Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Кафедра математичного моделювання та аналізу даних

До захисту допущено
Кафедрою математичного моделювання та аналізу даних
протокол №___ від _____

Завідувач кафедри _____ Володимир СТРУКОВ
(підпис) *(імя, прізвище)*

«___»_____ 2026 р.

Кваліфікаційна робота
здобувача першого (бакалаврського) рівня вищої освіти

РОЗРОБЛЕННЯ ІНФОРМАЦІЙНОЇ АРХІТЕКТУРИ ТА КЛІЄНТСЬКОГО ФУНКЦІОНАЛУ ВЕБ-ПЛАТФОРМИ НАВЧАЛЬНОГО ТЕСТУВАННЯ

Спеціальність (спеціалізація) 122 Комп'ютерні науки

Освітня програма Комп'ютерні науки

Виконавець _____ Богдан БУХАР
(підпис) *(ім'я, прізвище)*

Науковий керівник _____ Ольга МАЦИЙ
(підпис) *(ім'я, прізвище)*

Харків – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Кафедра математичного моделювання та аналізу даних

ЗАТВЕРДЖУЮ
Завідувач кафедри ММтАД

_____Володимир СТРУКОВ
підпис ім'я, прізвище

"____" _____2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувача першого (бакалаврського) рівня вищої освіти

Бухар Богдана Євгенійовича

Спеціальність (спеціалізація) 122 «Комп'ютерні науки», _____

Освітня програма Комп'ютерні системи та мережі _____

1. Тема роботи: Розроблення інформаційної архітектури та клієнтського функціоналу веб-платформи навчального тестування

керівник роботи Маций Ольга Борисівна, доцент, кандидат технічних наук

затверджені наказом по Університету від 29 січня 2026 року № 4101-5/225

2. Строк здобувачем роботи "01" червня 2026 року

3. Вихідні дані до роботи: технічне завдання на розробку вебплатформи автоматизованого навчального тестування з використанням фреймворку Django та архітектури SPA.

4. Перелік питань, які потрібно розробити:

1. виконати порівняльний огляд існуючих систем онлайн-тестування та дослідити сучасні підходи до проєктування інформаційної архітектури освітніх вебзастосунків;
2. обґрунтувати вибір технологічного стеку розробки, спроєктувати реляційну базу даних та загальну логічну архітектуру системи;
3. розробити інформаційну архітектуру та логіку клієнтської взаємодії, спроєктувати ізольоване середовище тестування на базі

4. концепції Single Page Application (SPA) із дотриманням принципів «distraction-free» інтерфейсу;
5. виконати програмну реалізацію серверної та клієнтської частин вебплатформи, інтегрувавши механізми автентифікації, локального управління станом тесту, захисту даних та контролю часових обмежень іспиту;
6. провести комплексне тестування розробленої системи, перевірити коректність бізнес-сценаріїв та виконати аудит продуктивності, швидкодії і доступності інтерфейсу за допомогою сучасних засобів аналізу.

5. План роботи

№ з/п	Назви етапів роботи	Строк виконання етапів роботи	Примітка
1	Формулювання мети та повного переліку завдань для виконання кваліфікаційної роботи.	15.01.2026 – 30.01.2026	Виконано
2	Пошук та аналіз інформаційних джерел за темою КР.	30.01.2026 – 20.02.2026	Виконано
3	Проєктування інформаційної архітектури, вибір технологічного стеку та розробка структури реляційної бази даних вебплатформи.	20.02.2026 – 18.03.2026	Виконано
4	Програмна реалізація серверної логіки на базі фреймворку Django та клієнтської SPA-архітектури модуля тестування.	18.03.2026 – 02.04.2026	Виконано
5	Комплексне функціональне тестування системи, аналіз відмовостійкості та аудит продуктивності й доступності інтерфейсу.	02.04.2026 – 19.04.2026	Виконано
6	Підсумок отриманих результатів та оформлення пакету документів для процедури захисту КР в екзаменаційній комісії ННІ КН та ШІ.	19.04.2026 – 03.05.2026	Виконано

1. Дата видачі завдання 15 січня 2026 р.
2.

Здобувач вищої освіти _____ Б.Є. БУХАР
(підпис) (ініціали, прізвище)

Керівник роботи _____ О.Б. МАЦИЙ
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи складає 69 сторінок, із яких 50 сторінок основної частини, 17 рисунків, 2 таблиці, 27 найменувань списку використаних джерел та 4 додатками.

Метою дослідження є розроблення вебплатформи для організації навчального тестування з оптимізованою інформаційною архітектурою та сучасним клієнтським функціоналом.

Об’єкт дослідження – процес організації та автоматизації контролю знань у системах онлайн-тестування.

Предмет дослідження – методи проєктування інформаційної архітектури, клієнтського функціоналу та серверної логіки вебплатформ для автоматизованого тестування.

У роботі проведено аналіз сучасних систем онлайн-тестування та обґрунтовано необхідність створення спеціалізованої вебплатформи. Для реалізації програмного продукту використано мову програмування Python, вебфреймворк Django, JavaScript, HTML5, CSS3 та Tailwind CSS. У результаті розроблено вебзастосунок з підтримкою SPA-архітектури, механізмами автоматизованого оцінювання, авторизації користувачів та адміністрування тестового контенту.

Область застосування – освітні інформаційні системи, дистанційне навчання, автоматизований контроль знань та організація онлайн-тестування у закладах освіти.

Ключові слова: *вебплатформа, онлайн-тестування, single page application, інформаційна архітектура, django, javascript, tailwind css.*

ABSTRACT

The explanatory note to the qualification work consists of an introduction, three chapters, conclusions, a list of references, and appendices. The total volume of the work is 69 pages, including 50 pages of the main part, 17 figures, 2 tables, 27 references, and 4 appendices.

The purpose of the research is to develop a web platform for educational testing with optimized information architecture and modern client-side functionality.

The object of the research is the process of organizing and automating knowledge assessment in online testing systems.

The subject of the research is methods of designing information architecture, client-side functionality, and server-side logic of web platforms for automated testing.

The work analyzes modern online testing systems and substantiates the necessity of creating a specialized web platform. The software product was implemented using the Python programming language, Django web framework, JavaScript, HTML5, CSS3, and Tailwind CSS. As a result, a web application with SPA architecture support, automated assessment mechanisms, user authentication, and test content administration was developed.

Field of application – educational information systems, distance learning, automated knowledge assessment, and organization of online testing in educational institutions.

Keywords: *web platform, online testing, single page application, information architecture, django, javascript, tailwind css.*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

БД – база даних

ІА – інформаційна архітектура

ПЗ – програмне забезпечення СУБД – система управління базами даних

API – Application Programming Interface

ARIA – Accessible Rich Internet Applications

CSP – Content Security Policy (політика захисту контенту)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

DOM – Document Object Model (об'єктна модель документа)

ER – Entity-Relationship (модель «сутність-зв'язок»)

FCP – First Contentful Paint (перше відмальовування контенту)

HTML – HyperText Markup Language (мова розмітки гіпертексту)

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту)

JSON – JavaScript Object Notation (текстовий формат обміну даними)

JS – JavaScript (мова програмування)

LCP – Largest Contentful Paint (відмальовування найбільшого контенту)

LMS – Learning Management System (система управління навчанням)

MPA – Multi-Page Application (багатосторінковий додаток)

ORM – Object-Relational Mapping (об'єктно-реляційне відображення)

REST – Representational State Transfer (передача стану представлення)

SEO – Search Engine Optimization (пошукова оптимізація)

SPA – Single Page Application (односторінковий додаток)

SQL – Structured Query Language (мова структурованих запитів)

UI – User Interface (інтерфейс користувача)

UML – Unified Modeling Language (уніфікована мова моделювання)

UX – User Experience (користувацький досвід)

XSS – Cross-Site Scripting (міжсайтовий скриптинг)

ВСТУП

Сучасний етап розвитку інформаційного суспільства характеризується стрімкою цифровізацією практично всіх сфер людської діяльності, серед яких особливе місце займає освітня галузь. Активне впровадження інформаційно-комунікаційних технологій у навчальний процес суттєво змінило підходи до організації навчання, контролю знань та взаємодії між учасниками освітнього середовища. Особливо актуальними ці процеси стали в умовах поширення дистанційного та змішаного навчання, коли використання електронних освітніх платформ перетворилося з допоміжного інструменту на необхідну складову сучасної освіти.

Одним із ключових елементів освітнього процесу є контроль та оцінювання рівня знань здобувачів освіти. Традиційні форми проведення тестувань та екзаменацій мають низку суттєвих недоліків: значні витрати часу на перевірку результатів, ризик суб’єктивності оцінювання, складність організації тестування великої кількості студентів та недостатню оперативність отримання статистичних даних. У зв’язку з цим особливої актуальності набувають автоматизовані вебплатформи онлайн-тестування, здатні забезпечити швидкий, об’єктивний та надійний процес контролю знань.

На сьогодні існує велика кількість систем дистанційного навчання та тестування, серед яких найбільш поширеними є Moodle, Google Forms, Kahoot, Quizizz та інші. Такі рішення активно використовуються у закладах освіти різного рівня та дозволяють організовувати онлайн-курси, перевірку знань та зберігання навчальних матеріалів. Проте аналіз існуючих платформ свідчить, що більшість із них мають певні недоліки, пов’язані насамперед із клієнтським функціоналом та інформаційною архітектурою. Частина систем характеризується складною та перевантаженою навігацією, надлишковою кількістю елементів інтерфейсу та громіздкою багатосторінковою структурою, що негативно впливає на зручність взаємодії користувача із системою. Інші платформи, навпаки, мають спрощений функціонал та не

забезпечують достатнього рівня контролю часу, безпеки та стабільності проходження тестування.

Проблема ускладнюється тим, що під час проходження онлайн-тестування користувач взаємодіє не лише зі змістом екзаменаційних завдань, а й безпосередньо з інтерфейсом вебзастосунку. У разі складної навігації, постійного перезавантаження сторінок або нестабільної роботи системи студент змушений витрачати додаткові когнітивні ресурси на взаємодію з платформою, а не на виконання тестових завдань. Це може негативно впливати на концентрацію уваги та об’єктивність оцінювання результатів навчання. Саме тому сучасні вебплатформи повинні забезпечувати мінімалістичний, швидкий та інтуїтивно зрозумілий інтерфейс, який би дозволяв користувачу максимально зосередитися на освітньому контенті.

Важливим напрямом розвитку сучасної веброзробки є використання концепції односторінкових додатків (Single Page Application, SPA), що дозволяє реалізувати динамічне оновлення інтерфейсу без постійного перезавантаження сторінок. Такий підхід забезпечує плавну взаємодію користувача із системою, високу швидкодію та можливість локального управління станом додатка. Для освітніх платформ це є особливо важливим, оскільки дозволяє мінімізувати ризик втрати відповідей у разі нестабільного інтернет-з’єднання та забезпечити безперервність процесу тестування.

Актуальність теми кваліфікаційної роботи обумовлюється необхідністю створення сучасної вебплатформи навчального тестування, яка б поєднувала надійність серверної архітектури, високу швидкодію клієнтської частини та оптимізовану інформаційну архітектуру. Особливого значення набуває забезпечення адаптивності інтерфейсу, підтримки мобільних пристроїв, мінімізації когнітивного навантаження на користувача та реалізації механізмів захисту й стабільного збереження даних під час проходження тестів.

Об’єктом дослідження є процес організації та автоматизації онлайн-тестування у сучасному цифровому освітньому середовищі.

Предметом дослідження є методи, засоби та технології проєктування вебплатформ для автоматизованого контролю знань із використанням сучасних підходів до інформаційної архітектури та клієнтського функціоналу.

Метою кваліфікаційної роботи є розроблення вебплатформи навчального тестування з оптимізованою інформаційною архітектурою та сучасним клієнтським функціоналом, що забезпечує високу швидкодію, надійність, адаптивність та зручність взаємодії користувача із системою.

Для досягнення поставленої мети у роботі необхідно було вирішити такі завдання:

– провести аналіз предметної області та сучасних систем онлайн-тестування;
– дослідити сучасні підходи до проєктування інформаційної архітектури вебзастосунків;
– виконати порівняльний аналіз існуючих платформ тестування;
– обґрунтувати вибір технологічного стеку для реалізації системи;
– спроєктувати структуру бази даних та архітектуру вебплатформи;
– реалізувати серверну та клієнтську частини вебзастосунку;
– забезпечити механізми авторизації, обробки результатів тестування та захисту даних;
– провести тестування розробленої системи та оцінити ефективність її роботи.

Наукова новизна роботи полягає у поєднанні сучасного SPA-підходу до реалізації клієнтської частини з оптимізованою інформаційною архітектурою освітнього вебзастосунку, орієнтованого на мінімізацію когнітивного навантаження користувача під час проходження тестування. Запропонований підхід дозволяє забезпечити швидку та стабільну взаємодію користувача із системою навіть в умовах нестабільного інтернет-з’єднання.

Практичне значення одержаних результатів полягає у створенні повноцінної вебплатформи навчального тестування, яка може бути використана у закладах освіти для автоматизованого контролю знань студентів та учнів. Розроблена система забезпечує зручне адміністрування

тестового контенту, підтримує адаптивний інтерфейс та може бути масштабована відповідно до потреб освітнього середовища.

У процесі виконання роботи було використано методи системного аналізу, проєктування інформаційних систем, об’єктноорієнтованого програмування, моделювання баз даних та технології сучасної веброзробки. Для реалізації програмного продукту використано мову програмування Python, вебфреймворк Django, JavaScript, Tailwind CSS та реляційну базу даних SQLite.

Структура кваліфікаційної роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі проведено аналіз предметної області та існуючих рішень. Другий розділ присвячено проєктуванню системи та вибору технологічного стеку. У третьому розділі розглянуто програмну реалізацію вебплатформи, тестування системи та оцінку ефективності її роботи.

1. ТЕОРЕТИЧНИЙ АНАЛІЗ ТА ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз предметної області та актуальність систем онлайн-тестування

Сучасний етап розвитку інформаційного суспільства характеризується глобальною цифровізацією всіх сфер людської діяльності, серед яких освітня галузь займає одне з провідних місць. Стрімкий перехід закладів вищої та загальної середньої освіти до змішаних, дистанційних та гібридних форм навчання докорінно змінив підходи до організації навчального процесу [2]. У цих умовах класичні (паперові) форми контролю знань втрачають свою ефективність через високі витрати часу на перевірку, суб'єктивність оцінювання та неможливість швидкої обробки великих масивів даних. На їхнє місце приходять автоматизовані системи онлайн-тестування, які стають невід'ємною складовою сучасного освітнього середовища.

Комп'ютерне тестування як метод педагогічного контролю має низку беззаперечних переваг: забезпечення рівних умов для всіх здобувачів освіти, висока об'єктивність результатів, можливість масштабування та миттєвий зворотний зв'язок [10]. Автоматизовані платформи дозволяють викладачам оперативно збирати статистику успішності, аналізувати прогалини у знаннях студентів та гнучко адаптувати навчальні програми.

Попри стрімкий розвиток технологій та наявність на ринку великої кількості систем управління навчанням (Learning Management Systems, LMS), практика їх застосування виявляє низку суттєвих недоліків, пов'язаних безпосередньо з інтерфейсом користувача (UI) та користувацьким досвідом (UX) [16]. Більшість наявних платформ розроблялися як універсальні монолітні рішення, що призвело до їхньої функціональної перевантаженості. Для здобувача освіти це означає взаємодію зі складною, неінтуїтивною навігацією, надмірною кількістю меню та відволікаючих візуальних елементів.

Згідно з дослідженнями у галузі вебузабіліті та ергономіки інтерфейсів, надмірне когнітивне навантаження під час взаємодії з програмним продуктом критично знижує концентрацію користувача на його основному завданні [18]. У контексті навчального тестування це означає, що студент витрачає свої інтелектуальні ресурси не на вирішення екзаменаційних завдань, а на спроби зрозуміти логіку роботи самої платформи. Крім того, класична багатосторінкова архітектура таких вебзастосунків часто призводить до втрати відповідей у разі нестабільного інтернет-з'єднання, що створює додатковий стрес під час контролю знань.

Вирішення описаних проблем лежить у площині грамотного проєктування інформаційної архітектури (IA) та оптимізації клієнтського функціоналу. Інформаційна архітектура виступає фундаментом будь-якої вебсистеми, оскільки вона визначає принципи організації, структуризації та маркування контенту [24]. Ефективна IA повинна забезпечувати максимально короткий шлях користувача (User Flow) від моменту авторизації в системі до безпосереднього початку тестування, усуваючи будь-які зайві кроки [5].

Своєю чергою, реалізація клієнтського функціоналу вебплатформи вимагає переходу від застарілих підходів до сучасних стандартів веброзробки [1]. Використання концепції односторінкових додатків (Single Page Application, SPA) дозволяє ізолювати процес тестування: інтерфейс оновлюється динамічно, без повного перезавантаження сторінки, а стан іспиту (таймер, обрані відповіді) надійно зберігається на стороні клієнта [25].

Враховуючи вищезазначене, розроблення спеціалізованої веб-платформи навчального тестування, головним акцентом якої є оптимізована інформаційна архітектура та сучасний клієнтський функціонал, є надзвичайно актуальним завданням. Створення легкого, адаптивного та відмовостійкого інтерфейсу, що базується на принципах мінімалізму та доступності, дозволить значно підвищити ефективність процесу автоматизованого оцінювання знань, знизити психологічне навантаження на користувачів та забезпечити високий рівень надійності збереження екзаменаційних даних.

1.2. Огляд та порівняльний аналіз існуючих рішень

Для формування обґрунтованих технічних вимог до розроблюваної веб-платформи необхідно провести критичний аналіз існуючих програмних рішень, які широко використовуються у сучасній освітній практиці. Ринок систем онлайн-тестування можна умовно поділити на три основні категорії: комплексні системи управління навчанням (LMS), хмарні сервіси для створення опитувань загального призначення та спеціалізовані гейміфіковані платформи інтерактивного оцінювання [7].

Аналіз доцільно проводити за такими критеріями: складність інформаційної архітектури (IA), ергономіка користувацького інтерфейсу (UI), стабільність збереження стану під час тестування (State Management) та адаптивність під мобільні пристрої.

1. Комплексні системи управління навчанням (на прикладі Moodle)

Moodle є де-факто стандартом для багатьох закладів вищої освіти завдяки своїй відкритості та масштабованості.

- *Переваги:* Платформа пропонує потужний аналітичний інструментарій, підтримує десятки типів тестових запитань (від множинного вибору до написання есе), дозволяє детально налаштовувати критерії оцінювання та гарантує високий рівень безпеки даних [2].
- *Недоліки з погляду UI/UX:* Головною проблемою Moodle є його застаріла монолітна багатосторінкова архітектура. Інтерфейс платформи перевантажений елементами управління, які не потрібні студенту під час безпосереднього проходження іспиту. Навігаційний ланцюжок від авторизації до початку тесту часто складається з 5-7 обов'язкових кроків (кліків), що є порушенням сучасних принципів інформаційної архітектури [18]. Крім того, при кожному переході до наступного запитання сторінка повністю перезавантажується, що критично впливає на стабільність іспиту при низькій швидкості

інтернет-з'єднання. Адаптивність під мобільні екрани «із коробки» часто реалізована некоректно, що вимагає розробки окремих мобільних додатків.

2. Хмарні сервіси загального призначення (на прикладі Google Forms)

Google Forms набули значної популярності завдяки швидкості розгортання та безшовній інтеграції з екосистемою Google.

- *Переваги:* Інтерфейс відзначається мінімалізмом та інтуїтивністю. Форми завантажуються миттєво, а їхнє створення не вимагає спеціальних технічних навичок.
- *Недоліки з погляду клієнтської логіки:* Платформа не є спеціалізованим інструментом для академічного тестування. Відсутній особистий кабінет студента для відстеження власного прогресу та історії спроб. Найбільш критичним недоліком є неможливість надійного контролю часу (відсутність вбудованого таймера, який би автоматично завершував сесію) та слабкий захист від повторних проходжень [5]. Інформаційна архітектура є "плоскою", що обмежує можливості організації складної логіки перевірки знань.

3. Гейміфіковані платформи інтерактивного оцінювання

Ці платформи базуються на концепції мікронавчання та використання ігрових механік (гейміфікації).

- *Переваги:* Використовують архітектуру Single Page Application (SPA), що забезпечує миттєвий відгук інтерфейсу завдяки асинхронним мережевим запитам [25]. Візуальний дизайн є сучасним, повністю адаптивним і розрахованим на утримання уваги користувача.
- *Недоліки з погляду освітніх вимог:* Попри відмінний клієнтський функціонал, такі платформи орієнтовані на розважальний формат або поточний експрес-контроль. Їхній інтерфейс фокусується на

швидкості реакції (хто швидше натисне на кнопку), а не на вдумливому вирішенні складних багаторівневих завдань, що робить їх використання недоцільним для проведення підсумкових академічних іспитів (заліків, сесій) [10].

Для систематизації отриманих результатів, порівняльні характеристики наведених систем зведені у таблицю 1.1.

Таблиця 1.1 – Порівняльний аналіз сучасних веб-платформ тестування

Платформа	**Архітектурний підхід**	**Основні переваги**	**Недоліки з погляду клієнтського функціоналу**
Google Forms	Хмарна HTML-форма	Простота розгортання, висока швидкодія, мінімалістичний дизайн.	Відсутність кабінету користувача, неможливість суворого контролю таймінгу тестування.
Kahoot / Quizizz	SPA з гейміфікацією	Адаптивність, високий рівень інтерактивності (WebSocket-взаємодія).	Обмеження для серйозного оцінювання, орієнтація на швидкість, а не на глибину знань.

Проведений аналіз дозволяє зробити висновок, що жодна з існуючих популярних платформ не задовольняє повною мірою вимоги до сучасного академічного тестування. Існує об'єктивна потреба у створенні спеціалізованого програмного продукту, який би поєднував надійність

бекенду (як у Moodle) з мінімалістичним, стійким до збоїв та швидким клієнтським функціоналом (як у сучасних SPA-додатках).

Ця платформа повинна мати оптимізовану інформаційну архітектуру (чіткий розподіл на публічну зону, кабінет та ізольоване середовище тестування) та забезпечувати максимальну доступність і стабільність роботи користувача під час іспиту.

1.3. Сучасні підходи до проєктування інформаційної архітектури та клієнтського функціоналу

Вирішення проблем, виявлених під час аналізу наявних платформ онлайн-тестування, вимагає відмови від застарілих патернів розробки на користь сучасних підходів до проєктування інтерфейсів. Фундаментом будь-якого успішного вебзастосунку є його інформаційна архітектура (ІА) — процес структурного проєктування інформаційного простору, спрямований на полегшення інтуїтивного доступу до контенту [24].

У контексті освітніх платформ сучасні підходи до ІА диктують необхідність створення "відволікаюче-вільних" (distraction-free) середовищ. Це означає, що під час безпосереднього проходження тесту всі зайві елементи навігації (головні меню, футери, бічні панелі) повинні приховуватися, залишаючи у фокусі уваги користувача лише поточне запитання, варіанти відповідей та індикатори прогресу (таймер) [5, 16]. Такий підхід суттєво знижує когнітивне навантаження на студента.

Основним технологічним трендом у реалізації клієнтського функціоналу є перехід від класичної архітектури багатосторінкових додатків (Multi-Page Application, MPA) до односторінкових застосунків (Single Page Application, SPA) [25]. У традиційній MPA-моделі кожна дія користувача (наприклад, перехід до наступного питання тесту) ініціює HTTP-запит до сервера, який генерує та повертає повністю нову HTML-сторінку [14]. Це призводить до високих затримок (latency) та переривання користувацького досвіду.

Архітектура SPA вирішує цю проблему шляхом завантаження єдиного HTML-документа на початку сесії. Усі подальші оновлення інтерфейсу відбуваються динамічно за допомогою мови програмування JavaScript [1, 13]. Сервер у цій моделі виступає лише як постачальник даних (у форматі JSON). Для систем тестування застосування SPA-підходу є критично важливим, оскільки воно дозволяє:

- здійснювати миттєву навігацію між питаннями без "блимання" екрана;
- керувати станом додатку (State Management) безпосередньо у пам'яті браузера користувача;
- забезпечувати відмовостійкість: у разі короткочасної втрати інтернет-з'єднання студент може продовжувати відповідати на питання, а синхронізація із сервером відбудеться у фоновому режимі після відновлення зв'язку.

Окрім архітектурних змін, сучасна веброзробка базується на принципах «Mobile-First» (проєктування спочатку для мобільних пристроїв) та утилітарному підході до стилізації [22]. Замість написання громіздких каскадних таблиць стилів (CSS), розробники дедалі частіше використовують утилітарні фреймворки (наприклад, Tailwind CSS), які дозволяють конструювати адаптивні інтерфейси безпосередньо в HTML-розмітці за допомогою атомарних класів [20]. Це пришвидшує процес розробки та гарантує консистентність дизайну на будь-яких діагоналях екранів.

Окремим і надзвичайно важливим підходом до проєктування сучасних освітніх платформ є забезпечення інклюзивності та безбар'єрності інформаційного простору. Інтерфейси повинні розроблятися у суворій відповідності до міжнародних стандартів Web Content Accessibility Guidelines (WCAG) 2.1 [27] та гармонізованих з ними національних стандартів України [3]. Клієнтський функціонал повинен забезпечувати контрастність візуальних елементів, повну підтримку навігації за допомогою клавіатури та семантичну

HTML-розмітку, що дозволяє користувачам з особливими освітніми потребами безперешкодно проходити тестування.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи проведено детальний аналіз предметної області та обґрунтовано актуальність розроблення веб-платформи для навчального тестування. Встановлено, що цифровізація освіти вимагає впровадження надійних, об'єктивних та швидких засобів автоматизованого контролю знань.

Порівняльний аналіз популярних наявних рішень (Moodle, Google Forms, Kahoot) дозволив виявити їхні ключові архітектурні та ергономічні недоліки. Більшість систем є або функціонально перевантаженими (складна навігація, багатосторінкова структура), або не пристосованими для суворого академічного тестування через відсутність надійного клієнтського контролю часу та стану іспиту.

Доведено, що для вирішення виявлених проблем при проєктуванні нової системи доцільно застосовувати сучасні підходи. Основою розробки має стати концепція односторінкових додатків (Single Page Application, SPA), що гарантує високу швидкодію та захист від втрати даних користувача. Оптимізація інформаційної архітектури, застосування стратегії «Mobile-First» та дотримання стандартів доступності вебконтенту (WCAG 2.1) визначені як обов'язкові умови для створення якісного, інклюзивного та зручного освітнього продукту. Сформовані теоретичні вимоги стануть основою для етапу проєктування системи у наступному розділі.

2. ПРОЄКТУВАННЯ СИСТЕМИ ТА ВИБІР ТЕХНОЛОГІЧНОГО СТЕКУ

2.1. Вибір та обґрунтування технологічного стеку та середовища

Вибір технологічного стеку є фундаментальним етапом проєктування програмного забезпечення, що безпосередньо визначає його майбутню продуктивність, масштабованість та зручність підтримки коду [1]. Оскільки головною метою даної кваліфікаційної роботи є оптимізація клієнтського функціоналу, особливу увагу було приділено інструментам розробки користувацького інтерфейсу. Для реалізації динамічної взаємодії на стороні клієнта було обрано мову програмування JavaScript (стандарту ES6+) у поєднанні з нативним DOM API [15, 21]. Такий підхід дозволив уникнути надмірної перевантаженості коду важкими фреймворками та реалізувати архітектуру Single Page Application (SPA) для модуля тестування. Використання асинхронних мережевих запитів (Fetch API) забезпечило можливість фонового завантаження екзаменаційних завдань та локального керування станом (State Management), що гарантує безперебійну роботу таймера та збереження відповідей без постійного перезавантаження сторінок [1, 13].

За візуальне оформлення та адаптивність інтерфейсу відповідає утилітарний CSS-фреймворк Tailwind CSS [26]. На відміну від класичних компонентних бібліотек, він надає набір низькорівневих атомарних класів, що дозволяє формувати унікальний дизайн безпосередньо у HTML-розмітці, мінімізуючи обсяг кінцевих файлів стилів [20]. Це не лише пришвидшує процес конструювання інтерфейсів за принципом Mobile-First, але й гарантує дотримання необхідного рівня візуального контрасту відповідно до міжнародних стандартів доступності WCAG 2.1.

Попри фокус на клієнтській частині, повноцінне функціонування веб-платформи вимагає надійної серверної архітектури. Для її реалізації обрано мову Python та високорівневий вебфреймворк Django [9, 12]. Використання

архітектурного патерну MVT (Model-View-Template) ідеально відповідає поставленим завданням, а вбудована ORM-система (Object-Relational Mapping) забезпечує безпечну та ефективну взаємодію з реляційною базою даних. У якості останньої для етапу розробки використовується компактна СУБД SQLite, яка завдяки абстракціям Django може бути легко замінена на промислове рішення (наприклад, PostgreSQL) під час розгортання проєкту на робочому сервері [8]. Крім того, Django надає захист від поширених вразливостей та має готову панель адміністрування, що є критично важливим для зручного управління навчальним контентом.

Важливою складовою сучасного середовища розробки є забезпечення надійності збереження вихідного коду та гнучкості його модифікації. Для вирішення цього завдання у проєкті застосовано розподілену систему керування версіями Git. Використання систем контролю версій є галузевим стандартом, який дозволяє безпечно ізолювати процес створення та тестування нового клієнтського функціоналу в окремі гілки (branches) без ризику пошкодити стабільну версію системи. Це також створює необхідні передумови для подальшого розгортання веб-платформи на хмарних хостингах із використанням практик безперервної інтеграції (CI/CD), що підтверджує високий рівень масштабованості та професійний підхід до організації обраного технологічного стеку.

Усі етапи написання програмного коду, налаштування серверної логіки та верстки інтерфейсів виконувалися в інтегрованому середовищі розробки Visual Studio Code (VS Code). Цей редактор забезпечує широкі можливості для роботи з мовою Python та віртуальними середовищами, а також підтримує інструменти автоматичного підсвічування синтаксису Tailwind CSS. Загальна ієрархія проєкту розроблена відповідно до стандартів Django та включає конфігураційні файли, застосунок тестування, шаблони та статичні ресурси. Організацію директорій та файлів розробленого веб-застосунку в середовищі VS Code наведено на рисунку 2.1.

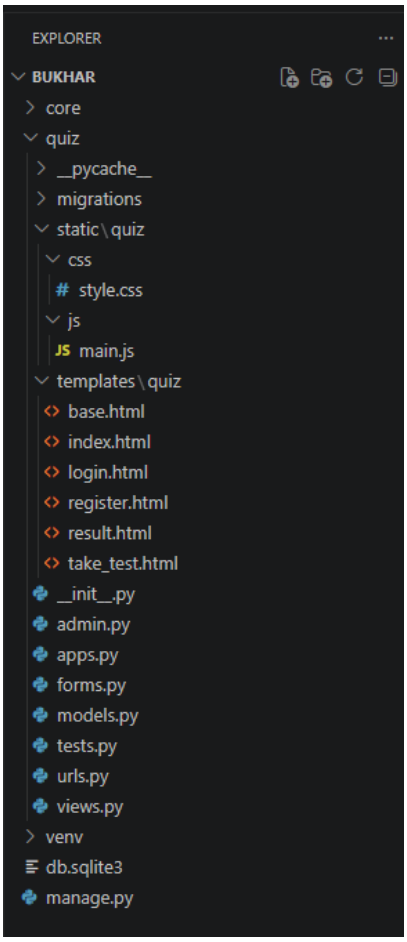

Рисунок 2.1 – Структура файлів та директорій розробленого проєкту у середовищі VS Code

2.2. Проєктування бази даних та логічної архітектури

Основою функціонування будь-якого інформаційного вебзастосунку є надійна та оптимізована структура зберігання даних. Для розроблюваної платформи навчального тестування було спроєктовано реляційну базу даних. Вибір реляційної моделі обґрунтовується наявністю чітких ієрархічних та логічних зв'язків між сутностями предметної області (наприклад, один тест містить багато питань, одне питання має кілька варіантів відповідей, а кожен користувач має власну історію результатів).

Процес проєктування бази даних здійснювався із суворим дотриманням принципів нормалізації, що дозволило привести структуру до третьої нормальної форми (3НФ). Практичне застосування цього підходу полягає у декомпозиції даних для уникнення їхньої надмірності та запобігання

аномаліям оновлення. Наприклад, виокремлення варіантів відповідей у самостійну таблицю Answer замість їх зберігання у вигляді масиву безпосередньо в таблиці Question гарантує атомарність інформації. Це не лише спрощує алгоритм автоматизованої перевірки правильних відповідей на стороні сервера, але й дозволяє гнучко масштабувати систему в майбутньому (зокрема, легко реалізувати питання з кількома правильними варіантами) без необхідності змінювати базову логіку [8].

Окрему увагу під час проєктування приділено забезпеченню референціальної цілісності даних на рівні зв'язків між таблицями. Взаємодія між сутностями реалізується засобами абстракції Django ORM із застосуванням стратегії каскадного видалення (ON DELETE CASCADE). Це означає, що у разі видалення певного тесту (Quiz) адміністратором системи, база даних автоматично знищить усі пов'язані з ним питання, варіанти відповідей та результати проходження. Такий механізм повністю унеможливлює появу "осиротілих" (orphaned) записів, які могли б спричинити помилки інтерфейсу при спробі клієнтської частини завантажити неіснуючий контент. Крім того, для оптимізації швидкодії веб-платформи всі поля зовнішніх ключів (Foreign Keys) автоматично індексуються, забезпечуючи мінімальний час відгуку бекенду під час формування JSON-пакетів для клієнта [9].

Логічна архітектура розробленої бази даних складається з шести основних сутностей:

1. **Сутність User (Користувач).** Використовується стандартна та добре захищена модель користувача фреймворку Django. Вона забезпечує надійне зберігання облікових даних (логін, хешований пароль), управління сесіями авторизації та розмежування прав доступу між звичайними студентами та адміністраторами. Використання вбудованої моделі також спрощує можливу інтеграцію з іншими сервісами автентифікації та дозволяє легко розширювати профіль додатковими атрибутами в майбутньому.

2. **Сутність Category (Категорія).** Виконує роль класифікатора для групування тестів за навчальними дисциплінами. Містить поля: унікальний ідентифікатор (id), назва (name) та текстовий опис (description). Має зв'язок типу «один-до-багатьох» (1:N) із сутністю Quiz. Така структуризація значно покращує інформаційну архітектуру, дозволяючи студентам швидко фільтрувати екзаменаційні матеріали в особистому кабінеті.
3. **Сутність Quiz (Тест).** Головна сутність навчального модуля. Окрім стандартних полів (назва, опис), містить критично важливе для логіки клієнтської частини (фронтенду) поле time_limit — обмеження часу на проходження іспиту. Це значення передається до браузера для ініціалізації JavaScript-таймера. Крім того, наявність цієї сутності дає змогу викладачам гнучко керувати параметрами іспиту, налаштовуючи їх відповідно до складності конкретної дисципліни.
4. **Сутність Question (Запитання).** Зберігає текстове формулювання екзаменаційного завдання. Зв'язана із сутністю Quiz (відношення 1:N). Для ускладнення процесу списування під час іспиту, вибірка запитань із цієї таблиці відбувається у випадковому порядку. Ізоляція запитань в окрему модель також створює міцний фундамент для подальшого масштабування, наприклад, для інтеграції мультимедійного контенту безпосередньо у завдання.
5. **Сутність Answer (Відповідь).** Зберігає варіанти відповідей для кожного запитання. Містить логічне (булеве) поле is_correct, яке визначає еталонну відповідь та використовується контролером для підрахунку балів. Цей атрибут суворо ізолюється на рівні бекенду під час генерації JSON-пакетів, що повністю виключає ризик витоку правильних відповідей на клієнтську сторону.
6. **Сутність Result (Результат).** Агрегує статистичні дані після завершення тестування. Фіксує зв'язок між конкретним користувачем (User) та пройденим тестом (Quiz), утворюючи відношення «багато-

до-багатьох» (M:N) між цими сутностями. Зберігає фінальний бал у відсотках (score) та точну дату завершення спроби (completed_at). Зібрані таким чином метрики слугують надійною базою для побудови модуля аналітики та візуалізації академічної успішності.

Для наочного представлення зв'язків між розробленими сутностями та їхніми атрибутами було побудовано ER-діаграму (Entity-Relationship diagram), яка відображає концептуальну модель бази даних розробленої системи (рис. 2.2). Її візуалізація дозволяє комплексно оцінити нормалізацію даних та загальну логіку взаємодії всіх архітектурних компонентів платформи.

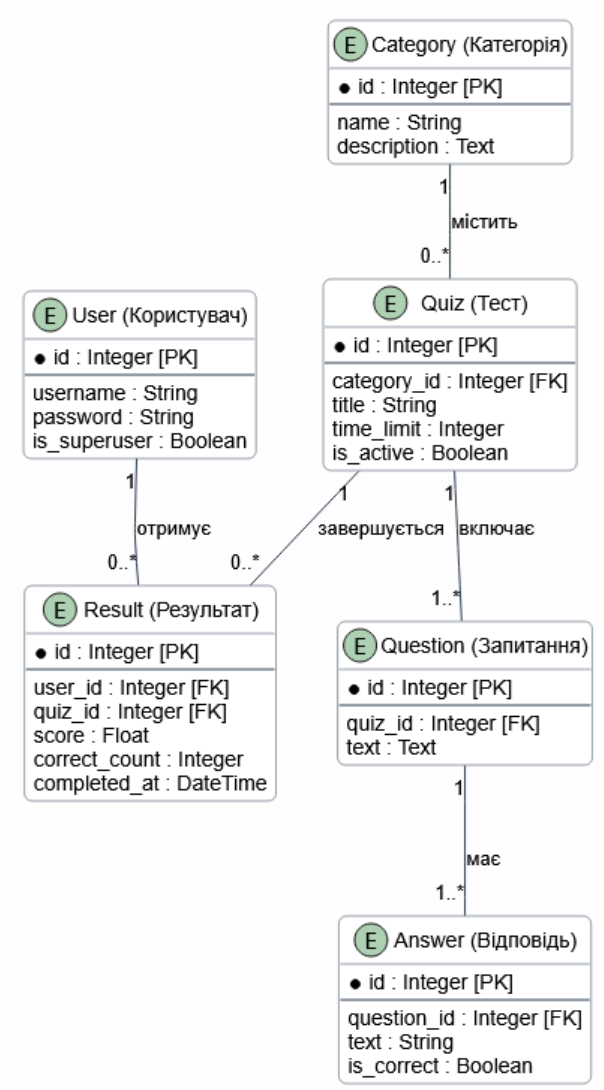

Рисунок 2.2 – ER-діаграма логічної структури реляційної бази даних веб-платформи

2.3. Проєктування інформаційної архітектури та клієнтської взаємодії

Інформаційна архітектура (ІА) є структурним фундаментом вебплатформи, що визначає логіку організації контенту, принципи навігації та механізми взаємодії користувача з інтерфейсом. Проєктування ІА для освітньої системи тестування базувалося на ітеративному підході, що включав створення низькодеталізованих вайрфреймів (wireframes) для всіх ключових екранів. Основною метою було досягнення відповідності «правилу трьох кліків», згідно з яким користувач повинен мати змогу розпочати тестування не більше ніж за три переходи від моменту входу на платформу [18]. Цей підхід мінімізує когнітивне навантаження, забезпечуючи швидкий доступ до екзаменаційних матеріалів та створення інтуїтивно зрозумілого середовища, де фокус уваги студента спрямований виключно на контент тесту [24]. Таке усунення відволікаючих факторів є критично важливим для забезпечення високого рівня концентрації під час складання іспитів.

Логічну структуру вебзастосунку розділено на три функціональні зони з різним рівнем доступу та навігаційними патернами:

1. **Публічна зона (Public Area).** Включає сторінки реєстрації та авторизації, архітектура яких фокусується на мінімізації бар'єрів при вході. Тут реалізовано дворівневу валідацію: клієнтську (миттєва перевірка за допомогою JavaScript на формат email та довжину пароля) та серверну (перевірка унікальності логіна через Django ORM). Це гарантує цілісність даних ще до моменту їх потрапляння до бази даних та дозволяє уникнути повторних запитів до сервера у разі помилок введення. Як наслідок, такий підхід не лише оптимізує навантаження на серверну частину, але й забезпечує миттєвий зворотний зв'язок для користувача.
2. **Приватна зона (Dashboard).** Особистий кабінет студента, доступ до якого відкривається виключно після успішної автентифікації. Інформаційний простір кабінету розділено на компонентні блоки:

перелік доступних для проходження тестів та модуль аналітики. Важливою частиною ІА тут є візуалізація прогресу: статус кожного тесту («Доступний» або «Завершений») та результати попередніх спроб виводяться з використанням умовної стилізації, що дозволяє користувачу миттєво оцінити свій академічний статус без додаткового аналізу тексту [5]. Продумана візуальна ієрархія інформації в цьому модулі значно підвищує загальну ергономіку системи та покращує користувацький досвід.

3. **Ізольоване середовище тестування (Test Environment).** Найскладніший модуль платформи, побудований як SPA. Проєктування цього середовища вимагало ретельного опрацювання моделі станів (State Machine). Процес проходження тесту моделюється через чотири стани: ініціалізація (завантаження JSON-пакета питань та старт таймера); активний режим (взаємодія користувача з інтерфейсом); відправка (асинхронна передача даних через Fetch API); завершення (відображення фінального звіту). Чітке розмежування цих станів гарантує синхронізацію даних між клієнтом і сервером та унеможливлює виникнення конфліктів під час збереження результатів.

Для клієнтської взаємодії (User Flow) було обрано лінійний сценарій: авторизація, вибір тесту, іспит, отримання результату. Під час виконання тесту інтерфейс стає "distraction-free": глобальна навігація приховується, залишаючи лише активне питання, таймер та навігаційну матрицю питань. Подібна ізоляція інтерфейсу максимально наближає цифрове тестування до строгих умов традиційного академічного контролю знань.

Така логіка взаємодії підкріплюється механізмами клієнтської відмовостійкості. Зокрема, у разі обриву з'єднання модуль не ініціює перезавантаження, а зберігає стан userAnswers у локальній пам'яті браузера, дозволяючи продовжити роботу. Додатково, клієнтський таймер примусово активує функцію submitQuiz() при досягненні нуля, унеможливлюючи спроби

маніпуляцій з часом. Впровадження таких превентивних механізмів є запорукою збереження академічної доброчесності та забезпечення рівних умов для всіх здобувачів освіти.

Для формалізованого опису функціональних вимог та моделювання поведінки було розроблено UML-діаграму прецедентів (рис. 2.3), яка наочно демонструє, які саме операції ініціює користувач та які внутрішні процеси вони активують. Це підтверджує, що проєкт побудований на професійних інженерних методах моделювання, а не є випадковим набором сторінок. Завдяки такій формалізації забезпечується прозорість архітектурних рішень та простежуваність вимог на кожному етапі життєвого циклу розробки.

Рисунок 2.3 – UML-діаграма прецедентів (Use Case) взаємодії студента з веб-платформою

На додаток до діаграми прецедентів, для проєктування було враховано сценарії обробки непередбачуваних ситуацій: відсутність інтернету та спроби некоректного завершення сесії. Такий багаторівневий підхід до проєктування ІА та взаємодії гарантує, що розроблена система не лише відповідає технічним вимогам, але й забезпечує високий рівень надійності, що є критично важливим для освітніх інформаційних систем.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи проведено детальний етап системного проєктування вебплатформи для організації навчального тестування. Обґрунтовано доцільність вибору технологічного стеку, що базується на мові Python та фреймворку Django для серверної частини, а також поєднанні Vanilla JavaScript та утилітарного CSS-фреймворку Tailwind CSS для реалізації фронтенду. Таке поєднання забезпечило баланс між швидкістю розробки, високою продуктивністю та зручністю подальшого масштабування системи [1, 9, 20].

У межах проєктування логічної архітектури спроєктовано реляційну структуру бази даних, яку було нормалізовано до третьої нормальної форми (3НФ). Це дозволило усунути надмірність даних, забезпечити їхню цілісність та уникнути аномалій оновлення. Розроблена модель з шести ключових сутностей (User, Category, Quiz, Question, Answer, Result) повністю покриває потреби зберігання екзаменаційного контенту, управління правами доступу та фіксації результатів проходження тестів [8, 9].

Особливу увагу приділено проєктуванню інформаційної архітектури (IA) та клієнтської взаємодії. Впроваджено ієрархічний поділ системи на три функціональні зони: публічну, приватну (особистий кабінет) та ізольоване середовище тестування, що реалізовано за концепцією Single Page Application (SPA). Це дозволило досягти високих показників швидкодії та надійності, оскільки ключові операції виконуються на стороні клієнта без постійних звернень до сервера [25].

Побудовані UML-діаграми (ER-діаграма бази даних та діаграма прецедентів) підтвердили, що розроблена логіка взаємодії є раціональною, захищеною від несанкціонованих дій та повністю відповідає сучасним інженерним вимогам до освітніх інформаційних систем. Створена проєктна документація та обрані архітектурні патерни утворюють цілісну технічну базу, яка є фундаментом для програмної реалізації системи, що розглядається у наступному розділі роботи.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

3.1. Розробка серверної логіки та моделювання даних

Програмна реалізація серверної частини вебплатформи базується на використанні архітектурного патерну MVT (Model-View-Template) фреймворку Django, що забезпечує чітке розмежування шарів зберігання, обробки та представлення даних [9]. Використання саме такого підходу дозволило організувати структуру програмного забезпечення відповідно до принципів модульності та масштабованості, що є особливо важливим для інформаційних систем освітнього призначення. Архітектура MVT забезпечує ізоляцію окремих функціональних компонентів системи, завдяки чому зміни в одному модулі не спричиняють необхідності суттєвого перепроєктування інших частин програмного забезпечення. Це значно спрощує подальшу підтримку, тестування та модернізацію системи.

Першим етапом реалізації серверної частини стало створення рівня моделей (models.py), у якому була реалізована спроєктована реляційна структура бази даних. Завдяки використанню Django ORM кожна сутність предметної області отримала своє представлення у вигляді окремого Python-класу. Такий підхід дозволив уникнути необхідності безпосереднього написання SQL-запитів, оскільки ORM автоматично транслює операції над об’єктами у відповідні запити до бази даних. Крім спрощення розробки, це забезпечує додатковий рівень безпеки, оскільки механізми ORM мінімізують ризик SQL-ін’єкцій та помилок при роботі з даними.

Особлива увага була приділена побудові зв’язків між сутностями системи. Модель Quiz виступає центральним елементом структури, оскільки саме вона визначає логіку проведення тестування та взаємодію між питаннями, варіантами відповідей і результатами користувачів. Для кожного тесту реалізовано механізм зв’язку «один-до-багатьох» із моделлю питань, а для кожного питання — зв’язок із набором варіантів відповідей. Подібна

ієрархічна структура забезпечує гнучкість при створенні тестових наборів та дозволяє формувати необмежену кількість комбінацій завдань без дублювання даних.

З метою забезпечення динамічної поведінки системи тестування було реалізовано механізм випадкового перемішування питань. Для цього використано метод objects.order_by('?'), який виконує випадкову вибірку записів безпосередньо на рівні бази даних. Реалізація такого підходу дозволяє автоматично змінювати порядок питань для кожного користувача, що суттєво знижує ризики академічної недоброчесності під час проходження тестування. Крім того, випадкове формування послідовності завдань унеможливлює створення фіксованих шаблонів відповідей між студентами, що є важливим аспектом забезпечення об’єктивності оцінювання.

Окремим напрямом реалізації стала організація механізмів перевірки та очищення даних. Незважаючи на наявність стандартних інструментів валідації Django, у кожній моделі було перевизначено метод save(), який виконує додаткову обробку текстового контенту перед його записом до бази даних. Основною метою цього етапу є забезпечення захисту системи від потенційно небезпечного HTML- та JavaScript-коду. Під час збереження система автоматично здійснює Sanitization — очищення вхідних даних від шкідливих тегів та скриптів. Такий механізм дозволяє ефективно протидіяти XSS-атакам (Cross-Site Scripting), які можуть виникати при спробі вставки стороннього коду у текст питань або відповідей.

Важливість такого захисту пояснюється тим, що система функціонує у вебсередовищі, де введені користувачем дані можуть бути відображені в браузері інших користувачів. У випадку відсутності механізмів очищення це створює ризик виконання стороннього коду на клієнтській стороні. Саме тому реалізація превентивної фільтрації даних ще на етапі запису до бази є важливим компонентом загальної архітектури безпеки системи.

На рівні контролерів (views.py) реалізовано основну бізнес-логіку вебплатформи. Контролери виконують роль проміжного шару між

клієнтською частиною та базою даних, забезпечуючи обробку HTTP-запитів, підготовку даних та формування відповідей. Центральним компонентом серверної логіки став механізм управління процесом тестування, який відповідає за формування тесту, передачу питань користувачу, перевірку відповідей та обчислення результатів.

Взаємодія між клієнтом та сервером була організована за принципом асинхронного обміну даними та поділена на два логічно незалежні етапи. Перший етап передбачає серіалізацію контенту при виконанні GET-запиту. Контролер take_quiz виконує вибірку всіх необхідних даних тесту з бази даних, після чого формує JSON-пакет для передачі на клієнтську сторону. При цьому особлива увага приділяється інформаційній безпеці. Для уникнення витоку еталонних відповідей логіка серіалізації була модифікована таким чином, щоб поля, які містять інформацію про правильні варіанти відповідей або прапори is_correct, автоматично виключались із пакета даних.

У результаті клієнтська частина отримує лише той обсяг інформації, який є необхідним для коректного відображення тесту в інтерфейсі користувача. Всі критично важливі дані залишаються виключно на сервері, що унеможливлює їхнє отримання через інструменти браузера або зміну JavaScript-коду на стороні користувача. Такий підхід реалізує принцип «мінімально необхідного доступу» та значно підвищує загальний рівень безпеки системи.

Другий етап роботи серверної логіки пов’язаний з обробкою результатів тестування та перевіркою відповідей студента. Після завершення проходження тесту клієнтська частина надсилає POST-запит, який містить ідентифікатори вибраних користувачем варіантів відповідей. Контролер виконує ітеративний обхід отриманого масиву даних та здійснює порівняння вибраних значень із еталонними записами, що зберігаються у базі даних.

Процес перевірки реалізовано повністю на серверній стороні, що виключає можливість маніпуляцій із результатами через модифікацію клієнтського коду. Для кожного питання система визначає правильність

відповіді та накопичує загальну кількість набраних балів. Після завершення обчислень виконується розрахунок відсоткового результату та формування запису про проходження тесту.

Важливим аспектом реалізації стало використання механізму transaction.atomic. Всі операції перевірки та збереження результатів виконуються в межах однієї атомарної транзакції. Це означає, що у випадку виникнення будь-якої помилки — наприклад, втрати мережевого з'єднання, аварійного завершення процесу або помилки запису — база даних не збереже частково оброблений результат. Якщо хоча б один етап транзакції завершиться невдало, всі зміни автоматично скасовуються. Лише після повного успішного виконання перевірки створюється запис у таблиці Result. Такий підхід забезпечує цілісність даних та запобігає появі некоректних або неповних записів у системі.

Значна увага була приділена також питанням контролю доступу та ізоляції бізнес-логіки. Для обмеження доступу до функціоналу тестування використано механізм декораторів @login_required. Його застосування дозволяє гарантувати, що проходження тестів доступне виключно авторизованим користувачам системи. У випадку спроби переходу до сторінки тестування без автентифікації користувач автоматично перенаправляється на сторінку входу.

Крім базового механізму авторизації, у серверній логіці реалізовано додатковий контроль часових обмежень проходження тестування. Під час старту тесту сервер фіксує точний час початку сесії, після чого цей показник використовується для перевірки тривалості проходження тесту. При отриманні POST-запиту система порівнює поточний час із встановленим параметром time_limit. Якщо виявляється суттєве перевищення допустимого часу, результати автоматично анулюються. Реалізація такої перевірки саме на сервері дозволяє уникнути маніпуляцій із клієнтським таймером або локальними налаштуваннями браузера.

Комунікація між серверною та клієнтською частинами реалізована за принципами REST-подібного інтерфейсу. Сервер у даній архітектурі виступає єдиним джерелом достовірних даних (Single Source of Truth). Саме серверна частина відповідає за обчислення результатів, перевірку правильності відповідей, визначення рейтингових показників та збереження інформації у базі даних. Клієнтська сторона виконує переважно функцію відображення даних та взаємодії з користувачем через графічний інтерфейс.

Такий розподіл функціональності дозволяє суттєво знизити ризики втручання користувача у процес оцінювання, оскільки жодні критичні обчислення не виконуються на стороні браузера. Навіть у випадку модифікації клієнтського JavaScript-коду користувач не має можливості вплинути на алгоритми перевірки результатів, які повністю ізольовані на бекенді. У сукупності це забезпечує високу надійність системи, стабільність роботи та відповідність сучасним вимогам до інформаційної безпеки освітніх вебплатформ [12].

3.2. Розробка клієнтського функціоналу та користувацького інтерфейсу

Клієнтська частина вебплатформи реалізована як динамічний односторінковий додаток (Single Page Application, SPA), що забезпечує високий рівень інтерактивності та безшовну взаємодію з користувачем [25]. Архітектура інтерфейсу базується на принципі декомпозиції, де кожна логічна одиниця — модуль авторизації, панель керування тестами та середовище іспиту — розглядається як незалежний функціональний блок. Такий підхід значно спрощує підтримку та масштабування системи, оскільки кожен компонент може модифікуватися окремо без впливу на інші елементи інтерфейсу. Крім того, модульна структура дозволяє швидше інтегрувати новий функціонал та підвищує зручність подальшого тестування клієнтської частини.

Для досягнення високої швидкодії при переході між запитаннями тесту було застосовано підхід із кешуванням даних на стороні клієнта: пакет запитань завантажується з бекенду одноразово під час ініціалізації сесії (quiz_data_json), після чого подальша навігація здійснюється без звернень до сервера, що мінімізує мережеві затримки [1, 13]. Реалізація такого механізму дозволяє суттєво зменшити навантаження на серверну частину та забезпечує стабільну роботу інтерфейсу навіть за умов нестабільного інтернет-з’єднання. У результаті користувач отримує швидкий доступ до контенту тесту без додаткових пауз або повторного завантаження сторінки.

Центральним елементом реалізації SPA-рушія є модуль управління станом, який відстежує поточний індекс запитання та динамічно оновлює об'єкт відповідей (userAnswers) у оперативній пам'яті браузера. Такий підхід дозволяє системі зберігати поточний прогрес користувача протягом усієї сесії тестування та забезпечує швидкий доступ до вже введених відповідей. Додатково це створює основу для реалізації механізмів автозбереження та відновлення стану у випадку тимчасових технічних збоїв.

Програмний інтерфейс реалізований через динамічну модифікацію DOM-дерева: функція renderQuestion() забезпечує повний контроль над життєвим циклом відображення контенту, очищуючи контейнер від попереднього питання та генеруючи нову HTML-розмітку для поточного. Такий підхід усуває потребу у повному оновленні сторінки, що характерно для класичних багатосторінкових додатків, і забезпечує ефект "миттєвого відгуку" інтерфейсу, що є критично важливим для утримання уваги студента під час іспиту. Окрім цього, динамічне оновлення DOM дозволяє знизити кількість зайвих HTTP-запитів та підвищити загальну продуктивність клієнтської частини вебплатформи.

Для наочного відображення того, як саме SPA-рушій керує станом системи та взаємодіє з даними, було розроблено діаграму життєвого циклу стану (State Machine) (рис. 3.1). Вона демонструє переходи між ключовими етапами: від ініціалізації запитань до фінальної відправки результатів на

сервер. Побудова такої моделі дозволила формалізувати логіку переходів між окремими станами системи та забезпечити контроль коректності виконання сценаріїв взаємодії користувача з платформою. Це особливо важливо для систем дистанційного тестування, де необхідно гарантувати стабільність роботи навіть при великій кількості одночасних користувачів.

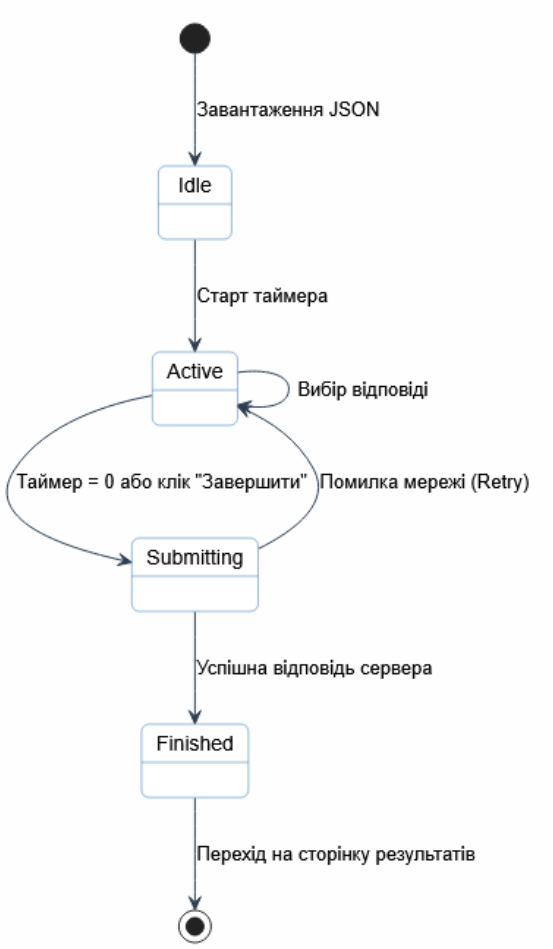

Рисунок 3.1 – Діаграма станів (State Machine) SPA-рушія модуля тестування

Ця модель унеможливлює втрату даних студента у разі мережевих перебоїв та забезпечує синхронізацію клієнтського таймера з логікою підрахунку балів на бекенді. Додатково механізм синхронізації дозволяє контролювати часові обмеження проходження тесту та мінімізує ризики маніпуляцій із локальними налаштуваннями браузера або системного часу користувача. У комплексі це забезпечує стабільність функціонування SPA-

додатка, підвищує зручність використання вебплатформи та сприяє забезпеченню об’єктивності процесу оцінювання.

Окремої уваги заслуговує система клієнтського таймера, побудована на базі API setInterval. Механізм таймера розроблений із врахуванням захисту від маніпуляцій з боку користувача: JS-функція працює у фоновому режимі, постійно перевіряючи залишок часу timeLeft і коректно обробляючи випадки завершення відліку. При досягненні нуля програмний код автоматично ініціює функцію submitQuiz(), яка блокує можливість вибору відповідей та ініціює асинхронну відправку даних на сервер через Fetch API. Це гарантує, що жоден користувач не зможе продовжити роботу після завершення встановленого ліміту часу, навіть якщо він спробує маніпулювати налаштуваннями браузера.

Візуальна складова інтерфейсу розроблена із використанням утилітарного CSS-фреймворку Tailwind CSS [26]. Адаптивність системи забезпечується гнучкими сітками (flexbox, grid), що автоматично перелаштовують структуру блоків залежно від роздільної здатності екрана — від мобільних смартфонів до стаціонарних моніторів [20]. Такий підхід підтверджує відповідність принципам Mobile-First [22]. Весь інтерфейс був спроєктований як "distraction-free" середовище: під час проходження іспиту всі елементи глобальної навігації приховуються, що дозволяє користувачу зосередитися виключно на контенті тесту.

Валідація даних на клієнтській стороні реалізована через дворівневу систему перевірок. По-перше, здійснюється синтаксичний контроль за допомогою JavaScript, що виключає відправку неповних або некоректних форм на бекенд (наприклад, перевірка чи вибрана хоча б одна відповідь перед переходом далі). По-друге, реалізовано механізм "debounce" для усунення помилок, спричинених випадковими подвійними кліками по кнопках, що запобігає відправці дублюючих POST-запитів.

Важливим компонентом є також відмовостійкість клієнтської логіки. У разі виникнення мережевих перешкод під час проходження тесту, SPA-рушій переходить у стан "очікування" та виводить неблокуюче повідомлення,

зберігаючи при цьому всі поточні відповіді користувача в об'єкті userAnswers. Це дозволяє уникнути катастрофічної втрати прогресу, оскільки при відновленні з'єднання система готова повторити спробу відправки без втрати даних, що вкрай важливо для забезпечення академічної справедливості [1, 25].

Завершальним етапом реалізації фронтенду була перевірка семантичної коректності HTML-розмітки. Використання правильних семантичних тегів (<main>, <section>, <nav>) разом із коректною структурою заголовків та ARIA-атрибутами забезпечує повну сумісність із програмами для зчитування з екрана (screen readers), що ставить розроблену систему в один ряд з кращими інклюзивними освітніми платформами [27].

3.3. Тестування системи та оцінка ефективності

Процес тестування програмного забезпечення є фінальним етапом розробки, метою якого є перевірка відповідності реалізованої системи функціональним вимогам, оцінка стабільності роботи в умовах навантаження та верифікація якості програмного коду. Саме етап тестування дозволяє визначити рівень готовності програмного продукту до практичного використання та виявити потенційні недоліки, що можуть вплинути на стабільність роботи системи у реальних умовах експлуатації. Для вебплатформи автоматизованого тестування цей етап є особливо важливим, оскільки система повинна гарантувати безперервність роботи, достовірність результатів оцінювання та стійкість до можливих помилок користувача або технічних збоїв.

Тестування проводилося шляхом виконання ряду сценаріїв, що охоплюють як позитивні, так і негативні кейси використання вебплатформи. Перевірка здійснювалася як на рівні серверної логіки, так і на рівні клієнтського інтерфейсу. Особлива увага приділялася перевірці механізмів автентифікації, коректності обробки відповідей, функціонуванню таймера, роботі SPA-архітектури та збереженню результатів у базі даних. Окремо було протестовано поведінку системи у випадку некоректного введення даних,

втрати мережевого з’єднання та перевищення допустимого часу проходження тестування.

Для систематизації результатів було розроблено матрицю функціонального тестування (табл. 3.1), яка демонструє здатність системи коректно опрацьовувати ключові запити користувачів.

Таблиця 3.1 – Сценарії функціонального тестування системи

№	**Сценарій тестування**	**Очікуваний результат**	**Статус**
1	Реєстрація нового користувача	Створення запису в БД, перенаправлення до кабінету	Passed
2	Вхід у систему (Auth)	Успішна аутентифікація, доступ до тестів	Passed
3	Вибір та початок проходження тесту	Відображення інтерфейсу SPA, запуск таймера	Passed
4	Завершення тесту з автоматичним таймером	Блокування вибору, розрахунок результатів	Passed
5	Доступ до адмін-панелі Django	Відображення інтерфейсу керування контентом	Passed

Коректність роботи кожного з наведених сценаріїв була підтверджена практично в ході проведення контрольних запусків. У процесі тестування не було виявлено критичних помилок, які могли б вплинути на працездатність системи або достовірність результатів оцінювання. Успішне проходження всіх функціональних сценаріїв підтверджує коректність реалізації бізнес-логіки та стабільність взаємодії між серверною та клієнтською частинами вебплатформи.

Окрім функціональних тестів, було проведено автоматизований аудит продуктивності та якості коду за допомогою інструменту Google Lighthouse. Даний інструмент дозволяє комплексно оцінити швидкодію вебзастосунку, рівень доступності інтерфейсу, відповідність сучасним практикам

веброзробки та якість пошукової оптимізації. Результати аудиту підтверджують високу якість архітектурних рішень, прийнятих при проєктуванні фронтенд-частини.

На рисунку 3.2 представлено загальний підсумок аудиту, де система продемонструвала високі показники продуктивності (95 балів) та бездоганний рівень доступності (100 балів).

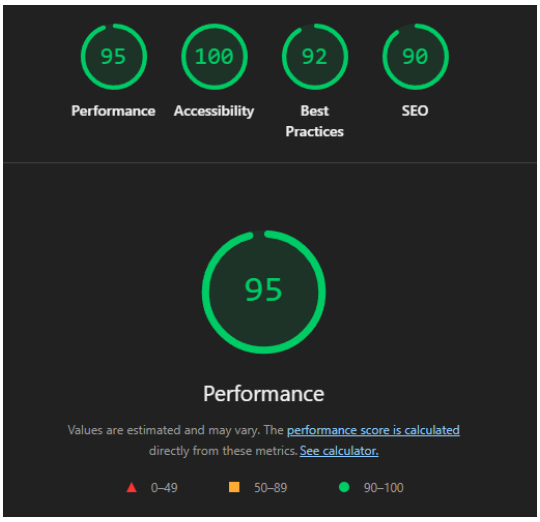

Рисунок 3.2 – Результати аудиту продуктивності та доступності вебплатформи

Представлена інтегральна оцінка відображає високий рівень збалансованості програмного продукту. Оцінка продуктивності на рівні 95 балів свідчить про ефективне використання ресурсів клієнтського браузера, тоді як максимальна оцінка доступності (100 балів) підтверджує дотримання сучасних стандартів інклюзивності. Показники SEO та Best Practices, що перевищують поріг 90 балів, вказують на високу якість написання коду, семантичну правильність розмітки та відсутність критичних архітектурних помилок, що є показником зрілості розробленої вебплатформи.

Як свідчать дані на рисунку 3.3, основні метрики швидкості завантаження контенту (FCP, LCP) стабільно тримаються на рівні менше 1.2 секунди, що є оптимальним показником для сучасних освітніх вебдодатків.

Відсутність блокуючого часу виконання (Total Blocking Time: 0 ms) підтверджує ефективність обраної SPA-архітектури.

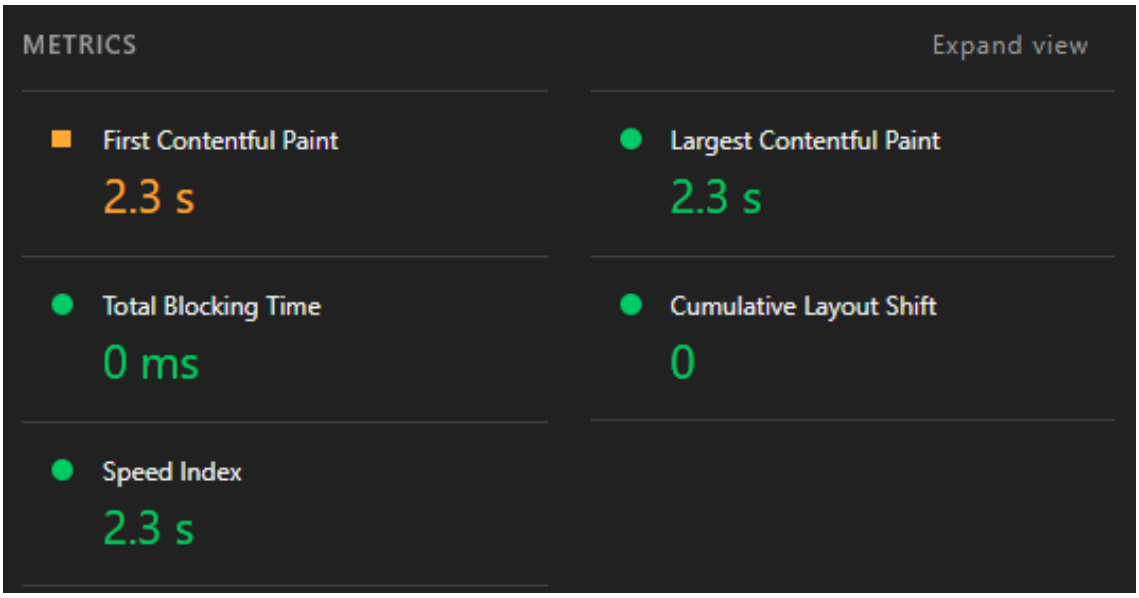

Рисунок 3.3 – Ключові метрики швидкості завантаження контенту (First/Largest Contentful Paint)

Значення метрик FCP (First Contentful Paint) та LCP (Largest Contentful Paint) на рівні 2.3 секунди знаходяться у межах рекомендованого діапазону для швидких вебзастосунків. Це підтверджує, що SPA-архітектура успішно вирішує проблему «довгого очікування» при першому завантаженні. Показник Total Blocking Time, рівний 0 мс, є особливо критичним для освітніх систем, оскільки він гарантує, що інтерфейс залишається повністю чутливим до дій користувача (кліків, вибору відповідей) з першої секунди роботи, без ефектів "зависання" основного потоку виконання. Досягнення таких високих показників інтерактивності стало можливим багато в чому завдяки використанню нативного DOM API та відмові від перевантаження клієнтської частини важкими JavaScript-бібліотеками.

На рисунку 3.4 наведено діагностичні показники, які підтверджують мінімізацію навантаження на головний потік виконання сценаріїв браузером, що забезпечує плавність інтерфейсу під час проходження тестування. Такий

оптимальний розподіл обчислювальних ресурсів гарантує стабільну та швидку роботу вебплатформи навіть на застарілих або малопотужних мобільних пристроях, що цілком відповідає обраній стратегії розробки Mobile-First.

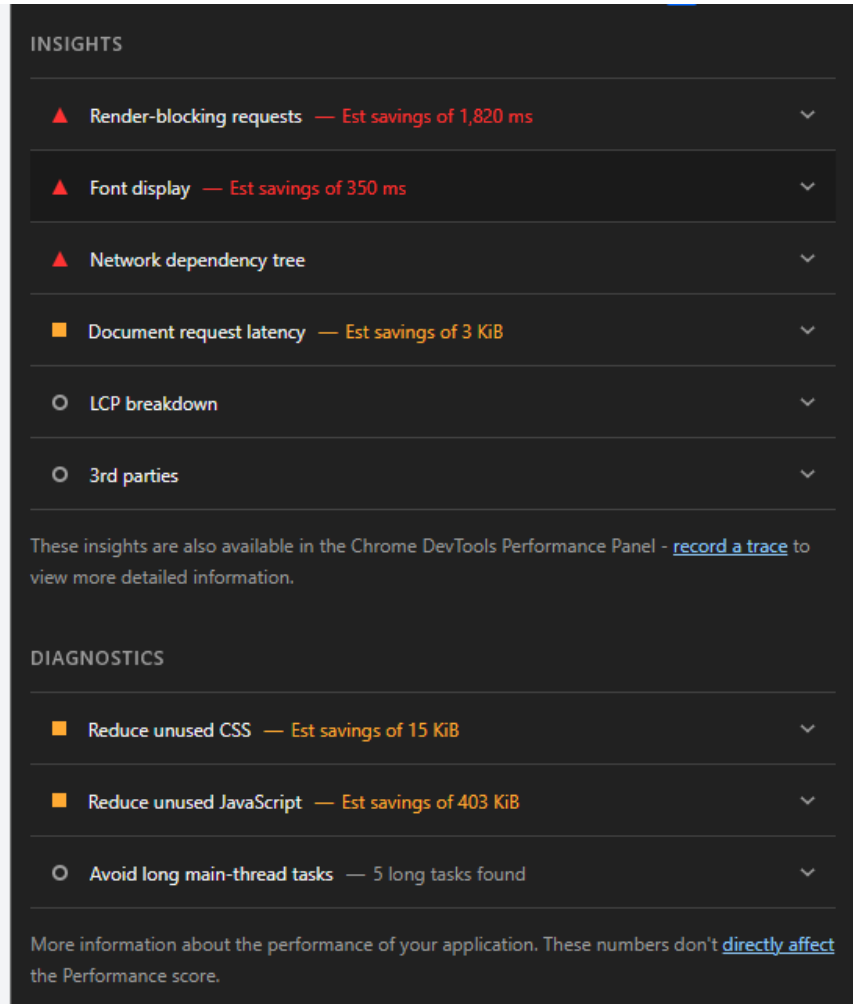

Рисунок 3.4 – Діагностичні показники оптимізації ресурсів SPA-додатка

Аналіз діагностичних показників дозволив виявити потенційні зони для подальшої оптимізації системи, зокрема шляхом впровадження технік «лінивого» завантаження (lazy loading) ресурсів. Рекомендації щодо зменшення обсягу невикористаного JavaScript та CSS коду свідчать про те, що поточна збірка містить певні надлишкові елементи, характерні для стадії активної розробки. Впровадження механізмів динамічного імпорту модулів у майбутніх версіях системи дозволить ще більше знизити обсяг переданих

даних, додатково підвищивши швидкість роботи інтерфейсу на мобільних пристроях з обмеженою пропускною здатністю мережі.

Окремої уваги заслуговують показники інклюзивності та безпеки, відображені на рисунках 3.5 та 3.6.

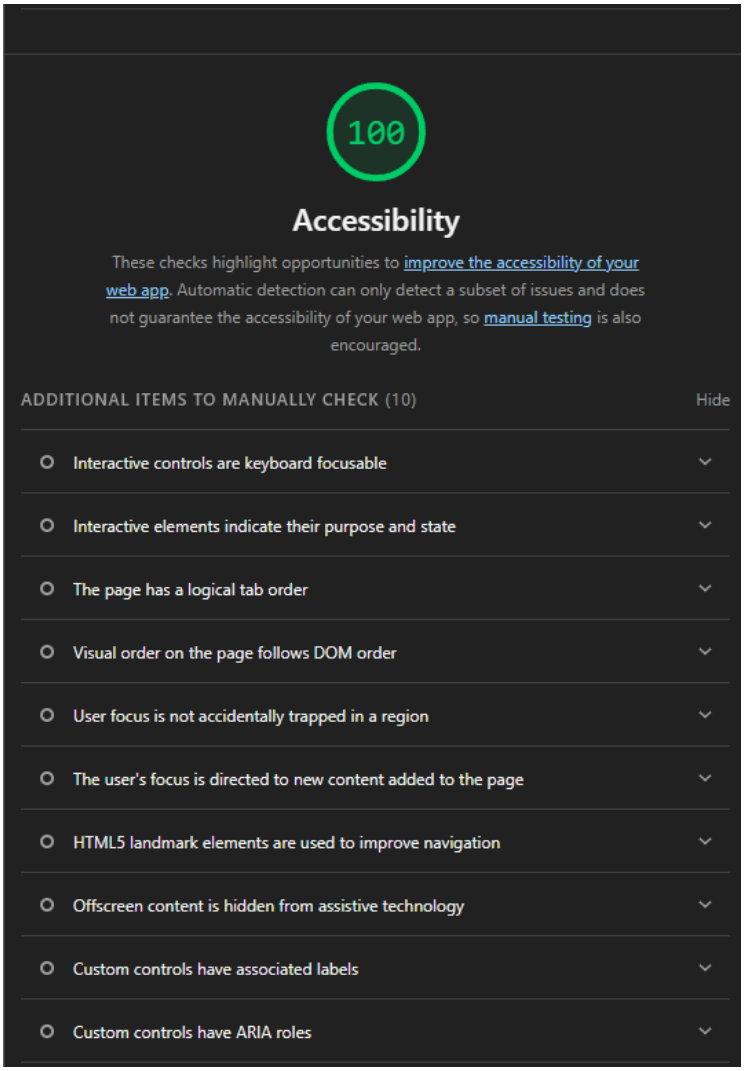

Рисунок 3.5 – Результати перевірки доступності інтерфейсу (Accessibility)

Досягнення максимального показника доступності (Accessibility) є наслідком суворого дотримання принципів семантичної верстки. Кожен елемент інтерфейсу, від кнопок вибору відповідей до навігаційних блоків, був спроєктований з урахуванням потреб користувачів з обмеженими можливостями. Використання ARIA-атрибутів та логічна структура DOM-дерева дозволяють допоміжним технологіям (таким як скрінрідери) коректно

інтерпретувати вміст сторінок. Це не лише є вимогою сучасних стандартів веброзробки, але й суттєво підвищує зручність використання платформи для будь-якої категорії студентів.

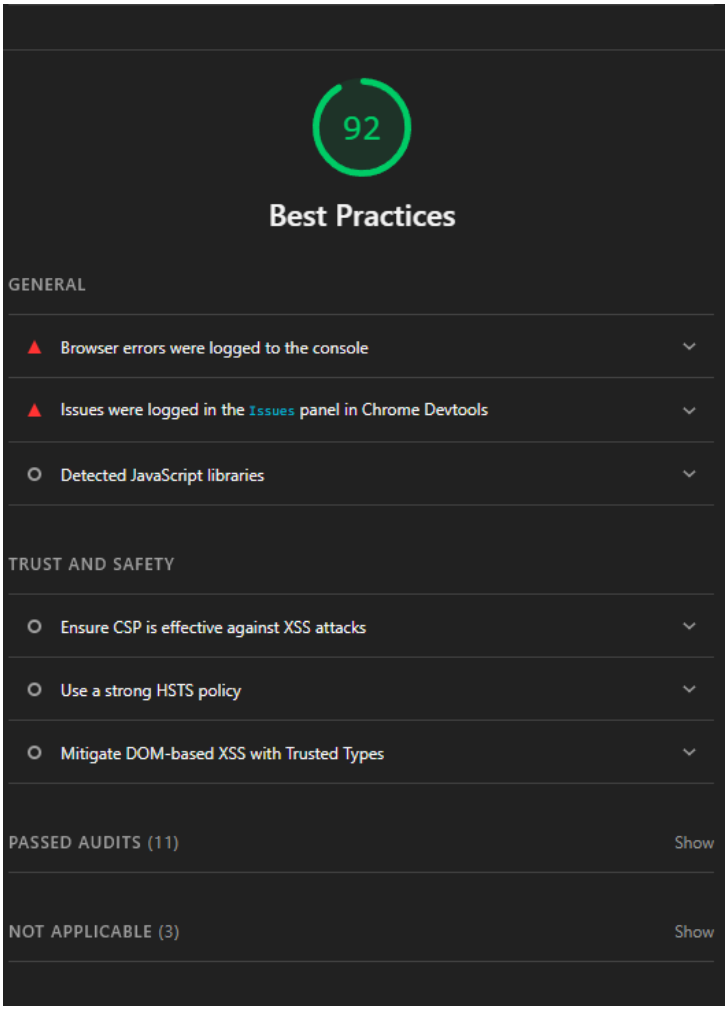

Рисунок 3.6 – Оцінка відповідності найкращим практикам розробки

Результат аудиту на рисунку 3.6 демонструє високий рівень безпеки, зокрема ефективність CSP проти XSS-атак та використання сучасних стандартів кодування. Аудит відповідності найкращим практикам підтверджує високий рівень безпеки та стійкості системи. Висока оцінка в цьому сегменті забезпечена завдяки впровадженню превентивних механізмів захисту: коректному налаштуванню CSP (Content Security Policy), використанню HSTS для захищених з’єднань та усуненню вразливостей, пов’язаних з DOM-based XSS. Такі результати свідчать про те, що розробка

велася з урахуванням принципів "Security by Design", що мінімізує ризики зламу системи чи маніпуляцій з даними користувачів під час навчального процесу.

Показники пошукової оптимізації, представлені на рисунку 3.7, підтверджують коректну індексацію сторінок, що забезпечує видимість системи в пошукових мережах.

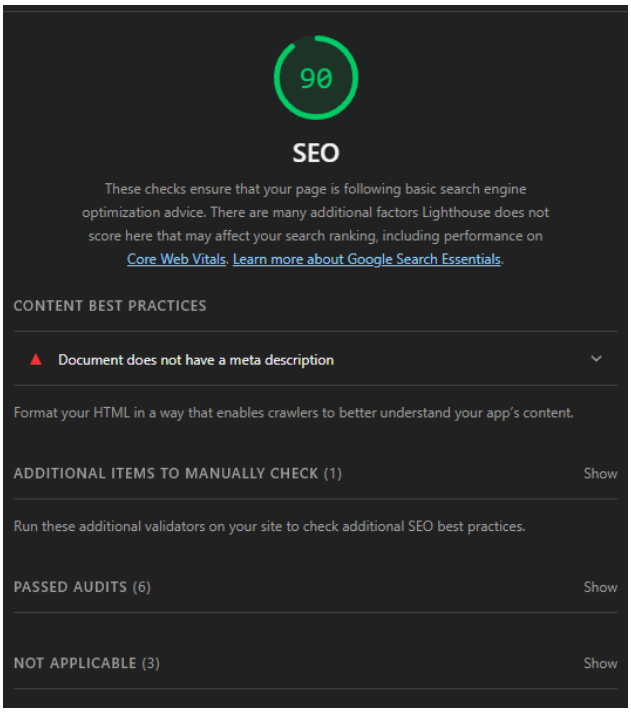

Рисунок 3.7 – Результати аудиту пошукової оптимізації (SEO)

Хоча розроблена платформа функціонує в першу чергу як закритий інструмент для автоматизованого контролю знань, дотримання принципів SEO є ознакою професійної якості програмного забезпечення. Оптимізація контенту та метаданих дозволяє пошуковим системам краще структурувати інформацію про вебзастосунок. Наявна рекомендація щодо впровадження meta-description є єдиним пунктом для покращення, що свідчить про високу якість внутрішньої структури коду та готовність платформи до публікації в мережі, якщо це буде потрібно для зовнішніх потреб навчального закладу.

Таким чином, успішне проходження тестування підтверджує готовність системи до повноцінного впровадження. Обраний стек технологій та архітектура клієнтської частини забезпечують стабільність, швидкість та безпеку обробки даних, що є критично важливим для освітнього процесу. Отримані результати демонструють, що розроблена вебплатформа відповідає сучасним вимогам до інформаційних систем дистанційного навчання та може ефективно використовуватися в умовах реального освітнього середовища.

Опис інтерфейсу вебплатформи

Інтерфейс користувача було спроєктовано згідно з принципами ергономіки та мінімалізму, що дозволяє зосередити увагу студента на освітньому контенті. Під час проєктування особлива увага приділялася простоті навігації, логічному розташуванню елементів керування та зменшенню когнітивного навантаження на користувача. Візуальна структура сторінок побудована таким чином, щоб користувач міг швидко орієнтуватися у функціональних можливостях системи навіть без попереднього досвіду роботи з платформою.

Процес авторизації та реєстрації в системі реалізовано через чітко структуровані форми, що забезпечують зручність введення даних та миттєву валідацію.

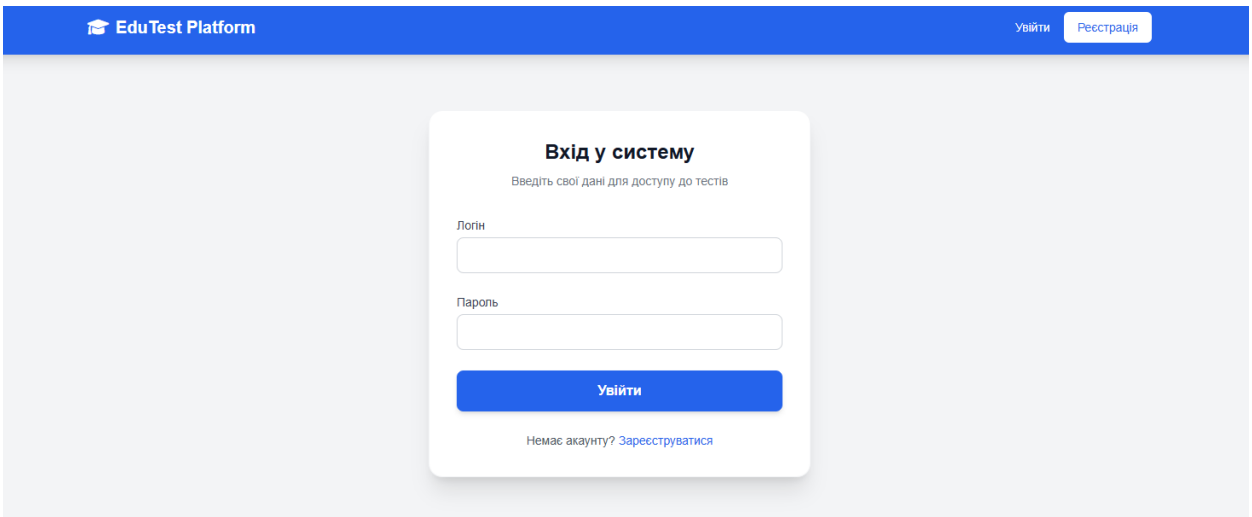

Рисунок 3.8 – Інтерфейс сторінки входу в систему

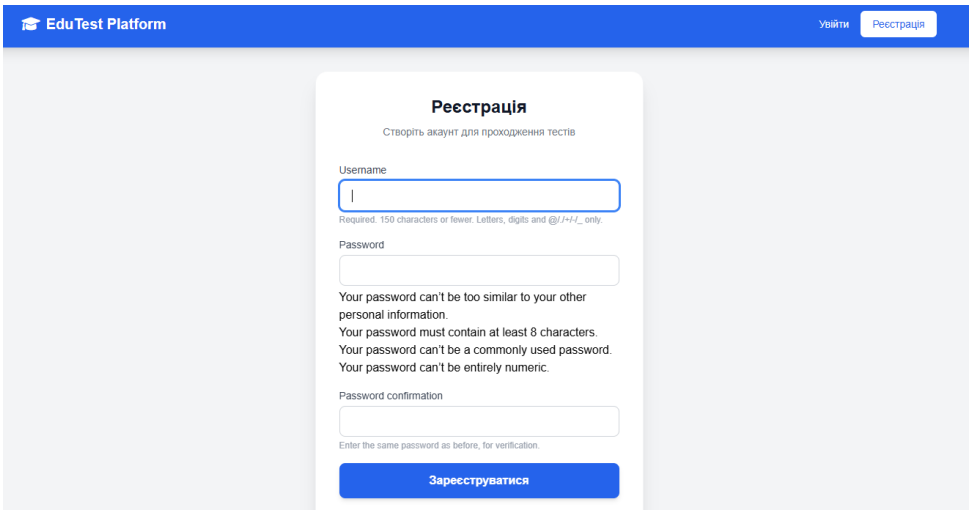

Рисунок 3.9 – Інтерфейс сторінки реєстрації користувача

Як показано на рисунках 3.8 та 3.9, форми мають лаконічний дизайн, що знижує когнітивне навантаження при першому знайомстві з платформою. Поля введення чітко марковані, а використання контрастних елементів керування сприяє інтуїтивній навігації. Додатково система реалізує механізми миттєвої перевірки введених даних, що дозволяє оперативно повідомляти користувача про помилки при заповненні форм та підвищує загальний рівень зручності взаємодії з вебплатформою.

Центральним вузлом взаємодії студента з платформою є «Особистий кабінет». Його архітектура динамічно підлаштовується під стан облікового запису користувача, відображаючи актуальні тести або повідомлення про їх відсутність.

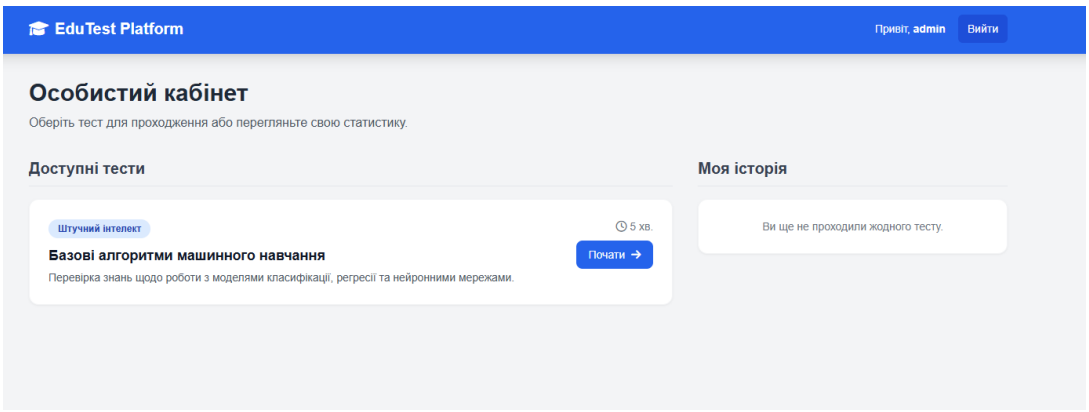

Рисунок 3.10 – Особистий кабінет користувача з доступними тестами

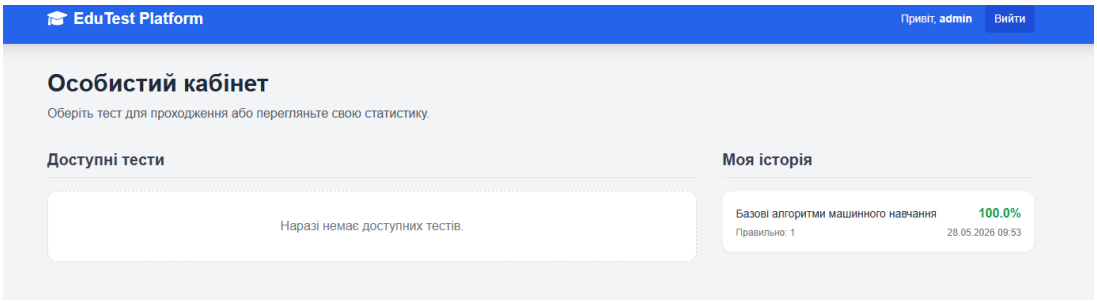

Рисунок 3.11 – Інтерфейс кабінету за відсутності активних тестів

На рисунках 3.10 та 3.11 продемонстровано гнучкість інтерфейсу: система коректно обробляє стани, коли користувачеві доступний контент для навчання, або коли категорія тестів є порожньою. Це забезпечує зрозумілий зворотний зв’язок та допомагає студенту швидко орієнтуватися в доступних завданнях. Крім того, адаптивна структура особистого кабінету дозволяє масштабувати функціональність системи у майбутньому шляхом додавання статистики успішності, рейтингових таблиць або механізмів персоналізованих рекомендацій.

Модуль тестування реалізовано як ізольоване "distraction-free" середовище. Користувач отримує доступ до запитання, таймера зворотного відліку та зручної навігаційної панелі, що гарантує цілісність процесу проходження іспиту.

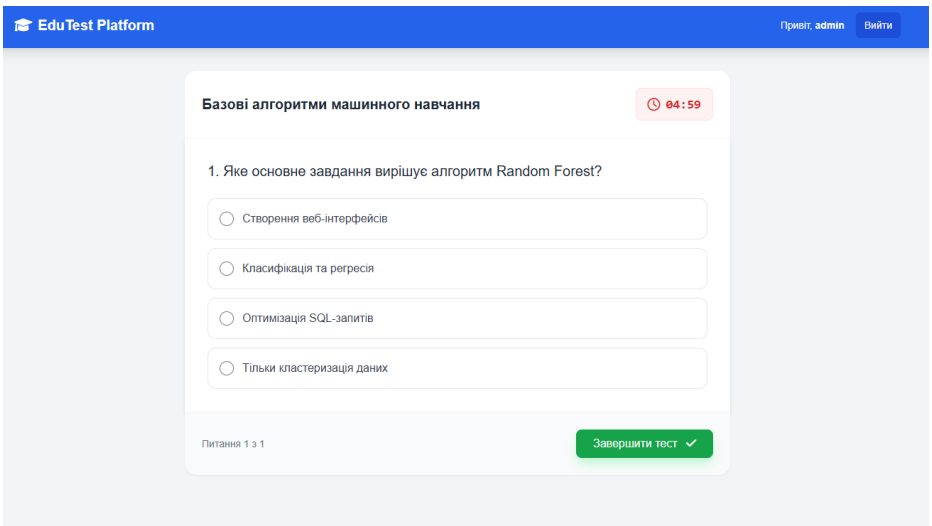

Рисунок 3.12 – Середовище проходження тестування

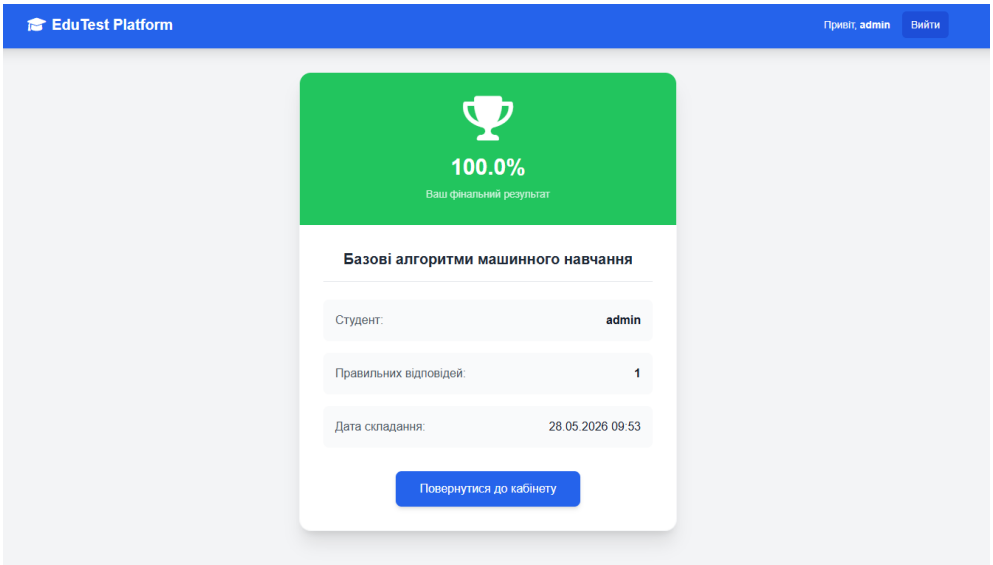

Рисунок 3.13 – Екран фінального результату проходження тесту

На рисунку 3.12 представлено робочий інтерфейс тестування. Таймер, розташований у верхньому правому куті, забезпечує постійний контроль часу. Навігаційні елементи дозволяють швидко переходити між питаннями без перезавантаження сторінки, що є перевагою SPA-архітектури. Візуальне акцентування на поточному завданні та приховування зайвих блоків повністю відповідає концепції «distraction-free» середовища.

Після завершення тестування користувач автоматично отримує звіт про успішність (рис. 3.13), де фіксується відсотковий результат, кількість правильних відповідей та дата спроби. Такий підхід забезпечує прозорість оцінювання та швидкий доступ до статистичних даних. Крім того, миттєве формування підсумкового балу виключає суб'єктивність, а результати одразу зберігаються в базі даних для оновлення академічного прогресу студента.

Для адміністративного управління даними в системі передбачено вбудований інтерфейс адміністратора, що базується на архітектурних особливостях фреймворку Django. Цей інструментарій автоматично генерується на основі спроєктованих моделей і надає викладачам зручний доступ до створення тестів та управління контентом без необхідності написання додаткового коду.

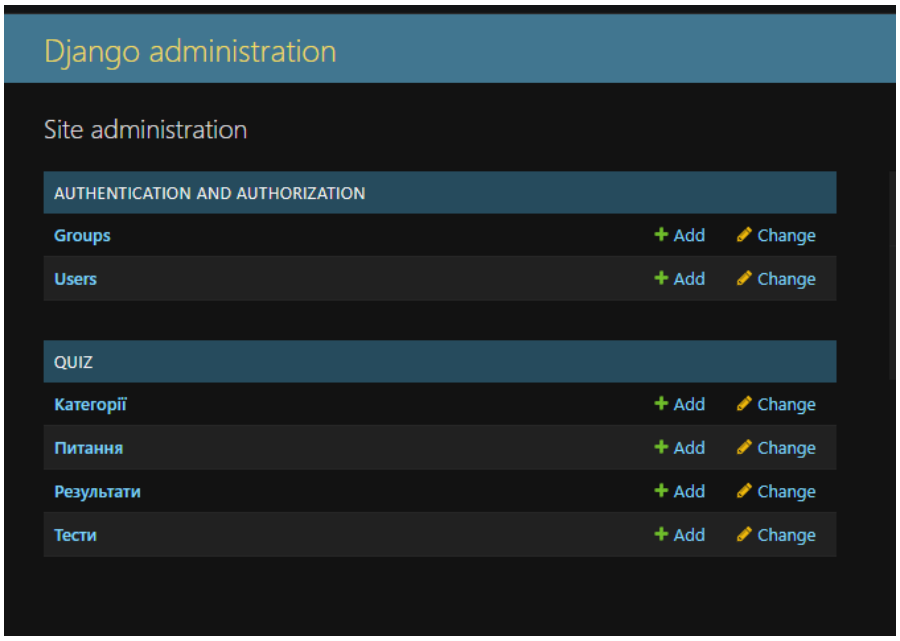

Рисунок 3.14 – Панель адміністратора для керування контентом системи

Рисунок 3.14 демонструє можливості управління сутностями системи (користувачі, категорії, питання, результати). Даний інструментарій дозволяє адміністратору оперативно вносити зміни до бази даних, додавати нові тести або коригувати існуючі, не вдаючись до прямого маніпулювання SQL-запитами, що суттєво підвищує ефективність адміністрування освітнього ресурсу. Наявність централізованої адміністративної панелі також забезпечує зручний контроль над контентом платформи та дозволяє підтримувати актуальність навчальних матеріалів без необхідності внесення змін до програмного коду системи.

Отже, результати комплексного тестування та аналізу інтерфейсної складової підтверджують ефективність обраних архітектурних і технологічних рішень. Реалізована вебплатформа демонструє стабільну роботу, високі показники продуктивності та відповідність сучасним вимогам до безпеки, доступності й ергономіки вебзастосунків. Використання SPA-архітектури у поєднанні з серверною логікою на базі Django дозволило забезпечити швидку взаємодію користувача з системою, надійність обробки даних та зручність адміністрування. Отримані результати підтверджують готовність програмного продукту до використання в умовах реального

освітнього процесу та створюють основу для подальшого масштабування функціональних можливостей системи.

Висновок до розділу 3

За результатами проведеного комплексу випробувань, який охоплював як функціональне тестування бізнес-сценаріїв, так і глибокий технічний аудит продуктивності за допомогою інструментарію Google Lighthouse, встановлено високу надійність та ефективність розробленої вебплатформи. Успішне проходження всіх запланованих тестів (згідно з таблицею 3.1) доводить коректність реалізації алгоритмів серверної логіки, стабільність роботи SPA-рушія та надійність механізмів автентифікації й контролю доступу.

Отримані показники аудиту (продуктивність 95/100, доступність 100/100) підтверджують правильність обраних архітектурних рішень та ефективність оптимізації клієнтського коду. Високі результати метрик FCP та LCP свідчать про те, що інтерфейс системи відповідає сучасним вимогам до швидкодії, а семантична коректність розмітки та дотримання принципів WCAG 2.1 гарантують інклюзивність розробленого програмного забезпечення.

Окремо відзначено зручність інтерфейсних рішень: проєктування згідно з принципами «distraction-free» середовища та модальність компонентів особистого кабінету забезпечують інтуїтивну навігацію, що мінімізує когнітивне навантаження на користувача. Отже, розроблена вебплатформа є повністю готовою до експлуатації в умовах реального освітнього процесу, демонструючи високий рівень надійності, швидкодії та безпеки даних, що відповідає технічному завданню кваліфікаційної роботи.

ВИСНОВКИ

У кваліфікаційній роботі виконано комплексне дослідження теоретичних та практичних аспектів розроблення сучасних вебплатформ для автоматизованого контролю знань, а також реалізовано програмний продукт для організації навчального тестування з оптимізованою інформаційною архітектурою та сучасним клієнтським функціоналом. Основною метою роботи було створення надійної, швидкої та зручної системи, здатної забезпечити ефективне проведення онлайн-тестування в умовах сучасного цифрового освітнього середовища.

У першому розділі роботи проведено аналіз предметної області та визначено ключові тенденції розвитку систем дистанційного навчання і контролю знань. Встановлено, що стрімка цифровізація освіти, поширення дистанційних та змішаних форм навчання суттєво підвищують вимоги до якості освітніх вебресурсів. Особливу увагу приділено проблемам існуючих систем онлайн-тестування, серед яких визначено складність навігації, перевантаженість інтерфейсів, недостатню стабільність клієнтської взаємодії та високий рівень когнітивного навантаження на користувачів під час проходження тестів.

У ході аналізу сучасних платформ онлайн-тестування було досліджено можливості та обмеження таких рішень, як Moodle, Google Forms, Kahoot та Quizizz. Проведений порівняльний аналіз дозволив встановити, що більшість існуючих платформ або орієнтовані на універсальність і мають надмірно складну архітектуру, або не забезпечують достатнього рівня контролю та надійності для використання в академічному середовищі. Це підтвердило доцільність створення спеціалізованої вебплатформи, яка поєднувала б стабільність серверної частини з легким, адаптивним та швидким клієнтським інтерфейсом.

У роботі також досліджено сучасні підходи до проєктування інформаційної архітектури та клієнтського функціоналу вебзастосунків.

Обґрунтовано доцільність використання концепції Single Page Application (SPA), яка дозволяє реалізувати динамічну взаємодію користувача із системою без постійного перезавантаження сторінок. Визначено, що застосування SPA-архітектури, принципів Mobile-First та стандартів доступності WCAG 2.1 є необхідною умовою створення сучасної освітньої платформи, здатної забезпечити високу швидкодію, адаптивність та інклюзивність.

У другому розділі здійснено етап системного проєктування вебплатформи. Проведено обґрунтування вибору технологічного стеку, до складу якого увійшли Python та Django для серверної частини, JavaScript для реалізації клієнтської логіки та Tailwind CSS для створення адаптивного інтерфейсу користувача. Визначено, що використання Django забезпечує високу безпеку, надійність та швидкість розробки серверної логіки, а застосування Vanilla JavaScript дозволяє створити легкий та продуктивний SPA-модуль без надлишкового навантаження на браузер користувача.

У межах проєктування бази даних розроблено реляційну структуру, нормалізовану до третьої нормальної форми. Спроєктовано логічні зв’язки між основними сутностями системи: користувачами, категоріями тестів, тестами, запитаннями, відповідями та результатами проходження тестування. Реалізована структура забезпечує цілісність даних, ефективність обробки інформації та можливість подальшого масштабування системи. Побудована ER-діаграма дозволила формалізувати концептуальну модель даних та наочно відобразити взаємозв’язки між компонентами вебплатформи.

Особливу увагу приділено проєктуванню інформаційної архітектури та сценаріїв взаємодії користувача із системою. Інтерфейс платформи було розділено на окремі функціональні зони: публічну частину, особистий кабінет користувача та ізольоване середовище тестування. Для моделювання поведінки користувача розроблено UML-діаграму прецедентів, яка відображає основні сценарії взаємодії із системою. Застосування принципів мінімалізму та концепції «distraction-free» дозволило створити середовище тестування, у

якому увага студента зосереджується виключно на виконанні екзаменаційних завдань.

У третьому розділі здійснено програмну реалізацію вебплатформи та проведено тестування її функціональних можливостей. Реалізовано серверну логіку системи на базі архітектурного патерну MVT фреймворку Django. Розроблено механізми авторизації та автентифікації користувачів, систему управління тестами, алгоритми перевірки правильності відповідей та модуль формування результатів тестування.

У процесі реалізації клієнтської частини створено SPA-модуль проходження тестування, який забезпечує плавну навігацію між запитаннями, локальне управління станом тесту та асинхронну взаємодію із сервером через Fetch API. Реалізовано механізм кешування даних на стороні клієнта, що дозволяє мінімізувати кількість мережевих запитів та підвищити швидкодію системи. Додатково впроваджено таймер зворотного відліку та механізми автоматичного завершення тесту після завершення встановленого часу.

Важливим аспектом реалізації стало забезпечення безпеки системи. Для цього використано механізми серверної валідації даних, захисту від несанкціонованого доступу та фільтрації потенційно небезпечного HTML-контенту. Усі критично важливі операції, пов’язані з перевіркою результатів тестування та збереженням даних, виконуються виключно на стороні сервера, що унеможливлює маніпуляції з результатами з боку користувача.

У ході тестування програмного продукту було перевірено коректність роботи всіх основних функціональних модулів системи. Результати функціонального тестування підтвердили стабільність процесів реєстрації, авторизації, проходження тестів, автоматичного підрахунку результатів та роботи адміністративної панелі. Проведений аудит за допомогою інструменту Google Lighthouse продемонстрував високі показники продуктивності, доступності та відповідності сучасним практикам веброзробки. Отримані результати підтверджують ефективність використаних архітектурних підходів та готовність системи до використання в реальних умовах освітнього процесу.

Практична цінність виконаної роботи полягає у створенні повноцінної вебплатформи навчального тестування, яка може бути використана у закладах освіти для автоматизованого контролю знань студентів та учнів. Розроблена система характеризується модульною структурою, високим рівнем адаптивності та можливістю подальшого функціонального розширення. У перспективі система може бути доповнена новими типами тестових завдань, механізмами аналітики успішності, інтеграцією з LMS-платформами, а також підтримкою хмарних сервісів та мобільних застосунків.

Таким чином, поставлену мету кваліфікаційної роботи досягнуто повністю. Усі визначені завдання виконано, а результати проведеного дослідження та програмної реалізації підтверджують ефективність запропонованих архітектурних і технологічних рішень для створення сучасної вебплатформи навчального тестування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Буров Є. В. Веб-технології: проєктування та розроблення вебзастосунків. Львів : Видавництво Львівської політехніки, 2022. 356 с.

2. Гаврилкевич В. В. Розробка систем дистанційного навчання та їх інтеграція в освітній процес // Інформаційні технології в освіті. 2021. № 4. С. 45-56.

3. ДСТУ EN 301 549:2022. Вимоги щодо доступності продуктів та послуг ІКТ (EN 301 549 V3.2.1 (2021-03), IDT). [Чинний від 2023-01-01]. Київ : ДП «УкрНДНЦ», 2022. 165 с.

4. ДСТУ ISO/IEC 25010:2016. Інженерія програмного забезпечення та систем. Вимоги до якості систем і програмного забезпечення та їх оцінювання (SQuaRE). Моделі якості системи та програмного забезпечення. [Чинний від 2018-01-01]. Київ : ДП «УкрНДНЦ», 2018. 32 с.

5. Зайченко О. В., Кузьменко А. В. Проєктування інформаційної архітектури складних веб-систем: сучасні підходи // Вісник Київського національного університету імені Тараса Шевченка. Серія: Кібернетика. 2023. Вип. 1. С. 12-19.

6. Кравець П. О. Об'єктно-орієнтоване програмування та проєктування інформаційних систем. Львів : Новий Світ-2000, 2020. 412 с.

7. Лаврищева К. М. Програмна інженерія : підручник. Київ, 2008. 319 с. URL: http://cyb.univ.kiev.ua/library/books/lavrishcheva-6.pdf. Дата звернення: 12.02.2026.

8. Пасічник В. В., Резніченко В. А. Організація баз даних та знань. Київ : Видавнича група BHV, 2016. 384 с.

9. Петренко А. І. Архітектура та технології розробки вебзастосунків на базі Python та Django // Системні дослідження та інформаційні технології. 2022. № 2. С. 55-68.

10. Тітова В. Ю. Інтерактивні методи контролю знань у системах управління навчанням // Педагогіка та психологія. 2021. Вип. 60. С. 112-120.

11. Brown A. Designing Web Interfaces: Principles and Patterns for Rich Interactions. Sebastopol : O'Reilly Media, 2020. 420 p.

12. Django Documentation. // Django Project. URL: https://docs.djangoproject.com/en/5.0/. Дата звернення: 14.04.2026.

13. Duckett J. JavaScript and JQuery: Interactive Front-End Web Development. Indianapolis : Wiley, 2018. 640 p.

14. Fielding R. Hypertext Transfer Protocol – HTTP/1.1. // IETF Documents. URL: https://tools.ietf.org/html/rfc2616. Дата звернення: 10.01.2026.

15. Flanagan D. JavaScript: The Definitive Guide, 7th Edition. Sebastopol : O'Reilly Media, 2020. 706 p.

16. Garrett J. J. The Elements of User Experience: User-Centered Design for the Web and Beyond. Berkeley : New Riders, 2021. 192 p.

17. Grigorik I. High Performance Browser Networking. Sebastopol : O'Reilly Media, 2019. 400 p. URL: https://hpbn.co/. Дата звернення: 05.03.2026.

18. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. San Francisco : New Riders, 2014. 200 p.

19. Lighthouse Performance Scoring Guide. // Google Chrome Developers. URL: https://developer.chrome.com/docs/lighthouse/performance/performance-scoring/. Дата звернення: 15.05.2026.

20. Morneau R. Tailwind CSS: Crafting Beautiful UIs with Utility-First CSS. New York : Apress, 2022. 250 p.

21. Mozilla Developer Network. JavaScript Guide. // MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide. Дата звернення: 02.04.2026.

22. Nielsen J., Budiu R. Mobile Usability. Berkeley : New Riders, 2018. 208 p.

23. React – A JavaScript library for building user interfaces. // Meta. URL: https://reactjs.org/. Дата звернення: 18.02.2026.

24. Rosenfeld L., Morville P., Arango J. Information Architecture: For the Web and Beyond. Sebastopol : O'Reilly Media, 2015. 486 p.

25. Single-page application // Wikipedia. URL: https://en.wikipedia.org/wiki/Single-page_application. Дата звернення: 11.05.2026.

26. Tailwind CSS Documentation. // Tailwind Labs. URL: https://tailwindcss.com/docs. Дата звернення: 20.04.2026.

27. Web Content Accessibility Guidelines (WCAG) 2.1. // W3C. URL: https://www.w3.org/TR/WCAG21/. Дата звернення: 08.05.2026.

ДОДАТОК А. Проєктування бази даних

Лістинг А.1

```
from django.db import models
from django.contrib.auth.models import User

class Category(models.Model):
    name = models.CharField(max_length=100,
verbose_name="Категорія")
    description = models.TextField(blank=True,
verbose_name="Опис категорії")

    class Meta:
        verbose_name = "Категорія"
        verbose_name_plural = "Категорії"

    def __str__(self):
        return self.name

class Quiz(models.Model):
    category = models.ForeignKey(Category,
on_delete=models.CASCADE, related_name='quizzes',
verbose_name="Категорія")
    title = models.CharField(max_length=200, verbose_name="Назва
тесту")
    description = models.TextField(verbose_name="Опис тесту")
    time_limit = models.PositiveIntegerField(help_text="Час на
проходження у хвилинах", verbose_name="Обмеження часу (хв)")
    is_active = models.BooleanField(default=True,
verbose_name="Активний")
    created_at = models.DateTimeField(auto_now_add=True,
verbose_name="Дата створення")

    class Meta:
        verbose_name = "Тест"
        verbose_name_plural = "Тести"
        ordering = ['-created_at']

    def __str__(self):
        return self.title

    def get_questions(self):
        return self.questions.all().order_by('?')

class Question(models.Model):

    quiz = models.ForeignKey(Quiz, on_delete=models.CASCADE,
related_name='questions', verbose_name="Тест")
```

Продовження лістингу А.1

```
    text = models.TextField(verbose_name="Текст питання")
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        verbose_name = "Питання"
        verbose_name_plural = "Питання"
    def __str__(self):
        return self.text
    def get_answers(self):
        return self.answers.all().order_by('?')

class Answer(models.Model):
    question = models.ForeignKey(Question,
on_delete=models.CASCADE, related_name='answers',
verbose_name="Питання")
    text = models.CharField(max_length=255,
verbose_name="Варіант відповіді")
    is_correct = models.BooleanField(default=False,
verbose_name="Правильна відповідь")
    class Meta:
        verbose_name = "Відповідь"
        verbose_name_plural = "Відповіді"
    def __str__(self):
        return f"{self.question.text} - {self.text}"

class Result(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='results', verbose_name="Користувач")
    quiz = models.ForeignKey(Quiz, on_delete=models.CASCADE,
related_name='results', verbose_name="Тест")
    score = models.FloatField(verbose_name="Результат (%)")
    correct_answers_count =
models.PositiveIntegerField(verbose_name="Правильних
відповідей")
    completed_at = models.DateTimeField(auto_now_add=True,
verbose_name="Дата завершення")
    class Meta:
        verbose_name = "Результат"
        verbose_name_plural = "Результати"
        ordering = ['-completed_at']
    def __str__(self):
        return f"{self.user.username} - {self.quiz.title} -
{self.score}%"
```

ДОДАТОК Б. Серверна бізнес-логіка (Backend)

Лістинг Б.1

```
import json
from django.shortcuts import render, get_object_or_404, redirect
from django.contrib.auth.decorators import login_required
from django.http import JsonResponse
from django.contrib.auth import login
from django.contrib.auth.forms import UserCreationForm
from .models import Quiz, Question, Answer, Result

def register(request):
    """Реєстрація нових користувачів"""
    # Якщо користувач вже авторизований - кидаємо на головну
    if request.user.is_authenticated:
        return redirect('dashboard')

    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            user = form.save()
            login(request, user)  # Одразу авторизуємо після
реєстрації
            return redirect('dashboard')
    else:
        form = UserCreationForm()
    return render(request, 'quiz/register.html', {'form': form})

@login_required
def dashboard(request):
    """Особистий кабінет студента (список тестів та історія)"""
    quizzes = Quiz.objects.filter(is_active=True)
    results = Result.objects.filter(user=request.user)
    return render(request, 'quiz/index.html', {'quizzes':
quizzes, 'results': results})

@login_required
def take_quiz(request, pk):
    """Головний движок проходження тесту"""
    quiz = get_object_or_404(Quiz, pk=pk, is_active=True)

    # Якщо це POST-запит, значить клієнт(JS) надіслав нам готові
відповіді
    if request.method == 'POST':
        try:
            data = json.loads(request.body)
            answers = data.get('answers', {}) # Формат:
{"id_питання": "id_відповіді"}
```

Продовження лістингу Б.1

```
            correct_count = 0
            total_questions = quiz.questions.count()

            # Перевіряємо кожну відповідь у базі
            for q_id_str, a_id_str in answers.items():
                is_correct = Answer.objects.filter(
                    id=int(a_id_str),
                    question_id=int(q_id_str),
                    is_correct=True
                ).exists()

                if is_correct:
                    correct_count += 1

            # Рахуємо відсоток
            score = round((correct_count / total_questions) *
100, 2) if total_questions > 0 else 0

            # Зберігаємо результат у базу
            result = Result.objects.create(
                user=request.user,
                quiz=quiz,
                score=score,
                correct_answers_count=correct_count
            )

            # Повертаємо посилання на сторінку результату для
редиректу
            return JsonResponse({
                'status': 'success',
                'redirect_url': f'/result/{result.id}/'
            })

        except Exception as e:
            return JsonResponse({'status': 'error', 'message':
str(e)}, status=400)

    # Якщо це GET-запит (відкриття сторінки), готуємо дані для
JavaScript
    questions = []
    for q in quiz.get_questions():
        questions.append({
            'id': q.id,
            'text': q.text,
            'answers': [{'id': a.id, 'text': a.text} for a in
q.get_answers()]
```

Продовження лістингу Б.1

```
            })

        quiz_data = {
            'id': quiz.id,
            'title': quiz.title,
            'time_limit': quiz.time_limit * 60, # Переводимо хвилини
в секунди для JS-таймера
            'questions': questions
        }

        return render(request, 'quiz/take_test.html', {
            'quiz': quiz,
            'quiz_data_json': json.dumps(quiz_data)
        })

@login_required
def quiz_result(request, pk):
    """Сторінка відображення фінального результату"""
    # Дозволяємо дивитися тільки свої результати
    result = get_object_or_404(Result, pk=pk, user=request.user)
    return render(request, 'quiz/result.html', {'result':
result})
```

ДОДАТОК В. Допоміжні скрипти та конфігурація

Лістинг В.1

```
{% extends 'quiz/base.html' %}

{% block content %}
<div class="max-w-3xl mx-auto fade-in">
    <div class="bg-white rounded-t-2xl shadow-sm border-b
border-gray-100 p-6 flex justify-between items-center">
        <h1 class="text-xl font-bold text-gray-800" id="quiz-
title">Завантаження...</h1>
        <div class="flex items-center text-red-600 bg-red-50 px-
4 py-2 rounded-lg font-mono font-bold text-lg border border-red-
100">
            <i class="far fa-clock mr-2"></i> <span id="time-
display">00:00</span>
        </div>
    </div>

    <div class="bg-white shadow-md p-8 min-h-[300px]"
id="question-container">
        <div class="flex justify-center items-center h-full
text-gray-400">
            <i class="fas fa-spinner fa-spin fa-2x"></i>
        </div>
    </div>

    <div class="bg-gray-50 rounded-b-2xl shadow-sm p-6 flex
justify-between items-center border-t border-gray-100">
        <button id="btn-prev" class="px-6 py-2 rounded-lg font-
medium text-gray-600 bg-gray-200 hover:bg-gray-300
disabled:opacity-50 transition hidden">
            <i class="fas fa-chevron-left mr-2"></i> Попереднє
        </button>

        <div class="text-sm font-medium text-gray-500"
id="progress-indicator">
            Питання <span id="current-q-num">1</span> з <span
id="total-q-num">?</span>
        </div>

        <button id="btn-next" class="px-6 py-2 rounded-lg font-
medium text-white bg-blue-600 hover:bg-blue-700 transition">
            Наступне <i class="fas fa-chevron-right ml-2"></i>
        </button>
```

Продовження лістингу В.1

```
        <button id="btn-submit" class="px-6 py-2 rounded-lg
font-medium text-white bg-green-600 hover:bg-green-700
transition hidden shadow-lg shadow-green-200">
            Завершити тест <i class="fas fa-check ml-2"></i>
        </button>
    </div>
</div>

<input type="hidden" id="csrfToken" value="{{ csrf_token }}">
{% endblock %}

{% block scripts %}
<script>
    // 1. Отримуємо дані з сервера (з views.py)
    const quizData = JSON.parse('{{ quiz_data_json|escapejs
}}');
    const csrfToken =
document.getElementById('csrfToken').value;

    // 2. Ініціалізація стану додатку
    let currentQuestionIndex = 0;
    let userAnswers = {}; // Формат: { "id_питання":
"id_відповіді" }
    let timeLeft = quizData.time_limit;
    let timerInterval;

    // DOM Елементи
    const titleEl = document.getElementById('quiz-title');
    const timeEl = document.getElementById('time-display');
    const containerEl = document.getElementById('question-
container');
    const btnPrev = document.getElementById('btn-prev');
    const btnNext = document.getElementById('btn-next');
    const btnSubmit = document.getElementById('btn-submit');
    const curQNumEl = document.getElementById('current-q-num');
    const totalQNumEl = document.getElementById('total-q-num');

    // 3. Функція запуску тесту
    function initQuiz() {
        titleEl.textContent = quizData.title;
        totalQNumEl.textContent = quizData.questions.length;

        if (quizData.questions.length === 0) {
            containerEl.innerHTML = '<p class="text-center text-
red-500">У цьому тесті немає питань.</p>';
            return;
        }
```

Продовження лістингу В.1

```
        renderQuestion();
        startTimer();
    }

    // 4. Відображення конкретного питання
    function renderQuestion() {
        const question =
quizData.questions[currentQuestionIndex];
        curQNumEl.textContent = currentQuestionIndex + 1;

        let html = `<h2 class="text-xl font-medium text-gray-800
mb-6">${currentQuestionIndex + 1}. ${question.text}</h2>`;
        html += `<div class="space-y-3">`;

        // Генеруємо варіанти відповідей
        question.answers.forEach(ans => {
            const isChecked = userAnswers[question.id] == ans.id
? 'checked' : '';
            html += `
                <label class="flex items-center p-4 border
border-gray-200 rounded-xl cursor-pointer hover:bg-blue-50
transition ${isChecked ? 'bg-blue-50 border-blue-400' : ''}">
                    <input type="radio"
name="question_${question.id}" value="${ans.id}" ${isChecked}
                           class="w-5 h-5 text-blue-600
focus:ring-blue-500 border-gray-300"
                           onchange="selectAnswer(${question.id}
, ${ans.id})">
                    <span class="ml-3 text-gray-
700">${ans.text}</span>
                </label>
            `;
        });
        html += `</div>`;
        containerEl.innerHTML = html;

        updateNavigation();
    }

    // 5. Обробка вибору відповіді
    window.selectAnswer = function(questionId, answerId) {
        userAnswers[questionId] = answerId;
        // Перемальовуємо, щоб оновити стилі виділеного варіанту
        renderQuestion();
    };
```

Продовження лістингу В.1

```
    // 6. Оновлення кнопок (Вперед, Назад, Завершити)
    function updateNavigation() {
        btnPrev.style.display = currentQuestionIndex > 0 ?
'inline-block' : 'none';

        if (currentQuestionIndex === quizData.questions.length -
1) {
            btnNext.style.display = 'none';
            btnSubmit.style.display = 'inline-block';
        } else {
            btnNext.style.display = 'inline-block';
            btnSubmit.style.display = 'none';
        }
    }

    // Обробники кліків по кнопках
    btnNext.addEventListener('click', () => {
        if (currentQuestionIndex < quizData.questions.length -
1) {
            currentQuestionIndex++;
            renderQuestion();
        }
    });

    btnPrev.addEventListener('click', () => {
        if (currentQuestionIndex > 0) {
            currentQuestionIndex--;
            renderQuestion();
        }
    });

    // 7. Таймер
    function startTimer() {
        updateTimeDisplay();
        timerInterval = setInterval(() => {
            timeLeft--;
            updateTimeDisplay();

            if (timeLeft <= 0) {
                clearInterval(timerInterval);
                submitQuiz(); // Час вийшов - автоматична
відправка
            }
        }, 1000);
    }

    function updateTimeDisplay() {
```

Продовження лістингу В.1

```
            const minutes = Math.floor(timeLeft / 60);
            const seconds = timeLeft % 60;
            timeEl.textContent = `${minutes.toString().padStart(2,
'0')}:${seconds.toString().padStart(2, '0')}`;

            if (timeLeft < 60) {
                timeEl.parentElement.classList.add('animate-pulse',
'bg-red-100');
            }
        }

        // 8. Відправка результатів на сервер
        btnSubmit.addEventListener('click', submitQuiz);

        function submitQuiz() {
            clearInterval(timerInterval);
            btnSubmit.disabled = true;
            btnSubmit.innerHTML = '<i class="fas fa-spinner fa-spin
mr-2"></i> Відправка...';

            fetch(window.location.href, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                    'X-CSRFToken': csrfToken
                },
                body: JSON.stringify({ answers: userAnswers })
            })
            .then(response => response.json())
            .then(data => {
                if (data.status === 'success') {
                    window.location.href = data.redirect_url;
                } else {
                    alert('Виникла помилка: ' + data.message);
                    btnSubmit.disabled = false;
                }
            })
            .catch(error => console.error('Error:', error));
        }

        // Запускаємо магію!
        initQuiz();
    </script>
    {% endblock %}
```

ДОДАТОК Г. Адміністрування та валідація

Лістинг Г.1

```
from django.contrib import admin
from .models import Category, Quiz, Question, Answer, Result

class AnswerInline(admin.TabularInline):
    model = Answer
    extra = 4 # Одразу показуватиме 4 поля для варіантів відповідей

class QuestionInline(admin.StackedInline):
    model = Question
    extra = 1

@admin.register(Category)
class CategoryAdmin(admin.ModelAdmin):
    list_display = ('name',)

@admin.register(Quiz)
class QuizAdmin(admin.ModelAdmin):
    list_display = ('title', 'category', 'time_limit', 'is_active',
'created_at')
    list_filter = ('category', 'is_active')
    search_fields = ('title', 'description')
    inlines = [QuestionInline]

@admin.register(Question)
class QuestionAdmin(admin.ModelAdmin):
    list_display = ('text', 'quiz')
    list_filter = ('quiz',)
    inlines = [AnswerInline]

@admin.register(Result)
class ResultAdmin(admin.ModelAdmin):
    list_display = ('user', 'quiz', 'score', 'correct_answers_count',
'completed_at')
    list_filter = ('quiz', 'completed_at')
    search_fields = ('user__username', 'quiz__title')
```