МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії
програмного забезпечення

_____ Євген ДАВИДЕНКО

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

«Вебзастосунок взаємодії фрілансерів та замовників»

Спеціальність 121 Інженерія програмного забезпечення
Освітня програма «Інженерія програмного забезпечення»

Здобувач _____ **Євген БАРАНОВСЬКИЙ**

«__» _____ 20__ р.

Керівник роботи
PhD, доцентка

_____ **Катерина АНТІПОВА**

«__» _____ 20__ р.

Миколаїв – 2026

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп'ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступінь	Бакалавр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___» _____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну бакалаврську роботу здобувача

Барановського Євгена

1. Тема кваліфікаційної роботи «Вебзастосунок взаємодії фрілансерів та замовників» затверджена наказом ректора ЧНУ ім. Петра Могили № 349 від «26» грудня 2025 р.
2. Строк представлення кваліфікаційної роботи «___» _____ 2026 р.
3. Очікуваний результат роботи є створений вебзастосунок для взаємодії фрілансерів та замовників – фріланс-марктеплейс.
4. Перелік питань, що підлягають розробці:

– провести аналіз предметної області та існуючих аналогів фріланс-маркетплейсів;

– визначити функціональні та нефункціональні вимоги до системи;

– обґрунтувати вибір архітектурного підходу та технологічного стеку розробки;

– спроєктувати архітектуру вебзастосунку та структуру бази даних;

– реалізувати механізми реєстрації та автентифікації користувачів;

– розробити функціонал створення, публікації та пошуку проєктів;

– реалізувати систему подання заявок та управління замовленнями;

– забезпечити внутрішню систему повідомлень для комунікації між користувачами;

– провести тестування розробленої системи та оцінити результати її роботи.

5. Презентація.

6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання «27» грудня 2026 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: **Вебзастосунок взаємодії фрілансерів та замовників**

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Затвердження теми КРБ	26.12.2025	26.12.2025	
2.	Розробка та затвердження завдання на виконання КБР	27.12.2025	27.12.2025	
3.	Огляд літератури за темою роботи	02.01.2026	03.02.2026	
4.	Складання календарного плану КБР	04.02.2026	04.02.2026	
5.	Аналіз предметної області	05.02.2026	06.02.2026	
6.	Розробка проєктних рішень	09.02.2026	13.02.2026	
7.	Моделювання та конструювання ПЗ	16.02.2026	20.02.2026	
8.	Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	23.02.2026	06.03.2026	
9.	Відгук керівника КБР	07.03.2026	07.03.2026	
10.	Оформлення КБР та презентації	10.03.2026	10.03.2026	
11.	Попередній захист	27.05.2026	27.05.2026	
12.	Завершення оформлення КБР та презентації			
13.	Рецензування			
14.	Захист кваліфікаційної роботи			

Здобувач _____ **Євген БАРАНОВСЬКИЙ**

«__» _____ 20_ р.

Керівник роботи

PhD, доцентка

_____ **Катерина АНТІПОВА**

«__» _____ 20_ р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

«Вебзастосунок взаємодії фрілансерів та замовників»

Здобувач 409 гр.: Барановський Євген

Керівник: PhD, доцентка Антіпова Катерина

Суть поточної роботи полягає в розробці вебзастосунку для зручної та ефективної взаємодії між фрілансерами та замовниками. Стрімке зростання попиту на дистанційну роботу та цифрові платформи, які спрощують процес пошуку проєктів для виконавців і кандидатів на виконання, визначають актуальність розробки даного проєкту.

Метою роботи є проєктування та розробка web-орієнтованої інформаційної системи – фріланс-маркетплейсу, призначеного для автоматизації процесів взаємодії між замовниками та фрілансерами, управління проєктами, комунікації та контролю виконання робіт.

Об’єктом роботи є процеси інформаційного забезпечення та організації взаємодії між замовниками та фрілансерами у web-орієнтованих інформаційних системах.

Предметом роботи є методи, моделі та програмні засоби проєктування і реалізації фріланс-маркетплейсу для автоматизації процесів управління проєктами, комунікації та контролю виконання робіт.

Застосовані практичні методи сприяли реалізації архітектуру веб-додатка з залученням таких технологій, як Laravel, системи управління базами даних PostgreSQL та фреймворку Bootstrap для створення адаптивного користувацького інтерфейсу. Реалізовано систему авторизації користувачів, збереження та обробки даних, обмін повідомленнями (чат), коментування, систему рейтингу та пошук проєктів.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків і списку використаних джерел. У першому розділі подано аналіз предметної області та огляд існуючих фріланс-маркетплейсів. Другий розділ присвячений змодельованій системі та дослідженню сучасних технологій, а також опису

сформованих функціональних та нефункціональних вимог. У третьому розділі розглянуто проєктування програмного забезпечення, що охоплює вибір інструментарію, моделювання графічного інтерфейсу та функціональних можливостей через UML-діаграми. Четвертий розділ описує програмну реалізацію, яка включає налаштування середовища, імплементацію бізнес-логіки, розробку інтерфейсу та приклади коду. Висновки – підсумок проведеного дослідження, де обґрунтовуються результати досягнення мети роботи та визначаються перспективи подальшого розвитку проєкту.

Кваліфікаційна робота викладена на 85 сторінках машинописного тексту, складається із вступу, 4 розділів, загальних висновків, переліку джерел посилання з 15 найменувань та 2 додатків. Праця містить 17 таблиць та 46 рисунків.

Ключові слова: PHP, Laravel, PostgreSQL, фріланс-маркетплейс.

ABSTRACT

to the qualifying bachelor's thesis

«Web application for interaction between freelancers and customers»

Student of 409 group: Baranovskyi Yevhen

Supervisor: Ph.D., Associate Professor Antipova Katerina

This work is dedicated to the development of a web application for convenient and effective interaction between freelancers and customers. The relevance of this development is due to the rapid growth of remote employment and the demand for digital platforms that simplify the processes of searching for projects, contractors, and organizing cooperation.

The goal of the project is to design and develop a web-based information system – a freelance marketplace–intended to automate the processes of interaction between customers and freelancers, project management, communication, and work performance control.

The object of the work is the processes of information support and organization of interaction between customers and freelancers in web-oriented information systems.

The subject of the work is methods, models, and software tools for designing and implementing a freelance marketplace for automating project management, communication, and work control processes.

Practical methods allowed us to implement the web application architecture using the Laravel framework, the PostgreSQL database management system, and the Bootstrap framework to create a responsive user interface. User authorization, data storage and processing mechanisms were implemented, as well as functional modules for messaging (chat), commenting, rating, and searching.

The thesis consists of an introduction, four chapters, conclusions, and a list of references. The first chapter provides an analysis of the subject area and an overview of existing freelance marketplaces. The second chapter is devoted to the modeled system and research of modern technologies, as well as a description of the functional and non-functional requirements. The third chapter discusses software design, covering the selection of tools, modeling of the graphical interface, and functional capabilities through

UML diagrams. The fourth section describes the software implementation, which includes environment configuration, business logic implementation, multilingual interface development, and code examples. The conclusions summarize the research results and confirm the achievement of the goal and outline prospects for further development of the work.

The thesis is presented on 85 pages of typed text and consists of an introduction, 4 chapters, general conclusions, a list of references with 15 titles, and 2 appendices. The work contains 17 tables and 46 figures.

Keywords: PHP, Laravel, PostgreSQL, freelance marketplace.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД	База даних
СКБД	Система керування базами даних
API	Application Programming Interface
CPU	Central Processing Unit (центральний процесор)
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets (каскадні таблиці стилів)
HTTPS	HyperText Transfer Protocol Secure
JS	JavaScript
MVC	Model-View-Controller (модель–вид–контролер)
MVP	Minimum Viable Product (мінімально життєздатний продукт)
NoSQL	Not Only SQL (нереляційні бази даних)
ORM	Object-Relational Mapping (об’єктно-реляційне відображення)
PHP	Hypertext Preprocessor (PHP)
RAM	Random Access Memory (оперативна пам’ять)
SQL	Structured Query Language (мова структурованих запитів)
UI	User Interface (користувацький інтерфейс)

ВСТУП

Цифрові технології проникають та розвиваються дуже швидко і використовуються суспільством майже у кожній сфері. Область віддаленої роботи є одною з ключових та залежною від якості взаємодії між замовниками та виконавцями, від прозорості процесу взаємодії, від співпраці та від зручності пошуку відповідних проєктів.

В даний момент клієнти задоволенні в неповному обсязі поточним форматом взаємодії між учасниками процесу організації віддаленої роботи, тому що недостатня гнучкість та зручність. Розробка системи, яка буде слугувати надійним способом взаємодії між виконавцями та замовниками є суттю для поточної роботи.

Попит на поліпшення користувацького досвіду та ефективності взаємодії походить від стану рівня конкуренції на ринку віддаленої праці. Вирішення цієї проблеми у практичному сенсі задовільнить ринковий попит.

Результат роботи пропонує рішення, що надає користувачам можливість створювати, публікувати та переглядати проєкти, шукати замовників або виконавців, спілкуватись в реальному часі, коментувати та оцінювати результати робіт. Прозорість та зручність процесу комунікації у рамках платформи є ключовою ціллю для розробки.

Метою роботи є проєктування та розробка web-орієнтованої інформаційної системи – фріланс-маркетплейсу, призначеного для автоматизації процесів взаємодії між замовниками та фрілансерами, управління проєктами, комунікації та контролю виконання робіт.

Відповідно до мети визначено такі завдання:

– провести аналіз предметної області та існуючих аналогів фріланс-маркетплейсів;

– визначити функціональні та нефункціональні вимоги до системи;

– обґрунтувати вибір архітектурного підходу та технологічного стеку розробки;

– спроєктувати архітектуру вебзастосунку та структуру бази даних;

– реалізувати механізми реєстрації та автентифікації користувачів;

– розробити функціонал створення, публікації та пошуку проєктів;

– реалізувати систему подання заявок та управління замовленнями;

– забезпечити внутрішню систему повідомлень для комунікації між користувачами;

– провести тестування розробленої системи та оцінити результати її роботи.

Об’єктом роботи є процеси інформаційного забезпечення та організації взаємодії між замовниками та фрілансерами у web-орієнтованих інформаційних системах.

Предметом роботи є методи, моделі та програмні засоби проєктування і реалізації фріланс-маркетплейсу для автоматизації процесів управління проєктами, комунікації та контролю виконання робіт.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ Й ІСНУЮЧИХ РІШЕНЬ

1.1 Опис предметної області

В поточних умовах економіки фріланс стає все більш актуальним. Переваги роботи віддалено полягають в тому що можна швидко знайти працівника і не зважати на складність проєкту та географічні дані.

Самий різний спектр задач може бути вирішеним у дистанційному форматі, наприклад, у сфері маркетингу, дизайну, копірайтингу, аналітики, програмування тощо.

Під час пошуку працівника для виконання певних робіт необхідно мати доступ до інформації про його порядність, професійні навички та мати можливість контролювати процес виконання роботи.

Пошук виконавців за допомогою загальноприйнятих методів (соціальні мережі, тематичні форуми або особисті контакти) вже не спроможні надати достатній рівень зручності та безпеки. Через це виникає потреба у наявності платформи взаємодії між учасниками співпраці в вебсередовищі.

Фріланс-маркетплейс виступає як багатофункціональна інформаційна система, що забезпечує:

- реєстрацію користувачів з різними ролями;
- створення та публікацію проєктів;
- пошук замовлень;
- подання заявок на виконання робіт;
- внутрішню комунікацію між учасниками;
- формування рейтингової системи на основі відгуків та оцінок.

Вище вказані функції реалізуються задля покращення довіри між сторонами, зниження складності вибору виконавця або замовника.

Фільтрація проєктів відбувається за категорією, рейтингом, терміном виконання тощо, а фільтрація спеціалістів відбувається за досвідом та рейтингом. Отже – забезпечення зручного інтерфейсу, пошуку та фільтрація проєктів, а також спеціалістів за вказаними критеріями є важливою складовою предметної області.

Захист персональних даних користувачів, обробка коментарів та повідомлень, відстеження статусу проєкту, засоби збереження історії замовлень – є не менш значущими. Сучасна цифрова платформа орієнтована на безпечну та комфортну взаємодію учасників ринку праці. У сукупності всі функції формують основу для її побудови.

Рейтингова система існує для отримання інформації про якість роботи, відповідальності, надійності користувача та є інструментом вибору виконавця для замовника. Можливість підтвердити професійний рівень та підвищити конкурентоспроможність на платформі – є інструментом для фрілансера. З метою підвищення ефективності співпраці існують методи безпечної взаємодії – відбувається контроль виконання робіт.

Вебзастосунок забезпечує автоматизацію основних процесів для взаємодії між замовниками та виконавцями. Також, підвищує прозорість співпраці та покращує користувацький досвід.

Розроблена система виступає інструментом організації, оптимізації, взаємодії між виконавцями та замовниками.

1.2 Огляд застосунків-аналогів

Існує достатня кількість вебзастосунків для реалізації взаємодії між фрілансерами та замовниками. Для складання специфікації вимог розроблюваного ПЗ треба розглянути та проаналізувати існуючі альтернативи та виділити їх переваги, недоліки. З усього різноманіття можливих варіантів було обрано: freelance.ua (вітчизняний фріланс-маркетплейс), freelancer.com та fiverr.com.

Freelance.ua [1]

Freelance.ua – це вебплатформа, яка призначена для організації взаємодії між замовниками та фрілансерами у сфері дистанційної роботи (рис. 1.1). Сервіс надає можливість користувачам розміщувати проєкти, знаходити виконавців або замовлення, а також здійснювати пошук за різними категоріями послуг.

Freelance.ua включає до основного функціоналу створення та публікацію завдань, подання заявок на їх виконання, систему оцінювання та відгуків. Подібні

функції надають можливість формувати репутацію користувачів. Також, на цьому сайті присутнє упорядкування проєктів та виконавців за наступними критеріями: спеціалізація, бюджет і рейтинг.

Крім того, платформа пропонує додаткові платні можливості для підвищення видимості проєктів або профілів виконавців, що сприяє більш ефективному пошуку партнерів для співпраці. Завдяки цьому Freelance.ua виступає як інструмент, що допомагає користувачам швидше приймати рішення та організовувати робочий процес у середовищі дистанційної зайнятості.

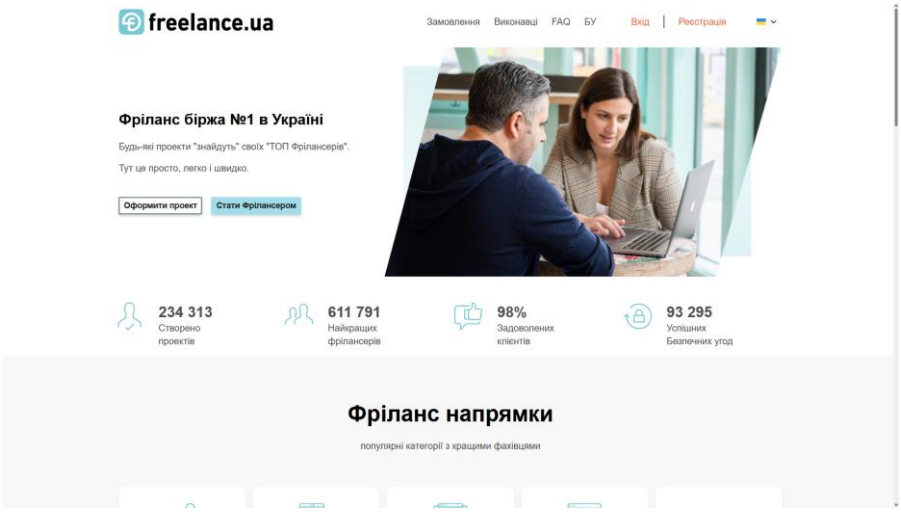


Рисунок 1.1 – Інтерфейс головної сторінки freelance.ua

Таблиця 1.1 – Характеристика «freelance.ua»

Засновник	ICAK ГмбХ (Німеччина)
Архітектура	Хмарна, клієнт-серверна (Nginx 1.10.3)
Мова реалізації	PHP, CSS, JS
Основні функції, характеристики	1) реєстрація в різних ролях; 2) створення замовлень; 3) виконання замовлень; 4) рейтинг виконавців та замовників; 5) безпечна угода.
Переваги	1) доступний та зручний інтерфейс на ключових сторінках;

Кінець таблиці 1.1

	2) зрозуміла та проста інструкція взаємодії з сервісом на головній сторінці; 3) проста і зрозуміла система оцінки підрядників і клієнтів; різноманітні платні послуги, які дозволяють збільшити охоплення аудиторії, що переглядає ваше замовлення (Виділення яскравим кольором тексту, закріплення пропозиції нагорі загальної стрічки пропозицій).
Недоліки	1) наявність сторінок, які частково не мають оформлення каскадними таблицями стилів (https://freelance.ua/price.html,); 2) не всі сторінки локалізовані українською (https://freelance.ua/help/); 3) платний перегляд замовлень – деякі замовлення є платними для перегляду; 4) відсутня можливість сортувати виконавців за рейтингом на сторінці з виконавцями.

Freelancer.com [2]

Freelancer.com – це міжнародна онлайн-платформа, яка об’єднує роботодавців і фахівців, що працюють на умовах віддаленої співпраці. Сервіс створений для того, щоб замовники могли швидко опублікувати свій проєкт, а виконавці – знайти підходящу роботу відповідно до власних навичок і досвіду.

На платформі «Freelancer.com» можна знайти проєкти, які стосуються програмування, дизайну та копірайтингу до маркетингу, перекладу й технічної підтримки. Користувачам надана можливість перегляду актуальних завдань, подання заявок на їх виконання, ведення комунікації всередині системи та вибір найприйнятніших умов співпраці. Окремо, на сайті «Freelancer.com» наявні

системи рейтингу, відгуків і перевірки профілів, які можна використати для оцінки надійності учасників співпраці.

Серед інструментарію сервісу варто виокремити систему пошуку та фільтрації, які дають змогу зручно знайти проєкт або спеціаліста за категорією, ціною, термінами виконання та іншими параметрами. Присутні додаткові платні послуги, що дозволяють покращати видимість проєктів серед інших. Аналогічно виконавці мають змогу просовувати у рейтингу свої профілі. В цілому цей сайт можна представити як зручне середовище для організації дистанційної роботи та налагодження професійної співпраці.

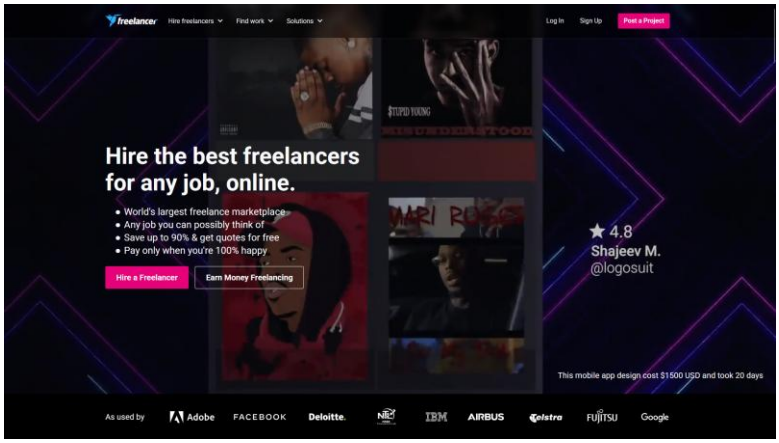


Рисунок 1.2 – Інтерфейс головної сторінки freelancer.com

Таблиця 1.2 – Характеристика «freelancer.com»

Засновник	Метт Баррі (Matt Barrie)
Архітектура	Хмарна, клієнт-серверна (Google cloud)
Мова реалізації	PHP, CSS, JS
Основні функції, характеристики	1) реєстрація в різних ролях; 2) створення замовлень; 3) виконання замовлень; 4) рейтинг виконавців та замовників; 5) безпечна угода, допомога ШІ під час створення замовлень та публікації відгуків.

Кінець таблиці 1.2

Переваги	1) зрозуміла та проста презентація взаємодії з сервісом на головній сторінці; 2) проста і зрозуміла система оцінки підрядників і клієнтів; 3) допомога ШІ під час створення замовлень та публікації відгуків; 4) можливість обрати замовників за мовою та за локацією.
Недоліки	1) можливо залишити лише шість безкоштовних відгуків на місяць виконавцям; 2) висока комісія (10% від замовлення).

Fiverr.com [3]

Ще один сервіс на який варто звернути увагу це «Fiverr». Його призначення полягає в надані можливості взаємодії між замовниками та фахівцями. Основні завдання на які він спрямований – виконання цифрових послуг у форматі короткочасних завдань. Модель взаємодії – «послуга за запитом» (Пропозиції називаються гігами).

До спектру категорій проєктів які можна знайти на «Fiverr» відносяться: графічний дизайн, розробка ПЗ, написання текстів, відеомонтаж, маркетинг, тощо. Кожен користувач має змогу переглядати портфоліо виконавців, аналізувати відгуки, рівень рейтингу та інші умови для співпраці. Всередині системи наявний механізм для обміну повідомлень завдяки якому легко та просто узгоджувати та контролювати виконання проєктів.

До головної особливості «Fiverr» можна віднести чітку структуру пакетів послуг, вони мають фіксовану вартість та термін виконання. Завдяки цьому замовники спроможні швидко приймати правильні рішення без довготривалих переговорів. Подібно як у «freelancer.com» є послуги просування та покращення видимості профілю серед інших користувачів. В загалом «Fiverr» є потужним та ефективним інструментом для доступу до замовлень цифрових послуг і розвитку професійної діяльності у сфері фрілансу.

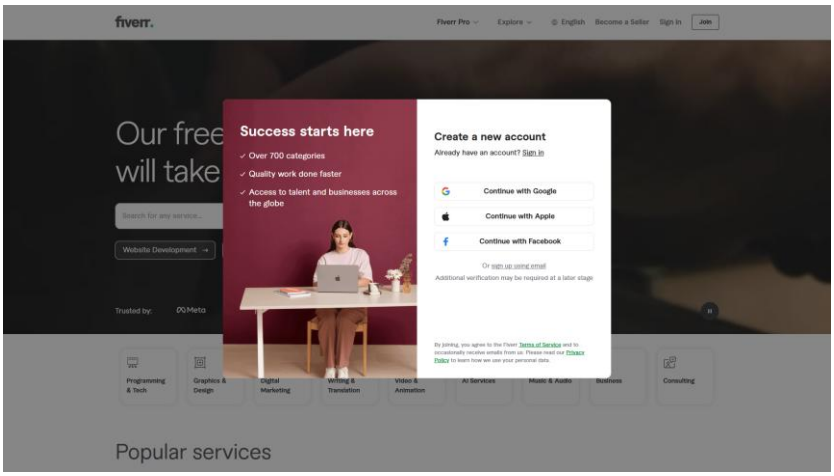


Рисунок 1.3 – Інтерфейс головної сторінки fiverr.com

Таблиця 1.3 – Характеристика «fiverr.com»

Засновник	Міха Кауфман (Micha Kaufman) та Шай Вінінгер (Shai Wininger)
Архітектура	Хмарна, клієнт-серверна (Google cloud, Amazon Web Services)
Мова реалізації	PHP, CSS, JS
Основні функції, характеристики	1) реєстрація користувачів з різними ролями; 2) створення та виконання замовлень; 3) обмін повідомленнями та файлами між користувачами система рейтингів і відгуків; 4) механізм безпечної угоди підтримка штучного інтелекту під час створення замовлень і відгуків.
Переваги	1) Використання ШІ для допомоги у створенні замовлень і відгуків; 2) гнучкі фільтри пошуку (мова, локація, рейтинг).
Недоліки	1) Складна бізнес-логіка пошуку та сортування результатів; 2) висока комісія; 3) громіздкий дизайн та інтерфейс – складно орієнтуватися.

Проведено дослідження та порівняльний аналіз існуючих аналогів. Наступний крок – поглиблене опрацювання ПЗ та його розробка. Визначенні переваги та недоліки існуючих сервісів, які можна використати як основу для побудови майбутньої системи та формування подальших функціональних і нефункціональних вимог.

1.3 Аналіз застосунку, що розробляється

Веб-застосунок призначений для ефективної взаємодії між фрілансерами та замовниками. У користувачів є можливість переглядати нові проєкти, здійснювати пошук замовників та виконавців відповідно до параметрів. Також є можливість ознайомлення з рейтингом та відгуками, створення, публікація власних пропозицій, заявок на виконання робіт.

У системі передбачено декілька ролей користувачів:

– замовник (customer) – це користувач, який створює проєкти, публікує завдання та обирає виконавців для їх реалізації. Він може переглядати профілі фрілансерів, аналізувати їхні відгуки та рейтинги, а також взаємодіяти з ними в межах платформи;

– виконавець (executor) – це фахівець, який шукає завдання, подає заявки на виконання проєктів, виконує роботу та отримує оплату. Виконавець формує власний профіль, портфоліо та репутацію через систему відгуків;

– адміністратор (admin) – це користувач, який здійснює модерацію платформи, контролює дотримання правил, управляє користувачами, проєктами та відгуками, а також вирішує спірні ситуації;

– сервісна роль (service) – технічна роль, яка використовується для забезпечення внутрішніх процесів системи (автоматичні перевірки, обробка платежів, сповіщення тощо).

Основні функції системи:

– реєстрація нових користувачів;

– авторизація та автентифікація користувачів;

– створення, перегляд і редагування профілю;

– публікація проєктів замовником;
– перегляд доступних проєктів виконавцями;
– подання заявок (пропозицій) на виконання завдань;
– вибір виконавця замовником;
– обмін повідомленнями між користувачами;
– управління процесом виконання проєкту;
– пошук проєктів і виконавців;
– фільтрація за категоріями, бюджетом та іншими параметрами;
– система рейтингів і відгуків;
– формування та перегляд історії співпраці;
– здійснення оплат і контроль транзакцій;
– сповіщення користувачів про події в системі;
– модерація контенту та користувачів адміністратором;
– редагування або видалення проєктів, профілів і відгуків відповідно до прав доступу.

Інтерфейс вебзастосунку повинен бути спроєктований, як зручний, зрозумілий у використанні, щоб не перевантажувати користувача. Від якості інтерфейсу залежить загальне враження користувача від роботи з системою.

Повний цикл взаємодії із системою включає в себе сторінки від реєстрації до завершення співпраці на проєктом. Для створення облікового запису, визначення ролі – замовника або виконавця існує сторінка реєстрації, для входу в систему існує сторінка авторизації.

На головній сторінці відображається перелік актуальних проєктів до якої користувач потрапляє після авторизації. Інструменти пошуку та фільтрації проєктів для швидкого вибору за категорією, бюджетом, термінами виконання допомагають знаходити релевантні пропозиції.

Наступна сторінка, яка заслуговує уваги – сторінка профілю користувача. Завдяки неї можна редагувати особисті дані, налаштовувати обліковий запис, переглядати рейтинг та історію активності.

Для замовників присутня можливість керувати створеними проєктами, переглядати історію замовлень.

В розробленій платформі існують функції створення та управління проєктами. Замовники мають можливість опублікувати нові завдання, надавати їх детальний опис, визначати строки виконання, бюджет, вимоги до виконавців, а для уточнення завдання – додавати файли. Для фахівців доступні функції перегляду проєкту, подача пропозицій для виконання роботи.

Після створення проєкту виконавці надсилають пропозиції, які складаються з коментаря, запропонованого бюджету та терміну виконання. Замовник веде комунікацію з кандидатами та переглядає їх пропозиції, обирає виконавця. Після вибору кандидата на виконання проєкт переходить у статус виконання.

При виконані замовлення обидві сторони мають можливість обговорювати деталі проєкту, обмінюватися повідомленнями, надсилати файли. Всі питання в системі розв’язуються прозоро за допомогою скарг.

Після виконання замовлення кожна сторона має можливість оцінити отриманий досвід взаємодії з партнером. В результаті формується рейтинг користувача. Система репутації платформи допоможе користувачам в подальшому виборі партнерів.

Адміністратор модерує контент, забезпечує дотримання правил платформи, редагує та опрацьовує скарги.

Висновки до розділу 1

У першому розділі досліджено наявні пропозиції на ринку дистанційної праці та визначено основні проблеми та переваги серед найпопулярніших фріланс платформ. Сформульовані основні вимоги зручної, безпечної та ефективної вебплатформи для взаємодії між її користувачами. Проведено аналіз предметної області.

Порівняльний аналіз виконаний серед наступних платформ: «freelance.ua», «freelancer.com» та «fiverr.com». Кожен з перерахованих сервісів надає базові можливості для публікації та виконання замовлень на достатньо високому рівні.

Серед загальних недоліків можна виокремити: складність інтерфейсу, платні послуги для базових функцій, негнучкі фільтри пошуку. У подальшому визначені недоліки та переваги слугуватимуть основою для майбутніх напрямків розвитку власного вебзастосунку.

Окреслено та проаналізовано вимоги до системи, яка розроблюється. Визначено основні ролі користувачів, перелік функціональних можливостей, а також особливості інтерфейсу та адміністративної частини. На основі проведеного аналізу та його результатів планується спроєктувати архітектуру системи, змоделювати бізнес-процеси та сформувати функціональні та нефункціональні вимоги.

2 МОДЕЛЮВАННЯ ВЕБЗАСТОСУНКУ

2.1 Опис інструментарію

Мова програмування. Для реалізації сучасного вебзастосунку варто обрати популярну, надійну, з змістовною документацією та широким ком'юніті мову програмування. За вказаними критеріями найбільш релевантним варіантом є PHP. Ця мова проста, має довгу історію еволюціонування та не є застарілою, так як продовжує покращуватись та підтримуватись на поточний момент. PHP не цурається запозичувати особливості з інших популярних рішень. Наприклад, одна з останніх особливостей, що була запроваджена це сувора типізація подібно, як в C# та Java.

Варто вказати на ту можливість, яку надає PHP, а саме пряма інтеракція з JS, CSS та будь-яким API, що суттєво для невеликих та простих проєктів, які розроблюються. Подібна особливість значно полегшує процес розробки користувацького інтерфейсу [4]. А використання такої технологій, як Blade взагалі може позбавити необхідності в використанні фронтенд фреймворків, в особливості, якщо проєкт не масштабний та має стислі терміни на виконання. За допомогою Blade розділюється прикладна та візуальна логіка сторінки, що покращує сприйняття коду та полегшує його супровід.

Також, мова програмування PHP характерна своєю простотою взаємодії з базами даних. Існує безліч варіантів легкої та ефективної інтеграції з сучасними реляційними СУБД, в особливості з PostgreSQL, MySQL та Oracle. Окрім цього, PHP підтримує взаємодію з окремими NoSQL-рішеннями, такими як MongoDB.

Завдяки простоті інтеграції з базами даних розробка динамічних вебдодатків суттєво спрощується. БД використовуються для генерації контенту, обробки запитів та реалізації інтерактивної взаємодії з користувачем. Якщо, планується інтенсивне використання значних обсягів даних, то PHP є одним з кращих серед можливих варіантів.

Framework. Варто зазначити що використання виключно чистого PHP в умовах сьогодення не доцільне. Індустрія стрімко розвивається та надає множину

готових та відточених рішень або наборів інструментів. Зокрема, реалізація кожного нового проєкту має відповідати певній структурі. Щоб не вигадувати таку структуру кожного разу, а використати те що є вже загальноприйнятим (наприклад, архітектура MVC) існують фреймворки.

Фреймворки виступають як набори структурованих інструментів, що надають чітку архітектуру. Вони пропонують розподіл відповідальності та механізми, що вирішують типові завдання: авторизація, права доступу, робота з файловою системою, робота з БД тощо. Кожна така підсистема може бути реалізована за допомогою «чистого» PHP, але це трудомістко й не завжди призводить до кращих результатів ніж вже готові «відточенні» рішення. Таким чином, фреймворки сприяють підвищенню продуктивності розробки, якості коду та загальної надійності програмних систем.

Серед найбільш популярних PHP-фреймворків слід виокремити наступні: Laravel, Symfony, CodeIgniter, Yii, CakePHP, Laminas (наступник Zend Framework), Phalcon (високопродуктивний, реалізований як розширення C), FuelPHP.

Кожен PHP-фреймворк, який використовується для розробки вебзастосунків має свої особливості. Наприклад, один з найкращих – Laravel – вирізняється інтуїтивно зрозумілою структурою проєкту, високим рівнем абстракції та великою кількістю вбудованих інструментів: система автентифікації, система черг, міграції бази даних, ORM Eloquent.

Якщо, існує потреба в створенні складних та масштабованих систем, то варто розглянути фреймворк «Symfony». Один з найкращих варіантів для корпоративних клієнтів, але потребує додаткових зусиль для його освоєння.

Для проєктів які є невеликими і потребують швидкого старту існує легковаговий фреймворк «CodeIgniter», який простий в конфігурації, але має мінімальний функціонал у порівняні з іншими.

Якщо брати до уваги «Yii», то його можна відзначити завдяки генераторам коду та зручній реалізації CRUD-операцій, що корисно при швидкій розробці, наприклад, таких частин, як адміністративні панелі.

CakePHP відзначається тим що має зручні початкові автоматичні налаштування й не потребує від розробника додаткових зусиль при створені проєкту. Гарний варіант для створення швидких рішень.

Згідно інформації з Google Trends рівень попиту на Laravel має чітку перевагу над іншими рішеннями [5]. Останні п’ять років найвищий рівень інтересу саме у цього фреймворку (рис. 2.1). Завдяки активності спільноти збільшується кількість якісних навчальних матеріалів. Існує безліч якісних прикладів та статей. Поріг входження є значно нижчим через доступність інформації у порівняні з іншими рішеннями.

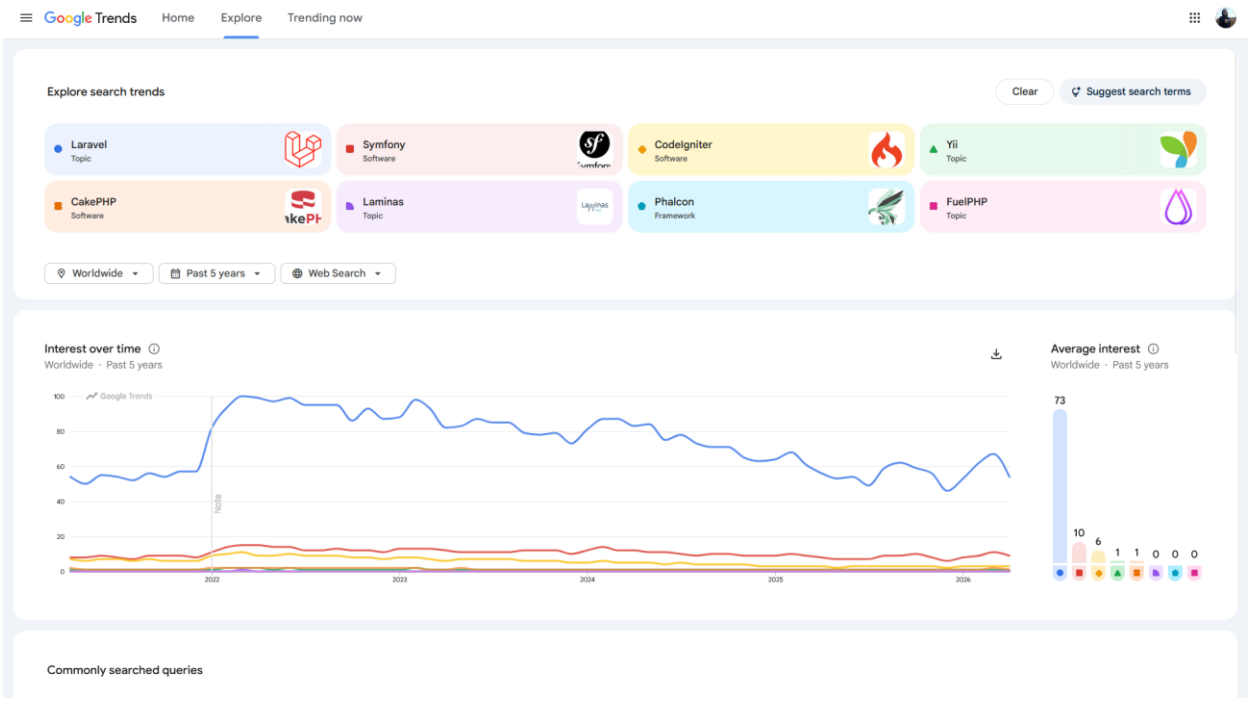


Рисунок 2.1 – Порівняння популярності фреймворків PHP

Якість офіційної документації не поступається тій інформації, що надходить від спільноти. Всі розділи чітко структуровані та змістовні. Завдяки цьому час на опанування технології нижчий. При виникненні типових проблем завжди варто почати їх вирішення саме з офіційної документації, що посприяє підвищенню якості коду та зменшить вірогідність виникнення помилок у майбутньому.

В межах поточного дипломного проєкту відповідно до вказаних вище даних та технічних характеристик (табл. 2.1), Laravel є доцільним вибором для розробки [6]. Швидкість розробки, наявність готових рішень для типових задач

(автентифікація, робота з базою даних тощо), масштабованість, а також активна підтримка спільноти – переваги, які дозволять ефективно реалізувати функціонал платформи та забезпечити її подальший розвиток.

Таблиця 2.1 – Порівняння PHP-фреймворків

Тест	**Laravel**	**Symfony**	**CodeIgniter**	**Yii**
Дослідження часу відгуку на запит сторінки в системі з низьким навантаженням	65 мс	70 мс	85 мс	90 мс
Середній час запису в базу даних	1.2 мс	1.4 мс	1.8 мс	1.6 мс
Швидкість виконання стандартного SQL SELECT-запиту	1 мс	1.1 мс	0.8 мс	1.3 мс
Кількість паралельних запитів, що обробляються за секунду	1200 запитів	1100 запитів	1000 запитів	1000 запитів
Використання пам’яті	10 мб/запит	14 мб/запит	12 мб/запит	16 мб/запит
Результати навантажувального тестування	3000 мб/запит	2600 мб/запит	2400 мб/запит	2200 мб/запит

На основі проведеного порівняльного аналізу можна зробити висновок, що Laravel є найпродуктивнішим PHP-фреймворком, з урахуванням усіх розглянутих показників.

СКБД. При розробці сучасного вебзастосунку неможливо знехтувати вибором СКБД. Правильний вибір цього елементу визначить продуктивність, надійність і масштабованість системи, оскільки вона відповідає за способи зберігання, обробки та аналізу інформації.

СКБД або СУБД класифікують переважно за моделлю організації даних: реляційна (SQL) та нереляційна (NoSQL).

Таблична структура організації даних і забезпечення підтримки складних зв’язків між сутностями за допомогою зовнішніх ключів, а також гарантування

цілісності і узгодженість даних завдяки транзакціям відноситься до реляційного типу.

Реляційний спосіб збереження даних притаманний для проєктів із чітко визначеною структурою і великою кількістю взаємопов’язаних об’єктів.

Доцільно використовувати нереляційну організацію даних, якщо вони не структуровані або слабко структуровані й мають великий обсяг. Подібні системи не мають підтримки складних зв’язків між сутностями, але серед переваг виділяють їх швидкість, гнучкість та наявність горизонтального способу масштабування [7].

До найпопулярніших рішень серед СКБД відносяться MySQL, Microsoft SQL Server, PostgreSQL та MariaDB. Серед нереляційних СКБД (NoSQL) найпоширеніші такі: MongoDB, Cassandra, Redis.

PostgreSQL є найбільш ефективними серед реляційних СКБД та MongoDB серед нереляційних СКБД [8].

PostgreSQL спроможний виконувати всі дії що властиві реляційним базам даних зі складними зв’язками між сутностями, транзакціями та виконанням складних аналітичних запитів. MongoDB не є доцільною для даних, що організовані реляційно, але у всіх інших випадках вона характеризується більшою гнучкістю та зручністю масштабування.

Сучасні наукові дослідження підтверджують переваги PostgreSQL у подібних сценаріях. Зокрема, у статті «Comparative Read Performance Analysis of PostgreSQL and MongoDB in E-Commerce» експериментально встановлено, що PostgreSQL виконує складні аналітичні запити у 1,6–15 разів швидше, а час виконання багатокритеріальних запитів є на 65–80% меншим порівняно з MongoDB [8].

Вебзастосунок взаємодії фрілансерів та замовників націлений на використання структурованих даних з множинною взаємозв’язків між сутностями (користувачі, проєкти, замовлення, відгуки та платежі). Функції пошуку, фільтрації проєктів і аналізу рейтингів потребують виконання складних запитів та забезпечення цілісності даних.

Отже, щоб задовольнити вказані вимоги найбільш раціональним вибором є реляційна модель баз даних.

Стилізації та адаптивний інтерфейс. Щоб відтворити користувацький інтерфейс у зручний спосіб раціонально використати фрейморк та шаблонізатор.

Серед найпопулярніших фреймворків для створення інтерфейсу виокремлюються Bootstrap та Tailwind, але перший варіант краще, тому що він дозволяє скоротити час створення прототипів приблизно на 25% завдяки використанню готових компонентів та стандартних шаблонів інтерфейсу [9]. Основна перевага Bootstrap полягає у наявності майже кожного типового UI-компоненту для вебзастосунків. Окрім цього, є адаптивна сітку, що дозволяє швидко створювати зручний інтерфейс для різних пристроїв.

Якщо порівнювати шаблонізатори, то серед всіх можна виділити: Blade, Twig та Smarty. Для Laravel-проєктів Blade є кращим вибором, адже він: вбудований, має простіший синтаксис, потребує менше шаблонного коду [10].

Поєднання Bootstrap та Blade робить процес розробки простішим та ефективнішим, так як мінімізує обсяг ручної верстки та пришвидшує створення UI.

2.2 Моделювання предметної області

Прийнято рішення схилитися до мінімалістичного та перевіреного технологічного стеку. Монолітна MVC архітектура передбачається як основа проєкту, що дозволяє інтегрувати серверну логіку, маршрутизацію, шаблонізацію та роботу з базою даних у межах єдиної структури застосунку [11, 12].

Blade-шаблони слугуватимуть, як рішення для відтворення клієнтської частини динамічного UI.

Адаптивність дизайну реалізується за допомогою фреймворку Bootstrap, який є одним з найпопулярніших для вирішення проблеми коректного відображення вмісту застосунку на різних пристроях.

Задля гарантії надійності, підтримки транзакцій та ефективній роботі із запитами використовується PostgreSQL.

Перераховані технології балансують між продуктивністю, простотою підтримки та можливістю подальшого масштабування та розвитку системи (табл. 2.2 та рис. 2.2).

Таблиця 2.2 – Стек технологій

Шар	Технологія	Обґрунтування вибору
Клієнтська частина (інтерфейс користувача)	Blade + Bootstrap	Blade забезпечує генерацію динамічних сторінок на стороні сервера та тісну інтеграцію з Laravel. Bootstrap використовується для швидкого створення адаптивного та зручного інтерфейсу, що коректно відображається на різних пристроях.
Серверна логіка та бізнес-процеси	Laravel	Laravel надає зручні засоби маршрутизації, автентифікації, роботи з формами, ORM Eloquent та вбудовані механізми безпеки. Це дозволяє швидко реалізувати функціонал фріланс-маркетплейсу та підтримувати структурованість коду.
СКБД	PostgreSQL	База даних забезпечує надійність зберігання інформації, підтримку транзакцій, ефективну обробку складних SQL-запитів та масштабованість, що важливо для роботи з користувачами, замовленнями та повідомленнями.

Схема взаємодії компонентів:

– користувач відкриває вебзастосунок у браузері, після чого сервер Laravel обробляє запит і формує HTML-сторінку за допомогою Blade-шаблонів;

– статичні ресурси інтерфейсу (CSS, JavaScript, зображення) завантажуються браузером, а Bootstrap забезпечує адаптивне відображення сторінок на різних пристроях;

– користувацькі дії (реєстрація, авторизація, створення замовлення, відгуки, повідомлення) надсилаються через HTTPS до серверної частини Laravel, де обробляються маршрути, контролери та бізнес-логіка системи;

– серверна частина звертається до PostgreSQL через ORM-прошарок Eloquent, виконує необхідні операції з даними та повертає сформовану відповідь;

– після обробки запиту користувач отримує оновлену сторінку або змінений інтерфейс безпосередньо у браузері.

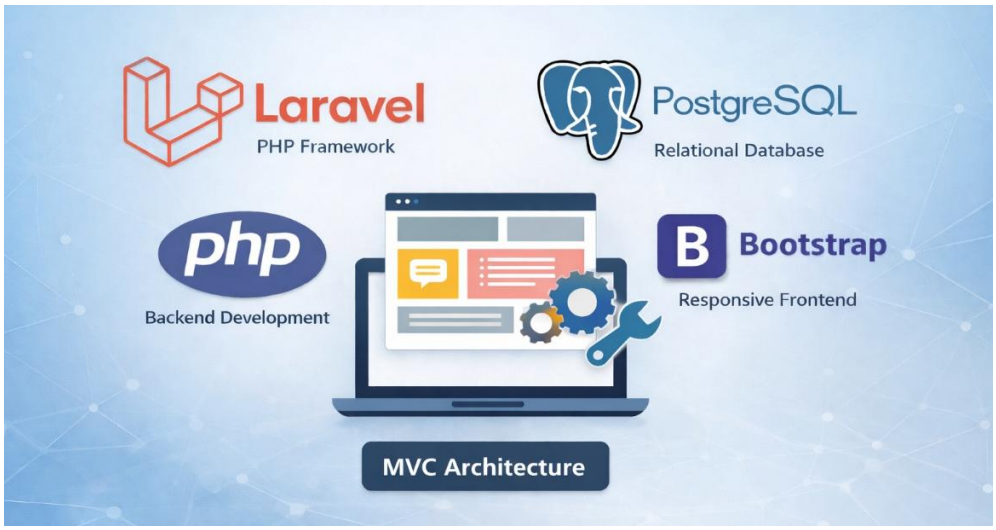


Рисунок 2.2 – Технології

Переваги обраного інструментарію:

– використання Laravel як основи монолітного MVC-застосунку забезпечує чітке розділення бізнес-логіки, маршрутизації, роботи з даними та шаблонізації, що спрощує розроблення й подальшу підтримку системи;

– застосування Blade-шаблонів дозволяє формувати динамічний інтерфейс без ускладнення архітектури, а також забезпечує тісну інтеграцію клієнтської частини з серверною логікою Laravel;

– використання Bootstrap сприяє швидкому створенню адаптивного та зручного інтерфейсу, який коректно відображається на різних пристроях, що є важливим для вебзастосунку з великою кількістю користувацьких сценаріїв;

– PostgreSQL гарантує надійне зберігання даних, підтримку транзакцій та ефективну обробку складних запитів, що особливо важливо для роботи з користувачами, замовленнями, повідомленнями та відгуками у фріланс-маркетплейсі.

2.3 Специфікації вимог для вебзастосунку що розробляється

1) Призначення та межі проєкту

1.1) Призначення системи

Забезпечення пошуку виконавців, розміщення проєктів, комунікації між сторонами та супровід виконання замовлень відносяться до основного призначення

вебзастосунку взаємодії фрілансерів та замовників. Система надає замовникам можливість створювати проєкти, а фрілансерам – подавати заявки, виконувати завдання та отримувати оплату.

1.2) Погодження, ухвалені в програмній документації

– система розробляється з використанням фреймворку Laravel;

– використовується СКБД PostgreSQL;

– передбачено клієнт-серверну архітектуру;

– реалізація базового функціоналу MVP включає реєстрацію, проєкти, відгуки та повідомлення;

– передбачено забезпечення базового рівня безпеки даних (шифрування паролів, захист від основних вебзагроз, таких як XSS та CSRF);

– інтерфейс має бути адаптивним та кросбраузерним.

1.3) Межі проєкту

– охоплює вебверсію платформи (без мобільного застосунку);

– включає серверну логіку;

– включає інтерфейс користувача;

– не включає платіжні шлюзи на початковому етапі (можлива подальша інтеграція);

– не охоплює зовнішні сервіси працевлаштування.

2) Загальний опис

2.1) Сфера застосування

Система використовується у сфері фріланс-ринку для взаємодії між замовниками та виконавцями в таких напрямах як IT, дизайн, маркетинг, копірайтинг та інші цифрові послуги.

2.2) Характеристики користувачів

– замовники – користувачі, які створюють проєкти та шукають виконавців;

– виконавці – залишають пропозиції та виконують замовлення;

– адміністратори – здійснюють модерацію замовлень та керування платформою;

– сервіси – виконують операції депонування коштів та обробки платежів;

– рівень підготовки користувачів – базовий або середній рівень роботи з вебсервісами.

2.3) Загальна структура і склад системи

– клієнтська частина (Blade та Bootstrap);

– серверна частина (Laravel);

– база даних PostgreSQL;

– модуль повідомлень (чат);

– модуль управління проєктами;

– система відгуків і рейтингу;

– адміністративна панель.

2.4) Загальні обмеження

– необхідне постійне підключення до Інтернету;

– продуктивність залежить від серверної інфраструктури;

– обмеження доступу реалізується через систему ролей;

– можливі затримки при великому навантаженні на систему;

– залежність від стабільності та доступності серверного обладнання.

3) Функції системи

3.1) Реєстрація та автентифікація користувачів

3.1.1) Опис функції

Система забезпечує створення облікових записів та вхід користувачів у систему з розподілом ролей (замовник, виконавець, адміністратор, системний користувач).

3.1.2) Вхідна і вихідна інформація

– вхідна: email, пароль, роль користувача;

– вихідна: токен сесії, профіль користувача.

3.1.3) Функціональні вимоги

– шифрування паролів;
– підтвердження email;
– відновлення пароля;
– авторизація через токени.

3.2) Створення та управління проєктами

3.2.1) Опис функції

Замовник створює проєкт із описом завдання, бюджету та термінів виконання.

3.2.2) Вхідна і вихідна інформація

– вхідна: назва, опис, бюджет, дедлайн;
– вихідна: ID проєкту, статус, список відгуків.

3.2.3) Функціональні вимоги

– редагування та видалення проєктів;
– публікація проєкту;
– фільтрація та пошук.

3.3) Подання пропозицій на проєкти

3.3.1) Опис функції

Фрілансер може подавати заявку на виконання проєкту.

3.3.2) Вхідна і вихідна інформація

– вхідна: ID проєкту, коментар, ставка, дедлайн;
– вихідна: статус заявки.

3.3.3) Функціональні вимоги

– обмеження кількості заявок;
– можливість редагування заявки (бюджет та дедлайн включно);
– сповіщення замовника.

3.4) Система повідомлень

3.4.1) Опис функції

Забезпечує обмін повідомленнями між користувачами.

3.4.2) Вхідна і вихідна інформація

– вхідна: текст повідомлення;
– вихідна: історія чату.

3.4.3) Функціональні вимоги

– real-time обмін повідомленнями;
– збереження історії;
– сповіщення про нові повідомлення.

3.5) Відгуки та рейтинг

3.5.1) Опис функції

Користувачі можуть залишати відгуки після завершення проєкту.

3.5.2) Вхідна і вихідна інформація

– вхідна: оцінка, коментар;
– вихідна: рейтинг профілю.

3.5.3) Функціональні вимоги

– оцінка за 5-бальною шкалою;
– заборона повторних відгуків;
– перерахунок рейтингу.

4) Вимоги до інформаційного забезпечення

4.1) Джерела і зміст вхідної інформації

– дані користувачів (профілі, ролі);
– інформація про проєкти;
– заявки та повідомлення;
– відгуки та рейтинги.

4.2) Нормативно-довідкова інформація

– класифікатор категорій послуг;
– ролі користувачів;
– статуси проєктів (скасований, відкритий, в процесі, завершений).

4.3) Вимоги до зберігання

– використання PostgreSQL;
– нормалізована структура даних;

– резервне копіювання;
– шифрування чутливих даних.

5) Вимоги до технічного забезпечення

– сервер з підтримкою PHP 8+;
– мінімум 2 CPU / 4 GB RAM;
– можливість розгортання у VPS або хмарі;
– стабільне інтернет-з’єднання;
– наявність резервного копіювання даних.

6) Вимоги до програмного забезпечення

6.1) Архітектура

– клієнт-серверна архітектура;
– MVC-підхід.

6.2) Системне ПЗ

– Linux (Ubuntu) або Windows OS / Windows Server;
– Nginx або Apache.

6.3) Мережне ПЗ

– HTTPS;
– AJAX (для чату).
– PostgreSQL 13+;
– ORM Eloquent.

6.4) Мова і технології

– PHP (Laravel);
– JavaScript;
– HTML/CSS (Bootstrap, Blade).

7) Вимоги до зовнішніх інтерфейсів

7.1) Інтерфейс користувача

– адаптивний дизайн;
– підтримка мобільних пристроїв;
– інтуїтивна навігація.

7.2) Апаратний інтерфейс

Стандартні ПК та смартфони.

7.3) Програмний інтерфейс

Взаємодія між клієнтською та серверною частинами здійснюється через серверні маршрути (Laravel routes);

– обробка запитів виконується безпосередньо через контролери Laravel;

– передача даних між компонентами здійснюється у форматі HTTP-запитів із використанням стандартних механізмів вебформи;

– використовується Blade-шаблонізація для формування динамічних сторінок.

7.4) Комунікаційний протокол

– HTTPS для захищеної передачі даних між клієнтом і сервером;

– AJAX для асинхронного обміну даними, зокрема для реалізації функціоналу чату та оновлення повідомлень без перезавантаження сторінки.

8) Властивості програмного забезпечення

8.1) Доступність

99% uptime.

8.2) Супроводжуваність

– монолітна структура;

– документація коду.

8.3) Переносимість

– можливість розгортання системи на різних операційних системах (Linux, Windows Server) без змін у вихідному коді;

– підтримка розгортання у різних середовищах: локальне (development), тестове (staging) та продуктивне (production);

– використання стандартних інструментів розгортання (Docker або CI/CD за потреби);

– незалежність від конкретного хостинг-провайдера (можливість роботи на VPS, виділених серверах або хмарних платформах);

– сумісність із популярними вебсерверами (Nginx, Apache) без додаткових модифікацій;

– можливість міграції бази даних PostgreSQL між різними серверами без втрати даних.

8.4) Продуктивність

– час відповіді системи на запити користувача не повинен перевищувати 1–2 секунд у стандартних умовах навантаження;

– система повинна забезпечувати одночасну роботу щонайменше 100–300 активних користувачів без суттєвого зниження продуктивності;

– оптимізація запитів до бази даних (індексація, кешування) для зменшення часу обробки;

– час завантаження основних сторінок (профіль, проєкти, список заявок) не повинен перевищувати 2 секунд.

8.5) Надійність

– забезпечення цілісності та збереження даних користувачів, проєктів, заявок та повідомлень;

– захист від втрати даних у разі збоїв програмного забезпечення або серверної частини;

– можливість відновлення системи після збоїв без втрати критичної інформації;

– регулярне автоматичне резервне копіювання бази даних із можливістю відновлення на визначений момент часу;

– використання транзакцій бази даних для забезпечення узгодженості даних (PostgreSQL).

8.6) Безпека

– захист від XSS, CSRF, SQL-injection;

– автентифікація через токени та контроль доступу за ролями.

9) Інші вимоги

– можливість інтеграції ШІ у майбутньому;

– можливість масштабування функціоналу;
– можливість інтеграції платіжних систем у майбутньому.

Висновки до розділу 2

Проведено аналіз стеку технологій та аргументовано його вибір через порівняльний аналіз. Сформовано цілісну модель вебзастосунку. Створено технічну та методологічну основу для подальшої реалізації системи й програмної частини проєкту.

Обрано PHP як мову програмування, Laravel як основний фреймворк для серверної частини, PostgreSQL як СУБД, а для реалізації UI – фреймворк Bootstrap та шаблонізатор Blade.

Специфіковано вимоги для вебзастосунку що розробляється. Виконано опис ключових функціональних можливостей системи, а саме реєстрація та автентифікація, управління проєктами, подання пропозицій, система сповіщення, а також відгуки та рейтинг. Визначено ролі користувачів у межах застосунку.

Отже, у другому розділі проведено моделювання вебзастосунку взаємодії фрілансерів та замовників.

3 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ

3.1 Розробка та опис діаграм UML

Під час розробки фріланс-маркетплейсу UML-діаграми доцільно використовувати як універсальний візуальний інструмент, що допомагає сформувати спільне розуміння системи між замовником, командою розробників і майбутніми користувачами платформи. Вони дають змогу чітко описати основні функціональні сценарії роботи сервісу, відобразити структуру програмних компонентів, а також визначити механізми взаємодії із зовнішніми сервісами та користувачами [13].

Мова моделювання UML охоплює різні рівні проєктування – від загального представлення бізнес-процесів до деталізованого опису технічної реалізації. Завдяки цьому модель системи можна послідовно уточнювати на кожному етапі створення маркетплейсу, зберігаючи логічну цілісність і узгодженість усіх її складових.

Для проєктування поточного проєкту буде застосовано ряд діаграм та інструментів для їх створення (табл. 3.1).

Таблиця 3.1 – Типи діаграм та їх опис

Тип діаграми	Мета використання	Інструмент	Охоплення в проєкті
Діаграма прецедентів (Use Case)	Визначення ключових функцій системи та ролей користувачів, які взаємодіють із платформою	PlantUML Editor	Замовник, виконавець (фрілансер), сервіс, адміністратор
Діаграма послідовності (Sequence)	Послідовність повідомлень між об’єктами у сценаріях	PlantUML Editor	Створення проєкту (замовлення), написання відгуку, поповнення рахунку тощо

Кінець таблиці 3.1

Діаграма класів (Class diagram)	Опис статичної архітектури системи, її сутностей, атрибутів і методів	PlantUML Editor	Множина класів: User, Order, Transaction, Balance, BankAccount, MainOrderCategory, SubOrderCategory, OrderStatus, TransactionType, UserRole, OrderApprove, OrderComment, Review, OrderFileAttachment, OrderCommentFileAttachment, UserAvatar
Діаграми діяльності (activity diagram)	Моделювання логіки виконання процесів та послідовності дій у межах операцій	PlantUML Editor	PlantUML Editor
Діаграма розгортання (Deployment Diagram)	Представлення фізичної інфраструктури системи та розміщення її компонентів	PlantUML Editor	Вебсервер, сервер БД

Сукупність UML-діаграм охоплює основні рівні проєктування фріланс-маркетплейсу – від бізнес-сценаріїв взаємодії користувачів до фізичної структури системи. Це дає змогу послідовно деталізувати модель платформи на всіх етапах її розроблення.

Спочатку формуються діаграми прецедентів для визначення ролей користувачів і функціональних можливостей системи. Далі створюються діаграми класів і взаємодії, що описують структуру сутностей та логіку виконання основних операцій. Після цього застосовуються діаграми діяльності для моделювання окремих процесів, а завершальним етапом є діаграма розгортання, яка відображає серверну інфраструктуру.

3.1.1 Діаграма прецедентів

На діаграмі прецедентів зображено чотири основні актори: «Гість», «Замовник», «Фрілансер», «Сервіс» та «Адміністратор». Їх взаємодія з системою описана через функціональні блоки (автентифікація, керування проєктами, робота із заявками, фінансові операції, комунікація, адміністрування).

Всі актори успадковуються від актора «Гість». Тобто, «Гість» є базовою сутністю для замовників, виконавців та адміністраторів. Гість неавторизований користувач системи, що обмежує доступ до функціоналу платформи, він може переглядати загальну інформацію про проєкти та профілі користувачів. Також здійснювати реєстрацію та вхід у систему. Замовник створює та керує проєктами, переглядає заявки та взаємодіє з виконавцями. Виконавець (фрілансер) переглядає проєкти, подає заявки та бере участь у комунікації. Сервіс відповідає за фінансові операції, включно з депонуванням та переказами коштів, а адміністратор здійснює модерацію користувачів, контенту та скарг.

Структура схеми поділена на блоки визначених за функціональністю. Поділ чітко виокремлює кожну підсистему платформи та покращує її сприйняття та розуміння.

Повна діаграма використання продемонстрована у **додатку А (рис. А.1).**

3.1.2 Діаграма послідовності

Завдяки діаграмі взаємодії демонструється порядок повідомлень між компонентами системи в межах конкретного сценарію. Виклики зображуються у послідовності за часом. Компоненти ініціюють дії та показують які саме дані передаються між ними та відповіді після запитів.

В рамках поточної роботи діаграми взаємодії використані для опису головних бізнес-процесів: реєстрації та авторизації користувачів, створення й завершення проєктів, роботи із заявками, а також фінансових операцій із використанням депонування. Вони демонструють взаємодію між користувачами, серверною частиною системи, сервісними модулями та адміністративними компонентами.

Під час створення та розбору діаграм взаємодії (на етапі проєктування) виявляються надлишкові залежності, валідується логіка комунікації між елементами системи та зменшується кількість потенційних помилок.

Діаграма послідовності на рис. 3.1 описує процес реєстрації користувача в системі: від заповнення форми та відправки запиту до валідації даних, створення запису користувача й пов’язаного балансу в базі даних із використанням транзакції для забезпечення цілісності даних.

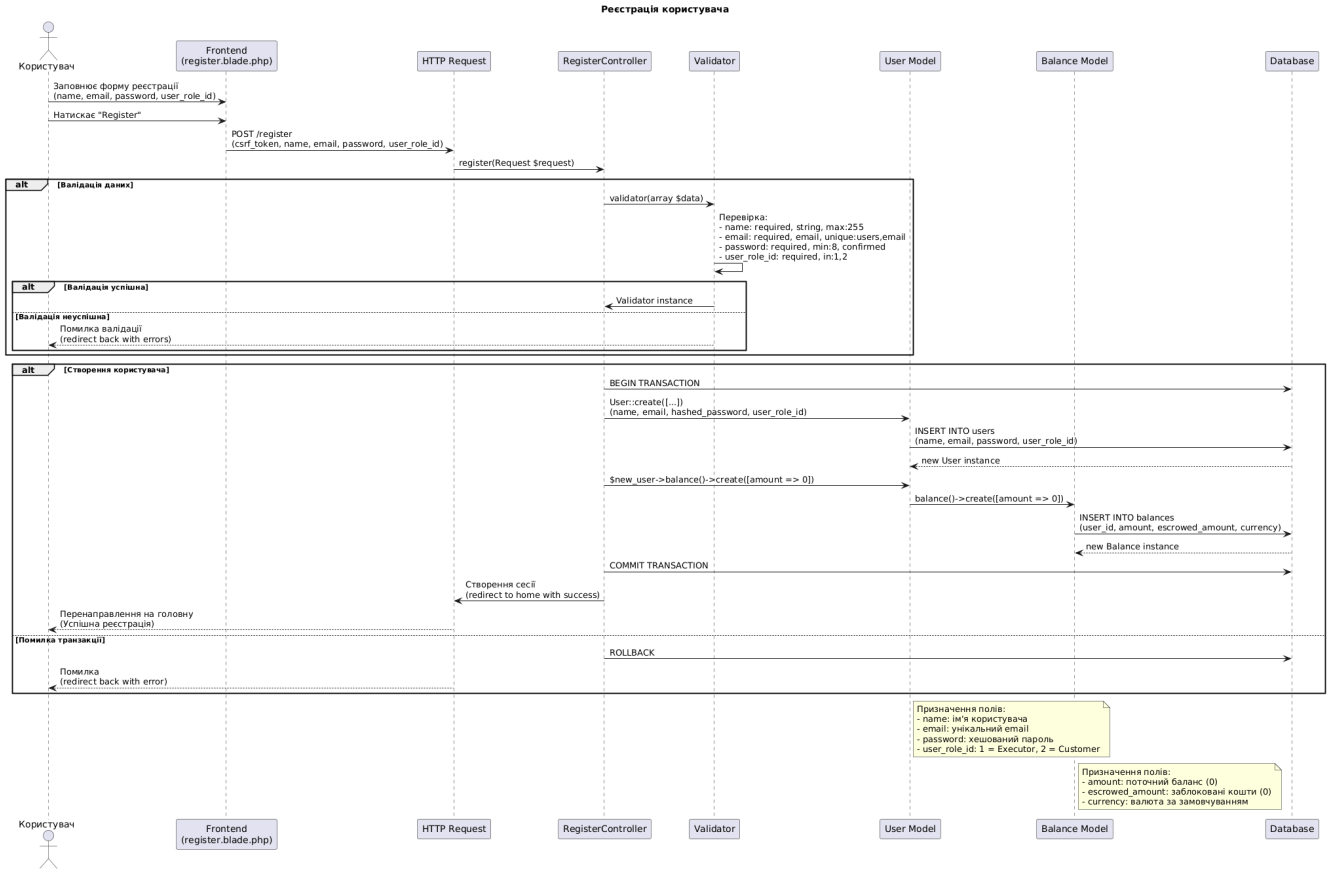


Рисунок 3.1 – Діаграма послідовності процесу реєстрації

Діаграма послідовності на рис. 3.2 описує процес створення замовлення замовником: перевірку ролі користувача, валідацію введених даних, перевірку достатності балансу, створення замовлення з можливими вкладеннями та резервування коштів через механізм «escrow» із використанням транзакції для забезпечення цілісності даних.

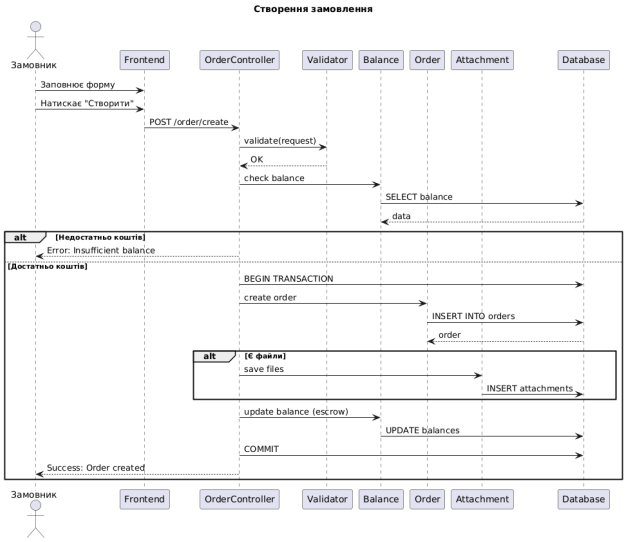


Рисунок 3.2 – Діаграма послідовності процесу подання пропозиції

Діаграма послідовності на рис. 3.3 описує процес подання пропозиції фрілансером до замовлення: перевірку статусу замовлення, валідацію даних, перевірку на дублювання пропозиції та створення запису пропозиції в базі даних.

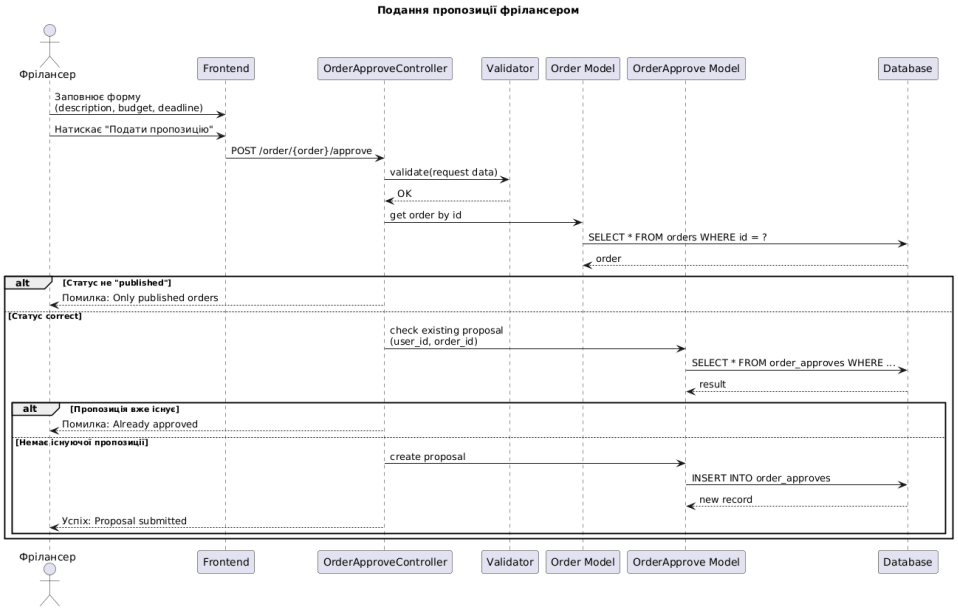


Рисунок 3.3 – Діаграма послідовності процесу подання пропозиції

Діаграма послідовності на рис. 3.4 описує процес поповнення балансу користувача: перевірку введених даних і права доступу до банківського рахунку, оновлення балансу із блокуванням запису для забезпечення цілісності даних, створення запису фінансової транзакції типу депозит та збереження всіх змін у базі даних у межах транзакції.

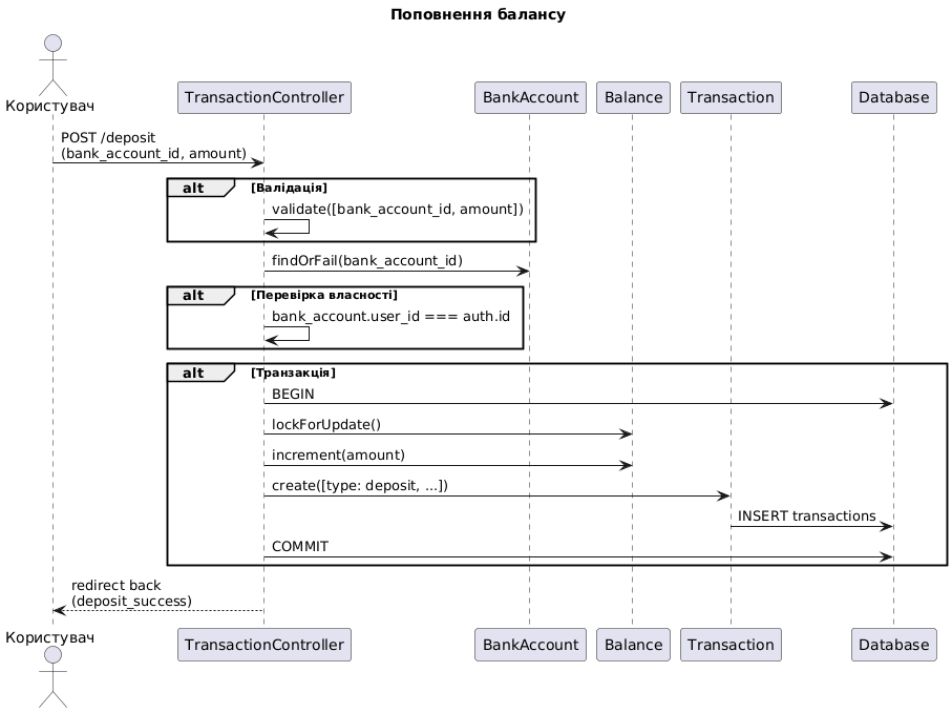


Рисунок 3.4 – Діаграма послідовності процесу депозиту

Діаграма послідовності на рис. 3.5 описує процес завершення замовлення та розподілу коштів із escrow-рахунку: перевірку прав замовника і статусу замовлення, виконання транзакції з блокуванням записів балансу, списання коштів із escrow, нарахування оплати виконавцю з урахуванням комісії та фіксацію всіх фінансових операцій у базі даних із забезпеченням цілісності через транзакцію.

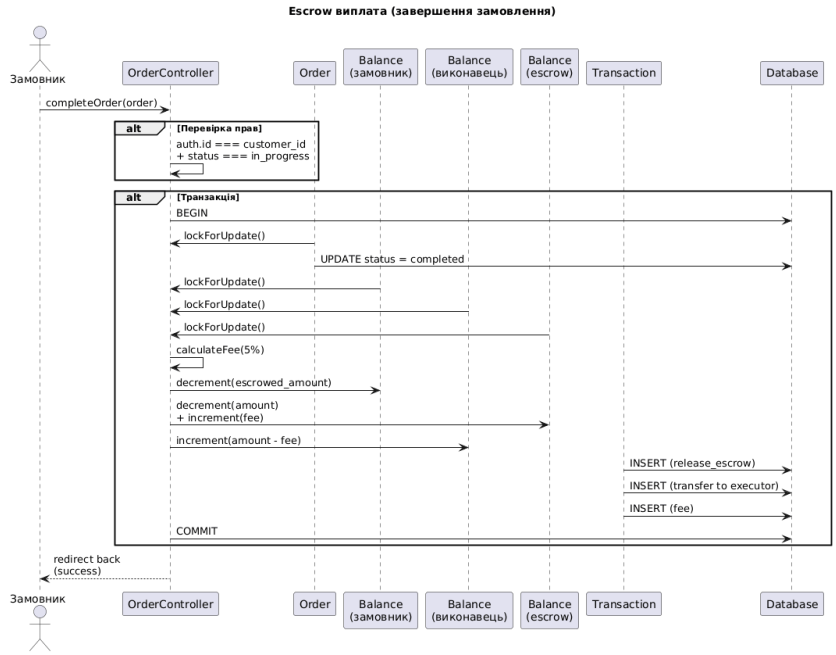


Рисунок 3.5 – Діаграма послідовності процесу депозиту

3.1.3 Діаграма класів

Діаграма класів відображає статичну структуру доменної моделі вебзастосунку для взаємодії фрілансерів та замовників: основні сутності, їхні атрибути, методи та взаємозв’язки. Вона слугує схемою даних і бізнес-логіки системи, що дозволяє узгодити структуру даних, назви полів і правила взаємодії між сутностями ще до реалізації бази даних [14].

У межах проєкту діаграма класів фіксує ключові доменні об’єкти, такі як User, Order, Balance, Transaction, OrderApprove, Review, а також допоміжні сутності (UserRole, OrderStatus, BankAccount, категорії замовлень, вкладення та коментарі). Вона відображає ролі користувачів (замовник і виконавець), механізми роботи із замовленнями, фінансовими операціями через escrow-модель, систему пропозицій фрілансерів і систему відгуків.

Завдяки діаграмі класів спрощується створення ORM-моделі, бізнес-логіки у контролерах та сервісах, структури бази, так як вона є загальною основою для всього проєкту.

Класи системи структуровані таким чином, щоб забезпечити відповідність функціональним вимогам платформи. Визначено розмежування ролей користувачів через сутність UserRole, що дозволяє реалізувати різні сценарії взаємодії: замовник створює замовлення та резервує кошти, а виконавець подає пропозиції та виконує роботу. Адміністрування доступу реалізується на рівні логіки користувача та ролей.

Клас User є центральною сутністю системи та пов’язаний із балансом, банківськими рахунками, замовленнями та пропозиціями. Клас Order реалізує життєвий цикл замовлення та пов’язаний зі статусами, категоріями, вкладеннями, пропозиціями, коментарями та відгуками. Фінансова логіка представлена через класи Balance і Transaction, які забезпечують роботу депонування, поповнення балансу та перекази коштів між користувачами.

Створена діаграма класів продемонстрована у **додатку А (рис. А.2).**

3.1.4 Діаграми діяльності

Послідовність та логіка основних процесів може бути зображена за допомогою діаграм діяльності. У рамках поточного проєкту діаграми діяльності використовується для моделювання таких сценаріїв, як реєстрація користувача, створення проєкту, подача заявки виконавцем, вибір кандидата замовником, обмін повідомленнями, завершення, відгук тощо.

Етапи виконання процесу та можливі умови переходів між діями демонструються завдяки цьому типу діаграми: зліва на право у вигляді спадання. Зображуються паралельні та синхронні процеси між компонентами системи. У процесі проєктування можливо виявити залежності між модулями, усунути зайві кроки та визначити потенційні проблеми у роботі платформи.

Бізнес-процеси у графічному вигляді спрощують комунікацію між розробником, керівником роботи та іншими учасниками проєкту. Діаграма діяльності слугує основою для подальшого тестування функціоналу, створення користувацьких сценаріїв і перевірки коректності реалізації вебзастосунку.

Діаграма на рис. 3.6 показує процес реєстрації користувача: заповнення форми, перевірку даних, створення акаунта та балансу при успіху або повернення до форми з помилками при невдалій валідації.

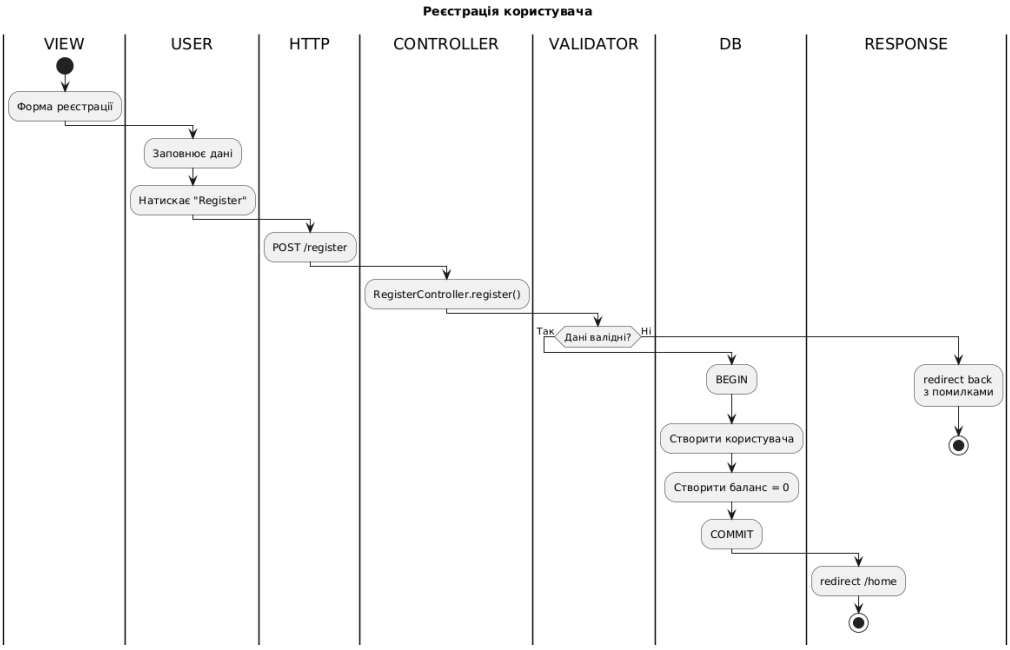


Рисунок 3.6 – Діаграма діяльності «реєстрація користувача»

Діаграма на рис. 3.7 демонструє процес створення замовлення: користувач відкриває форму, заповнює дані та надсилає запит. Система перевіряє роль користувача, валідність введених даних і наявність достатнього балансу. Якщо всі умови виконані, створюється замовлення, кошти резервуються, після чого користувач перенаправляється на головну сторінку. Якщо виникає помилка, користувач повертається назад із повідомленням.

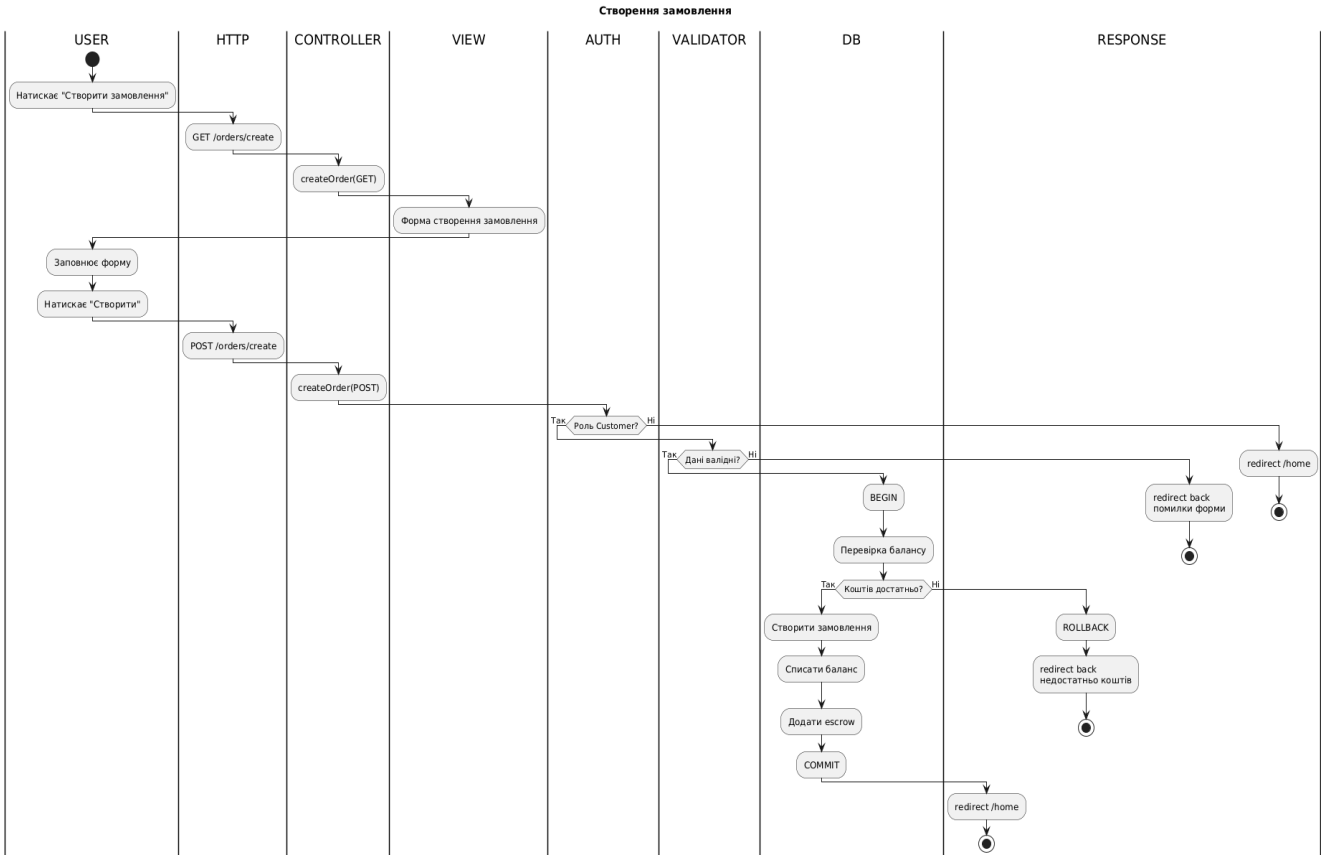


Рисунок 3.7 – Діаграма діяльності «Створення замовлення»

Діаграма на рис. 3.8 демонструє процес виведення коштів: користувач відкриває сторінку фінансів, заповнює форму та надсилає запит на виведення. Система перевіряє коректність даних, доступ до картки та наявність достатнього балансу. Якщо всі умови виконані, зменшується баланс і створюється транзакція, після чого користувач отримує підтвердження успішної операції. У разі помилки процес завершується з відповідним повідомленням.

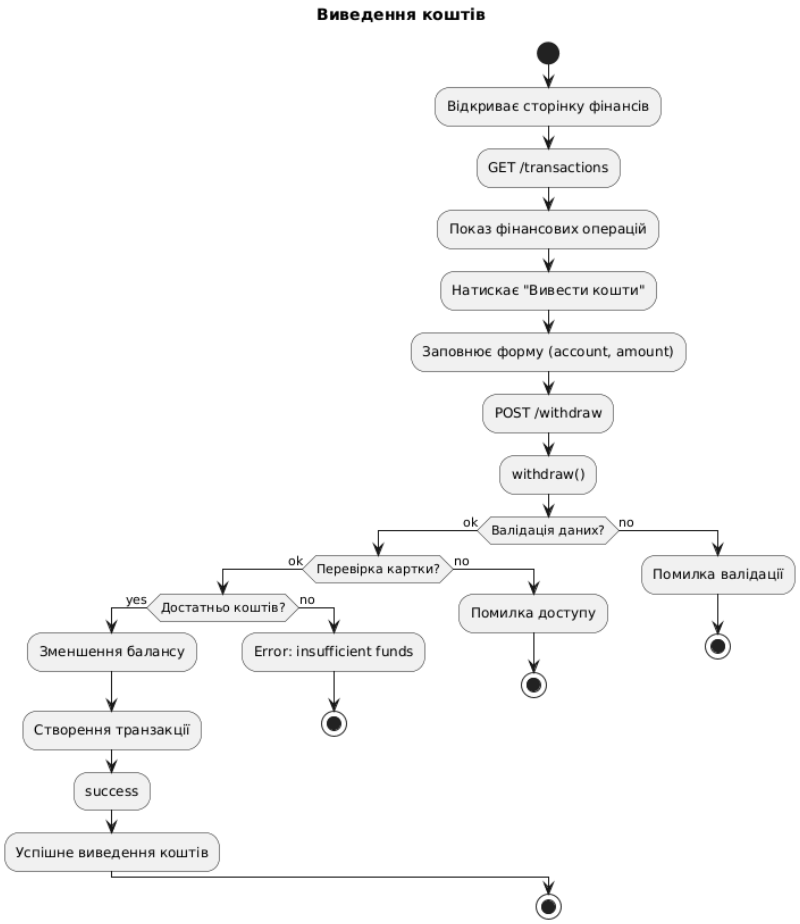


Рисунок 3.8 – Діаграма діяльності «Створення замовлення»

Діаграми діяльності, що продемонстровані вище надають загальну картину архітектури проєкту, яку можна використати для подальшої реалізації й тестування функціоналу платформи.

3.1.5 Діаграма розгортання

Діаграма розгортання (рис. 3.9) відображає фізичну структуру системи та взаємодію її основних компонентів у робочому середовищі. Клієнтська частина представлена браузером, який надсилає HTTP/HTTPS-запити до вебсервера Apache. Вебсервер обробляє статичний контент і передає динамічні запити до серверу застосунків через FastCGI.

На сервері застосунків використовується PHP-FPM, який виконує логіку Laravel-застосунку. У межах фреймворку обробка запитів здійснюється через HTTP-контролери, бізнес-логіку, моделі, middleware та сервіс-провайдери. Для збереження файлів використовується файлове сховище (storage disk).

Застосунок взаємодіє з сервером бази даних PostgreSQL за допомогою SQL-запитів. Додатково використовується кешування на рівні бази даних для підвищення продуктивності та зменшення навантаження.

Повна діаграма розгортання продемонстрована у **додатку А (рис. А.3),** яка ілюструє мережеві зв’язки між компонентами та розподіл відповідальності між ними, що дозволяє краще зрозуміти архітектуру системи, а також спрощує процеси масштабування, оптимізації продуктивності та адміністрування.

3.2 Варіанти використання системи

Використання будь якої системи може бути описано за допомогою прецедентів. Варіанти використання системи без дотику до глибокої реалізації описують поведінку застосунку. По суті своєї кожен прецедент це послідовність дій, що визначає конкретний результат й тим самим розкриває функціональні можливості.

Головними акторами у рамках застосунку визначено гостя, виконавця, замовника та адміністратора. Користувачі з такими ролями взаємодіють з застосунком за посередництвом браузера. Сценарії запускаються користувачами, наприклад сценарій реєстрації та авторизації, перегляду і взаємодії з контентом тощо.

Сценарій використання «Реєстрація користувача» (табл. 3.2) надає опис процесу реєстрації нового облікового запису. Гість заповнює реєстраційну форму, вводить персональні дані та обирає роль у системі. Виконується валідація введених даних та створюється акаунт. БД зберігає інформацію. Надається доступ до обмеженого функціоналу.

Таблиця 3.2 – Прецендент «Реєстрація користувача»

UseCase Name	Реєстрація акаунта
Scope	Вебзастосунок взаємодії фрілансерів та замовників
Level	User goal level
Primary Actor	Гість (неавторизований користувач)

Кінець таблиці 3.2

Stakeholders and interests	**Гість:** бажає створити акаунт і отримати доступ до системи. **Адміністратор:** отримання коректних та унікальних акаунтів. **Система:** безпечна реєстрація та збереження даних.
Preconditions	Користувач не авторизований у системі. Користувач має доступ до сторінки реєстрації. Email ще не використовується в системі.
Success guarantee	Новий акаунт успішно створено, дані збережено в базі даних, користувачу надіслано лист підтвердження або виконано автоматичний вхід у систему.
Main Success Scenario	1) Користувач відкриває сторінку реєстрації; 2) система відображає форму реєстрації; 3) користувач вводить ім’я, email, пароль, роль (замовник / фрілансер); 4) користувач підтверджує відправку форми; 5) система перевіряє коректність даних; 6) система створює новий акаунт.

Прецендент «Створення проєкту» (табл. 3.3) описує процес публікації нового замовлення у системі. Замовник заповнює форму із параметрами проєкту, після чого система перевіряє дані, зберігає запис у базі даних та робить проєкт доступним для фрілансерів.

Таблиця 3.3 – Прецендент «Створення проєкту»

UseCase Name	Створення проєкту
Scope	Вебзастосунок взаємодії фрілансерів та замовників
Level	User goal level
Primary Actor	Замовник
Stakeholders and interests	**Замовник:** бажає опублікувати проєкт та знайти виконавця. **Фрілансер:** отримання нових доступних замовлень. **Система:** коректне збереження даних проєкту та відображення його в каталозі.
Preconditions	Користувач авторизований у системі та має роль замовника.
Success guarantee	Проєкт успішно створено, дані збережено в базі даних, проєкт доступний для перегляду фрілансерами.
Main Success Scenario	1) Замовник відкриває сторінку створення проєкту; 2) система відображає форму створення; 3) замовник вводить назву, опис, бюджет, категорію та термін виконання; 4) замовник надсилає форму; 5) система перевіряє коректність даних; 6) система створює проєкт.

Кінець таблиці 3.3

Extensions	3а. Не заповнені обов'язкові поля → система показує помилку. 5а. Некоректний бюджет або дата → система вимагає виправлення. 6а. Помилка збереження БД → система повідомляє про помилку.
Special Requirements	HTTPS-з'єднання. Валідація даних. Адаптивний інтерфейс. Захист від несанкціонованого доступу.
Technology List and Variations Data	Frontend: Blade, Bootstrap. Backend: Laravel. База даних: PostgreSQL.
Frequency of Occurrence	Висока – замовники регулярно створюють нові проєкти.
Miscellaneous	Після створення проєкт може бути відредагований або видалений замовником.

Прецендент «Вибір виконавця» (табл. 3.4) описує процес призначення фрілансера до проєкту після перегляду поданих заявок.

Таблиця 3.4 – Прецендент «Вибір виконавця»

UseCase Name	Вибір виконавця
Scope	Вебзастосунок взаємодії фрілансерів та замовників
Level	User goal level
Primary Actor	Замовник
Stakeholders and interests	**Замовник:** обрати виконавця. **Фрілансер:** отримати замовлення. **Система:** коректно зафіксувати вибір.
Preconditions	Замовник авторизований. Проєкт активний. Є заявки від виконавців.
Success guarantee	Виконавця призначено до проєкту, статус оновлено.
Main Success Scenario	1) Замовник відкриває список заявок. 2) Переглядає кандидатів. 3) Обирає виконавця. 4) Система підтверджує вибір. 5) Статус проєкту оновлюється.

Кінець таблиці 3.4

Extensions	2a. Немає заявок → система показує повідомлення. 4a. Помилка збереження → система повідомляє про помилку.
Special Requirements	HTTPS. Захист доступу. Швидке оновлення статусу.
Technology List and Variations Data	Blade, Bootstrap, Laravel, PostgreSQL.
Frequency of Occurrence	Середня – після отримання заявок.
Miscellaneous	Інші заявки можуть бути автоматично закриті.

Кожен прецендент логічно описує типову поведінку користувача. Такий опис надає систематизуванні функціональні вимоги та визначає межі системи. У подальшому кожен прецедент стає основою для подальшого проєктування, тестування й реалізації серверної логіки.

3.3 Проєктування архітектури та вибір технологічного стеку

Для моделі застосунку визначено три рівні: представлення, серверна логіка та БД. Відповідальність між елементами, яка розподілена задля спрощення підтримки дозволяє у простий спосіб масштабувати окремі компоненти.

Blade та Bootstrap обрано для реалізації клієнтської частини. Шаблонізатор Blade застосовується для створення динамічних сторінок. Bootstrap використовується для відтворення адаптивного інтерфейсу на десктпоп і мобільних пристроях.

Фреймворк Laravel обрано задля побудови серверної частини. Засоби маршрутизації, автентифікації, авторизації, валідації даних і роботи з БД включенні в стандартні модулі даного фреймворку. Laravel гармонійно реалізує MVC виокремлюючи логіку застосунку, дані та інтерфейс користувача.

БД використовується для збереження інформації про користувачів, проєкти, заявки, повідомлення та відгуки. У якості реляційної СКБД для зберігання даних

використано PostgreSQL. Вибір аргументується її надійністю, підтримкою транзакцій і складних SQL-запитів.

Хешуванням паролів, використанням HTTPS та розмежуванням прав доступу залежно від ролі користувача (замовник, замовник або адміністратор) визначають безпеку системи.

PHPUnit застосовано для тестування. Даний фреймворк тестування дозволяє перевіряти коректність роботи модулів і виявити помилки своєчасно.

При необхідності масштабування, наприклад, через збільшення навантаження системи розширення може бути виконано на окремий сервер з застосуванням кешування та балансування навантаження.

Загальний стек технологій продемонстровано у таблиці 3.5.

Таблиця 3.5 – Стек технологій

Напрям	**Технологія**	**Призначення**
Серверна мова	PHP 8	Основа backend-частини системи
Backend-фреймворк	Laravel	Логіка застосунку, маршрути, безпека
Шаблони сторінок	Blade	Генерація вебсторінок
Оформлення	Bootstrap 5	Адаптивний дизайн та UI-елементи
Клієнтські скрипти	JavaScript	Динамічна взаємодія з інтерфейсом
Робота з БД	Eloquent ORM	Зручне керування даними через моделі
Сховище даних	PostgreSQL	Надійне зберігання інформації
Архітектура	MVC	Поділ логіки, даних і вигляду
Захист	HTTPS, Hash	Безпечна передача і зберігання паролів
Тестування	PHPUnit	Перевірка стабільності системи

Таким чином, обрана архітектура є простою, надійною та гнучкою, повністю відповідає потребам проєкту й забезпечує можливість подальшого розвитку вебзастосунку.

3.4 Мокапи інтерфейсів

На початковому етапі проєктування для відображення структури та зовнішнього вигляду основних сторінок платформи фрілансерів і замовників було розроблено серію мокапів.

Мокап головної сторінки вебзастосунку (рис. 3.9), що відображає список доступних проєктів для фрілансерів. Інтерфейс містить навігаційне меню, фільтри категорій, картки замовлень з основною інформацією та кнопки перегляду деталей.

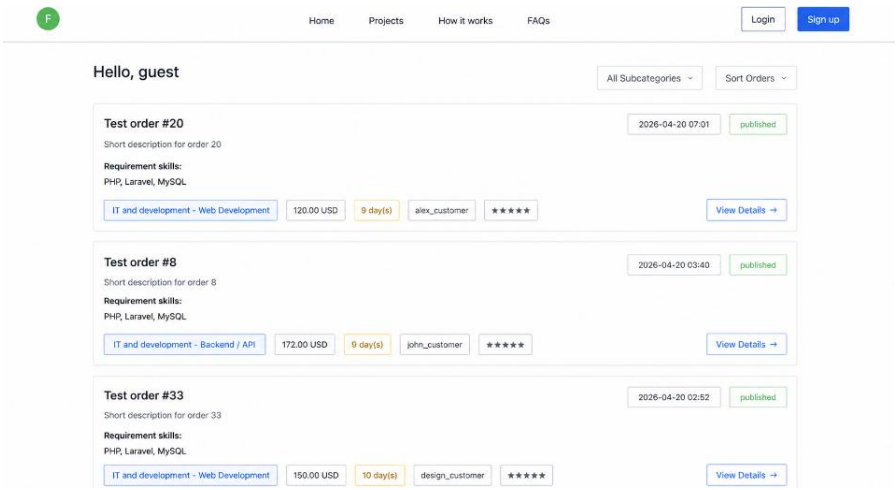


Рисунок 3.9 – Мокап головної сторінки

Мокап сторінки керування проєктом (рис 3.10), що містить форму редагування замовлення, блок завантаження файлів та список заявок від виконавців. Інтерфейс забезпечує зручне керування параметрами проєкту, перегляд пропозицій і взаємодію із кандидатами.

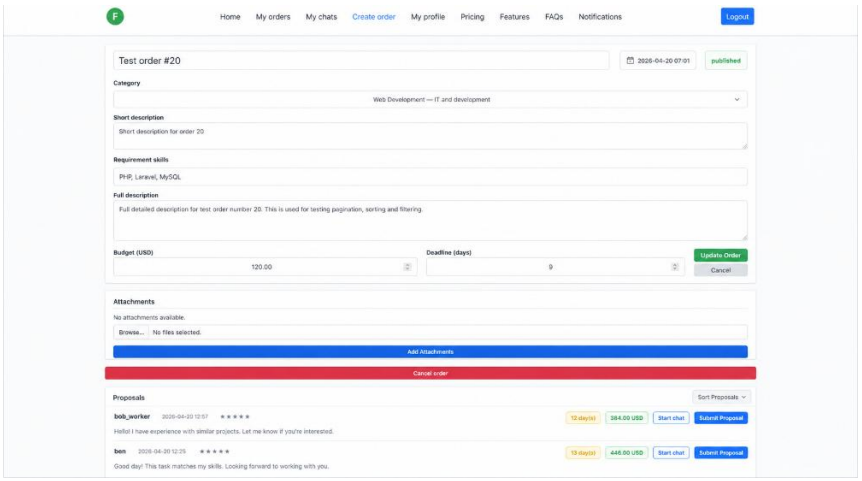


Рисунок 3.10 – Мокап сторінки створення замовлення

Мокап сторінки з відкритим чатом (рис. 3.11) містить список діалогів, область повідомлень та форму введення нового повідомлення для швидкої комунікації користувачів.

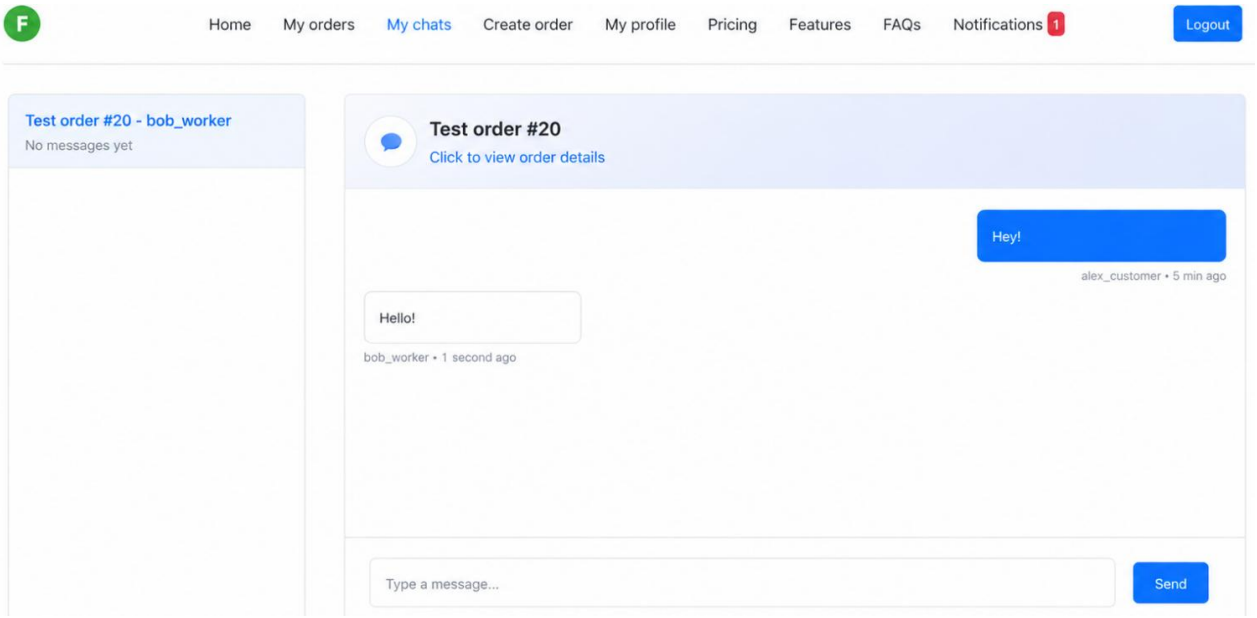


Рисунок 3.11 – Мокап сторінки з відкритим чатом

Мокап сторінки профілю користувача (рис. 3.12) демонструє особисту інформацію, можливість зміни аватара та пароля, з навігацією по розділах і зручним поділом на блоки.

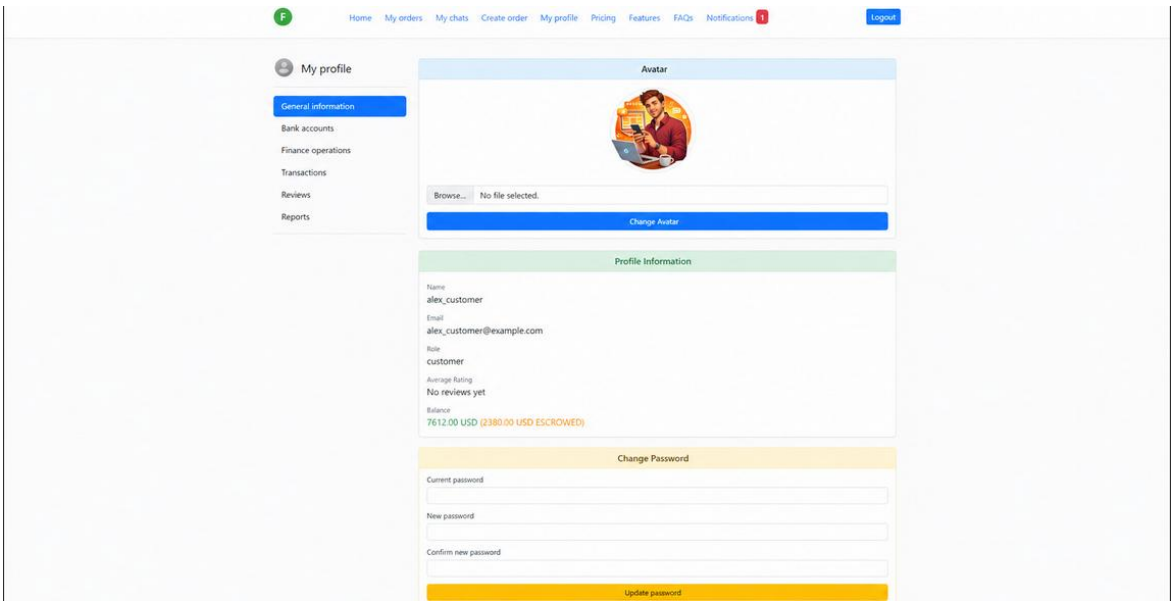


Рисунок 3.12 – Мокап сторінки перегляду профілю

Мокап сторінки керування банківськими картками (рис. 3.13) демонструє: перегляд доданої карти, її видалення та форма для додавання нової.

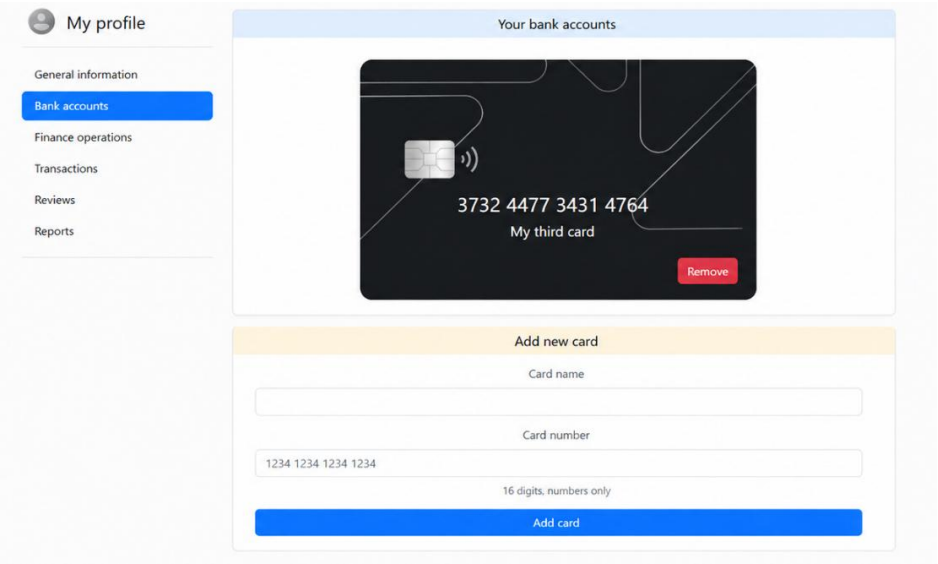


Рисунок 3.13 – Мокап сторінки керування банківськими картками

Мокап демонструє сторінку деталей скарги (рис. 3.14): інформацію про проблему, статус, пов’язане замовлення та блок для коментарів.

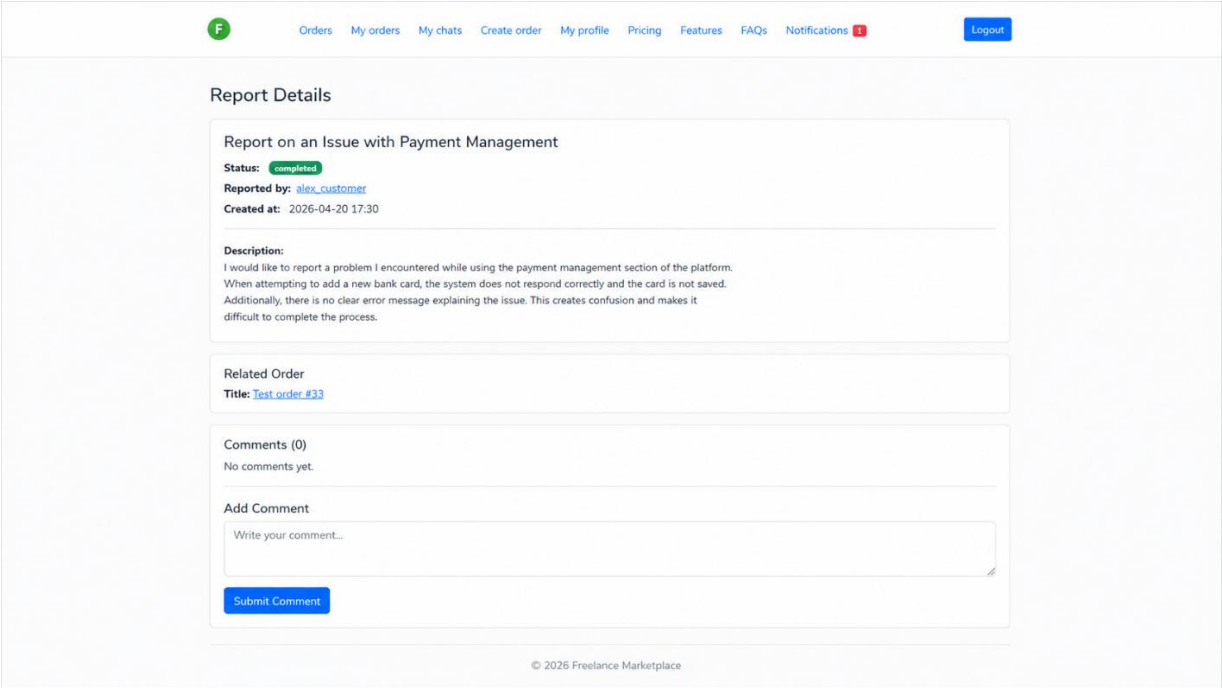


Рисунок 3.14 – Мокап сторінки керування банківськими картками

В загалом розроблені мокапи демонструють графічні прототипи сторінок з основним функціоналом.

Висновки до розділу 3

Застосовано комплекс UML-діаграм для проєктування вебзастосунку взаємодії фрілансерів. Розроблено: діаграми варіантів використання, діаграми послідовності, діаграми класів, діаграми діяльності та діаграми розгортання. Формалізовано основні бізнес-процеси взаємодії користувачів.

Визначено мови програмування для реалізації проєкту. Обрано архітектурну модель та технологічний стек, що базується на Laravel, PostgreSQL, Blade, Bootstrap, PHPUnit. Розроблено мокапи інтерфейсу основних сторінок запланованої платформи.

У сукупності, як результат третій розділ представляє цілісний фундамент для практичної розробки вебзастосунку.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

До основних сторінок вебзастосунку, які призначені лише для авторизованих користувачів відносяться: головна сторінка – каталог проєктів, сторінка окремого проєкту, сторінка заявок (пропозицій), чат (повідомлення), особистий кабінет користувача, сторінка перегляду скарг, сторінка перегляду транзакцій, сторінка з банківськими рахунками.

Множина ролей, які передбачені у межах платформи складається з замовника, виконавця, адміністратора, а також сервісна роль для внутрішніх процесів системи.

До функцій замовника відносяться: можливість створювати та публікувати проєкти, переглядати заявки виконавців, обирати підрядників, контролювати виконання завдань і залишати відгуки.

До функцій виконавця відносяться: можливість переглядати доступні проєкти, фільтрувати їх за різними параметрами, подавати заявки на виконання, комунікувати із замовниками та виконувати роботи.

До функцій адміністратора відносяться: модерація контент, контроль дотримання правил, керування користувачами, проєктами та відгуками.

Ролі інтегровані в єдину логічну модель. Кожен користувач має доступ лише до відповідного функціоналу. Права адміністратора призначаються вручну. Така структура є заставою для масштабування функціоналу й подальшого розвитку вебзастосунку.

4.1 Структура та специфікація коду

Вебзастосунок реалізовано на основі архітектури MVC та структурно поданий у вигляді множин: контролерів, моделей та представлень. Додатково відповідно до кожної моделі створено набір міграцій та сідерів, які надають можливість легко створювати, змінювати та поширювати структуру БД. Маршрутизація для проєктів реалізованих за допомогою фреймворку Laravel визначається в спеціальному файлі «web.php».

Опис застосованих контролерів наведено **у додатку Б (табл. Б.1).** Крім зазначених у таблиці, у проєкті використано стандартні контролери, автоматично згенеровані фреймворком Laravel: LoginController, ForgotPasswordController, ResetPasswordController, VerificationController, ConfirmPasswordController та інші.

Перелік використаних моделей подано **у додатку Б (табл. Б.2).** Частина моделей містить додаткові методи для зручної роботи зі зв'язками між сутностями.

Представлення та їх короткий опис наведено **у додатку Б (табл. Б.3).** У структурі інтерфейсу також використано часткові шаблони.

Список створених сідерів подано **у додатку Б (табл. Б.4)**, а перелік міграцій з описом – **у додатку Б (табл. Б.5)**.

Опис створених маршрутів наведено **у додатку Б (табл. Б.6).**

Файлова структура проєкту зображена на рис. 4.1. За директорією «app» розміщена бізнес-логіка застосунку, включно з контролерами для обробки HTTP-запитів, моделями для взаємодії з базою даних і сервісними провайдерами, що відповідають за налаштування та підключення компонентів системи.

```
freelance-marketplace/
├── app/
│   ├── Http/Controllers/ (18 контролерів)
│   ├── Models/ (16 моделей)
│   └── Providers/ (1 провайдер)
├── resources/views/ (36 Blade файлів)
│   ├── components/pages/ (home, orders, profile, chats, reports...)
│   └── auth/ (login, register, passwords, verify)
├── database/
│   ├── migrations/ (24 міграції)
│   └── seeders/ (23 сідери)
├── routes/ (web.php - 49 маршрутів)
├── public/ (css, js, images, storage)
└── config/ (app, auth, database, cache...)
```

Рисунок 4.1 – Файлова структура проєкту

Шар представлення знаходиться у каталозі «resources/views» і містить Blade-шаблони користувацького інтерфейсу, зокрема сторінки головної, профілю, чатів та авторизації, які формують візуальну частину застосунку.

Міграції та сідери для роботи з БД знаходяться в папці «database». Міграції визначають структуру таблиць, а сідери використовуються для заповнення бази тестовими або початковими даними.

Ресурси статичного типу розміщені в каталозі «public» (стилі, скрипти та зображення). Файли конфігурації розміщені у папці «config», де задаються основні налаштування застосунку.

Проєкт поділено архітектурно за структурою, що розділяє дані, бізнес-логіку та інтерфейс.

Зв’язки між сутностями на рівні моделей описані за допомогою ORM Eloquent. Наприклад, зв’язок між користувачем і проєктами реалізовано наступним чином:

```
// User.php
public function projects()
{
    return $this->hasMany(Project::class);
}

// Project.php
public function user()
{
    return $this->belongsTo(User::class);
}
```

Це дозволяє зручно отримувати пов’язані дані, наприклад:

```
$userProjects = auth()->user()->projects;
```

Ще однією особливістю є використання Blade-шаблонів із повторно використовуваними компонентами, що підвищує модульність інтерфейсу, наприклад:

```
@include('components.project.card', ['project' => $project])
```

Такий підхід дозволяє уникнути дублювання коду і спрощує підтримку UI. Для обмеження доступу до функціоналу реалізовано рольову авторизацію:

```
public function __construct()
{
    $this->middleware('auth');
    $this->middleware('role:admin')->only(['destroy', 'banUser']);
}
```

Це гарантує, що певні дії доступні лише адміністраторам.

Окрему увагу приділено валідації даних, яка виконується на рівні контролерів, наприклад в методі «validateOrderData()» класу «OrderController»:

```
$request->validate([
    'title' => 'required|string|max:255',
    'budget' => 'required|numeric|min:0',
]);
```

Це дозволяє запобігти некоректним даним ще до їх обробки.

Приклад міграції, що використовуються та забезпечують контроль версій структури БД для таблиці «projects»:

```
Schema::create('projects', function (Blueprint $table) {
    $table->id();
    $table->string('title');
    $table->text('description');
    $table->decimal('budget', 10, 2);
    $table->timestamps();
});
```

Ще одна важлива особливість – використання сідерів для початкових даних, наприклад для ролей:

```
DB::table('roles')->insert([
    ['name' => 'customer'],
    ['name' => 'freelancer'],
    ['name' => 'admin'], ]);
```

У системі також реалізовано динамічні обчислювані атрибути, наприклад для рейтингу користувача:

```
public function getAverageRatingAttribute()
{
    return $this->reviews()->avg('rating');
}
```

Це дозволяє звертатися до значення як до звичайного поля, наприклад:

```
$user->average_rating;
```

Варто звернути увагу на механізм депонування (escrow), який відповідає за безпечне проведення фінансових операцій між замовником і виконавцем.

Депонування передбачає тимчасове резервування (блокування) коштів замовника на період виконання замовлення. Даний механізм гарантує виконавцю оплату після успішного завершення роботи, так як кошти замовника заморожуються до виплати або повернення.

Реалізація цього механізму базується на використанні транзакцій бази даних:

```
DB::transaction(function () use ($data, $request) {
```

Використання транзакції забезпечує атомарність операцій, тобто або всі дії виконуються успішно, або у випадку помилки зміни скасовуються.

На першому етапі відбувається блокування запису балансу користувача для уникнення конкурентного доступу:

```
$balance = Balance::where('user_id', Auth::id())->lockForUpdate()->first();
```

Це гарантує, що під час виконання операції жоден інший процес не зможе змінити баланс користувача.

Далі виконується перевірка достатності коштів:

```
if ($data['budget'] > $balance->amount)
    throw new \Exception('Insufficient balance to create this order.');
```

У разі нестачі коштів створення замовлення переривається.

Після цього створюється нове замовлення:

```
$order = Auth::user()->ordersAsCustomer()->create([...]);
```

Ключовим етапом є переміщення коштів із доступного балансу до зарезервованого (escrow):

```
$balance->decrement('amount', $data['budget']);
$balance->increment('escrowed_amount', $data['budget']);
```

Таким чином, кошти не списуються остаточно, а лише блокуються до завершення угоди.

Для фіксації фінансової операції створюється запис у таблиці транзакцій:

```
Transaction::create([
    'user_id' => Auth::id(),
    'order_id' => $order->id,
    'amount' => $data['budget'],
    'transaction_type_id' => TransactionType::where('name', 'escrow')->value('id'),
    'bank_account_id' => null,
    'related_user_id' => User::where('name', 'escrow_service')->value('id'),
    'transfer_uuid' => (string) Str::uuid(),
    'meta' => ['type' => 'escrow', 'recipient' => 'escrow_service'],
]);
```

Вищевказаний запис визначає, що кошти передані не безпосередньо виконавцю, а на спеціальний сервісний акаунт (escrow_service), який виступає як посередник.

Після підтвердження виконання кошти мають два напрями для переводу:

– транзакція виконавцю у випадку успішного завершення замовлення;

– повернення замовнику у випадку спору чи скасування.

Транзакції для функцій блокування записів та розділення балансу на доступний і зарезервований застосовані для покращення надійності системи та відповідність вимогам фінансових вебзастосунків.

4.2 Тестування системи

У розробленому вебзастосунку реалізовано комплексне тестування, що охоплює як модульний (Unit), так і функціональний (Feature) рівні. Такий підхід дозволяє перевірити як окремі компоненти системи, так і їхню взаємодію в межах бізнес-логіки й вважається найкращим, бо кожен закриває свою частину проблем [15].

Модульні тести (Unit-тести) спрямовані на перевірку внутрішньої логіки моделей та їхніх зв’язків. Зокрема, проведено тести:

– коректності зв’язків між сутностями (користувач-роль, користувач-баланс, замовлення-статус, транзакція-тип тощо);

– роботи аксесорів атрибутів (наприклад, для аватара користувача);

– бізнес-правил, такі як неможливість від’ємного балансу;

– коректності обчислень (наприклад, розрахунок комісії);

– роботи локальних запитів (scopes) для фільтрації замовлень.

Ці тести забезпечують надійність базових компонентів системи та правильність роботи доменної логіки.

Функціональні тести (Feature-тести) перевіряють поведінку системи з точки зору користувача та охоплюють основні бізнес-процеси:

– робота із замовленнями (OrderController): перевіряється контроль доступу, створення замовлень, валідація даних, редагування, скасування, завершення та управління термінами виконання;

– фінансові операції (TransactionController): тестуються процеси поповнення та виведення коштів, включаючи перевірки валідності даних, належності платіжних реквізитів та достатності балансу;

– керування профілем (ProfileController): перевіряється оновлення аватара та пароля, а також відповідна валідація даних (розмір файлу, довжина пароля тощо);

– механізм депонування (Escrow): окремо протестовано критично важливу бізнес-логіку блокування, вивільнення та повернення коштів залежно від стану замовлення, а також зміну суми депонування при редагуванні бюджету;

– базовий тест (ExampleTest): підтверджує доступність головної сторінки та коректну відповідь сервера.

Приклад PHP Unit тесту, який перевіряє логіку оновлення балансу користувача при збільшенні бюджету замовлення:

```
public function test_escrow_adjusts_on_budget_increase(): void {
        $customer = User::factory()->create(['user_role_id' => 2]);
        $balance = Balance::create([
            'user_id' => $customer->id,
            'amount' => 1000,
            'escrowed_amount' => 500,]);
        $order = Order::create([
            'title' => 'Edit Test',
            'short_description' => 'Test short desc',
            'full_description' => 'Test full desc',
            'requirement_skills' => 'PHP, Laravel',
            'customer_id' => $customer->id,
            'budget' => 500,
            'deadline_in_days' => 10,]);
        $difference = 100;
        $balance->decrement('amount', $difference);
        $balance->increment('escrowed_amount', $difference);
        $balance->refresh();
        $this->assertEquals(900, $balance->amount); // 1000 - 100
        $this->assertEquals(600, $balance->escrowed_amount);// 500+100
}
```

Після розробки тестів виконано повне тестування застосунку та перевірено їх успішність (рис. 4.2):

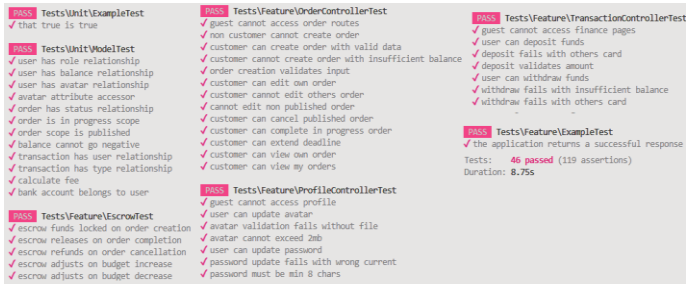


Рисунок 4.2 – Тестування застосунку

Опис кожного тесту поданий у **додатку Б (табл. Б.7)**.

4.3 Керівництво користувача

Робота з вебсервісом кожного нового користувача починається з реєстрації. Для реєстрації створено окрему сторінку з формою, яка має поля: ім’я, електронна адреса, пароль та роль (виконавець або замовник) (рис. 4.3).

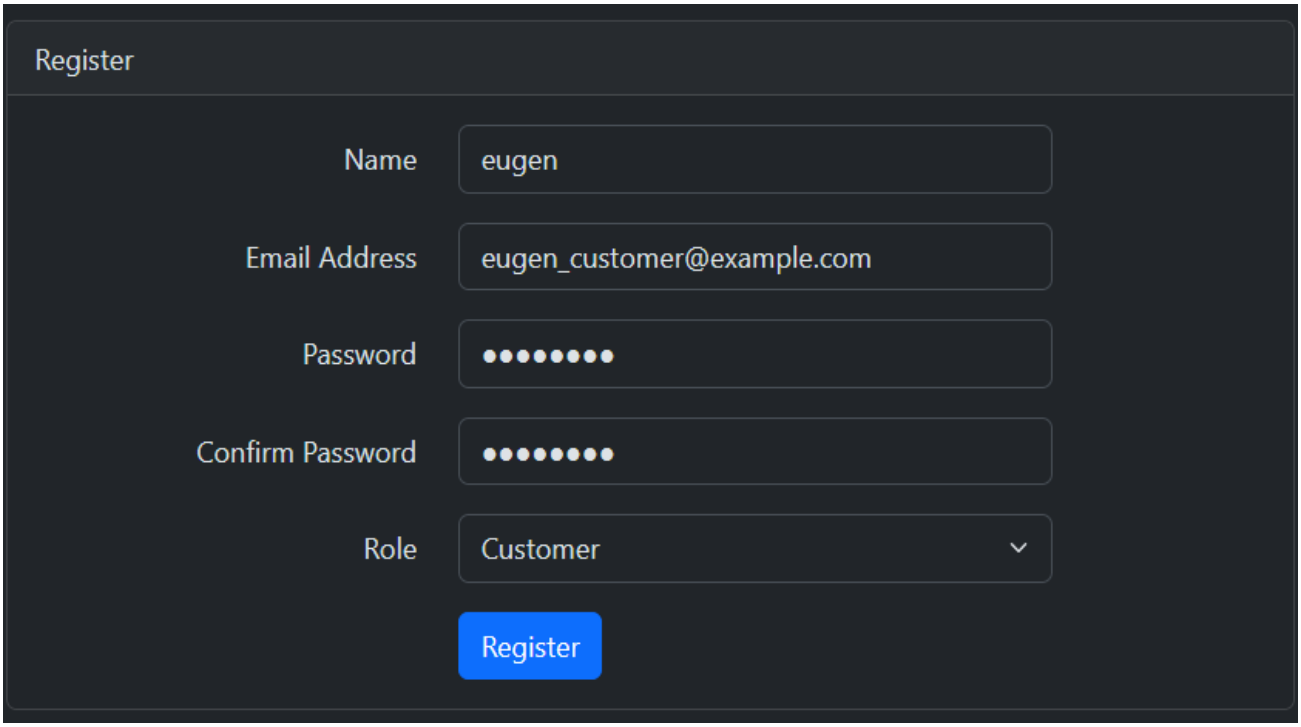


Рисунок 4.3 – Форма реєстрації

Для авторизації створено сторінку з формою, яка має поле вводу для електронної адреси та пароля (рис. 4.4).

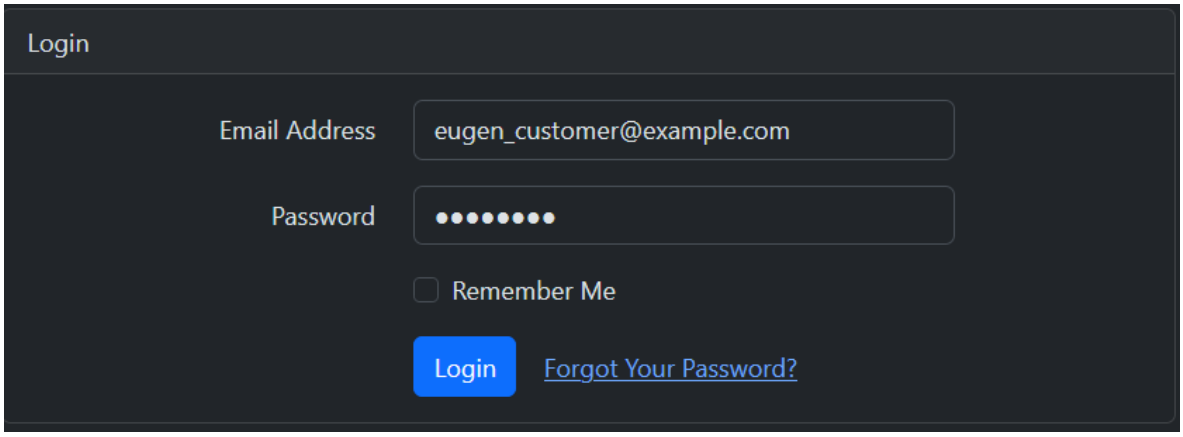


Рисунок 4.4 – Форма авторизації

Після авторизації користувач потрапляє на головну сторінку з переліком доступних замовлень, які можна сортувати за категоріями, датою, бюджетом (рис. 4.5). В залежності від ролі користувача у верхній частині сторінки з’являються відповідно до ролі пункти меню: замовлення (головна сторінка), мої замовлення, чати, створити замовлення (тільки для виконавця), мій профіль, ціни, особливості сервісу та нотифікації. Також відображається кнопка деавторизації.

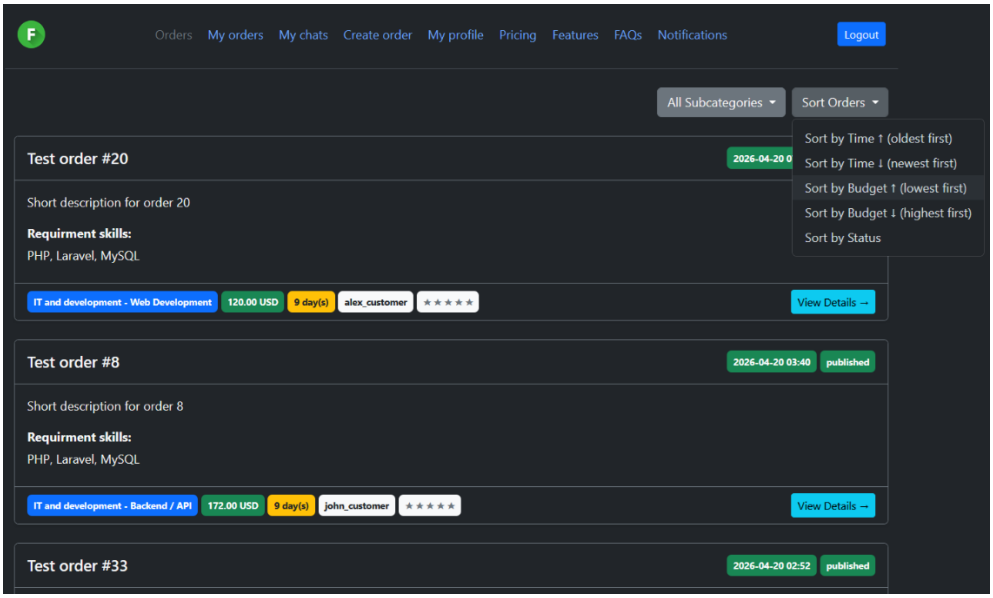


Рисунок 4.5 – Форма авторизації

Сторінка повного перегляду замовлення містить: повний опис, файлові вкладення, необхідні навички, категорію, бюджет, дедлайн, посилання на замовники та його рейтинг (рис. 4.6). Окрім цього, присутня кнопку для створення скарги.

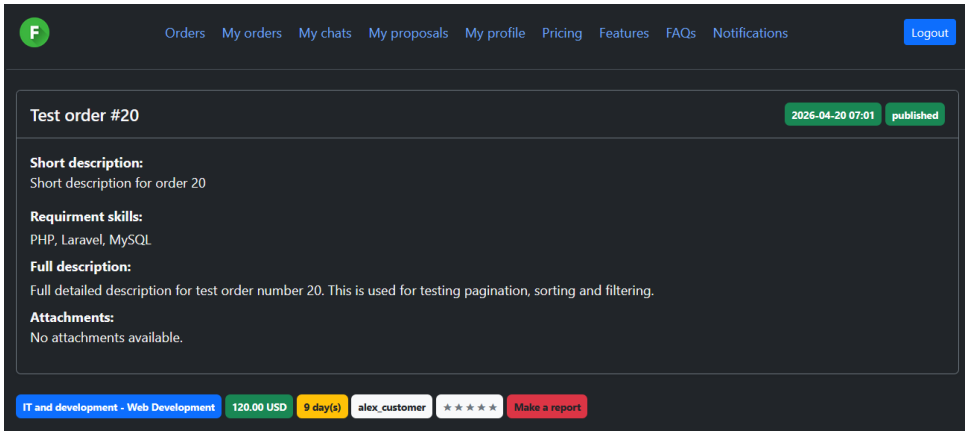


Рисунок 4.6 – Сторінка з замовленням

У нижній частині сторінки з замовленням виконавці мають змогу запропонувати свої послуги, бюджет та дедлайн за допомогою форми та переглянути пропозиції конкурентів (рис. 4.7).

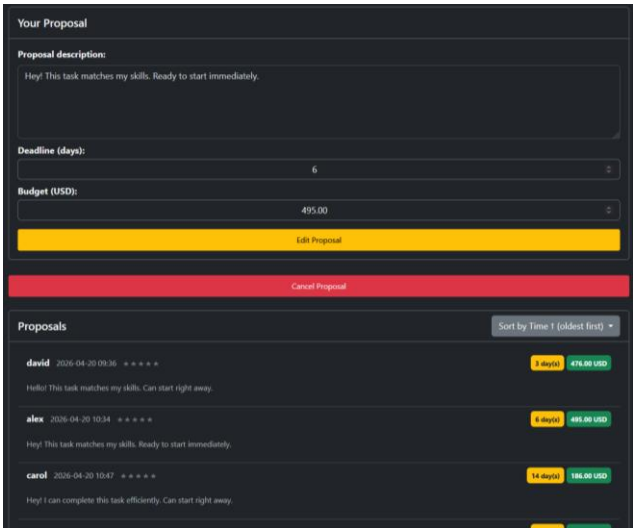


Рисунок 4.7 – Форма для пропозиції послуги від виконавця

З точки зору замовника перегляд сторінки з замовленням виглядає інакше. Користувач має змогу змінити: опис, категорію, бюджет, дедлайн, необхідні навички за допомогою форми редагування, додати нові файлові вкладення або скасувати замовлення (рис. 4.8).

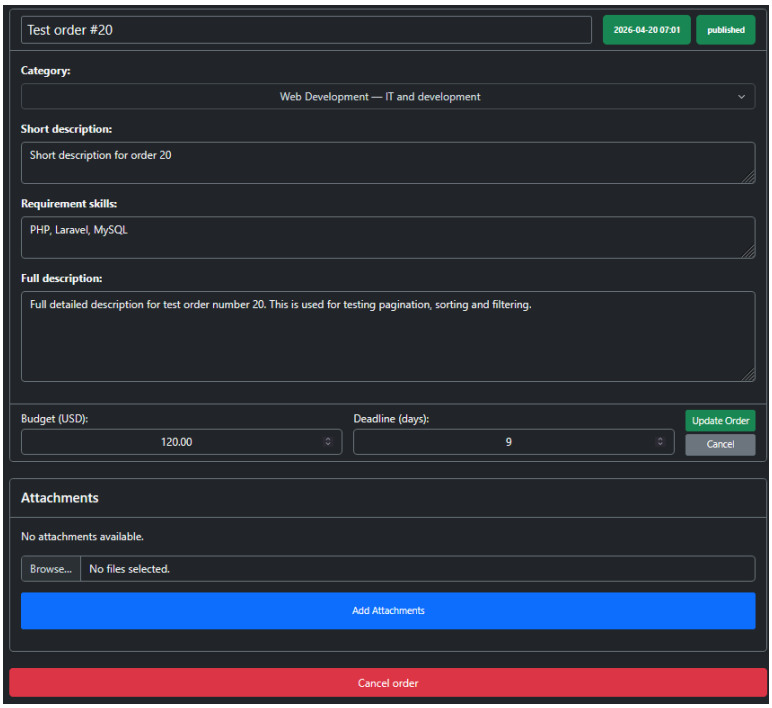


Рисунок 4.8 – Форма для пропозиції послуги від виконавця

Перегляд пропозицій для замовника також виглядає інакше: доступні кнопки для створення діалогу в чаті та кнопка для підтвердження пропозиції (рис. 4.9).

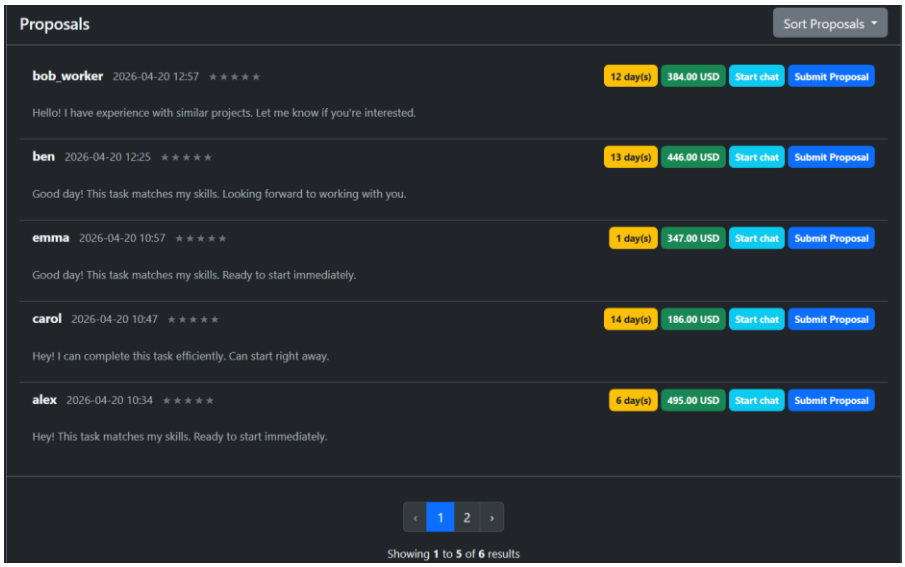


Рисунок 4.9 – Пропозиції для замовника

Сторінка з чатами містить в собі у лівій частині перелік створених чатів відсортованих за датою отримання останнього повідомлення та активний чат з формою у нижній частині за допомогою якої можна залишити нове повідомлення (рис. 4.10).

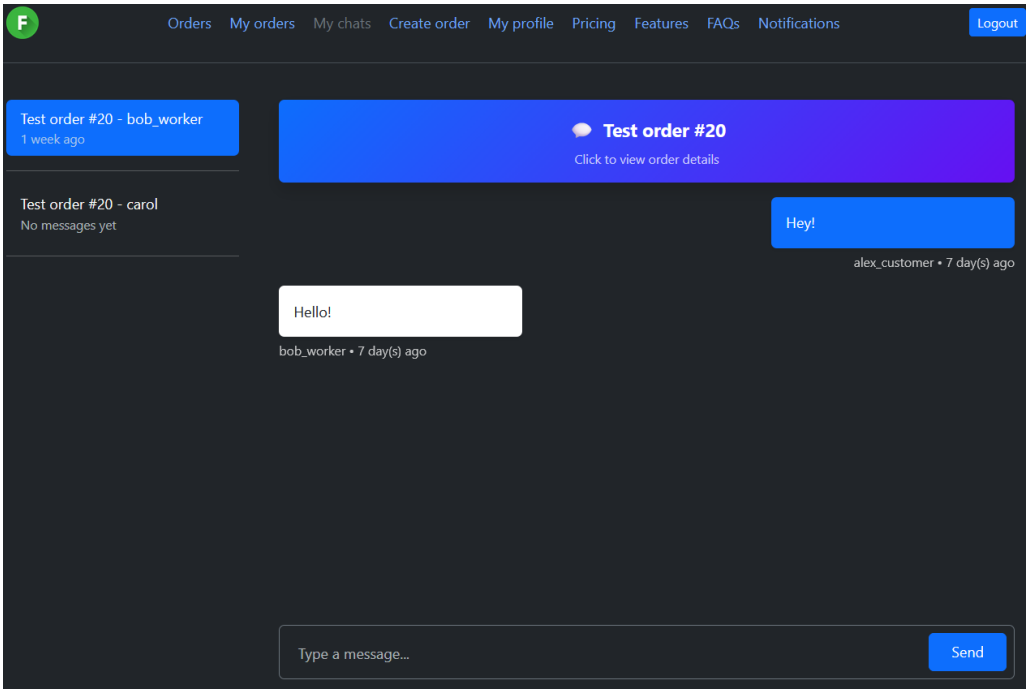


Рисунок 4.10 – Сторінка з чатами

Сторінка створення замовлення надає форму на якій можна вказати категорію, заголовок, короткий опис, необхідні навички, повний опис, бюджет, дедлайн та прикріпити файлові додатки (рис. 4.11).

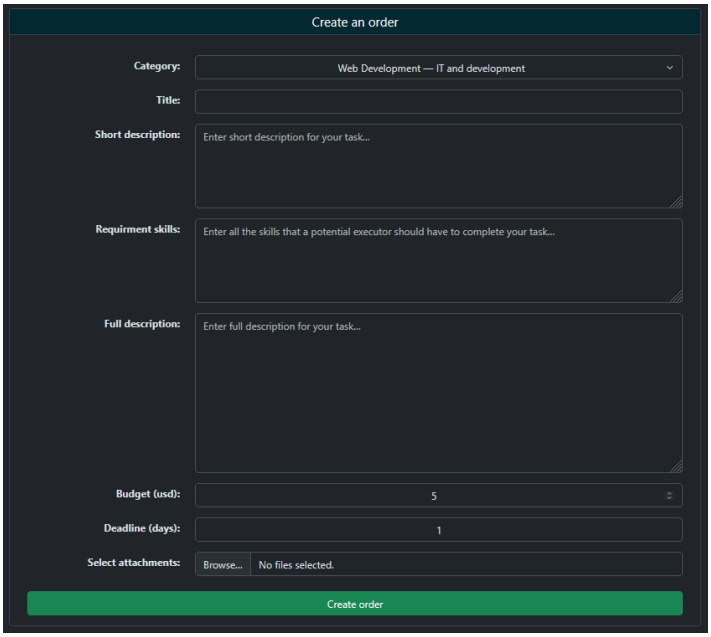


Рисунок 4.11 – Сторінка створення замовлення

Після вибору виконавця замовлення змінює свій статус на «in progress». Сторінка з замовленням демонструє форму для обміну повідомлень у порядку виконання та дозволяє замовнику розширити дедлайн, підтвердити виконання або відмінити замовлення (якщо, воно вийшло за межі дедлайну) (рис. 4.12).

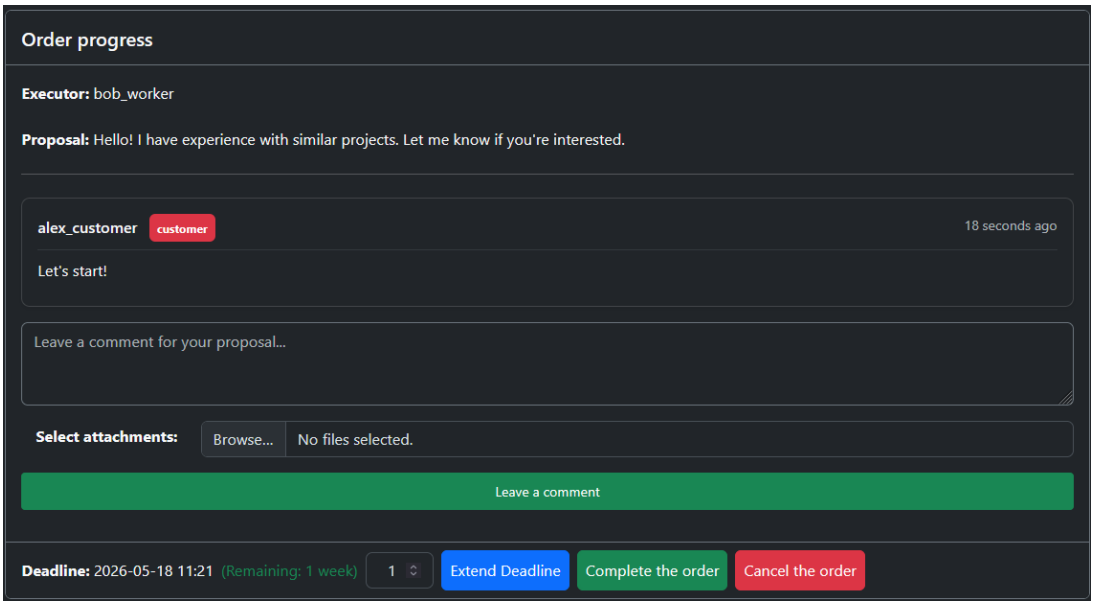


Рисунок 4.12 – Сторінка створення замовлення

На сторінці виконаного замовлення для виконавця та замовника надається форма для відгуку Відгук можна видалити та залишити знову (рис. 4.13).

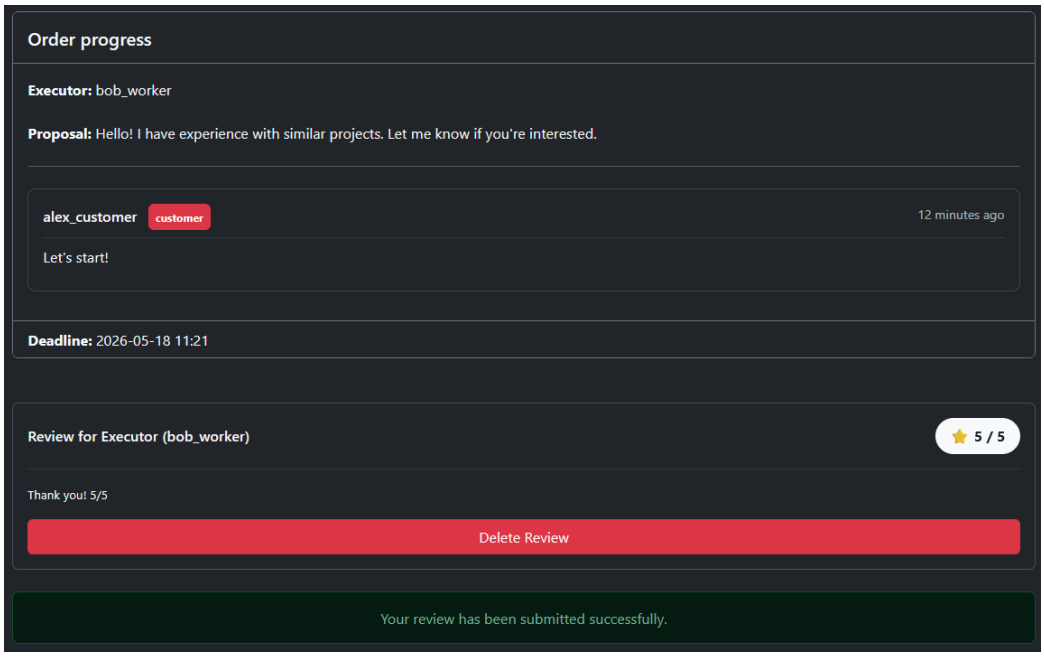


Рисунок 4.13 – Виконане замовлення та відгук

Користувачі мають публічну сторінку профілю та приватну. На публічній сторінці можна переглянути ім’я, роль, середній рейтинг та перелік відгуків (рис. 4.14).

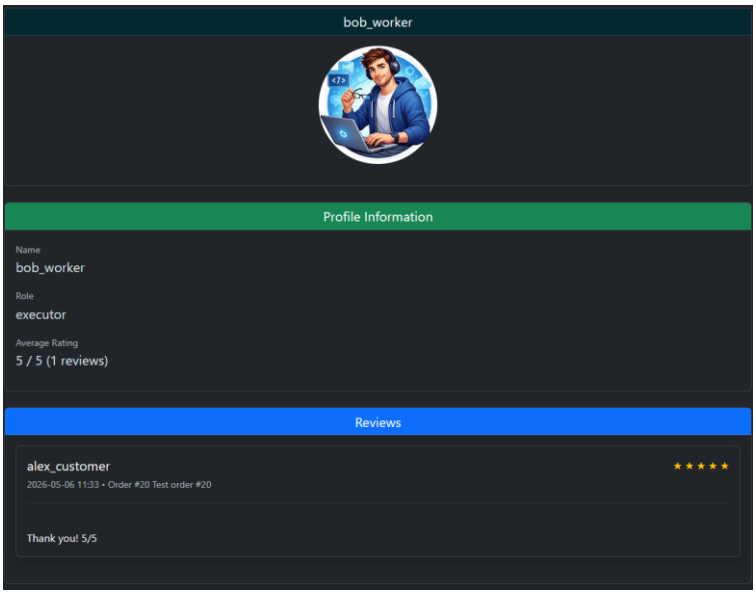


Рисунок 4.14 – Публічна сторінка профілю користувача

Приватна сторінка профілю надає користувачу можливість змінити аватар та пароль, а також переглянути поточний баланс. У лівій частині розташовані посилання на інші частини налаштування та фінансову інформацію користувача: банківські картки, історія фінансових транзакцій, сторінку для поповнення або виводу коштів, відгуки та скарги (рис. 4.15).

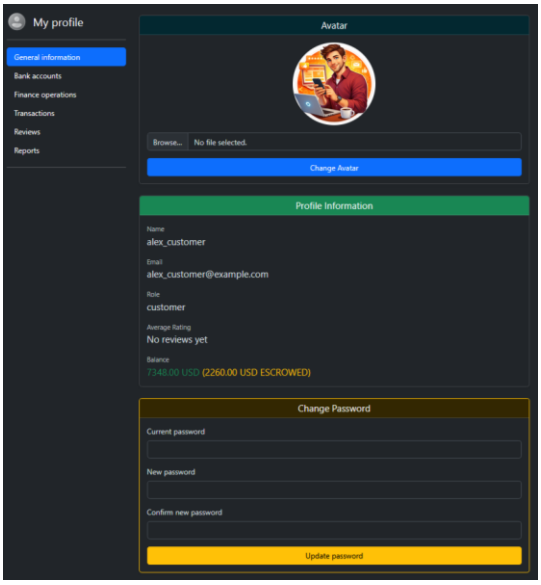


Рисунок 4.15 – Приватна сторінка профілю користувача

На сторінці з банківськими рахунками користувач має можливість прив’язати або видалити картку (рис. 4.16).

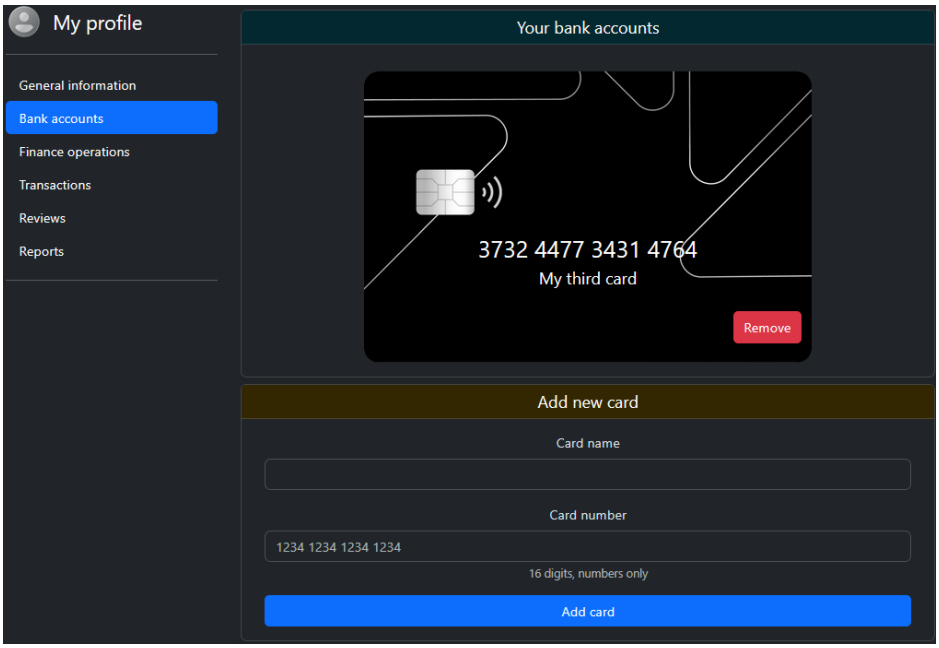


Рисунок 4.16 – Сторінка з банківськими рахунками

На сторінці для фінансових операцій користувач має можливість поповнити або вивести кошти (рис. 4.17).

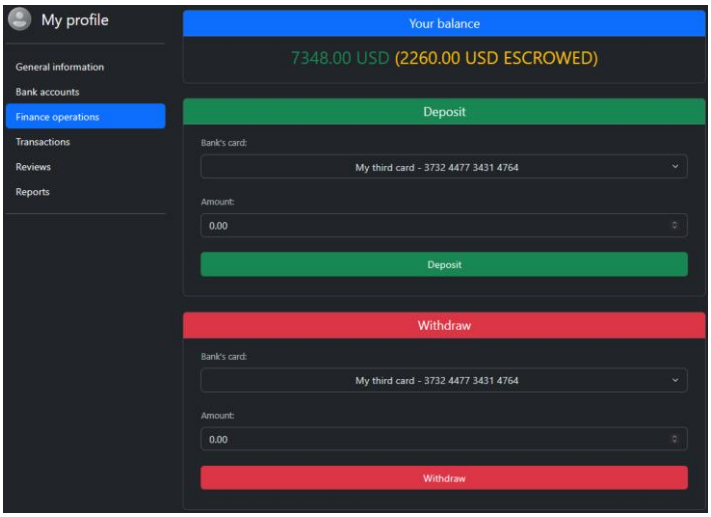
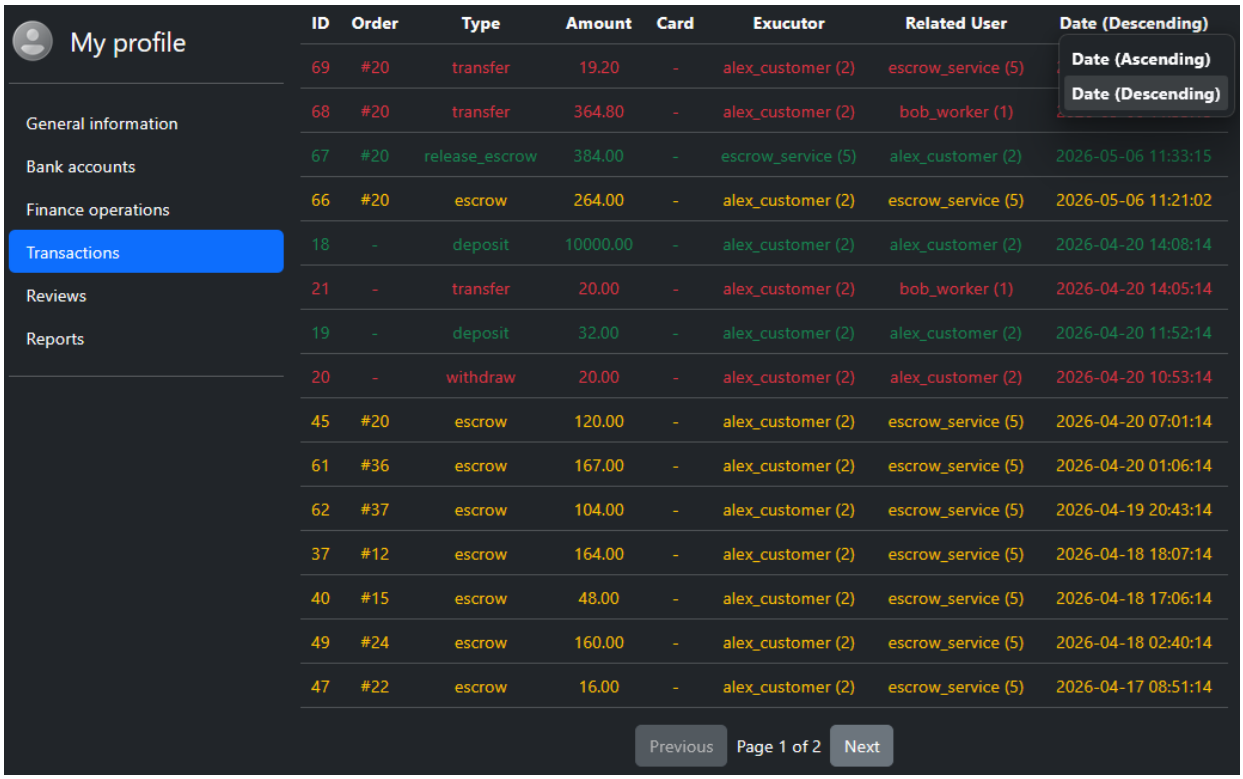


Рисунок 4.17 – Сторінка вводу та виводу коштів

На сторінці для транзакцій користувач отримує історію фінансових операцій до яких він має відношення (рис. 4.18). Присутня можливість сортування за датою.



ID	Order	Type	Amount	Card	Exucutor	Related User	Date (Descending)
69	#20	transfer	19.20	-	alex_customer (2)	escrow_service (5)	
68	#20	transfer	364.80	-	alex_customer (2)	bob_worker (1)	
67	#20	release_escrow	384.00	-	escrow_service (5)	alex_customer (2)	2026-05-06 11:33:15
66	#20	escrow	264.00	-	alex_customer (2)	escrow_service (5)	2026-05-06 11:21:02
18	-	deposit	10000.00	-	alex_customer (2)	alex_customer (2)	2026-04-20 14:08:14
21	-	transfer	20.00	-	alex_customer (2)	bob_worker (1)	2026-04-20 14:05:14
19	-	deposit	32.00	-	alex_customer (2)	alex_customer (2)	2026-04-20 11:52:14
20	-	withdraw	20.00	-	alex_customer (2)	alex_customer (2)	2026-04-20 10:53:14
45	#20	escrow	120.00	-	alex_customer (2)	escrow_service (5)	2026-04-20 07:01:14
61	#36	escrow	167.00	-	alex_customer (2)	escrow_service (5)	2026-04-20 01:06:14
62	#37	escrow	104.00	-	alex_customer (2)	escrow_service (5)	2026-04-19 20:43:14
37	#12	escrow	164.00	-	alex_customer (2)	escrow_service (5)	2026-04-18 18:07:14
40	#15	escrow	48.00	-	alex_customer (2)	escrow_service (5)	2026-04-18 17:06:14
49	#24	escrow	160.00	-	alex_customer (2)	escrow_service (5)	2026-04-18 02:40:14
47	#22	escrow	16.00	-	alex_customer (2)	escrow_service (5)	2026-04-17 08:51:14

Рисунок 4.18 – Сторінка з історією транзакцій

Окремо існує для приватного перегляду сторінка з отриманими відгуками (рис. 4.19). Кожен відгук містить в собі коментар та оцінку від одного до п’яти (зірки в правому верхньому куту відгука).

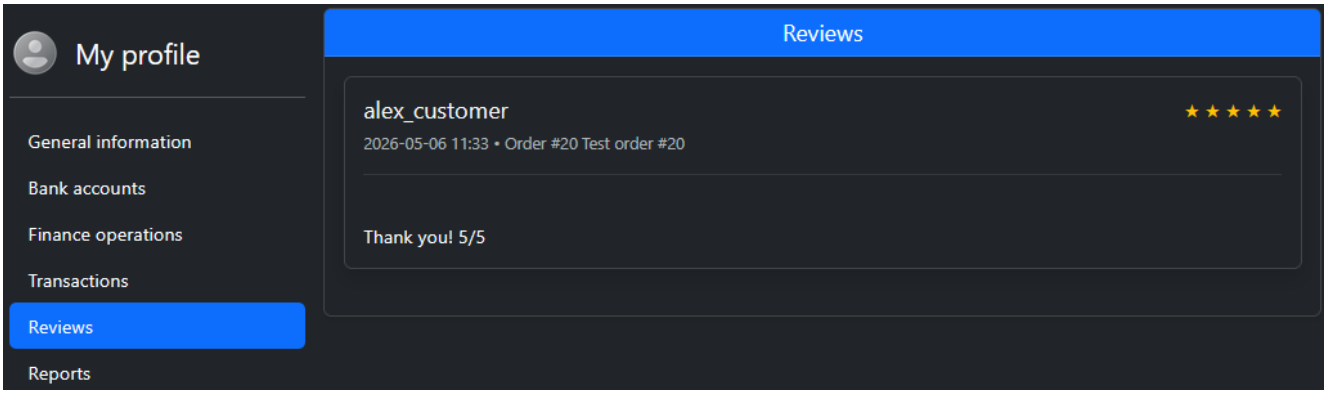


Рисунок 4.19 – Сторінка з отриманими відгуками

Для створення скарги використовується сторінка з формою в якій вказується заголовок та детальний опис проблеми (рис. 4.20).

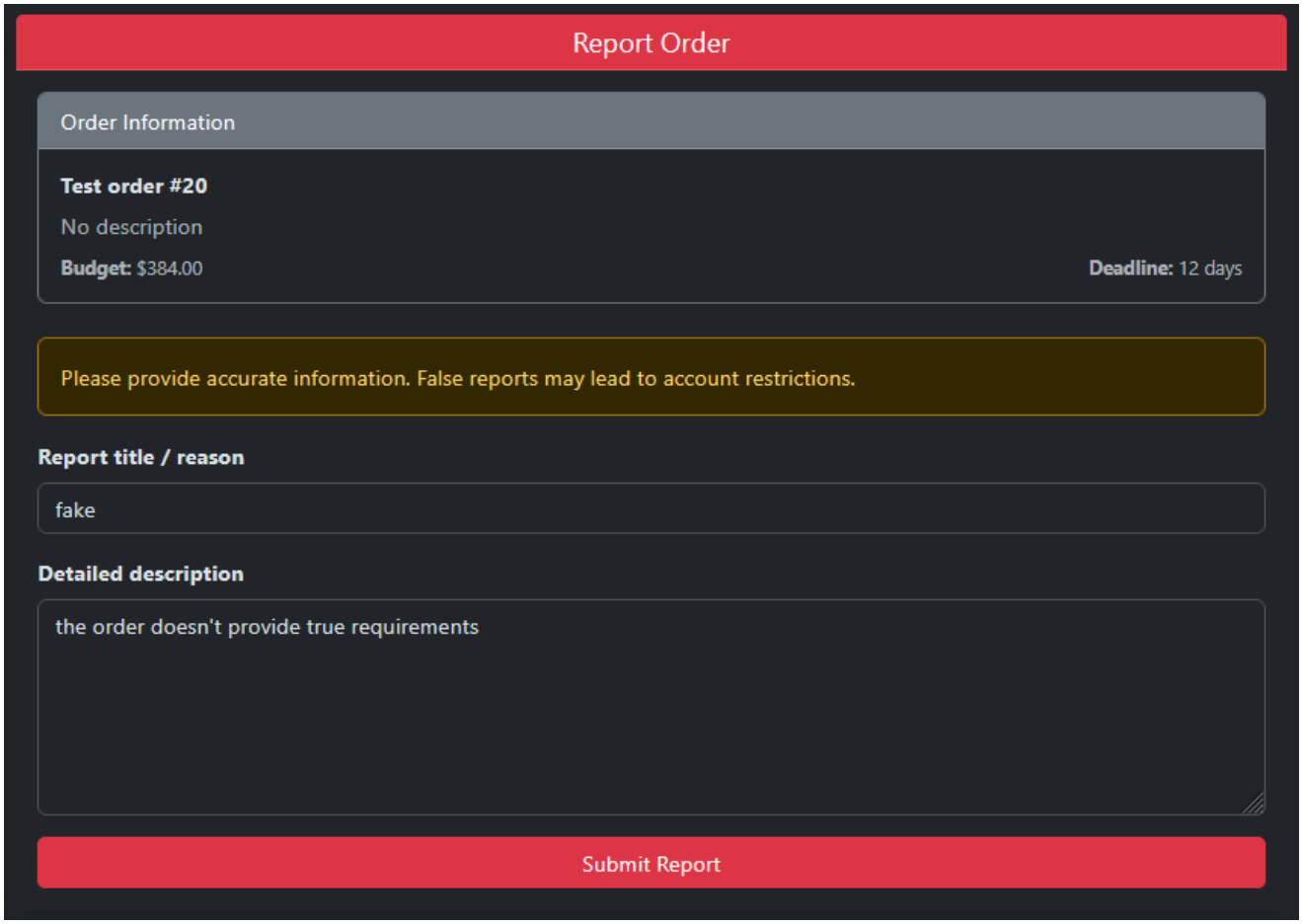


Рисунок 4.20 – Сторінка для створення скарги

Створені скарги та процес їх вирішення можна переглянути у профілі в розділі «скарги» (рис. 4.21). Кожна скарга виводиться окремо та має короткий опис, статус та посилання на замовлення до якого вона відноситься.

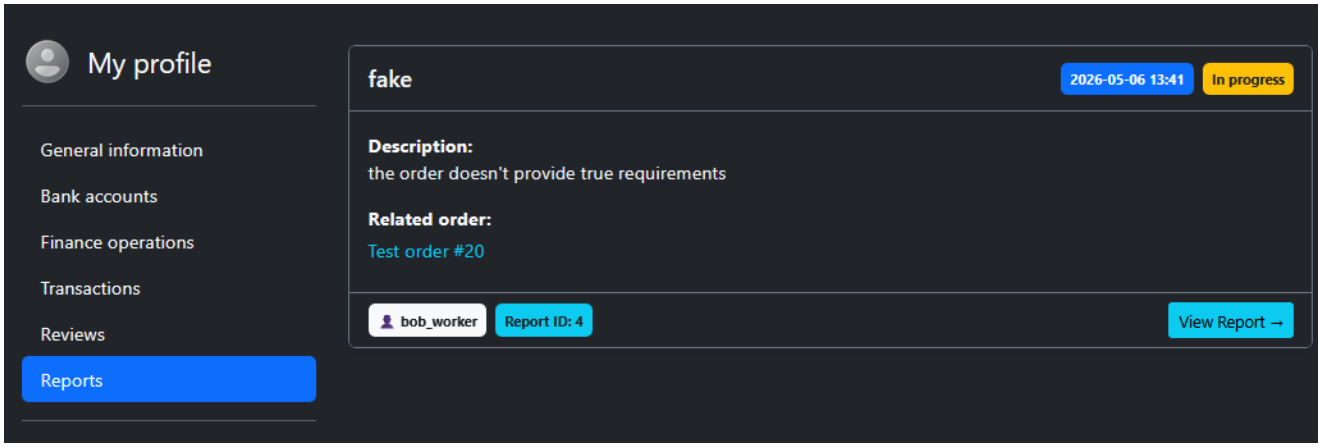


Рисунок 4.21 – Сторінка зі скаргами користувача

Кожна скарга має окрему сторінку де можна вести діалог з адміністратором та переглядати детальний процес її обробки (рис. 4.22). Присутня форма для створення коментарів.

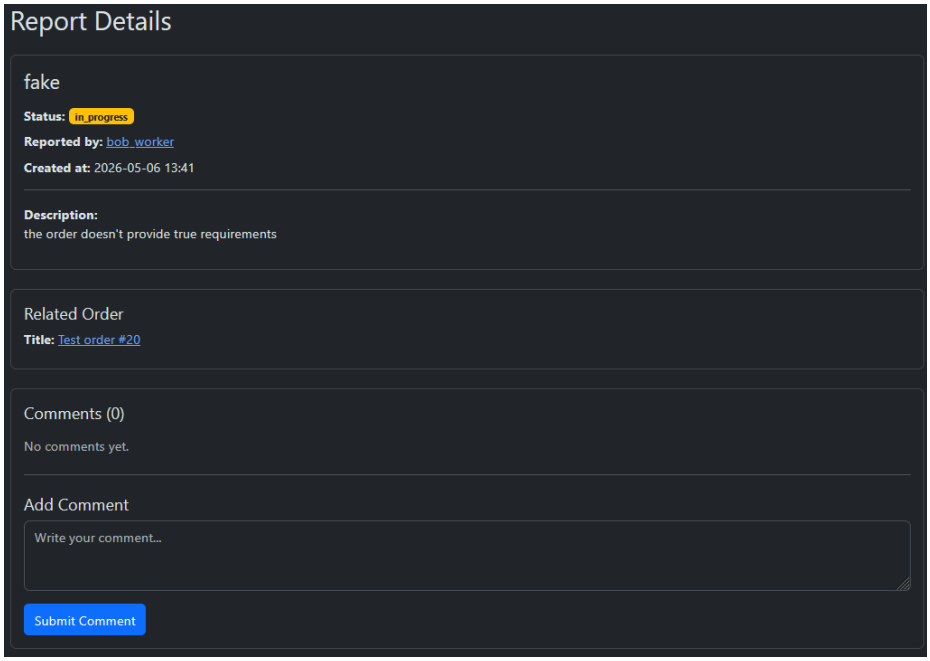


Рисунок 4.22 – Сторінка окремої скарги

На сторінці сповіщень користувачеві надається можливість переглядати нові повідомлення, що його стосуються. Наприклад, сповіщення про отримання відгуку

або про вибір його, як кандидата на виконання завдання (рис. 4.23). Кожне нове сповіщення, яке ще не переглянуто позначається надписом «New».

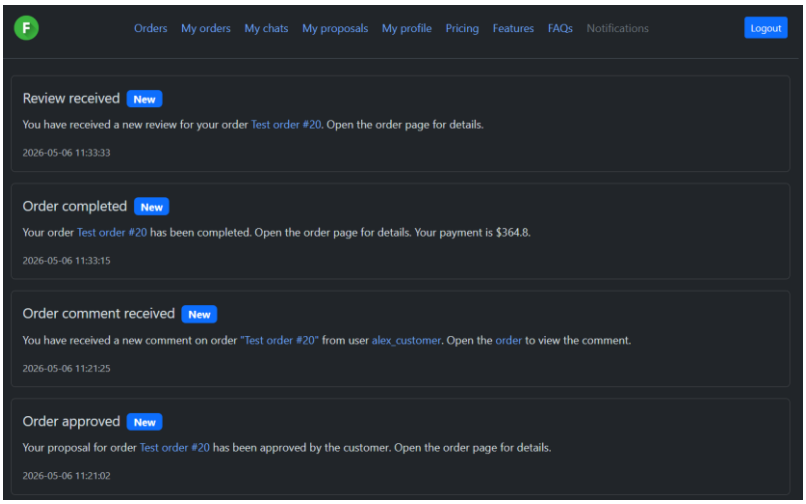


Рисунок 4.23 – Сторінка сповіщень

Для виконавців існує окремо сторінка з замовленнями до яких користувач залишив свої пропозиції на виконання. Всі оголошення можна сортувати та переглядати окремо (рис. 4.24).

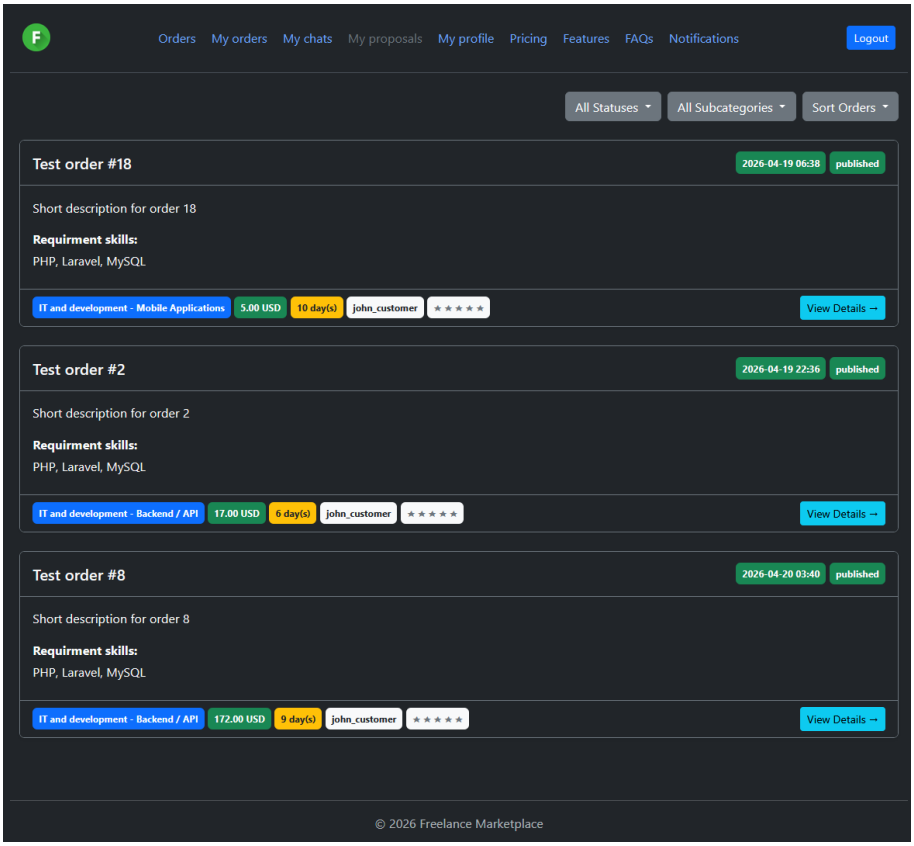


Рисунок 4.24 – Сторінка оголошень з пропозиціями від користувача

Отже, всі основні сторінки вебзастосунку, детально описані вище та успішно реалізовані у фінальній версії проєкту. Керівництво охоплює повний шлях користувача – від реєстрації та входу до створення замовлення та отримання відгуку після його виконання. Розглянуто роботу з сторінками, які стосуються фінансів та комунікації між користувачами. Також, надано інформацію про створення скарг та перегляд сповіщень.

Висновки до розділу 4

Продемонстровано основні аспекти програмної реалізації вебзастосунку фріланс-маркетплейсу на основі архітектури MVC та фреймворку Laravel. Виконано чітке розподілення відповідальності між компонентами системи. Розмежовано доступ до функціоналу для замовника, виконавця та адміністратора.

Використано міграції, сідери та ORM Eloquent для роботи з даними у спрощеному вигляді. Створено зв’язки між сутностями та продемонстровано їх приклад.

Створено та описано функціонал для депонування коштів (escrow): на основі транзакцій і блокування записів для гарантування наявності коштів, що призначені виконавцю після успішного завершення проєкту.

Впроваджено валідацію даних, обчислювані атрибути та повторно використовувані компоненти інтерфейсу та продемонстровано їх приклади.

Виконано тестування модульного та функціонального типу. Підтверджено валідність роботи бізнес-логіки платформи.

Охоплено основні сценарії роботи із застосунком та розроблено керівництво користувача з детальним описом.

Отже, у четвертому розділі представлено ключові компоненти реалізації вебзастосунку та враховано сучасні підходи до розробки, безпеки та тестування.

ВИСНОВКИ

У ході виконання кваліфікаційної бакалаврської роботи досягнуто поставлену мету, а саме: розроблено вебзастосунок взаємодії фрілансерів та замовників у вигляді фріланс-маркетплейсу.

Для досягнення мети виконано такі завдання:

– проведено аналіз предметної області та існуючих аналогів фріланс-маркетплейсів, визначено їх переваги та недоліки;

– визначено функціональні та нефункціональні вимоги до системи;

– обґрунтовано вибір архітектурного підходу та технологічного стеку розробки;

– спроєктовано архітектуру вебзастосунку та структуру бази даних;

– реалізовано функціонал реєстрації та автентифікації користувачів;

– розроблено функціонал створення, публікації та пошуку проєктів;

– реалізовано систему подання заявок та управління замовленнями;

– забезпечено внутрішню систему повідомлень для комунікації між користувачами;

– проведено тестування розробленої системи та оцінено результати її роботи.

У процесі виконання роботи було проаналізовано предметну область фріланс-ринку, визначено основні проблеми взаємодії між замовниками та виконавцями, а також досліджено існуючі програмні рішення. На основі отриманих результатів сформовано вимоги до системи, що розробляється, з урахуванням сучасних тенденцій розвитку вебтехнологій.

Аргументовано вибір стеку для реалізації застосунку, зокрема використання фреймворку Laravel, СУБД PostgreSQL, Blade, Bootstrap, PHPUnit та мови програмування PHP. Проведено моделювання системи із застосуванням UML-діаграм, визначено основні сценарії використання та структуру взаємодії компонентів.

У межах проєктування розроблено архітектуру вебзастосунку, структуру бази даних та користувацькі інтерфейси. Під час реалізації створено серверну та клієнтську частини системи, реалізовано основні функціональні модулі, зокрема управління проєктами, систему заявок, внутрішній чат, депонування, а також рейтинг та відгуки.

Проведено тестування, яке підтвердило працездатність та коректність функціонування розробленої системи. Також було складено керівництво користувача для ефективного використання вебзастосунку.

Результатом виконаної роботи є повнофункціональний вебзастосунок взаємодії фрілансерів та замовників, який забезпечує автоматизацію процесів пошуку проєктів, комунікації між користувачами та управління виконанням робіт, а також сприяє підвищенню зручності, прозорості та ефективності взаємодії у сфері дистанційної зайнятості.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1) About Freelance.ua. Freelance.ua. URL: https://freelance.ua/ (Accessed: 20.04.2026)

2) About Freelancer.com. Freelancer.com. URL: https://www.freelancer.com/about/ (Accessed: 20.04.2026)

3) About Fiverr.com. Fiverr.com. URL: https://www.fiverr.com/about-us (Accessed: 20.04.2026)

4) Apiag C. P. W., Cadiz E. B. S., Lincopinis D. R. A Review on PHP Programming Language. pp. 1–7. URL: https://www.researchgate.net/publication/371166635_A_Review_on_PHP_Programming_Language (Accessed: 23.04.2026)

5) Laravel, Symfony, Yii, CodeIgniter, CakePHP. Google Trends. URL: https://trends.google.com/trends/explore?q=Laravel,Symfony,CodeIgniter,Yii,CakePHP&date=today%205-y (Accessed: 23.04.2026)

6) Węgrzecki K. S., Dzieńkowski M. Performance analysis of Laravel and Yii2 frameworks based on the MVC architectural pattern and PHP language. Journal of Computer Sciences Institute. 2022. Vol. 24. pp. 265–272. doi: 10.35784/jcsi.3002

7) Khan W., Kumar T., Zhang C., Raj K., Roy A. M., Luo B. SQL and NoSQL database software architecture performance analysis and assessments: A systematic literature review. 2023. Vol. 7, No. 2. pp. 1–44. doi: 10.3390/bdcc7020097

8) Urnikienė J., Steponavičienė V., Atanasov S. Comparative Read Performance Analysis of PostgreSQL and MongoDB in E-Commerce: An Empirical Study of Filtering and Analytical Queries. 2026. Vol. 10, No. 2. pp. 1–23. doi: 10.3390/bdcc10020066

9) Santoso M. F. Perbandingan Efektivitas Bootstrap dan Tailwind CSS dalam Pengembangan UI Web Responsif. 2025. Vol. 7, No. 4. pp. 489–497. doi: 10.47233/jteksis.v7i4.2260

10) Woś A., Pańczyk B. Comparison of selected view creation technologies in applications using the Laravel framework. 2021. Vol. 20. pp. 175–182. doi: 10.35784/jcsi.2674

11) Nguyen L. A. T., Huynh T. S., Tran D. T., Vu Q. H. Design and Implementation of Web Application Based on MVC Laravel Architecture. 2022. Vol. 6(4). pp. 23–29. doi: 10.24018/ejece.2022.6.4.448

12) Pop D.-P., Altar A. Designing an MVC Model for Rapid Web Application Development. 2014. Vol. 69. pp. 1172–1179. doi: 10.1016/j.proeng.2014.03.106

13) Hidayati A. T., Widyantoro A. E., Ramadhani H. J. Perancangan Sistem Informasi Wirausaha Mahasiswa (Siwirma) Berbasis Web dengan Unified Modeling Language (UML). 2023. Vol. 2(4). pp. 86–107. doi: 10.55606/juprit.v2i4.2906

14) Planas E., Cabot J. How are UML class diagrams built in practice? A usability study of two UML tools: MagicDraw and Papyrus. 2020. Vol. 67. Article 103363. doi: 10.1016/j.csi.2019.103363

15) Superson W., Smyk T., Plechawska-Wójcik M. Comparative Analysis of Methods for Testing Web Applications. 2023. Vol. 28. pp. 1–10. doi: 10.35784/jcsi.3697

ДОДАТОК А

UML діаграми

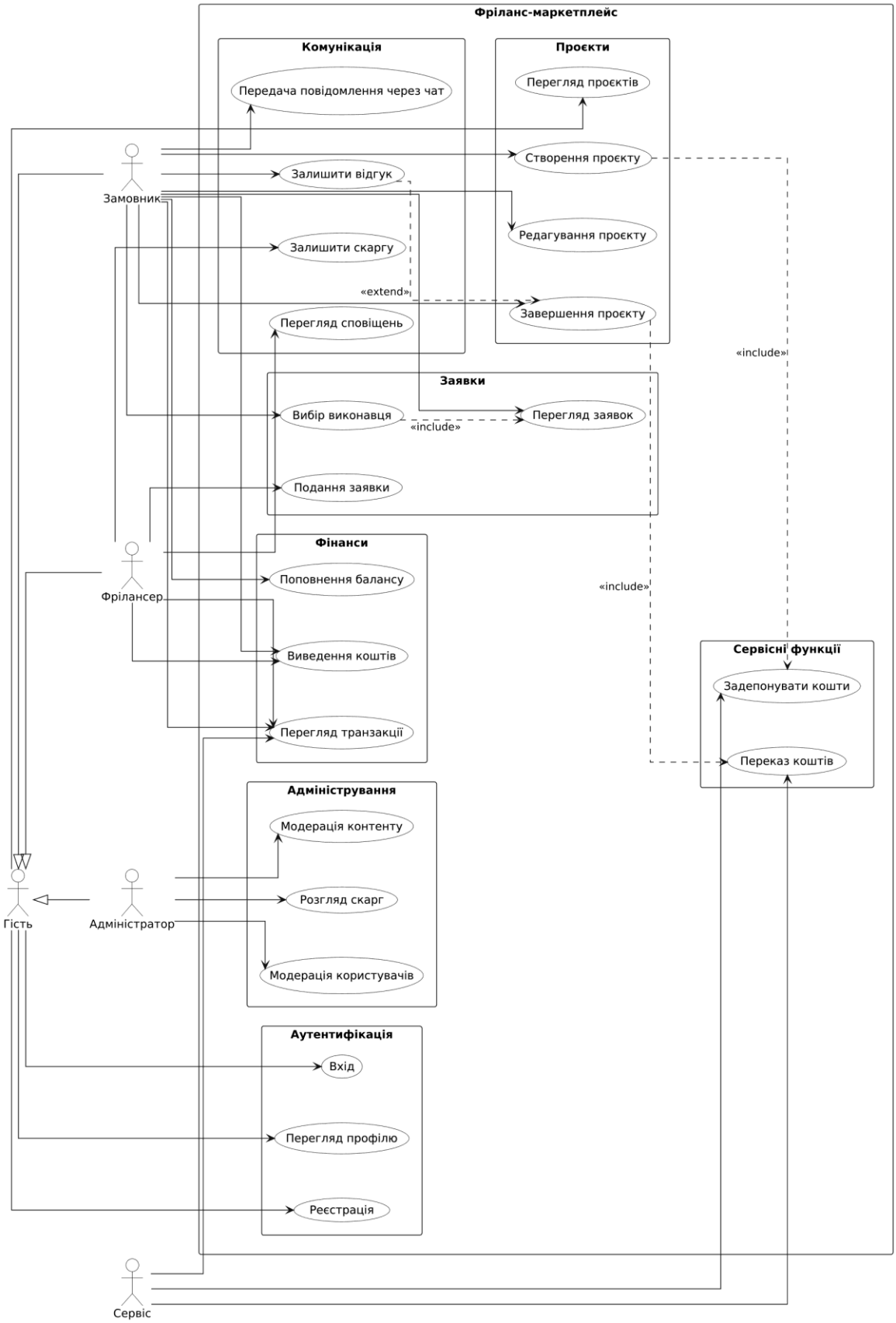


Рисунок А.1 – Діаграма використання

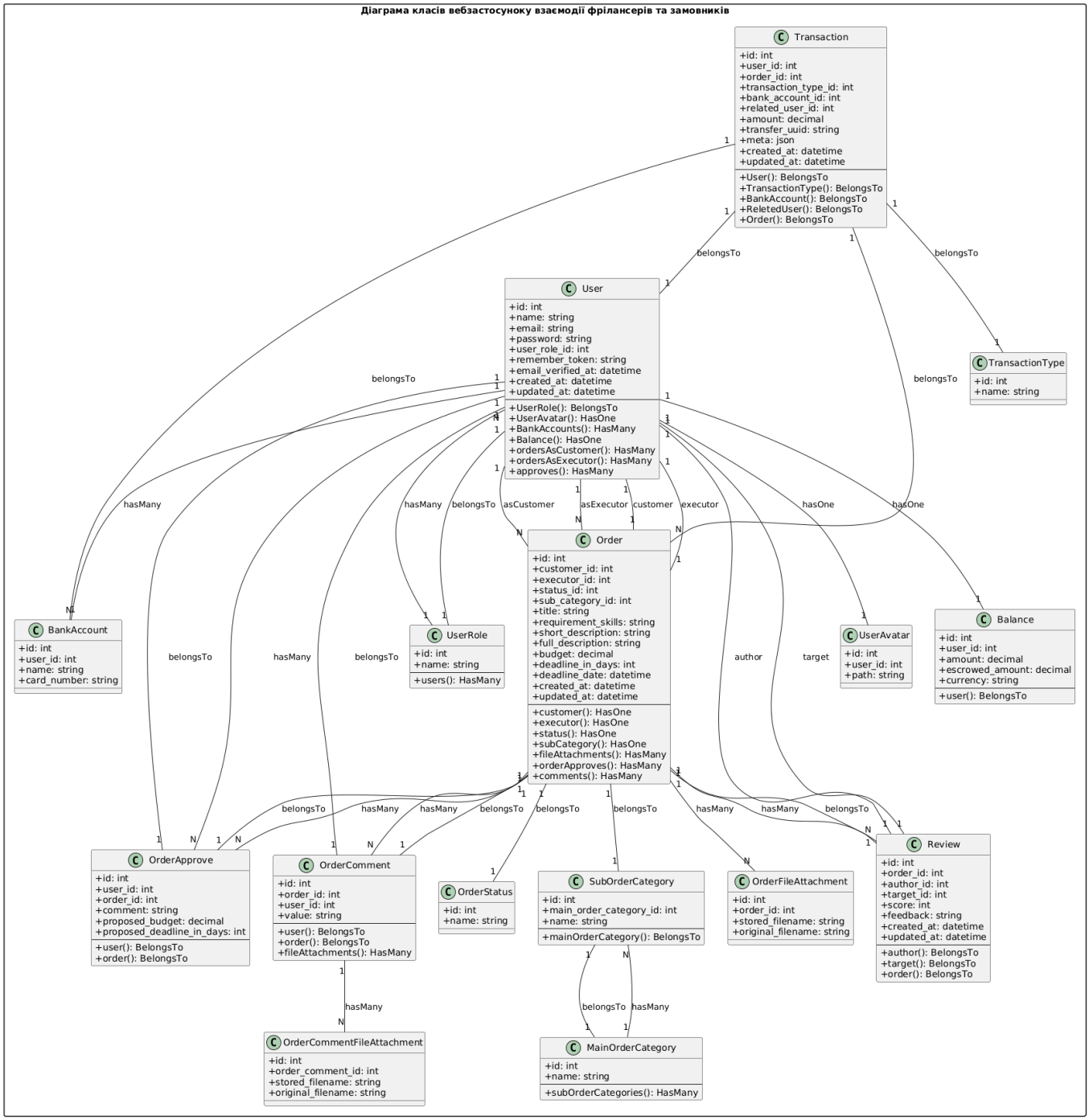


Рисунок А.2 – Діаграма класів

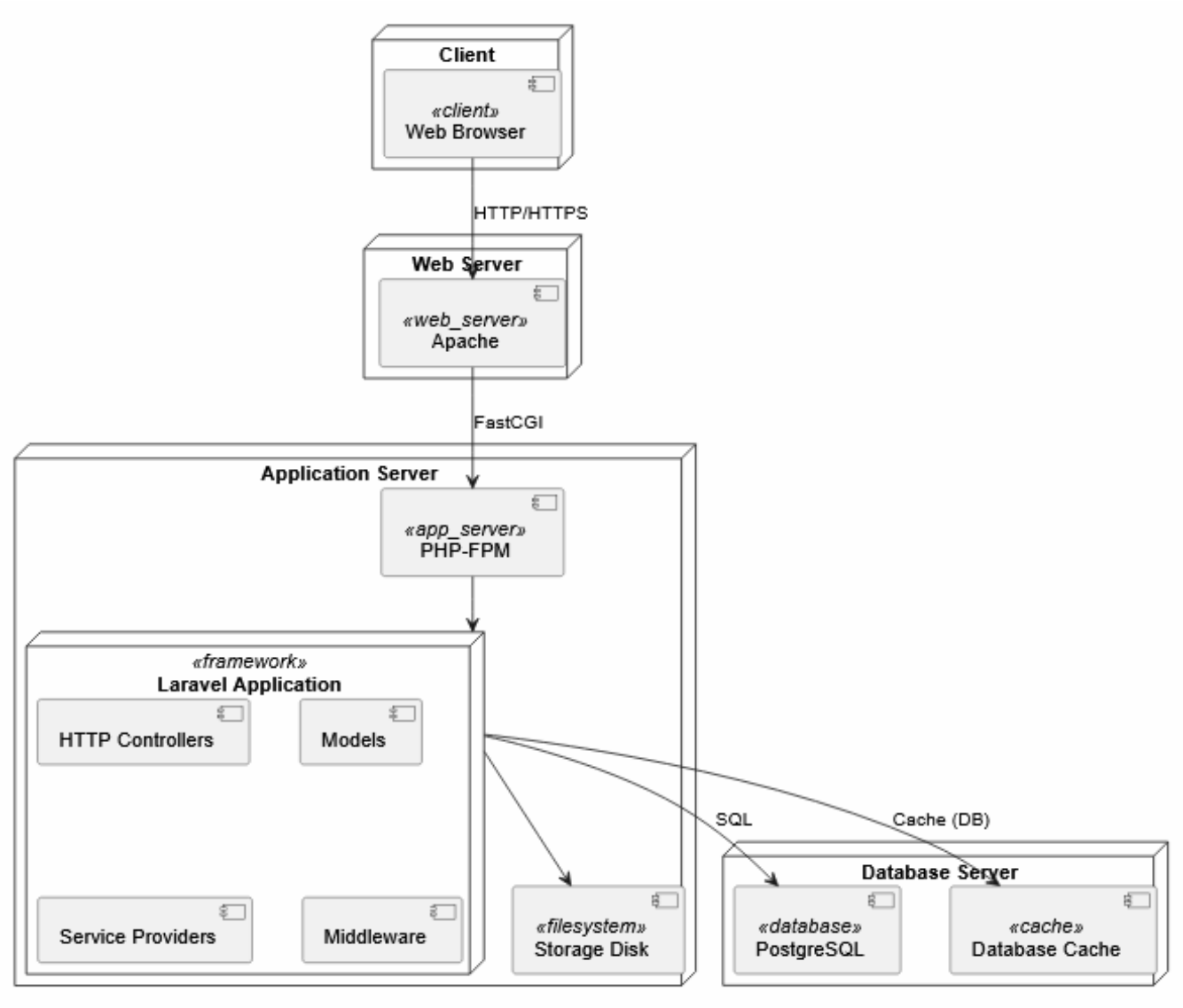


Рисунок А.3 – Діаграма розгортання

ДОДАТОК Б

Опис основних сутностей проєкту

Таблиця Б.1 – Контролери та їх опис

Назва	Короткий опис	Основні методи
ProfileController	Керування профілем користувача (пароль, аватар)	updatePassword(), updateAvatar()
OrderController	Управління замовленнями та escrow-операціями	validateOrderData(), sortProposals(), createOrder(), deleteAttachment(), editOrder(), showOrder(), showMyOrders(), addAttachment(), cancelOrder(), extendDeadline(), completeOrder()
TransactionController	Фінансові операції (депозит, зняття)	history(), deposit(), withdraw()
BankAccountController	Керування банківськими картками	deleteCard(), createCard()
RegisterController	Реєстрація нових користувачів	validator(), create()
HomeController	Головна сторінка сайту	pricing(), faq()
ReviewController	Управління відгуками	leaveReview(), delete()
OrderApproveController	Управління пропозиціями виконавців	showProposals(), submit(), update(), cancel(), makeApprove()
OrderCommentController	Управління коментарями до замовлень	leaveComment(), addAttachment()
ReportController	Управління скаргами	show(), storeComment(), create(), store()
NotificationController	Сповіщення користувачів	getUnreadCount()
ChatController	Чати та обмін повідомленнями	getUserChats(), show(), getNewMessages(), startChat(), sendMessage(), getOlderMessages(), getUnreadStatus()

Таблиця Б.2 – Моделі та їх опис

Назва	Короткий опис	Атрибути
User	Користувачі системи (замовники, фрілансери)	id, name, email, password, user_role_id, remember_token, email_verified_at, created_at, updated_at
UserRole	Ролі користувачів (customer, executor)	id, name
UserAvatar	Аватари користувачів	id, user_id, path
Balance	Баланс користувача	id, user_id, amount, escrowed_amount, currency
BankAccount	Банківські картки користувачів	id, user_id, name, card_number
Order	Замовлення/проєкти	id, title, requirement_skills, short_description, full_description, budget, customer_id, executor_id, status_id, sub_category_id, deadline_in_days, deadline_date, created_at, updated_at
OrderStatus	Статуси замовлень	id, name
MainOrderCategory	Основні категорії замовлень	id, name
SubOrderCategory	Підкатегорії замовлень	id, main_order_category_id, name
OrderApprove	Пропозиції виконавців	id, user_id, order_id, comment, proposed_budget, proposed_deadline_in_days
OrderFileAttachment	Вкладення до замовлень	id, order_id, stored_filename, original_filename
OrderComment	Коментарі до замовлень	id, order_id, user_id, value
OrderCommentFileAttachment	Вкладення до коментарів	id, order_comment_id, stored_filename, original_filename
Transaction	Фінансові транзакції	id, user_id, order_id, transaction_type_id, bank_account_id, related_user_id, amount, transfer_uuid, meta, created_at, updated_at
TransactionType	Типи транзакцій	id, name
Review	Відгуки про виконану роботу	id, order_id, author_id, target_id, score, feedback, created_at, updated_at

Таблиця Б.3 – Представлення та їх опис

Категорія	Файл	Опис
Layout	main.blade.php	Головний шаблон, підключення CSS/JS, хедер/футер
Components	header.blade.php	Навігація, меню, логотип, сповіщення
	footer.blade.php	Підвал сторінки
Home	pages/home/index.blade.php	Головна, список замовлень, фільтри
	pages/home/pricing.blade.php	Тарифи
	pages/home/faq.blade.php	Часті питання
Orders	pages/orders/create.blade.php	Форма створення замовлення
	pages/orders/show-order.blade.php	Перегляд замовлення
	pages/orders/show-order-as-editable.blade.php	Редагування замовлення
	pages/orders/my-orders.blade.php	Мої замовлення
	pages/orders/proposals.blade.php	Пропозиції виконавців
	pages/orders/orders.blade.php	Список замовлень
	pages/orders/review-card.blade.php	Картка для відгуку
Profile	pages/profile/general.blade.php	Основна інформація
	pages/profile/bank-accounts.blade.php	Банківські картки
	pages/profile/transaction-history.blade.php	Історія транзакцій
	pages/profile/reviews.blade.php	Відгуки
	pages/profile/public-overview.blade.php	Публічний профіль
	pages/profile/menu.blade.php	Меню профілю
Chats	pages/chats/index.blade.php	Список чатів
	pages/chats/particular-chat.blade.php	Конкретний чат
Reports	pages/reports/index.blade.php	Список скарг
	pages/reports/create.blade.php	Створення скарги
	pages/reports/show.blade.php	Перегляд скарги
	pages/profile/reports.blade.php	Скарги профілю
Notifications	pages/notifications/index.blade.php	Сповіщення
Users	pages/users/registration.blade.php	Реєстрація
Auth	auth/register.blade.php	Форма реєстрації
	auth/login.blade.php	Форма входу

Таблиця Б.4 – Створені сідери та їх короткий опис

Назва	Короткий опис
DatabaseSeeder	Головний сідер, який викликає всі інші сідери в правильному порядку (забезпечує послідовність через call() метод)
UserRoleSeeder	Заповнює таблицю user_roles початковими ролями: customer (ID=1), executor (ID=2), admin (якщо є)
UserAvatarSeeder	Заповнює таблицю user_avatars стандартними аватарами: admin.png, customer.png, executor.png, no-avatar.png
BalanceSeeder	Створює початкові баланси для користувачів (amount=0, escrowed_amount=0, currency='UAH')
BankAccountSeeder	Заповнює таблицю bank_accounts тестовими картками (card_number, name) для користувачів
MainOrderCategorySeeder	Створює основні категорії замовлень: "Web Development", "Mobile Apps", "Design", "Writing" тощо
SubOrderCategorySeeder	Заповнює sub_order_categories підкатегоріями, які пов'язані з MainOrderCategory (наприклад, "PHP" для "Web Development")
OrderStatusSeeder	Створює статуси замовлень: published, in_progress, completed, cancelled, expired
TransactionTypeSeeder	Заповнює transaction_types: deposit, withdraw, escrow, release_escrow, refund_escrow, transfer, fee
OrderSeeder	Генерує тестові замовлення з різними статусами, бюджетами та категоріями
OrderFileAttachmentSeeder	Додає тестові вкладення до замовлень (stored_filename, original_filename)
OrderApproveSeeder	Створює пропозиції від виконавців (proposed_budget, proposed_deadline_in_days, comment)
OrderCommentSeeder	Заповнює order_comments коментарями до замовлень від замовників та виконавців
TransactionSeeder	Генерує тестові транзакції для різних типів (deposit, escrow, transfer)
ReviewSeeder	Створює відгуки про виконавців/замовників (score, feedback)
ChatSeeder	Створює тестові чати між замовниками та виконавцями
ChatMessageSeeder	Заповнює chat_messages повідомленнями в чатах
NotificationTypeSeeder	Створює типи сповіщень: order_completed, new_proposal, new_message тощо
NotificationSeeder	Генерує тестові сповіщення для користувачів
ReportSeeder	Створює тестові скарги (reports) на користувачів або замовлення
ReportCommentSeeder	Додає коментарі до скарг (report_comments)
ReportStatusSeeder	Заповнює report_statuses: pending, investigating, resolved, rejected

Таблиця Б.5 – Створені міграції та їх опис

Скорочена назва	**Опис таблиці**	**Атрибути**
create_users_table	Користувачі системи: замовники, фрілансери, адміністратори.	id, name, email, password, role_id
create_cache_table	Кеш Laravel.	key, value, expiration
create_jobs_table	Черга фонових завдань.	id, queue, payload
create_user_avatars_table	Аватари користувачів.	id, user_id, path
create_bank_accounts_table	Банківські картки.	id, user_id, name, card_number
create_balances_table	Баланси користувачів.	id, user_id, amount, escrowed_amount
create_transaction_types_table	Типи транзакцій.	id, name
create_transactions_table	Фінансові операції.	id, user_id, order_id, amount
create_main_order_categories_table	Основні категорії замовлень.	id, name
create_sub_order_categories_table	Підкатегорії замовлень.	id, category_id, name
create_order_statuses_table	Статуси замовлень.	id, name
create_orders_table	Замовлення з бюджетом і строками.	id, title, budget, customer_id
create_order_file_attachments_table	Файли до замовлень.	id, order_id, filename
create_order_approves_table	Заявки виконавців.	id, user_id, order_id, proposed_budget
create_order_comments_table	Коментарі до замовлень.	id, order_id, user_id, value
create_order_comment_file_attachments_table	Файли до коментарів.	id, comment_id, filename
create_reviews_table	Відгуки та оцінки.	id, order_id, score
create_chats_table	Чати користувачів.	id, user1_id, user2_id
create_chat_messages_table	Повідомлення в чатах.	id, chat_id, message
create_notification_types_table	Типи сповіщень.	id, name
create_notifications_table	Сповіщення користувачів.	id, user_id, title
create_report_statuses_table	Статуси скарг.	id, name
create_reports_table	Скарги на користувачів/замовлення.	id, author_id, target_id
create_report_comments_table	Коментарі до скарг.	id, report_id, user_id

Таблиця Б.6 – Опис створених маршрутів

Назва маршруту	**Метод**	**URL**	**Опис**
home.index	GET	/	Відображає головну сторінку платформи зі списком актуальних замовлень.
profile.index	GET	/profile	Надає доступ до особистого кабінету користувача з основною інформацією профілю.
profile.edit	GET	/profile/edit	Дозволяє редагувати персональні дані користувача.
profile.transactions.history	GET	/profile/transactions/history	Історія фінансових операцій користувача.
order.create-order	GET/POST	/orders/create-order	Забезпечує створення нового замовлення із зазначенням усіх необхідних параметрів.
order.show-orders	GET	/my-orders	Виводить список замовлень, створених користувачем.
order.show-order	GET	/orders/{order}	Відображає детальну інформацію про конкретне замовлення.
order.cancel-order	PATCH	/orders/cancel-order/{order}	Дозволяє скасувати активне замовлення.
order.complete-order	PATCH	/orders/complete-order/{order}	Позначає замовлення як виконане після завершення роботи.
order.approve	POST	/orders/{order}/approve	Дозволяє виконавцю подати пропозицію на виконання замовлення.
order.approval-submit	POST	/orders/{order}/approval-submit/{orderApprove}	Надає можливість замовнику обрати виконавця.
chat.index	GET	/chats	Відображає список чатів користувача для комунікації.
chat.send-message	POST	/chats/{chat}/send-message	Забезпечує надсилання повідомлень у межах чату.
notifications.index	GET	/notifications	Відображає список сповіщень користувача.
report.store	POST	/reports/create/{order}	Дозволяє створити та зберегти скаргу щодо замовлення.
report.complete	PATCH	/reports/{report}/complete	Позначає скаргу як вирішену після її розгляду.

Таблиця Б.7 – Опис створених тестів

Клас та назва тесту	Опис
Unit\ModelTest «user has role relationship»	Перевіряє коректний зв’язок моделі користувача з моделлю ролі та правильне отримання ролі користувача.
Unit\ModelTest «user has balance relationship»	Перевіряє наявність і коректність зв’язку користувача з балансом рахунку.
Unit\ModelTest «user has avatar relationship»	Перевіряє зв’язок користувача із завантаженим аватаром профілю.
Unit\ModelTest «avatar attribute accessor»	Перевіряє аксесор атрибута аватара та правильне формування шляху до файлу.
Unit\ModelTest «order has status relationship»	Перевіряє зв’язок замовлення з його поточним статусом.
Unit\ModelTest «order is in progress scope»	Перевіряє scope-фільтр для вибірки замовлень у процесі виконання.
Unit\ModelTest «order scope is published»	Перевіряє scope-фільтр для опублікованих замовлень.
Unit\ModelTest «balance cannot go negative»	Перевіряє бізнес-правило, за яким баланс не може стати від’ємним.
Unit\ModelTest «transaction has user relationship»	Перевіряє зв’язок транзакції з користувачем-власником.
Unit\ModelTest «transaction has type relationship»	Перевіряє зв’язок транзакції з типом операції.
Unit\ModelTest «calculate fee»	Перевіряє правильність автоматичного розрахунку комісії у розмірі 5%.
Unit\ModelTest «bank account belongs to user»	Перевіряє належність банківського рахунку конкретному користувачу.
Feature\OrderControllerTest «guest cannot access order routes»	Перевіряє, що неавторизований користувач не має доступу до маршрутів замовлень.
Feature\OrderControllerTest «non customer cannot create order»	Перевіряє заборону створення замовлення користувачам без ролі замовника.
Feature\OrderControllerTest «customer can create order with valid data»	Перевіряє успішне створення замовлення з валідними даними та блокування коштів в депонування.
Feature\OrderControllerTest «customer cannot create order with insufficient balance»	Перевіряє неможливість створення замовлення при недостатньому балансі.
Feature\OrderControllerTest «order creation validates input»	Перевіряє валідацію обов’язкових і коректних полів при створенні замовлення.
Feature\OrderControllerTest «customer can edit own order»	Перевіряє редагування власного замовлення та коректне коригування депонування.
Feature\OrderControllerTest «customer cannot edit others order»	Перевіряє заборону редагування чужих замовлень.
Feature\OrderControllerTest «cannot edit non published order»	Перевіряє неможливість редагування замовлення, яке не має статусу published.
Feature\OrderControllerTest «customer can cancel published order»	Перевіряє скасування опублікованого замовлення з поверненням коштів.

Закінчення таблиці Б.7

Feature\OrderControllerTest «customer can complete in progress order»	Перевіряє завершення замовлення в роботі та виплату виконавцю.
Feature\OrderControllerTest «customer can extend deadline»	Перевіряє можливість продовження строку виконання замовлення.
Feature\OrderControllerTest «customer can view own order»	Перевіряє перегляд деталей власного замовлення.
Feature\OrderControllerTest «customer can view my orders»	Перевіряє перегляд списку власних замовлень.
Feature\TransactionControllerTest «guest cannot access finance pages»	Перевіряє заборону доступу гостя до фінансових сторінок.
Feature\TransactionControllerTest «user can deposit funds»	Перевіряє успішне поповнення балансу через банківську картку.
Feature\TransactionControllerTest «deposit fails with others card»	Перевіряє заборону поповнення з чужої банківської картки.
Feature\TransactionControllerTest «deposit validates amount»	Перевіряє валідацію суми поповнення.
Feature\TransactionControllerTest «user can withdraw funds»	Перевіряє успішне виведення коштів на банківську картку.
Feature\TransactionControllerTest «withdraw fails with insufficient balance»	Перевіряє неможливість виведення коштів при недостатньому балансі.
Feature\TransactionControllerTest «withdraw fails with others card»	Перевіряє заборону виведення коштів на чужу картку.
Feature\ProfileControllerTest «guest cannot access profile»	Перевіряє заборону доступу гостя до сторінки профілю.
Feature\ProfileControllerTest «user can update avatar»	Перевіряє успішне оновлення аватара користувача.
Feature\ProfileControllerTest «avatar validation fails without file»	Перевіряє помилку валідації при спробі оновлення аватара без файлу.
Feature\ProfileControllerTest «avatar cannot exceed 2mb»	Перевіряє обмеження максимального розміру файлу аватара – 2 МБ.
Feature\ProfileControllerTest «user can update password»	Перевіряє успішну зміну пароля користувача.
Feature\ProfileControllerTest «password update fails with wrong current»	Перевіряє відхилення зміни пароля при невірному поточному паролі.
Feature\ProfileControllerTest «password must be min 8 chars»	Перевіряє вимогу мінімальної довжини пароля – 8 символів.
Feature\EscrowTest «escrow funds locked on order creation»	Перевіряє блокування коштів у системі депонування після створення замовлення.
Feature\EscrowTest «escrow releases on order completion»	Перевіряє переказ коштів виконавцю після успішного завершення замовлення.
Feature\EscrowTest «escrow refunds on order cancellation»	Перевіряє повернення коштів замовнику після скасування замовлення.
Feature\EscrowTest «escrow adjusts on budget increase»	Перевіряє збільшення суми депонування при підвищенні бюджету замовлення.
Feature\EscrowTest «escrow adjusts on budget decrease»	Перевіряє зменшення суми депонування при зниженні бюджету замовлення.