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 СЕРВІСУ «FREELANCEVORTEX» З ПОШУКУ ФРІЛАНСЕРІВ З ВИКОРИСТАННЯМ REACT, LARAVEL ТА MYSQL

DEVELOPMENT OF THE «FREELANCEVORTEX» SERVICE FOR FREELANCER SEARCH USING REACT, LARAVEL AND MYSQL

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

Виконав: здобувач вищої освіти
групи ІПЗ-41
Бодяк Віталій Олегович

Керівник:
Ліщина Наталія Миколаївна

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025р.
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* 20*24* р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Бодяку Віталію Олеговичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка сервісу«FreelanceVortex» з пошуку фрілансерів з використанням React, Laravel та MySQL»*
Керівник роботи: *к.т.н., доцент Ліщина Н. М.*
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: *Laravel, PHP, JavaScript, React, MySQL, Pusher Chatify, Inertia.js Tailwind CSS, Розробка веб-застосунку.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз сучасного стану проблем, постановка завдання, сформувати вимоги до розроблюваного веб-сайту, обгрунтувати вибір технологій для розробки, практично реалізувати об'єкт проєктування, розробити базу даних, обгрунтувати принципи захисту веб-сайту, проаналізувати можливості сайту в масштабуванні, протестувати та налогодити інформаційну системи.*

5. Перелік графічного матеріалу: *17 рисунків, 1 таблиця, 3 додатки.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Ліщина Н. М.*		
Специфікація вимог до розробленої системи	*Ліщина Н. М.*		
Розробка об'єкта проектування	*Ліщина Н. М.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	5,06 %		
Академічна доброчесність	*Ліщина Н. М.*		

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.	Викон.
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	Викон.
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	Викон.
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	Викон.
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	Викон.
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	Викон.
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	Викон.

Здобувач вищої освіти _____ Віталій БОДЯК

Керівник кваліфікаційної роботи _____ Наталія ЛІЩИНА

АНОТАЦІЯ

Бодяк В. О. Розробка сервісу «FreelanceVortex» з пошуку фрілансерів з використанням React, Laravel 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 актуальності проблеми і сформульовано завдання. В другому розділі визначено вимоги до розроблюваної системи і вибір засобів, методів та алгоритмів для вирішення задачі. У третьому розділі представлено практичну реалізацію вебзастосунку «FreelanceVortex», описано процес розробки бази даних, реалізовано заходи щодо захисту інформаційно-комп’ютерної системи, розглянуто можливості модульного розширення та масштабування проєкту, а також проведено тестування на стійкість до типових кіберзагроз та функціональну справність. У висновках узагальнено отримані результати, підсумовано досягнення поставлених цілей та окреслено перспективи подальшого розвитку системи.

Ключові слова: Laravel, ORM Eloquent, Inertia, React, Tailwind CSS, Pusher, Chatify, XSS, SQL, MySQL, HTML, CSS, JavaScript, TypeScript, PHP.

ABSTRACT

Bodiak V. Development of the "FreelanceVortex" Service for Freelancer Search Using React, Laravel and MySQL. Manuscript. Qualification work for bachelor’s degree in Software Engineering, specialty "Software Engineering". Lutsk National Technical University. Lutsk, 2025.

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

The first chapter analyzes the relevance of the problem and formulates the task. The second section defines the requirements for the system to be developed and the choice of tools, methods and algorithms to solve the problem. The third section presents the practical implementation of the «FreelanceVortex» web application, describes the database development process, implements measures to protect the information and computer system, considers the possibilities of modular expansion and scaling of the project, and tests for resistance to typical cyber threats and functional performance. The conclusions summarize the results obtained, summarize the achievement of the set goals, and outline the prospects for further development of the system.

Keywords: Laravel, ORM Eloquent, Inertia, React, Tailwind CSS, Pusher, Chatify, XSS, SQL, MySQL, HTML, CSS, JavaScript, TypeScript, PHP.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі, відзначається стрімкий прогрес інформаційних технологій і підходів до роботи. Фріланс виокремився як затребувана альтернатива традиційній зайнятості. Це обумовлено тим, що дедалі більше підприємств та фахівців надають перевагу дистанційній роботі як зручному та гнучкому формату співпраці. Фріланс сприяє скороченню операційних витрат на утримання офісів, оптимізує робочі процеси та відкриває доступ до ширшого кола талановитих фахівців з усього світу. Проте, для успішного знаходження виконавців та замовників, а також для створення атмосфери взаємної довіри, необхідна наявність практичної та безпечної онлайн-платформи, яка забезпечить легкість у взаємодії та оперативний зв’язок.

Такі платформи суттєво спрощують процеси найму, надаючи стабільну та захищену основу для співробітництва, завдяки чому фрілансери та роботодавці можуть зосереджуватися на своїх професійних завданнях, уникаючи зайвих проблем. Для користувачів платформи вирішальне значення має не тільки її функціональність, але й інтуїтивно зрозумілий інтерфейс, який дає можливість оперативно знаходити необхідні послуги або кандидатів, а також зручні механізми для здійснення фінансових операцій та перевірки репутації.

Мета роботи полягає в тому, щоб проаналізувати існуючі платформи для пошуку фрілансерів та роботодавців, а також розробити та створити сайт, який буде максимально ефективним та зручним для користувачів.

Об’єктом роботи являється процес розробки веб-застосунку, для організації взаємодії фрілансерів та роботодавців.

Предметом роботи є архітектурні, функціональні особливості розробки веб-сайту «FreelanceVortex» з використанням Laravel, React, MySQL.

Завдання на кваліфікаційну роботу:

– проаналізувати сучасний стан проблем;
– сформувати вимоги до розроблюваного веб-сайту;
– обгрунтувати вибір технологій для розробки;

- практично реалізувати об’єкт проєктування;
- розробити базу даних;
- обгрунтувати принципи захисту веб-сайту;
- проаналізувати можливості сайту в масштабуванні;
- протестувати та налогодити інформаційну системи.

Апробацію результатів кваліфікаційної роботи здійснено шляхом участі в X Міжнародній науково-практичній конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)», м. Луцьк, 23-24 травня 2025 р. (додаток А).

РОЗДІЛ 1
АНАЛІЗ МЕТОДІВ РОЗРОБКИ ВЕБ-САЙТІВ ТА ПОСТАНОВКА ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Сучасний ринок інформаційних послуг розвивається високими темпами, що викликано швидким прогресом технологій та глобалізацією економіки. Водночас, цей ринок характеризується високою конкуренцією, створюючи нові виклики для фрілансерів та роботодавців. Зростаюча популярність віддаленої роботи відкриває широкі можливості для людей, які шукають гнучкості у своїй професійній діяльності. Проте, це також породжує певні труднощі, особливо у сфері пошуку відповідних фахівців та забезпечення їх надійності.

Серед найпопулярніших платформ для пошуку фрілансерів виділяються Upwork, Fiverr та Freelancer.com, кожна з яких застосовує свою власну модель взаємодії з користувачами.

Upwork працює як універсальний майданчик, орієнтований на проєктну та погодинну співпрацю [13]. Завдяки гнучкій системі найму, можливості створення тривалих контрактів та вбудованим інструментам контролю продуктивності (зокрема, тайм-трекер), платформа забезпечує відносно високий рівень довіри між сторонами. Проте система комісій (до 20 %) та складний процес модерації профілів можуть зменшувати привабливість для нових користувачів.

Fiverr пропонує іншу модель – засновану на фіксованих послугах, де фрілансери попередньо описують перелік завдань та вартість їхнього виконання [5]. Такий підхід знижує транзакційні витрати та прискорює ухвалення рішень замовником, однак ускладнює реалізацію складних або гнучких проєктів. Fiverr також потерпає від проблеми «інформаційного шуму» – велика кількість пропозицій різного рівня якості ускладнює пошук кваліфікованого виконавця.

Freelancer.com, своєю чергою, поєднує елементи класичної біржі праці з можливістю участі у відкритих тендерах [6]. Це створює умови для конкурентного відбору, проте водночас стимулює демпінг цін та призводить до перенасичення платформи низькоякісними заявками. Комісійна політика та нав’язування платних опцій для підвищення видимості також зменшують загальну ефективність платформи з точки зору спеціаліста.

Однією з основних проблем, з якою зіштовхуються фрілансери-початківці, є пошук свого першого завдання. Оскільки більшість роботодавців віддають перевагу досвідченим виконавцям, новачки часто залишаються без роботи або вимушені встановлювати завідомо низьку ціну за свої послуги. Це призводить до ситуації, коли фрілансери вимушені знижувати вартість, щоб залучити перших клієнтів, що може негативно вплинути на їх реальний потенціал та професійну вартість. Цей дисбаланс між попитом та пропозицією на ринку фріланс-послуг є однією з ключових проблем, що потребують вирішення.

Іншою важливою проблемою є наявність недобросовісних роботодавців. Фрілансери часто стикаються з випадками, коли роботодавці не виплачують належну винагороду, не надають точних технічних завдань або вимагають внесення безкоштовних змін. Ці фактори значно зменшують ефективність та безпеку співпраці, що, у свою чергу, призводить до зниження рівня довіри до таких платформ та може негативно вплинути на репутацію самої індустрії фріланс-роботи.

Для замовників також існують складнощі в оцінці реального рівня знань та досвіду фрілансера. Часто неможливо наочно перевірити кваліфікацію виконавця, що може призвести до зриву термінів виконання проєкту або до отримання неякісного продукту. Недосконаліть процесів відбору фахівців, відсутність чітких критеріїв оцінювання та недостатній контроль за виконанням робіт можуть значно ускладнити співпрацю між роботодавцями та фрілансерами, що підвищує ризики для обох сторін.

Щоб подолати ці труднощі та зменшити рівень невизначеності на ринку фріланс-послуг, необхідно впроваджувати низку заходів для захисту прав як

фрілансерів, так і замовників. Одним з таких заходів є розробка та інтеграція механізмів перевірки кваліфікації обох сторін співпраці. Для фрілансерів це можуть бути системи сертифікації, тести на знання або перевірка портфоліо. Для роботодавців це можуть бути рейтинги на основі відгуків попередніх виконавців, а також верифікація компанії та її репутації.

Додатково, важливим є впровадження прозорих правил співпраці, які включають чітке визначення умов роботи, оплати та термінів виконання завдання. Залучення механізмів посередництва, що гарантують безпеку транзакцій, таких як ескроу-системи (коли гроші зберігаються на спеціальному рахунку до завершення роботи), дозволяє знизити ризик невиплат та незадоволення виконаною роботою.

Таким чином, для створення більш надійного та безпечного ринку фріланс-послуг необхідно впроваджувати комплексні рішення, які сприятимуть покращенню умов співпраці між фрілансерами та замовниками. Забезпечення прозорості, впровадження справедливої системи оцінювання та створення ефективних механізмів захисту прав користувачів дозволить значно покращити якість ринку фріланс-послуг та знизити рівень недовіри серед учасників цього ринку.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Метою дослідження є розробка сервісу «FreelanceVortex» який має містити мінімальний необхідний функціонал для взаємодії фрілансерів та замовників. Також було сформовано мету роботи, а саме розробити веб застосунок, який дозволить фрілансерам і замовникам ефективно співпрацювати між собою.

Для досягнення поставленої мети було сформовано такі завдання:

– проаналізувати сучасний стан проблем;
– сформувати вимоги до розроблюваного веб-сайту;
– обгрунтувати вибір технологій для розробки;
– практично реалізувати об’єкт проєктування;

– розробити базу даних;
– обгрунтувати принципи захисту веб-сайту;
– проаналізувати можливості сайту в масштабуванні;
– протестувати та налогодити інформаційну системи.

Висновки до розділу 1

Було проведено комплексний розбір стану ринку інформаційних послуг, зокрема стосовно платформ для пошуку фрілансерів та роботодавців. Процес розбору дав змогу визначити ключові тенденції, що притаманні цій галузі, а також виявити проблеми, які виникають у взаємодії між фрілансерами та замовниками. Було вказано, що хоча ринок фріланс-послуг розвивається швидкими темпами, існує потреба в покращенні інфраструктури задля забезпечення ефективної і безпечної співпраці.

У результаті проведеного розбору було визначено основні цілі та завдання для виконання кваліфікаційної роботи. Однією з ключових цілей є розробка онлайн-платформи для пошуку фрілансерів і роботодавців, котра повинна враховувати сучасні вимоги ринку та потреби користувачів.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

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

На даний момент, коли цифрові технології стрімко розвиваються, віддалена робота та фріланс набувають все більшої популярності, виникає необхідність у створенні ефективних веб-платформ. Такі платформи мають забезпечувати безперебійну комунікацію між замовниками та виконавцями проектів. Розробка «FreelanceVortex» є відповіддю на потреби ринку, прагнучого зручного, масштабованого та функціонально насиченого рішення для пошуку, організації та супроводу фріланс-проектів. Враховуючи поточні вимоги до подібних систем, було проведено всебічний аналіз предметної області. Цей аналіз включає вивчення цільової аудиторії, огляд наявних рішень, формалізацію як функціональних, так і нефункціональних вимог, а також архітектурне проектування.

Аналіз предметної області виявив ключові функціональні сценарії, які необхідно втілити у системі «FreelanceVortex». Зокрема, йдеться про створення облікових записів для фрілансерів та замовників, публікацію та пошук проектів. Важливим є інтеграція механізму ставок, подання заявок або прямих запрошень. Також передбачена можливість ведення діалогу між учасниками проекту через інтегрований чат, а також системи оцінювання, відгуків та модулів верифікації користувачів. Кожен з цих модулів має відповідати критеріям масштабованості, доступності, безпеки та високої продуктивності.

Для моделювання сценаріїв взаємодії користувача з сайтом «FreelanceVortex», було розроблено діаграму активностей користувача. Вона включає в себе ключові функціональні компоненти інтерфейсу користувача, такі як вітальна сторінка, аутентифікація, особистий кабінет, чат-систем, профіль, налаштування профілю та облікового запису.

Активності згруповані відповідно до ролей користувачів. Всі ролі мають доступ до загального функціоналу, а адміністратор наділений розширеними правами, зокрема – доступом до сторінки з репортами, що дозволяє здійснювати модерацію на сайті. Умовні оператори та паралельні гілки в діаграмі використовуються для відображення як послідовних, так і одночасних дій користувача. Такий розподіл функціоналу сприяє підтриманню принципів безпеки та розмежування прав доступу згідно з ролями. На рисунку 2.1 зображено вигляд діаграми активностей користувачів на сайті.

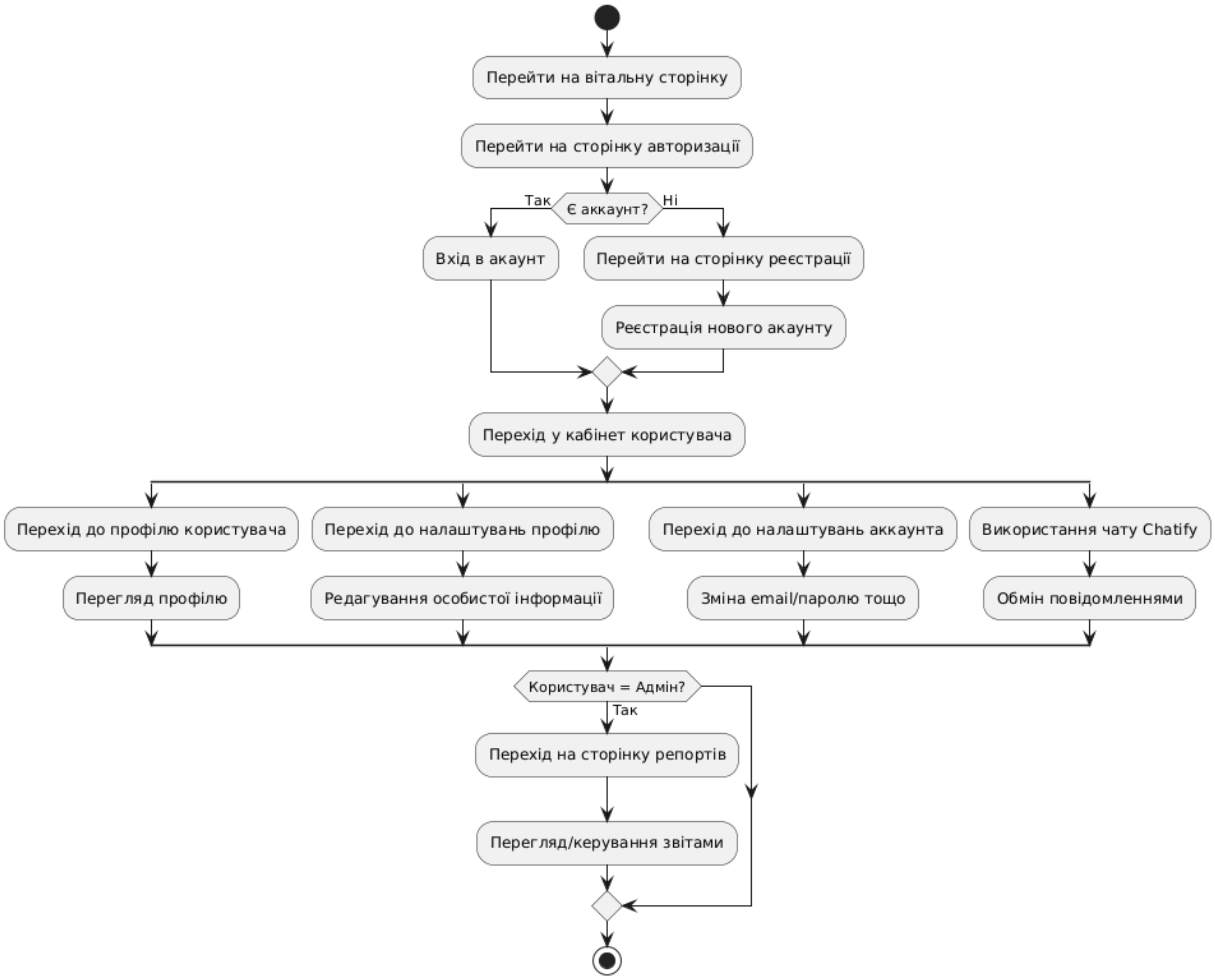

Рисунок 2.1 – Діаграма активностей

Для створення надійної та гнучкої системи, було обрано певний технологічний стек. Він включає React.js як бібліотеку для динамічного клієнтського інтерфейсу. React дозволяє розробникам створювати великі

вебзастосунки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки [10]. Laravel для реалізації бекенд-логіки. Laravel – безкоштовний, з відкритим кодом PHP-фреймворк [9]. Для взаємодії клієнту і серверу було використано Inertia. Inertia не є фреймворком і не замінює існуючі серверні або клієнтські фреймворки [8]. Вона працює як інструмент зв'язку між фронтендом та бекендом без необхідності створення окремого API-шару, а також Tailwind CSS для швидкого та адаптивного стилю компонентів. Функціональність реального часу, а саме чат, реалізується з використанням Pusher у поєднанні з бібліотекою Chatify, що дозволяє інтегрувати двосторонню комунікацію у середовище Laravel. Pusher – це хмарна платформа, яка в першу чергу використовується для додавання функцій, пов'язаних з комунікацією в реальному часі, в мобільні або веб-додатки [15].

При формулюванні вимог особливу увагу було приділено розробці користувацьких історій (user stories) для кожного типу користувачів. Ці сценарії охоплюють увесь процес взаємодії з платформою, від реєстрації до отримання оплати або завершення проекту. Використання підходу, заснованого на сценаріях, дозволяє сфокусуватися на реальних потребах кінцевих користувачів, визначити критичні точки взаємодії та покращити загальну ергономіку платформи. На етапі визначення вимог було враховано ймовірні проблемні ситуації, зокрема відновлення доступу до облікового запису, вирішення спорів, автоматичне закриття неактивних заявок, а також підтримка багатомовності.

Архітектурне проектування системи здійснено з урахуванням принципів розділення відповідальності, модульності та повторного використання коду (рис. 2.2). Компонентний підхід у реалізації інтерфейсу з використанням React забезпечує гнучке масштабування UI, впровадження нових компонентів без впливу на існуючий функціонал. Поєднання React з Inertia.js дозволяє уникнути традиційного REST API, спрощуючи взаємодію між клієнтом та сервером, що, у свою чергу, зменшує обсяг необхідного коду та потенційні точки відмови.

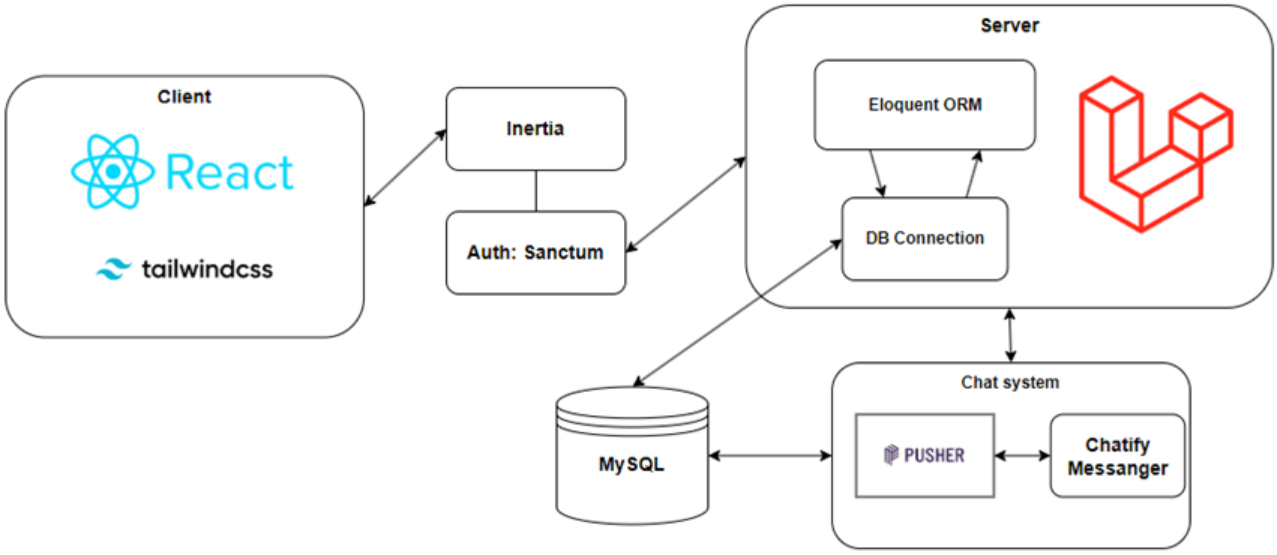

Рисунок 2.2 – Діаграми архітектури сайту «FreelanceVortex»

Діаграма зображує архітектуру додатку, що демонструє взаємозв'язки між ключовими компонентами: фронтенд на основі React, серверну логіку на Laravel, канали реального часу через Pusher, модуль обміну повідомленнями Chatify, а також підсистему управління базою даних на основі MySQL.

Laravel, як основа для серверної логіки, обрано не лише через його популярність, а й завдяки наявності потужної екосистеми. Вона дозволяє легко реалізувати автентифікацію, управління БД, обробку запитів, трансляцію подій у реальному часі та інші критично важливі функції без надмірного навантаження на архітектуру. Використання Laravel Echo разом з Pusher забезпечує ефективну підтримку подій у реальному часі, а Chatify мінімізує час на реалізацію надійного механізму обміну повідомленнями.

Завдяки використанню Tailwind CSS, система володіє високою адаптивністю та можливістю гнучкої кастомізації інтерфейсу без надмірного обсягу додаткового CSS-коду. Це дозволяє підтримувати єдиний стиль оформлення, забезпечувати доступність та адаптивність інтерфейсу на різних пристроях.

Щодо безпеки, було розроблено ряд механізмів захисту, включно з захистом від CSRF-атак, валідацією вхідних даних, захистом від XSS-атак, а також безпечним зберіганням паролів із використанням сучасних алгоритмів

хешування. Додатково, впроваджено багаторівневу систему прав доступу, яка забезпечує належний контроль над ресурсами залежно від ролі користувача.

На етапі проектування бази даних було враховано зв’язки між користувачами, проектами, повідомленнями, транзакціями та модулем сповіщень. Схема передбачає підтримку відносин «один до багатьох» та «багато до багатьох» з використанням індексації та нормалізації для покращення продуктивності. У якості СКБД обрано MySQL, яка добре інтегрується з Laravel та забезпечує високу швидкість обробки великих обсягів даних.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Головною метою при виборі технологій для виконання кваліфікаційної роботи полягала у створенні зручного, швидкого та адаптивного сервісу, що дозволяє користувачам публікувати проєкти, подавати заявки, обмінюватися повідомленнями, а також керувати профілями. Розробка такого функціонального сервісу вимагала виваженого вибору інструментів, методів та алгоритмів для ефективного вирішення поставленої задачі. Критеріями відбору були: продуктивність, зручність використання, гнучкість у розширенні функціоналу, простота інтеграції та масштабованість. Всі застосовані технології гармонійно поєднуються між собою, забезпечуючи стабільну та швидку роботу системи як на боці клієнта, так і на сервері.

Основою клієнтської частини став фреймворк React, котрий надає широкі можливості для створення динамічних інтерфейсів. React використовує компонентний підхід до розробки, що суттєво спрощує розробку, тестування та повторне використання коду. У поєднанні з Tailwind CSS, що забезпечує швидке створення адаптивного дизайну за допомогою утилітарних класів, інтерфейс вийшов легким, чистим та водночас функціональним. Tailwind CSS дозволяє уникнути зайвих CSS-файлів, прискорюючи завантаження сторінки та полегшуючи подальше супроводження.

Для взаємодії клієнта з сервером було обрано технологію Inertia.js, яка поєднує зручність розробки односторінкових застосунків із традиційною серверною архітектурою. Завдяки Inertia, вся логіка лишається на бекенді, що дозволяє використовувати перевірені серверні підходи Laravel без потреби створення API. Це значно зменшує складність розробки та дозволяє уникнути дублювання логіки, зокрема в частині маршрутизації та обробки запитів. Замість REST API або GraphQL, які вимагали б додаткової інфраструктури для обміну даними між фронтендом і бекендом, Inertia дає змогу передавати дані у вигляді пропсів компонентам React без зайвих перетворень. На рисунку 2.3 зображено приклад роботи REST API.

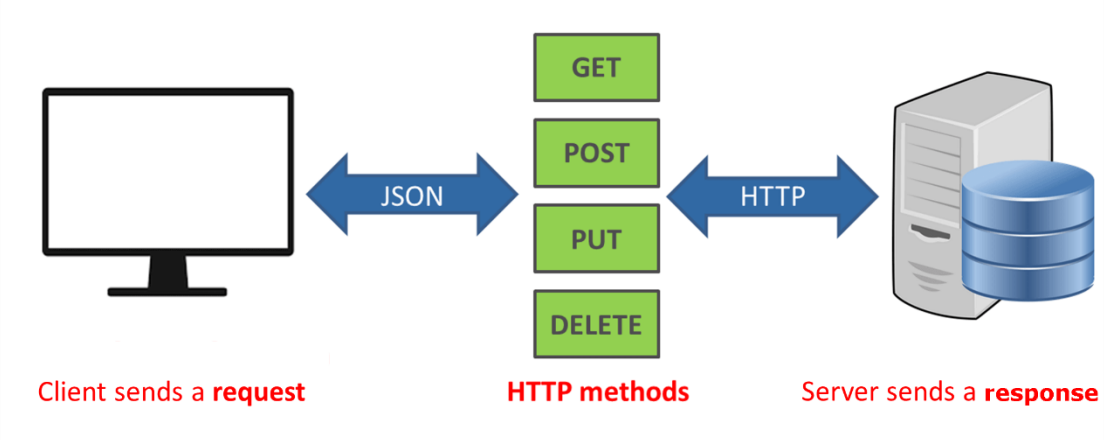

Рисунок 2.3 – Приклад роботи REST API [16]

На серверній стороні було використано Laravel – популярний PHP-фреймворк, котрий поєднує високу продуктивність із гнучкістю та зручністю. Laravel забезпечує чітку структуру проєкту, що полегшує масштабування та супроводження. Його вбудовані механізми авторизації, маршрутизації, валідації, черг, подій та планувальників задач дозволяють створювати розвинену логіку застосунку з мінімальними витратами часу. Однією з ключових переваг є застосування шаблонів проєктування, зокрема MVC (Model-View-Controller), що розділяє логіку, представлення та дані, забезпечуючи чистоту та організованість коду.

Збереження даних було реалізовано за допомогою реляційної бази даних MySQL. Цей вибір зумовлений надійністю, продуктивністю та широкою підтримкою. MySQL дозволяє ефективно працювати з великими обсягами структурованих даних, забезпечує підтримку транзакцій, індексацію, складні запити та можливість реплікації. Було створено чітку схему бази даних із розподілом таблиць на користувачів, профілі, відгуки, повідомлення та інші сутності, які відображають основну бізнес-логіку платформи.

Для роботи з базою даних використовувалась ORM Eloquent, котра входить до складу Laravel. Вона дозволяє маніпулювати даними у вигляді об’єктів, зберігаючи при цьому зв’язок з таблицями бази даних. Це значно підвищує читабельність та безпеку коду, а також спрощує тестування. Наприклад, зв’язки типу «один до багатьох», «багато до багатьох», або «належить до» реалізуються за допомогою вбудованих методів, що дозволяє зручно отримувати всі заявки до певного проєкту або всі повідомлення від конкретного користувача. Також Eloquent дозволяє реалізувати локальні та глобальні скоупи, автоматично обробляти мітки часу, використовувати м’яке видалення та багато інших корисних функцій.

Особливу увагу було приділено використанню принципів об’єктно-орієнтованого програмування. Вся логіка платформи реалізована у вигляді класів, що дає змогу досягти високого рівня абстракції, повторного використання та тестованості. Наприклад, кожна бізнес-операція – створення заявки, відправлення повідомлення, оновлення профілю – реалізована як окремий сервіс або контролер. Це дозволяє легко змінювати логіку без впливу на інші частини системи. Також було реалізовано інтерфейси та абстрактні класи для створення єдиних підходів до взаємодії між модулями. Це особливо важливо для реалізації можливих майбутніх змін, таких як впровадження інших платіжних систем, інтеграція сторонніх API або перехід на іншу базу даних.

Безпека також була одним з ключових факторів при виборі технологій. Laravel надає вбудовані засоби захисту від розповсюджених вразливостей, як-от SQL-ін’єкції, XSS, CSRF. Також реалізовано захист маршрутів за допомогою

middleware, що дозволяє обмежити доступ до певних частин застосунку. Для автентифікації та авторизації використано стандартний механізм Laravel Breeze, адаптований під Inertia та React. Реєстрація та вхід реалізовані з валідацією даних, хешуванням паролів та підтвердженням електронної пошти. Крім того, було реалізовано ролі та права доступу, що розділяють функціональність між фрілансерами, роботодавцями та адміністраторами.

Розробка інтерфейсу також враховувала потреби адаптивності – Tailwind дозволяє швидко створювати сторінки, які однаково зручно відображаються як на десктопах, так і на мобільних пристроях. Усі компоненти інтерфейсу протестовані на різних розширеннях екранів. Крім того, платформа має інтуїтивно зрозумілу навігацію, що знижує поріг входження для нових користувачів. Багато уваги приділено доступності, зокрема семантичній структурі HTML, контрастності кольорів, логічному фокусуванню для клавіатурної навігації.

В рамках розробки було реалізовано чат у реальному часі за допомогою Laravel Pusher. Це дозволяє користувачам миттєво отримувати нові повідомлення без необхідності оновлення сторінки. Також було впроваджено сповіщення про нові заявки, відповіді та зміни в проєктах, що підвищує взаємодію між користувачами платформи.

Висновки до розділу 2

Внаслідок здійсненого аналізу, формування вимог та проектування програмного забезпечення детально вивчено функціональні та нефункціональні вимоги до системи «FreelanceVortex». На підставі аналізу цільової аудиторії, логіки користувацьких сценаріїв і принципів UX-дизайну було побудовано архітектурну модель, яка відповідає сучасним стандартам розробки web-застосунків. В рамках цього етапу розглянуто логіку розподілу ролей користувачів, ключові бізнес-процеси взаємодії між фрілансером та замовником, а також визначено критично важливі функції, зокрема пошук проектів, створення

заявок, обробку повідомлень та модерацію контенту. Проектування передбачало використання багаторівневої архітектури, що дозволяє чітко розмежовувати логіку представлення, бізнес-логіку та доступ до даних. На рисунку 2.1 відображено архітектуру додатку, яка демонструє ключові компоненти системи та їхню взаємодію.

Було обґрунтовано вибір засобів, методів і алгоритмів, потрібних для реалізації поставленої задачі. Зокрема, як основний серверний фреймворк обрано Laravel – популярне PHP-рішення з розширеною підтримкою MVC-парадигми, вбудованими інструментами безпеки, роутингом та ORM. Для фронтенд-розробки застосовано React у поєднанні з Inertia.js, що дозволило реалізувати SPA-архітектуру без використання REST API на кожний запит, забезпечивши при цьому високий рівень інтерактивності інтерфейсу. Tailwind CSS було використано для швидкого та адаптивного створення стилів, що дозволило значно прискорити етап UI-розробки. Для реалізації реального часу у чаті між користувачами інтегровано Pusher та бібліотеку Chatify, які забезпечують миттєву доставку повідомлень.

Також у межах цього етапу визначено та проаналізовано алгоритмічну складову деяких функцій, зокрема сортування та фільтрації результатів пошуку, побудову рекомендацій за тегами технологій, а також алгоритми перевірки унікальності й безпеки даних.

РОЗДІЛ 3
РОЗРОБКА ВЕБ-САЙТУ «FREELANCEVORTEX»

3.1 Практична реалізація об’єкта проєктування

Першим етопом є створення та ініціалізація проєкту. У фреймворці Laravel наявна команда «laravel new», за допомогою якої створюється базовий проєкт, у якому є все необхідне для розробки, а якщо й немає, то можна інсталювати через пакетний менеджер composer. Laravel Breeze – це мінімальна, проста реалізація всіх функцій автентифікації Laravel, включаючи вхід, реєстрацію, скидання пароля, перевірку електронної пошти та підтвердження пароля [12].

На рисунку 3.1 зображено виконання команди створення проєкту з базовим стартовим пакетом Breeze.

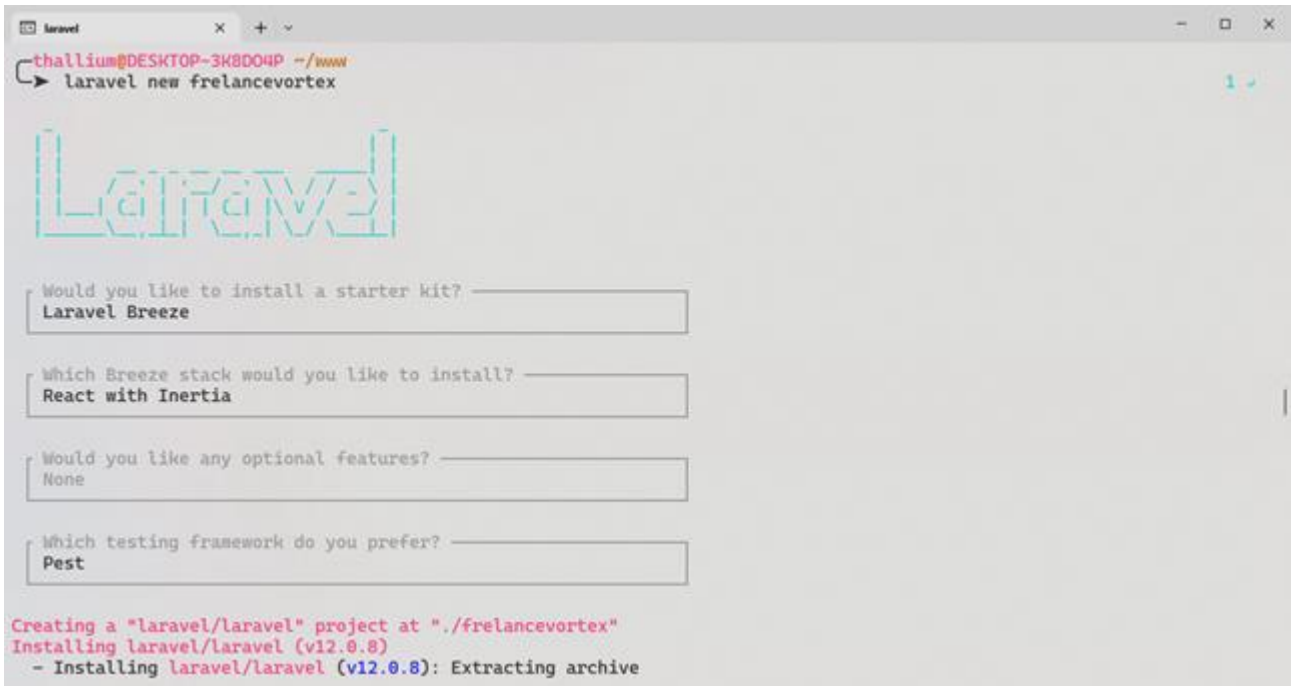

Рисунок 3.1 – Меню створення проєкту на Laravel з стартовим пакетом

Після інсталяції пакетів, необхідно вказати яка база даних буде використовуватися у проєкті. Для розробки була обрана MySQL.

На рисунку 3.2 зображено обрану базу даних.

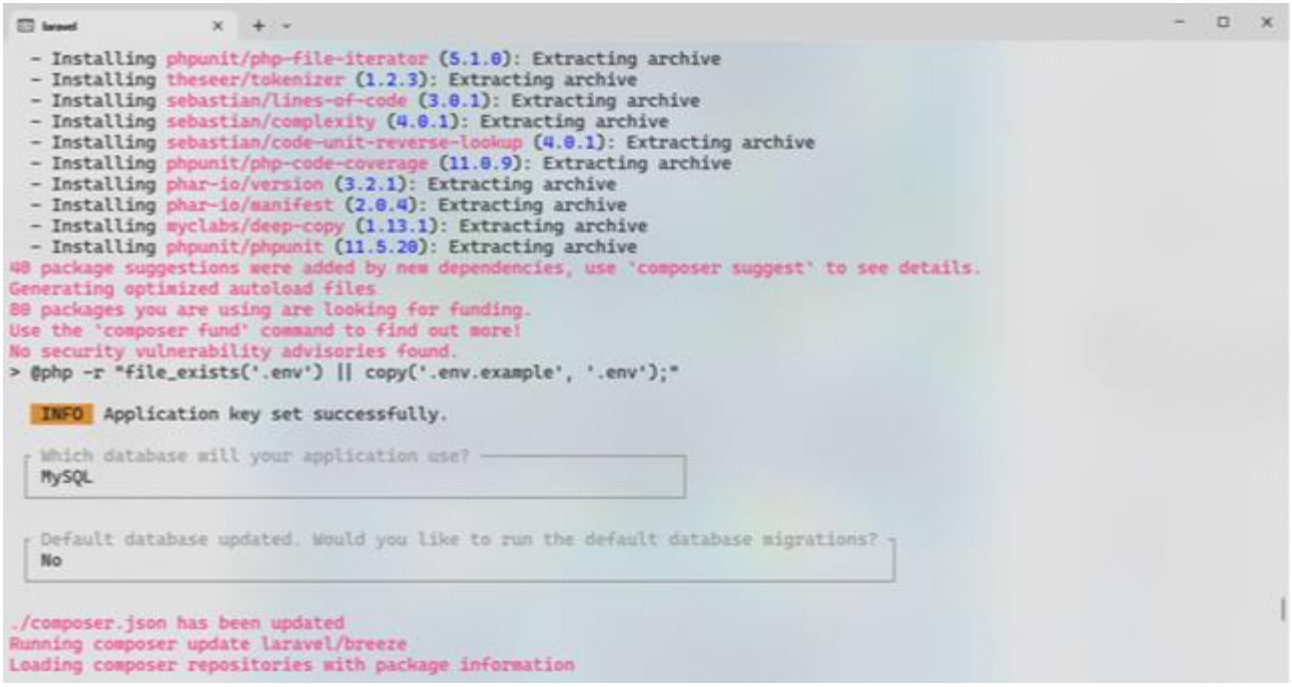

Рисунок 3.2 – Вибір бази даних

Після чого ми можемо запустити проєкт в режимі розробки за допомогою команди «composer run dev», після чого за посиланням http://127.0.0.1:8000/ відкриється вітальна сторінка проєкту.

На рисунку 3.3 зображена вітальна сторінка створеного проєкту.

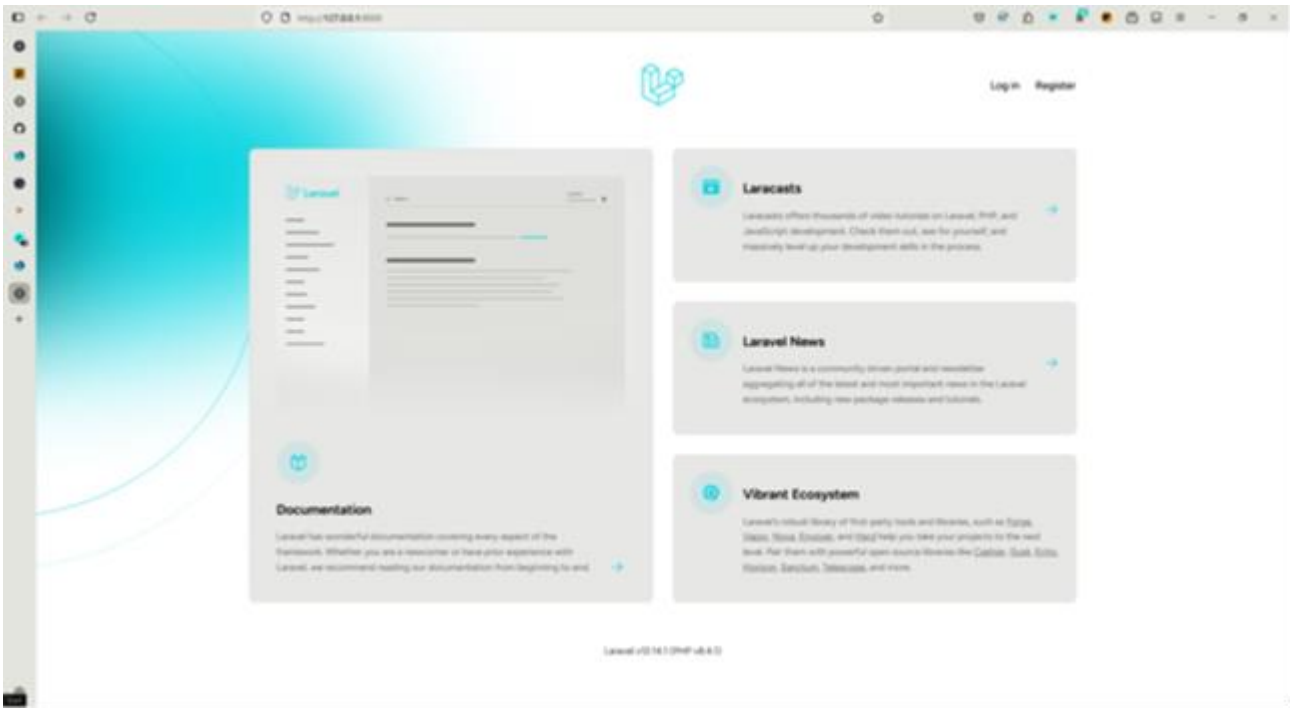

Рисунок 3.3 – Вітальна сторінка створеного проєкту

Оскільки проєкт розгорнутий, то можна приступити до розробки. Першим етапом було змінено вітальну сторінку проекту. У лістингу 3.1 зображено код головної сторінки.

Лістинг 3.1 – React код вітальної сторінки

```
import { Head, Link } from "@inertiajs/react";
import MainWelcome from "../Components/MainWelcome";

export default function Welcome({ auth, isAuth }) {
  return (
    <div className="min-h-screen bg-gray-100">
      <Head title="Головна" />
      <header className="border-b border-gray-100 bg-white">
        <div className="mx-auto max-w-7xl px-4 sm:px-6 lg:px-8">
          <div className="flex h-16 items-center justify-between">
            <div className="flex shrink-0 items-center">
              <Link href="/">
                Freelance<span style={{ color: "blue" }}>Vortex</span>
              </Link>
            </div>
            <nav className="flex space-x-4">
              {auth.user ? (
                <Link
                  href={route("dashboard")}
                  className="px-4 py-2 rounded-md text-black transition
hover:text-gray-700"
                >
                  Мій кабінет
                </Link>
              ) : (
                <>
                  <Link
                    href="/login"
                    className="px-4 py-2 rounded-md text-black transition
hover:text-gray-700"
                  >
                    Увійти
                  </Link>
                  <Link
                    href="/register"
                    className="px-4 py-2 rounded-md bg-blue-600 text-
white hover:bg-blue-700"
                  >
                    Реєстрація
                  </Link>
                </>
              )}
            </nav>
```

```
            </div>
          </div>
        </header>
        <main className="bg-white">
          <MainWelcome auth={auth} isAuth={isAuth}></MainWelcome>
        </main>
      </div>
    );
}
```

кінець лістингу 3.1

Після написання всіх стилів та компонентів було отримано сторінку яка складається з двох головних частин, а саме Header, де знаходиться логотип та кнопки реєстрації та входу в систему, а також головна частина, у якій знаходиться контент цієї сторінки. Головна ціль цієї сторінки це привернути увагу користувача.

На рисунку 3.4 зображено результат виконання коду.

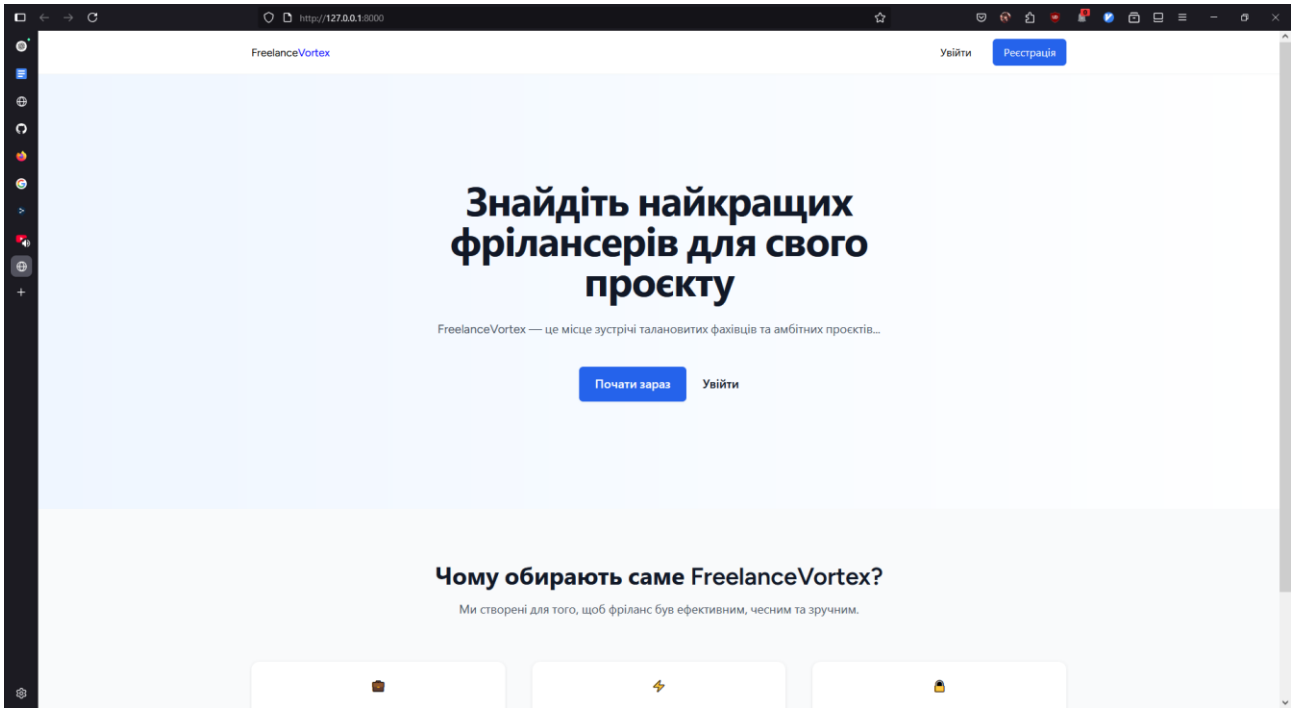

Рисунок 3.4 – Головна сторінка

Для авторизації та для переходу потрібно описати шляхи. Для цього у файлі web.php потрібно оголосити відповідні маршрути, які відповідатимуть за

обробку запитів до сторінок авторизації та переходу між розділами сайту. У файлі web.php ці маршрути задаються за допомогою методів Route::get, Route::post, Route::middleware. Також хорошою практикою являється розбиття коду на окремі частини, що дозволить розбити основну логіку та авторизацію на окремі частини. У лістингу 3.2 зображено код файлу web.php.

Лістинг 3.2 – Код файлу web.php

```
<?php

use Inertia\Inertia;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Route;
use Illuminate\Foundation\Application;
use App\Http\Controllers\UserController;
use App\Http\Controllers\ProfileController;
use GuzzleHttp\Psr7\Request;

Route::get('/', function () {
    return Inertia::render('Welcome', [
        'canLogin' => Route::has('login'),
        'canRegister' => Route::has('register'),
        'laravelVersion' => Application::VERSION,
        'phpVersion' => PHP_VERSION,
        'isAuth' => Auth::check()
    ]);
});

Route::get('/user-data', [UserController::class, 'getUsers']);

Route::get('/dashboard', [UserController::class, 'dasboard'])-
>middleware(['auth', 'verified'])->name('dashboard');

Route::middleware('auth')->group(function () {
    Route::get('/fillter', [UserController::class, 'fillter'])-
>name('fillter');

    Route::get('/user-profile/{user}', [ProfileController::class,
'userProfile'])->name('profile.user.index');
    Route::post('/user-profile/{user}/review/',
[ProfileController::class, 'storeUserReview'])-
>name('profile.user.store.review');

    Route::get('/profile', [ProfileController::class, 'index'])-
>name('profile.edit');
    Route::put('/profile', [ProfileController::class, 'updateResume'])-
>name('profile.update');
```

```
    Route::get('/profile/settings', [ProfileController::class, 'edit'])-
>name('settings.edit');
    Route::patch('/profile/settings', [ProfileController::class,
'update'])->name('settings.update');
    Route::delete('/profile/settings', [ProfileController::class,
'destroy'])->name('settings.destroy');
});

require __DIR__.'/auth.php';
```

кінець лістингу 3.2

Для реєстрації було створено файл у директорії app/Http/Controllers/Auth RegisteredUserController.php у якому було описано два методи. Перший це відображення сторінки реєстрації, у ній знаходиться такі форми як ім'я та прізвище, емейл, роль, фото, пароль, кнопка реєстрації, а якщо акаунт вже існує, то можна перейти на сторінку авторизації. Другий метод – це створення користувача у базі даних. Сюди приходить інформація з форми, яка перевіряється правилами валідації, якщо вони пройдені, то користувач створюється. У додатку Б знаходиться код контролера для реєстрації користувача. На рисунку 3.5 зображено форму реєстрації.

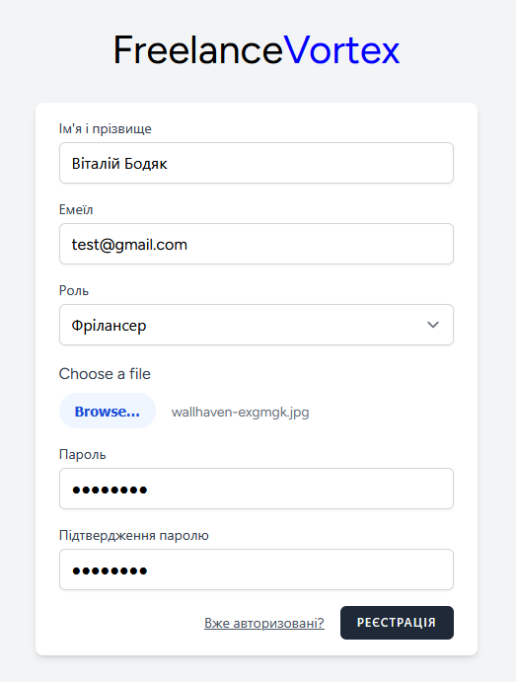

Рисунок 3.5 – Форма реєстрації користувача з заповненими формами

Після того як користувач був створений він повинен потрапити на сторінку свого кабінету. Для цього було створено нову сторінку dashboard. У лістингу 3.3 відображено react код цієї сторінки.

Лістинг 3.3 – Код сторінки Dashboard

```
import UserCard from "@/Components/UserCard";
import UsersFillter from "@/Components/UsersFillter";
import AuthenticatedLayout from "@/Layouts/AuthenticatedLayout";

import { Head } from "@inertiajs/react";

export default function Dashboard({ users }) {
  return (
    <AuthenticatedLayout
      header={
        <h2 className="text-xl font-semibold leading-tight text-gray-
800">
          Огляд
        </h2>
      }
    >

      <Head title="Dashboard" />
      <div className="py-12">
        <div className="mx-auto max-w-7xl sm:px-6 lg:px-8 flex gap-6">
          <UsersFillter/>

          <div className="w-full">
            {users && users.length > 0 ? (users.map((user) => (
              <UserCard key={user.id} user={user}/>
            ))) : (
              <h1 className="text-xl font-semibold leading-tight text-
gray-800 text-center">Нікого не знайдено...</h1>
            )}
          </div>
        </div>
      </div>
    </AuthenticatedLayout>
  );
}
```

кінець лістингу 3.3

Після чого при переходжені на роут /dashboard користувачу буде відображена ця сторінка. Вона складається з верхньої частини, у якій знаходиться логотип, навігаційна панель, фото та ім’я користувача, а також

спадаюче меню для налаштування акаунту. Знизу знаходиться головна частина, у якій з лівої сторони розміщується фільтрація пошуку, а з правої список фрілансерів, або роботодавців. На рисунку 3.6 зображено вигляд цієї сторінки.

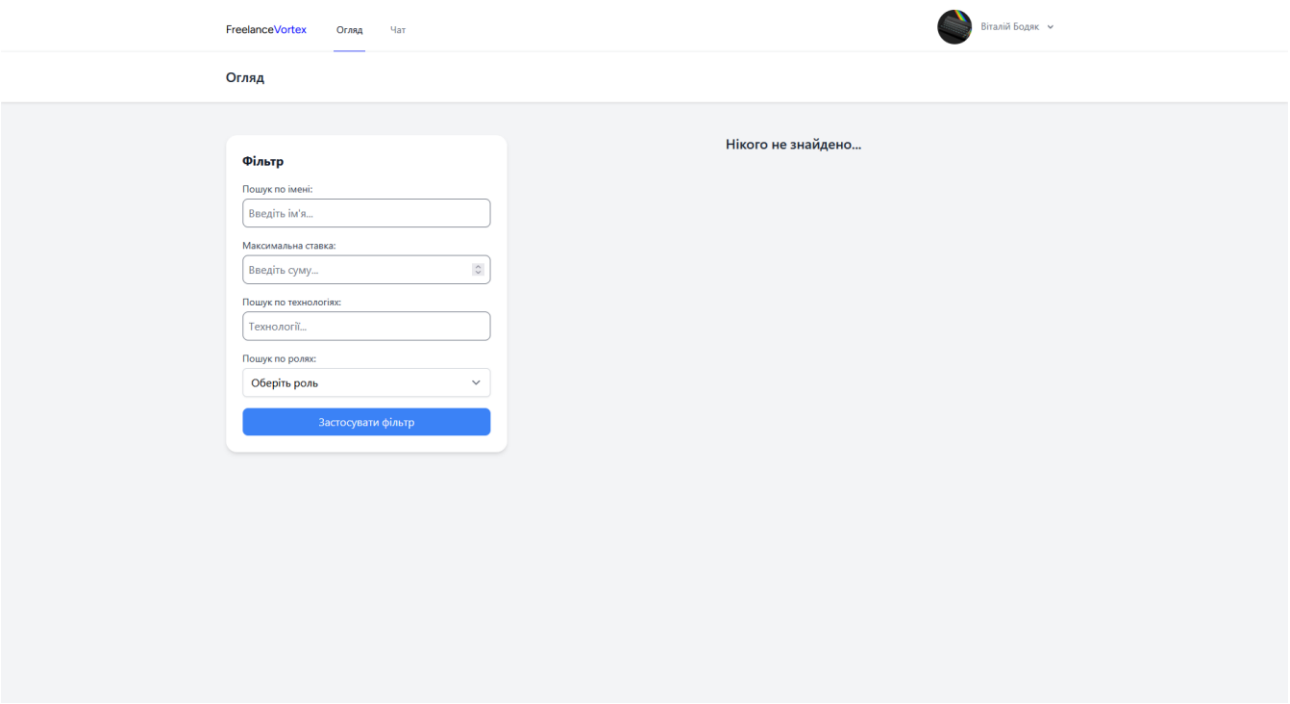

Рисунок 3.6 – Вигляд особистого кабінету користувача

Для відображення користувачів у на сторінці «Мій кабінет» обхідно створити спеціальний контролер – UserController, який відповідатиме за логіку отримання, обробки та передачі даних про користувачів до представлення.

Для реалізації фільтрації користувачів на сайті було створено окремий метод filter. Він отримує дані з форми пошуку та, залежно від того, які параметри вказав користувач, будує SQL-запит з динамічними умовами. Такий підхід забезпечує гнучкість, дозволяючи розширювати або змінювати критерії фільтрації без потреби в суттєвій перебудові логіки пошуку У лістингу 3.4 відображено код методу.

Лістинг 3.4 – Код методу фільтрації

```
public function filter(Request $request) {
    $query = User::query()->with('aboutUser')
        ->whereHas('aboutUser', function ($q) {
```

```
            $q->where('description', '!=', '')->where('skills', '!=',
''')->where('hourly_rate', '!=', '');
        });
    if ($request->filled('name')) {
        $query->where('name', 'like', '%' . $request->name . '%');
    } if ($request->filled('hourlyRate')) {
        $query->whereHas('aboutUser', function ($q) use ($request) {
            $q->where('hourly_rate', '<=', $request->hourlyRate);
        });
    } if ($request->filled('skills')) {
        $skills = array_map('trim', explode(',', $request->skills));

        foreach ($skills as $skill) {
            $query->whereHas('aboutUser', function ($q) use ($request,
$skill) {
                $q->where('skills', 'like', '%' . $skill . '%');
            });
        }
    } if ($request->filled('role')) {
        $query->where('role', $request->role);
    }
    $users = $query->with(['aboutUser'])->get();
    return Inertia::render('Dashboard', ['users' => $users]);
}
```

кінець лістингу 3.4

Також необхідно створити реакт компонент для відображення інформації про користувача в списку. Для цього було створено UserCard.jsx. У додатку В відображено код цього компоненту.

Для впровадження функціональності чату в рамках веб-проекту було використано передові технології Pusher та Chatify, які гарантують ефективну організацію взаємодії користувачів у режимі реального часу. Застосування цих інструментів дало змогу розробити надійний, масштабований та інтерактивний чат, що відповідає сучасним стандартам веб-додатків.

Pusher – це хмарний сервіс, який забезпечує механізм обміну даними в реальному часі (real-time messaging) за допомогою протоколу WebSocket. Головною перевагою Pusher є його здатність автоматично управляти з'єднаннями, гарантувати доставку повідомлень з мінімальною затримкою та підтримувати масштабування зі зростанням кількості користувачів. Завдяки Pusher, реалізація двосторонньої комунікації в чаті значно спростилася, адже він

абстрагує складність роботи з WebSocket на низькому рівні. Окрім того, сервіс надає механізми авторизації, що дозволяють організувати безпечний обмін повідомленнями виключно між авторизованими користувачами.

Chatify – це фреймворк, спеціально розроблений для швидкого створення функціональності чату у проектах на основі PHP та Laravel. Chatify забезпечує базову структуру для зберігання повідомлень, керування користувачами, груповими чатами та підтримки медіа-контенту. Застосування Chatify дає можливість уникнути значних витрат часу на розробку чату «з нуля», оскільки він включає готові компоненти, інтегровані з популярними бібліотеками фронтенду. Користувацький інтерфейс, реалізований через Chatify, є зручним, інтуїтивно зрозумілим та забезпечує швидку взаємодію без потреби перезавантаження сторінки.

Для роботи з цими технологіями необхідно спочатку зареєструватися на платформі Pusher. Там потрібно отримати ключі, після чого вказати їх в .env файлі в спеціальних полях. На рисунку 3.7 зображено поля для ключів Pusher.

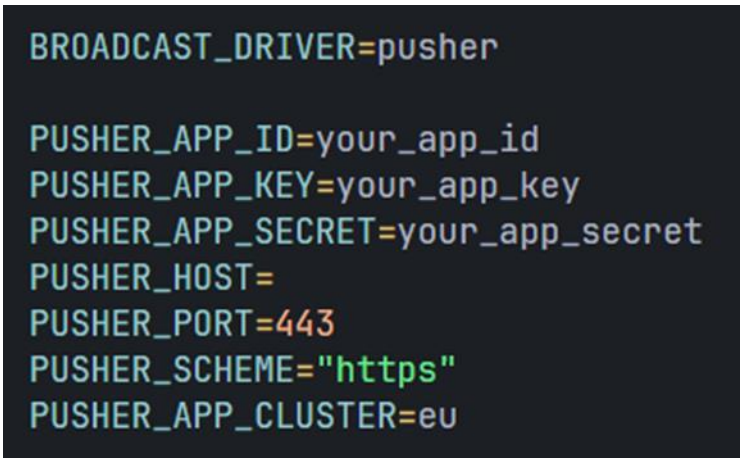

Рисунок 3.7 – Поля для ключів Pusher

Після цього необхідно інсталювати пакет Chatify за допомогою команди composer require munafio/chatify. Вона створить всі необхідні сторінки, міграції, а також моделі для роботи з чат-повідомленнями та користувачами. На рисунку 3.8 зображена сторінка месенджеру Chatify.

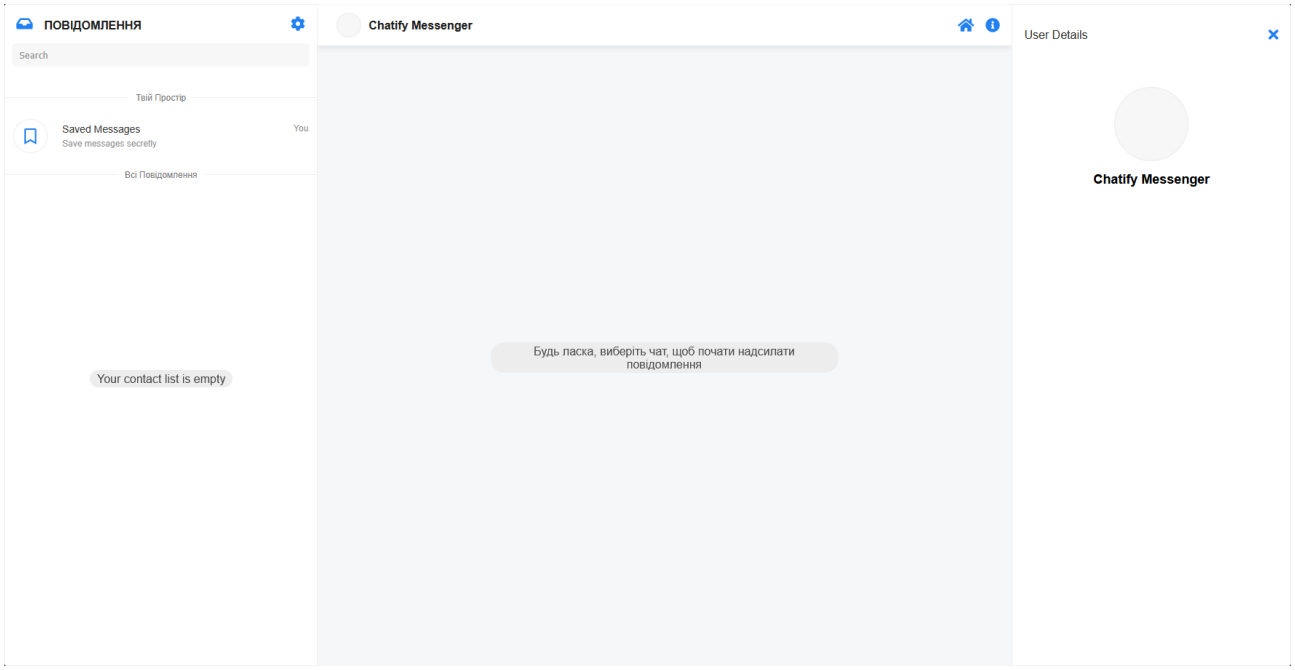

Рисунок 3.8 – Месенджер Chatify

У ході виконання кваліфікаційної роботи було розроблено ключові маршрути, що гарантують функціональну взаємодію користувача із системою. Зокрема, ці маршрути відповідають за автентифікацію та авторизацію, охоплюючи процедури входу, виходу, реєстрації, відновлення та зміни пароля. Також розроблено шляхи для управління профілем користувача, його персональними налаштуваннями та резюме. Особливу увагу було зосереджено на маршрутах, які забезпечують підтвердження електронної пошти та захист від CSRF-атак, використовуючи Sanctum. Крім того, передбачені маршрути для роботи з даними користувачів, їх фільтрації, а також для організації доступу до локального сховища файлів. Впроваджено функціонал відгуків користувачів, що спрямовано на покращення якості сервісу. Маршрути наведені у таблиці 3.1.

Таблиця 3.1 – Основні маршрути проекту

Метод	Шлях	Опис
GET	/broadcasting/auth	Аутентифікація для трансляцій
POST	/broadcasting/auth	Аутентифікація для трансляцій
GET	/confirm-password	Показати форму підтвердження пароля
POST	/confirm-password	Підтвердити пароль
GET	/dashboard	Панель користувача
POST	/email/verification-notification	Надіслати повторне підтвердження email
GET	/fillter	Фільтрація (фільтр даних)

Продовження таблиці 3.1

Метод	Шлях	Опис
GET	/forgot-password	Форма для відновлення пароля
POST	/forgot-password	Відправити посилання на відновлення пароля
GET	/login	Форма для входу
POST	/login	Виконати вхід користувача
POST	/logout	Вийти з системи
PUT	/password	Оновлення пароля
GET	/profile	Показати профіль користувача
PUT	/profile	Оновлення резюме у профілі
GET	/profile/settings	Показати налаштування профілю
PATCH	/profile/settings	Оновлення налаштувань профілю
DELETE	/profile/settings	Видалити налаштування профілю
GET	/register	Форма реєстрації
POST	/register	Зареєструвати нового користувача
POST	/reset-password	Встановити новий пароль
GET	/reset-password/{token}	Форма скидання пароля за токеном
GET	/sanctum/csrf-cookie	Отримання CSRF cookie для Sanctum
GET	/user-data	Отримати список користувачів
GET	/user-profile/{user}	Профіль конкретного користувача
POST	/user-profile/{user}/review	Залишити відгук про користувача
GET	/verify-email	Повідомлення про необхідність підтвердження email
GET	/verify-email/{id}/{hash}	Підтвердження email за посиланням

3.2 Розробка бази даних

Одним із завдань кваліфікаційної роботи була розробка бази даних. За основу було взято реляційну базу даних MySQL. MySQL – це реляційна система управління базами даних (СКБД) з відкритим вихідним кодом, яка дозволяє користувачам ефективно зберігати, керувати та отримувати структуровані дані [14]. Для взаємодії з нею було використано програму Dbeaver. DBeaver Community – це безкоштовний крос-платформний інструмент для роботи з базами даних для розробників, адміністраторів баз даних, аналітиків та всіх, хто працює з даними. Він підтримує всі популярні бази даних SQL, такі як MySQL, MariaDB, PostgreSQL, SQLite, Apache Family та інші [4].За допомогою нього можна створювати бази даних, їх таблиці, виконувати запити, транзакції, а також відображати дані в форматі діаграм.

Для створення таблиць в базі даних використовувалися міграції, які є частиною функціоналу Laravel. Вони дозволяють описувати структури таблиць

такі як ідентифікатори, поля, зовнішні ключі та інше за допомогою синтаксису PHP. Також міграції дозволяють швидко вносити зміни або модифікувати структуру бази даних.

Для створення міграції використовується команда: php artisan make:migration. Після її виконання з’явиться повідомлення про те, що міграція успішно створена, або якщо вона вже створена, то помилка.

На рисунку 3.9 зображено приклад виконання команди.

Рисунок 3.9 – Результат виконання команди створення міграції

Після цього необхідно вказати всі необхідні поля, зовнішні ключі для таблиці. У лістингу 3.5 відображено код міграції для створення таблиці користувача.

Лістинг 3.5 – Код міграції створення таблиці користувача

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
```

```
            $table->string('email')->unique();
            $table->string('role');
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->string('avatar');
            $table->rememberToken();
            $table->timestamps();
        });
    }
};
```

кінець лістингу 3.5

В ході аналізу завдання на кваліфікаційну роботу, було визначено, що необхідно створити такі таблиці у базі даних як: user, user_reviews, about_user, user_rating, report, sessions, ch_messages, ch_favorites.

На рисунку 3.10 зображена діаграма зі створеними таблицями та їх зв'язками у базі даних.

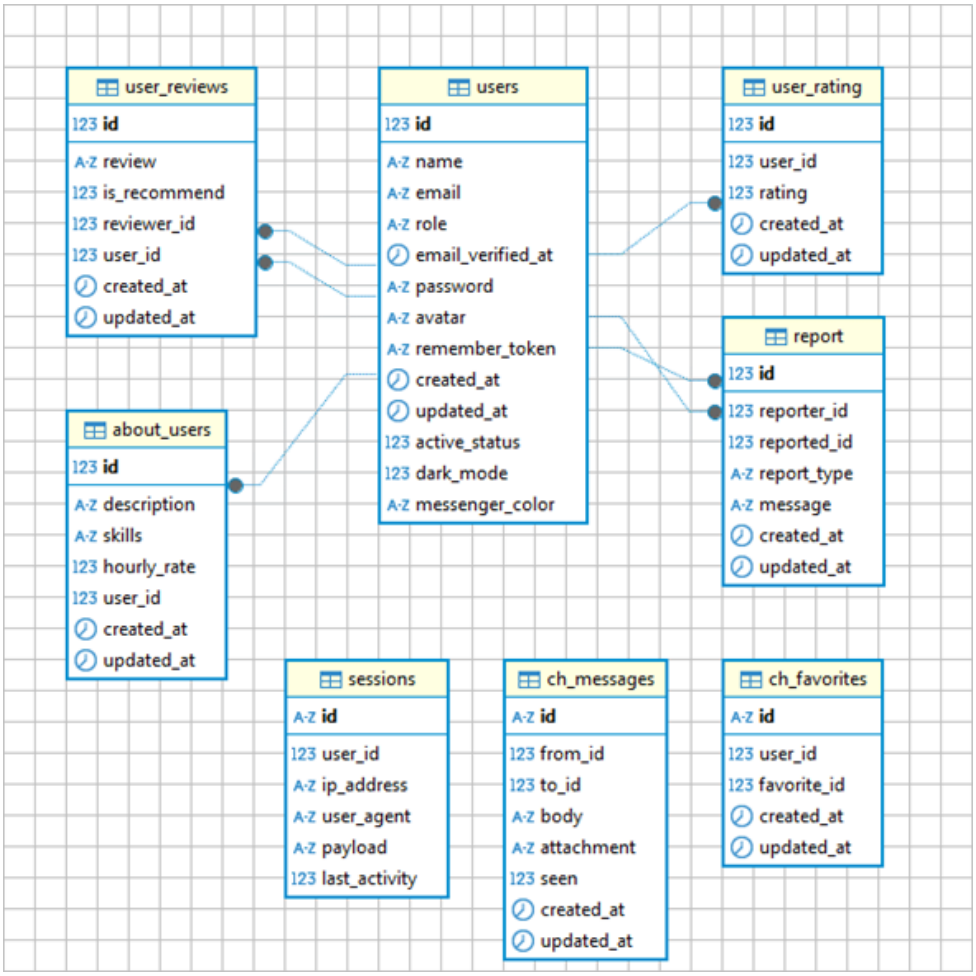

Рисунок 3.10 – Діаграма створених таблиць у базі даних

Центральною сутністю в структурі бази даних є таблиця users, яка виконує роль основного джерела інформації про користувачів системи. Вона містить усі необхідні атрибути, що описують користувача як об’єкт доменної моделі: унікальний ідентифікатор, облікові дані, контактну інформацію, часові мітки.

У таблиці users використовуються два типи зв’язків з іншими таблицями бази даних, що забезпечує можливість реалізації складних відносин між сутностями. Перший зв’язок – це «один до одного» (one-to-one), використовується, з таблицею about_user, де зберігається додаткова інформація про користувача. Другий – це «багато до багатьох» (many-to-many), використовується, з таблицями user_reviews, user_rating, report.

Також для використання цих таблиць в середині проекту, було створено моделі таблиць бази даних. Модель – це частина шаблону Model-View-Controller (MVC), яка відповідає за взаємодію з базою даних. Вона являється об’єктно-орієнтованим представленням таблиці. У лістингу 3.6 відображено код молелі Report, яка відповідає за збереження репортів про користувачів.

Лістинг 3.6 – Код моделі Report

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;

class Report extends Model
{
    protected $fillable = [
        ‘reporter_id’,
        ‘reported_id’,
        ‘report_type’,
        ‘message’
    ];

    public function reporter(): BelongsTo
    {
        return $this->belongsTo(User::class, ‘reporter_id’);
    }

    public function reported(): BelongsTo
```

```
    {
        return $this->belongsTo(User::class, ‘reported_id’);
    }
}
```

кінець лістингу 3.6

3.3 Захист інформаційно-комп’ютерної системи

Одним з методів реалізації захисту в Laravel є система автентифікації та авторизації. Laravel має вбудовані інструменти для створення системи ідентифікації користувачів, що використовує сесії або токени доступу, такі як Laravel Sanctum чи Laravel Passport. З огляду на використання Inertia.js, який не застосовує REST API у звичній формі, а натомість послуговується серверним рендерингом JSON-відповідей для клієнтської частини, механізми автентифікації мусять бути інтегровані в контекст традиційних веб-сесій. Це забезпечує надійний захист від атак, таких як «перехоплення токена» або повторне використання сесії, якщо правильно налаштовані обмеження тривалості сесії, механізми тайм-аутів та перевірок CSRF.

Перевагою використання Inertia.js у порівнянні з класичним REST-підходом є зменшення площини для атак, пов’язаних з JSON API, зокрема – уникнення проблем, пов’язаних з CORS, оскільки запити відправляються в межах тієї ж сесії та домену. Але це не виключає потреби впровадження захисту від міжсайтових підробок запитів (Cross-Site Request Forgery). Laravel має засоби захисту від CSRF, додаючи CSRF-токен до форм автоматично та перевіряючи його на сервері. Під час розробки з Inertia та React треба слідкувати, щоб всі форми, що відправляють дані на сервер, містили актуальний CSRF-токен, а механізм його генерації був коректно інтегрований в шаблони Blade або в початковий стан сторінки.

Наступним критичним аспектом є управління доступом до ресурсів, яке реалізується за допомогою політик (policies) і гейтів (gates) Laravel. Авторизація є важливою для гарантування того, що користувач має право виконувати певну дію. Особливо це важливо, коли React-компоненти можуть динамічно

запитувати дані або взаємодіяти з сервером через Inertia-запити. Отже, необхідно, щоб перевірка доступу виконувалась не тільки на клієнтському інтерфейсі (який може бути змінений користувачем), а й дублювалась суворою перевіркою на боці сервера. Така перевірка повинна бути інтегрована у контролери або у відповідні політики, які призначені для кожного типу ресурсу.

Для шифрування інформації було використано криптографічну бункцію bcrypt. Bcrypt – це потужний алгоритм хешування паролів, який спеціально розроблений для підвищення безпеки зберігання паролів у системі [18].

Захист від атак типу SQL-ін’єкції в Laravel досягається завдяки використанню Eloquent ORM або Query Builder, які автоматично екранують вхідні дані. SQL-ін’єкція – це метод отримання несанкціонованого доступу (різновид злому) до бази даних, при якому шкідливий код виконується прямо з поля вводу звичайної форми [11].

React як фронтенд-феймворк розроблений з урахуванням принципів безпеки, зокрема для захисту від атак типу XSS (Cross-Site Scripting). XSS – це тип уразливості, який дає змогу зловмисникам імплементувати шкідливий код на сторінку, яку переглядає користувач. React автоматично екранує будь-який вміст, який виводиться у DOM за допомогою JSX. Під час роботи з формами React застосовує односпрямований потік даних та контрольовані компоненти (controlled components), що гарантує централізований контроль над значеннями полів. Завдяки цьому формується ефективна модель перевірки введених даних (input validation), котра може бути доповнена механізмами клієнтської та серверної валідації задля запобігання впровадженню небезпечного контенту. Шаблонізатор Blade від Laravel автоматично екранує всі дані, що виводяться, щоб запобігти XSS [1]. Це означає, що навіть якщо користувач введе шкідливий код, то React та Blade не інтерпретують це як HTML, а виведуть звичайний текст.

На рисунку 3.11 наведено приклад роботи XSS ін’єкції.

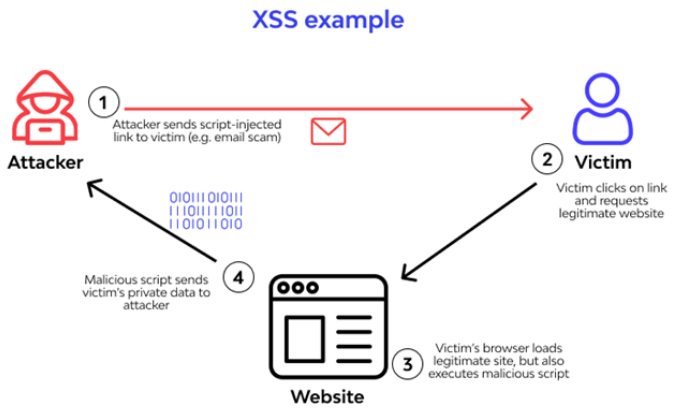

Рисунок 3.11 – Принцип роботи XSS ін’єкції [3]

Також при роботі з React необхідно передбачити механізми захисту користувача при зберіганні стану програми. Якщо додаток використовує локальне або сесійне сховище, в ньому не повинні зберігатися спеціальні дані, такі як токени або ідентифікаційна інформація.

Головною особливістю використання Inertia як моста між клієнтом і сервером є те, що вся логіка маршрутизації реалізується на сервері, а клієнтська частина тільки відображає компонент відповідно до отриманих даних. Це дає додатковий рівень контролю над безпекою маршрутів, оскільки вони повністю керуються через Laravel, що дає змогу централізовано застосовувати middleware, політики доступу та інші механізми перевірки. Важливо, щоб кожен маршрут, який оперує з чутливими або критичними операціями, був захищений відповідним middleware і проходив належну авторизацію.

3.4 Модульне розширення та масштабування системи

В сучасних інформаційних системах, особливо в тих, що розраховані на велику кількість користувачів та складний функціонал, надзвичайно важлива здатність програмного забезпечення до масштабування і гнучкого розширення. Розширення функціональності, підтримка нових можливостей і адаптація до зростаючих навантажень неможливі без належної архітектурної організації коду. Одним із ключових підходів, що забезпечує ефективність подальшої розробки, є

модульне розширення системи, при якому логіка програми ділиться на окремі логічні складові – модулі або пакети. Цей підхід дозволяє ізолювати різні функціональні частини програми, забезпечуючи високий ступінь повторного використання коду, легкість у підтримці, а також полегшує масштабування системи.

Модульна архітектура передбачає поділ системи на незалежні або напівзалежні компоненти, які містять у собі повний набір необхідних для виконання завдань ресурсів – від бізнес-логіки до схеми збереження даних. В рамках створення сучасних веб-застосунків часто використовується розподіл на пакети, кожен з яких може розвиватися незалежно від інших, маючи власну структуру та функціональність. Однією з основних переваг підходу з модульним розширенням є полегшення підтримки та розвитку системи. Розробникам набагато легше працювати з відокремленими частинами програми, які мають чітко визначену зону відповідальності. Це зменшує ймовірність помилок, пов’язаних із взаємодією між компонентами, та прискорює процес внесення змін або додавання нового функціоналу.

Крім того, модульна архітектура забезпечує кращу масштабованість. Оскільки окремі пакети розробляються та оптимізуються незалежно, можна зосереджувати ресурси на оптимізації саме тих частин системи, які найактивніше використовуються або вимагають поліпшень. Такий підхід сприяє більш ефективному використанню серверних ресурсів, дає можливість розподіляти навантаження між різними сервісами, а також полегшує впровадження балансування навантаження.

Кожен пакет містить власні міграції бази даних, що дозволяє управляти структурою даних незалежно від інших частин системи. Наприклад, пакет оплати містить міграції для таблиць, що пов’язані з транзакціями, платіжними методами, історією платежів, а пакет реєстрації – міграції для таблиць користувачів, ролей, прав доступу. Такий підхід дозволяє ізолювати зміни у базі даних, мінімізуючи ризик виникнення конфліктів при одночасній роботі над

різними функціональними блоками. На рисунку 3.12 зображено приклад структури пакету оплати.

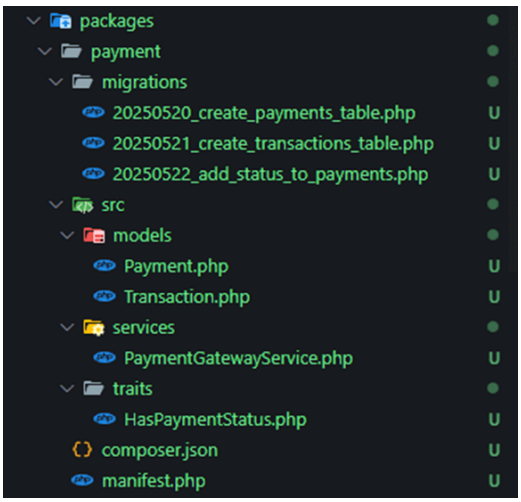

Рисунок 3.12 – Структура пакету оплати

Крім того, у складі кожного пакету розміщені моделі, які реалізовують бізнес-логіку предметної області, що пов’язана з конкретним модулем. Наприклад, у пакеті оплати будуть моделі транзакцій, платіжних сесій, у пакеті управління проектами – моделі завдань, проектів, користувачів, пов’язаних із проектом. Використання моделей дозволяє централізувати роботу з даними, що сприяє підтримці цілісності і послідовності інформації.

Окремо слід зазначити використання трейтів для повторного використання, які можна підключати до різних класів для розширення їх функціональності без дублювання коду. Трейти можуть містити, наприклад, загальні методи валідації, логування, формування відповіді користувачу. Розміщення трейтів безпосередньо в межах пакету дозволяє краще організувати код та підвищити його повторне використання.

3.5 Тестування та налагодження інформаційно-комп’ютерної системи

По завершенні основного етапу розробки інформаційно-комп’ютерної системи «FreelanceVortex», значну увагу було приділено тестуванню. Цей етап

відіграє вирішальну роль у забезпеченні стабільності, безпеки та відповідності системи заявленим функціям. Тестування дозволяє не лише виявити логічні помилки у роботі, а й своєчасно усунути потенційні вразливості, що можуть бути використані зловмисниками для несанкціонованого доступу або порушення роботи системи.

Одним з найважливіших напрямків тестування стала перевірка захисту від SQL-ін’єкцій. Це різновид атаки, при якій зловмисник вводить спеціально сформований SQL-код у поля введення з метою виконання небажаних запитів до бази даних. Неправильна фільтрація запитів може призвести до витоку конфіденційних даних або навіть повного знищення бази даних. Щоб запобігти цьому, під час розробки було впроваджено механізми використання параметризованих запитів, які забезпечують автоматичне екранування вхідних даних. У ході тестування здійснювалися спроби ручного введення коду через форми пошуку, реєстрації, авторизації. На рисунку 3.13 зображено приклад введення SQL-ін’єкції в формі авторизації.

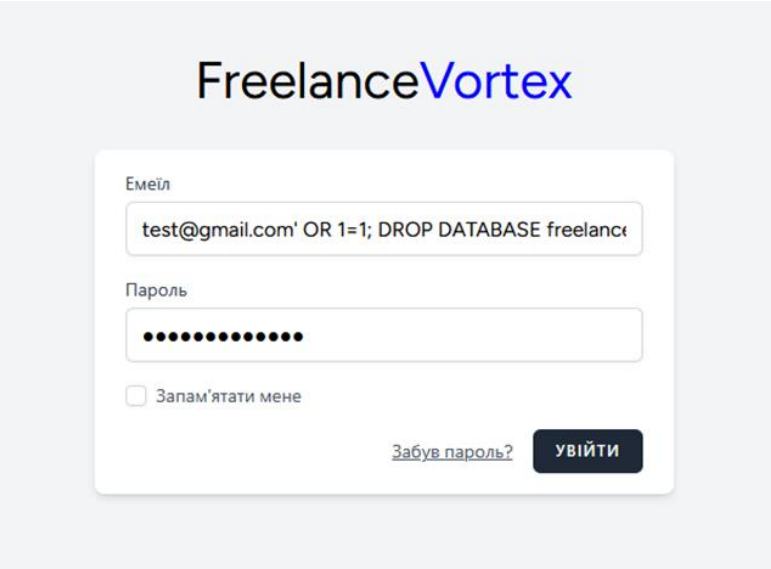

Рисунок 3.13 – Введення SQL-ін’єкції в формі авторизації

Після введення цього шкідливого коду, відбувається декілька етапів обробки цього запиту. Спочатку React отримує цей рядок з форми, екранує його і проводить валідації, якщо валідація була не правильно налаштована, то

відбувається передача даних на сервер на якому Laravel приймає ці дані, та розбиває їх, після чого частина з кодом ігнорується, що запобігає виконанню шкідливих дій.

Не менш важливою є перевірка на XSS атаки. XSS – тип вразливості інтерактивних інформаційних систем у вебі [17]. Цей код може використовуватися для викрадення cookie-файлів, перенаправлення користувачів на фішингові сайти або відображення недостовірної інформації. Для мінімізації ризику XSS атак було впроваджено політику автоматичного екранування HTML-тегів у користувацьких повідомленнях, коментарях, назвах проєктів та інших динамічних елементах. На рисунку 3.14 зображено введення шкідливого коду у формі налаштування профілю.

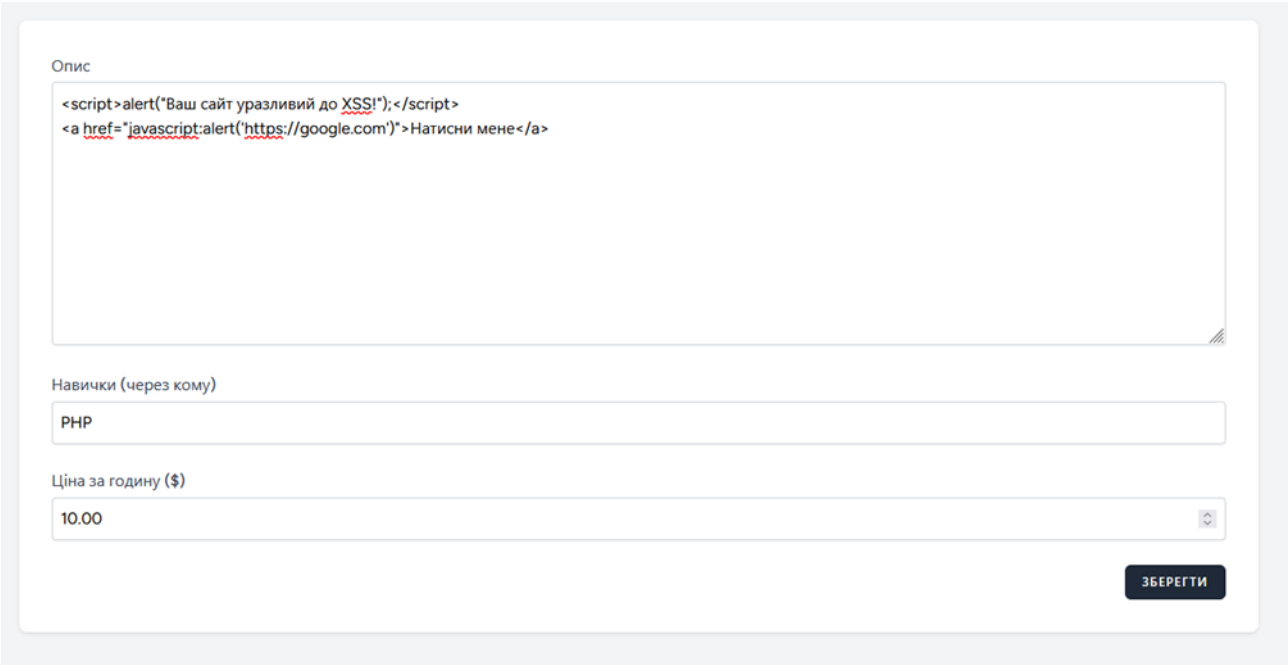

Рисунок 3.14 – Введення шкідливого коду в форму

Оскільки React провів всі етапи по обробці даних форми, то на сервер прийшов не шкідливий код, а звичайний текст і ці дані було збережно у базу даних. Якщо б ці механізми не працювали б, то на сторінці було б виведено повідомлення і створилося б посилання, на сайт, але цього не відбувається. На рисунку 3.15 вигляд сторінки користувача після невдалої XSS-ін’єкції.

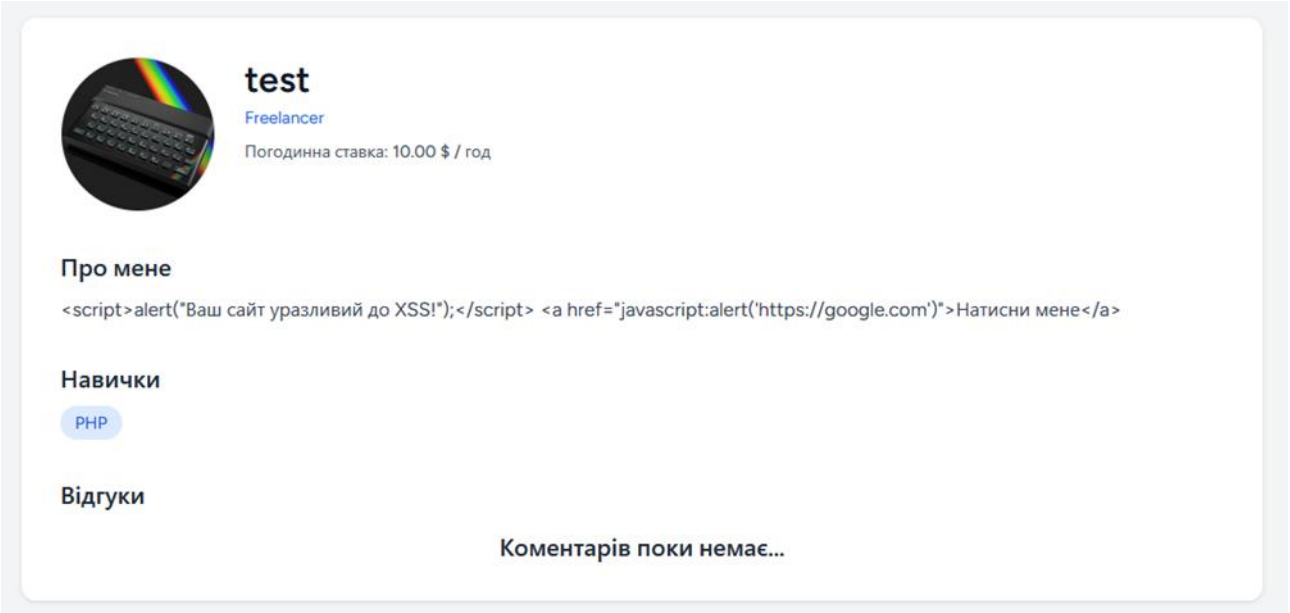

Рисунок 3.15 – Сторінка користувача після невдалої XSS-ін’єкції

Окрему увагу було приділено перевірці захисту від CSRF-атак (Cross-Site Request Forgery), CSRF – або Міжсайтова підробка запиту – це метод, яким зловмисний користувач намагається змусити ваших користувачів, не знаючи про це, відправляти дані, які вони не збиралися відправляти [19]. Всі форми, які відправляються користувачем, містять унікальний CSRF-токен, що перевіряється на стороні сервера. У процесі тестування робилися спроби надсилати POST-запити без дійсного токену, що дозволило переконатися у тому, що сервер не обробляє такі запити.

Одним із потенційних векторів атаки є неналежне використання функції JSON.stringify(), яка перетворює будь-які дані у рядкове представлення без вбудованих механізмів виявлення шкідливих значень. Це створює можливості для зловмисників маніпулювати, зокрема, такими даними, як ім’я користувача або пароль, шляхом вставлення спеціально сформованих об’єктів, здатних модифікувати легітимні параметри [7].

Також важливим етапом була перевірка автентифікації та авторизації. Тестувалися сценарії, коли користувачі намагалися отримати доступ до сторінок або функцій, які не відповідають їхній ролі. Наприклад, фрілансер не повинен мати доступу до панелі модератора чи адміністратора, а неавторизований

користувач – до внутрішнього функціоналу. Для цього було проведено ручне тестування маршрутизації, перевірку middleware. На рисунку 3.16 зображено логи, які були записані в момент переходу не авторизованого користувача на сторінку /dashboard, на котру у нього немає дозволу.

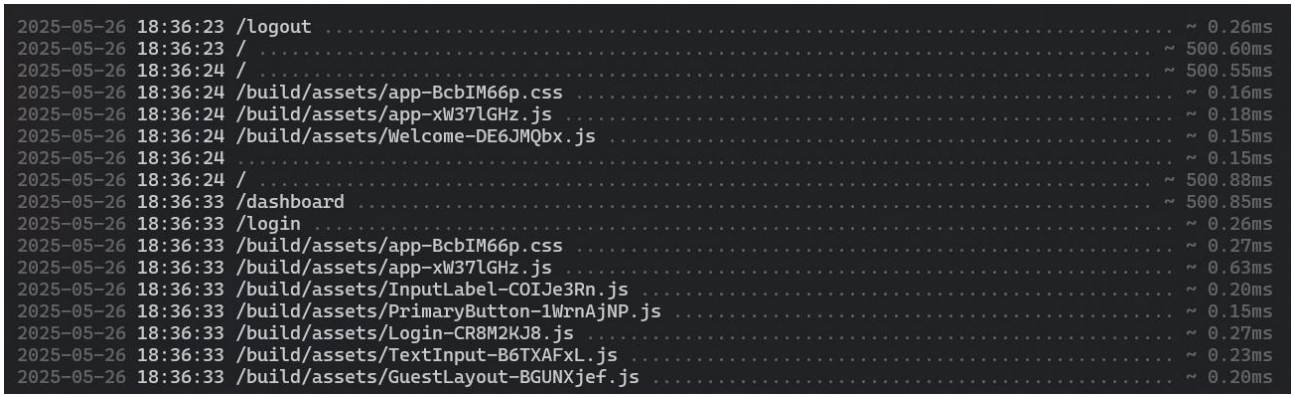

Рисунок 3.16 – Логи дій неавторизованого користувача

Після переходу користувача було перенаправлено на сторінку авторизації. На рисунку 3.17 зображена сторінка авторизації, на котру потрапляють користувачі без авторизації.

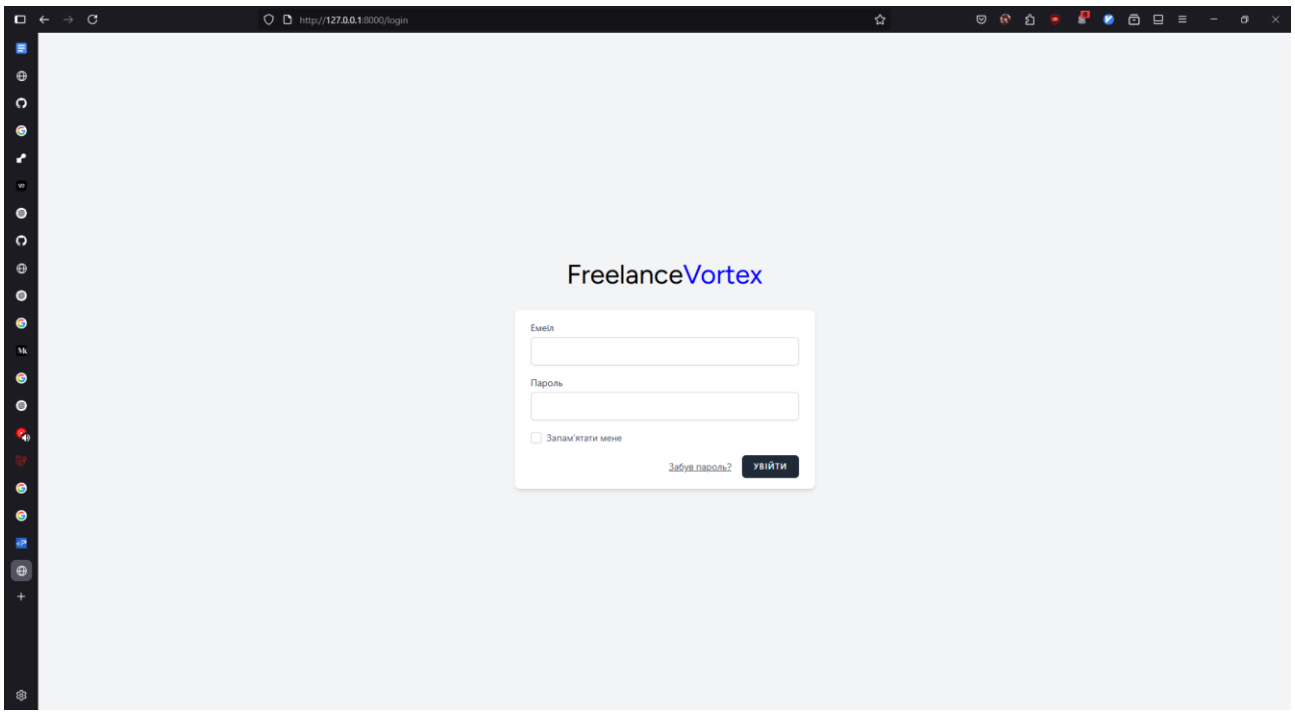

Рисунок 3.17 – Сторінка авторизації

Також було передбачено, що якщо користувач спробує перейти на сторінку якої не існує, то його буде перенаправлено на сторінку 404.

Одним із ключових механізмів у цьому контексті є використання захищених протоколів зв’язку. HTTP-заголовок Strict-Transport-Security (HSTS) використовується для підвищення безпеки веб-сайту, вказуючи браузерам завжди використовувати HTTPS для всіх майбутніх запитів до сайту. Це допомагає захиститися від різних типів атак, таких як атаки типу «зловмисник посередині», гарантуючи, що з’єднання з сервером зашифроване [2].

Крім перевірки на вразливості, було проведено функціональне тестування ключових компонентів, зокрема реєстрації, входу, створення проєктів, перегляду заявок, взаємодії через чат та іншого. Для цього використовувалися як ручні, так і автоматизовані методи тестування. Ручне тестування дозволило моделювати поведінку реального користувача та виявляти нестандартні сценарії, які важко передбачити заздалегідь.

Висновки до розділу 3

В результаті розробки було створено повнофункціональний веб-додаток «FreelanceVortex». Було використано фреймворк Laravel для серверної частини, React з Inertia.js для фронтенду, а також Tailwind CSS для адаптивної візуалізації. Система забезпечує зручну взаємодію між фрілансерами та замовниками, реалізовуючи повний цикл співпраці.

У процесі розробки бази даних було спроектовано та реалізовано реляційну модель даних, яка описує основні складові системи: користувачів, заявки, повідомлення та ролі. Завдяки коректній нормалізації структури було забезпечено мінімізацію надмірності даних та збереження цілісності зв’язків між таблицями. Під час розробки враховано потенційну потребу в масштабуванні бази даних при збільшенні кількості користувачів.

Захист інформаційно-комп’ютерної системи здійснено згідно з сучасними вимогами кібербезпеки. Впроваджено механізми автентифікації та авторизації

на основі ролей, захист від SQL-ін’єкцій, XSS-атак, CSRF-запитів, а також шифрування конфіденційної інформації. Це дало змогу значно знизити ризики, пов’язані з обробкою особистої та фінансової інформації користувачів. Окрему увагу приділено безпечній серіалізації даних у середовищі JavaScript.

У рамках модульного розширення та масштабованості системи було визначено можливості системи у розширені та розділенні на окремі незалежні пакети.

Також було проведено комплексне тестування та налагодження системи, під час якого були перевірені функціональні можливості та безпека сайту. Особливу увагу приділено виявленню критичних уразливостей, перевірці прав доступу, правильності обробки помилок.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було проаналізовано та реалізовано всі етапи розробки інформаційної системи «FreelanceVortex», що орієнтована на взаємодію між замовниками та виконавцями фріланс-проєктів.

Початковим етапом став аналітичний блок, в рамках якого здійснено ґрунтовний аналіз актуального стану проблеми автоматизації пошуку спеціалістів у сфері фрілансу. Ринок веб-розробок та віддаленої роботи динамічно розвивається, але більшість існуючих платформ або перенасичені зайвим функціоналом, або не забезпечують належного рівня зручності та адаптивності. Відсутність локалізованих і технологічно гнучких рішень формує нішу, що може бути заповнена за рахунок створення спеціалізованої системи з акцентом на користувацький досвід, швидкість роботи та можливості розширення. Проведений аналіз наявних аналогів дозволив виокремити основні обмеження існуючих рішень та окреслити шляхи їх подолання в рамках запропонованого проєкту.

На основі результатів аналітичного етапу було сформульовано завдання для кваліфікаційної роботи. Основною метою визначено створення інформаційної системи, що забезпечує ефективну взаємодію між користувачами з різними ролями, з реалізацією механізмів пошуку, фільтрації, комунікації, зберігання відгуків та профілів користувачів. Додатковими вимогами визначено потребу у підтримці адаптивного дизайну, побудові системи рейтингу, впровадженні механізмів захисту даних та забезпеченні можливості масштабування шляхом інтеграції нових модулів без порушення цілісності архітектури.

Після уточнення цілей було сформульовано функціональні та нефункціональні вимоги до веб-сайту. Серед функціональних вимог виокремлено необхідність у: реєстрації та автентифікації користувачів, створенні та управлінні профілями, підтримці внутрішньої системи повідомлень, реалізації фільтрації та сортування фрілансерів за навичками.і іншими

параметрами. До нефункціональних вимог віднесено стабільність функціонування при високому навантаженні, швидкий час відгуку інтерфейсу, безпечне зберігання даних, а також можливість подальшого розширення функціоналу.

Також було обґрунтуванно вибір технологій. Як основну серверну платформу обрано фреймворк Laravel, який надає чітку архітектурну структуру, потужну ORM-систему (Eloquent), вбудовану підтримку безпеки та широкі можливості RESTful-розробки. На фронтенді використано React з інтеграцією через Inertia.js, що дозволяє поєднати гнучкість SPA-додатку з перевагами традиційного серверного рендерингу. Tailwind CSS використано як інструмент швидкого створення адаптивного інтерфейсу з гнучкою кастомізацією компонентів. Для організації чату в режимі реального часу інтегровано Pusher та бібліотеку Chatify, що дозволяє реалізувати асинхронну комунікацію між користувачами.

Практична реалізація об’єкта проєктування відбувалася з використанням модульного підходу. Веб-додаток було структуровано за принципом розподілу відповідальності між модулями: аутентифікація, управління профілем, проєкти, повідомлення, фільтрація користувачів, адміністрування. Це забезпечує можливість масштабування функціоналу системи. Було створено окремі контролери, шаблони представлень та компоненти інтерфейсу, які легко адаптуються або перевизначаються при зміні вимог користувача.

Було проведено проєктування та реалізація бази даних. Розроблено реляційну модель даних, що включає таблиці користувачів, повідомлень, категорій та інших таблиць. Нормалізація таблиць дала змогу уникнути надлишковості, а використання зовнішніх ключів забезпечило цілісність даних. За допомогою Eloquent ORM було реалізовано зручну логіку запитів, що дає змогу ефективно взаємодіяти з базою даних на рівні бізнес-логіки.

Особлива увага була приділена безпеці системи. Визначено та впроваджено принципи захисту від основних типів уразливостей, таких як SQL-ін’єкції, міжсайтове виконання скриптів (XSS), підробка міжсайтових запитів

(CSRF), атаки перебором паролів. Для захисту конфіденційної інформації було реалізовано шифрування паролів за допомогою bcrypt, а також токенізацію для управління сесіями користувачів. Всю взаємодію з критичними компонентами системи було ізольовано відповідними middleware-фільтрами.

В рамках аналізу масштабованості платформи розглянуто можливості її розширення через модульну архітектуру. Завдяки застосуванню компонентної структури React, шаблонів Laravel Blade та REST-орієнтованої організації логіки стало можливим додавання нових функціональних блоків без конфліктів з наявним кодом. Наприклад, система чату, рейтингів, адміністративна панель або аналітичні модулі можуть бути легко інтегровані в загальну архітектуру проєкту.

Було здійснено тестування та налагодження інформаційної системи. Проведено функціональне тестування основних модулів, ручне тестування з імітацією дій користувача, а також первинне навантажувальне тестування. Окрему увагу було приділено тестам на уразливість: перевірено обробку несанкціонованих запитів, спроб виконання шкідливого коду, ін’єкцій, а також введення нестандартних даних. Всі виявлені проблеми було оперативно усунено, що дозволило досягти стабільної та безпечної роботи системи.

Розроблена платформа FreelanceVortex має великий потенціал для подальшого розвитку, що дасть змогу розширити її функціонал та поліпшити якість обслуговування клієнтів. Одним з перспективних напрямків є інтеграція сучасних технологій штучного інтелекту та машинного навчання. Зокрема, впровадження алгоритмів рекомендаційних систем на основі аналізу даних про поведінку користувачів дасть змогу пропонувати найрелевантніших виконавців або проєкти, беручи до уваги особисті вподобання та історію взаємодії. Такий підхід значно збільшить ефективність пошуку та скоротить час на вибір відповідного спеціаліста.

До того ж, можливе використання методів обробки природної мови для автоматичного аналізу й класифікації текстового контенту, зокрема описів проєктів та відгуків користувачів. Це допоможе автоматизувати модерацію, поліпшити точність фільтрації за тематикою й рівнем складності завдань, а також

виявляти потенційні порушення політик платформи. Аналіз настроїв у відгуках також може сприяти формуванню об’єктивних рейтингів виконавців.

Також перспективним є розвиток мобільних додатків для основних операційних систем, що дозволить значно підвищити доступність платформи та забезпечить користувачам зручний інтерфейс для роботи з мобільних пристроїв.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 12 top security best practices for your laravel application. SaaSykit Laravel SaaS Starter Kit. URL: https://saasykit.com/blog/12-top-security-best-practices-for-your-laravel-application (дата звернення: 10.02.2025).

2. 8 security best practices in laravel. Medium. URL: https://medium.com/@dsjayamal/8-security-best-practices-in-laravel-ad7513798cfb (дата звернення: 10.02.2025).

3. Cross site scripting (XSS) attack – Types and Examples. Wallarm. Advanced API Security. URL: https://www.wallarm.com/what/what-is-xss-cross-site-scripting (date of access: 10.02.2025).

4. DBeaver community – free universal database tool. URL: https://dbeaver.io/ (дата звернення: 10.02.2025).

5. Fiverr. URL: https://www.fiverr.com (дата звернення: 10.02.2025).

6. Hire freelancers & find freelance jobs online. Freelancer. URL: https://www.freelancer.com/ (дата звернення: 10.02.2025).

7. How to secure your react.js application. URL: https://www.freecodecamp.org/news/best-practices-for-security-of-your-react-js-application/ (date of access: 11.02.2025).

8. Inertia. URL: https://inertiajs.com/ (дата звернення: 11.02.2025).

9. Laravel. Вікіпедія. URL: https://uk.wikipedia.org /wiki/Laravel (дата звернення: 11.02.2025).

10. React. Вікіпедія. URL: https://uk.wikipedia.org/wiki/React (дата звернення: 11.02.2025).

11. SQL-ін’єкції. Acode. URL: https://acode.com.ua/sql-injection/ (дата звернення: 11.02.2025).

12. Starter kits – laravel 12.x – the PHP framework for web artisans. Laravel – The PHP Framework For Web Artisans. URL: https://laravel.com/docs/12.x/starter-kits (дата звернення: 11.02.2025).

13. Upwork. URL: https://www.upwork.com/ (дата звернення: 11.02.2025).

14. What is mysql and how does it work. Hostinger Tutorials. URL: https://www.hostinger.com/tutorials/what-is-mysql (дата звернення: 10.02.2025).

15. What is pusher? HowDev. URL: https://how.dev/answers/what-is-pusher (дата звернення: 12.02.2025).

16. What is REST API. PHPenthusiast. Learn object oriented PHP. URL: https://phpenthusiast.com/blog/what-is-rest-api (дата звернення: 15.05.2025).

17. Міжсайтовий скриптинг. Вікіпедія. URL: https://uk.wikipedia.org/wiki /Міжсайтовий_скриптинг (дата звернення: 12.02.2025).

18. Що таке Bcrypt. Терміни та визначення кібербезпеки. VPN Unlimited – Fast & Secure VPN service. URL: https://www.vpnunlimited.com /ua/help/cybersecurity/bcrypt (дата звернення: 12.02.2025).

19. Як реалізувати CSRF-захист. Symfony. URL: https://symfony.com.ua/doc/current/security/csrf.html (дата звернення: 12.02.2025).

ДОДАТКИ

Додаток А – Апробація

Міністерство освіти і науки України
Волинська обласна рада
Луцька міська рада
Луцький національний технічний університет
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Національний університет «Львівська політехніка»
Військовий інститут телекомунікацій та інформатизації імені Героїв Крут (м. Київ)
Тернопільський національний педагогічний університет імені В. Гнатюка
Рівненський державний гуманітарний університет
Навчально-науковий інститут «Українська інженерно-педагогічна академія» Харківського національного університету ім. В.Н. Каразіна
Люблінська політехніка (Польща)
Фрайберзька гірнича академія (Німеччина)
Гронінгенський університет (Нідерланди)
Кавказький університет (Грузія)
Політехнічний університет Браганси (Португалія)

ITОНВ

ТЕЗИ ДОПОВІДЕЙ
X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)

23-24 травня 2025 р.

Луцьк 2025

УДК 004.4

ОСОБЛИВОСТІ РОЗРОБКИ ВЕБ ЗАСТОСУНКІВ З ВИКОРИСТАННЯМ LARAVEL ТА REACT

Бодяк Віталій Олегович
Луцький національний технічний університет, студент групи ІПЗ-41
Ліщина Наталія Миколаївна
Луцький національний технічний університет, к.т.н., доцент, завідувач кафедри інженерії програмного забезпечення, n.lishchyna@lutsk-ntu.com.ua

FEATURES OF WEB APPLICATION DEVELOPMENT USING LARAVEL AND REACT

Vitalii Bodiak
Lutsk National Technical University, Student of the Group IPZ-41
Nataliia Lishchyna
Lutsk National Technical University, Ph.D., Associate Professor, Head of the Department of Software Engineering, n.lishchyna@lutsk-ntu.com.ua

The paper explores web application development using Laravel and React. It highlights backend API integration, component-based UI, and deployment specifics. The combination ensures flexibility, scalability, and effective client-server communication.

Keywords: *Laravel, React, web application, API, frontend, backend.*

Щодня потреби людства стрімко зростають, змінюючи ритм життя, формуючи нові звички споживання інформації, товарів і послуг, а також стимулюючи розвиток технологій. У світі, де все більше процесів автоматизується, виникає постійна потреба в інструментах, здатних спростити виконання рутинних або повторюваних завдань. Саме тому на перший план виходить ідея оптимізації повсякденного життя та бізнес-процесів за допомогою цифрових рішень. Сьогодні практично кожна сфера діяльності, від малого підприємництва до великих корпорацій

повинна мати власний сайт. Це може бути сайт візитка, що репрезентує компанію в інтернеті, так і повноцінний веб застосунок, через який здійснюється основна частина бізнес-операцій по типу продажу, надання послуг, зворотний зв'язок з клієнтами та інші.

Розробка веб ресурсів дає змогу бізнесу масштабуватися, охоплювати ширшу аудиторію, автоматизувати внутрішні процеси, збирати й аналізувати дані про користувачів. За допомогою спеціально розроблених платформ можна не лише оптимізувати витрати, але й вивільнити людський ресурс, делегуючи прості й повторювані задачі цифровим системам.

React та Laravel – це сучасні інтсрументи для розробки веб застосунків. Вони надають широкий спектр можливостей в розробці починаючи від роботи закінчуючи безпекою. Laravel – це міжплатформовий PHP-фреймворк для створення веб-додатків, що має багато переваг у використанні, оскільки дозволяє розробнику скористатися можливостями великої бібліотеки попередньо запрограмованих функцій (таких як автентифікація, маршрутизація та створення шаблонів HTML) [1]. Він надає готові рішення для аутентифікації, роботи з базою даних через Eloquent ORM, обробки запитів, маршрутизації, черг, подій та багатьох інших аспектів розробки бекенду. Це дозволяє значно зекономити час розробника і зменшити кількість шаблонного коду. Laravel також підтримує шаблонізатор Blade, але при роботі з React фронтенд повністю відокремлюється, і Blade зазвичай не використовується [3]. React – відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту вебсторінки, з якими стикаються в розробці односторінкових застосунків [2]. Його компонентна структура, віртуальний DOM та підхід до керування роблять його гарним вибором для створення динамічних SPA додатків. React дозволяє створювати компоненти, що використовуються повторно, що значно полегшує підтримку та масштабування інтерфейсу.

Однією з ключових особливостей при поєднанні Laravel та React є необхідність розробки API. Laravel в цьому випадку виступає як RESTful API або GraphQL сервер, який обробляє запити з React-додатку та повертає відповіді у форматі JSON. Це забезпечує чітке розділення клієнтської та серверної логіки, що полегшує тестування, налагодження та повторне використання коду. Хоча в залежності від задачі і якщо немає потреби розробки API, то можна використовувати Inertia, яка виступає мостом між клієнтом і сервером за допомогою адаптерів.

Також важливо враховувати процес збірки та розгортання. React-застосунок зазвичай збирається за допомогою збірника Vite, після чого його можна розмістити у публічній директорії Laravel (наприклад, у public), або ж використовувати окремий сервер для фронтенду наприклад, через Nginx або Vercel у зв'язці з Laravel, що працює на окремому бекенд-сервері.

Ще однією особливістю є необхідність налаштування CORS (Cross-Origin Resource Sharing), коли фронтенд і бекенд працюють на різних доменах або портах. Laravel надає middleware для обробки CORS, який забезпечує безпечну взаємодію між клієнтом і сервером.

Також при розробці хорошою практикою є явна типізація коду. При роботі з React можна використовувати мову програмування TypeScript яка є обгорткою над JavaScript, але при цьому надає можливість строго типізувати код. PHP у свою чергу з версії 7.4 отримав можливість можливість явно оголошувати типи, а у зв'язці з phpstan можна ефективно типізувати серверну частину застосунку. Це надасть можливість зробити код чіткішим і уникнути плутанини в типах даних.

Створення веб-сайту – це трудомісткий та багатогранний процес, який вимагає глибокого занурення як у технічні тонкощі, так і в потреби кінцевого споживача. Цей процес включає в себе цілий спектр етапів, починаючи від планування та дизайну, і закінчуючи розробкою, тестуванням та подальшим супроводом. Шляхом комплексного підходу до розробки, веб-

сайт здатний перетворитися на ефективний інструмент для бізнесу чи особистих проектів.

Список використаних джерел

1. Inertia. *Inertia.js*. URL: https://inertiajs.com/ (дата звернення: 10.05.2025).

2. React. *React*. URL: https://react.dev/ (дата звернення: 10.05.2025).

3. Laravel. *Laravel - The PHP Framework For Web Artisans*. URL: https://laravel.com/ (дата звернення: 10.05.2025).

Додаток Б – Код контролера реєстрації

```
<?php

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;
use App\Models\AboutUser;
use App\Models\User;
use Illuminate\Auth\Events\Registered;
use Illuminate\Http\RedirectResponse;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;
use Illuminate\Validation\Rules;
use Inertia\Inertia;
use Inertia\Response;

class RegisteredUserController extends Controller
{
    /**
     * Display the registration view.
     */
    public function create(): Response
    {
        return Inertia::render('Auth/Register');
    }

    /**
     * Handle an incoming registration request.
     *
     * @throws \Illuminate\Validation\ValidationException
     */
    public function store(Request $request): RedirectResponse
    {
        $request->validate([
            'name' => 'required|string|max:255',
            'email' =>
'required|string|lowercase|email|max:255|unique:'.User::class,
            'role' => 'required',
            'password' => ['required', 'confirmed',
Rules\Password::defaults()],
            'avatar' => 'required|image|mimes:jpeg,png,jpg'
        ]);

        if($request->hasFile('avatar')) {
            $avatar = $request->file('avatar')->store(options: 'public');
        }

        $user = User::create([
            'name' => $request->name,
```

```
            'email' => $request->email,
            'role' => $request->role,
            'password' => Hash::make($request->password),
            'avatar' => $avatar ??= null
        ]);

        AboutUser::createTable($user->id);

        event(new Registered($user));

        Auth::login($user);

        return redirect(route('dashboard', absolute: false));
    }
}
```

кінець додатку Б

Додаток В – Код компоненту UserCard

```
import React, { useState } from "react";

export default function UserCard({ user }) {
  const [showAll, setShowAll] = useState(false);

  const skills = user.about_user?.skills?.split(", ") || [];
  const visibleSkills = showAll ? skills : skills.slice(0, 3);

  let role =
    user.role === "freelancer"
      ? "Фрілансер"
      : user.role === "employer"
      ? "Роботодавець"
      : "Адмін";

  return (
    <div
      key={user.id}
      className="w-full bg-white shadow-md rounded-2xl p-6 mb-6"
    >
      <div className="flex justify-between items-center">
        <div className="flex items-center space-x-4">
          <img
            src={
              user.avatar
                ? `/storage/${user.avatar}`
                : "/images/default-avatar.png"
            }
            alt={user.name}
            className="w-16 h-16 rounded-full border object-cover"
          />
          <h3 className="text-lg font-bold text-gray-900">
            {user.name} | <span>{role}</span>
          </h3>
        </div>
        <div className="flex gap-1">
          <a
            href={`/user-profile/${user.id}`}
            className="border border-blue-500 text-blue-500 hover:bg-
blue-500 hover:text-white focus:bg-blue-500 focus:text-white font-
semibold py-2 px-4 rounded-lg"
          >
            Профіль
          </a>
          <a
            href={`#`}
            title={‘Репорт’}
```

```
            className="border border-red-500 text-red-500 hover:bg-red-
500 hover:text-white focus:bg-red-500 focus:text-white font-semibold py-2
px-4 rounded-lg"
          >
            !
          </a>
        </div>
      </div>
      <h1 className="mt-3 font-semibold text-gray-800">Про мене:</h1>
      <p className="mt-2 text-gray-600 whitespace-pre-line">
        {user.about_user?.description}
      </p>
      <div className="mt-4">
        <h4 className="text-sm font-semibold text-gray-800">Навички:</h4>
        <ul className="list-disc list-inside text-gray-600 space-y-1">
          {visibleSkills.map((skill, index) => (
            <li key={index}>{skill}</li>
          ))}
        </ul>
        {skills.length > 3 && (
          <button
            onClick={() => setShowAll(!showAll)}
            className="mt-2 text-blue-500 text-sm hover:underline
focus:outline-none"
          >
            {showAll ? "Показати менше" : "Показати всі"}
          </button>
        )}
      </div>
      <div className="mt-4 text-sm text-gray-700">
        <strong>Погодинна ставка:</strong>
${user.about_user?.hourly_rate}/год
      </div>
      <div className="mt-6">
        <a
          href={`/chatify/${user.id}`}
          className="block text-center bg-blue-500 text-white py-2 px-4
rounded-lg hover:bg-blue-600 transition"
        >
          Написати мені в повідомлення
        </a>
      </div>
    </div>
  );
}
```

кінець додатку В