ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)
«_____»___202_ р.

КВАЛІФІКАЦІЙНА РОБОТА
на тему

«ПРОГРАМНА РЕАЛІЗАЦІЯ САЙТУ ДЛЯ ФРІЛАНСЕРІВ»

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавр

Виконавець роботи Куцовол Руслан Сергійович
_____«____» _____202_ р.
(підпис)

Науковий керівник доцент, к.ф.-м.н. Черненко О. О.
_____«____»_____202_ р.
(підпис)

Рецензент

ПОЛТАВА 2025

РЕФЕРАТ

Записка: 59 с., 13 рис., 1 таблиця, 1 додаток, 11 джерел.

FREELANCE-ПЛАТФОРМА, ВЕБРОЗРОБКА, NODE.JS, REACT, МОНГОДБ, API, АВТОРИЗАЦІЯ, ПРОЄКТИ, РЕЙТИНГ, АНАЛІТИКА

Об’єктом розробки є програмне забезпечення — вебплатформа для організації співпраці між замовниками та фрілансерами.

Предметом розробки є реалізація інтерактивного вебсайту з підтримкою управління проєктами, обміну повідомленнями, рейтингової системи та аналітики в межах повноцінної системи керування життєвим циклом фріланс-завдань.

Метою роботи є створення сучасної, безпечної та масштабованої платформи, що дозволяє автоматизувати процеси публікації проєктів, подання пропозицій, виконання робіт та формування зворотного зв’язку.

У ході реалізації розроблено такі можливості:

- багаторівнева система ролей: фрілансер, замовник, адміністратор;
- створення, редагування та фільтрація проєктів за категоріями, бюджетами, складністю;
- система заявок (пропозицій) із можливістю їх прийняття;
- профілі користувачів із портфоліо, навичками, досвідом і рейтингом;
- чат між сторонами в межах конкретного проєкту;
- реалізована система відгуків з метриками якості;
- адміністративна панель з аналітикою (успішність, дохід, топ фрілансери, активність);
- інтерфейс створення проєктів із валідацією;
- система кешування та індексації пошуку;
- інструкція користувача з прикладами та скріншотами.

Архітектура проєкту базується на стеку Node.js (Express) для бекенду, MongoDB — як документоорієнтована база даних, React + TypeScript — для фронтенду, із використанням REST API, JWT для авторизації та bcrypt для безпечного зберігання паролів. Внутрішня логіка модулів структурована згідно з

принципами SOLID і дозволяє легко масштабувати функціонал.

Результатом роботи стала стабільна, протестована вебсистема з чітким поділом прав доступу, аналітичними панелями та зручним користувацьким інтерфейсом. Усі основні функції перевірені на практиці та адаптовані для подальшого комерційного або навчального використання.

Практична цінність розробки полягає у створенні фреймворку, який може використовуватися як основа для власних бірж фрілансу, систем з керування підрядниками або у якості навчального проєкту з дисциплін «Веброзробка», «Проєктування ПЗ», «Бази даних».

Платформа має потенціал розширення, зокрема через:

- підключення платіжних систем (Stripe, LiqPay);
- інтеграцію з ML-рекомендаціями для пошуку виконавців;
- створення мобільної версії;
- підключення модуля техпідтримки та розсилки повідомлень.

Розроблене рішення відповідає сучасним вимогам до SaaS-продуктів, має гнучку структуру та може бути успішно застосоване в освітніх або бізнес-цілях.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕННЬ, СИМВОЛІВ, СКОРОЧЕННЬ, ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
API	Інтерфейс прикладного програмування (Application Programming Interface) — набір ендпоінтів для взаємодії клієнта і сервера
JWT	JSON Web Token — метод авторизації з використанням цифрових токенів
CRUD	Створення, читання, оновлення, видалення (Create, Read, Update, Delete) — базові операції з даними
UI/UX	Інтерфейс користувача / досвід користувача (User Interface / User Experience)
REST	Архітектурний стиль взаємодії між клієнтом і сервером через HTTP-запити
MongoDB	Документоорієнтована база даних, яка зберігає дані у форматі BSON (JSON-подібному вигляді)
Node.js	Середовище виконання JavaScript на сервері
React	JavaScript бібліотека для побудови інтерфейсів користувача
Express.js	Фреймворк для створення серверної частини на Node.js
Bcrypt	Бібліотека для хешування паролів
Frontend	Клієнтська частина вебзастосунку, з якою взаємодіє користувач
Backend	Серверна частина застосунку, що відповідає за обробку логіки, даних і запитів

Freelancer	Користувач, який пропонує свої послуги на платформі
Клієнт (замовник)	Користувач, який розміщує проєкти для виконання
Dashboard	Панель керування з основною статистикою та швидким доступом до модулів
Rating	Система оцінювання користувачів на основі відгуків
Proposal	Пропозиція від фрілансера щодо виконання певного проєкту
Project	Завдання, розміщене замовником, що потребує реалізації
API	Інтерфейс прикладного програмування (Application Programming Interface) — набір ендпоінтів для взаємодії клієнта і сервера

ВСТУП

У сучасному світі цифрових технологій стрімко зростає попит на дистанційну та проєктну роботу, що сприяє розвитку ринку фрілансу. Все більше компаній і приватних осіб шукають фахівців у форматі віддаленої співпраці для виконання завдань у сферах розробки програмного забезпечення, дизайну, маркетингу, копірайтингу та інших цифрових послуг. У зв’язку з цим виникає потреба у створенні сучасних веб-платформ, які забезпечують безпечну, зручну та ефективну взаємодію між замовниками та виконавцями.

Існуючі фріланс-платформи часто або перенасичені функціоналом, або ж не відповідають специфіці локального ринку праці. Крім того, багато з них мають обмежені можливості для керування повним життєвим циклом проєкту, не забезпечують прозорої системи рейтингу або належної інтеграції з платіжними та аналітичними системами.

Актуальність теми обумовлена необхідністю створення гнучкої, адаптивної та безпечної веб-платформи, яка дозволяє користувачам швидко створювати проєкти, знаходити виконавців, комунікувати в межах системи, здійснювати оцінювання та завершувати співпрацю з максимальною прозорістю. Особливу увагу при цьому слід приділити архітектурі, масштабованості, валідації та системі моніторингу.

Мета роботи — розробити програмну реалізацію платформи для фрілансерів, яка охоплює повний цикл проєктної взаємодії, включає систему ролей, систему рейтингів, захист персональних даних, API-інтеграції та механізми масштабування.

Завдання роботи:

- Проаналізувати існуючі платформи та визначити вимоги цільової аудиторії;
- Розробити логічну та фізичну структуру бази даних;
- Реалізувати ключові модулі системи: проєкти, пропозиції, повідомлення, відгуки, профілі користувачів;
- Інтегрувати зовнішні API та механізми безпеки (включно з авторизацією, шифруванням і rate-limiting);
- Налаштувати систему логування, моніторингу та оптимізації запитів;

- Створити інтерфейс користувача з використанням сучасних фронтенд-технологій;
- Розробити інструкцію користувача з прикладом використання платформи.

Об’єкт дослідження — процес розробки багатофункціональної веб-платформи для управління фріланс-проєктами.

Предмет дослідження — методи алгоритмізації, архітектури та розробки програмного забезпечення з урахуванням вимог до сучасних веб-додатків.

Практичне значення роботи полягає у створенні веб-платформи, яка може бути використана як комерційний сервіс або навчальний приклад комплексної розробки системи з урахуванням UX, безпеки, API-інтеграцій та роботи з базами даних.

ПОСТАНОВКА ЗАДАЧІ

У сучасних умовах розвитку цифрової економіки зростає попит на гнучкі форми зайнятості, зокрема фріланс. Онлайн-платформи для фрілансерів стають основним засобом пошуку проєктів, виконавців, а також побудови ефективної віддаленої співпраці. Проте більшість наявних платформ мають складний інтерфейс, закриту бізнес-логіку або не враховують специфіку локального ринку.

Метою цього дипломного проєкту є створення функціонального веб-сервісу, що забезпечує повноцінну взаємодію між клієнтами та фрілансерами, управління життєвим циклом проєктів, систему рейтингів і відгуків, комунікацію та захист персональних даних. Розробка платформи передбачає застосування сучасних технологій, побудову масштабованої архітектури та реалізацію інструментів моніторингу й аналітики.

Для досягнення цієї мети необхідно реалізувати наступні завдання:

1. Проаналізувати функціональні та технічні вимоги до платформи з урахуванням особливостей фріланс-ринку;
2. Спроєктувати логічну та фізичну структуру системи з урахуванням основних бізнес-процесів (створення проєктів, заявки, рейтинг, повідомлення);
3. Розробити та реалізувати ключові модулі: реєстрація, авторизація, управління проєктами, система пропозицій, відгуки та повідомлення;
4. Побудувати захищену систему доступу та валідації даних користувачів;
5. Реалізувати REST API для взаємодії між фронтендом та бекендом;
6. Створити адаптивний та зручний інтерфейс користувача з використанням сучасних фреймворків;
7. Інтегрувати платіжну систему, логування, моніторинг та базові KPI;
8. Провести тестування функціоналу, виявити та усунути критичні помилки;
9. Розробити інструкцію користувача щодо роботи з платформою.

Виконання вказаних завдань дозволить створити масштабований і сучасний веб-додаток, що може бути використаний як приклад комерційної розробки, а також як навчальна база для вивчення сучасних підходів до створення веб-систем із повноцінною backend- і frontend-логікою, захистом, API та бізнес-аналізом.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1. Аналіз ринку фріланс-платформ: огляд існуючих рішень та їх особливості

Фріланс-платформи стали невід'ємною частиною сучасного ринку праці, надаючи можливість замовникам швидко знайти виконавців, а фрілансерам — знайти проєкти, що відповідають їхнім навичкам. На ринку представлені численні рішення, що відрізняються функціональністю, цільовою аудиторією та рівнем забезпечення безпеки. Розглянемо основні платформи, їх особливості та переваги.

1. **Upwork** — одна з найбільших міжнародних платформ для фрілансерів, що пропонує широкий спектр можливостей як для початківців, так і для досвідчених фахівців. Платформа забезпечує широкий вибір проєктів, систему рейтингів та відгуків, а також забезпечує захист фінансових операцій через спеціальні механізми, такі як ескроу (див. рис. 1.1).

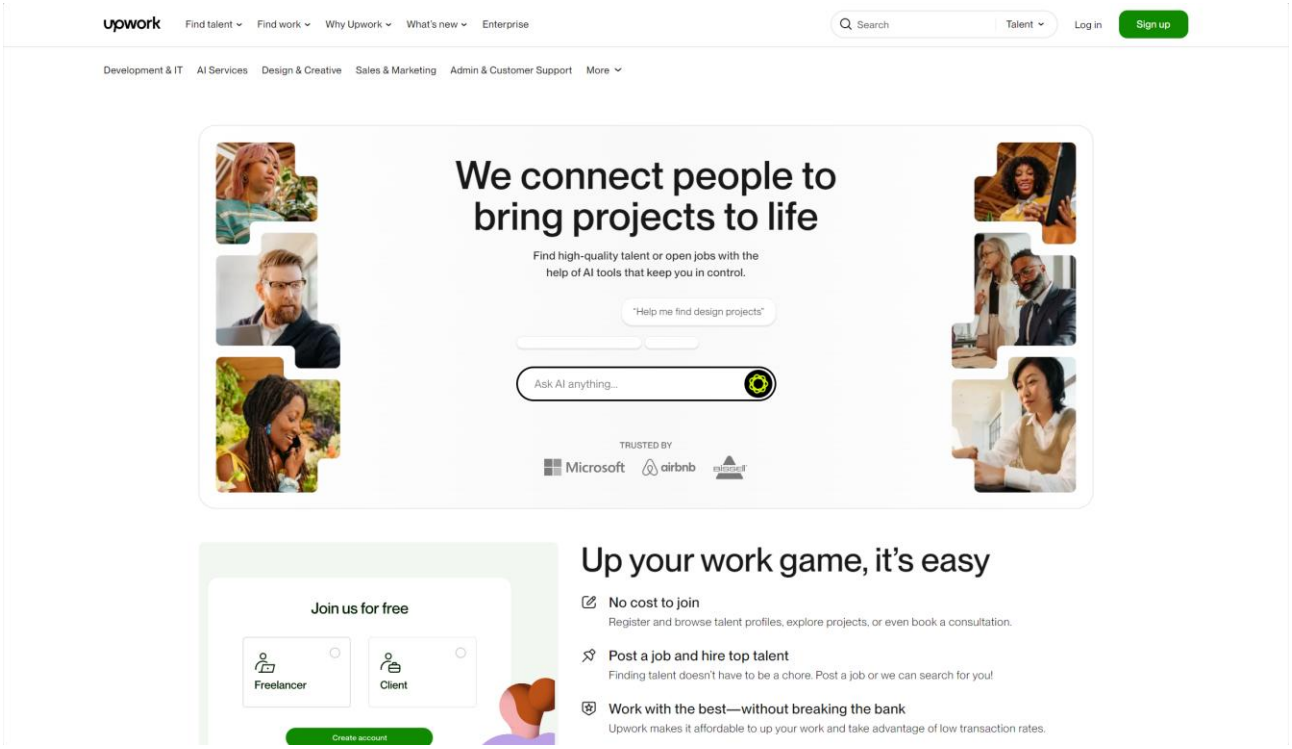

Рисунок 1.1 – Дизайн сайту Upwork

2. **Freelancer.com** — платформа, що спеціалізується на короткострокових проєктах. Вона дозволяє фрілансерам брати участь у конкурсах, щоб довести

свою кваліфікацію та отримати роботу. Однією з особливостей платформи є можливість швидко отримати досвід роботи через конкурси, однак велика конкуренція часто ускладнює отримання проєкту (див. рис. 1.2).

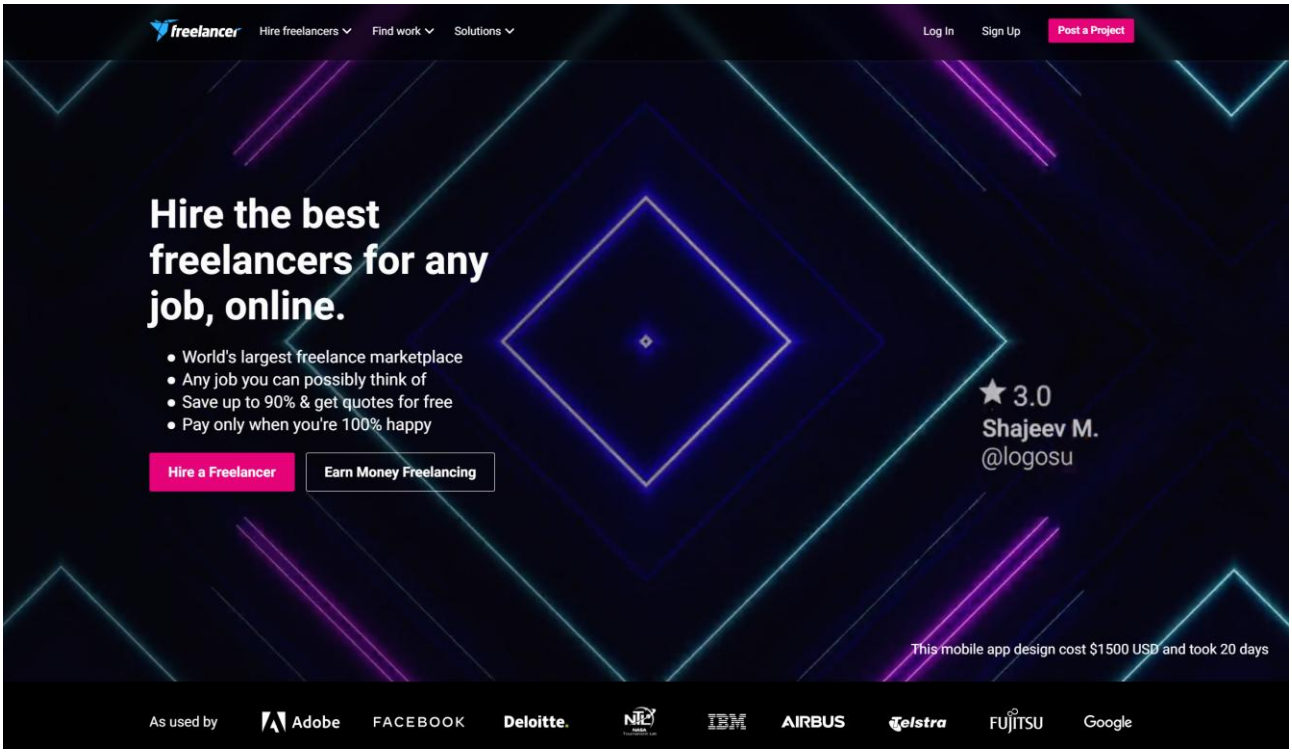

Рисунок 1.2 – Дизайн сайту Freelancer

3. **Fiverr** — платформа, де фрілансери пропонують свої послуги за фіксовану ціну. Fiverr більше орієнтована на невеликі проєкти та мікрозадачі. Її особливістю є простота використання і можливість швидко отримати виконавця на невелике завдання, однак обмеження по складності проєктів може бути мінусом для професіоналів (див. рис. 1.3).

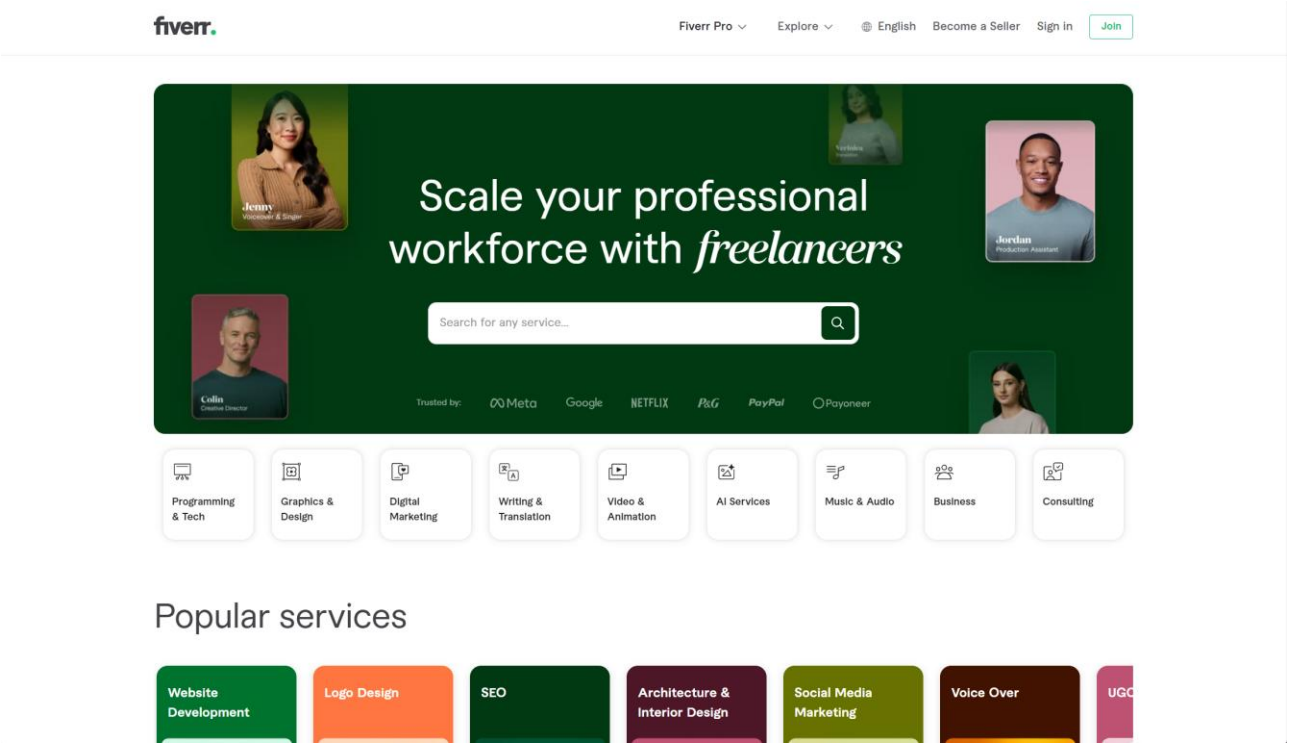

Рисунок 1.3 – Дизай сайту Fiverr

4. **Toptal** — платформа, що орієнтована на фахівців високого рівня. Toptal має суворий процес відбору виконавців, що гарантує високий рівень якості послуг. Ця платформа більше підходить для виконання складних та висококваліфікованих проєктів, але не є доступною для початківців через високі вимоги до кандидатів.

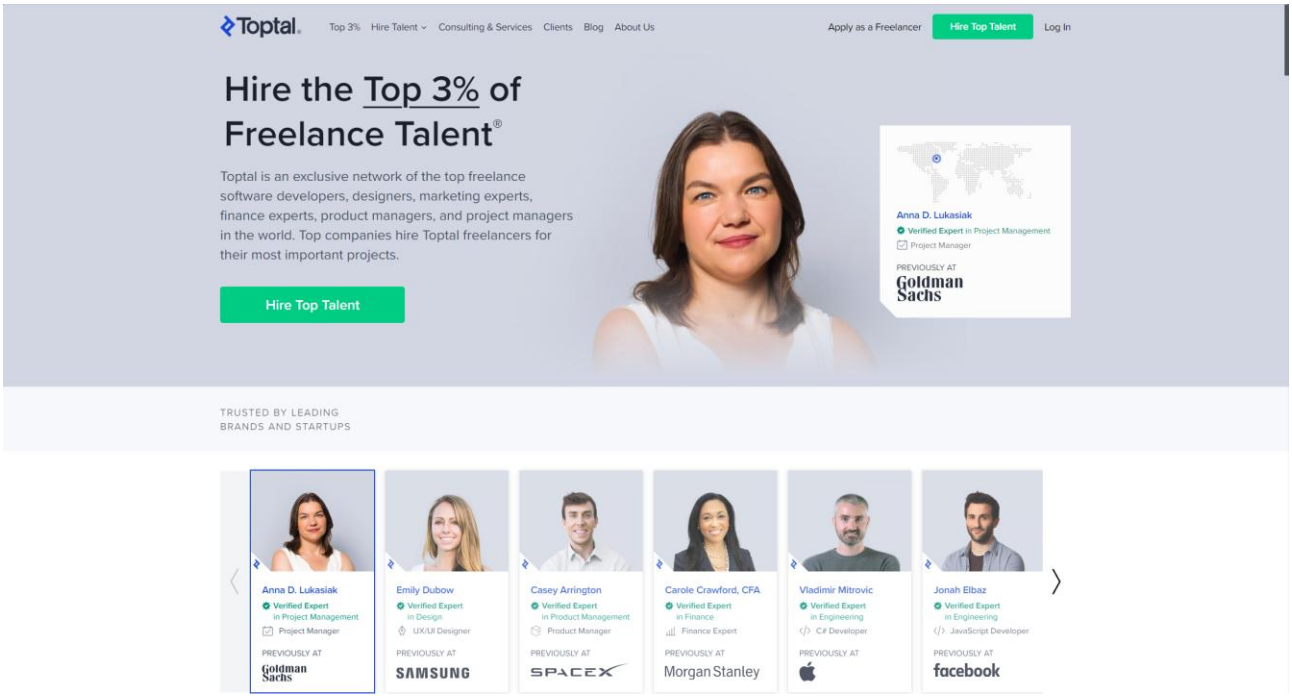

Рисунок 1.4 – Дизайн сайту Toptal

5. **WeWorkRemotely** — платформа, що спеціалізується на дистанційній роботі, орієнтована на довгострокову співпрацю. Її особливістю є відсутність комісії з боку виконавців, що є значною перевагою для фрілансерів, які шукають постійну зайнятість (див. рис. 1.5).

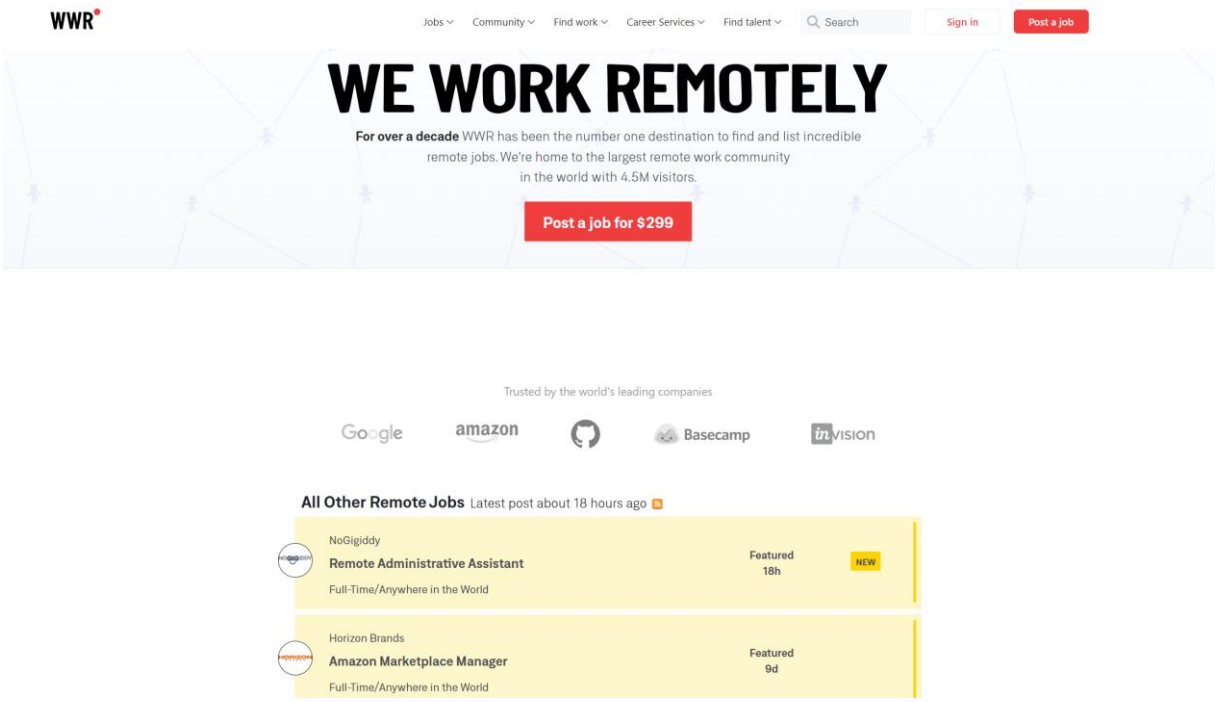

Рисунок 1.5 – Дизайн сайту WWR

Порівняльний аналіз

Кожна з вищезазначених платформ має свої особливості та переваги. Наприклад, Upwork та Freelancer.com пропонують широкий спектр проєктів і можливостей, однак великий рівень конкуренції може бути викликом для новачків. Fiverr добре підходить для невеликих завдань, але обмежений за складністю проєктів. Toptal забезпечує високу якість виконання робіт, але вимагає від фахівців високого рівня знань та навичок. WeWorkRemotely більше підходить для довгострокових проєктів, але не має такого широкого спектру проєктів, як Upwork.

Таким чином, вибір платформи залежить від конкретних потреб користувачів: масштаб проєкту, рівень досвіду виконавця та тип співпраці (короткострокова або довгострокова). Аналізуючи ринок фріланс-платформ, стає зрозумілим, що для успішної розробки нового порталу важливо врахувати основні переваги кожної з цих платформ, щоб створити продукт, який буде зручним, безпечним та ефективним для користувачів.

1.2. Порівняння популярних платформ для розробки веб-порталів

У цьому розділі буде розглянуто та порівняно найпопулярніші платформи для розробки веб-порталів. Основна мета — зрозуміти, які інструменти найкраще підходять для створення сучасних, швидких та зручних порталів. Ми звернемо увагу на такі платформи, як WordPress, Django, React, та Laravel, і розберемо їхні переваги та недоліки.

WordPress — це одна з найпоширеніших платформ, яка, перш за все, відома як система управління контентом (CMS). Її популярність обумовлена простотою використання, великою кількістю готових шаблонів та плагінів. Це ідеальний варіант для невеликих проєктів або сайтів з обмеженим бюджетом. Однак, коли мова йде про створення складних веб-порталів, WordPress може бути недостатньо гнучким. Інколи виникають проблеми з масштабованістю та безпекою, особливо якщо використовуються застарілі плагіни.

Django — це фреймворк на мові Python, який підходить для створення більш

складних веб-додатків. Django має багато вбудованих інструментів, що допомагають швидко розробляти функціональні додатки з використанням найкращих практик безпеки. Це хороший вибір для проєктів, які потребують надійності та масштабованості. Основною перевагою є висока швидкість розробки, але для роботи з Django потрібні знання Python, тому поріг входу дещо вищий, ніж у WordPress.

React — це бібліотека JavaScript, яка використовується для розробки інтерфейсів користувача. React стає дедалі популярнішим завдяки своїй гнучкості та швидкості. [1] Використання React дозволяє створювати сучасні динамічні веб-портали з високою інтерактивністю. Завдяки компонентному підходу, розробка інтерфейсів стає структурованою та зрозумілою. Основна складність у тому, що React — це лише фронтенд-інструмент, і для роботи над повним проєктом потрібні додаткові бекенд-інструменти, такі як Node.js.

Laravel — це фреймворк на мові PHP, який відомий своєю простотою та елегантністю. Laravel ідеально підходить для створення веб-додатків, які мають багато серверних функцій. Він забезпечує зручну інтеграцію з базами даних, має зрозумілу структуру та багато вбудованих функцій для аутентифікації, авторизації тощо. Laravel добре підходить для розробки як невеликих, так і середніх за складністю проєктів. Проте, якщо проєкт потребує високої продуктивності або складної бізнес-логіки, можливо, варто розглянути інші рішення.

Порівняльний аналіз

Якщо порівнювати ці платформи, то можна сказати, що вибір залежить від вимог конкретного проєкту. **WordPress** підходить для швидкого запуску сайтів з мінімальною кількістю кастомізацій. **Django** та **Laravel** є чудовими варіантами для створення повнофункціональних веб-додатків, де потрібна висока надійність та серверна логіка. **React** більше підходить для створення сучасних та інтерактивних інтерфейсів, але потребує додаткових інструментів для повноцінного бекенду.

Отже, для нашого порталу фрілансерів, який має бути динамічним, зручним і масштабованим, оптимальним вибором буде використання **React** для фронтенду та поєднання його з бекенд-фреймворком **Node.js**. [2] Це дозволить отримати

потужний інструмент з привабливим інтерфейсом та надійною серверною частиною, що забезпечить стабільну роботу платформи та зручність для користувачів.

1.3. Аналіз вимог цільової аудиторії

Перш ніж приступати до розробки порталу для фрілансерів, важливо зрозуміти потреби та очікування нашої цільової аудиторії. Платформа орієнтована на дві основні групи користувачів: фрілансерів та замовників. Кожна з цих груп має свої особливості та вимоги, які потрібно враховувати для успішної реалізації проєкту.

Фрілансери — це люди, які шукають можливість працювати віддалено, вибираючи цікаві для себе проєкти та керуючи власним робочим графіком. Вони зацікавлені в прозорих умовах роботи, гарантії оплати праці та можливості ефективно презентувати свої навички. Основні вимоги фрілансерів до платформи включають:

- **Зручність реєстрації та створення профілю**: фрілансери хочуть швидко створити профіль, що міститиме всю необхідну інформацію про їхні навички та досвід.
- **Можливість пошуку відповідних проєктів**: необхідно реалізувати зручну систему фільтрації та пошуку проєктів, щоб фрілансери могли легко знаходити завдання, що відповідають їхнім навичкам та інтересам.
- **Безпека та гарантії оплати**: одним з ключових факторів є впевненість у тому, що робота буде оплачена. Для цього важливо передбачити механізми, що забезпечують безпеку фінансових операцій, наприклад, систему ескроу.
- **Система рейтингів та відгуків**: фрілансери зацікавлені у створенні свого професійного іміджу, тому важливо мати можливість отримувати відгуки від замовників та накопичувати рейтинг.

Замовники — це компанії або окремі особи, які шукають виконавців для виконання певних завдань. Вони зацікавлені в тому, щоб швидко знайти

кваліфікованих спеціалістів, що виконають роботу якісно та вчасно. Основні вимоги замовників до платформи включають:

- **Зручний процес публікації проєктів**: замовники хочуть мати можливість швидко публікувати проєкти, вказуючи всі необхідні деталі та вимоги до виконавців.
- **Можливість оцінки кандидатів**: платформа має надавати інструменти для перегляду профілів фрілансерів, аналізу їх досвіду, портфоліо та відгуків від попередніх клієнтів.
- **Безпека співпраці**: замовники прагнуть бути впевненими в тому, що фрілансери виконають роботу якісно, тому важливим аспектом є механізм забезпечення виконання завдань, наприклад, через використання етапів (міле́стоуни).
- **Комунікація з виконавцями**: платформа повинна мати зручні інструменти для обміну повідомленнями, обговорення умов роботи та уточнення вимог по проєкту.

Потреби спільноти. Крім основних вимог фрілансерів і замовників, важливо врахувати й потреби всієї спільноти користувачів платформи. Платформа повинна сприяти розвитку професійних контактів, обміну досвідом і знаннями, тому важливо передбачити можливість створення спільнот за інтересами, форуми або групи для обговорення тем, що цікавлять користувачів.

Таким чином, для успішної реалізації порталу важливо забезпечити зручність використання для обох груп користувачів, гарантувати безпеку фінансових операцій, створити прозорі умови співпраці та сприяти розвитку професійних зв'язків. Аналіз вимог цільової аудиторії дозволяє створити платформу, яка буде затребуваною і корисною як для фрілансерів, так і для замовників.

2. ТЕОРЕТИЧНА ЧАСТИНА

2.1. Вибір платформи для розробки: огляд технологій

Для розробки нашого порталу фрілансерів необхідно вибрати відповідні технології, які дозволять створити зручний, швидкий та масштабований продукт. У цьому розділі ми розглянемо та обґрунтуємо вибір технологій для фронтенду, бекенду та бази даних, які будуть використовуватися в розробці порталу.[1]

Фронтенд: React

Для створення користувацького інтерфейсу ми обрали бібліотеку **React**. Це одна з найпопулярніших JavaScript-бібліотек для розробки інтерфейсів користувача, яка має багато переваг. Основні причини вибору React включають:

- **Компонентний підхід**: React дозволяє створювати багаторазові компоненти, що спрощує розробку та підтримку проєкту. Компоненти можна використовувати повторно, що економить час і ресурси.
- **Висока продуктивність**: React використовує технологію віртуального DOM, яка допомагає зменшити кількість перерендерів на сторінці, що забезпечує високу швидкість роботи додатку.
- **Велика спільнота та підтримка**: React має велику спільноту розробників і хорошу документацію, що спрощує навчання та пошук рішень у разі виникнення проблем.
- **Гнучкість та інтеграція**: React легко інтегрується з іншими технологіями та фреймворками, що дозволяє створювати динамічні та інтерактивні додатки.

Бекенд: Node.js

Для серверної частини нашого порталу ми обрали **Node.js**. Це середовище виконання JavaScript, яке дозволяє використовувати одну мову програмування як на стороні клієнта, так і на стороні сервера. Основні переваги Node.js:

- **Одномовність**: використання JavaScript і на фронтенді, і на бекенді спрощує розробку, оскільки команда розробників може працювати з однією мовою програмування.

- **Асинхронність**: Node.js забезпечує асинхронну обробку запитів, що дозволяє ефективно обробляти велику кількість одночасних запитів без блокування.
- **Масштабованість**: завдяки своїй архітектурі, Node.js легко масштабується та підходить для розробки додатків, які потребують обробки великої кількості даних у реальному часі.
- **Велика кількість модулів**: Node.js має доступ до npm (Node Package Manager), що дозволяє використовувати багато готових модулів і бібліотек, значно скорочуючи час на розробку.

База даних: PostgreSQL

Для зберігання даних ми вирішили використовувати реляційну базу даних **PostgreSQL**. Це одна з найпопулярніших і надійних баз даних з відкритим вихідним кодом, яка має такі переваги:

- **Надійність та масштабованість**: PostgreSQL підтримує транзакції, що гарантує цілісність даних, і добре масштабується для великих обсягів інформації. [3]
- **Гнучкість**: вона дозволяє працювати як зі структурованими, так і з неструктурованими даними завдяки підтримці JSON-типів, що зручно для зберігання додаткової інформації.
- **Активна спільнота**: PostgreSQL має велику спільноту, що забезпечує швидку підтримку і регулярне оновлення.

Комбінований вибір

Вибір технологій **React**, **Node.js**, та **PostgreSQL** дозволяє створити сучасний веб-портал, який буде швидким, зручним у використанні та надійним. Використання однієї мови програмування — JavaScript — на всіх етапах розробки (фронтенд і бекенд) допоможе оптимізувати процес розробки та забезпечить узгодженість коду. Така комбінація технологій забезпечить високу продуктивність, легкість у підтримці та масштабуванні, що є ключовими вимогами для успішного запуску порталу для фрілансерів.

2.2. Основи архітектури веб-порталу: модель MVC, REST API

Для успішної розробки порталу фрілансерів необхідно створити добре продуману архітектуру, яка забезпечить чітке розділення функцій, зручність підтримки та масштабованість системи. У цьому розділі ми розглянемо дві ключові концепції, які будуть використовуватися при розробці порталу: модель MVC (Model-View-Controller) та REST API (Representational State Transfer Application Programming Interface).

Модель MVC

Модель MVC (Model-View-Controller) є однією з найбільш поширених архітектурних моделей у розробці веб-додатків. [4] Вона дозволяє розділити логіку додатку на три окремі частини: модель, представлення та контролер, що допомагає краще організувати код та зробити його зрозумілішим для підтримки і розвитку. Розглянемо компоненти моделі MVC:

- **Model (Модель)** — це частина додатку, яка відповідає за бізнес-логіку та роботу з даними. Модель безпосередньо працює з базою даних, зберігаючи і отримуючи дані, що стосуються користувачів, проєктів та інших елементів порталу.
- **View (Представлення)** — це частина додатку, яка відповідає за відображення інформації користувачам. Вона визначає, як саме будуть виглядати інтерфейси користувача, та відображає дані, отримані з моделі, у зручному для користувача вигляді.
- **Controller (Контролер)** — це частина додатку, яка обробляє запити від користувачів. Контролер отримує запити, взаємодіє з моделлю для отримання або зміни даних, а потім передає ці дані в представлення для відображення.

Використання моделі MVC дозволяє досягти чіткого розділення завдань у додатку, що полегшує підтримку та розвиток проєкту. Наприклад, зміни в інтерфейсі (View) можна здійснювати незалежно від логіки роботи з даними (Model), що значно прискорює процес розробки.

Як буде виглядати наша архітектура MVC

У нашому порталі фрілансерів модель MVC буде реалізована наступним чином:

- **Model** буде відповідати за управління даними, такими як інформація про користувачів, проєкти, замовлення, рейтинги. Наприклад, для кожного користувача буде зберігатися його профіль, історія проєктів, відгуки та рейтинг.
- **View** буде реалізовуватися на фронтенді за допомогою React, де ми створимо динамічний і зручний інтерфейс для користувачів. Представлення буде включати сторінки реєстрації, пошуку проєктів, перегляду профілів та інші важливі компоненти.
- **Controller** буде написаний на Node.js і буде виконувати роль посередника між користувачем та базою даних. Контролер оброблятиме запити від користувачів, такі як реєстрація нового користувача, створення проєкту або залишення відгуку, і відповідно оновлюватиме дані в моделі.

REST API

REST API (Representational State Transfer) є популярним підходом до створення веб-сервісів, який дозволяє обмінюватися даними між клієнтом і сервером через HTTP-запити. [5] REST API забезпечує стандартизований спосіб взаємодії між фронтендом та бекендом додатку, що особливо важливо для масштабованих систем.

Основні принципи REST API включають:

- **Клієнт-серверна архітектура**: клієнт (наприклад, фронтенд на React) надсилає запити серверу (бекенд на Node.js), який обробляє ці запити та повертає відповіді.
- **Безстанова взаємодія**: кожен запит від клієнта до сервера є незалежним і не залежить від попередніх запитів. Це означає, що сервер не зберігає інформацію про стан клієнта між запитами, що робить взаємодію більш гнучкою та масштабованою.
- **Стандартизовані методи HTTP**: REST API використовує стандартні методи HTTP, такі як GET (отримання даних), POST (створення нових даних),

PUT/PATCH (оновлення даних) та DELETE (видалення даних). Це забезпечує зрозумілу та послідовну взаємодію з ресурсами сервера.

- **Ресурсна орієнтованість**: в REST API всі елементи додатку розглядаються як ресурси (наприклад, користувачі, проєкти, завдання), кожен з яких має унікальний URL. Це дозволяє легко отримувати доступ до необхідних даних та керувати ними.

Як буде виглядати наша REST API

У нашому порталі фрілансерів REST API буде використовуватися для зв'язку між фронтендом (React) [2] і бекендом (Node.js). Наприклад:

- Для реєстрації нового користувача фронтенд буде надсилати POST-запит до кінцевої точки /api/users, де контролер обробить запит, створить нового користувача в базі даних і поверне відповідь з підтвердженням.
- Для отримання списку доступних проєктів фронтенд буде надсилати GET-запит до кінцевої точки /api/projects, де контролер звернеться до моделі, отримає дані про проєкти та поверне їх у вигляді JSON.
- Для оновлення інформації про користувача, наприклад, додавання нового відгуку, фронтенд буде використовувати PUT-запит до кінцевої точки /api/users/:userId/review, де контролер оновить дані в базі.

Взаємодія MVC та REST API

У нашому проєкті модель MVC буде використовуватися на бекенді для чіткого розділення логіки додатку, тоді як REST API забезпечить комунікацію між фронтендом і сервером. Контролери будуть приймати HTTP-запити від клієнта, взаємодіяти з моделями для отримання або модифікації даних та повертати ці дані у форматі, зрозумілому для фронтенду (наприклад, JSON). [6]

Таке поєднання MVC та REST API дозволяє створити архітектуру, яка є структурованою, зрозумілою і легко підтримуваною. Це забезпечить гнучкість у майбутньому розвитку системи, дозволить швидко вносити зміни та доповнення, а також забезпечить ефективну комунікацію між клієнтом та сервером, що є важливим для стабільної роботи порталу фрілансерів.

2.3. Проектування бази даних: вибір СУБД, структура даних

Для розробки порталу фрілансерів важливо правильно вибрати систему управління базами даних (СУБД) та спроектувати структуру даних таким чином, щоб вона відповідала потребам додатку, забезпечувала масштабованість і надійність. У цьому розділі розглянемо вибір СУБД та опишемо структуру бази даних для проєкту.

Було вирішено використовувати реляційну базу даних **PostgreSQL** для зберігання даних нашого порталу. PostgreSQL — це одна з найпопулярніших реляційних СУБД з відкритим вихідним кодом, яка має багато переваг, включаючи високу надійність, можливість роботи з великими обсягами даних, а також підтримку складних запитів. Основні причини вибору PostgreSQL включають:

- **Масштабованість та продуктивність**: PostgreSQL добре підходить для обробки великих обсягів даних і легко масштабується, що дозволяє працювати з великими проєктами.
- **Підтримка транзакцій**: СУБД підтримує ACID-транзакції, що гарантує цілісність даних та захист від помилок під час роботи з базою даних.
- **Гнучкість**: PostgreSQL дозволяє працювати як зі структурованими, так і з неструктурованими даними, що корисно для зберігання різноманітної інформації про користувачів, проєкти та інші ресурси порталу. [3]
- **Широка спільнота та підтримка**: PostgreSQL має активну спільноту, що дозволяє отримувати підтримку та доступ до великої кількості навчальних матеріалів та інструментів.

Структура даних

База даних для порталу фрілансерів буде містити кілька основних таблиць, які зберігатимуть інформацію про користувачів, проєкти, замовлення, відгуки тощо. Основні таблиці та їх поля будуть виглядати наступним чином:

1. **Users (Користувачі)**
    - **user_id** (PRIMARY KEY) — унікальний ідентифікатор користувача.
    - **name** — ім'я користувача.
    - **email** — електронна пошта користувача (унікальна).

- **password_hash** — хеш паролю для забезпечення безпеки.
- **profile_description** — опис профілю користувача.
- **rating** — рейтинг користувача, заснований на відгуках замовників.
- **created_at** — дата та час створення профілю.

2. **Projects (Проєкти)**
   - **project_id** (PRIMARY KEY) — унікальний ідентифікатор проєкту.
   - **title** — назва проєкту.
   - **description** — детальний опис проєкту.
   - **budget** — бюджет, виділений на проєкт.
   - **status** — статус проєкту (наприклад, "відкритий", "в роботі", "завершений").
   - **client_id** (FOREIGN KEY) — посилання на користувача, який створив проєкт.
   - **created_at** — дата та час створення проєкту.
3. **Bids (Пропозиції)**
   - **bid_id** (PRIMARY KEY) — унікальний ідентифікатор пропозиції.
   - **project_id** (FOREIGN KEY) — посилання на проєкт, на який зроблено пропозицію.
   - **freelancer_id** (FOREIGN KEY) — посилання на фрілансера, який подав пропозицію.
   - **amount** — сума, запропонована фрілансером.
   - **created_at** — дата та час подачі пропозиції.
4. **Reviews (Відгуки)**
   - **review_id** (PRIMARY KEY) — унікальний ідентифікатор відгуку.
   - **project_id** (FOREIGN KEY) — посилання на проєкт, до якого належить відгук.
   - **freelancer_id** (FOREIGN KEY) — посилання на фрілансера, якого оцінюють.
   - **client_id** (FOREIGN KEY) — посилання на замовника, який залишив відгук.

- **rating** — рейтинг, присвоєний фрілансеру.
- **comment** — текстовий відгук від замовника.
- **created_at** — дата та час створення відгуку.

5. **Messages (Повідомлення)**
   - **message_id** (PRIMARY KEY) — унікальний ідентифікатор повідомлення.
   - **sender_id** (FOREIGN KEY) — посилання на користувача, який надіслав повідомлення.
   - **receiver_id** (FOREIGN KEY) — посилання на користувача, який отримав повідомлення.
   - **content** — текст повідомлення.
   - **created_at** — дата та час відправки повідомлення.

У базі даних встановлено кілька важливих зв'язків між таблицями, що забезпечують цілісність даних та ефективну роботу порталу:

- Таблиця **Projects** пов'язана з таблицею **Users** через поле **client_id**, яке вказує, хто створив проєкт.
- Таблиця **Bids** має зв'язок із таблицями **Projects** та **Users**, оскільки кожна пропозиція стосується конкретного проєкту і надсилається певним фрілансером.
- Таблиця **Reviews** пов'язує фрілансера, замовника та проєкт, забезпечуючи можливість залишати відгуки після завершення проєкту.
- Таблиця **Messages** використовується для зберігання повідомлень між користувачами, що забезпечує можливість спілкування між замовниками та фрілансерами.

Проектування бази даних з урахуванням таких зв'язків дозволяє забезпечити логічну структуру даних, яка відповідає вимогам порталу та забезпечує зручність доступу до необхідної інформації. Нижче представлена UML-діаграма (див. рис. 2.1), яка демонструє структуру бази даних та зв'язки між основними таблицями. Вона допомагає візуально зрозуміти, як взаємодіють між собою різні елементи бази даних.

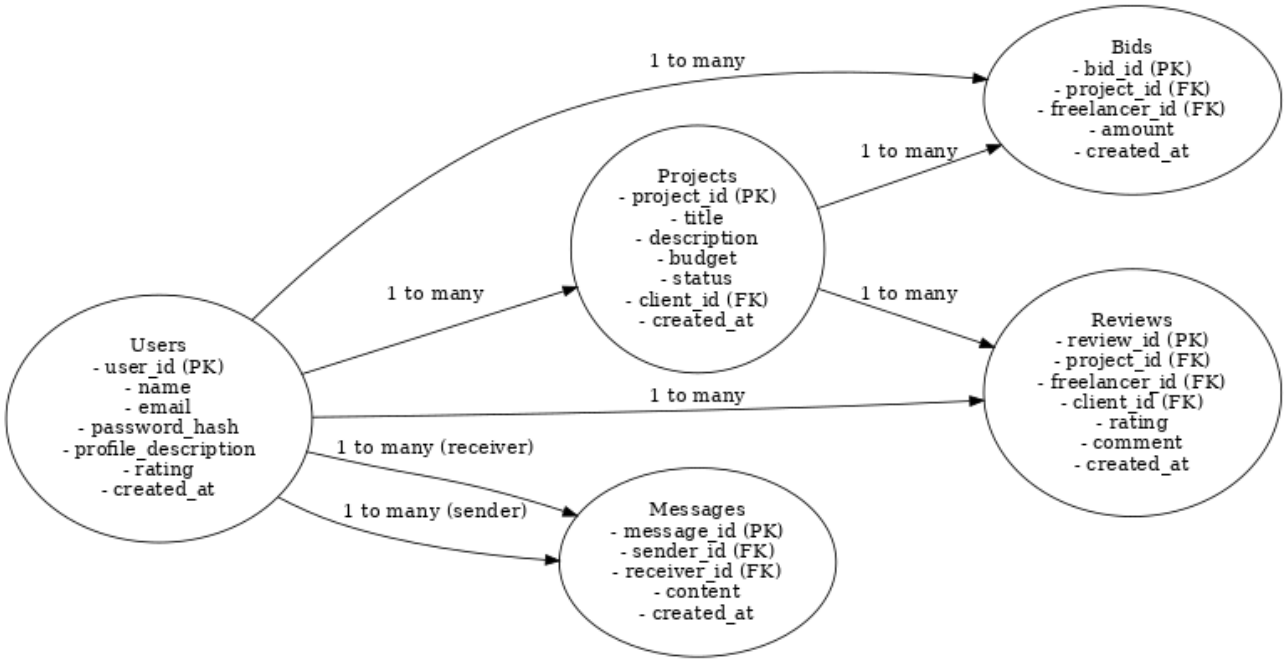

Рисунок 2.1 - Структура бази даних

На UML-діаграмі відображені основні таблиці бази даних і їхні зв'язки, що ілюструє, як дані структуровані та як вони взаємодіють між собою. Ця діаграма допомагає розробникам і аналітикам краще розуміти структуру даних і забезпечити ефективну реалізацію бізнес-логіки системи.

3. ПРАКТИЧНА ЧАСТИНА

3.1. Вибір та обґрунтування стеку технологій

Для розробки веб-платформи для фрілансерів було обрано сучасний, перевірений у комерційних продуктах технологічний стек, який забезпечує масштабованість, безпеку, продуктивність та зручність у підтримці проєкту. Нижче наведено обґрунтування вибору кожного компонента.

Backend: Node.js + Express

Node.js було обрано як основу серверної частини через його асинхронну, подієво-орієнтовану модель, яка ідеально підходить для веб-додатків із великою кількістю одночасних запитів. Express.js — мінімалістичний фреймворк для Node.js, що спрощує створення REST API, дозволяє ефективно організувати маршрутизацію, обробку помилок і middleware.

Frontend: React

React було обрано для реалізації інтерфейсу користувача завдяки його компонентній структурі, високій продуктивності та широкій підтримці спільноти. Використання React забезпечує гнучкість у побудові UI, повторне використання компонентів, інтеграцію зі сторонніми бібліотеками та хорошу підтримку інтерактивності на сторінках.

TypeScript

Використання TypeScript замість звичайного JavaScript дозволяє забезпечити статичну типізацію, що зменшує кількість помилок на етапі розробки, полегшує підтримку коду та покращує досвід роботи в редакторах коду завдяки автодоповненню та перевірці типів.

База даних: PostgreSQL

Як СУБД було обрано PostgreSQL — потужну реляційну систему управління базами даних з підтримкою транзакцій, складних запитів, перевірки цілісності даних та індексації. PostgreSQL забезпечує надійність та продуктивність для зберігання структурованих даних таких, як користувачі, проєкти, пропозиції, повідомлення, відгуки тощо.

ORM: Prisma

Для взаємодії між сервером і базою даних було використано Prisma — сучасний ORM, який дозволяє працювати з БД на основі схем, підтримує міграції, автогенерує типи для TypeScript і спрощує доступ до даних без написання SQL-запитів вручну.

Аутентифікація: JSON Web Tokens (JWT)

JWT було обрано для реалізації безпечної системи аутентифікації. Вона дозволяє створювати токени з обмеженим часом життя, що зберігаються на стороні клієнта, та не вимагає серверного збереження сесій.

Інші технології:

- bcrypt — для шифрування паролів;
- dotenv — для зберігання конфігурацій у середовищах розробки;
- axios / fetch — для HTTP-запитів з фронтенду;
- Tailwind CSS — для швидкого й адаптивного стилізування інтерфейсу;
- Postman — для тестування API під час розробки.

Резюме вибору:

Обраний стек дозволяє створити сучасний, швидкий і масштабований веб-застосунок, який відповідає усім функціональним, технічним і бізнес-вимогам платформи для фрілансерів. Крім того, ці технології добре документовані та активно підтримуються спільнотою, що зменшує ризики під час розробки та обслуговування проєкту.

3.2. Архітектура системи

Архітектура веб-платформи для фрілансерів побудована за принципом розділення клієнтської та серверної частин із чітко визначеними зонами відповідальності. Такий підхід забезпечує масштабованість, модульність і гнучкість системи, а також дозволяє легко впроваджувати нові функції.

Загальна архітектура

Система реалізована у форматі класичної клієнт-серверної моделі:

- Клієнтська частина (frontend) — відповідальна за відображення інтерфейсу, обробку взаємодії з користувачем та передачу запитів до API.
- Серверна частина (backend) — виконує бізнес-логіку, валідацію, обробку запитів, взаємодію з базою даних та сторонніми сервісами.
- База даних — централізоване сховище для всіх структурованих даних (користувачі, проєкти, пропозиції, відгуки, повідомлення тощо).

Схема взаємодії компонентів

Користувач

↓

React (TypeScript) ————→ REST API (Node.js + Express)

↓

Prisma ORM

↓

PostgreSQL Database

Структура backend-додатку

/backend/

```
├── controllers/      // Обробники HTTP-запитів
├── routes/           // REST API маршрути
├── models/           // Опис моделей БД (через Prisma schema)
├── middleware/       // Middleware для перевірки токенів, ролей, логування
├── services/         // Бізнес-логіка (наприклад, підбір фрілансера)
├── config/           // Файли конфігурацій (dotenv, підключення БД)
├── utils/            // Допоміжні функції (хешування, генерація токенів)
└── index.ts          // Точка входу у сервер
```

Структура frontend-додатку

/frontend/

```
├── pages/            // Сторінки (реєстрація, профіль, список проєктів)
├── components/       // UI-компоненти (Navbar, ProjectCard, Modal)
├── services/         // HTTP-запити до API (axios/fetch)
├── hooks/            // Кастомні хуки (auth, API-інтеграції)
```

```
├── context/          // React Context API для глобального стану
├── assets/           // Зображення, шрифти, стилі
└── App.tsx           // Основний компонент
```

Модель ролей та доступів

Система реалізує рольову модель доступу, яка відрізняє права:

- Адміністратор – має повний доступ до всіх модулів;
- Клієнт – може створювати проєкти, переглядати пропозиції, залишати відгуки;
- Фрілансер – може подавати заявки, вести переписку, отримувати рейтинг.

Переваги архітектурного підходу

- Масштабованість – можливість розділення бекенду та фронтенду на окремі сервіси.
- Захищеність – реалізація автентифікації через JWT, обмеження доступу до маршрутів.
- Підтримка та розвиток – модульна структура спрощує внесення змін у майбутньому.
- Можливість автоматичного тестування – логічно відокремлені шари сприяють ефективному юніт- та інтеграційному тестуванню.

3.3. Реалізація базових модулів

У цьому розділі розглянуто реалізацію основних модулів системи, які забезпечують повний життєвий цикл взаємодії користувачів, управління проєктами, подання пропозицій, комунікацію, формування відгуків, пошук і аналітику. Кожен модуль побудований із урахуванням принципів масштабованості, безпеки, модульності та повторного використання коду.

Модуль користувачів (User Module)

Модуль користувачів є базовим компонентом системи, що відповідає за зберігання та обробку персональних даних користувачів усіх типів (адміністратор,

клієнт, фрілансер).

Модель користувача:

```
const userSchema = {
  username: String,
  email: String,
  password: String,
  role: Enum['admin', 'client', 'freelancer'],
  fullName: String,
  bio: String,
  skills: Array,
  rating: Float,
  completedProjects: Integer,
  portfolio: [{ title, description, link, technologies, completionDate }],
  experience: [{ title, company, description, startDate, endDate }],
  education: [{ institution, degree, field, graduationYear }],
  socialLinks: { linkedin, github }
}
```

Політика безпеки:

```
const userSecurity = {
  password: {
    minLength: 8,
    requireSpecialChar: true,
    requireNumber: true
  },
  session: {
    timeout: '24h',
    refreshToken: true
  },
  rateLimit: {
    windowMs: 15 * 60 * 1000,
    max: 100
  }
};
```

Модуль проєктів (Project Module)

Модуль проєктів реалізує створення, редагування, перегляд та зміну статусу проєктів. Дані зберігаються у структурованій формі з категоріями, складністю,

бюджетом та вимогами.

Модель проєкту:

```
const projectSchema = {
  title: String,
  description: String,
  category: String,
  subcategory: String,
  tags: Array,
  budgetMin: Number,
  budgetMax: Number,
  budgetType: String,
  currency: String,
  deadline: Date,
  clientId: ObjectId,
  freelancerId: ObjectId,
  status: Enum['Відкритий', 'В роботі', 'Завершено'],
  requirements: Array,
  skills: Array,
  complexity: Enum['Початковий', 'Середній', 'Експертний'],
  totalAmount: Number
}
```

Валідація проєкту:

```
const projectValidation = {
  title: { minLength: 10, maxLength: 200, required: true },
  budget: { min: 15000, max: 300000, currency: 'UAH' },
  deadline: { minDays: 1, maxDays: 365 }
};
```

Модуль пропозицій (Proposal Module)

Цей модуль забезпечує подання заявок фрілансерами на відкриті проєкти, керування статусами заявок, а також вибір виконавця клієнтом.

API інтерфейси:

```
const proposalEndpoints = {
  create: {
    method: 'POST',
    path: '/api/proposals',
    body: {
```

```
    projectId: 'ObjectId',
    freelancerId: 'ObjectId',
    price: 'Number',
    description: 'String',
    estimatedDuration: 'Number'
   }
  },
  list: {
   method: 'GET',
   path: '/api/proposals',
   query: {
    projectId: 'ObjectId',
    status: 'String'
   }
  },
  accept: {
   method: 'PUT',
   path: '/api/proposals/:id/accept'
  }
};
```

Модуль комунікації (Communication Module)

Обмін повідомленнями між клієнтом і фрілансером реалізується через систему повідомлень, які мають маркер прочитаності та можливість додавання вкладень.

Модель повідомлення:

```
const messageSchema = {
  projectId: ObjectId,
  senderId: ObjectId,
  receiverId: ObjectId,
  content: String,
  createdAt: Date,
  attachments: Array,
  readStatus: Boolean
}
```

Модуль відгуків (Review Module)

Після завершення проєкту сторони можуть залишити взаємні відгуки з

деталізацією за ключовими метриками якості.

Модель відгуку:

```
const reviewSchema = {
  projectId: ObjectId,
  fromUserId: ObjectId,
  toUserId: ObjectId,
  rating: Float,
  comment: String,
  type: Enum['client_to_freelancer', 'freelancer_to_client'],
  createdAt: Date,
  metrics: {
    communication: Number,
    quality: Number,
    expertise: Number,
    deadlines: Number
  }
}
```

Модуль пошуку та фільтрації (Search Module)

Забезпечує швидкий пошук проєктів та користувачів. Використовується індексація полів з різними ваговими коефіцієнтами.

Індексація:

```
const searchIndexes = {
  projects: {
    fields: ['title', 'description', 'category', 'skills'],
    weights: { title: 10, description: 5, category: 3, skills: 8 }
  },
  users: {
    fields: ['username', 'skills', 'bio'],
    weights: { username: 5, skills: 10, bio: 3 }
  }
};
```

Модуль статистики та аналітики (Analytics Module)

Відповідає за збирання, обчислення та зберігання ключових метрик щодо роботи користувачів і проєктів.

Метрики:

```
const analyticsMetrics = {
  projects: {
    successRate: {
      calculation: 'completedProjects / totalProjects',
      period: 'monthly'
    },
    averageCompletion: {
      calculation: 'sum(completionTime) / completedProjects',
      period: 'monthly'
    },
    categoryDistribution: {
      type: 'aggregation',
      groupBy: 'category',
      period: 'all'
    }
  },
  users: {
    activeFreelancers: {
      condition: 'proposals.length > 0',
      period: 'monthly'
    },
    topRated: {
      sort: 'rating',
      limit: 10,
      period: 'all'
    }
  }
};
```

Взаємодія модулів

Усі модулі мають чітко визначені точки взаємодії між собою. Наприклад, при створенні проєкту оновлюється індексація для пошуку та записується подія в аналітику.

```
const moduleInteractions = {
  project: {
    dependencies: ['user', 'proposal', 'review'],
    events: {
```

```
    onCreate: ['notification', 'search_index_update'],
    onStatusChange: ['notification', 'analytics_update'],
    onComplete: ['review_request', 'analytics_update']
   }
  },
  proposal: {
   dependencies: ['project', 'user'],
   events: {
    onCreate: ['notification', 'project_update'],
    onAccept: ['project_update', 'communication_channel_create']
   }
  },
  review: {
   dependencies: ['project', 'user'],
   events: {
    onCreate: ['user_rating_update', 'analytics_update']
   }
  }
};
```

Усі модулі реалізовано згідно з принципами **SOLID**, що забезпечує зручну підтримку, розширення та повторне використання коду в майбутньому. Завдяки чіткій модульній структурі система готова до масштабування, інтеграцій з зовнішніми сервісами та додавання нових ролей чи функцій без порушення загальної логіки платформи.

3.4. Інструкція для користувача

У цьому розділі розглянуто основні етапи використання веб-платформи для фрілансерів з боку зареєстрованих користувачів: замовників і виконавців. Інтерфейс розроблений таким чином, щоб бути зрозумілим та доступним для користувачів із різним рівнем технічної підготовки.

1. Головна сторінка (див. рис. 3.1)

Після входу в систему користувач потрапляє на головну сторінку, де доступна інформація про переваги платформи, кнопки для створення або перегляду проєктів,

а також загальний огляд функціональності.

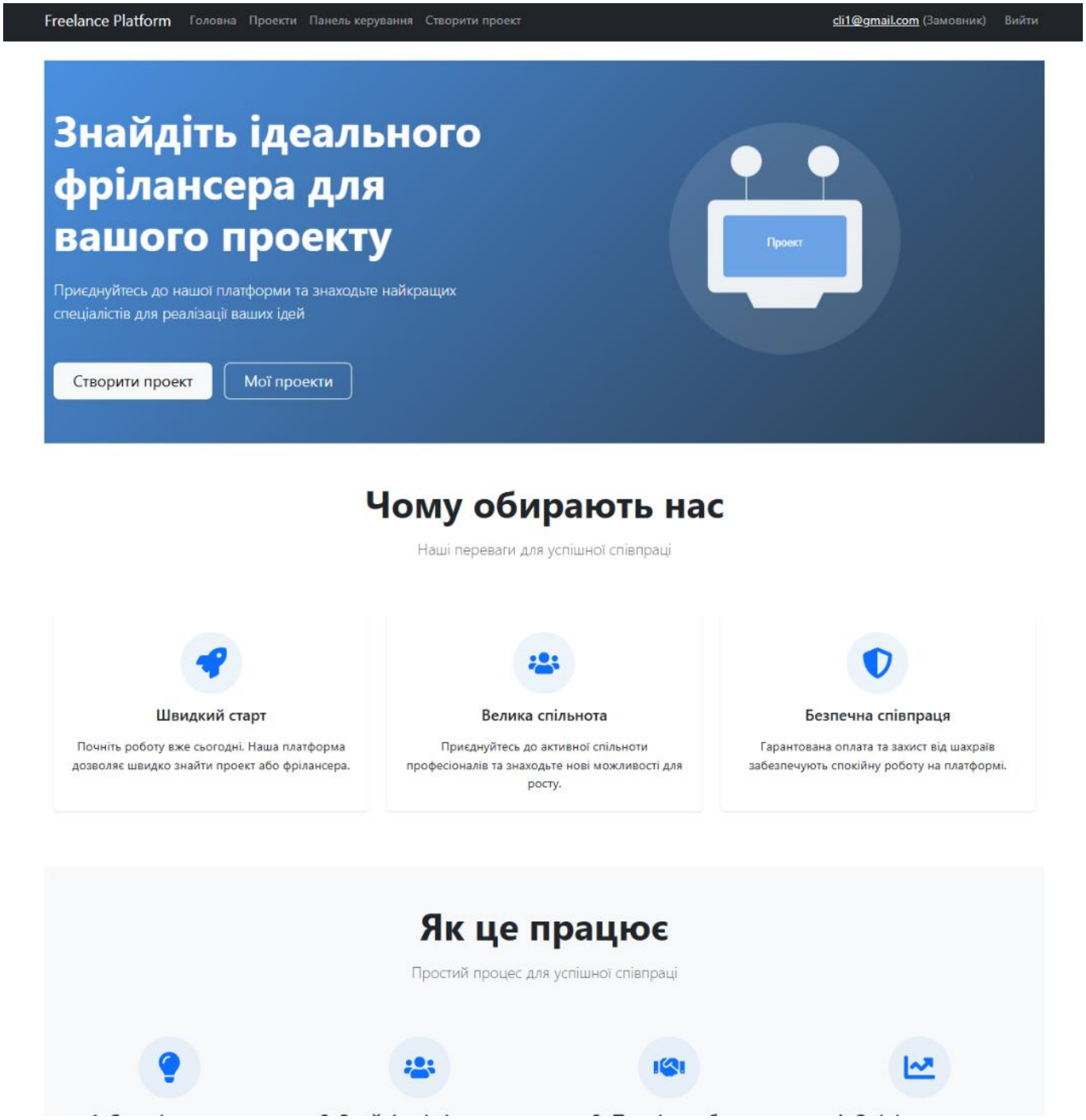

Рисунок 3.1 – Головна сторінка

2. Перегляд списку проєктів (див. рис. 3.2)

На сторінці "Проєкти" відображаються всі активні й завершені проєкти з коротким описом, категорією, бюджетом та статусом. Користувач може переглянути деталі або подати пропозицію.

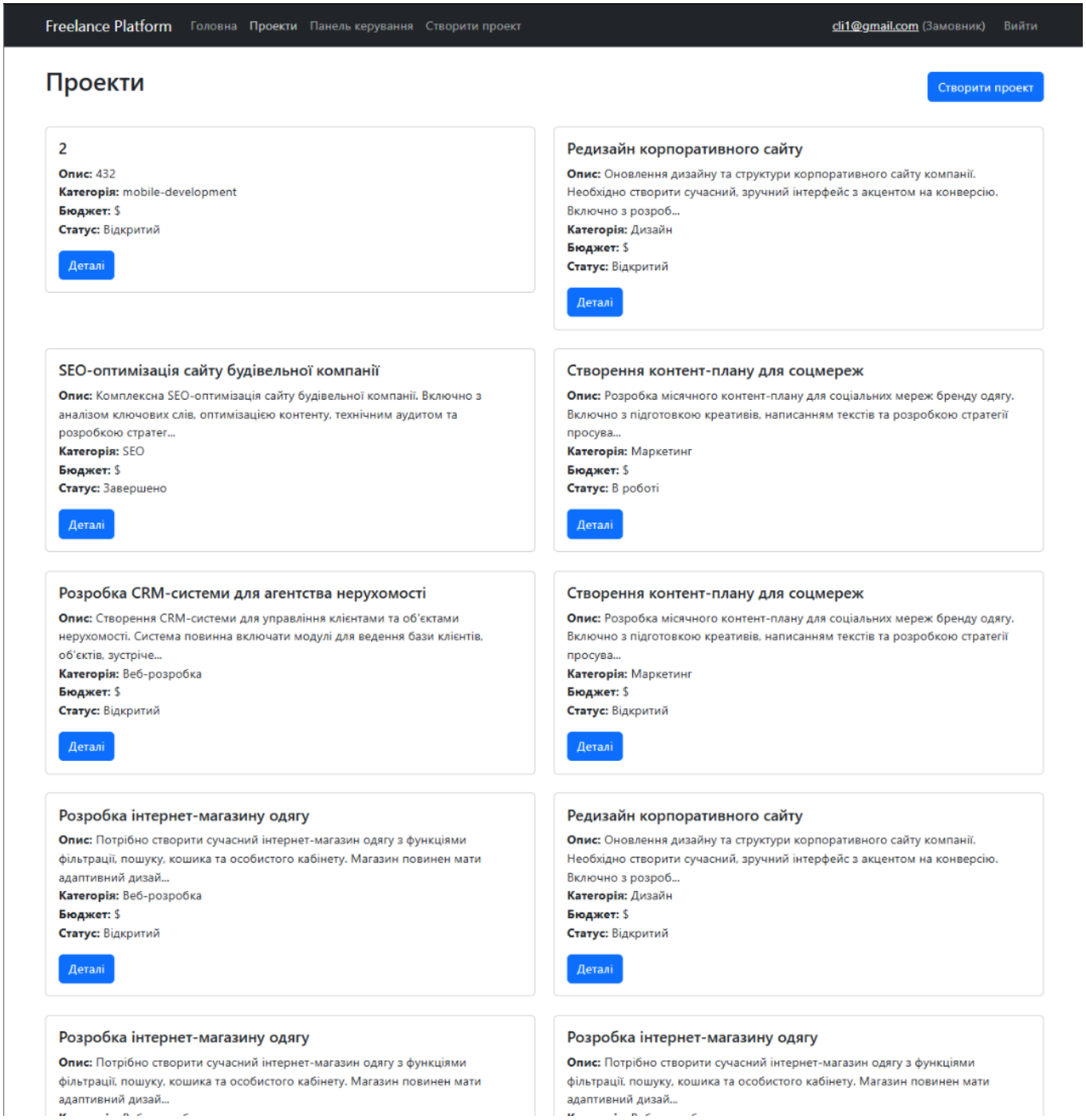

Рисунок 3.2 – Список доступних проєктів

3. Перегляд конкретного проєкту (див. рис. 3.3)

Після натискання кнопки «Деталі» відкривається повна інформація про проєкт: замовник, бюджет, дедлайн, опис, статус. Також доступний чат і блок для подання пропозицій.

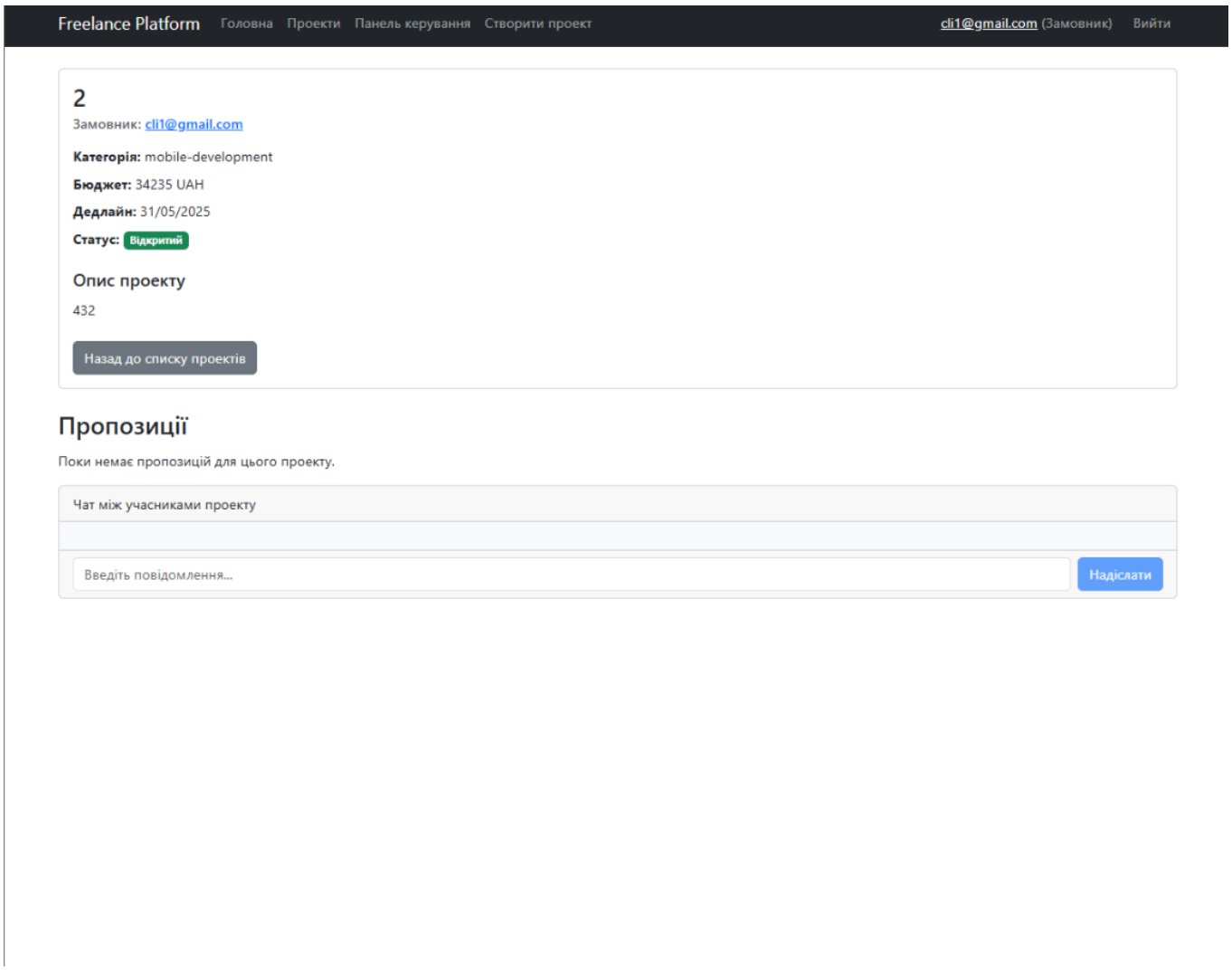

Рисунок 3.3 – Сторінка проєкту з можливістю подачі заявки

4. Панель керування (див. рис. 3.4)

У панелі керування відображається статистика: кількість користувачів, проєктів, активність, загальний дохід, а також таблиця останніх проєктів і рейтинг фрілансерів.

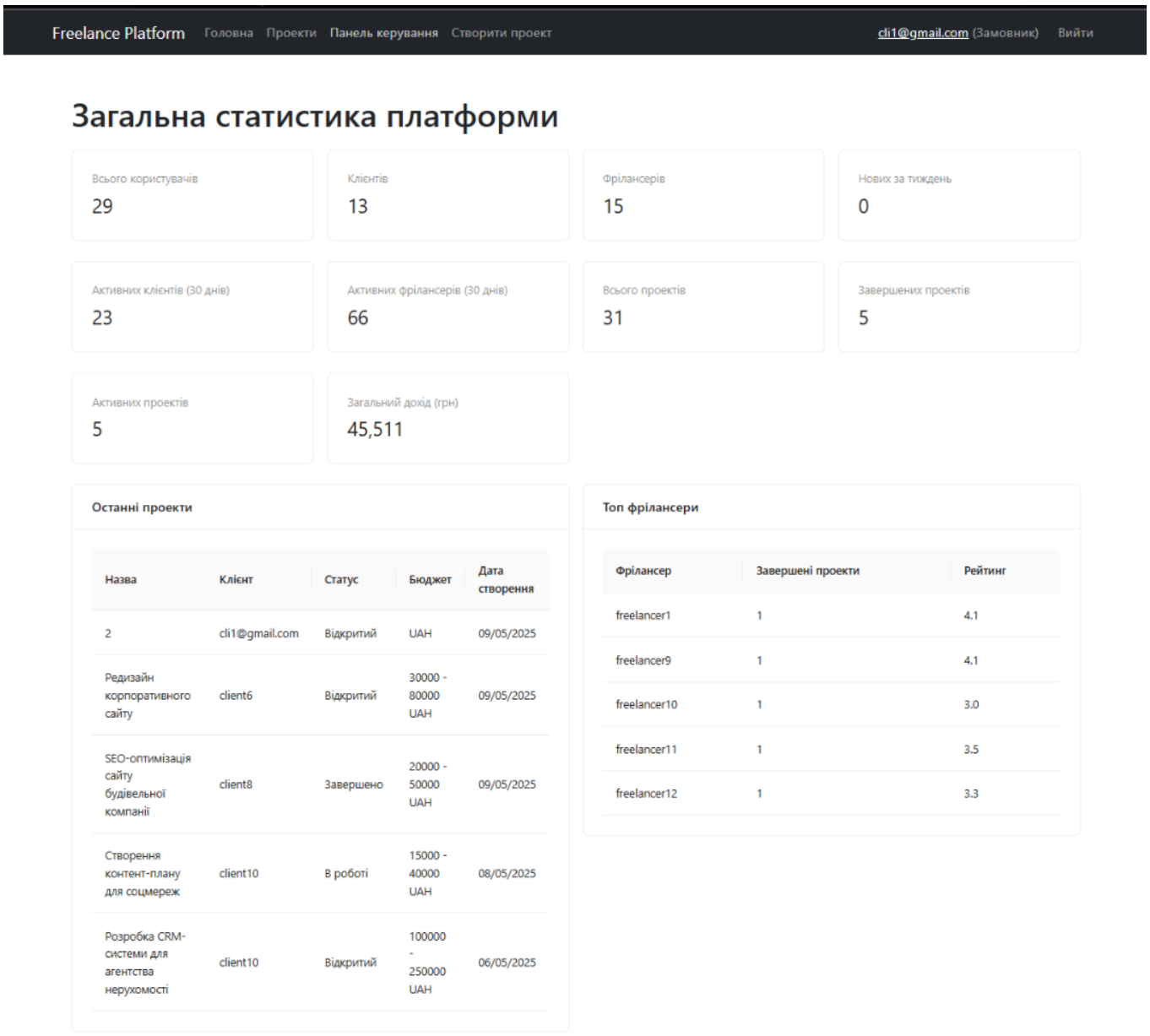

Рисунок 3.4 – Адмін-панель статистики платформи

5. Створення нового проєкту (див. рис. 3.5)

Замовник має змогу створити новий проєкт, заповнивши форму: назва, опис, бюджет, дедлайн та категорія. Після заповнення форма надсилається на сервер і відображається в загальному списку.

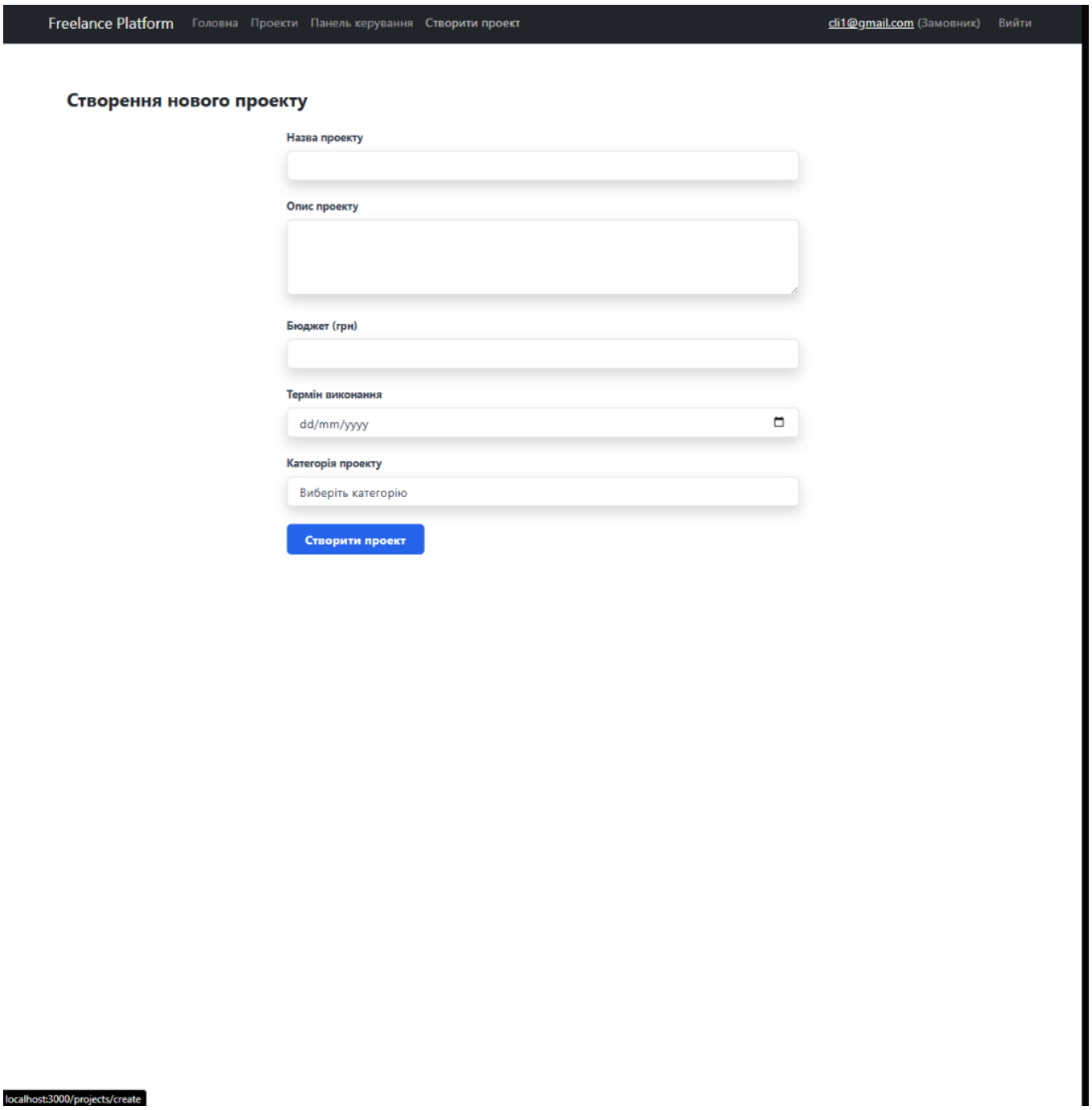

Рисунок 3.5 – Форма створення проєкту

6. Профіль фрілансера (див. рис. 3.6)

У профілі фрілансера відображаються персональні дані, навички, завершені проєкти, рейтинг, а також список поточних пропозицій. Доступний список відгуків замовників.

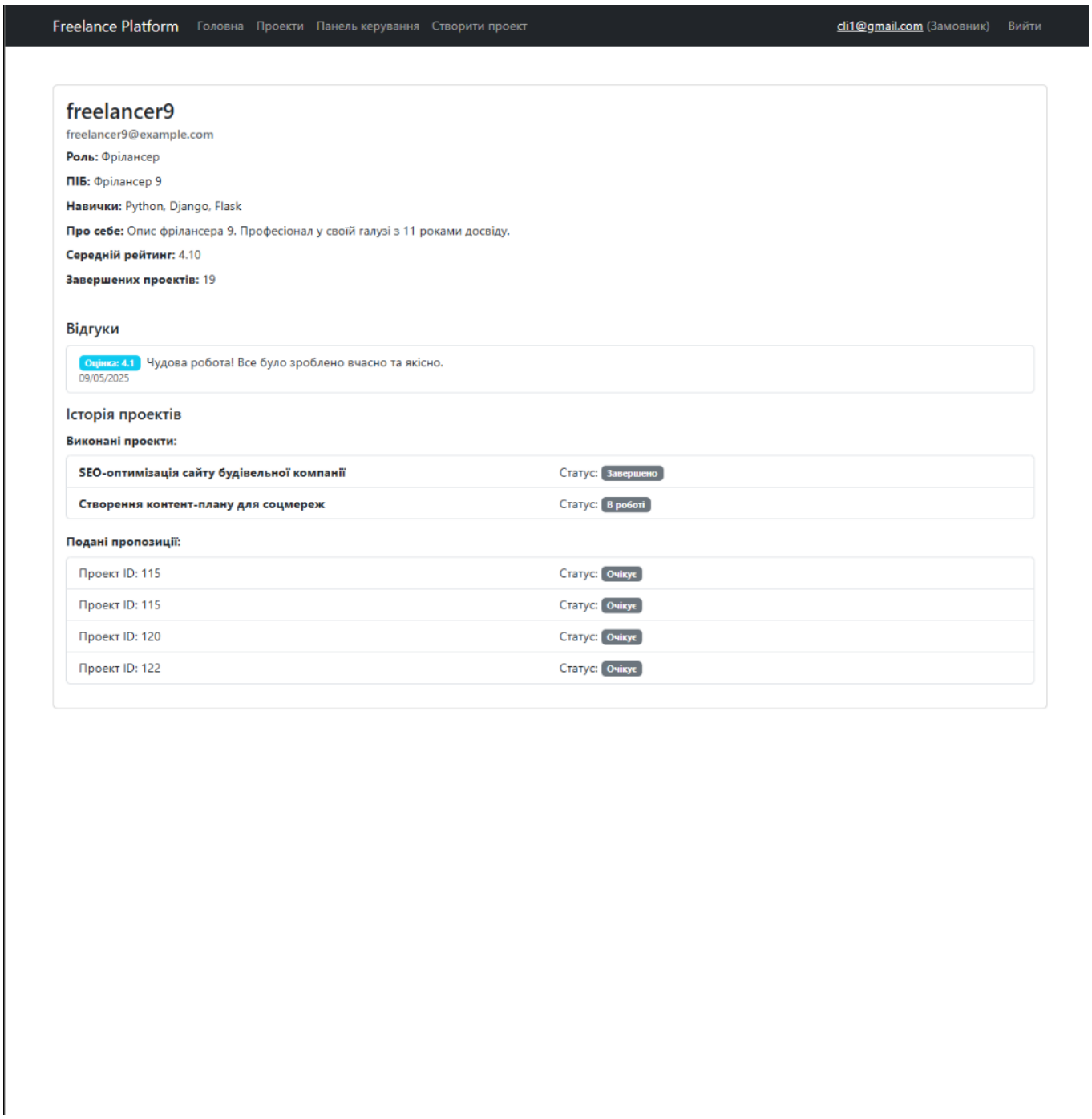

Рисунок 3.6 – Профіль фрілансера

7. Профіль замовника (див. рис. 3.7)

Профіль замовника містить його інформацію, створені проєкти зі статусами та відгуки від виконавців. Це дозволяє фрілансерам оцінювати надійність потенційного клієнта.

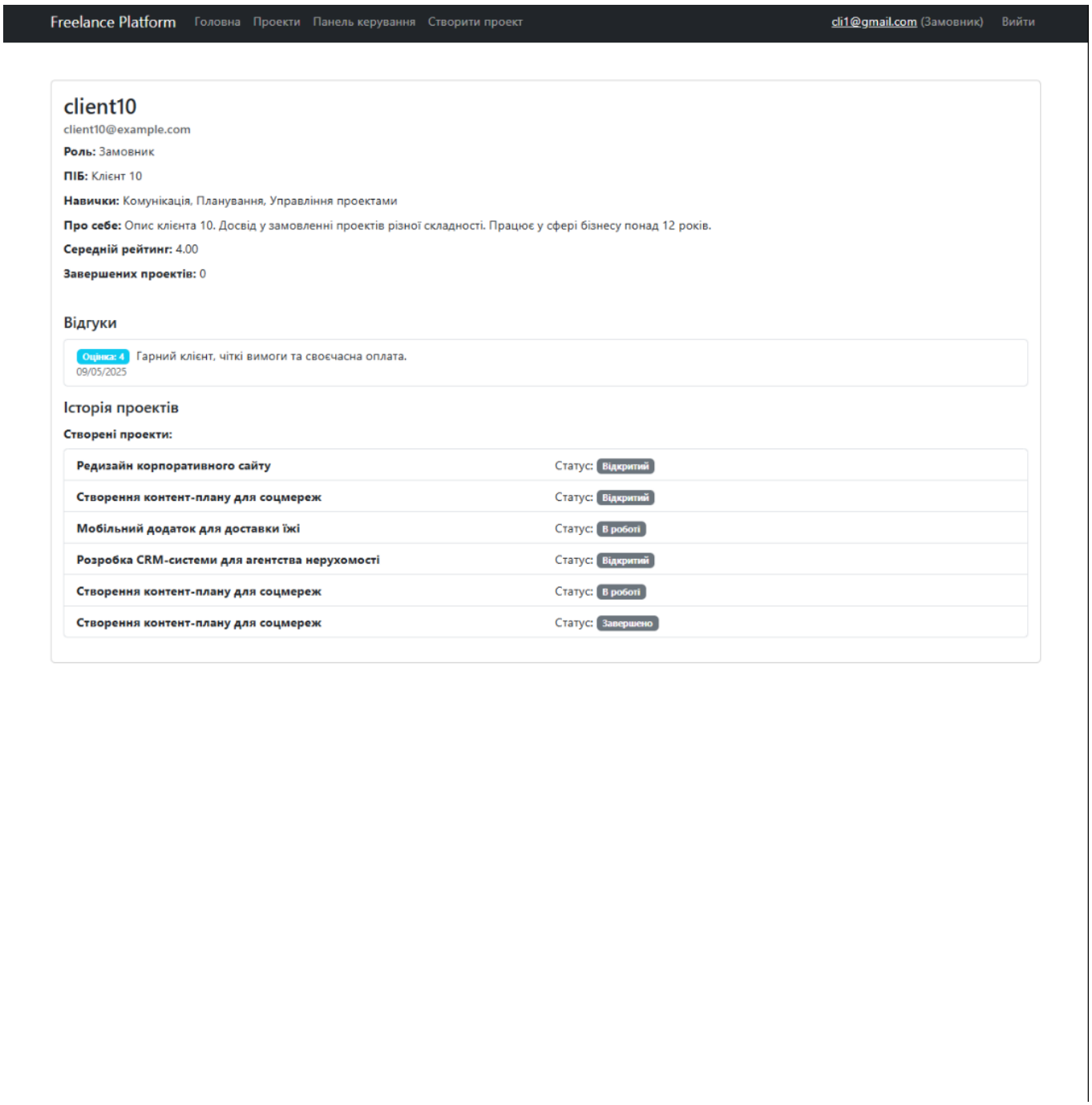

Рисунок 3.7 – Профіль замовника

ВИСНОВКИ

У роботі було спроєктовано та реалізовано повнофункціональну веб-платформу для взаємодії між замовниками та фрілансерами, яка охоплює повний життєвий цикл співпраці: від публікації проєкту до його завершення й оцінки результатів. Розроблена система поєднує сучасні інструменти керування проєктами, модулі комунікації, рейтингову систему та аналітичний блок.

У процесі реалізації було досягнуто наступних результатів:

- Проведено інформаційний огляд ринку фріланс-платформ, визначено ключові вимоги цільової аудиторії та сформульовано архітектурну модель системи.
- Обґрунтовано вибір технологічного стеку: **React**, **Node.js**, **TypeScript**, **Express**, **MongoDB** — для забезпечення масштабованості, зручності підтримки та швидкості розробки.
- Реалізовано основні функціональні модулі: реєстрацію та авторизацію користувачів, створення та керування проєктами, подання пропозицій, комунікацію в чаті, систему відгуків і рейтингу.
- Забезпечено безпеку даних користувачів шляхом шифрування паролів, реалізації ролей і обмеження запитів.
- Створено модулі пошуку, фільтрації та аналітики, що дозволяють ефективно відстежувати активність, дохід, популярні категорії та успішність виконавців.
- Інтерфейс розроблено відповідно до принципів UX-дизайну з урахуванням доступності, простоти та адаптивності.
- Реалізовано адміністративну панель для моніторингу ключових метрик системи.
- Створено докладну інструкцію користувача з прикладами використання та скріншотами, що полегшує входження нових користувачів у систему.

Практичне значення реалізованої системи полягає у наданні повноцінного програмного інструменту для автоматизації пошуку виконавців, організації віддаленої роботи та керування фріланс-проєктами. Платформа може бути використана як самостійне SaaS-рішення або інтегрована в корпоративні екосистеми для управління зовнішніми підрядниками.

СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Офіційна документація React [Електронний ресурс] – Режим доступу: React
2. Офіційна документація Node.js [Електронний ресурс] – Режим доступу: https://nodejs.org/en/docs/
3. Офіційна документація PostgreSQL [Електронний ресурс] – Режим доступу: https://www.postgresql.org/docs/
4. React + Node.js + Express + PostgreSQL example: Build a CRUD App [Електронний ресурс] – Режим доступу: Bezkoder
5. React Підручник. Уроки для початківців. W3Schools українською [Електронний ресурс] – Режим доступу: W3Schools UA
6. Веб-розробка з Python і Django для початківців | Віталій Подоба [Електронний ресурс] – Режим доступу: Віталій Подоба
7. Книги для веб-дизайнерів і розробників сайтів [Електронний ресурс] – Режим доступу: Arthuss
8. Книги з програмування українською – ТОП література для програмістів [Електронний ресурс] – Режим доступу: ГуруКниги
9. React + Node.js + PostgreSQL: CRUD example - DEV Community [Електронний ресурс] – Режим доступу: Dev.to
10. Getting started with Postgres in your React app [Електронний ресурс] – Режим доступу: LogRocket Blog
11. Ольховська, О. В., Черненко, О. О. Методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп'ютерні науки освітня програма «Комп'ютерні науки» ступеня бакалавра. Полтава: ПУЕТ, 2024. – 67 с. – 1 електрон. опт. диск (CD-ROM).

ДОДАТОК А.

```javascript
// User.js
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const userSchema = new mongoose.Schema({
  username: {
    type: String,
    required: true,
    unique: true,
    trim: true
  },
  email: {
    type: String,
    required: true,
    unique: true,
    lowercase: true
  },
  password: {
    type: String,
    required: true,
    minlength: 8
  },
  role: {
    type: String,
    enum: ['admin', 'client', 'freelancer'],
    required: true
  },
  fullName: String,
  bio: String,
  skills: [String],
  rating: {
    type: Number,
    default: 0,
    min: 0,
    max: 5
  },
```

```
  completedProjects: {
    type: Number,
    default: 0
  },
  portfolio: [{
    title: String,
    description: String,
    link: String,
    technologies: [String],
    completionDate: Date
  }],
  experience: [{
    title: String,
    company: String,
    description: String,
    startDate: Date,
    endDate: Date
  }],
  education: [{
    institution: String,
    degree: String,
    field: String,
    graduationYear: Number
  }],
  socialLinks: {
    linkedin: String,
    github: String
  },
  createdAt: {
    type: Date,
    default: Date.now
  }
});

userSchema.pre('save', async function(next) {
  if (!this.isModified('password')) return next();
  this.password = await bcrypt.hash(this.password, 12);
  next();
```

```
});

userSchema.methods.checkPassword = async function(candidatePassword) {
  return await bcrypt.compare(candidatePassword, this.password);
};

module.exports = mongoose.model('User', userSchema);

// Project.js
const mongoose = require('mongoose');

const projectSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
    minlength: 10,
    maxlength: 200
  },
  description: {
    type: String,
    required: true
  },
  category: {
    type: String,
    required: true
  },
  subcategory: String,
  tags: [String],
  budgetMin: {
    type: Number,
    required: true,
    min: 15000
  },
  budgetMax: {
    type: Number,
    required: true,
    max: 300000
  },
```

```
budgetType: {
  type: String,
  enum: ['Фіксована', 'Погодинна'],
  default: 'Фіксована'
},
currency: {
  type: String,
  default: 'UAH'
},
deadline: {
  type: Date,
  required: true
},
clientId: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'User',
  required: true
},
freelancerId: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'User'
},
status: {
  type: String,
  enum: ['Відкритий', 'В роботі', 'Завершено'],
  default: 'Відкритий'
},
requirements: [String],
skills: [String],
complexity: {
  type: String,
  enum: ['Початковий', 'Середній', 'Експертний'],
  required: true
},
totalAmount: Number,
createdAt: {
  type: Date,
  default: Date.now
```

```
  }
});

projectSchema.index({ title: 'text', description: 'text', skills: 'text' });

projectSchema.virtual('proposals', {
  ref: 'Proposal',
  localField: '_id',
  foreignField: 'projectId'
});

module.exports = mongoose.model('Project', projectSchema);

// Proposal.js
const mongoose = require('mongoose');

const proposalSchema = new mongoose.Schema({
  projectId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Project',
    required: true
  },
  freelancerId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  price: {
    type: Number,
    required: true
  },
  description: {
    type: String,
    required: true
  },
  status: {
    type: String,
    enum: ['Очікує', 'Прийнято', 'Відхилено'],
```

```
    default: 'Очікує'
  },
  estimatedDuration: {
    type: Number,
    required: true
  },
  createdAt: {
    type: Date,
    default: Date.now
  }
});

module.exports = mongoose.model('Proposal', proposalSchema);

// ProjectController.js
const Project = require('../models/Project');
const User = require('../models/User');
const ApiError = require('../utils/apiError');

exports.createProject = async (req, res, next) => {
  try {
    const project = await Project.create({
      ...req.body,
      clientId: req.user.id
    });
    res.status(201).json({
      status: 'success',
      data: { project }
    });
  } catch (err) {
    next(err);
  }
};

exports.getAllProjects = async (req, res, next) => {
  try {
    const projects = await Project.find()
      .populate('clientId', 'username fullName rating')
```

```
      .populate('freelancerId', 'username fullName rating');
    res.status(200).json({
      status: 'success',
      results: projects.length,
      data: { projects }
    });
  } catch (err) {
    next(err);
  }
};

exports.getProject = async (req, res, next) => {
  try {
    const project = await Project.findById(req.params.id)
      .populate('clientId')
      .populate('freelancerId')
      .populate('proposals');
    if (!project) {
      return next(new ApiError('Проект не знайдено', 404));
    }
    res.status(200).json({
      status: 'success',
      data: { project }
    });
  } catch (err) {
    next(err);
  }
};

exports.updateProject = async (req, res, next) => {
  try {
    const project = await Project.findByIdAndUpdate(
      req.params.id,
      req.body,
      {
        new: true,
        runValidators: true
      }
```

```
    );
    if (!project) {
      return next(new ApiError('Проект не знайдено', 404));
    }
    res.status(200).json({
      status: 'success',
      data: { project }
    });
  } catch (err) {
    next(err);
  }
};

exports.deleteProject = async (req, res, next) => {
  try {
    const project = await Project.findByIdAndDelete(req.params.id);
    if (!project) {
      return next(new ApiError('Проект не знайдено', 404));
    }
    res.status(204).json({
      status: 'success',
      data: null
    });
  } catch (err) {
    next(err);
  }
};

// UserController.js
const User = require('../models/User');
const jwt = require('jsonwebtoken');
const ApiError = require('../utils/apiError');

const signToken = id => {
  return jwt.sign({ id }, process.env.JWT_SECRET, {
    expiresIn: process.env.JWT_EXPIRES_IN
  });
};
```

```
exports.signup = async (req, res, next) => {
  try {
    const newUser = await User.create({
      username: req.body.username,
      email: req.body.email,
      password: req.body.password,
      role: req.body.role,
      fullName: req.body.fullName
    });
    const token = signToken(newUser._id);
    res.status(201).json({
      status: 'success',
      token,
      data: { user: newUser }
    });
  } catch (err) {
    next(err);
  }
};

exports.login = async (req, res, next) => {
  try {
    const { email, password } = req.body;
    if (!email || !password) {
      return next(new ApiError('Будь ласка, вкажіть email та пароль', 400));
    }
    const user = await User.findOne({ email }).select('+password');
    if (!user || !(await user.checkPassword(password))) {
      return next(new ApiError('Невірний email або пароль', 401));
    }
    const token = signToken(user._id);
    res.status(200).json({
      status: 'success',
      token
    });
  } catch (err) {
    next(err);
```

```
  }
};

exports.updateProfile = async (req, res, next) => {
  try {
    const updatedUser = await User.findByIdAndUpdate(
      req.user.id,
      {
        fullName: req.body.fullName,
        bio: req.body.bio,
        skills: req.body.skills,
        portfolio: req.body.portfolio,
        experience: req.body.experience,
        education: req.body.education,
        socialLinks: req.body.socialLinks
      },
      {
        new: true,
        runValidators: true
      }
    );
    res.status(200).json({
      status: 'success',
      data: { user: updatedUser }
    });
  } catch (err) {
    next(err);
  }
};

// projectRoutes.js
const express = require('express');
const projectController = require('../controllers/projectController');
const authMiddleware = require('../middleware/auth');

const router = express.Router();

router.use(authMiddleware.protect);
```

```
router
  .route('/')
  .get(projectController.getAllProjects)
  .post(
    authMiddleware.restrictTo('client'),
    projectController.createProject
  );

router
  .route('/:id')
  .get(projectController.getProject)
  .patch(
    authMiddleware.restrictTo('client'),
    projectController.updateProject
  )
  .delete(
    authMiddleware.restrictTo('client', 'admin'),
    projectController.deleteProject
  );

module.exports = router;

// auth.js
const jwt = require('jsonwebtoken');
const User = require('../models/User');
const ApiError = require('../utils/apiError');

exports.protect = async (req, res, next) => {
  try {
    let token;
    if (
      req.headers.authorization &&
      req.headers.authorization.startsWith('Bearer')
    ) {
      token = req.headers.authorization.split(' ')[1];
    }
    if (!token) {
```

```
      return next(new ApiError('Будь ласка, увійдіть в систему', 401));
    }
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    const user = await User.findById(decoded.id);
    if (!user) {
      return next(new ApiError('Користувача не знайдено', 401));
    }
    req.user = user;
    next();
  } catch (err) {
    next(err);
  }
};

exports.restrictTo = (...roles) => {
  return (req, res, next) => {
    if (!roles.includes(req.user.role)) {
      return next(new ApiError('У вас немає прав для цієї дії', 403));
    }
    next();
  };
};

// apiError.js
class ApiError extends Error {
  constructor(message, statusCode) {
    super(message);
    this.statusCode = statusCode;
    this.status = `${statusCode}`.startsWith('4') ? 'fail' : 'error';
    this.isOperational = true;
    Error.captureStackTrace(this, this.constructor);
  }
}

module.exports = ApiError;

// validation.js
const validator = require('validator');
```

```
exports.validateProject = project => {
  const errors = [];
  if (!project.title || project.title.length < 10) {
    errors.push('Назва проекту повинна містити мінімум 10 символів');
  }
  if (!project.description) {
    errors.push('Опис проекту обов\'язковий');
  }
  if (project.budgetMin < 15000 || project.budgetMax > 300000) {
    errors.push('Бюджет повинен бути в межах від 15,000 до 300,000 UAH');
  }
  if (!validator.isDate(project.deadline)) {
    errors.push('Невірний формат дати дедлайну');
  }
  return errors;
};

exports.validateUser = user => {
  const errors = [];
  if (!validator.isEmail(user.email)) {
    errors.push('Невірний формат email');
  }
  if (!user.password || user.password.length < 8) {
    errors.push('Пароль повинен містити мінімум 8 символів');
  }
  if (!['admin', 'client', 'freelancer'].includes(user.role)) {
    errors.push('Невірна роль користувача');
  }
  return errors;
};
```