак блок-схема показує процеси в системі, які змінюють стан об’єкта, а не фактичні зміни стану. Першим кроком до створення діаграми стану є визначення початкового та кінцевого станів системи. Потім усі можливі існуючі стани розміщуються по відношенню до початку та кінця. Вкінці, усі події, які викликають зміни стану, позначені як елементи переходу.

2.3 Розробка моделей даних вебсистеми для проведення тестувань

2.3.1 Перелік інформаційних сутностей та способів їх зберігання

Відповідно до предметної області та вимог до системи сформуємо перелік інформаційних сутностей.

Таблиця 2.1 – Інформаційні сутності

Назва сутності	Призначення
user	Облікові записи користувачів
token	Токени, отримані після процедури авторизації користувачем
quiz	Тести
quiz_question	Питання тесту
quiz_answer	Відповіді до запитань
take	Спроба складання тесту
take_question	Запитання, що відповідають спробі
take_answer	Відповіді на запитання спроби

Тепер, ми можемо провести глибокий аналіз сутностей. Результати аналізу подані у додатку Б.

2.3.2 Проєктування концептуальної моделі даних

Концептуальна модель містить 8 таблиць бази даних необхідних для системи. Між усіма таблицями встановлено зв’язок «один до багатьох». Концептуальну модель зображено на рисунку 2.5.

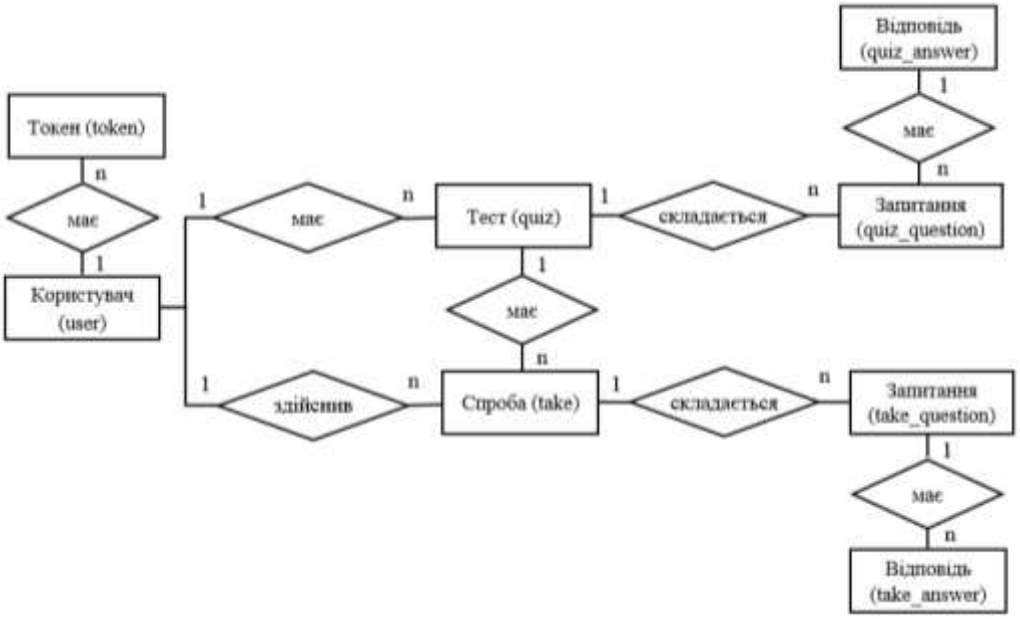

Рисунок 2.5 – Концептуальна модель бази даних

Аналіз та опис зв’язків між сутностями продемонстровано у таблиці 2.2.

Таблиця 2.2 – Аналіз та опис зв’язків між сутностями

Сутності	Назва та тип зв’язку		Зміст зв’язку
user – token	має	1:n	Один користувач може мати багато токенів різних сесій; один токен може належати тільки одному користувачу
user – quiz	має	1:n	Одному користувачу може належати багато тестів; один тест належить одному користувачу

Продовження таблиці 2.2

Сутності	Назва та тип зв’язку		Зміст зв’язку
quiz – quiz_question	складається	1:n	Один тест складається із багатьох запитань; одне запитання належить одному тесту
quiz_question – quiz_answer	складається	1:n	Одне запитання складається із декількох відповідей; одна відповідь належить одному запитанню
user – take	здійснив	1:n	Один користувач може мати багато спроб одного тесту; одна спроба належить одному користувачу
take – take_question	складається	1:n	Одна спроба складається із багатьох запитань; одне запитання належить одній спробі
take_question – take_answer	складається	1:n	Одне запитання складається із декількох відповідей; одна відповідь належить одному запитанню

2.3.3 Проєктування логічної та фізичної моделей даних

Діаграма потоків даних (DFD) – це традиційне візуальне представлення потоків інформації в системі [25]. Дана діаграма показує те, як дані надходять і залишають систему, що змінює інформацію та де зберігаються дані. Метою DFD є показати масштаб і межі системи в цілому. Діаграму потоків даних розроблюваного застосунку подано на рисунку 2.6.

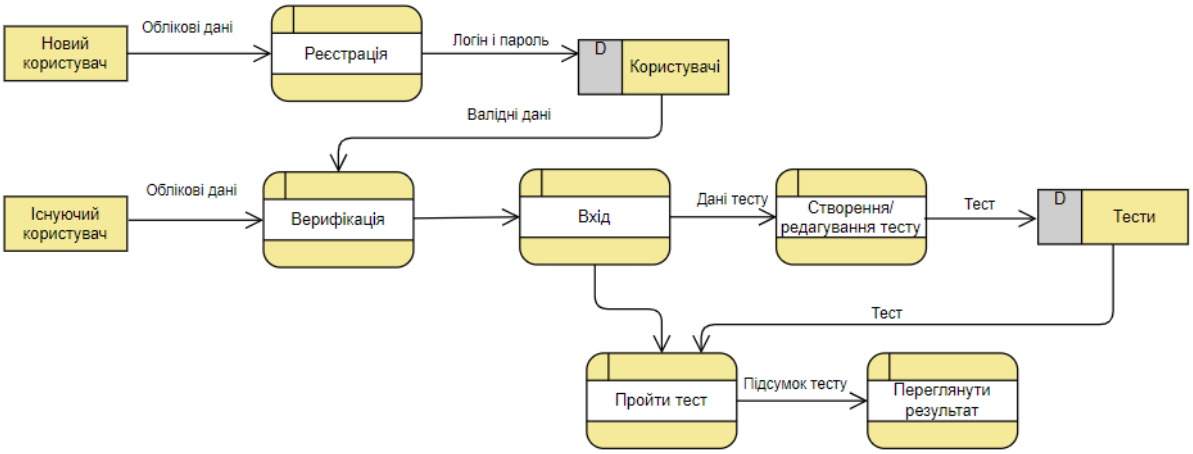

Рисунок 2.6 – Діаграма потоків даних

Фізична модель даних визначає компоненти, які є необхідними для створення нової бази даних чи можуть бути частиною уже створеної. Дана модель складається з назв і значень стовпців таблиці, з первинних та зовнішніх ключів, зв’язків між таблицями.

Часто, фізичні моделі створюються під час розробки нової системи, щоб команда розробників могла зрозуміти, як структурувати базу даних. Діаграми такого типу, використовують певний набір символів, наприклад фігури та стрілки, для зображення системи та бази даних.

Сутність – це річ, яка може бути формою зберігання даних. Це може бути фізичний об'єкт, концепція або подія. Зазвичай вони представлені у вигляді прямокутників із назвою об’єкта всередині прямокутника. Зв’язок визначає, як дві сутності пов’язані одна з одною. Є кілька типів зв’язків: один до одного, один до багатьох, багато до багатьох. Атрибут – це властивість сутності або те, що можна використовувати для її опису. Вони часто представлені у вигляді записів всередині сутності. [26]

Фізична модель даних розроблюваної вебсистеми для проведення тестувань зображена на рисунку 2.7.

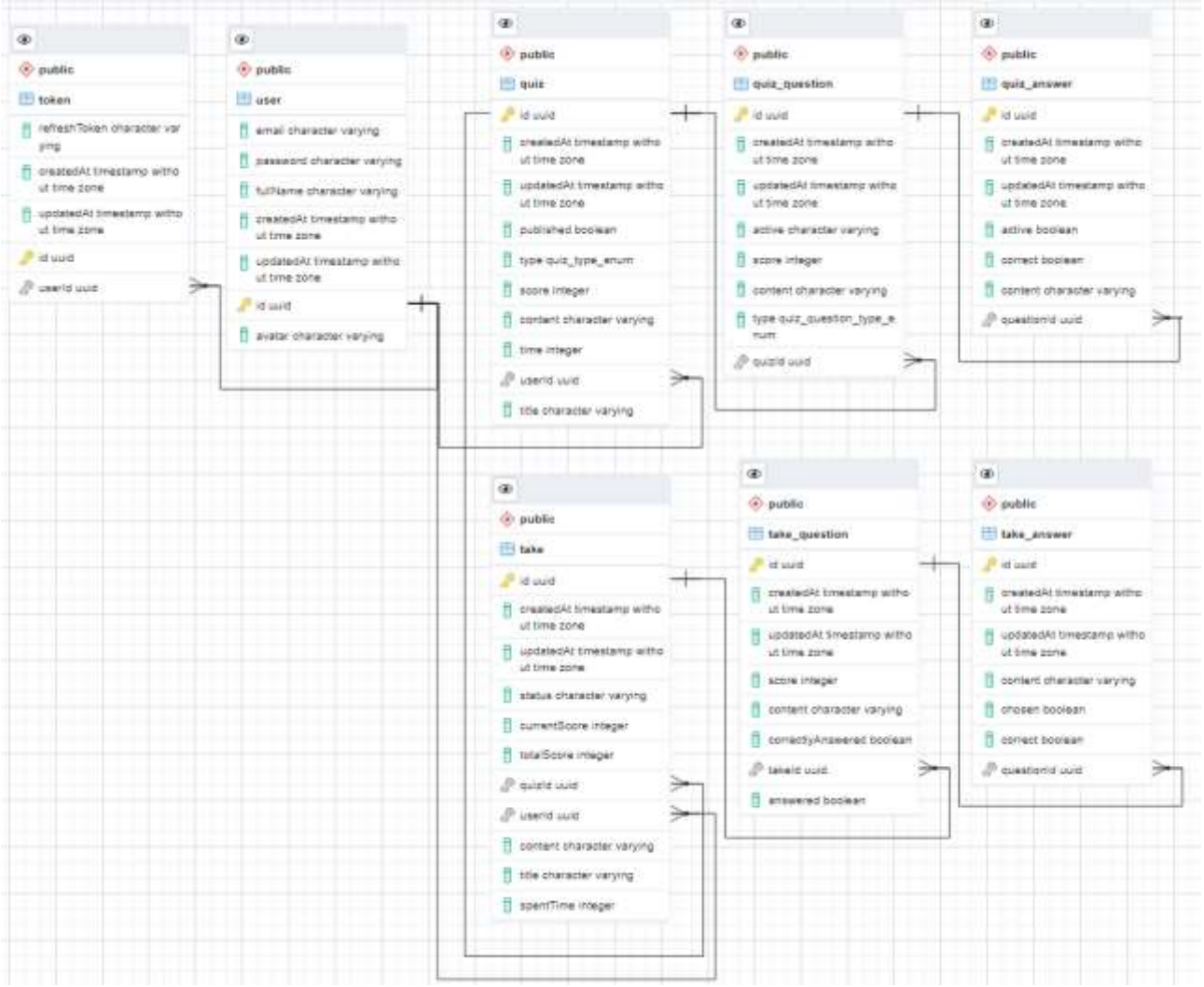

Рисунок 2.7 – Фізична модель даних

2.4 Проєктування інтерфейсу вебсистеми для проведення тестувань

2.4.1 Обґрунтування вибору колірної схеми

Чистий дизайн зазвичай характеризується однотонними кольорами та великою кількістю простору між елементами. Колірною схемою розроблюваного вебзастосунку є схема у вигляді градацій сірого.

Легкі відтінки сірого значно покращують сприйняття інтерфейсу. Сірий забезпечує більший контраст для кнопок та й взагалі дозволяє багато рівнів ієрархії. Це тому, що існує багато відтінків сірого. Оскільки сірий колір використовується для підтримки вмісту в макеті, це дає змогу відображати сам вміст на білому фоні, безпосередньо вказуючи на чистий дизайн.

Сірий привертає увагу до змісту. Найсвітліша біла область на сторінці надає змісту необхідний акцент. Даний колір дозволяє використовувати більше пробілів і пробілів поверх пробілів. Він майже виглядає білим, тож можна додати трохи сірого, щоб розділити білі ділянки, не ускладнюючи дизайн і зробивши це максимально непомітно. [27]

Таким чином, сірий колір дозволяє нам проєктувати вміст і користувацькі інтерфейси так, щоб другорядні елементи та структурні напрямні були подалі від візуальної чи свідомої уваги, а це саме те, що завжди має на меті хороший дизайн.

2.4.2 Обґрунтування структури шаблонів інтерфейсу

Інтерфейс користувача може складатись із багатьох елементів. На рисунку 2.8 продемонстровано базовий макет сторінок розроблюваного вебсистеми для проведення тестувань.

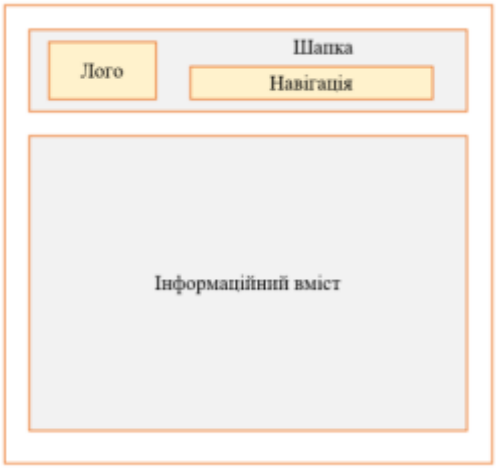

Рисунок 2.8 – Базовий макет сторінок

Також, впродовж реалізації, є хорошим тоном використовувати наступні семантичні теги для структурування нашої сторінки:

– header – цей зазвичай знаходиться у верхній частині документа, розділу або статті та зазвичай містить головний заголовок і деякі інструменти навігації та пошуку;

– main – елемент main використовується для зберігання всього значущого вмісту сторінки, тобто всі теги розділів, статей, aside знаходяться всередині нього. У документі HTML має бути лише один основний елемент;

– section – розділ використовується для групування пов’язаного вмісту, наприклад різних статей, інформаційних карток тощо;

– aside – це елемент, який використовується для розміщення бічних панелей на нашій сторінці;

– footer – нижній колонтитул, використовується для розміщення такого вмісту, як інформація про авторські права, посилання на сайти, форми інформаційних бюлетенів тощо.

2.5 Розробка основних модулів вебсистеми для проведення тестувань

2.5.1 Реалізація клієнтської частини системи

Створення інтерфейсу користувача та браузерної логіки з використанням бібліотеки React супроводжується створенням компонентів. Вони мають власну структуру, методи та власні API. Компоненти можна використовувати багаторазово та вставляти в різні частини застосунку. Незалежний характер компонентів дозволяє створювати інтерфейс користувача з багатьма різними змінними частинами. Оскільки компоненти є незалежними одиницями, один компонент може змінюватись, не впливаючи на інші чи інтерфейс користувача в цілому.

Розглянемо реалізацію компонентів на прикладі сторінки авторизації «Login» та сторінки проходження тесту «UserQuizPassing». Основними елементами сторінки «Login» є: розмітка сторінки, хуки – для управління життєвим циклом компонента, обробники подій форми.

Розмітка сторінки відображає форму для введення логіну та паролю. Кожне поле містить назву, елемент вводу та місце для відображення помилок валідації, що подано у лістингу 2.1.

Лістинг 2.1 – Розмітка поля вводу форми входу в систему

```
<label className="auth-form__label">
 Password<span className="required">*</span>
</label>
          <FormInput
            className={passwordInput.isDirty &&
!passwordInput.isValid ? 'error-input' : ''}
            name="password"
            type="password"
            value={passwordInput.value}
            placeholder="Password"
            icon={<LockIcon />}
            onChange={passwordInput.onChange}
            onBlur={passwordInput.onBlur}
          />

  {passwordInput.isDirty && passwordInput.isEmpty.value && ( <span
className="auth-
form__error">{passwordInput.isEmpty.errorMessage}</span> )}
```

Програмний код сторінки входу винесено у додаток В.

Для управління значеннями цих полів та їх параметрами валідації було створено власні хуки – «useInput» та «useValidation». Хук «useInput» приймає початкове значення вводу та об’єкт «validations» із параметрами валідації. Код з релізацією хука продемонстровано в лістингу 2.2.

Лістинг 2.2 – Реалізація хука «useInput»

```
const useInput = (initialValue: string, validations:
IValidations): IUseInput => {
  const [value, setValue] = useState(initialValue);
  const [isDirty, setIsDirty] = useState(false);
  const valid = useValidation(value, validations);

  const onChange = (e: ChangeEvent<HTMLInputElement>) =>
setValue(e.target.value);
  const onBlur = () => setIsDirty(true);

  return {
    value,
    isDirty,
    onChange,
    onBlur,
    ...valid,
  };
};

export default useInput;
```

Хук «useInput» використовує «useValidation». У логіці хука useVlidation передбачено перевірку коректності значення поля вводу за чотирма опціональними критеріями: «minLength» – мінімальна довжина тексту, «maxLength» – максимальна довжина тексту, «isEmpty» – перевірка на відсутність значення, «isEmail» – перевірка на відповідність електронної адреси регулярному виразу. Центральна логіка хука покладена на конструкцію switch-case. У лістингу 2.3 відображено частину програмного коду даного функціоналу.

Лістинг 2.3 – Логіка валідації хука «useValidation»

```
for (const validation in validations) {
  switch (validation) {
    case 'minLength':
      validations?.minLength && value.length <
validations.minLength
        ? setMinLengthError({
            value: true,
            errorMessage: `Should have ${validations.minLength}
characters at least`,
          })
        : setMinLengthError({ value: false });
      break;
    case 'isEmail':
      const regularExpression =
/^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(".+"))@((\[[
0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-
9]+\.)+[a-zA-Z]{2,}))$/;
      regularExpression.test(String(value).toLowerCase())
        ? setEmailError({ value: false })
        : setEmailError({ value: true, errorMessage: 'Email format
is not valid' });
      break;
   }
}
```

Реалізацію хука у повному обсязі подано у додатку Г.

Відправка даних форми на сервер відбувається наступним чином: користувач, ввівши облікові дані, натискає кнопку входу. Обробником події натискання на дану кнопку є функція, яка надсилає запит на сервер через Redux. Він у свою чергу діє за такими основними принципами:

– store (сховище даних) підтримує стан додатку;

– reducers є чистими функціями та виконують трансформацію стану на основі об’єкта action, який включає властивість «type» та «payload»;

– actions являються простими описовими об’єктами, що повідомляють reducer про тип події та місять не обов’язкове корисне навантаження (payload).

Потік даних додатку з використанням Redux продемонстровано на рисунку 2.9.

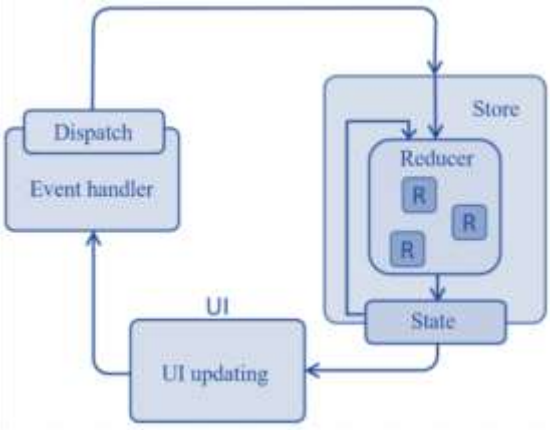

Рисунок 2.9 – Потік даних у Redux

За створення об’єкта «action», який необхідний редюсеру, відповідає окрема функція – action creator. Для події «LOGIN» вона має вигляд як у лістингу 2.10.

Лістинг 2.10 – Функція-створювач об’єкта «action» події «LOGIN»

```
export const login = createAsyncThunk<IAuthResponse, ILoginUser>(
  AuthAction.LOGIN,
  async (userData: ILoginUser, { rejectWithValue }) => {
    try {
      const response = await authService.login(userData);
      storageService.setItem('token', response.accessToken);
      return response;
    } catch (e) {
      return rejectWithValue(e);
    }
  },
);
```

Як можна побачити на рисунку, всередині даної функції викликається метод сервісу із назвою «AuthService». Це один із сервісів, що здійснюють запити на сервер. «AuthService» сервіс відповідає за реєстрацію та авторизацію користувача. У лістингу 2.11 показано метод «login» даного сервісу.

Лістинг 2.11 – Метод «login» сервісу «AuthService»

```
public login = async (payload: ILoginUser): Promise<IAuthResponse>
=> {
  return this.http.load<IAuthResponse>(`${this.path}/auth/login`,
{
    method: HttpMethod.POST,
    payload,
  });
};
```

Крім цього методу, у сервісі також визначено інші:

– signup – використовується при реєстрації;

– logout – викликається, коли користувач виходить із акаунту;

– refresh – застосовується для оновлення токену.

Програмний код сервісу «AuthService» представлено у додатку Ґ.

Після отримання відповіді редюсер, що відповідає за автентифікацію («authReducer»), змінює стан об’єкта, що містить дані автентифікації та встановлює, що користувач є автентифікованим, що подано у лістингу 2.12.

Лістинг 2.12 – Зміна стану у «authReducer»

```
builder.addMatcher(
isAnyOf(authActions.login.fulfilled,authActions.signup.fulfilled),
  (state: IAuthState, payload) => {
    state.user = payload.payload.user;
    state.isAuth = true;
  },
);
```

Для того, щоб отримати дані із глобального стану існує метод useSelector. Вказуємо в ньому колбек із об’єктом, значення якого ми хочемо отримати. Код отримання даних стану автентифікації виглядає наступним чином:

```
const { isAuth, isLoading }= useAppSelector((state) =>state.auth);
```

Маючи інформацію про те, чи поточний користувач увійшов в систему ми виконуємо відповідні дії: якщо спроба зареєструватись була неуспішною, то на формі реєстрації буде показано інформацію з помилкою входу, а у випадку успішної реєстрації – користувача буде перенаправлено на початкову сторінку.

Тепер, розглянемо компонент, що являє собою сторінку проходження тестування. Першим чином, коли компонент вмонтовано нам необхідно отримати дані тесту. Це ми можемо виконати за допомогою хука «useEffect» із пустим масивом залежностей:

```
  useEffect(() => {
    dispatch(takeActions.start(params.id));
  }, []);
```

Надіславши подію «START» в store, ми виконуємо запит на сервер, що фіксує початок проходження тестування та повертає об’єкт із даними тесту (див. лістинг 2.13).

Лістинг 2.13 – Відправка запиту на початок тестування

```
public start = async (quizId: string): Promise<ITakeStartResponse>
=> {
  return await this.http.load<ITakeStartResponse>
(`${this.path}/start/${quizId}`, {method: HttpMethod.POST });
};
```

Дані, які повернув сервер формують корисне навантаження об’єкта «action». Редюсер, використовуючи даний об’єкт, модифікує значення області глобального об’єкта стану, яка необхідна при проходженні тесту. Представленням такої частини даних є інтерфейс «ITakeState» (див. лістинг 2.14).

Лістинг 2.14 – Інтерфейс об’єкта глобального стану, необхідний при проходженні тестування.

```
export interface ITakeState {
  isFinished: boolean;
  take: ITakeStartResponse | null;
  isLoadingTake: boolean;
  failedToLoadTake: boolean;
  results: ITakeFinishResponse | null;
  isLoadingResults: boolean;
  failedToLoadResults: boolean;
}
```

Маючи дані тесту у сховищі, ми можемо використати їх у своїх компонентах. У нашому випадку, виймаємо ці дані для компонента «UserQuizPassing» уже знайомим способом – за допомогою «useAppSelect»:

```
const { isLoadingTake, take, isFinished, isLoadingResults, results
} = useAppSelector((state) => state.take,);
```

Вийнявши дані тесту, використовуємо їх у компоненті «PassingCore», що відповідає за ключову функціональність проходження тестування. Рендеримо цей компонент на сторінці за умови, якщо тест не завершено, як продемонстровано у лістингу 2.15.

Лістинг 2.15 – Рендеринг компонента складання тесту

```
if (!isFinished && take) {
  return (
    <Helmet title={`${take.title} started`}>
      <Container className="active-quiz">
        <PassingCore
          quiz={mapTakeForPassing(take)}
          onClose={handleCloseQuiz}
          onFinish={handleFinishQuiz}
          onAnswer={handleAnswerQuestion}
        />
      </Container>
    </Helmet>
  );
}
```

У випадку завершення проходження тесту, ми відображаємо результат із підсумками. Для цього використовується компонент «Result» (див. лістинг 2.16).

Лістинг 2.16 – Рендеринг компонента відображення результатів

```
  if (isFinished && results) {
    const { spentTime, score, totalScore, questionsNumber,
correctNumber } = results;

    return (
      <Helmet title="Results">
        <Container className="active-quiz">
          <Result
            totalQuestions={questionsNumber}
            correctAnswers={correctNumber}
```

```
            score={score}
            totalScore={totalScore}
            time={spentTime}
            onClose={handleCloseQuiz}
          />
        </Container>
      </Helmet>
    );
  }
```

За похожим принципом та з використанням подібних підходів було створено решту логіки клієнтської частини даної системи. В тому числі, до основних компонентів, які також було реалізовано, належать наступні:

– «QuizInfo» – сторінка, що відповідає за відображення даних про тест. Специфікою реалізації даної сторінки є отримання ідентифікатора тесту із url-адреси, щоб в подальшому мати змогу відправити запит на сервер на отримання даних вибраного тесту. Для того, щоб отримати значення Id було використано хук «useParams», що повертає об'єкт із ключами та значення параметрів переданих через url;

– «SingleChoice» – являється окремо виділеним компонентом, що використовується сторінками «SingleChoiceCreator» та «SingleChoiceEditor». У ньому розміщена логіка конструювання тесту. Функціональність цього компоненту включає додавання запитання, встановлення балів за правильно надану відповідь на запитання, перемикання видимості цього запитання після публікації, видалення створеного елемента запитання, прикріплення та додавання відповідей, вказування за допомогою перемикача коректності даної відповіді, задання видимості відповіді після публікації тесту. Також, в процесі створення чи редагування тесту відбувається валідація введень;

– «SingleChoiceCreator» – являє собою сторінку для створення тесту. Вона використовує логіку компонента «SingleChoice» та додатково реалізовує відправку запиту на сервер із додавання тесту. Не допускається відправки даних на сервер у випадку, коли компонент «SingleChoice» повертає повідомлення про не валідність тесту у поточному його вигляді;

– «SingleChoiceCreator» – сторінка для редагування тесту. Також використовує логіку компонента «SingleChoice» та відправляє запит на сервер на

модифікацію тесту. Також, передбачено блокування функціональності відправки даних на сервер при наявності помилки конструювання тесту.

– «UserQuizPassing» та «VisitorQuizPassing» – сторінки, що відображаються в процесі проходження тесту. В кожній з них імплементована логіка здачі тесту, а саме: управління вибором відповіді, підтвердженням відповіді на запитання, поверненням та зміною відповіді, обліком кількості пройдених запитань, завершенням тесту та відображенням результатів, закриттям форми проходження тестування. Відмінність між ними полягає у тому, хто проходить тест та у збереженні даних. Якщо це не зареєстрований користувач, то дані проходження не зберігаються в базі даних, а якщо зареєстрований – зберігаються;

– «Profile» – сторінка з даними профілю та історією проходження тестів. Складається із трьох основних компонентів «ProfileHeader» - містить фото профілю та загальні його дані, «ProfileCharts» - діаграми успішності тестувань та активності користувача (кількість опублікованих тестів), «QuizSummary» - відображає історію проходження тестувань з усіма спробами та їх результатами.

Стилізація сторінок розміщена та згрупована у окремій директорії. Організацію файлів стилів показано на рисунку 2.10.

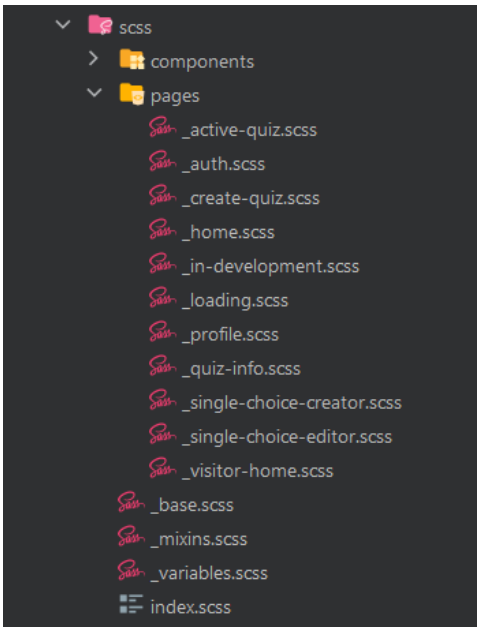

Рисунок 2.10 – Файли стилів застосунку

Як помітно із розширень файлів, в якості інструменту стилізації було використано препроцесор SCSS. Він дозволяє легше працювати із стилями CSS, додає більшої функціональності та структурованості коду.

2.5.2 Реалізація серверної частини системи

Для створення бекенду вебзастосунку було використано фреймворк для Node.js, а саме – Express.js. У найпростішому вигляді створення сервера з використанням даної технології потребує декілька рядків коду, як показано у лістингу 2.17.

Лістинг 2.17 – Реалізація найпростішого серверу на Express.js

```
import * as express from 'express';
const app = express();
 app.get('/', (request, response) => {
  response.send('Hello world!');
});
app.listen(5000);
```

Спочатку ми імпортуємо модуль «express», потім викликаємо функцію «express()», яка створює застосунок з яким ми будемо взаємодіяти. Метод get приєднує функцію зворотного виклику до вказаного шляху для GET-запитів. Коли хтось робить запит GET на вказаний шлях, виконується функція зворотного виклику. Функція «response.send()» надсилає відповідь клієнту. Функція прослуховування «listen()» змушує програму прослуховувати з’єднання на вказаному порту. У наведеному прикладі, сервер запущений локально та робить нашу програму доступною за адресою http://localhost:5000.

Серверну частину було реалізовано за архітектурним шаблоном MVC. Одними з ключових його елементів є контролери. Вони містять логіку програми, що займається обробкою запитів клієнтів та відправкою відповідей. Їх ініціалізація відбувається в класі «App». Даний клас має наступні методи:

– constructor(controllers: IController[]) – конструктор класу. Тут створюється Express-застосунок, викликаються методи для ініціалізації мідлварів та контролерів;

– listen() – метод, який використовується для запуску сервера та прослуховування певного порту;

– initializeControllers(controllers: IController[]) – здійснює ініціалізацію контролерів, які приймає в якості аргумента;

– initializeMiddlewares – метод, що забезпечує ініціалізацію наступних мідлварів: express.json(), що є вбудованою функцією та парсить вхідні запити у формат JSON; cookieParser() – парсить заголовок «Cookie» і встановлює об’єкт req.cookies із ключами імен файлів «Cookie»; cors() – використовується для ввімкнення CORS з різними параметрами; authMiddleware – перевіряє на наявність та валідує токен, що приходить від клієнта, а також прикріпляє до об’єкт запиту дані користувача та його роль;

– initializeErrorHandling – визначає проміжну функцію для обробки помилок; являється таким ж мідлваром, як і будь-який інший, за винятком того, що ми використовуємо чотири аргументи замість трьох, де помилка є додатковим першим аргументом.

Код класу «App» подано у додатку Д.

Кожен контролер імплементує інтерфейс IController, що виглядає наступним чином:

```
interface IController {
  path: string;
  router: Router;
}
```

Тобто, для кожного контролера ми визначаємо маршрут, за яким ми ідентифікуємо його та на який будуть надсилатись запити із сторони клієнта.

Точкою входу коду на сервері є файл «server.ts», в якому реалізовано функцію, де відбувається підключення до БД, використовуючи метод «createConnection()» із TypeORM – як подано у лістингу 2.18:

Лістинг 2.18 – Підключення до бази даних

```
try {
   await createConnection(config);
 } catch (error) {
   console.log('Error while connecting to the database', error);
   return error;
}
```

В createConnection передаємо об’єкт з параметрами підключення: хост, порт, ім’я користувача, пароль, назву бази даних і тд., що показано у лістингу 2.19.

Лістинг 2.19 – Об’єкт з конфігурацією підключення до БД

```
const config: ConnectionOptions = {
  type: 'postgres',
  host: process.env.POSTGRES_HOST,
  port: Number(process.env.POSTGRES_PORT),
  username: process.env.POSTGRES_USER,
  password: process.env.POSTGRES_PASSWORD,
  database: process.env.POSTGRES_DB,
  entities: [__dirname + '/../**/*.entity{.ts,.js}'],
  cli: {
    migrationsDir: 'src/migrations',
  },
  migrations: ['src/migrations/**/*.ts'],
  synchronize: false,
};
```

Також, в цьому класі створюється екземпляр класу «App», в який ми передаємо контролери, що показано у лістингу 2.20.

Лістинг 2.20 – Використання класу «App»

```
const app = new App([
    new AuthController(
      new AuthService(new TokenService(), new
AwsS3Service(ConfigService.getInstance())),
    ),
    new UserController(new UserService()),
    new QuizController(new QuizService()),
    new QuizQuestionController(new QuizQuestionService()),
    new QuizAnswerController(new QuizAnswerService()),
    new TakeController(
      new TakeService(new QuizService(), new
TakeQuestionService(), new TakeAnswerService()),
    ),
```

```
  ]);
```

Нижче відбувається запуск серверу через виклик методу «listen»: app.listen();

Для управління доступом до маршрутів контролерів, використовується мідлвар валідації доступу «validatePermission», що приймає масив ролей системи та, у випадку невідповідності цим ролям, викидає помилку з повідомленням «You don`t have permissions for this action». Одним із прикладів його використання, можна навести запит на видалення тесту, що продемонстровано у лістингу 2.21.

Лістинг 2.21 – Застосування валідації дозволів по ролях

```
this.router.delete(
  `${this.path}/:id`,
  validatePermission([RoleType.USER]),
  this.deleteQuiz.bind(this),
);
```

Для обробки запиту, що надійшов на контролер, використовуються сервіси. Контролер викликає методи сервісів, які опрацьовують вхідні параметри та звертаються до бази даних, через репозиторії, щоб здійснити відповідні маніпуляції над даними. В результаті, методи сервісу можуть повертати значення, або, при наявності помилок – викидати виключення. За обробку цих виключень відповідає конструкція «try/catch», застосована у контролері (див. лістинг 2.22).

Лістинг 2.22 – Вигляд методу контролера

```
private async deleteQuiz(req: IAuthRequest, res: Response, next:
NextFunction) {
  try {
    const quizId: string = req.params.id as unknown as string;
    await this.quizService.delete(quizId);
    res.status(HttpCode.NO_CONTENT).send();
  } catch (e) {
    next(e);
  }
}
```

Код контролера подано у додатку Е.

При наявності виключення, за допомогою виклику функції next(), ми застосовуємо мідлвар для обробки помилок, який надсилає відповідь користувачеві із відповідним кодом помилки та повідомленням. Код цього мідлвару подано у лістингу 2.23.

Лістинг 2.23 – Мідлвар обробки помилок

```
function errorHandlerMiddleware(
  error: HttpException, req: Request, res: Response, next:
NextFunction,) {
  const status = error.status || HttpCode.INTERNAL_SERVER_ERROR;
  const message = error.message || 'Internal Server Error';
  res.status(status).send({ status, message });
}
export default errorHandlerMiddleware;
```

Наступний метод видаляє тест користувача за вказаними його ідентифікатором. Він розміщений у сервісі «QuizService». Код даного методу подано у лістингу 2.24.

Лістинг 2.24 – Метод сервісу QuizService

```
  public async delete(id: Quiz['id']): Promise<void> {
    const quiz = await this.quizRepository.findOne({ id });

    if (!quiz) {
      throw new HttpException(HttpCode.NOT_FOUND, 'Quiz not
found');
    }

    await this.quizRepository.delete({ id });
  }
```

Якщо сервіс щось повертає, то контролер здійснює відправку цих даних в якості відповіді клієнту.

Усього у бекенд застосунку містить 6 контролерів, які використовують 10 сервісів:

– AuthController – приймає запити на реєстрацію, авторизацію, вихід із системи та оновлення токена. Використовує AuthService, TokenService, AwsS3Service та ConfigService сервіси;

– UserController – приймає запити, що відповідають за взаємодію із даними користувача. Використовує UserService сервіс;

– QuizController, QuizQuestionController, QuizAnswerController – обслуговують запити на отримання даних тестів, створення, видалення та оновлення тесту. Для виконання даних операцій контролери використовують QuizService, QuizQuestionService та QuizAnswerService сервіси.

– TakeController – обслуговує запити на старт тестування, його завершення, надання відповіді на запитання та отримання результатів. Використовує TakeService, TakeQuestionService та TakeAnswerService сервіси.

2.6 Тестування вебсистеми для проведення тестувань

Оцінка коректності роботи розробленого вебсистеми для для проведення тестувань та її компонентів є важливим елементом процесу розробки. Необхідно протестувати систему, виявити будь-які прогалини, помилки або невідповідність поставленим вимогам.

Виконаємо тестування форми реєстрації. Спочатку, перевіримо коректність роботи валідаторів форми. Допоки форма не пройде валідацію, кнопка підтвердження реєстрації буде заблокована. При наявності некоректних даних, під полем вводу відображається відповідне повідомлення (див. рис. 2.11).

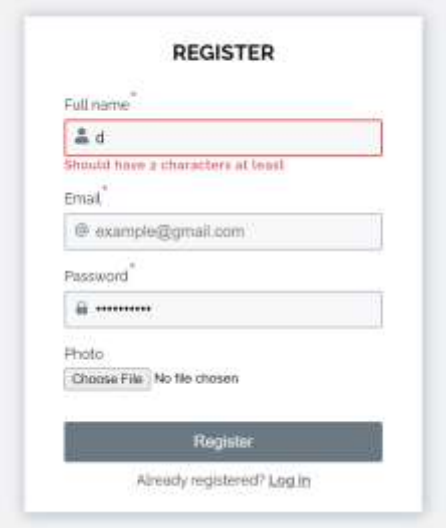

Рисунок 2.11 –Тестування форми реєстрації

Поле вибору фото не є обов’язковим. Проте, протестуємо дану функціональність. Як можна побачити на рисунку 2.12, після вибору зображення профілю із провідника, користувач має можливість змінити його розмір.

Рисунок 2.12 – Обрізання фото профілю

Після обрізання зображення та ввівши всі поля коректно, нам стає доступною можливість реєстрації – кнопка «Register» стає клікабельною (див. рис. 2.13).

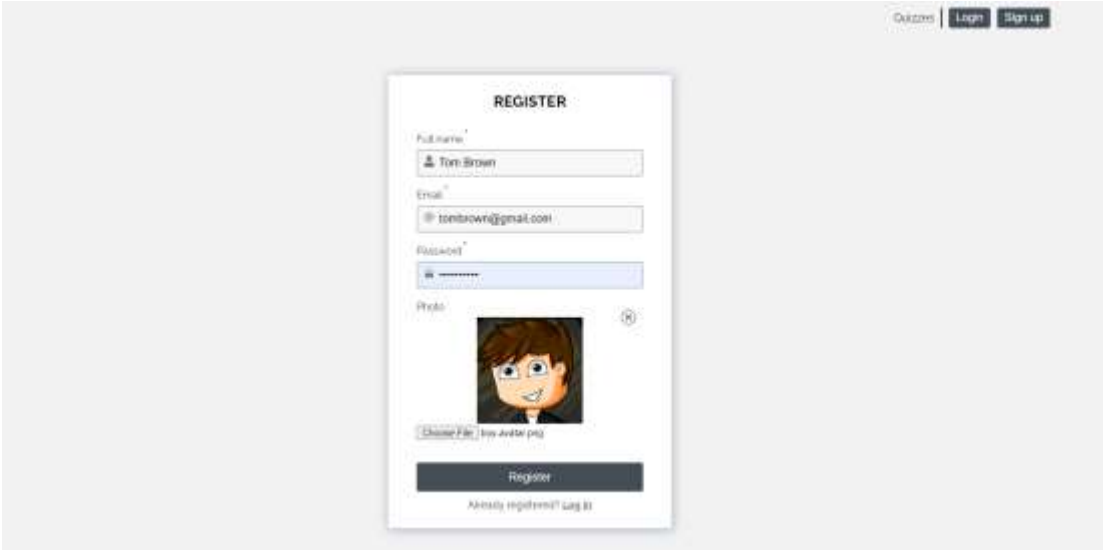

Рисунок 2.13 – Заповнення форми реєстрації

Після реєстрації нас перенаправляє на головну сторінку (див. рис. 2.14).

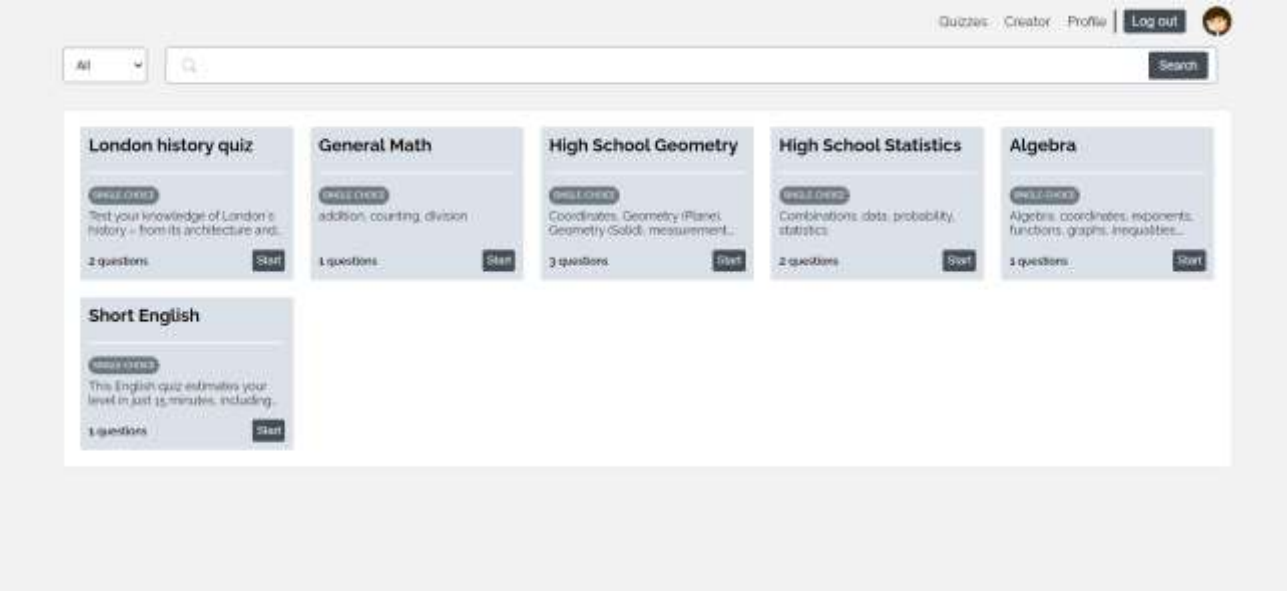

Рисунок 2.14 – Головна сторінка застосунку

Протестуємо функціонал даної сторінки. Спершу, здійснимо фільтрацію тестів, використовуючи випадаючий список, що розміщено зліва від поля пошуку. Виберемо із списку значення «Created», що відповідає за відображення тільки тестів, що були створені поточним користувачем. Задавши даний фільтр ми бачимо, що жоден тест не відобразився, що підтверджує коректність роботи даної логіки, адже з новоствореного акаунту ще не відбувалось створення тестів.

Також, ми можемо протестувати функціонал сторінки «Creator». При переході із навігаційної панелі по посиланню на дану сторінку, нас перенаправляє на конструктор тесту (див. рис. 2.15).

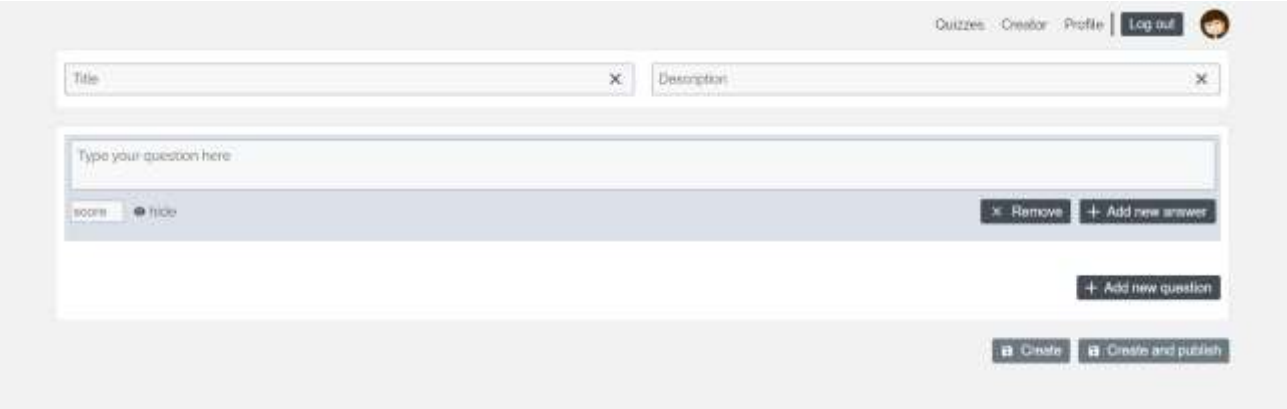

Рисунок 2.15 – Тестування конструктора тестів

Відразу помічаємо, що кнопки, що знаходяться внизу є неактивними. Для розуміння користувачів, чому вони не можуть створити тест, при наведенні на кнопки «Create» та «Create and publish» з’являється повідомлення із текстом помилки (див. рис. 2.16).

Рисунок 2.16 – Повідомлення про некоректність поточного стану тесту в конструкторі

В процесі вирішення однієї проблеми, вміст даного повідомлення змінюється. Наприклад, виконаємо вимогу першого повідомлення та додамо заголовок до тесту. Як результат, текст повідомлення змінився і тепер нам потрібно додати опис до нашого тесту (див. рис. 2.17).

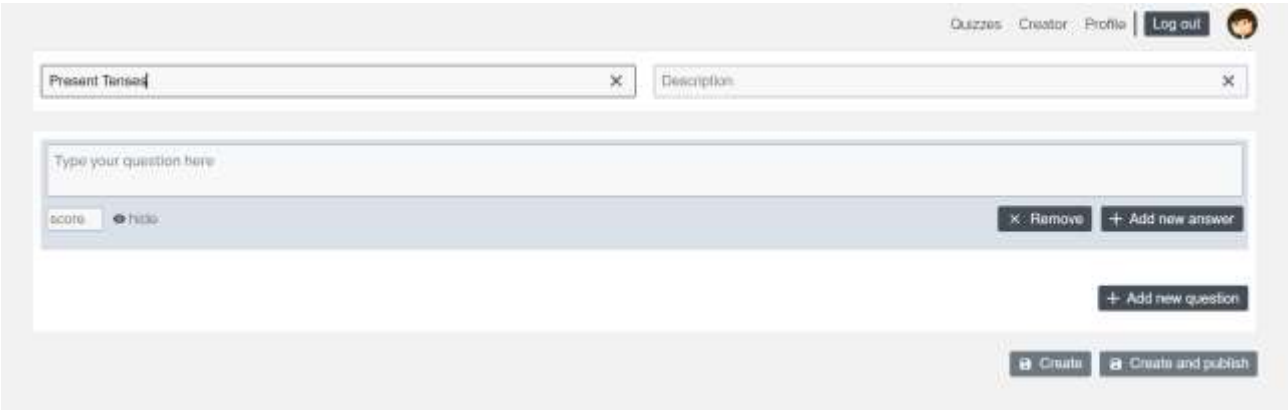

Рисунок 2.17– Зміна повідомлення про некоректність поточного стану тесту в конструкторі

Додавши опис, питання та відповіді, ми маємо можливість зберегти даний тест, натиснувши на кнопку «Create and publish» (див. рис. 2.18).

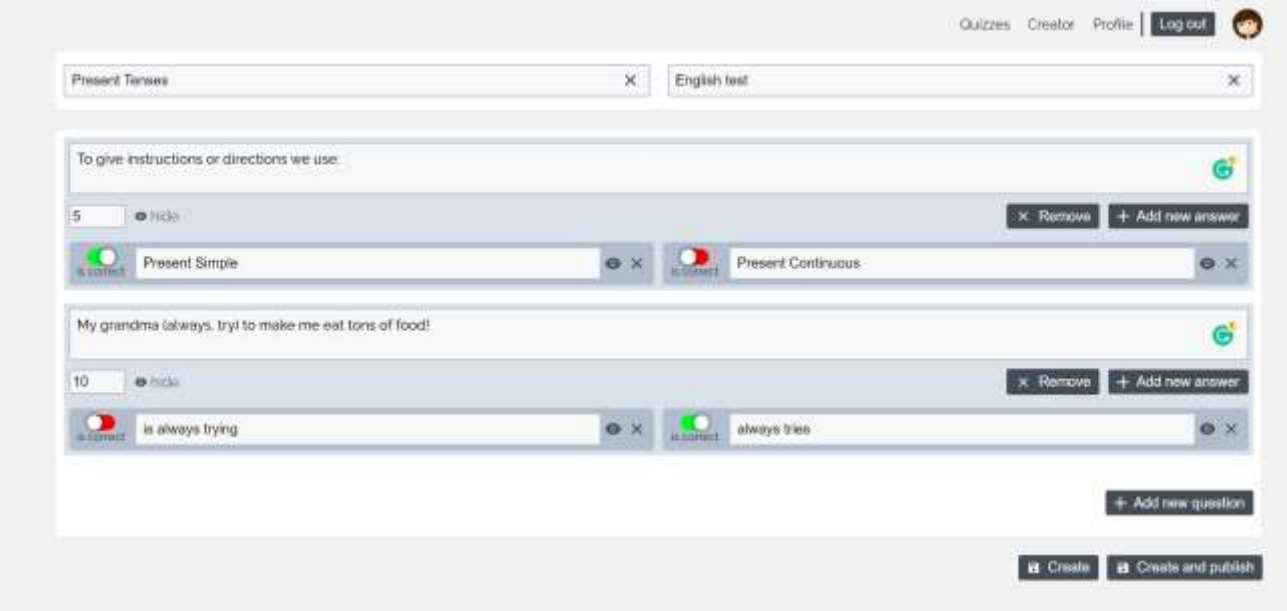

Рисунок 2.18 – Завершення створення тесту

В результаті нас перекидає на сторінку із інформацією про тест, де усі дані відповідають щойно створеному тесту (див. рис. 2.19).

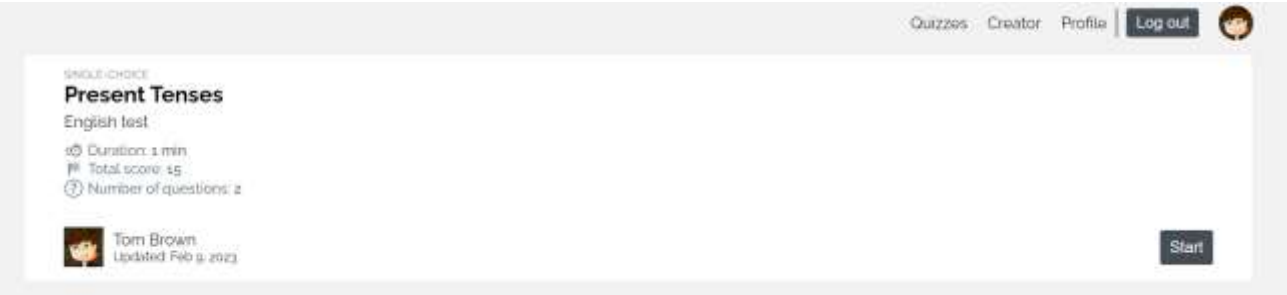

Рисунок 2.19 – Результат створення тесту

Отже, це означає, що тест було успішно створено.

Таким чином, користувачі можуть створювати різного роду тестування та публікувати їх. При наявності помилок у введених даних тесту, існує можливість відредагувати тест. Це можна виконати з початкової сторінки «Quizzes». Знайшовши там власний тест та натиснувши «Edit», відкривається конструктор, де можна внести зміни.

Тепер, іншим користувачам доступний даний тест (див. рис. 2.20).

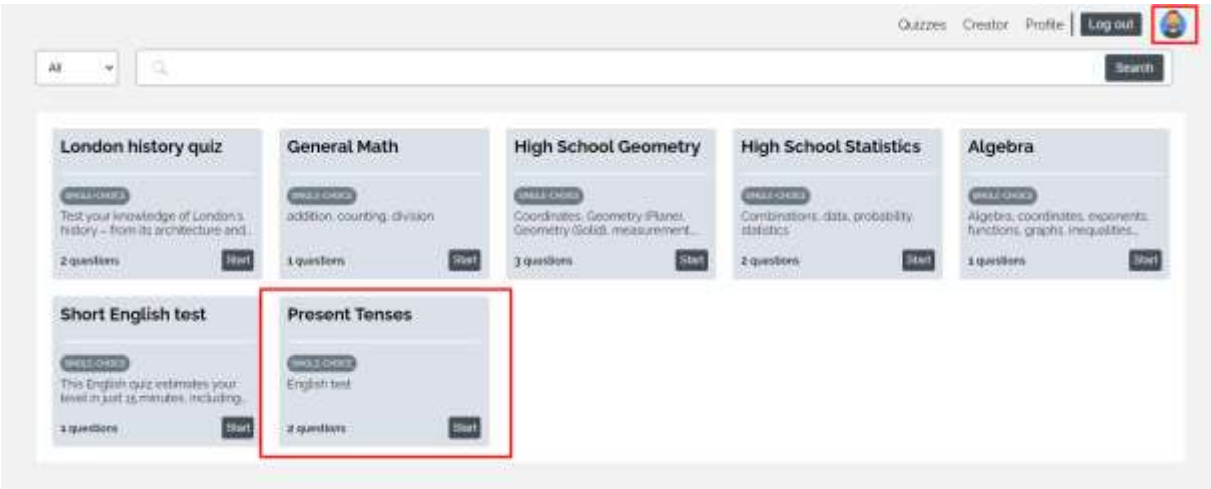

Рисунок 2.20 – Доступність тесту після його публікації

Натиснувши кнопку «Start» на ньому ми можемо розпочати тестування. Нам відкривається вікно із запитанням та відповідями (див. рис. 2.21).

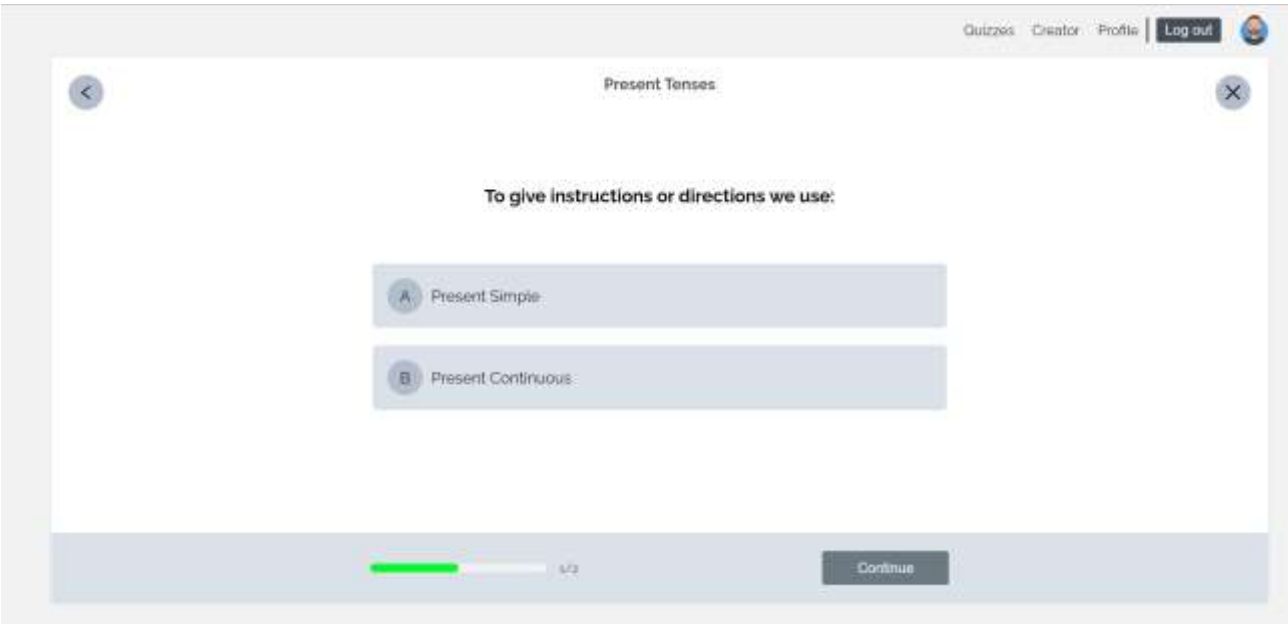

Рисунок 2.21 – Сторінка проходження тестування

Після надання відповідей, ми можемо завершити тест натиснувши на кнопку «Finish». Врешті решт, нам відображаються результати тестування у формі діаграми та додаткових даних із результатом (див. рис. 2.22).

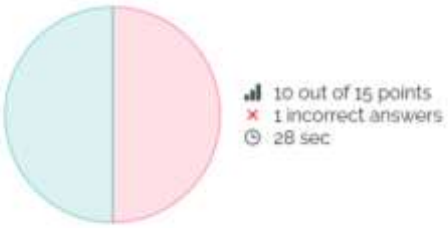

Рисунок 2.22 – Результат проходження тестування

Дані профілю та історія проходження тестів розміщена на сторінці «Profile». Її структуру можна поділити на дві частини. Перша частина – загальна інформація про акаунт користувача. Даний користувач пройшов лиш один тест та не створив жодного, що ми й можемо спостерігати на даній сторінці та відповідних діаграмах (див. рис. 2.23).

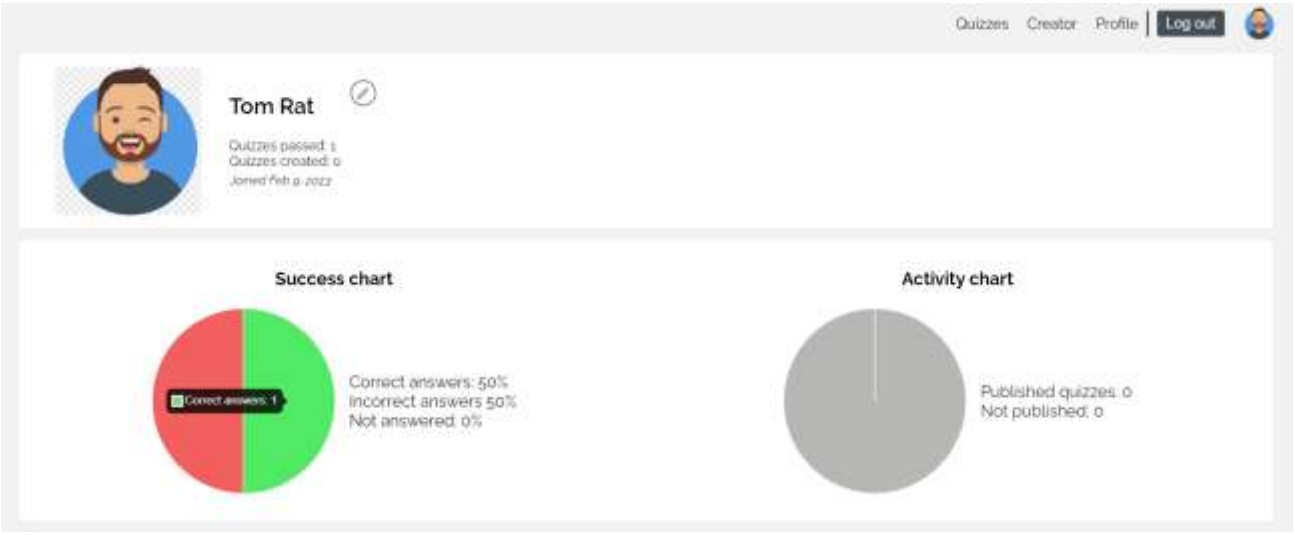

Рисунок 2.23 – Загальна інформація про користувача на сторінці «Profile»

Друга частина – узагальнення по пройдених тестах. Тут передбачалось відображення підсумків по кожному проходженню тесту та кожній спробі. Переконатись у коректності роботи цієї логіки застосунку ми можемо на рисунку 2.24.

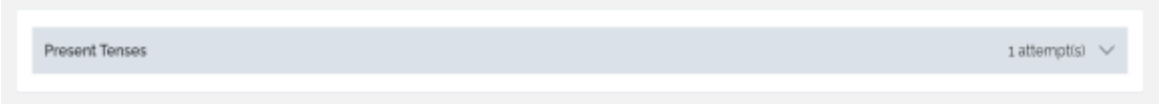

Рисунок 2.24 – Елемент з підсумками спроб складання тесту

Для перегляду детальнішої інформації натискаємо на елемент тесту. Під ним з’являється розгорнута інформація із спробами проходження даного тесту (див. рис. 2.25).

Рисунок 2.25 – Вміст елементу з підсумками спроб складання тесту

Отже, вебсистема була протестована. В ході тестування усі знайдені недоліки у роботі системи було усунено.

2.7 Висновок до другого розділу

У другому розділі кваліфікаційної роботи було здійснено проєктування, реалізацію та виконано ручне тестування вебсистеми для проведення тестувань. Описано структурну модель та розроблено структурну схему реалізації клієнтської і серверної частини системи. Також, було запроєктовано поведінку системи, що супроводжувалось демонстрацією діаграми станів та діаграми діяльності.

Окрім того, було розроблено моделі даних, детально описано інформаційні сутності, їх найменування, призначення, атрибути та зв’язки. Для представлення даних у системі, було розроблено концептуальну, логічну та фізичну моделі даних. Завершенням проєктування системи стало проєктування інтерфейсу користувача, а саме обґрунтування вибору колірної схеми та структури шаблонів інтерфейсу.

Після завершення вищезгаданого, було описано реалізацію основних модулів системи. З клієнтської частини, показано реалізацію компонентів, сторінок, загальний опис основних компонентів інтерфейсу та спосіб задання стилів, реалізацію форм та здійснення їх валідації, інтеграцію власних хуків, зображено потік даних застосунку та сервіси для взаємодії із сервером. Із сторони бекенду, було представлено реалізацію сервера з використанням фреймворку Express.js, показано підключення до бази даних, наведено приклади застосування мідлварів, продемонстровано клас застосунку, описано реалізовані контролери та сервіси.

Фінальним етапом даного розділу було проведення тестування та валідації роботи системи. В ході роботи над цією частиною було протестовано реалізацію реєстрації користувача, головну сторінку застосунку, конструктор тестів, роботу модуля проходження тестування, підрахунок балів та відображення результатів, перевірено коректність даних, відповідність результатів тестувань діаграмам профілю користувача,та інші.

РОЗДІЛ 3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Соціальні та психологічні фактори ризику при користуванні вебсистемою для проведення тестувань

Користувачі інформаційних систем стикаються із значним впливом сторонніх факторів, що безпосередньо випливають на їх безпеку. Безпека є ключовим поняттям у сфері безпеки життєдіяльності. При наявності різних соціальних та психологічних форм тиску на користувачів інформаційних технологій, стан їх захищеності від ризику зазнати шкоди та стати суб’єктом небезпеки високий. Безпека може визначатись як рівень відсутності ризику або неприпустимого ризику, пов'язаного з можливістю завдання будь-якої шкоди життю та здоров'ю людини незалежно від умов існування. Як можемо побачити, термін «ризик», «безпека» і «небезпека» тісно пов’язані між собою. Відмінність полягає в тому, що ризик конкретизує характер небезпеки, яка в свою чергу лише вказує на потенційну можливість завдання шкоди. Він вказує на ймовірність та тяжкість наслідків [29].

Користувач розробленої вебсистеми і користувачі інформаційних систем загалом перебувають в зоні ризику стати жертвою онлайн-шахрайства, насадження різних протиправних ідей і тд. Тобто, вони підвергається соціальним небезпекам. Соціальними називаються небезпеки, що широко розповсюджуються в суспільстві і загрожують життю і здоров’ю людей. До соціальних небезпек належать всі форми насилля, вживання алкоголю, наркотиків, паління, шахрайство, самогубство та інші дії, що здатні принести шкоду здоров’ю людей [30].

Звичайно, користувачами даної вебсистеми, та й взагалі більшості інформаційних систем та технологій, є люди. Саме вони є носіями соціальних небезпек, що створюють певні соціальні групи. Розповсюдження соціальних небезпек зумовлено особливостями поведінки цих людей.

Соціальні небезпеки є досить чисельні, наприклад, всі протиправні (незаконні) форми насилля, вживання речовин, що порушують психологічну і

фізіологічну рівновагу людини (алкоголь, наркотики тощо), шахрайство, шарлатанство, самогубство тощо. Причини соціальних небезпек породжуються соціально-економічними процесами, що відбуваються у суспільстві [31].

У контексті розробки і використання інформаційних систем соціальні небезпеки можуть представляти потенційні загрози для безпеки їх користувачів. Наприклад, соціальний інжиніринг, шахрайство, крадіжка особистої інформації та інші соціально-орієнтовані атаки можуть становити ризик для безпеки користувачів інформаційних систем. Основною причиною соціальних небезпек є недобросовісна поведінка людей або зловживання інформаційними ресурсами. Ось декілька типових соціальних небезпек, з якими можуть стикатися користувачі інформаційних систем:

– фішинг і шахрайство: це методи соціального обману, коли зловмисники намагаються отримати конфіденційні дані (такі як паролі, номери кредитних карток тощо) шляхом імітації довіреної особи або організації. Користувачі можуть отримувати підроблені електронні листи або веб-сторінки, які спонукають їх розкрити свої особисті дані;

– кіберзагрози: це включає в себе різноманітні види кібератак, такі як віруси, троянські програми, шпигунське програмне забезпечення та інші шкідливі програми, які можуть пошкодити інформаційну систему або викрасти конфіденційні дані користувача;

– кібербулінг і онлайн-злочини: інтернет надає анонімність, що може призвести до зловживання та надмірної агресії в онлайн-середовищі. Кібербулінг включає в себе загрози, образи, переслідування та інші форми небажаної поведінки, які можуть негативно впливати на психологічний стан користувачів;

– втрата конфіденційності: неправильне керування інформацією, слабкі паролі, недостатні заходи безпеки можуть призвести до витоку конфіденційної інформації. Це може мати серйозні наслідки, зокрема втрату фінансових даних, особистої ідентифікації та іншої конфіденційної інформації;

– залежність від технологій: інтенсивне використання інформаційних систем та соціальних мереж може викликати залежність, що впливає на психологічний стан користувачів та їхнє соціальне життя.

Для запобігання соціальним небезпекам користувачів інформаційних систем, необхідно приділяти увагу кібербезпеці. Це включає в себе навчання користувачів про безпекові практики, використання надійного програмного забезпечення, регулярне оновлення систем, аутентифікацію та авторизацію, а також встановлення механізмів виявлення та реагування на потенційні загрози. Крім того, важливо підтримувати правила етики та взаємодії в онлайн-середовищі, щоб забезпечити безпеку та повагу до користувачів інформаційних систем.

Стійко підвищують імовірність наразитись на небезпеку постійні функціональні зміни в нервовій системі або інших системах чи органах, що мають хворобливий характер або близький до цього стан. Такі зміни не означають непрацездатності, однак можуть чинити несприятливий вплив на людину з точки зору її безпеки (наприклад, головні болі, серцеві захворювання, цукровий діабет та інші). В основному перебіг хвороби позначається на поведінці людини, частково безпосередньо – у вигляді слабкості, недомагання, а частково побічно шляхом загального впливу на психіку (наприклад, подавленість, депресія, роздратованість), підвищуючи тим самим імовірність наразитись на небезпеку [32].

Частим явищем у користувачів інформаційних систем, в тому числі вебсистем, є постійне зловживання комп'ютера, смартфона або інших пристроїв з екранами. Це може призвести до надмірного напруження очей і спричинити синдром комп'ютерного зору. Це може викликати симптоми, такі як сухість, свербіння, роздратування очей, головні болі і тд. Підвищення захищеності осіб, що страждають такими недугами можна досягти перш за все шляхом постійних медичних оглядів та необхідного лікування.

Завантаження великою кількістю інформації, постійні сповіщення, відкриті вкладки, електронна пошта та соціальні мережі можуть призвести до надмірного стресу і перенавантаження мозку. Це може спричиняти втому, розсіяність, погіршення концентрації та психологічний дискомфорт.

Підвищують імовірність наразитись на небезпеку порушення зв'язку між сенсорними та руховими центрами вищих відділів нервової системи. Внаслідок

таких порушень людина не здатна з необхідною швидкістю та точністю реагувати на зовнішні впливи, що сприймаються її органами чуття [32]. Вказані порушення можуть бути компенсовані в першу чергу завдяки правильному розподілу уваги, яка часто в активних користувачів ІС розсіюється.

Використання інформаційних систем може призводити до підвищеного рівня стресу, особливо в сучасному світі, де постійно доступна інформація, високі вимоги до продуктивності та постійний контакт. Постійний стрес може мати негативний вплив на нервову систему та загальне психофізичне здоров'я.

На імовірність наразитись на небезпеку впливає неврівноваженість емоційних процесів. Наприклад, підвищена емоційна збудливість, раптові зміни радості та злоби, гострі емоційні реакції на незначні зовнішні подразнення підвищують загрозу нещасного випадку. Зовнішній вплив неврівноваженості емоційних процесів іноді позначається побічно, наприклад, у формі легковажності, необдуманості вчинків, поспішності їх виконання. Щоб позбутися неврівноваженості емоційних процесів необхідно займатись самовихованням та виробляти самовладання [32].

3.2 Психофізіологічне розвантаження для користувачів вебсистеми

Сучасне робоче середовище часто характеризується високою конкуренцією, терміновими строками, постійним тиском і вимогами до продуктивності. Це може призводити до стресу, вибухів емоцій, втрати мотивації та виживання на роботі.

Часто у користувачів ПК та працівників, які використовуються ПК спостерігається відсутність м'язової активності. Це помітно погіршує психофізіологічний стан людини та супроводжується низькою мотивацією при виконанні завдань, зниженій емоційній стійкості, погіршеному настрою, підвищеній дратівливості та порушеному сну. Тому важливо вживати заходи для психофізіологічного розвантаження [33].

При проведенні сеансів психофізіологічного розвантаження рекомендується використовувати деякі елементи методу аутогенного

тренування, який ґрунтується на свідомому застосуванні комплексу взаємопов'язаних прийомів психічної саморегуляції й виконанні нескладних фізичних вправ із словесним самонавіюванням [34].

У рекомендованому сеансі, який має проводитися в кімнаті психофізіологічного розвантаження з відповідним інтер'єром та кольоровим оформленням, виділяються три періоди, що відповідають фазам відновлювального процесу.

Перший період – абстрагування працівників від робочої обстановки - відповідає фазі залишкового збудження. Лунає повільна мелодійна музика, пташиний спів. Обравши зручну позу, працівники адаптуються і психологічно готуються до наступних періодів. Другий – заспокоєння – відповідає фазі відновлювального гальмування. Пропонується показ фотослайдів із зображеннями квітучого лугу, березового гаю, гладенької поверхні ставка тощо. Через навушники транслюється спокійна музика, а на її фоні негучно, повільно висловлюються заспокійливі формули аутогенного тренування (тричі): «Я повністю розслаблений, спокійний»; «Моє дихання рівне, спокійне»; «Моє тіло важке, гаряче, розслаблене, я абсолютно розслаблений, лоб холодний, голова легка». Як функціональне освітлення застосовують зелене світло. Третій період – активізація – відповідає фазі підвищеної збудженості [34].

На початку періоду світло вимкнене, через певний час на екрані з'являється червона пляма, розміри і яскравість якої поступово збільшуються. Наприкінці періоду лунає бадьора музика. Тричі вимовляються мобілізуючі формули аутогенного тренування, яким мають передувати глибоке вдихання та довге глибоке видихання: «Я бадьорий, свіжий, веселий, у мене гарний настрій»; «Я повний енергією, я готовий діяти».

Після сеансів психофізіологічного розвантаження у працівників зменшується відчуття втоми, з'являються бадьорість, гарний настрій. Загальний стан відчутно поліпшується [34].

3.3 Висновок до третього розділу

В результаті виконання даного розділу було проаналізовано наступні питання з безпеки життєдіяльності та основ охорони праці: соціальні та психологічні фактори ризику, психофізіологічне розвантаження для користувачів розробленої вебсистеми. Було окреслено термін «ризик», з'ясовано як він пов'язаний з небезпекою. Описано фактори, які відносяться до соціального та психологічного ризиків. Це стосується як загального бачення так і в контексті інформаційних систем, в тому числі розробленої вебсистеми.

Також, в даному розділі здійснено огляд психофізіологічного розвантаження та було описано необхідність приймання дій для досягнення цієї мети користувачами інформаційних систем. Описано проблеми, які може спричинити відсутність розвантаження та фактори, що виникають при постійному психофізіологічному навантаженні. Наведено методи для профілактики і збереження фізичного та психічного здоров'я, а також техніки психофізіологічної релаксації користувачів вебсистеми для проведення тестувань.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи освітнього рівня «Бакалавр» було реалізовано вебсистему для проведення тестувань з використанням Node.js, React.js та PostgreSQL. Було сформовано вимоги до системи, спроєктовано, реалізовано та протестовано застосунок.

За допомогою таких технологій як Typescript, React, SCSS, Redux було реалізовано клієнтську частину системи. Серверну частина було реалізовано засобами Typescript, Express.js, TypeORM та СУБД PostgreSQL. Розробка застосунку проводилась у середовищі WebStorm.

У першому розділі кваліфікаційної роботи було:

– проаналізовано предметну область вебсистеми для проведення тестувань;

– сформовано вимоги до структури, безпеки та інтерфейсу користувача вебсистеми;

– продемонстровано діаграми варіантів використання та показано, яким чином актори можуть взаємодіяти із системою;

– обґрунтовано вибір методу вирішення задачі;

– описано технології, що використовуватимуться в ході реалізації системи.

У другому розділі кваліфікаційної роботи було:

– описано структурну модель та розроблено структурну схему реалізації клієнтської і серверної частини системи;

– запроєктовано поведінку системи з демонстрацією діаграми станів та діаграми діяльності;

– описано інформаційні сутності, їх найменування, призначення, атрибути та зв’язки, подано концептуальну, логічну та фізичну моделі даних;

– запроєктовано інтерфейс користувача, обґрунтовано вибір колірної схеми та структури шаблонів інтерфейсу;

– описано реалізацію основних модулів клієнтської частини системи;

– описано реалізацію основних модулів серверної частини системи;

– проведено тестування системи.

В ході тестування усі виявлені недоліки та помилки у роботі системи було усунено.

У третьому розділі було проаналізовано соціальні та психологічні фактори ризику, а також психофізіологічне розвантаження для користувачів розробленої вебсистеми. Було визначено поняття «ризик» та його зв'язок з небезпекою. Описані фактори, які впливають на соціальний та психологічний ризики, як загальні для різних ситуацій, так і специфічні для інформаційних систем, зокрема розробленої вебсистеми. Також, у цьому розділі було розглянуто питання психофізіологічного розвантаження та висвітлено необхідність застосування відповідних заходів користувачами інформаційних систем для досягнення цієї мети. Описано проблеми, які виникають при відсутності розвантаження та фактори, що супроводжують постійне психофізіологічне навантаження. Наведено методи профілактики та збереження фізичного та психічного здоров'я, а також техніки психофізіологічної релаксації для користувачів користувачів вебсистеми для проведення тестувань.

ПЕРЕЛІК ДЖЕРЕЛ

1. Тест як форма навчання, контролю та оцінювання знань. Методика викладання економічних дисциплін. Українські підручники та статті – Бібліотека Posibniki.com.ua. Українські підручники, посібники та статті - Бібліотека Posibniki. Електронна бібліотека підручників онлайн. URL: https://posibniki.com.ua/post-test-yak-forma-navchannya-kontrolu-ta-ocinuvannya-znan (дата звернення: 08.02.2023).

2. How to ensure the availability, integrity, and confidentiality of your apps. F5. URL: https://www.f5.com/company/blog/how-to-ensure-the-availability-integrity-and-confidentiality-of-your-apps (дата звернення: 10.02.2023).

3. Діаграми UML для моделювання процесів і архітектури проекту. Evergreen - web розробка і діджиталізація бізнесу за допомогою AI продуктів. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html (дата звернення: 18.02.2023).

4. Учасники проектів Вікімедіа. Сценарій використання – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Сценарій_використання (дата звернення: 20.02.2023).

5. Contributor T. What is web application development? | Definition from TechTarget. Cloud Computing. URL: https://www.techtarget.com /searchcloudcomputing/definition/web-application-development (дата звернення: 02.03.2023).

6. Що таке фреймворки і для чого вони використовуються при веб-розробці - Блог VOLL. Веб студія VOLL - Інтернет-маркетинг Агентство. URL: https://voll.com.ua/uk/blog/frejmvorki-dlya-veb-rozrobki (дата звернення: 07.03.2023).

7. Учасники проектів Вікімедіа. Бібліотека підпрограм – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Бібліотека_підпрограм (дата звернення: 09.03.2023).

8. Веб технології – що це таке та які найпопулярніші?. FutureNow. URL: https://futurenow.com.ua/veb-tehnologiyi-shho-tse-take-ta-yaki-najpopulyarnishi/.

9. Переваги веб-розробки на React у 2022. Маркетинговое агентство MABP в Харькове - услуги интернет-маркетинга в Украине. URL: https://mavr.ua/ua/perevagi-veb-rozrobki-na-react-u-2022/ (дата звернення: 13.03.2023).

10. Учасники проектів Вікімедіа. WebStorm – вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/WebStorm (дата звернення: 17.03.2023).

11. WebStorm: The Smartest JavaScript IDE, by JetBrains. JetBrains. URL: https://www.jetbrains.com/webstorm/ (дата звернення: 21.03.2023).

12. Учасники проектів Вікімедіа. React – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/React (дата звернення: 25.03.2023).

13. Що таке Virtual DOM?. Codeguida. URL: https://codeguida.com/post/1561 (дата звернення: 28.03.2023).

14. Учасники проектів Вікімедіа. Node.js – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Node.js (дата звернення: 05.04.2023).

15. Express JS – платформа для створення сторінок. Глянець – Розробка і підтримка сайтів. URL: https://glyanec.net/ua/blog/express-js-platforma-dlya-stvorennya-storinok (дата звернення: 09.04.2023).

16. Учасники проектів Вікімедіа. PostgreSQL – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/PostgreSQL (дата звернення: 13.04.2023).

17. Рендеринг елементів – React. React – JavaScript-бібліотека для створення користувацьких інтерфейсів. URL: https://uk.reactjs.org/docs/rendering-elements.html (дата звернення: 15.04.2023).

18. Огляд хуків – React. React – JavaScript-бібліотека для створення користувацьких інтерфейсів. URL: https://uk.reactjs.org/docs/hooks-overview.html (дата звернення: 17.04.2023).

19. Redux Fundamentals, Part 3: State, Actions, and Reducers. Redux. URL: https://redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers (дата звернення: 20.04.2023).

20. Contributors to Wikimedia projects. Schema migration - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Schema_migration (дата звернення: 24.04.2023).

21. Model View Controller (MVC). Phalcon. URL: https://docs.phalcon.io/4.0/en/mvc (дата звернення: 29.04.2023).

22. Nadel B. A Better Understanding Of MVC (Model-View-Controller) Thanks To Steven. Bennadel. URL: https://www.bennadel.com/blog/2379-a-better-understanding-of-mvc-model-view-controller-thanks-to-steven-neiland.htm (дата звернення: 01.05.2023).

23. Діаграма діяльності. Studwood. URL: https://studwood.net/1884530/informatika/diagrama_diyalnosti (дата звернення: 07.05.2023).

24. Wikiwand - Діаграма станів (UML). Wikiwand. URL: https://www.wikiwand.com/uk/Діаграма_станів_(UML) (дата звернення: 12.05.2023).

25. Діаграма потоків даних. www.wiki.uk-ua.nina.az. URL: https://www.wiki.uk-ua.nina.az/Діаграма_потоків_даних.html (дата звернення: 16.05.2023).

26. Онлайн-конспект з інформатики - Урок 20. Поняття сутності, атрибута, ключа, зв'язку. sites.google.com. URL: https://sites.google.com/view/onlayn-konspekt-z-informatiku/10-клас/урок-20-поняття-сутності-атрибута-ключа-звязку (дата звернення: 21.05.2023).

27. Darina. Колористика в дизайні. Основи теорії кольору – поради веб-дизайнерам початківцям. Komarov.design - Графічний дизайн. URL: https://www.komarov.design/koloristika-v-dizaini-osnovi-tieoriyi-koloru-poradi-vieb-dizainieram-pochatkivtsiam/ (дата звернення: 24.05.2023).

28. Contributors to Wikimedia projects. Social risk management - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Social_risk_management (дата звернення: 29.05.2023).

29. Желібо Є.П., Зацарний В.В. Безпека життєдіяльності. Підручник. – К.: Каравела, 2009.

30. Безпека життєдіяльності: Навчально-методичний посібник. / [Укладачі: В.І. Кошель, Г.П. Сав'юк, Б.С. Дзундза] – Івано-Франківськ: НАІР, 2018. – 163 с

31. Соціальні та політичні небезпеки. Безпека життєдіяльності, охорона праці.Allreferat.com.ua.URL: https://allreferat.com.ua/uk/Bezpeka_guttediyalnosti_ohorona_praci/referat/4042 (дата звернення: 05.06.2023).

32. Джигирей В. С., Жидецький В. Ц. Безпека життєдіяльності. Навчальний посібник. Вид. 3-те, доповнене. Львів: Афіша,. 2000. 256 с.

33. Грибан В.Г., Негодченко О.В. Охорона праці. – К.: Центр учбової літератури, 2009. 209 с.

34. Лапін, В.М. Безпека життєдіяльності людини / В.М. Лапін. – К. : Знання, 2007. – 332 с.

ДОДАТКИ

Додаток А

Найменування та формулювання варіантів використання вебсистеми

Актор	Найменування	Формулювання
Незареєстрований користувач	Здійснити реєстрацію	Дозволяє створити обліковий запис користувача згідно з даними, які він ввів у форму реєстрації
	Знайти тест	Надає можливість здійснити пошук публічних тестів
	Виконати пошук за іменем	Розширює можливості знаходження тесту та дозволяє здійснити пошук тесту за його іменем
	Дізнатись інформацію про тест	Дозволяє детальніше дізнатись про тест
	Розпочати тест	Дозволяє розпочати складання тесту
	Переглянути результат тестування	Після надання відповідей на усі запитання, система підсумовує та виводить дані про кількість правильних та неправильних відповідей
	Переглянути результат тестування	Після надання відповідей на усі запитання, система підсумовує та виводить дані про кількість правильних та неправильних відповідей
Зареєстрований користувач	Виконати вхід	Дозволяє автентифікувати користувача за електронною адресою та паролем

Актор	Найменування	Формулювання
	Знайти тест	Дозволяє знайти тест серед публічних та приватних тестів
	Виконати пошук за іменем	Розширює можливості знаходження тесту та дозволяє здійснити пошук тесту за його іменем
	Виконати фільтрацію	Розширює можливості знаходження тесту та дозволяє вивести публічні тести, особисті тести та усі тести разом відповідно до параметра фільтра
	Розпочати тестування	Дозволяє розпочати складання тесту
	Переглянути результат тестування	Після надання відповідей на усі запитання, система підсумовує та виводить дані про кількість правильних та неправильних відповідей
	Створити тест	Дозволяє створити тест, вказавши інформацію про нього, питання, відповіді та надавши дані про коректні варіанти відповідей
	Редагувати тесту	Дозволяє виконати редагування стовреного даним користувачем тесту
	Видалити тест	Дає можливість видалити створений поточним користувачем тест

Актор	Найменування	Формулювання
	Переглянути та змінити дані профілю	Дозволяє переглянути і відредагувати фото та ім’я користувача
	Переглянути підсумки тестувань	Дає можливість переглянути історію спроб проходження тестів та їх результати
	Вийти	Надає можливість вийти із системи

Додаток Б

Глибокий аналіз сутностей

Інформаційний об’єкт сутності	Атрибут сутності	Тип сутності	Ключі
Усі об’єкти	id	uuid	P-ключ
	createdAt	timestamp without time zone	-
	updatedAt	timestamp without time zone	-
user	email	character varying	F-ключ
	password	character varying	-
	fullName	character varying	-
token	userId	uuid	F-ключ
	refreshToken	character varying	-
quiz	userId	uuid	F-ключ
	published	boolean	-
	type	character varying	-
	score	integer	-
	content	character varying	-
	time	integer	-
	title	character varying	-
quiz_question	quizId	uuid	F-ключ
	active	boolean	-
	score	integer	-
	content	character varying	-
	type	character varying	-
quiz_answer	questionId	uuid	F-ключ
	active	boolean	
	correct	boolean	-
	content	character varying	-

Інформаційний об’єкт сутності	Атрибут сутності	Тип сутності	Ключі
take	quizId	uuid	F-ключ
	userId	uuid	F-ключ
	status	character varying	-
	currenrScore	integer	-
	totalScore	integer	-
	content	character varying	-
	title	character varying	-
	spentTime	integer	-
take_question	takeId	uuid	F-ключ
	score	integer	-
	content	character varying	-
	correctlyAnswered	boolean	-
	answered	boolean	-
take_answer	questionId	uuid	F-ключ
	content	character varying	
	chosen	boolean	-
	correct	boolean	-

Додаток В

Реалізація сторінки авторизації

```
import React, { FormEvent, useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import { useAppDispatch, useAppSelector } from
'../hooks/useAppDispatch';
import { authActions } from '../store/auth';

import FormInput from '../components/form-input';
import Button from '../components/button';

import { MdAlternateEmail as EmailIcon } from 'react-icons/md';
import { AiFillLock as LockIcon } from 'react-icons/ai';
import { UserRoutes } from '../common/enums';

import useInput from '../hooks/useInput';

const Login = () => {
  const dispatch = useAppDispatch();
  const navigate = useNavigate();

  const [loginError, setLoginError] = useState('');

  const { isAuth, isLoading } = useAppSelector((state) =>
state.auth);

  const emailInput = useInput('', { isEmpty: true, isEmail: true
});
  const passwordInput = useInput('', {
    isEmpty: true,
    minLength: 8,
    maxLength: 15,
  });

  const handleSubmitForm = (e: FormEvent) => {
    e.preventDefault();

    dispatch(authActions.login({ email: emailInput.value,
password: passwordInput.value }))
      .unwrap()
      .catch((e) => setLoginError(e.response.data.message));
  };

  if (isLoading) {
    return <div>Loading...</div>;
  }

  if (isAuth) {
    navigate(UserRoutes.Quizzes);
  }
```

```
  return (
    <div className="auth-form-wrapper">
      <form className="auth-form" onSubmit={handleSubmitForm}>
        <h2 className="auth-form__title">Log In</h2>
        {loginError && <div className="auth-form__submit-
error">{loginError}</div>}
        <div className="auth-form__fieldset">
          <label className="auth-form__label">
            Email<span className="required">*</span>
          </label>
          <FormInput
            className={emailInput.isDirty && !emailInput.isValid ?
'error-input' : ''}
            type="email"
            name="email"
            value={emailInput.value}
            icon={<EmailIcon />}
            placeholder="example@gmail.com"
            onChange={emailInput.onChange}
            onBlur={emailInput.onBlur}
          />
          {emailInput.isDirty && emailInput.isEmpty.value && (
            <span className="auth-
form__error">{emailInput.isEmpty.errorMessage}</span>
          )}

          {emailInput.isDirty && !emailInput.isEmpty.value &&
emailInput.emailError.value && (
            <span className="auth-
form__error">{emailInput.emailError.errorMessage}</span>
          )}

          <label className="auth-form__label">
            Password<span className="required">*</span>
          </label>
          <FormInput
            className={passwordInput.isDirty &&
!passwordInput.isValid ? 'error-input' : ''}
            name="password"
            type="password"
            value={passwordInput.value}
            placeholder="Password"
            icon={<LockIcon />}
            onChange={passwordInput.onChange}
            onBlur={passwordInput.onBlur}
          />

          {passwordInput.isDirty && passwordInput.isEmpty.value &&
(
            <span className="auth-
form__error">{passwordInput.isEmpty.errorMessage}</span>
          )}

          {passwordInput.isDirty &&
            !passwordInput.isEmpty.value &&
```

```
            passwordInput.minLengthError.value && (
              <span className="auth-
form__error">{passwordInput.minLengthError.errorMessage}</span>
            )}

          {passwordInput.isDirty &&
            !passwordInput.isEmpty.value &&
            passwordInput.maxLengthError.value && (
              <span className="auth-
form__error">{passwordInput.maxLengthError.errorMessage}</span>
            )}
        </div>

        <Button
          className="auth-form__submit"
          type="submit"
          disabled={!emailInput.isValid || !passwordInput.isValid}
        >
          Log in
        </Button>
        <p className="auth-form__text">
          Don`t have an account? <Link to={'/signup'}>Sign
up</Link>
        </p>
      </form>
    </div>
  );
};

export default Login;
```

Додаток Г

Програмний коду хука «useValidation»

```
import { useEffect, useState } from 'react';
import { IValidations, IUseValidation } from
'./common/interfaces/use-validation';

interface ICondition {
  value: boolean;
  errorMessage?: string;
}

const useValidation = (value: string, validations: IValidations):
IUseValidation => {
  const [isEmpty, setIsEmpty] = useState<ICondition>({ value: true
});
  const [minLengthError, setMinLengthError] =
useState<ICondition>({ value: false });
  const [maxLengthError, setMaxLengthError] =
useState<ICondition>({ value: false });
  const [emailError, setEmailError] = useState<ICondition>({
value: false });

  const [isValid, setIsValid] = useState(false);

  useEffect(() => {
    for (const validation in validations) {
      switch (validation) {
        case 'minLength':
          validations?.minLength && value.length <
validations.minLength
            ? setMinLengthError({
                value: true,
                errorMessage: `Should have
${validations.minLength} characters at least`,
              })
            : setMinLengthError({ value: false });
          break;
        case 'isEmail':
          const regularExpression =

/^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(".+"))@((\[[
0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-
9]+\.)+[a-zA-Z]{2,}))$/;
          regularExpression.test(String(value).toLowerCase())
            ? setEmailError({ value: false })
            : setEmailError({ value: true, errorMessage: 'Email
format is not valid' });
          break;
        case 'maxLength':
          validations?.maxLength && value.length >
validations.maxLength
```

```
            ? setMaxLengthError({
                value: true,
                errorMessage: `Shouldn't have more characters then
${validations.maxLength}`,
              })
            : setMaxLengthError({ value: false });
          break;
        case 'isEmpty':
          value
            ? setIsEmpty({ value: false })
            : setIsEmpty({ value: true, errorMessage: 'Cannot be
empty' });
          break;
      }
    }
  }, [value]);

  useEffect(() => {
    if (isEmpty.value || emailError.value || minLengthError.value
|| maxLengthError.value)
      setIsValid(false);
    else setIsValid(true);
  }, [isEmpty, emailError, minLengthError, maxLengthError]);

  return {
    isEmpty,
    minLengthError,
    maxLengthError,
    emailError,
    isValid,
  };
};

export default useValidation;
```

Додаток Ґ

Програмний код клієнтського сервісу «AuthService»

```
import HttpService from '../http/http.service';
import { ILoginUser, ILogoutResponse, IAuthResponse, ISignupUser }
from '../../common/interfaces';
import { ContentType, HttpMethod } from '../../common/enums';

class AuthService {
  private readonly path: string;
  private http: HttpService;
  constructor(path: string, http: HttpService) {
    this.path = path;
    this.http = http;
  }

  public login = async (payload: ILoginUser):
Promise<IAuthResponse> => {
    return
this.http.load<IAuthResponse>(`${this.path}/auth/login`,
{ method: HttpMethod.POST, payload,});};

  public signup = async (payload: ISignupUser): Promise<IAuthResp
onse > => {
    let formData = new FormData();

    if (payload.fullName) formData.append('fullName',
payload.fullName);
    if (payload.avatar) formData.append('avatar', payload.avatar);
    formData.append('email', payload.email);
    formData.append('password', payload.password);
    return
this.http.load<IAuthResponse>(`${this.path}/auth/register`, {
      method: HttpMethod.POST,
      contentType: ContentType.MULTIPART_FORM_DATA,
      payload: formData,
    });
  };

  public logout = async (): Promise<ILogoutResponse> => {
return this.http.load<ILogoutResponse>(`${this.path}/auth/logout`,
{ method: HttpMethod.POST});
  };

  public refresh = async (): Promise<IAuthResponse> => {
    return
this.http.load<IAuthResponse>(`${this.path}/auth/refresh`, {
      method: HttpMethod.GET,
    });
  };
}
export default AuthService;
```

Додаток Д

Реалізація класу «App»

```
import express from 'express';
import cors from 'cors';
import cookieParser from 'cookie-parser';
import authMiddleware from './middlewares/authMiddleware';
import errorHandlerMiddleware from
'./middlewares/errorHandler.middleware';

import { IController } from './common/interfaces';

class App {
  public app: express.Application;
  constructor(controllers: IController[]) {
    this.app = express();
    this.initializeMiddlewares();
    this.initializeControllers(controllers);
    this.initializeErrorHandling();
  }

  public listen() {
    this.app.listen(process.env.PORT, () => {
      console.log(`Server is running on ${process.env.PORT}
port`);
    });
  }

  private initializeControllers(controllers: IController[]) {
    controllers.forEach((controller) => {
      this.app.use('/', controller.router);
    });
  }

  private initializeMiddlewares() {
    this.app.use(express.json());
    this.app.use(cookieParser());
    this.app.use(
      cors({
        credentials: true,
        origin: process.env.CLIENT_URL,
      }),
    );
    this.app.use(authMiddleware);
  }

  private initializeErrorHandling() {
    this.app.use(errorHandlerMiddleware);
  }
}

export default App;
```

Додаток Е

Реалізація контролера «QuizController»

```
import { NextFunction, Request, Response, Router } from 'express';

import { HttpCode, RoleType } from '../common/enums';
import { IAuthRequest, IController, IDeepQuiz, IDeepUpdateQuiz }
from '../common/interfaces';

import validatePermission from
'../middlewares/validatePermission.middleware';

import QuizService from './quiz.service';

class QuizController implements IController {
  public path = '/quizzes';
  public router = Router();

  constructor(private readonly quizService: QuizService) {
    this.initializeRoutes();
  }

  private initializeRoutes() {
    this.router.get(`${this.path}/`,
this.getAllQuizzes.bind(this));
    this.router.get(`${this.path}/:id`,
this.getQuizById.bind(this));
    this.router.get(`${this.path}/:quizId/answer/:answerId`,
this.checkIfAnswerCorrect.bind(this));
    this.router.post(
      `${this.path}/`,
      validatePermission([RoleType.USER]),
      this.createDeepQuiz.bind(this),
    );
    this.router.put(
      `${this.path}/:id`,
      validatePermission([RoleType.USER]),
      this.updateDeepQuiz.bind(this),
    );
    this.router.delete(
      `${this.path}/:id`,
      validatePermission([RoleType.USER]),
      this.deleteQuiz.bind(this),
    );
  }

  private async getQuizById(req: IAuthRequest, res: Response,
next: NextFunction) {
    try {
      const quizId: string = req.params.id;
      let quiz;
```

```
      if (req.user) {
        const userId: string = req.user.id;
        quiz = await this.quizService.getDeepById(quizId, userId);
      } else {
        quiz = await this.quizService.getPublicDeepById(quizId);
      }


res.status(HttpCode.OK).send(this.quizService.hideCorrectAnswers(q
uiz));
    } catch (e) {
      next(e);
    }
  }

  private async getAllQuizzes(req: IAuthRequest, res: Response,
next: NextFunction) {
    try {
      let quizzes;

      if (req.user) {
        const userId: string = req.user.id;
        quizzes = await this.quizService.getAll(userId);
      } else {
        quizzes = await this.quizService.getPublic();
      }

      res.status(HttpCode.OK).send(quizzes);
    } catch (e) {
      next(e);
    }
  }

  private async createDeepQuiz(req: IAuthRequest, res: Response,
next: NextFunction) {
    try {
      const quizData: IDeepQuiz = req.body as unknown as
IDeepQuiz;
      const newQuiz = await
this.quizService.createDeep(req.user.id, quizData);

res.status(HttpCode.CREATED).send(this.quizService.hideCorrectAnsw
ers(newQuiz));
    } catch (e) {
      next(e);
    }
  }

  private async deleteQuiz(req: IAuthRequest, res: Response, next:
NextFunction) {
    try {
      const quizId: string = req.params.id as unknown as string;
      await this.quizService.delete(quizId);
      res.status(HttpCode.NO_CONTENT).send();
    } catch (e) {
```

```
      next(e);
    }
  }

  private async updateDeepQuiz(req: IAuthRequest, res: Response,
next: NextFunction) {
    try {
      const quizId: string = req.params.id as unknown as string;
      const quizData: IDeepUpdateQuiz = req.body as unknown as
IDeepUpdateQuiz;
      const updatedQuiz = await
this.quizService.updateDeep(quizId, quizData);
      res.status(HttpCode.CREATED).send(updatedQuiz);
    } catch (e) {
      next(e);
    }
  }

  private async checkIfAnswerCorrect(req: Request, res: Response,
next: NextFunction) {
    try {
      const { quizId, answerId } = req.params;
      const isCorrect = await
this.quizService.checkIfAnswerCorrect(quizId, answerId);
      res.status(HttpCode.OK).send({ correct: isCorrect });
    } catch (e) {
      next(e);
    }
  }
}

export default QuizController;
```