Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ПЛАТФОРМИ ДЛЯ УПРАВЛІННЯ ПЕРСОНАЛЬНИМ ПОРТФОЛІО З ВИКОРИСТАННЯМ REACT, NODE.JS ТА MYSQL

DEVELOPMENT OF A PLATFORM FOR MANAGING A PERSONAL PORTFOLIO USING REACT, NODE.JS AND MYSQL

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-42
Симончук Вадим Володимирович

Керівник:
Бойко Лев Степанович

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«20» 12 2024р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Симончуку Вадиму Володимировичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка платформи для управління персональним портфоліо з використанням React, Node.js та MySQL»*
Керівник роботи: *к.т.н., доцент Бойко Л. С.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *Visual Studio Code, React, Node.js, Express, MySQL, методичні вказівки до виконання кваліфікаційної роботи бакалавра*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *Провести аналіз предметної області, розглянути аналоги, їх переваги та недоліки; визначити вимоги до розроблювальної платформи; вибрати засоби для реалізації платформи; розробити структуру платформи і створити базу даних; розробити серверну сторону на Node.js; розробити клієнтську сторону на React; протестувати та налагодити систему*
5. Перелік графічного матеріалу: *11 рисунків, 1 додаток*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Бойко Л. С.*		
Специфікація вимог до розробленої системи	*Бойко Л. С.*		
Розробка об'єкта проектування	*Бойко Л. С.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	3,56 %		
Академічна доброчесність	*Бойко Л. С.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	виконано
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	виконано
3	Обгрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	виконано
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	виконано
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	виконано
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	виконано
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	виконано

Здобувач вищої освіти _____ Вадим СИМОНЧУК

Керівник кваліфікаційної роботи _____ Лев БОЙКО

АНОТАЦІЯ

Симончук В. В. Розробка платформи для управління персональним портфоліо з використанням React, Node.js та MySQL. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі здійснено аналіз існуючих аналогів, а також їх переваги і недоліки та сформульовано завдання.

В другому розділі здійснено моделювання системи, а також було обрано засоби та інструменти для вирішення завдань.

У третьому розділі реалізовано клієнтську та серверну частини веб-додатку на основі React, Node.js та бази даних MySQL, проведено тестування, усунено виявлені помилки. У висновках зазначено розроблену платформу для управління персональним портфоліо на основі React, Node.js та MySQL. Система реалізує повний цикл CRUD-операцій, аутентифікацію користувачів, API інтеграцію та адаптивний інтерфейс. Проведено комплексне тестування та налагодження всіх компонентів системи. Розроблене рішення відповідає функціональним і нефункціональним вимогам та готове до подальшого масштабування.

Ключові слова: React, Node.js, MySQL, ExpressJS, веб-розробка, сервер, клієнт, API, база даних, портфоліо.

ABSTRACT

Symonchuk V. Development of a platform for managing a personal portfolio using React, Node.js and MySQL. Manuscript. Qualification work of the bachelor's program "Software engineering" in the specialty "Software engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices.

The first chapter presents the existing analogs, their advantages and disadvantages, and formulates the objectives.

The second chapter, the system was modeled, and tools and instruments for solving the tasks were selected.

In the third section, the client and server parts of the web application based on React, Node.js, and the MySQL database are implemented, tested, and the errors found are eliminated. The conclusions summarize the developed platform for personal portfolio management based on React, Node.js and MySQL. The system implements a full cycle of CRUD operations, user authentication, API integration, and an adaptive interface. Comprehensive testing and debugging of all system components was carried out. The developed solution meets functional and non-functional requirements and is ready for further scaling.

Keywords: React, Node.js, MySQL, ExpressJS, web-development, server, client, API, database, portfolio.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасних умовах стрімкого розвитку інформаційних технологій на ринку праці дедалі більшої ваги набуває можливість професійної самопрезентації у вигляді інтерактивного портфоліо. Воно слугує не лише вітриною завершених проєктів і досягнень, а й платформою для демонстрації навичок, досвіду та індивідуального стилю роботи. Існуючі онлайн-сервіси часто не задовольняють усіх потреб спеціалістів: одні обмежені шаблонними інтерфейсами, інші не забезпечують гнучкого управління бекенд-даними та розширення через RESTful API або аналітичні модулі. Водночас поєднання стеку React, Node.js і MySQL дозволяє розробити масштабоване, безпечне та настроюване рішення з можливістю розширення функціоналу і глибокої аналітики. Тому створення такої платформи є актуальним завданням як для IT-фахівців, так і для широкої аудиторії користувачів, які прагнуть максимально ефективно презентувати свої професійні компетенції.

Таким чином, метою даної кваліфікаційної роботи є розробка комплексної веб-платформи для управління персональним портфоліо з використанням React, Node.js та MySQL, яка забезпечить повний цикл створення, редагування, демонстрації та аналітики професійних даних користувача.

Об’єктом кваліфікаційної роботи є процес розробки веб-платформи для самопрезентації фахівців у цифровому середовищі.

Предметом кваліфікаційної роботи є методи та інструменти реалізації клієнтської, серверної частини та бази даних для створення персонального портфоліо

Для досягнення поставленої мети були поставлені такі завдання:

- провести аналіз існуючих рішень на ринку та оцінити їхні переваги й недоліки;
- визначити функціональні та нефункціональні вимоги до розроблюваної платформи та спроектувати її архітектуру;

- обрати оптимальний технологічний стек і засоби для реалізації клієнтської та серверної частин;
- розробити структуру бази даних у MySQL і забезпечити її нормалізацію;
- реалізувати серверну логіку на Node.js з RESTful API та обробкою JWT-автентифікації;
- створити клієнтський інтерфейс на React з реактивними компонентами, маршрутизацією та управлінням станом;
- провести тестування різними методами та налагодити інформаційно-комп’ютерну систему.

РОЗДІЛ 1
АНАЛІЗ РОЗРОБКИ ПЛАТФОРМИ ДЛЯ УПРАВЛІННЯ ПЕРСОНАЛЬНИМ ПОРТФОЛІО І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Сучасний цифровий ринок праці висуває до фахівців нові вимоги, серед яких ключовою стає наявність персонального портфоліо, що не лише демонструє професійні навички й досягнення, а й дозволяє виділитися серед конкурентів, підкреслити індивідуальний стиль і підвищити шанси на працевлаштування або отримання замовлення [1]. Якісне портфоліо стає візитівкою спеціаліста, яке дає уявлення про його досвід, компетенції, підхід до вирішення завдань і навіть визначає мотивацію до подальшого розвитку.

Існуючими аналогами платформ для управління персональним портфоліо є кілька сервісів, які займають провідні позиції в різних нішах творчої та IT-індустрії:

– Behance [2] – це одна з найвідоміших платформ для творчих професіоналів, що дозволяє художникам, ілюстраторам, графічним дизайнерам і мультимедіа-спеціалістам не лише публікувати свої роботи у вигляді візуальних кейсів, а й отримувати відгуки від колег і потенційних замовників. Користувачі можуть створювати детальні проєктні сторінки з описом процесу, використовувати теги для полегшеного пошуку та долучатися до тематичних спільнот. Behance також інтегрований із каталогами вакансій Adobe Talent, що дає додаткові можливості трудового працевлаштування та фрілансу. Проте, попри широку аудиторію, кастомізація шаблонів обмежена вбудованим інструментарієм і не передбачає глибокого налаштування бекенд-логіки або власних аналітичних модулів;

– Dribbble [3] позиціонує себе як спільнота для дизайнерів та креативних спеціалістів, що фокусується на так званих «шотах» – невеликих прев'ю робіт, які демонструють фрагменти інтерфейсів, іконок, ілюстрацій або анімацій. Цей

формат дозволяє швидко отримувати відгуки на ранніх етапах розробки концепцій і відстежувати тенденції у дизайні. Dribbble пропонує окремий розділ вакансій і можливість залучення клієнтів через відкриті запити на проєкти. З іншого боку, обмежений обсяг інформації на сторінках «шотів» і запрошувальна система (invite-only для деяких типів акаунтів) можуть ускладнити доступ нових користувачів та створити бар’єр входу для тих, хто хоче розгорнути повноцінне портфоліо;

– Adobe Portfolio [4] інтегрується з обліковим записом Adobe Creative Cloud і дає змогу швидко генерувати портфоліо на основі професійно розроблених шаблонів. Підтримка імпорту з Lightroom і Behance автоматично оновлює галерею робіт без необхідності ручного додавання зображень, а вбудовані інструменти SEO і адаптивний дизайн забезпечують хорошу видимість і зручність перегляду на різних пристроях. Однак Adobe Portfolio не передбачає розширень через сторонні RESTful API, а гнучкість у налаштуванні макетів обмежена рамками готових тем, що може бути недостатньо для фахівців, які прагнуть створити унікальне брендування свого портфоліо.

Хоча всі перераховані сервіси надають швидкий старт і охоплюють великі спільноти, вони не вирішують задачі комплексного управління даними користувачів, гнучкої кастомізації бізнес-логіки та глибокої аналітики відвідувань і взаємодій. Саме це зумовлює необхідність створення нової платформи, яка б об’єднала переваги готових рішень і одночасно відкривала користувачеві повний контроль над власними даними, розширювані інтеграції та інструменти аналітики.

З огляду на це, існує обґрунтована потреба у розробці нової платформи для управління персональним портфоліо, яка б поєднувала гнучкість і масштабованість, забезпечуючи інтуїтивно зрозумілий інтерфейс та можливості для глибокої кастомізації. Використання сучасних технологій, таких як React для фронтенду, Node.js для серверної частини та MySQL для збереження даних, дозволяє створити ефективну, швидку та надійну систему. Така платформа зможе вирішити проблеми існуючих аналогів, зокрема обмеження в дизайні,

складність інтеграцій та високу вартість користування, що робить її особливо актуальною для фахівців, які прагнуть мати персоналізований, професійний та доступний інструмент для презентації своїх досягнень і проєктів.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

У процесі виконання кваліфікаційної роботи бакалавра:

- провести аналіз існуючих рішень на ринку та оцінити їхні переваги й недоліки;
- визначити функціональні та нефункціональні вимоги до розроблюваної платформи та спроектувати її архітектуру;
- обрати оптимальний технологічний стек і засоби для реалізації клієнтської та серверної частин;
- розробити структуру бази даних у MySQL і забезпечити її нормалізацію;
- реалізувати серверну логіку на Node.js з RESTful API та обробкою JWT-автентифікації;
- створити клієнтський інтерфейс на React з реактивними компонентами, маршрутизацією та управлінням станом;
- провести тестування різними методами та налагодити інформаційно-комп’ютерну систему.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Основною ціллю розроблюваної платформи для управління персональним портфоліо є створення комплексної веб-платформи, який дозволить користувачам ефективно презентувати свої професійні досягнення, навички та досвід роботи у зручному та структурованому форматі. Система має забезпечити можливість створення індивідуалізованих портфоліо для фахівців різних галузей IT із підтримкою мультимедійного контенту та інтерактивних елементів.

Ключові задачі платформи включають:

– надання користувачам інструментів для створення та редагування персональної інформації, управління проектами з детальним опис, зображенням та посиланнями на GitHub та на демо-сайт;

– додавання та відображення професійних навичок із можливістю оцінки рівня володіння;

– додавання та відображення трудового досвіду з хронологічним відображенням кар’єрного шляху;

– збирання та відображення відгуків від колег та клієнтів.

Система також має забезпечувати можливість публічного доступу до портфоліо через унікальні посилання та підтримку соціальної взаємодії між користувачами.

На основі цілей та задач були створені функціональні та нефункціональні вимоги. Функціональні вимоги системи включають реєстрацію та авторизацію користувачів, управління особистим профілем, створення та редагування проектів з підтримкою мультимедіа та категоризації, додавання та відображення навичок з оцінкою рівня володіння, документування професійного досвіду в хронологічному порядку, а також функціонал залишення відгуків користувачами.

А нефункціональні вимоги включають в себе підтримку мобільних пристроїв з повноцінною функціональністю на смартфонах та планшетах, безпеку, яка забезпечується через надійне шифрування паролів та конфіденційної інформації користувачів, комплексний захист від SQL-ін'єкцій, XSS-атак та інших поширених загроз веб-безпеки.

На основі цілей та задач, а також вимог, для проектування платформи було використано UML (Unified Modelling Language), це універсальна мова моделювання, що використовується для графічного опису та відображення систем та процесів [5]. Діаграма класів, яка зображена на рисунку 2.1 розкриває структуру системи, яка складається з шести основних сутностей, що взаємодіють між собою через чітко визначені зв'язки і описують базу даних.

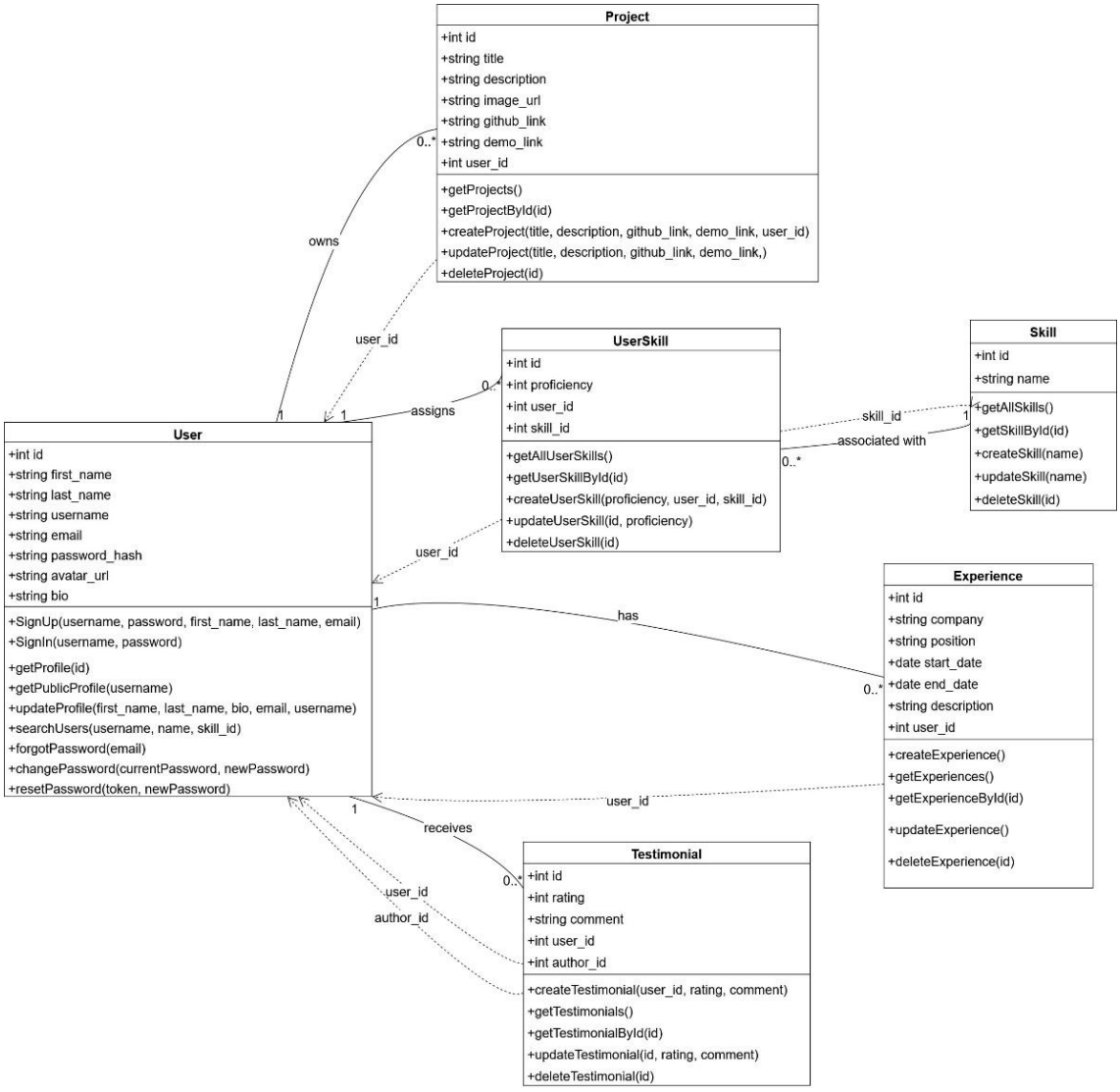

Рисунок 2.1 – Діаграма класів

Центральною сутністю є клас User, який представляє зареєстрованих користувачів системи та містить основні атрибути для ідентифікації та аутентифікації, включаючи унікальний ідентифікатор, ім’я, прізвище, ім’я користувача, електронну адресу, хешований пароль, URL аватара та біографію.

Клас Project представляє проекти користувачів і містить детальну інформацію про кожну роботу, включаючи назву, опис, зображення, посилання на репозиторій та демо-версію. Зв’язок «один до багатьох» між User та Project відображає можливість кожного користувача мати множину проектів, при цьому кожен проект належить конкретному користувачеві.

Управління навичками реалізовано через два взаємопов’язані класи: Skill та UserSkill. Клас Skill містить довідкову інформацію про доступні навички, тоді як UserSkill представляє проміжну сутність, що зберігає інформацію про рівень володіння конкретною навичкою конкретним користувачем. Такий підхід дозволяє уникнути дублювання назв навичок та забезпечує нормалізацію бази даних.

Клас Experience представляє професійний досвід користувачів з атрибутами для зберігання назви компанії, посади, дат початку та закінчення роботи, а також детального опису діяльності.

Зв’язок з класом User дозволяє кожному користувачеві мати множину записів про досвід роботи.

Функціональність відгуків реалізована через клас Testimonial, який зберігає оцінки та коментарі від інших користувачів. Особливістю цієї сутності є наявність двох зв’язків з класом User: один представляє отримувача рекомендації, а інший – її автора.

Діаграма варіантів використання демонструє взаємодію двох основних акторів системи – «Гостя» та «Авторизованого користувача» – з функціональними можливостями платформи управління портфоліо.

Ця діаграма зображена на рисунку 2.2.

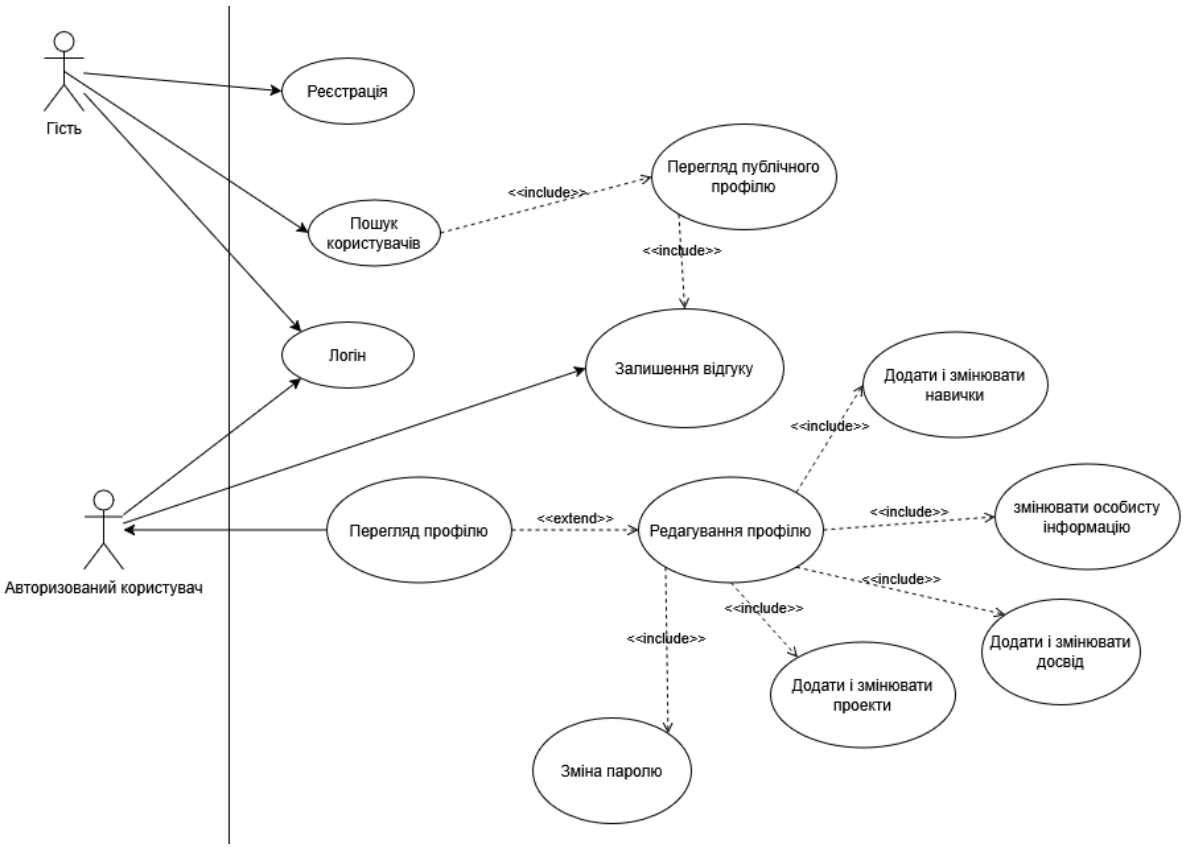

Рисунок 2.2 – Діаграма варіантів використання

«Гість» має обмежений набір функцій, включаючи можливість реєстрації в системі, пошук зареєстрованих користувачів та перегляд публічних профілів. Після успішної авторизації користувач отримує доступ до розширеного функціоналу системи.

«Авторизований користувач» може виконувати логін в систему та отримувати повний доступ до управління своїм портфоліо. Основний варіант використання «Перегляд профілю» розширюється функціональністю «Редагування профілю», яка включає декілька підваріантів: зміну особистої інформації, додавання та зміну навичок, управління професійним досвідом, а також додавання та модифікацію проектів. Окремо виділено функціональність зміни пароля як важливий аспект безпеки облікового запису.

Система передбачає також можливість залишення відгуків між користувачами, що реалізовано через варіант використання «Залишення відгуку». Така структура забезпечує комплексний підхід до управління професійним портфоліо з урахуванням різних рівнів доступу користувачів.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Вибір технологічного стеку для розробки платформи управління персональним портфоліо ґрунтувався на аналізі сучасних інструментів, їхньої відповідності вимогам до продуктивності, масштабованості та зручності використання. Ключовими критеріями стали: підтримка кросплатформності, наявність активного співтовариства та можливість гнучкої кастомізації. На основі цих вимог були обрані наступні інструменти:

– редактор коду Visual Studio Code обраний як основний інструмент розробки через його збалансований підхід між легкістю використання та потужними функціями. Він надає вбудовану підтримку JavaScript і TypeScript, що критично для роботи з React і Node.js, а також інтеграцію з Git для контролю версій. Його модульна архітектура дозволяє розширювати функціонал за рахунок плагінів, таких як ESLint для лінтингу коду та Prettier для автоматичного форматування. Незважаючи на необхідність ручних налаштувань для підключення зовнішніх бібліотек, редактор забезпечує швидкий цикл редагування-тестування завдяки вбудованому терміналу та інтерактивному дебагеру. На рисунку 2.3 зображений його зовнішній вигляд;

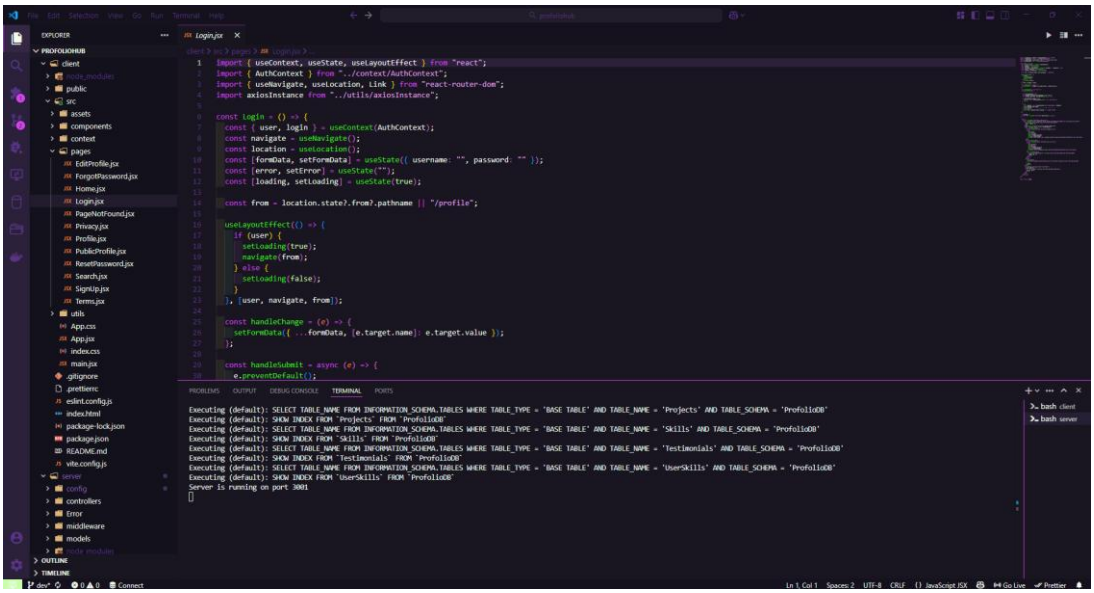

Рисунок 2.3 – Зовнішній вигляд VS Code

– серверна частина на базі Express.js (Node.js) обрана через її мінімалістичний підхід і високу продуктивність у обробці HTTP-запитів [6]. За його допомогою можна створювати прості API-ендпоінти для маршрутизації, що дозволяє ефективно керувати даними портфоліо. Використання проміжного ПЗ (middleware) дозволяє реалізувати автентифікацію через JWT-токени, валідацію вхідних даних та обробку помилок. Хоча фреймворк не надає вбудованих рішень для ORM або шаблонізаторів, цей недолік компенсується можливістю підключення бібліотек на кшталт Sequelize [7] для роботи з MySQL. Асинхронна модель Node.js забезпечує обробку до 10,000 одночасних з'єднань, що критично для масштабування системи;

– клієнтський інтерфейс на React [8] обраний завдяки компонентному підходу, який дозволяє створювати перевикористовувані UI-елементи для різних розділів портфоліо. Virtual DOM забезпечує оптимізацію оновлення інтерфейсу при зміні даних, що особливо важливо для динамічних сторінок з великою кількістю інтерактивних елементів. Використання React-хуків (useState, useEffect) дозволяє керувати локальним станом компонентів без необхідності підключення додаткових бібліотек управління станом, а react-router-dom забезпечує клієнтську маршрутизацію без повного перезавантаження сторінок. Для стилізації інтерфейсу застосовано TailwindCSS – утилітарно-орієнтований CSS фреймворк, який дозволяє швидко будувати адаптивні сторінки та легко налаштовувати їх дизайн за допомогою класів утиліт [9]. Це значно прискорює розробку UI і забезпечує консистентність стилів по всій платформі;

– система керування базами даних MySQL [10] обрана для зберігання структурованої інформації про користувачів, проєкти, навички та відгуки завдяки її перевіреній реляційній моделі, високій продуктивності та широкій підтримці інструментів адміністрації. Використання MySQL дозволяє забезпечити цілісність даних через транзакції, налагодити складні SQL-запити для фільтрації та агрегації.

РОЗДІЛ 3
РОЗРОБКА ПЛАТФОРМИ ДЛЯ УПРАВЛІННЯ ПЕРСОНАЛЬНИМ ПОРТФОЛІО

3.1 Практична реалізація об’єкта проектування

Практична реалізація платформи керування персональним портфоліо побудована за клієнт-серверною архітектурою, що дозволяє розділити відповідальність між компонентами й забезпечити високу гнучкість розвитку. Серверна частина створена на Node.js із використанням Express.js, що полегшує побудову RESTful API [11] та інтеграцію з базою даних MySQL через ORM Sequelize. Клієнтська частина реалізована на React, де для управління станом застосовані функціональні компоненти та React Hooks.

Розробку серверної частини було розпочато зі створення структури проєкту та підключення необхідних залежностей: Express.js відповідає за маршрутизацію, mysql2 та sequelize – за роботу з базою даних, jsonwebtoken – за генерацію й верифікацію JWT-токенів, а bcrypt – за безпечне хешування паролів користувачів, а express-fileupload для завантаження статичних зображень, які надходять із клієнта. Першим кроком у коді було налаштування з’єднання з MySQL через конфігурацію Sequelize. Саме там вказані параметри хоста, порту, назви бази та облікових даних, а також оптимальні налаштування розміру пулу з’єднань. Це рішення дозволяє уникнути надмірного відкриття/закриття сполучень при високих навантаженнях та забезпечити стабільність роботи сервера. Усе це підключено до файлу index.js, який зображено на лістингу 3.1.

Лістинг 3.1 – Файл index.js сервера

```
require('dotenv').config();
const express = require('express');
const app = express();
const path = require('path');
const fileUpload = require('express-fileupload');
const db = require('./models');
const cors = require('cors');
```

```
const ErrorHandler =
require('./middleware/ErrorHandlingMiddleware');
app.use(express.json());
app.use(express.static(path.resolve(__dirname, 'static')))
app.use(fileUpload({}))
app.use(cors());

// routers
const projectsRouter = require('./routes/Projects');
app.use('/api/projects', projectsRouter);
const usersRouter = require('./routes/Users');
app.use('/api/user', usersRouter);
const ExperiencesRouter = require('./routes/Experiences');
app.use('/api/Experiences', ExperiencesRouter);
const TestimonialsRouter = require('./routes/Testimonials');
app.use('/api/Testimonials', TestimonialsRouter);
const SkillsRouter = require('./routes/Skills');
app.use('/api/Skills', SkillsRouter);
const UserSkillsRouter = require('./routes/UserSkill');
app.use('/api/UserSkills', UserSkillsRouter);

// Error handling middleware
app.use(ErrorHandler);

db.sequelize.sync({ force: false }).then(() => {
  app.listen(3001, () => {
    console.log('Server is running on port 3001');
  });
});
```

кінець лістингу 3.1

Після успішної ініціалізації Sequelize наступним етапом стала розробка моделей (ORM-класів) для кожної сутності: Users, Projects, Experiences, Skills, UserSkills та Testimonials. У кожній моделі задекларовані атрибути (назви полів таблиці) з відповідними типами даних, перевірками (унікальність, allowNull, довжина рядків) та описаними зв'язками між таблицями за допомогою методів hasMany, belongsTo і belongsToMany. Наприклад, вираз User.hasMany(Project) у поєднанні з Project.belongsTo(User) гарантує, що кожний запис у таблиці Projects міститиме зовнішній ключ user_id і що видалення користувача призведе до каскадного видалення його проектів.

Для захисту та зручної обробки запитів застосовано дві групи middleware. Перший – Error Handler виконує критично важливу роль у забезпеченні

стабільності та надійності роботи додатка, реалізуючи централізований механізм обробки помилок на рівні сервера. Цей компонент перехоплює всі непередбачені виключення, які можуть виникнути під час виконання запитів, незалежно від того, в якій частині додатка вони з'явилися. Завдяки централізованому підходу до обробки помилок, ErrorHandler забезпечує консистентність у відповідях сервера та запобігає неконтрольованому завершенню роботи додатка. Middleware автоматично форматує всі помилки у стандартизований JSON-формат, який містить відповідний HTTP-код статусу та зрозуміле повідомлення про помилку, що значно спрощує процес діагностики проблем як для розробників, так і для кінцевих користувачів. Його код зображений у лістингу 3.2.

Лістинг 3.2 – Універсальний обробник API-помилок

```
const ApiError = require('../Error/ApiError');
module.exports = function (err, req, res, next) {
    if (err instanceof ApiError) {
        return res.status(err.status).json({ message: err.message });
    }
    return res.status(500).json({ message: 'Невідома помилка' });
}
```

кінець лістингу 3.2

Власний клас ApiError служить основою для створення структурованих виключень у межах додатка та забезпечує уніфікований підхід до обробки різних типів помилок. Цей клас розширює стандартний клас Error JavaScript, додаючи додаткові властивості та методи, які дозволяють точно категоризувати та описувати помилки, що виникають під час роботи API. Кожен екземпляр ApiError містить не лише текстове повідомлення про помилку, але й відповідний HTTP-код статусу, що дозволяє клієнтським додаткам правильно інтерпретувати тип помилки та реагувати відповідним чином. Як показано у лістингу 3.3, використання цього класу забезпечує повернення клієнту чітко структурованої JSON-відповіді з усією необхідною інформацією для обробки помилкової ситуації.

Лістинг 3.3 – Клас ApiError

```
class ApiError extends Error {
    constructor(status, message) {
        super();
        this.status = status;
        this.message = message
    }

    static badRequest(message) {
        return new ApiError(400, message);
    }

    static unauthorized(message) {
        return new ApiError(401, message);
    }

    static forbidden(message) {
        return new ApiError(403, message);
    }

    static notFound(message) {
        return new ApiError(404, message);
    }

    static internalServerError(message) {
        return new ApiError(500, message);
    }
}

module.exports = ApiError
```

кінець лістингу 3.3

Другий – Auth Middleware – перевіряє наявність і валідність JWT-токена в заголовку Authorization, декодує корисне навантаження, отримуючи об’єкт користувача, і додає його до запиту, що забезпечує захист приватних маршрутів та можливість контролю доступу на рівні бізнес-логіки (ліст. 3.4).

Лістинг 3.4 – Middleware для верифікації jwt токену

```
const jwt = require(‘jsonwebtoken’);

module.exports = function (req, res, next) {
    if (req.method === ‘OPTIONS’) {
        return next();
```

```
    }
    try {
          const token = req.headers.authorization.split(‘ ‘)[1]; //
Bearer TOKEN
        if (!token) {
            return res.status(401).json({ message: ‘Не авторизований’
});
        }
        const decoded = jwt.verify(token, process.env.JWT_SECRET);
        req.user = decoded;
        next();
    } catch (error) {
        return res.status(401).json({ message: ‘Не авторизований’ });
    }
}
```

кінець лістингу 3.4

Далі було розроблено контролери для API. Вони представляють собою комплексну систему управління портфоліо користувачів з розгалуженою архітектурою маршрутів.

Основою системи є користувацька аутентифікація, яка реалізована через JWT токен [12] та включає повний цикл роботи з обліковими записами. Маршрути користувачів починаються з базових операцій реєстрації та входу через /api/user/register та /api/user/login, де система перевіряє унікальність username та email, хешує паролі за допомогою bcrypt та генерує JWT токени з терміном дії 24 години.

Особливістю системи аутентифікації є наявність механізму відновлення паролів, що включає маршрути /api/user/forgot-password та /api/user/reset-password, які використовують email-сервіс Nodemailer [13] для надсилання токенів відновлення з обмеженим терміном дії в одну годину. Додатково реалізований маршрут /api/user/change-password для аутентифікованих користувачів, що дозволяє змінювати пароль після перевірки поточного. Система профілів користувачів організована навколо двох ключових маршрутів: /api/user/profile для отримання та оновлення власного профілю і /api/user/public/:username для публічного перегляду профілів інших користувачів, що забезпечує приватність даних через виключення password_hash з відповідей.

Функціональність пошуку користувачів реалізована через маршрут /api/user/search з підтримкою фільтрації за username, ім’ям та навичками, що інтегрується з системою навичок користувачів. Управління навичками здійснюється через два взаємопов’язані модулі: базові навички через /api/Skills та зв’язки користувачів з навичками через /api/UserSkills. Система навичок дозволяє створювати унікальні навички з валідацією на рівні бази даних, а зв’язки користувач-навичка включають показник profiency від 0 до 100 відсотків, що дає можливість кількісно оцінювати рівень володіння навичками. Усі ці особливості контролера для користувача зображені в коді в додатку А.

Управління проектами реалізована через маршрути /api/projects і підтримує повний CRUD (Create, Read, Update, Delete) [14] функціонал з особливістю обробки файлових завантажень для зображень проектів. Система використовує express-fileupload для обробки зображення та зберігає їх в статичній папці з унікальними іменами, згенерованими через UUID. При оновленні проектів система зберігає існуючий шлях до зображення, якщо нове не завантажується, що забезпечує стабільність даних. Аналогічно працює система досвіду роботи через /api/Experiences, яка дозволяє користувачам додавати інформацію про свій професійний досвід з датами початку та закінчення, причому end_date може бути null для поточних позицій.

Система відгуків і рекомендацій реалізована через маршрути /api/Testimonials з унікальною логікою, що запобігає залишенню відгуків самому собі через перевірку user_id проти author_id з JWT токена. Відгуки включають рейтинг від 1 до 5 та текстові коментарі, створюючи двостороннє посилання між автором та отримувачем відгуку. Всі маршрути, що вимагають аутентифікації, захищені AuthMiddleware, який перевіряє JWT токени та додає інформацію про користувача до req.user.

Сервер підтримує CORS для взаємодії з фронтенд додатками та використовує express-static для обслуговування статичних файлів, що дозволяє прямий доступ до завантажених зображень. База даних синхронізується із сервером через Sequelize ORM з опцією force: false, що зберігає існуючі дані при

перезапуску сервера, а всі моделі містять timestamps для відстеження часу створення та оновлення записів. Саме через ці API клієнтська частина взаємодіє із серверною частиною.

Далі була написана клієнтська частина з використанням функціональних компонентів та хуків для управління станом. Компоненти і сторінки пишуться через JSX. Це розшифровується як JavaScript XML і її особливістю є те, що можна писати розмітку і код у файл формату .jsx. Для HTTP запитів до серверної частини використовується бібліотека Axios [15], налаштована з автоматичним додаванням JWT токену до заголовків через interceptors. Створено окремий сервіс API для інкапсуляції всіх HTTP запитів з методами для кожного ресурсу системи.

Головна сторінка платформи, доступна навіть без авторизації, одразу знайомить відвідувача з ключовими можливостями сервісу й мотивує створити власне портфоліо. Великий заголовок, що закликає презентувати досягнення світові, супроводжується інтерактивними кнопками: для нових гостей доступна реєстрація, а для тих, хто вже зареєстрований, – перехід безпосередньо до персонального профілю. Далі знаходиться лаконічний опис місії проекту – допомогти фахівцям сформувати сильний особистий бренд – а під ним розташовано чотири яскраві картки з іконками, які пояснюють основні функціональні можливості: від створення та редагування портфоліо до інтеграцій із зовнішніми сервісами, аналітики відвідувань і збору відгуків

Сторінка профілю користувача, яка зображена на рисунку 3.1 та рисунку 3.2 представляє собою комплексний інтерфейс, який забезпечує детальне відображення всієї необхідної інформації про користувача системи. Дана сторінка структурована таким чином, щоб охопити всі ключові аспекти користувацького профілю, включаючи особисті дані (ім'я, контактну інформацію, фотографію), детальний список реалізованих проектів, професійні навички та компетенції, а також повний професійний досвід користувача. Така організація інформації дозволяє отримати цілісне уявлення про користувача та його професійні здобутки в одному місці.

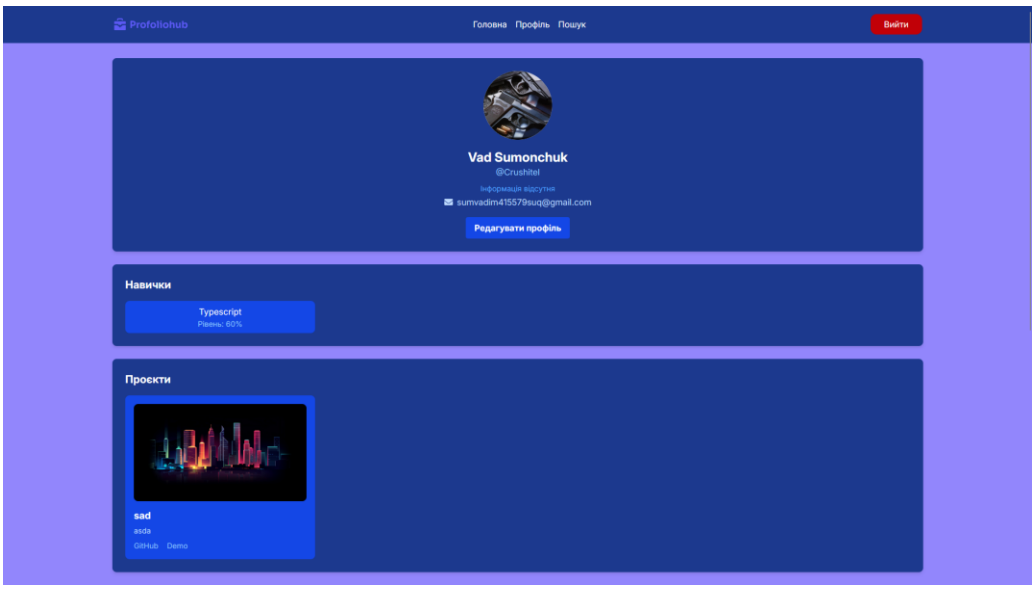

Рисунок 3.1 – Сторінка профілю користувача

З технічної точки зору, компонент сторінки профілю реалізований із використанням сучасних підходів React-розробки, зокрема використовує хук useEffect для ефективного завантаження даних профілю в момент монтування компонента до DOM. Це забезпечує оптимальну продуктивність та користувацький досвід, оскільки дані завантажуються автоматично при відкритті сторінки. Особливу увагу приділено візуальному представленню проектів користувача – вони організовані у вигляді адаптивної сітки карток, що забезпечує зручний перегляд та навігацію між різними проектами, дозволяючи користувачам швидко оцінити масштаб та різноманітність виконаних робіт.

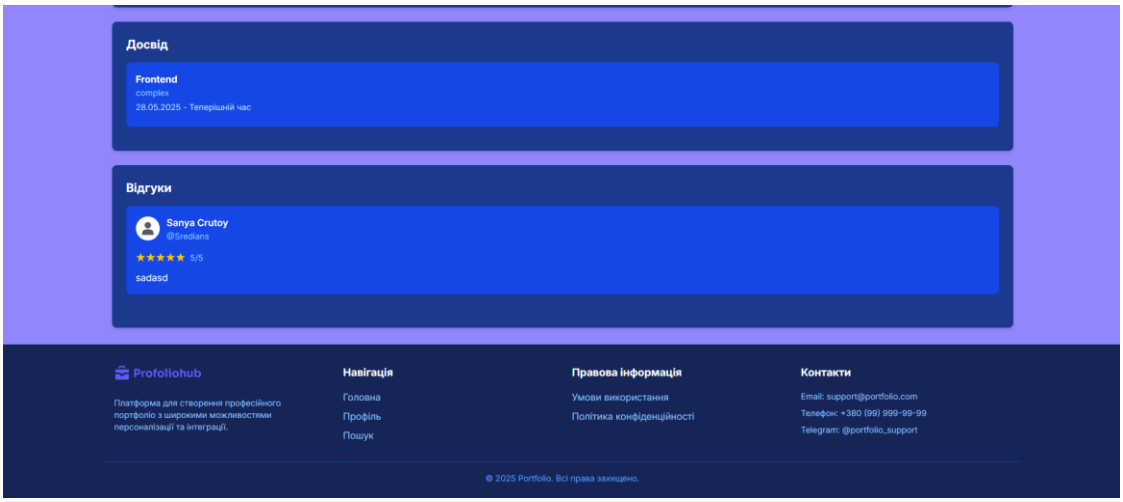

Рисунок 3.2 – Секції досвід і відгуки в профілі користувача

Сторінка налаштування профілю, яка зображена на рисунку 3.3 розбита по категоріях, що дозволяє користувачу редагувати особисту інформацію, змінювати аватар, змінювати пароль додавати і видаляти проекти, досвід, навички.

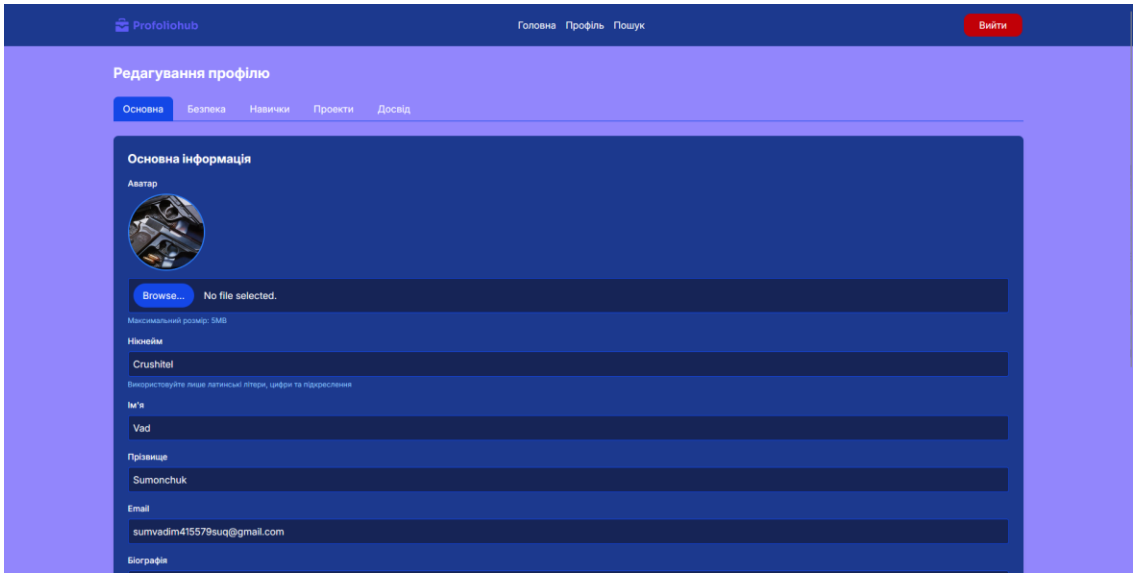

Рисунок 3.3 – Сторінка редагування профілю

Форми на цій сторінці реалізовані з валідацією на клієнтській стороні. Завантаження файлів обробляється через FormData API.

Сторінка пошуку користувачів, представлена на рисунку 3.4, являє собою комплексний інструмент для ефективного пошуку та фільтрації учасників системи. Ця функціональність забезпечує користувачам широкі можливості для точного відбору потрібних профілів завдяки реалізації багаторівневої системи фільтрації.

Основними критеріями пошуку виступають персональні дані користувачів, включаючи ім'я, прізвище та унікальний нікнейм, що дозволяє швидко знаходити конкретних осіб у базі даних. Особливо цінною є можливість фільтрації за професійними навичками та компетенціями, що надає змогу користувачам відбирати кандидатів або колег відповідно до специфічних вимог проекту чи завдання.

Інтерфейс сторінки розроблений з урахуванням принципів зручності користування, забезпечуючи інтуїтивно зрозумілу навігацію та логічне розташування елементів управління. Система пошуку підтримує як точний збіг параметрів, так і часткове співпадіння, що значно розширює можливості пошуку та підвищує ймовірність знаходження релевантних результатів.

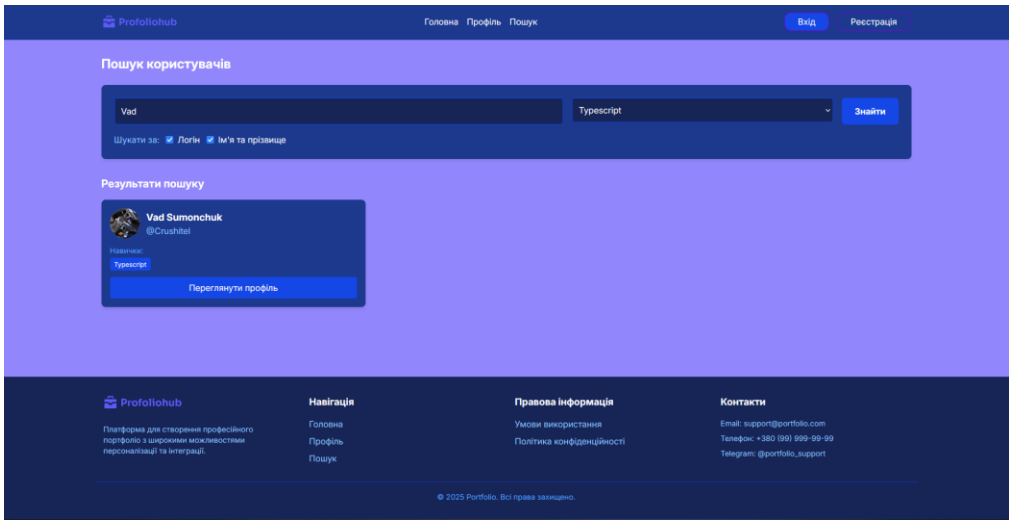

Рисунок 3.4 – Сторінка пошуку користувачів

Компонент містить форму пошуку з drop-down списком в якому можна вибрати за якою навичкою слід шукати користувача і ці навички підванатажуються через API (ліст. 3.5).

Лістинг 3.5 – Фрагмент коду для підвантаження навичок у фільтр

```
// Завантаження списку навичок для фільтрування
  useEffect(() => {
    const fetchSkills = async () => {
      try {
        const response = await axiosInstance.get('/skills');
        setSkills(response.data || []);
      } catch (err) {
        console.error('Помилка завантаження навичок:', err);
      }
    };

    fetchSkills();
  }, []);
```

кінець лістингу 3.5

При натисканні кнопки «знайти» результати пошуку завантажуються через API та відображаються у вигляді списку або сітки з можливістю переходу до публічного профілю користувача. Фрагмент коду відображений на лістингу 3.6.

Лістинг 3.6 – Фрагмент коду для пошуку користувачів

```
const handleSearch = async () => {
    if (!searchTerm && !selectedSkill) {
      setError('Введіть пошуковий запит або виберіть навичку');
      return;
    }
    setError('');
    setLoading(true);
    try {
      // Формуємо параметри запиту
      const params = new URLSearchParams();

      if (searchTerm) {
        if (filters.username) params.append('username', searchTerm);
        if (filters.name) params.append('name', searchTerm);
      }

      if (selectedSkill) {
        params.append('skill_id', selectedSkill);
      }

      // Виконуємо запит до API
      const response = await
axiosInstance.get(`/user/search?${params.toString()}`);
      setSearchResults(response.data);

      if (response.data.length === 0) {
        setError('Користувачів не знайдено');
      }
    } catch (err) {
      console.error('Помилка пошуку:', err);
      setError('Сталася помилка під час пошуку. Спробуйте
пізніше.');
      setSearchResults([]);
    } finally {
      setLoading(false);
    }
  };
```

кінець лістингу 3.6

Сторінка публічного профілю призначена для перегляду портфоліо іншими користувачами та гостями системи. Вона так само містить інформацію

як у профілі, але в ця сторінка включає можливість залишення відгуків для авторизованих користувачів через форму оцінки та коментаря, яка зображена на рисунку 3.5.

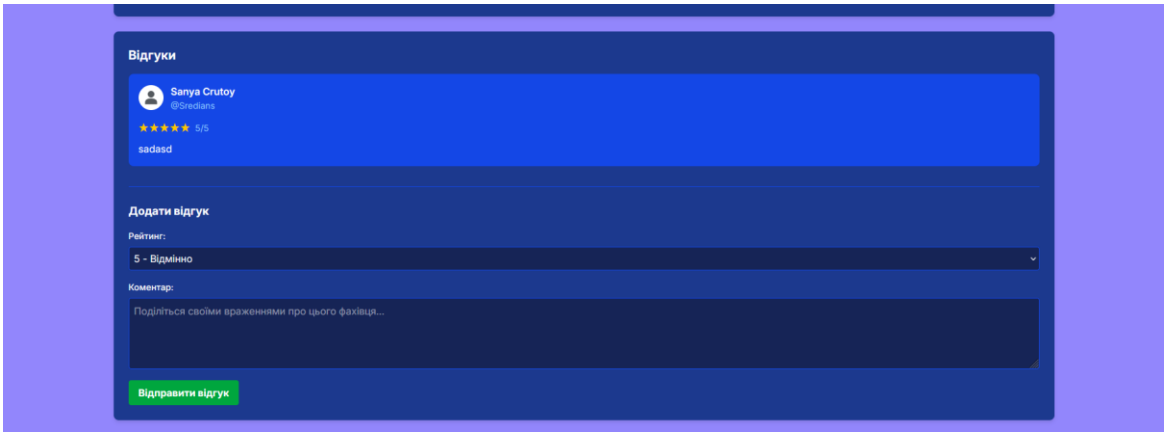

Рисунок 3.5 – Форма для залишення відгуку

Сторінка входу, яка зображена на рисунку 3.6 містить форму аутентифікації з полями для email та пароля, валідацією даних та обробкою помилок авторизації. При успішному вході JWT токен зберігається в localStorage, а користувач перенаправляється на головну сторінку. Компонент включає посилання на сторінку реєстрації та відновлення паролю.

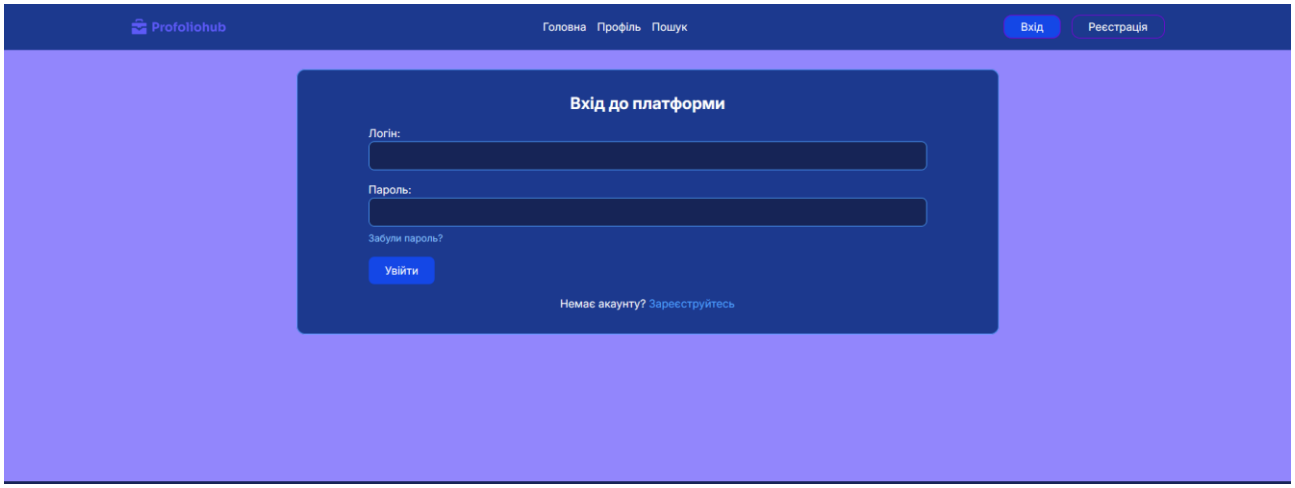

Рисунок 3.6 – Сторінка входу

Сторінка реєстрації на рисунку 3.7 реалізована з розширеною формою, що включає поля для імені, прізвища, email, нікнейму та пароля. Валідація включає перевірку унікальності email та нікнейму через API запити. При успішній реєстрації користувач автоматично перенаправляється на сторінку входу.

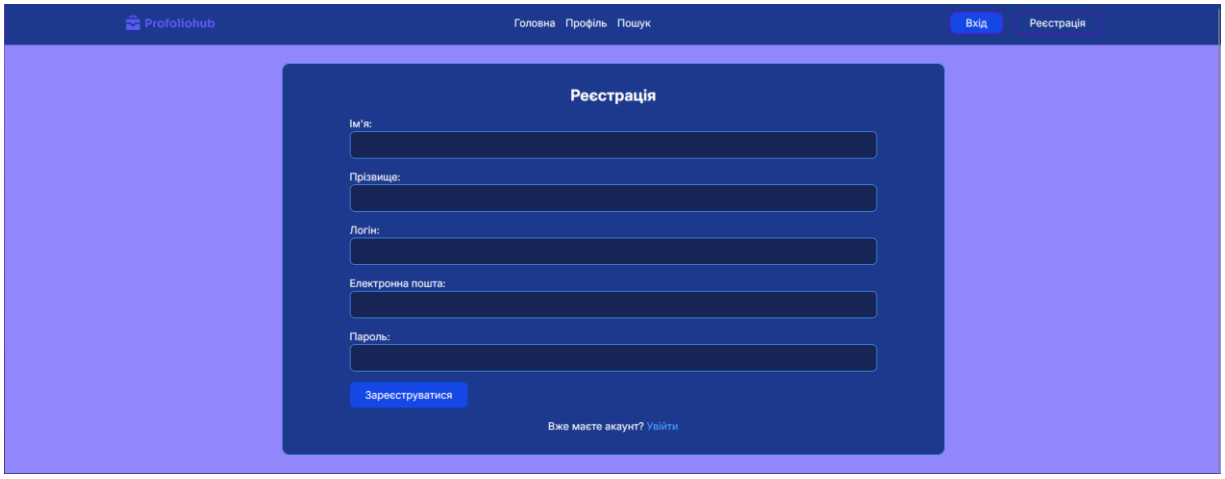

Рисунок 3.7 – Сторінка реєстрації

Сторінка «Забули пароль» містить форму для введення email адреси з відправкою запиту на сервер для генерації токену відновлення. Компонент обробляє різні сценарії відповіді сервера та відображає відповідні повідомлення користувачеві. Після відправки email користувач інформується про наступні кроки процесу відновлення.

Сторінка відновлення паролю активується через посилання з email та містить форму для введення нового пароля. Компонент валідує токен відновлення через API запит та дозволяє встановити новий пароль з відповідною валідацією складності. Після успішної зміни пароля користувач перенаправляється на сторінку входу з підтвердженням операції.

Маршрутизація реалізована через React Router з захищеними маршрутами для приватних сторінок. Компонент ProtectedRoute перевіряє наявність валідного JWT токену та перенаправляє неавторизованих користувачів на сторінку входу. Якщо JWT токену немає, то перенаправляє користувача Глобальний стан користувача управляється через Context API з провайдером, що

забезпечує доступ до інформації про аутентифікацію в усіх компонентах додатку.

3.2 Розробка бази даних

Проєкт використовує MySQL як основну реляційну систему керування базами даних, оскільки вона поєднує високу продуктивність із перевіреною надійністю та широкою екосистемою інструментів адміністрування. Архітектура даних відображена у діаграмі класів й включає шість взаємопов'язаних таблиць: Users, Projects, Skills, UserSkills, Experiences та Testimonials, кожна з яких відіграє чітко визначену роль у загальній моделі портфоліо. Ці таблиці зображені на схемі бази даних на рисунку 3.8.

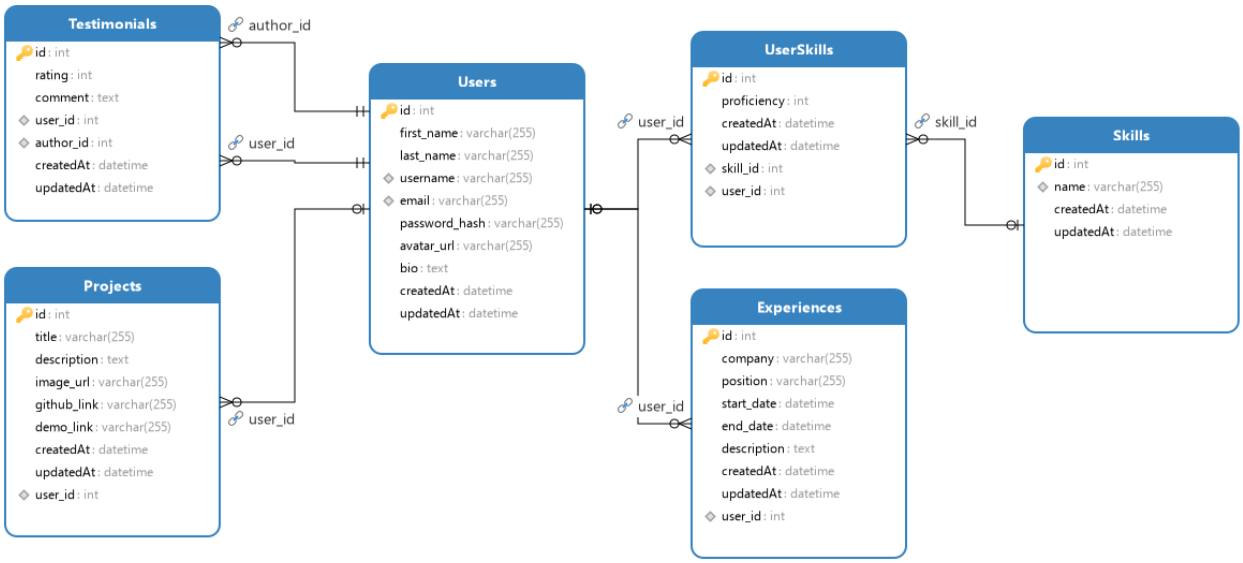

Рисунок 3.8 – Схема бази даних

Центральна таблиця Users містить повний набір інформації про користувачів системи, починаючи з поля id типу int як первинного ключа, який забезпечує унікальну ідентифікацію кожного облікового запису. Персональна інформація зберігається у полях first_name та last_name типу varchar(255), які є обов'язковими для заповнення та містять ім'я і прізвище користувача відповідно.

Поле username типу varchar(255) виступає унікальним ідентифікатором для входу в систему і має відповідне обмеження унікальності, а email також типу varchar(255) зберігає електронну адресу з валідацією формату та унікальним індексом для швидкого пошуку. Безпека забезпечується полем password_hash типу varchar(255), яке містить хешований пароль, створений за допомогою алгоритму bcrypt. Додаткова інформація профілю включає avatar_url типу varchar(255) для посилання на зображення користувача та bio типу text для розгорнутого опису діяльності чи інтересів. Системні поля createdAt та updatedAt типу datetime автоматично відстежують час створення облікового запису та останнього оновлення інформації.

Таблиця Projects призначена для зберігання портфоліо робіт користувачів і містить поле id типу int як первинний ключ з автоматичним інкрементом. Основна інформація про проект зберігається у полі title типу varchar(255) для назви проекту та description типу text для детального опису функціоналу, технологій та особливостей реалізації. Візуальне представлення забезпечується полем image_url типу varchar(255), яке містить посилання на превью, скриншот або логотип проекту. Посилання на репозиторій зберігається у полі github_link типу varchar(255), а demo_link того ж типу містить URL робочої демонстрації проекту. Зв’язок з автором реалізований через поле user_id типу int, яке виступає зовнішнім ключем до таблиці Users і забезпечує референтну цілісність даних. Системні поля createdAt та updatedAt типу datetime відстежують час публікації проекту та останніх змін до його опису.

Таблиця Skills містить перелік доступних навичок і технологій з полем id типу int як первинний ключ та name типу varchar(255) для зберігання назви навички з унікальним обмеженням, що запобігає дублюванню записів. Поля createdAt та updatedAt типу datetime дозволяють відстежувати час додавання нових навичок до системи та їх модифікацію адміністраторами.

Проміжна таблиця UserSkills реалізує відношення багато до багатьох між користувачами та навичками, маючи складений первинний ключ з полів user_id та skill_id типу int, які одночасно виступають зовнішніми ключами до

відповідних таблиць. Унікальність цієї моделі полягає у полі proficiency типу int, яке зберігає числове значення рівня володіння навичкою від 0 до 100 відсотків, дозволяючи користувачам точно вказати свою компетентність у кожній технології. Системні поля createdAt та updatedAt типу datetime фіксують час додавання навички до профілю та останнього оновлення рівня володіння.

Таблиця Experiences призначена для ведення професійного резюме користувачів з полем id типу int як первинний ключ з автоматичним інкрементом. Інформація про роботодавця зберігається у полі company типу varchar(255), а конкретна посада у полі position того ж типу, обидва поля є обов'язковими для заповнення. Хронологічні рамки роботи фіксуються через поля start_date та end_date типу datetime, при цьому поле завершення може залишатися порожнім для поточного місця роботи. Детальний опис виконуваних обов'язків, досягнень та ключових проектів зберігається у полі description типу text. Зв'язок з користувачем забезпечується полем user_id типу int як зовнішнім ключем до таблиці Users. Поля createdAt та updatedAt типу datetime відстежують час додавання запису про досвід роботи та його останні зміни.

Таблиця Testimonials реалізує систему відгуків між користувачами з полем id типу int як первинний ключ з автоматичним інкрементом. Оцінка зберігається у полі rating типу int з валідацією діапазону від 1 до 5, що відповідає п'ятизірковій системі оцінювання, а текстовий відгук у полі comment типу text може залишатися порожнім у випадку, коли користувач обмежується лише числовою оцінкою. Ключова особливість моделі полягає у двох зовнішніх ключах: user_id типу int ідентифікує користувача, якому залишається відгук, а author_id того ж типу вказує на автора рекомендації, що дозволяє розрізняти ролі учасників у процесі оцінювання. Системні поля createdAt та updatedAt типу datetime фіксують час публікації відгуку та можливих подальших редагувань.

Реляційні зв'язки між таблицями побудовані на принципах нормалізації і забезпечують референтну цілісність даних. Таблиця Users виступає центральним вузлом архітектури, маючи відношення один до багатьох з Projects, Experiences та UserSkills через відповідні зовнішні ключі user_id. Особливістю таблиці

Testimonials є подвійний зв’язок з Users через поля user_id та author_id, що дозволяє відстежувати як отримані, так і залишені користувачем рекомендації. Таблиця Skills пов’язана з Users через проміжну таблицю UserSkills, реалізуючи класичне відношення багато до багатьох з додатковим атрибутом рівня володіння.

Індексування таблиць оптимізоване для найчастіших запитів: унікальні індекси на username та email у таблиці Users прискорюють аутентифікацію та реєстрацію, індекси на зовнішніх ключах user_id у всіх залежних таблицях забезпечують швидке завантаження профільної інформації, а складений індекс на парі (user_id, skill_id) у таблиці UserSkills оптимізує запити на отримання навичок конкретного користувача. Такий підхід до індексування гарантує високу продуктивність навіть при значному зростанні кількості користувачів та їхніх даних.

Архітектура даних повністю відповідає третій нормальній формі, що мінімізує надлишковість інформації та запобігає аномаліям оновлення. Використання автоматичних часових міток у всіх таблицях забезпечує можливість аудиту змін та відновлення історичних станів даних. Типи даних обрано з урахуванням специфіки кожного поля: varchar(255) для коротких текстових значень з обмеженою довжиною, text для довгих описів без фіксованого ліміту, int для числових ідентифікаторів та оцінок, datetime для точного зберігання часових міток. Така структура забезпечує оптимальне використання дискового простору та високу швидкість виконання запитів, створюючи надійну основу для масштабування платформи портфоліо.

Взаємодія проекту з базою даних реалізована через ORM Sequelize, що забезпечує об’єктно-реляційне мапування та дозволяє працювати з базою даних у стилі JavaScript об’єктів замість написання сирих SQL запитів. Кожна таблиця бази даних представлена відповідною моделлю Sequelize з детальним описом структури полів, типів даних та валідаційних правил. Модель Users визначає поля через DataTypes.INTEGER для числових ідентифікаторів, DataTypes.STRING для текстових полів з автоматичним мапуванням на

varchar(255), DataTypes.TEXT для довгих описів та DataTypes.DATE для часових міток, при цьому кожне поле має налаштування allowNull для контролю обов'язковості заповнення.

Особливу увагу приділено валідації даних на рівні моделей, де поле email має вбудовану перевірку isEmail для забезпечення коректного формату електронної пошти, поле rating у моделі Testimonial обмежене діапазоном від 1 до 5 через validate з параметрами min та max, а поле proficiency у UserSkill має валідацію від 0 до 100 для відсоткового представлення рівня володіння навичкою. Унікальні обмеження реалізовані через параметр unique у полях username, email та name, що гарантує неповторюваність критичних ідентифікаторів на рівні ORM без необхідності додаткових перевірок у бізнес-логіці.

Асоціації між моделями визначені через методи belongsTo та hasMany, які автоматично генерують відповідні JOIN запити при завантаженні пов'язаних даних. Модель Users має hasMany зв'язки з Projects, Experiences та UserSkill через зовнішній ключ user_id, що дозволяє отримувати всі пов'язані записи одним запитом через методи include або eager loading. Складніша структура реалізована у моделі Testimonial з двома belongsTo асоціаціями до Users через різні зовнішні ключі user_id та author_id з аліасами UserProfile та Author відповідно, що дозволяє в одному запиті отримати інформацію як про отримувача відгуку, так і про його автора.

Модель UserSkill виступає класичною junction table для зв'язку багато до багатьох між Users та Skill, при цьому Sequelize автоматично обробляє складені зовнішні ключі та забезпечує цілісність даних через каскадні операції. Зворотні асоціації у моделі Users включають два окремі hasMany зв'язки з Testimonial через аліаси TestimonialsReceived та AuthoredTestimonials, що дозволяє розрізняти отримані та залишені відгуки при виконанні запитів.

Ініціалізація ORM відбувається через центральний файл index.js, який автоматично сканує директорію models, завантажує всі файли моделей та виконує їх реєстрацію у Sequelize з передачею інстансу sequelize та DataTypes.

Конфігурація підключення до бази даних зчитується з файлу config.json з урахуванням поточного оточення (development, test, production), при цьому підтримується як пряме підключення через параметри database, username, password, так і через змінні оточення для production середовища.

Автоматичне виконання асоціацій забезпечується через перебір усіх зареєстрованих моделей та виклик методу associate для кожної з них, що створює двосторонні зв'язки між моделями без необхідності дублювання коду. Timestamps автоматично додаються до моделей Projects та Testimonials через параметр timestamps: true, що створює поля createdAt та updatedAt з автоматичним оновленням при кожній операції створення або модифікації запису.

Така архітектура ORM забезпечує високий рівень абстракції над базою даних, автоматичну генерацію SQL запитів з оптимізацією для конкретної СКБД, вбудовану валідацію даних та захист від SQL-ін'єкцій через параметризовані запити. Використання async/await паттернів у поєднанні з API, Sequelize дозволяє писати чистий асинхронний код для роботи з базою даних, а автоматичне управління з'єднаннями через connection pool оптимізує продуктивність при високому навантаженні. Така структура забезпечує оптимальне використання дискового простору та високу швидкість виконання запитів, створюючи надійну основу для масштабування платформи портфоліо з можливістю легкого розширення функціоналу через додавання нових моделей та асоціацій.

3.3 Тестування та налагодження інформаційно-комп'ютерної системи

Тестування та налагодження інформаційно-комп'ютерної системи це невід'ємний етап розробки, спрямований на забезпечення належної якості, функціональності, надійності та безпеки платформи. Комплексний підхід до тестування дозволив виявити та усунути потенційні проблеми на ранніх стадіях, забезпечивши відповідність розробленого продукту визначеним вимогам.

У процесі розробки було застосовано різноманітні методи тестування такі як модульне тестування серверної частини, яке охоплювало перевірку окремих контролерів, таких як userController.js та projectsController.js, сервісів, наприклад mail-service.js, та моделей Sequelize, включаючи Users.js і Projects.js. Тестувалася коректність реалізації бізнес-логіки, обробки даних, взаємодії з ORM, генерації JWT-токенів, хешування паролів за допомогою bcrypt, а також коректність роботи проміжного програмного забезпечення, зокрема AuthMiddleware та ErrorHandlingMiddleware. На клієнтській частині модульне тестування фокусувалося на окремих React-компонентах, відповідальних за відображення UI елементів, наприклад, компонентів форм зі сторінки налаштування профілю, та утилітах, таких як конфігурація Axios в axiosInstance.js. Основна увага приділялася коректності рендерингу компонентів залежно від вхідних props та внутрішнього стану (useState), обробці подій та взаємодії з React Context API, наприклад AuthContext.

Інтеграційне тестування на серверній частині перевіряло коректність взаємодії між різними компонентами, включаючи повні сценарії обробки API-запитів – від маршрутизації Express.js до взаємодії контролерів з моделями Sequelize та базою даних MySQL, а також роботу ланцюжків middleware. На клієнтській частині тестувалася взаємодія між React-компонентами, робота клієнтської маршрутизації за допомогою react-router-dom та передача даних через Context API. Ключовим аспектом було тестування клієнт-серверної взаємодії, перевіряючи HTTP-запити з Axios, їх обробку на сервері та коректність отриманих відповідей, особливо в сценаріях автентифікації з JWT-токенами.

Функціональне тестування проводилося переважно вручну, методом «чорної скриньки», на основі визначених функціональних вимог та діаграм варіантів використання. Перевірялися ключові сценарії, такі як реєстрація та вхід, управління профілем, проектами, навичками, досвідом, пошук користувачів, перегляд публічних профілів, залишення відгуків та процедури відновлення пароля. Тестування користувацького інтерфейсу забезпечувало

коректне відображення всіх сторінок та їх елементів, відповідність дизайну, читабельність текстів та роботу інтерактивних елементів. Перевірялася також адаптивність інтерфейсу для різних пристроїв, відповідно до нефункціональних вимог. З огляду на нефункціональні вимоги безпеки, тестування безпеки включало перевірку надійності механізмів автентифікації та авторизації на основі JWT-токенів, захисту від SQL-ін’єкцій (мінімізовано використанням Sequelize ORM), а також коректності хешування паролів.

Процес налагодження супроводжував усі етапи розробки. Для серверної частини використовувалися вбудовані інструменти налагодження Node.js у Visual Studio Code та виведення інформації в консоль, при цьому стандартизовані повідомлення про помилки від ErrorHandler та класу ApiError спрощували ідентифікацію проблем. Для клієнтської частини активно застосовувалися інструменти розробника в браузері (Console, Network, Application, React DevTools) та можливості налагоджувача VS Code.

У ході розробки та тестування було виявлено та усунуто низку недоліків. Наприклад, початкові проблеми з валідацією даних на формах були вирішені шляхом додавання строгіших правил у моделях Sequelize та компонентах React. Окремі помилки в логіці API при граничних умовах були виправлені через ретельніше тестування контролерів; зокрема, було оптимізовано логіку оновлення профілю користувача для коректної обробки зміни email та username з перевіркою їх унікальності. Також усувалися дрібні візуальні дефекти інтерфейсу, покращувалася адаптивність та доопрацьовувалися механізми обробки JWT-токенів. Удосконалено процес завантаження файлів, додавши генерацію унікальних імен за допомогою uuid. Ітеративний підхід до тестування та налагодження, де кожна нова функціональність супроводжувалася відповідними тестами, дозволив створити стабільну та надійну платформу, що відповідає поставленим вимогам.

ВИСНОВКИ

У цій кваліфікаційній роботі бакалавра було послідовно виконано всі поставлені завдання, що дозволило розробити функціональну платформу для управління персональним портфоліо.

У рамках першого завдання було здійснено аналіз ринку існуючих платформ для управління персональним портфоліо, таких як Behance, Dribbble та Adobe Portfolio. Оцінка їхніх переваг та недоліків, зокрема обмежень у кастомізації та контролі даних, обґрунтувала необхідність створення нової, більш гнучкої системи.

На наступному етапі було визначено ключові функціональні (включаючи реєстрацію, управління профілем, проектами, навичками, досвідом та відгуками) та нефункціональні вимоги (такі як адаптивність та безпека) до розроблюваної платформи. Також було спроектовано архітектуру системи з використанням UML-діаграм: діаграми класів для опису сутностей (User, Project, Skill, UserSkill, Experience, Testimonial) та їх взаємозв’язків, і діаграми варіантів використання для візуалізації взаємодії акторів із системою.

Для реалізації клієнтської та серверної частин було обрано оптимальний технологічний стек та засоби розробки. Зокрема, для серверної частини було використано Node.js з фреймворком Express.js, для клієнтської – бібліотеку React, а для управління базою даних – MySQL з ORM Sequelize. Середовищем розробки слугував Visual Studio Code, а для стилізації інтерфейсу застосовано TailwindCSS.

Було розроблено структуру бази даних у MySQL, що складається з шести взаємопов’язаних таблиць (Users, Projects, Skills, UserSkills, Experiences, Testimonials), забезпечивши її нормалізацію до третьої нормальної форми (3NF) для мінімізації надлишковості даних та забезпечення їх цілісності. Взаємодія з базою даних на рівні коду реалізована через ORM Sequelize, а створення та оновлення схеми бази даних здійснювалося за допомогою інструментарію Sequelize CLI для генерації файлів міграцій.

Серверну логіку було реалізовано на Node.js із використанням фреймворку Express.js для побудови RESTful API. Впроваджено механізми JWT-автентифікації для захисту маршрутів (AuthMiddleware), розроблено контролери для обробки бізнес-логіки кожної сутності, а також реалізовано централізовану обробку помилок (ErrorHandler, ApiError). Забезпечено функціонал завантаження файлів за допомогою express-fileupload.

Клієнтський інтерфейс було створено на React із застосуванням функціональних компонентів та хуків для управління станом. Для взаємодії з RESTful API серверної частини використовувалася бібліотека Axios, налаштована з автоматичним додаванням JWT-токенів. Розроблено ключові сторінки платформи, включаючи головну, профіль користувача, налаштування профілю, пошук, публічний профіль, сторінки реєстрації, входу та відновлення пароля. Реалізовано клієнтську маршрутизацію за допомогою React Router та глобальне управління станом через Context API.

На завершальному етапі було проведено комплексне тестування інформаційно-комп’ютерної системи різними методами, включаючи модульне, інтеграційне, функціональне, тестування користувацького інтерфейсу та безпеки. Процес налагодження, що супроводжував усі етапи розробки з використанням відповідних інструментів для серверної та клієнтської частин, дозволив виявити та усунути виявлені недоліки, забезпечивши стабільність, функціональність та відповідність системи поставленим вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке портфоліо. URL: https://ksw.net.ua/shho-take-portfolio/ (дата звернення: 02.02.2025).

2. Behance. URL: https://www.behance.net/ (дата звернення: 18.02.2025).

3. Dribbble. URL: https://dribbble.com/ (дата звернення: 12.03.2025).

4. Adobe Portfolio. URL: https://portfolio.adobe.com/ (дата звернення: 14.03.2025).

5. What is a UML diagram? – different types and benefits. Miro. URL: https://miro.com/diagramming/what-is-a-uml-diagram/ (дата звернення: 20.03.2025).

6. Express.js explained: what it is and how to use it in your javascript project. Codecademy. URL: https://www.codecademy.com/article/what-is-express-js (дата звернення: 22.03.2025).

7. Sequelize – Feature-rich ORM for modern TypeScript & JavaScript. URL: https://sequelize.org/ (дата звернення: 24.03.2025).

8. React. URL: https://react.dev (дата звернення: 22.04.2025).

9. Tailwind CSS – rapidly build modern websites without ever leaving your HTML. URL: https://tailwindcss.com/ (дата звернення: 25.04.2025).

10. MySQL. URL: https://www.mysql.com/ (дата звернення: 28.04.2025).

11. How does a REST API work with examples and challenges. Document360. URL: https://document360.com/blog/what-is-rest-api/ (дата звернення: 30.03.2025).

12. What is a JWT & how it works. Descope. URL: https://www.descope.com/learn/post/jwt (дата звернення: 25.04.2025).

13. Akshay. Why nodemailer is the best package for sending emails and ideal for on-premise deployments. Medium. URL: https://mrfreelancer9.medium.com/why-nodemailer-is-the-best-package-for-sending-emails-and-ideal-for-on-premise-deployments-f0e2420237e2 (дата звернення: 27.04.2025).

14. What is CRUD? Explanation & examples. manubes. URL: https://www.manubes.com/what-is-crud/ (дата звернення: 28.04.2025).

15. Mwaura W. Making HTTP requests with Axios. CircleCI. URL: https://circleci.com/blog/making-http-requests-with-axios/ (дата звернення: 01.05.2025).

ДОДАТКИ

Додаток А – Програмний код userController

```
const {
  Users,
  UserSkill,
  Projects,
  Experiences,
  Testimonial,
  Skill,
  sequelize,
} = require("../models");
const uuid = require("uuid");
const path = require("path");
const bcrypt = require("bcrypt");
const ApiError = require("../Error/ApiError");
const { Op } = require("sequelize");
const jwt = require("jsonwebtoken"); // Імпортуємо jwt
const  {  sendPasswordResetEmail  }  =  require('../service/mail-
service');

class UserController {
  async SignUp(req, res, next) {
    const {
      first_name,
      last_name,
      username,
      email,
      password,
      avatar_url,
      bio,
    } = req.body;

    try {
      // Перевірка, чи існує користувач із таким username або email
      const existingUser = await Users.findOne({
        where: {
          [Op.or]: [{ username }, { email }],
        },
      });

      if (existingUser) {
        return next(
          ApiError.badRequest(
            "Користувач із таким username або email уже існує"
          )
        );
      }

      // Хешування пароля та створення нового користувача
      const hash = await bcrypt.hash(password, 10);
      await Users.create({
        first_name,
```

```
        last_name,
        username,
        email,
        password_hash: hash,
        avatar_url,
        bio,
      });

      res.status(200).json("User created");
    } catch (error) {
      next(ApiError.internalServerError("Помилка при створенні користувача"));
    }
  }

  async SignIn(req, res, next) {
    const { username, password } = req.body;

    try {
      const user = await Users.findOne({ where: { username } });
      if (!user) {
        return next(ApiError.unauthorized("Неправильний логін"));
      }

      const isPasswordValid = await bcrypt.compare(
        password,
        user.password_hash
      );
      if (!isPasswordValid) {
        return next(ApiError.unauthorized("Неправильний пароль"));
      }

      // Генерація JWT токена
      const token = jwt.sign(
        { id: user.id, username: user.username, email: user.email }, // Дані, які будуть у токені
        process.env.JWT_SECRET, // Секретний ключ
        { expiresIn: "24h" } // Час дії токена
      );

      res.json({ token });
    } catch (error) {
      console.log(error);
      next(ApiError.internalServerError("Помилка при вході"));
    }
  }

  async Check(req, res, next) {
    const token = jwt.sign(
      { id: req.user.id, username: req.user.username, email: req.user.email }, // Дані, які будуть у токені
      process.env.JWT_SECRET,
```

```
    { expiresIn: "24h" }
  );
  return res.json({ token });
}

async getProfile(req, res, next) {
  try {
    const user = await Users.findOne({
      where: { id: req.user.id },
      attributes: { exclude: ["password_hash"] },
      include: [
        {
          model: UserSkill,
          include: [{ model: Skill, attributes: ["id", "name"] }],
        },
        { model: Projects },
        { model: Experiences },
        {
          model: Testimonial,
          as: "TestimonialsReceived",
          include: [
            {
              model: Users,
              as: "Author",
              attributes: [
                "id",
                "first_name",
                "last_name",
                "username",
                "avatar_url",
              ],
            },
          ],
          attributes: ["id", "rating", "comment", "createdAt"],
        },
      ],
      order: [
        [Experiences, "start_date", "DESC"],
        [Projects, "createdAt", "DESC"],
        [
          { model: Testimonial, as: "TestimonialsReceived" },
          "createdAt",
          "DESC",
        ],
      ],
    });
    if (!user) {
      return next(ApiError.notFound("Користувача не знайдено"));
    }
    res.json(user);
  } catch (error) {
    console.error("Error fetching profile:", error);
```

```
      next(ApiError.internalServerError("Помилка при отриманні
профілю"));
    }
  }

  async getPublicProfile(req, res, next) {
    try {
      const { username } = req.params;

      const user = await Users.findOne({
        where: { username },
        attributes: { exclude: ["password_hash"] },
        include: [
          {
            model: UserSkill,
            include: [{ model: Skill, attributes: ["id", "name"] }],
          },
          { model: Projects },
          { model: Experiences },
          {
            model: Testimonial,
            as: "TestimonialsReceived",
            include: [
              {
                model: Users,
                as: "Author",
                attributes: [
                  "id",
                  "first_name",
                  "last_name",
                  "username",
                  "avatar_url",
                ],
              },
            ],
            attributes: ["id", "rating", "comment", "createdAt"],
          },
        ],
        order: [
          [Experiences, "start_date", "DESC"],
          [Projects, "createdAt", "DESC"],
          [
            { model: Testimonial, as: "TestimonialsReceived" },
            "createdAt",
            "DESC",
          ],
        ],
      });

      if (!user) {
        return next(ApiError.notFound("Користувача не знайдено"));
      }
```

```
      res.json(user);
    } catch (error) {
      console.log(error);
      next(ApiError.internalServerError("Помилка при отриманні профілю"));
    }
  }

  async updateProfile(req, res, next) {
    try {
      const { first_name, last_name, bio, email, username } = req.body;
      let avatar_path = null;

      // Перевірка, чи існує користувач
      const user = await Users.findByPk(req.user.id);
      if (!user) {
        return next(ApiError.notFound("Користувача не знайдено"));
      }

      // Перевіряємо, чи є файл avatar_url в запиті
      if (req.files && req.files.avatar_url) {
        const avatar_url = req.files.avatar_url;
        avatar_path = uuid.v4() + path.extname(avatar_url.name);
        avatar_url.mv(path.resolve(__dirname, "..", "static", avatar_path));
      }

      // Якщо email змінився, перевіряємо чи він унікальний
      if (email && email !== user.email) {
        const existingUserWithEmail = await Users.findOne({
          where: { email },
        });

        if (existingUserWithEmail) {
          return next(
            ApiError.badRequest("Користувач з таким email вже існує")
          );
        }
      }

      // Якщо username змінився, перевіряємо чи він унікальний
      if (username && username !== user.username) {
        const existingUserWithUsername = await Users.findOne({
          where: { username },
        });

        if (existingUserWithUsername) {
          return next(
```

```
            ApiError.badRequest("Користувач з таким username вже існує")
          );
        }
      }

      // Оновлення базової інформації користувача
      await user.update({
        first_name: first_name || user.first_name,
        last_name: last_name || user.last_name,
        avatar_url: avatar_path || user.avatar_url,
        bio: bio || user.bio,
        email: email || user.email,
        username: username || user.username, // Додано оновлення username
      });

      // Повертаємо оновлені дані користувача
      res.json({
        message: "Профіль успішно оновлено",
        avatar_url: avatar_path || user.avatar_url,
        username: username || user.username, // Повертаємо оновлений username
      });
    } catch (error) {
      console.log(error);
      next(ApiError.internalServerError("Помилка при оновленні профілю"));
    }
  }

  async searchUsers(req, res, next) {
    try {
      const { username, name, skill_id } = req.query;

      // Базові умови пошуку
      const whereConditions = {};
      const includeOptions = [];

      // Формуємо умови пошуку за username або ім’ям/прізвищем
      if (username || name) {
        const nameConditions = [];

        if (username) {
          nameConditions.push({
            username: { [Op.like]: `%${username}%` },
          });
        }

        if (name) {
          nameConditions.push({
            [Op.or]: [
```

```
        { first_name: { [Op.like]: `%${name}%` } },
        { last_name: { [Op.like]: `%${name}%` } },
      ],
    });
  }

  whereConditions[Op.or] = nameConditions;
}

// Налаштування для включення навичок
const skillInclude = {
  model: UserSkill,
  include: [
    {
      model: Skill,
      attributes: ["id", "name"],
    },
  ],
  attributes: ["id", "proficiency"],
};

// Якщо шукаємо за конкретною навичкою
if (skill_id) {
  skillInclude.where = { skill_id };
}

includeOptions.push(skillInclude);

// Включаємо проекти для відображення в результатах пошуку
includeOptions.push({
  model: Projects,
  attributes: ["id", "title", "description", "image_url"],
  limit: 3,
});

// Виконання пошуку з обмеженням кількості результатів
const users = await Users.findAll({
  where: whereConditions,
  attributes: [
    "id",
    "username",
    "first_name",
    "last_name",
    "avatar_url",
    "bio",
  ],
  include: includeOptions,
  limit: 20,
  order: [["first_name", "ASC"]],
});

res.json(users);
```

```
    } catch (error) {
      console.log(error);
      next(ApiError.internalServerError("Помилка при пошуку користувачів"));
    }
  }

  // Запит на відновлення пароля
  async forgotPassword(req, res, next) {
    try {
      const { email } = req.body;

      // Перевірка наявності користувача з таким email
      const user = await Users.findOne({ where: { email } });
      if (!user) {
        return next(ApiError.notFound("Користувача з такою електронною поштою не знайдено"));
      }

      // Генерація токена відновлення
      const resetToken = jwt.sign(
        { id: user.id, email: user.email },
        process.env.JWT_SECRET,
        { expiresIn: "1h" } // Токен дійсний 1 годину
      );

      // Відправка токена через email
      await sendPasswordResetEmail(email, resetToken);

      res.json({
        message: "Інструкції для відновлення пароля надіслано на вашу електронну пошту"
        // Тепер ми не повертаємо resetToken у відповіді
      });
    } catch (error) {
      console.error("Error in forgotPassword:", error);
      next(ApiError.internalServerError("Помилка при обробці запиту на відновлення пароля"));
    }
  }

  // Скидання пароля за токеном
  async resetPassword(req, res, next) {
    try {
      const { token, newPassword } = req.body;

      // Перевірка валідності токена
      let decoded;
      try {
        decoded = jwt.verify(token, process.env.JWT_SECRET);
      } catch (err) {
```

```
        return      next(ApiError.unauthorized("Недійсний      або
прострочений токен"));
      }

      // Пошук користувача
      const user = await Users.findByPk(decoded.id);
      if (!user) {
        return next(ApiError.notFound("Користувача не знайдено"));
      }

      // Хешування та оновлення пароля
      const hash = await bcrypt.hash(newPassword, 10);
      await user.update({ password_hash: hash });

      res.json({ message: "Пароль успішно змінено" });
    } catch (error) {
      console.error("Error in resetPassword:", error);
      next(ApiError.internalServerError("Помилка     при     зміні
пароля"));
    }
  }

  // Зміна пароля аутентифікованого користувача
  async changePassword(req, res, next) {
    try {
      const userId = req.user.id;
      const { currentPassword, newPassword } = req.body;

      // Перевірка, чи існує користувач
      const user = await Users.findByPk(userId);
      if (!user) {
        return next(ApiError.notFound("Користувача не знайдено"));
      }

      // Перевірка поточного пароля
      const isPasswordValid = await bcrypt.compare(
        currentPassword,
        user.password_hash
      );

      if (!isPasswordValid) {
        return    next(ApiError.badRequest("Неправильний    поточний
пароль"));
      }

      // Хешування та оновлення пароля
      const hash = await bcrypt.hash(newPassword, 10);
      await user.update({ password_hash: hash });

      res.json({ message: "Пароль успішно змінено" });
    } catch (error) {
      console.error("Error in changePassword:", error);
```

```
      next(ApiError.internalServerError("Помилка при зміні
пароля"));
    }
  }
}

module.exports = new UserController();
```