Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Бакалавр
(назва освітнього ступеня)

на тему: Розробка інформаційної системи для надання фріланс послуг з використанням фреймворків React, Node.js

Виконав(ла): студент(ка) 4 курсу, групи СП-41
спеціальності 121
«Інженерія програмного забезпечення»
(шифр і назва спеціальності)

_____ Денега Т.І.
(підпис) (прізвище та ініціали)

Керівник _____ Петрик М. Р.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю. М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М. Р.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль 2025

АНОТАЦІЯ

Кваліфікаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2025. Сторінок 65, рисунків 26, додатків 3, презентація.

Тема: Розробка інформаційної системи для надання фріланс послуг з використанням фреймворків React, Node.js та MongoDB.

Кваліфікаційна робота бакалавра присвячена розробці сучасної інформаційної системи для організації фріланс послуг на основі стека MERN (MongoDB, Express.js, React, Node.js). Метою роботи є створення ефективної платформи, яка забезпечує взаємодію між замовниками та виконавцями, автоматизацію процесів пошуку роботи, управління проектами та оплати послуг.

У першому розділі виконано аналіз предметної області, вивчено вимоги до функціоналу платформи та її користувачів. Проведено обґрунтування вибору технологій: React – для реалізації інтерфейсу користувача, Node.js і Express – для обробки запитів та побудови REST API, MongoDB – для зберігання структурованих даних.

У другому розділі детально описано архітектуру платформи, особливості реалізації функціоналу: реєстрація й авторизація користувачів, створення та редагування блогів, сповіщення та оплата замовлень. Окрему увагу приділено тестуванню функціональності, зокрема автоматизованому тестуванню API через Postman.

Об'єктом дослідження є сучасна веб-платформа для фріланс послуг та нереляційна база даних MongoDB.

Предметом дослідження є інструменти та методи розробки веб-додатків для блогінгу з використанням стека MERN (MongoDB, Express.js, React, Node.js).

Ключові слова: веб-платформа, MongoDB, React, Node.js, Express.js, база даних, REST API, користувач, коментарі, створення контенту, фул-стек розробка.

ABSTRACT

Bachelor's Qualification Work. Ternopil Ivan Puluj National Technical University, Department of Software Engineering, Specialty 121 "Software Engineering". TNTU, 2025. Pages: 65, Figures: 26, Appendices: 3, Presentation.

Title: Development of an Information System for Freelance Services Using React, Node.js, and MongoDB Frameworks.

The bachelor's qualification work is dedicated to the development of a modern information system for organizing freelance services based on the MERN stack (MongoDB, Express.js, React, Node.js). The aim of the work is to create an efficient platform that enables interaction between customers and freelancers, automates job search processes, project management, and payment of services.

The first chapter includes the analysis of the subject area, definition of platform functionality requirements and user needs. The rationale for choosing technologies is provided: React – for implementing the user interface, Node.js and Express – for request handling and building the REST API, MongoDB – for storing structured data.

The second chapter describes the architecture of the platform in detail, the implementation features of core functionalities: user registration and authentication, blog creation and editing, notification system, and payment processing. Particular attention is paid to functionality testing, especially automated API testing using Postman.

The object of the study is a modern web platform for freelance services and the non-relational MongoDB database.

The subject of the study is the tools and methods of developing web applications for blogging using the MERN stack (MongoDB, Express.js, React, Node.js).

Keywords: web platform, blog, MongoDB, React, Node.js, Express.js, authentication, database, REST API, user, comments, content creation, interactive interface, full-stack development.

ЗМІСТ

ВСТУП

В умовах стрімкого розвитку цифрових технологій фріланс став однією з най динамічніших складових світової економіки. Індивідуальні фахівці, стартапи та великі компанії все частіше використовують онлайн-платформи для пошуку виконавців або реалізації власних професійних навичок. У зв’язку з цим виникає потреба у створенні сучасних, безпечних та зручних інформаційних систем, які спрощують взаємодію між замовниками та фрілансерами з використанням сучасних фреймворків React, Node.js та MongoDB.

Метою розробки даної фріланс-платформи є створення сучасного, технологічно просунутого середовища для ефективної взаємодії між замовниками та виконавцями. Використовуючи потужні можливості стеку MERN (MongoDB, Express.js, React, Node.js), система забезпечує швидкий, безпечний та зручний спосіб пошуку фахівців, обговорення умов співпраці та проведення транзакцій. Ця платформа не просто автоматизує процес пошуку фрілансерів, а створює цілісний екосистему для професійної співпраці, де кожен користувач - незалежно від того, чи він шукає виконавця чи пропонує свої послуги - може знайти оптимальні умови для реалізації своїх цілей. Інноваційний підхід до побудови фріланс-ринку робить цю систему унікальним рішенням у своїй категорії.

React став основним інструментом для створення інтерфейсу нашої фріланс-платформи завдяки своїй унікальній здатності поєднувати продуктивність із розробничою гнучкістю. Використання віртуального DOM дозволяє системі миттєво реагувати на дії користувачів – чи то публікація нового проекту, оновлення профілю чи активна переписка у чаті. Специфіка фріланс-ринку (необхідність швидкої реакції, персоналізація, робота з великими обсягами динамічного контенту) ідеально узгоджується з технологічними перевагами React, роблячи його оптимальним вибором для нашого проекту

Node.js — це середовище виконання JavaScript, яке базується на подієво-орієнтованій моделі та забезпечує ефективну розробку масштабованих мережевих

застосунків. Однією з його ключових переваг є можливість використання однієї мови програмування як на клієнтському, так і на серверному боці, що сприяє спрощенню процесу розробки значно спрощує процес розробки та інформаційної системи, дозволяє здійснювати швидкий обмін даними між клієнтською і серверною частинами.

MongoDB як документно-орієнтовану NoSQL-базу даних було обрано завдяки її здатності ефективно працювати з неструктурованими й напівструктурованими даними, а також можливості масштабування при зростанні обсягів інформації. Її формат зберігання, що базується на JSON-подібних документах, забезпечує зручну взаємодію з JavaScript-орієнтованими веб-застосунками та полегшує інтеграцію.

Основні етапи створення інформаційної системи для організації фріланс-послуг включали проєктування архітектури з урахуванням вимог до продуктивності, безпеки, а також реалізацію розмежування прав доступу між різними категоріями користувачів. На подальшому етапі реалізовано базовий функціонал системи, зокрема реєстрацію та автентифікацію користувачів, створення й управління проєктами, подання заявок фрілансерами, а також У процесі розробки активно застосовувалися методики гнучкої розробки (Agile) з метою поетапної реалізації системи, ефективного тестування та своєчасного врахування змін у вимогах до функціоналу.

У процесі створення веб-платформи велика увага приділялася тестуванню та стабільності роботи застосунку. Було застосовано різні підходи до перевірки якості, , інтеграційні перевірки та тестування взаємодії з інтерфейсом користувача. Для автоматизації тестових сценаріїв використовувалися інструменти, сумісні з Node.js для серверної частини та React Testing Library для фронтенд-компонентів.

Етап впровадження системи охоплював розгортання веб-додатку на хмарній платформі AWS із попереднім налаштуванням середовища. Також було реалізовано механізми безперервної інтеграції та розгортання (CI/CD), що забезпечує автоматизоване тестування, збирання коду й оновлення програми за допомогою сучасних інструментів DevOps.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ СИСТЕМИ

1.1 Аналіз предметної області інформаційної системи для фріланс послуг

У сучасному цифровому середовищі онлайн-платформи для фрілансу відіграють важливу роль у забезпеченні взаємодії між замовниками та виконавцями послуг. Із розширенням можливостей Інтернету та зростанням популярності дистанційної роботи такі системи стали не лише інструментом для пошуку роботи чи виконавців, створення партнерств та формування цифрової економіки. Вони відкривають нові можливості для самореалізації, монетизації навичок та побудови власного кар’єрного шляху незалежно від географічного розташування користувача.

Проте існуючі фріланс-платформи часто мають обмежену функціональність, складний інтерфейс або недостатній рівень безпеки та конфіденційності для користувачів. Деякі з них не враховують потреби окремих категорій користувачів, не підтримують гнучку систему управління проєктами або мають обмеження щодо інтеграції з платіжними сервісами.

Через це виникає потреба у розробці сучасних веб-платформ, які поєднують функціональність, простоту використання, адаптивний інтерфейс та високий рівень захисту персональних даних. Застосування сучасних фреймворків, таких як React і Node.js, дозволяє створювати ефективні та масштабовані рішення, здатні задовольнити потреби як замовників, так і виконавців фріланс-послуг.

У зв’язку з цим створення новітньої веб-платформи для фріланс-послуг на основі прогресивних технологій, таких як React, Node.js і MongoDB, є важливим і своєчасним завданням. Така система покликана задовольнити сучасні запити користувачів, забезпечуючи зручну, захищену та ефективну цифрову екосистему для взаємодії між клієнтами та виконавцями.

Щоб сформувати конкурентні переваги для власного програмного продукту, доцільно провести детальний аналіз ринку та існуючих рішень. Такий аналіз дозволяє виявити потреби цільової аудиторії, оцінити сильні та слабкі

сторони аналогічних сервісів, а також окреслити напрями, в яких можливо вдосконалити функціональність. Завдяки цьому розробник може не лише уникнути типових помилок конкурентів, але й запропонувати більш адаптоване та цінне рішення для користувачів.

Однією з найвідоміших сучасних платформ для фрілансерів є Upwork – онлайн-маркетплейс, який об'єднує замовників і виконавців з усього світу. Платформа вирізняється широким функціоналом для розміщення проєктів, фільтрації виконавців, ведення переговорів і здійснення оплат. Зручний інтерфейс і можливість виставляти погодинну або фіксовану оплату робить її привабливою як для початківців, так і для досвідчених спеціалістів.

Серед основних переваг Upwork можна виділити наявність системи захисту платежів (Escrow), рейтингової системи та аналітики активності користувача. Водночас, платформа має й суттєві недоліки: висока комісія з боку виконавців, обмеження на безкоштовні заявки та сувора модерація, яка іноді може призводити до блокування акаунтів без пояснення причин.

Іншим популярним ресурсом є Freelancer.com, який має подібний набір функцій, але більш відкриту систему участі в тендерах на проєкти. Платформа дозволяє створювати детальні профілі, портфоліо та брати участь у конкурсах. До позитивних сторін можна віднести гнучку систему ставок, широкий вибір категорій завдань і доступ до міжнародного ринку. Проте недоліками є велика кількість неконкурентоспроможних проєктів і перевантаження системи малозначущими пропозиціями, що ускладнює відбір якісних замовлень.

Також варто згадати Fiverr – платформу, яка реалізує підхід «робота за фіксовану суму». На відміну від Upwork чи Freelancer, де виконавці змагаються за проєкти, Fiverr орієнтований на створення виконавцем пропозиції (гігу), яку замовники можуть придбати. Основні переваги – швидкість замовлення, простота взаємодії та великий ринок готових послуг. Серед недоліків – висока конкуренція, стандартизовані ціни та обмеженість у гнучкості налаштування умов співпраці.

Проведений аналіз дозволяє визначити, що незважаючи на широкі можливості сучасних платформ, у них все ще наявні певні обмеження щодо

персоналізації, гнучкості проєктного менеджменту, комісійної політики та відкритості до нових користувачів. Це створює передумови для розробки нової фріланс-системи, яка зможе врахувати ці недоліки та реалізувати вдосконалений користувацький досвід.

1.2 Формування вимог до веб-платформи фріланс послуг

Щоб створити конкурентоспроможну веб-платформу для фріланс-послуг, необхідно зосередити увагу на ключових технічних та користувацьких аспектах. Формалізація функціональних, нефункціональних та інтерфейсних вимог є основою для формування зручного, ефективного й адаптивного продукту, що відповідатиме очікуванням користувачів.

Одним із пріоритетних факторів є інтуїтивність у використанні. Як фахівці, так і замовники повинні без зусиль взаємодіяти із системою: створювати профілі, додавати або приймати замовлення, вести комунікацію та здійснювати фінансові транзакції. Продумана структура інтерфейсу — з реалізованими механізмами фільтрації, категоризації та маркування — значно покращує навігацію та загальний досвід користування платформою.

Функціональні вимоги платформи включають реалізацію можливостей для двосторонньої взаємодії між користувачами — через чат, систему відгуків, рейтингів і сповіщень. Додатково до цього платформа повинна підтримувати модерацію контенту та механізми вирішення спорів, що сприяє формуванню довіри між учасниками. Нефункціональні вимоги охоплюють, зокрема, вимоги до дизайну інтерфейсу. Інтерфейс системи має бути логічно структурованим, візуально приємним та адаптивним до різних типів пристроїв і розширень екрана. Це забезпечує комфортну взаємодію для користувачів з різним рівнем технічної підготовки. Важливою складовою функціоналу є захист персональних даних та збереження цілісності облікових записів, що реалізується через надійні механізми

автентифікації та контролю доступу. Продуктивність інформаційної системи для надання фріланс послуг визначається швидкістю відгуку інтерфейсу користувача, оптимізованою обробкою запитів на сервері та ефективною взаємодією між клієнтською та серверною частинами, реалізованими за допомогою фреймворків React і Node.js. Важливим аспектом є також оптимізація структури та запитів до бази даних для забезпечення стабільної роботи при зростанні навантаження. Забезпечення масштабованості системи дозволяє адаптувати її до поступового зростання кількості користувачів та збільшення обсягів оброблюваного контенту без втрати продуктивності чи надійності [1].

1.3 Опис варіантів використання фріланс-платформи

Для створення діаграми варіантів використання було використано інструмент PlantUML. PlantUML — це засіб для автоматизованого створення UML-діаграм із текстових описів. Завдяки простому синтаксису, PlantUML дозволяє легко створювати діаграми без потреби у складних графічних редакторах. Інструмент підтримує різні типи UML-діаграм, зокрема діаграми варіантів використання (Use Case diagrams), діаграми класів, послідовностей та інші. Однією з ключових переваг є інтеграція PlantUML з середовищем розробки Visual Studio Code через відповідний плагін. Це дає змогу розробникам створювати, переглядати та оновлювати UML-діаграми прямо у процесі роботи над кодом, що значно підвищує ефективність документування архітектури програмного забезпечення [6].

Діаграма варіантів використання (Use Case diagram) допомагає візуалізувати взаємодію між системою фріланс-платформи та різними категоріями користувачів, а також визначити основні функціональні можливості, які мають бути реалізовані. Це ефективний інструмент для формування функціональних вимог до системи, орієнтованих на потреби цільової аудиторії.

На діаграмі варіантів використання представлені наступні основні

компоненти:

– Актори — користувачі або зовнішні системи, які взаємодіють із платформою. У даному випадку виділено чотири типи акторів: Гість, Замовник (Клієнт), Фрілансер (Виконавець), Адміністратор.

– Варіанти використання — функціональні сценарії, що демонструють можливі дії кожного з акторів у межах системи.

– Зв’язки — лінії взаємодії між акторами та варіантами використання.

У межах фріланс-платформи передбачено такі основні ролі та функціональність:

– Гість — неавторизований користувач, який має доступ до перегляду доступних послуг та пошуку по платформі, а також може здійснити реєстрацію для отримання розширених можливостей.

– Замовник (Клієнт) — зареєстрований користувач, що має змогу переглядати доступні послуги, шукати виконавців, створювати замовлення, здійснювати оплату за виконані роботи, обмінюватися повідомленнями з фрілансерами, а також залишати оцінки та відгуки щодо виконаних замовлень.

– Фрілансер (Виконавець) — зареєстрований користувач, який може додавати власні послуги, переглядати доступні замовлення, брати участь у їх виконанні, обмінюватися повідомленнями з замовниками, а також оцінювати та залишати відгуки.

– Адміністратор — користувач із розширеними правами, що здійснює модерацію контенту, вирішення спорів між користувачами та контроль за загальним функціонуванням системи.

До основних дій, які реалізуються у системі, належать:

– Для Гостя: перегляд послуг, пошук послуг, реєстрація.

– Для Замовника: реєстрація, вхід у систему, перегляд послуг, пошук послуг, створення замовлення, оплата замовлення, обмін повідомленнями, залишення оцінок та відгуків.

– Для Фрілансера: реєстрація, вхід у систему, перегляд замовлень, виконання замовлень, додавання власних послуг, обмін повідомленнями, залишення оцінок та

відгуків.

– Для Адміністратора: модерація контенту, вирішення спорів, перегляд замовлень і послуг.

Діаграма варіантів використання наведена на рисунку (див. рис. 1.1), де наочно показано ключові сценарії взаємодії користувачів із платформою.

Рисунок 1.1 – Діаграма варіантів використання

Опис типових сценаріїв використання фріланс-платформи демонструє взаємодію користувачів із системою. Один із базових сценаріїв — створення нового замовлення. Замовник, увійшовши до системи, створює нове замовлення, заповнюючи необхідну інформацію щодо завдання: назву, опис, бажані терміни виконання та бюджет. Після підтвердження, замовлення публікується на платформі та стає доступним для перегляду фрілансерами.

Ще одним важливим сценарієм є взаємодія фрілансера із замовленнями.

Фрілансер має змогу переглядати доступні завдання, відправляти пропозиції на участь у виконанні обраних замовлень, а також вести комунікацію із замовниками через вбудовану систему повідомлень.

Після досягнення домовленостей фрілансер виконує роботу та надсилає результат через платформу. Замовник перевіряє виконане завдання, у разі задоволення приймає його та проводить оплату через безпечну платіжну систему.

Окремим важливим елементом взаємодії є система зворотного зв’язку. Після завершення співпраці як замовник, так і фрілансер можуть залишити відгук і оцінку, що впливає на рейтинг і репутацію користувачів у системі.

Користувачі платформи також мають змогу керувати своїм профілем. Вони можуть редагувати особисту інформацію, змінювати налаштування безпеки (наприклад, оновлення пароля чи email-адреси), переглядати історію своїх замовлень або наданих послуг.

Для забезпечення безпеки та порядку на платформі передбачено роль адміністратора. Адміністратор здійснює модерацію контенту, слідкує за дотриманням правил користування системою, вирішує спірні ситуації між замовниками та фрілансерами, а також забезпечує стабільну роботу самої платформи.

Таким чином, система підтримує повний цикл взаємодії між учасниками процесу надання фріланс-послуг — від створення завдання до оплати та зворотного зв’язку, забезпечуючи прозорість, зручність і безпеку для всіх сторін.

Для забезпечення наочності процесів взаємодії між компонентами інформаційної системи фріланс послуг були побудовані діаграми послідовності. Діаграма послідовності (Sequence Diagram) демонструє динаміку виконання конкретного функціоналу системи, тобто які об'єкти (компоненти) взаємодіють між собою у часі та які запити надсилаються у процесі виконання тієї чи іншої дії.

Функціонал створення нового замовлення

У межах функціоналу платформи підтримується можливість створення нового замовлення, яке ініціюється користувачем (фрілансером) через клієнтський

інтерфейс. Процес створення замовлення реалізовано за допомогою взаємодії фронтенд-компонентів із серверною частиною платформи та базою даних.

Учасники процесу:

- Користувач (Фрілансер)
- Компонент Add.jsx
- React Query (useMutation)
- Сервер (Backend API)
- База даних (MongoDB)

Основні кроки взаємодії:

1. Фрілансер переходить до сторінки створення нового замовлення (компонент Add.jsx).

2. Заповнює форму з даними про замовлення (назва, категорія, опис, зображення, ціна, строки виконання тощо).

3. Після заповнення форми користувач натискає кнопку «Create», що викликає метод handleSubmit().

4. У методі handleSubmit ініціюється мутація React Query (useMutation), яка формує POST-запит до серверного API за маршрутом /gigs.

5. Серверна частина приймає запит, проводить валідацію даних і зберігає інформацію у колекцію Gig бази даних MongoDB.

6. Після успішного збереження сервер повертає відповідь про створене замовлення.

7. Клієнтський інтерфейс оновлює стан та, за необхідності, перенаправляє користувача до розділу з його замовленнями.

Таким чином, забезпечується зручний механізм створення нового замовлення з мінімальними діями зі сторони користувача та автоматизованою обробкою даних на сервері.

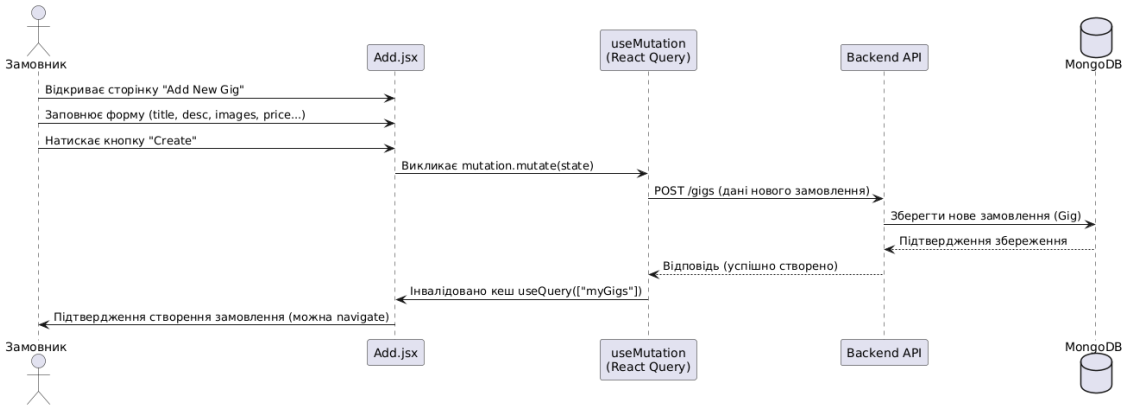

Рисунок 1.2 – Діаграма послідовності

Функціонал виконання замовлення

У системі надання фріланс-послуг важливу роль відіграє функціонал взаємодії між замовником та виконавцем (фрілансером) після створення замовлення. Користувач (фрілансер) може переглядати список отриманих замовлень, а також здійснювати взаємодію з замовником для уточнення деталей та підтвердження виконання завдання.

Учасники процесу:

- Користувач (Фрілансер)
- Компонент Orders.jsx
- React Query (useQuery)
- Сервер (Backend API)
- База даних (MongoDB)

Основні кроки взаємодії:

1. Фрілансер переходить на сторінку «Orders», яка реалізована у компоненті Orders.jsx.

2. При завантаженні сторінки автоматично викликається useQuery, що надсилає GET-запит до серверного API /orders для отримання актуального списку замовлень.

3. Серверна частина обробляє запит та отримує дані з колекції Order бази даних MongoDB.

4. Список замовлень передається у відповідь клієнту.

5. Компонент Orders.jsx відображає отриманий список замовлень у табличному вигляді.

6. Для кожного замовлення передбачено можливість ініціювати контакт з замовником (кнопка «Contact» → перехід у чат).

7. При натисканні кнопки «Contact» відбувається navigate до сторінки чату (message.jsx), де користувач може спілкуватися щодо виконання замовлення.

Таким чином, функціонал «Виконання замовлення» забезпечує фрілансеру зручний доступ до актуального списку замовлень та інтегровані інструменти комунікації із замовниками.

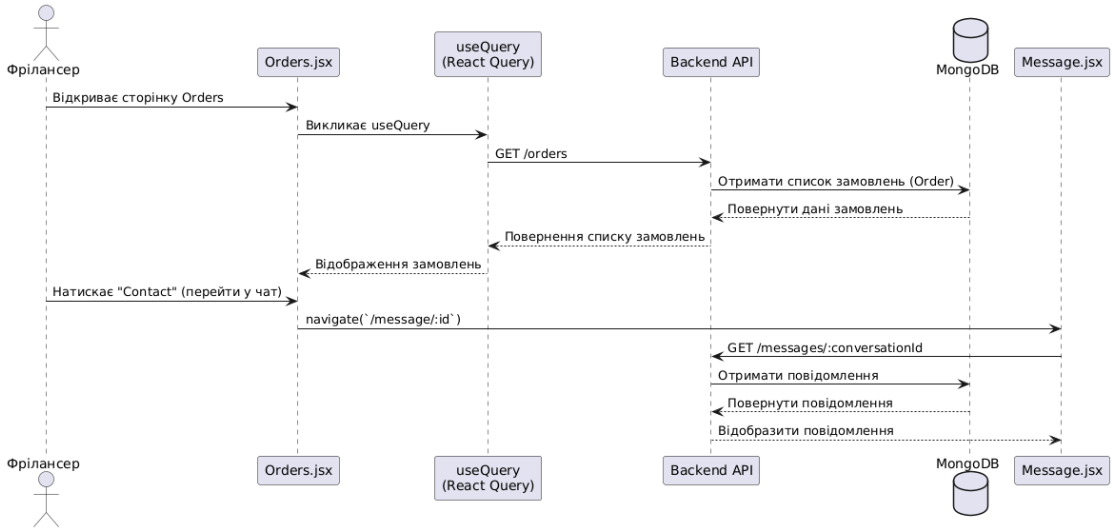

Рисунок 1.3 – Діаграма послідовності

Функціонал проведення оплати за замовлення

Учасники:

- Користувач (Замовник)
- Компонент: Pay.jsx
- React Query / useEffect (ініціалізація платежу)
- Платіжна система: Stripe (через create-payment-intent API)
- Сервер (Backend API)
- База даних (MongoDB → збереження платежу у колекції Payment)

Опис процесу:

1. Користувач переходить на сторінку оплати після створення нового

замовлення → шлях /pay/:id.

2. При завантаженні компонента Pay.jsx викликається функція makeRequest(), яка надсилає POST-запит на бекенд за адресою /orders/create-payment-intent/:id.

3. Backend створює Payment Intent через Stripe API та повертає clientSecret → у відповідь зберігається у стані clientSecret.

4. Після отримання clientSecret, на сторінці ініціалізується компонент Elements від Stripe з використанням clientSecret та темою Stripe.

5. Користувач вводить дані платіжної картки у компоненті CheckoutForm.jsx та підтверджує оплату.

6. Після успішної транзакції, Stripe автоматично викликає webhook / callback (опціонально) → сервер оновлює статус платежу у базі → статус "сплачено".

7. Користувач отримує повідомлення про успішну оплату (перенаправлення на сторінку Success).

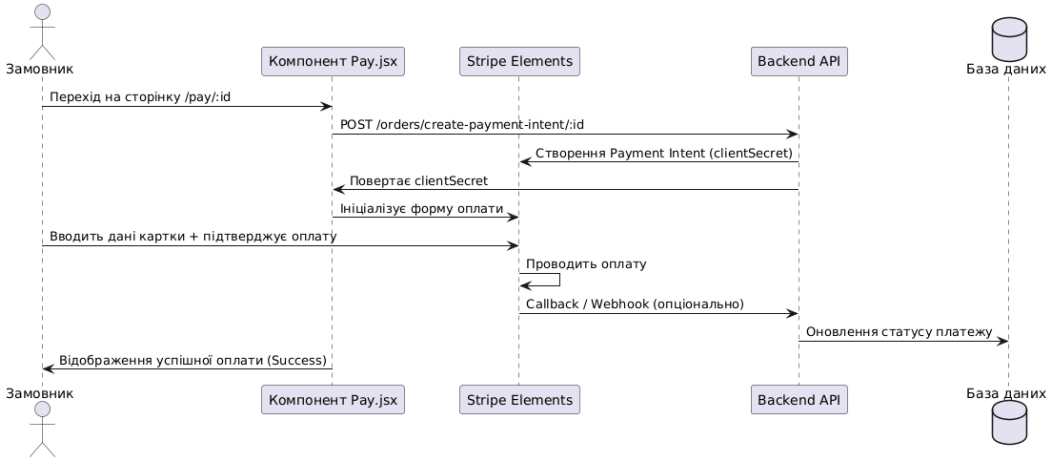

Рисунок 1.4 – Діаграма послідовності

Діаграми діяльності для ключових функціоналів системи дозволяють наочно відобразити логіку виконання дій користувачами та системою, відображаючи послідовність етапів та можливі варіанти переходів між ними. Ці діаграми відображають взаємодію користувача з системою в процесі виконання певної бізнес-функції.

На рисунку 1.4 представлено діаграму діяльності процесу створення нового

замовлення на платформі фріланс послуг.

Діаграма відображає етапи взаємодії користувача (Замовник) з клієнтським інтерфейсом (форма додавання замовлення), виклик API для створення нового запису у базі даних, а також обробку можливих результатів (успішне створення або помилку).

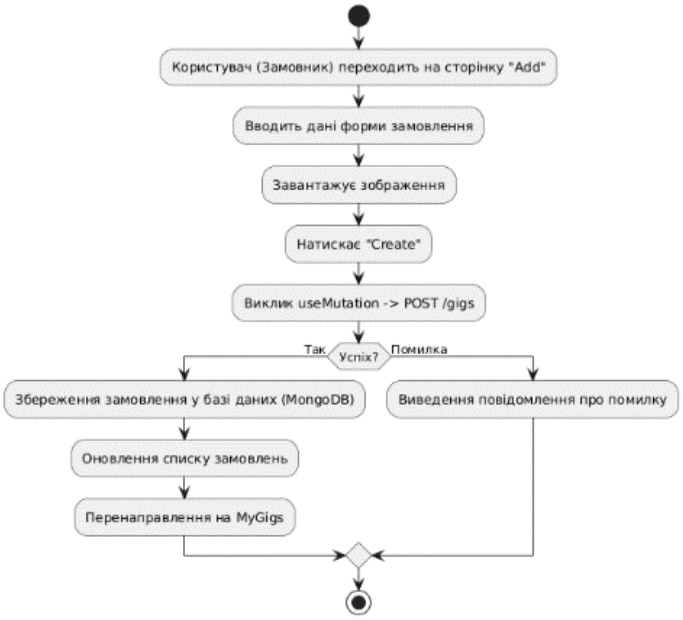

Рисунок 1.5 – Діаграма діяльності

На рисунку 1.5 зображено діаграму діяльності для функціоналу виконання замовлення. Процес включає перегляд фрілансером доступних замовлень, ознайомлення з деталями замовлення, виконання роботи, а також оновлення статусу замовлення у системі та сповіщення замовника про завершення.

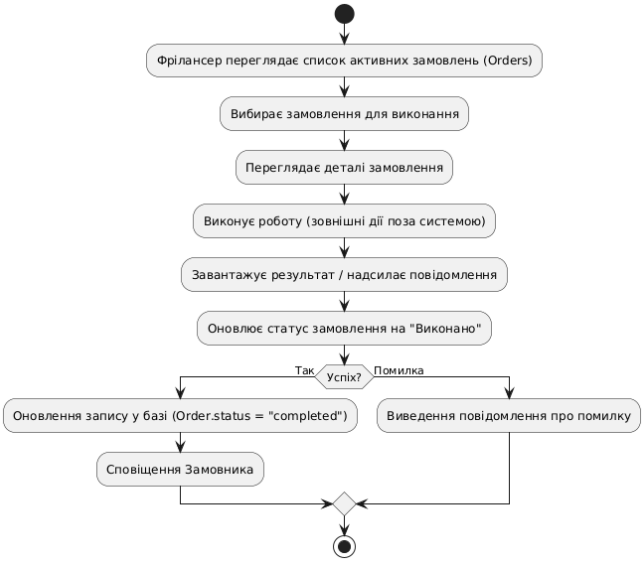

Рисунок 1.6 – Діаграма діяльності

На рисунку 1.6 наведено діаграму діяльності процесу оплати за замовлення. Цей процес охоплює ініціалізацію платіжного процесу через інтеграцію з платіжною системою (Stripe), введення користувачем платіжних даних, підтвердження транзакції, обробку відповіді від платіжного шлюзу та оновлення статусу замовлення у базі даних.

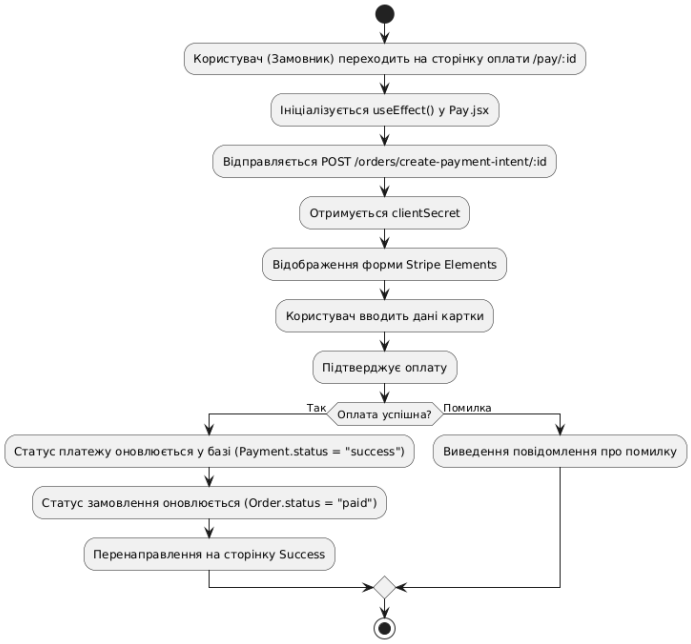

Рисунок 1.7 – Діаграма діяльності

Правильний вибір середовища розробки має визначальне значення при створенні інформаційної системи для надання фріланс послуг. Це рішення безпосередньо впливає на ключові показники платформи, зокрема її продуктивність, масштабованість, рівень безпеки та подальшу зручність у супроводі й модернізації.

В процесі розробки було обрано сучасні інструменти та технології, які відповідають вимогам проєкту. Для реалізації клієнтської частини застосовано фреймворк React, що базується на мові програмування JavaScript. Основною перевагою React є використання концепції віртуального DOM, що дозволяє ефективно керувати оновленнями інтерфейсу. Завдяки цьому значно знижується навантаження на браузер, що особливо важливо для динамічних веб-додатків з великою кількістю інтерактивних елементів.

Крім того, активна спільнота розробників та багатий вибір готових бібліотек і рішень для React значно прискорюють процес створення користувацького інтерфейсу. Це дозволяє зосередитися на розробці унікальної бізнес-логіки платформи, мінімізуючи витрати часу на типові елементи. Окремої уваги заслуговує можливість використання серверного рендерингу (Server-Side Rendering), що покращує оптимізацію для пошукових систем (SEO). Для фріланс-платформи це є важливим аспектом, оскільки висока видимість профілів виконавців і доступних послуг у пошукових системах сприяє залученню нових користувачів.

Для реалізації серверної частини платформи було обрано Node.js — сучасне середовище виконання JavaScript на сервері. Основною перевагою такого підходу є можливість використовувати одну мову програмування — JavaScript — на всіх рівнях розробки: як у клієнтській, так і у серверній частинах. Це дозволяє забезпечити високу уніфікацію коду, спрощує спільну роботу над проєктом для розробників фронтенду та бекенду, а також полегшує повторне використання окремих модулів між різними частинами системи.

Окрім цього, розвинена екосистема пакунків Node.js (npm) надає доступ до широкого спектра готових рішень та модулів, що дозволяє суттєво скоротити час

розробки функціональних можливостей платформи для надання фріланс послуг. Завдяки цьому розробники мають змогу оперативно інтегрувати такі важливі сервіси, як системи миттєвого обміну повідомленнями, сповіщення про зміни у замовленнях, інструменти редагування замовленнями та інші необхідні для платформи функціональні компоненти.

Однією з ключових переваг Node.js є також його здатність до горизонтального масштабування, що дозволяє ефективно розподіляти обчислювальні ресурси між кількома процесами або навіть окремими серверами. Це дає змогу забезпечити високу стабільність та продуктивність системи у випадках інтенсивного зростання кількості користувачів та збільшення обсягу транзакцій на платформі. Такий підхід є критично важливим для фріланс-платформи, де передбачається активна взаємодія між великою кількістю замовників та виконавців, постійний потік повідомлень та операцій з оплатою.

Для зберігання даних у системі було обрано базу даних MongoDB. Ця документно-орієнтована СУБД забезпечує зберігання інформації у гнучкому форматі BSON (розширений формат JSON), що дозволяє динамічно змінювати структуру даних у процесі еволюції платформи. Такий підхід є особливо доцільним для систем з багатим і динамічним функціоналом, як-от платформи з надання фріланс послуг, де у процесі розвитку часто виникає необхідність додавання нових атрибутів та зв’язків між сутностями [2].

Крім того, MongoDB підтримує широкі можливості індексування, що забезпечує високу продуктивність при виконанні пошукових операцій і складних вибірок. Завдяки підтримці вкладених документів та складних ієрархічних структур, база даних дозволяє ефективно моделювати дані для таких ключових компонентів системи, як профілі користувачів, історія замовлень, повідомлення, рейтинги, а також взаємодії між замовниками та виконавцями. Така структура даних ідеально відповідає потребам веб-платформи, яка оперує великою кількістю взаємопов’язаних об’єктів та має високі вимоги до гнучкості та масштабованості.

2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИТЕМИ

2.1 Огляд підходу до тестування та розробки інформаційної системи

У цьому розділі детально розглядається підхід до проєктування та розробки веб-платформи для надання фріланс послуг. Основна мета — створення зручного, функціонального та ефективного онлайн-інструменту для взаємодії між замовниками та виконавцями послуг. Платформа має забезпечувати можливість реєстрації користувачів, розміщення пропозицій та замовлень, ведення переговорів, здійснення безпечних транзакцій і управління замовленнями. Цей підпункт має на меті представити загальний огляд процесу розробки системи, підходів і методологій, що використовувалися для досягнення поставлених цілей. Зокрема, буде висвітлено вибір технологій, організацію архітектури платформи, процес тестування та підходи до забезпечення якості програмного продукту.

Процес створення інформаційної системи для надання фріланс послуг розпочався з детального планування та аналізу вимог. Було здійснено збір ключових вимог шляхом вивчення ринку аналогічних платформ, а також за результатами аналізу очікувань потенційних користувачів. Визначено як функціональні, так і нефункціональні параметри системи. Цей етап мав критичне значення для подальшого проєктування, оскільки дозволив сформувати чітке бачення структури платформи та її можливостей, що безпосередньо впливають на зручність використання і відповідність очікуванням різних груп користувачів.

Для розробки інформаційної системи з надання фріланс послуг було обрано гнучку методологію Agile, яка забезпечує можливість оперативного реагування на зміну вимог і дозволяє поступово вдосконалювати продукт. Використання Agile стало доцільним вибором для проєкту, оскільки платформа орієнтована на різноманітні категорії користувачів і потребує постійної адаптації до ринкових тенденцій та зворотного зв’язку з боку аудиторії. Гнучкість і адаптивність даного підходу дає змогу своєчасно впроваджувати необхідні зміни й удосконалення. Завдяки циклічній структурі розробки, регулярному тестуванню та активному

залученню зворотного зв’язку команда мала змогу виявляти й усувати недоліки на ранніх етапах, що сприяло підвищенню стабільності й якості платформи. Таким чином, застосування Agile-методологій дозволило забезпечити динамічний процес розробки, що відповідає високим вимогам сучасних веб-сервісів у сфері фріланс-послуг [7].

Архітектурне проєктування є одним із найважливіших етапів під час створення інформаційної системи для надання фріланс послуг, оскільки саме на цьому етапі формується загальна структура платформи та визначається взаємодія її компонентів. Проєктування архітектури розпочалося з аналізу функціональних та нефункціональних вимог, серед яких особливу увагу було приділено продуктивності, можливості масштабування системи у разі зростання кількості користувачів, а також забезпеченню високого рівня безпеки для захисту персональних даних і фінансових транзакцій. Грамотно спроєктована архітектура платформи забезпечує її стабільність, розширюваність та надійність у роботі.

Вибір архітектурного стилю є важливим етапом створення інформаційної системи для надання фріланс послуг, оскільки він визначає спосіб взаємодії між різними компонентами платформи. У даному проєкті була обрана архітектура типу "клієнт-сервер", яка забезпечує чіткий розподіл відповідальності між клієнтською та серверною частинами. Для реалізації інтерфейсу користувача застосовується React — завдяки його гнучкій компонентній моделі та високій продуктивності. Серверна частина розробляється з використанням Node.js, що дозволяє ефективно обробляти великий обсяг одночасних запитів. Як система зберігання даних використовується MongoDB, яка забезпечує необхідну гнучкість структурування даних, що є важливим для платформи з різними типами послуг і контенту [3].

Процес проєктування функціональних модулів веб-платформи для фріланс-сервісів охоплює побудову логіки користувацького інтерфейсу, розробку API-інтерфейсів, структуру бази даних та засобів автентифікації. Особливе значення має забезпечення узгодженого обміну між цими складовими через стандартизовані протоколи (зокрема HTTP/HTTPS) та формат JSON для передачі даних. Питання

безпеки розв’язуються шляхом інтеграції шифрування, реалізації двофакторної автентифікації, а також впровадження заходів захисту від типових кіберзагроз, що гарантує надійну взаємодію всіх користувачів платформи [5].

Масштабованість та продуктивність інформаційної системи для надання фріланс послуг забезпечуються за рахунок реалізації плаваючого навантаження, оптимізації серверного та клієнтського коду, а також ефективної роботи з базою даних. Грамотною спроєктована архітектура платформи дозволяє їй безперешкодно масштабуватися відповідно до зростання кількості користувачів та обсягу даних, а також забезпечує варіативність у впровадженні нових функціональних можливостей, що сприяє подальшому розвитку системи та її стабільній підтримці.

Створення бази даних є одним із ключових етапів у процесі розробки веб-платформи, особливо якщо йдеться про ресурс для управління блог-контентом. Для реалізації проєкту було обрано MongoDB — документоорієнтовану NoSQL систему, яка забезпечує високу гнучкість і масштабованість. Однією з переваг MongoDB є відсутність жорстко заданої схеми, що дозволяє легко адаптувати структуру збережених даних у процесі розвитку застосунку без потреби в складних міграціях. База даних підтримує зберігання документів у форматі, подібному до JSON, що ідеально узгоджується з JavaScript-орієнтованими технологіями. Крім того, MongoDB здатна до ефективного горизонтального масштабування — при зростанні обсягів даних можливо безболісно додавати нові вузли, не змінюючи логіку роботи всієї системи. Такий підхід робить її оптимальним вибором для високонавантажених або динамічно зростаючих проєктів.

MongoDB зберігає інформацію у вигляді документів, подібних до формату JSON, що робить її зручною для інтеграції з веб-застосунками. Такий підхід дозволяє напряму працювати з даними у вигляді об’єктів JavaScript без потреби у складній трансформації. Завдяки своїй гнучкій структурі та простоті у використанні MongoDB значно полегшує процес розробки.

Клієнтська частина платформи була реалізована за допомогою бібліотеки React, що дало змогу створити гнучкий, інтерактивний інтерфейс користувача.

Завдяки компонентній структурі, React забезпечує можливість розробки окремих модулів, які можна повторно використовувати в різних частинах застосунку, що значно полегшує масштабування й супровід системи.

Серверна логіка була реалізована на основі середовища Node.js, що забезпечило ефективну обробку клієнтських запитів у реальному часі. Бекенд виконує ключові завдання, зокрема автентифікацію користувачів, опрацювання переданих даних та взаємодію з MongoDB. Завдяки використанню JavaScript як на клієнтській, так і на серверній стороні, вдалося досягти єдності технологій, що спростило процес розробки та супроводу проєкту.

Інтеграція модулів системи здійснювалася поетапно, з регулярною перевіркою стабільності та функціональності після кожного доданого компонента. Для контролю якості використовувалися як автоматизовані інструменти тестування, так і ручна перевірка. Основна увага приділялася перевірці працездатності ключових функцій, відповідності системи вимогам безпеки, а також оцінці її продуктивності при різному навантаженні.

2.2 Проектування бази даних

У межах розробки веб-платформи для надання фріланс послуг базу даних створено за допомогою MongoDB. Це документно-орієнтована NoSQL база даних, що дозволяє ефективно працювати з динамічними структурами даних та масштабувати систему горизонтально. У якості засобу моделювання даних було використано бібліотеку Mongoose для Node.js, що забезпечує зручне визначення схем і валідацію даних.

Users (користувачі)

Колекція users містить інформацію про всіх зареєстрованих користувачів платформи:

– personal_info: ім’я, email, пароль, ім’я користувача, біо, аватар.

– social_links: посилання на соцмережі.

– account_info: кількість постів, кількість переглядів.

– google_auth: чи була авторизація через Google.

– blogs: масив ID створених блогів.

– timestamps: дата реєстрації.

Колекція users є центральною складовою моделі даних платформи. Вона забезпечує зберігання всієї основної інформації про зареєстрованих користувачів, що взаємодіють із системою як замовники, фрілансери або адміністратори. Ця колекція виступає як головне джерело ідентифікації та аутентифікації користувачів під час авторизації на платформі. Саме через неї система визначає, хто є автором створених публікацій, хто залишає коментарі, а також кому адресовано сповіщення про взаємодії в системі.

Коментарі зберігаються у вигляді масиву і реалізовані через зовнішні посилання в полі orders, яке містить відповідні ідентифікатори записів з колекції коментарів у базі даних.

Схема orders:

– order_id: Посилання на замовлення,

– order_client: Посилання на замовника (клієнта), якому належить замовлення.

– comment: Сам коментар (уточнення деталей замовлення або коментар до етапу виконання).

– commented_by: Посилання на фрілансера або замовника, який залишив коментар.

– timestamps: Дата створення коментаря.

Колекція orders зберігає опис завдання, яке залишають замовлення в межах замовлення. Вона забезпечує ефективну комунікацію між замовниками та фрілансерами під час процесу виконання замовлень.

Це дозволяє фіксувати важливі домовленості, уточнення до завдань, хід виконання роботи та інші аспекти співпраці.

Крім того, підтримується структура відповідей, що дає змогу вести

багаторівневі дискусії у вигляді вкладених коментарів.

2.3 Моделювання архітектури інформаційної системи фріланс послуг

Фронтенд додатку реалізовано відповідно до концепції компонентно-орієнтованої архітектури, що є основоположною в React. Уся структура інтерфейсу розподілена на окремі функціональні модулі — компоненти та сторінки, кожен з яких виконує конкретну роль у забезпеченні взаємодії з користувачем.

Одним із ключових елементів є компонент Navbar (розміщений у файлі components/navbar.component.js). Він відповідає за формування верхньої навігаційної панелі. У цій панелі розміщено логотип платформи, поле для пошуку, а також інтерактивні посилання на основні розділи системи — профіль користувача, перелік доступних замовлень, модуль повідомлень, центр сповіщень та інші секції, які забезпечують навігацію по функціоналу платформи.

– UserAuthForm (pages/login.jsx)**:** Компонент для реєстрації та авторизації користувача (User).

– HomePage (pages/home.js)**:** Головна сторінка додатку, де відображаються актуальні послуги (Gig), популярні фрілансери (User) та рекомендовані категорії.

– ProfilePage (pages/profile.js)**:** Сторінка профілю користувача (User), де можна переглядати і редагувати персональну інформацію, список власних послуг (Gig), історію замовлень (Order), отримані відгуки (Review).

– GigPage (аналог BlogPage) *(pages/gig.page.js)*: Сторінка окремої послуги (Gig) з можливістю перегляду її опису, ціни, портфоліо та відгуків (Review), а також створення замовлення (Order).

– Notifications (pages/notifications.page.js)**:** Сторінка зі списком сповіщень (Notification), які інформують користувача про нові повідомлення (Message), нові замовлення (Order), оплати (Payment), нові відгуки (Review).

– Messages (pages/messages.js)**:** Сторінка списку діалогів (Conversation), у межах яких користувач веде обговорення з іншими учасниками (замовником або фрілансером).

– Message (pages/message..js)**:** Сторінка конкретної розмови (Conversation), що відображає список повідомлень (Message) у цій розмові та дозволяє користувачеві надсилати нові повідомлення.

– Orders (pages/orders..js)**:** Сторінка перегляду замовлень (Order), де користувач може бачити статус своїх замовлень або замовлень, які він виконує як фрілансер.

– Payments (неявно реалізовано): Модуль для роботи з оплатами (Payment), інтегрований у процес оформлення замовлення (Order).

– Reviews (неявно реалізовано)**:** Інтерфейс для залишення та перегляду відгуків (Review) про послуги (Gig) після виконання замовлення.

У реалізації маршрутизації застосовується бібліотека react-router-dom, яка надає гнучкий механізм для налаштування переходів між сторінками на основі URL-адрес. Визначені маршрути інтегруються безпосередньо в структуру інтерфейсу, що забезпечує динамічне відображення відповідних компонентів залежно від шляху.

Для керування глобальним станом у додатку використовуються контекстні механізми, зокрема UserContext, що зберігає інформацію про поточного користувача. Також можуть бути реалізовані додаткові контексти, як-от ThemeContext для зберігання параметрів теми чи NotificationContext для обробки системних повідомлень. Застосування контекстів забезпечує доступ до цих даних на будь-якому рівні компонування без необхідності передавання їх через послідовні пропси.

Для наочного представлення архітектури системи доцільно створити UML-діаграму класів. Даний тип діаграм є ключовим елементом мов моделювання UML та слугує ефективним засобом для візуального зображення структури об’єктно-орієнтованих програм. Діаграма класів дозволяє описати основні сутності системи, їх атрибути, методи та взаємозв’язки між класами.

Використання таких діаграм забезпечує кілька важливих переваг. Зокрема, це сприяє глибшому розумінню побудови програмного продукту, дозволяє визначити залежності між складовими частинами системи та спрощує обговорення рішень у командній розробці. Крім того, під час етапу проєктування UML-діаграми класів дають змогу формалізувати архітектуру майбутнього застосунку, визначити ключові класи та реалізувати логіку взаємодії між ними у відповідності до принципів об'єктно-орієнтованого підходу.

Діаграма класів є важливою складовою технічної документації інформаційної системи фріланс-послуг, оскільки вона забезпечує структуровану та зрозумілу форму представлення взаємозв’язків між основними сутностями системи — користувачами, послугами, замовленнями, повідомленнями, сповіщеннями, оплатами тощо.

Застосування діаграми класів дозволяє розробникам ефективно орієнтуватися у структурі системи, зокрема зрозуміти, як окремі її модулі пов’язані між собою, які атрибути притаманні кожному класу та які логічні зв’язки між ними встановлено. Такий підхід є надзвичайно корисним як на етапі початкового проєктування, так і під час подальшого супроводу та розширення програмного забезпечення. Крім того, діаграма класів сприяє глибшому аналізу структури коду, дозволяє виявити надмірну складність, дублювання функцій або потенційні архітектурні недоліки, що, в свою чергу, підвищує підтримуваність і масштабованість застосунку.

У межах створення системи для організації фріланс-послуг UML-діаграма класів виконує функцію інструмента моделювання ключових процесів і логіки взаємодії між об’єктами. Вона охоплює модулі, відповідальні за керування обліковими записами користувачів (User), управління послугами та пропозиціями (Gig), реалізацію системи замовлень (Order), обмін повідомленнями (Message), платіжні механізми (Payment) та систему зворотного зв’язку (Review).

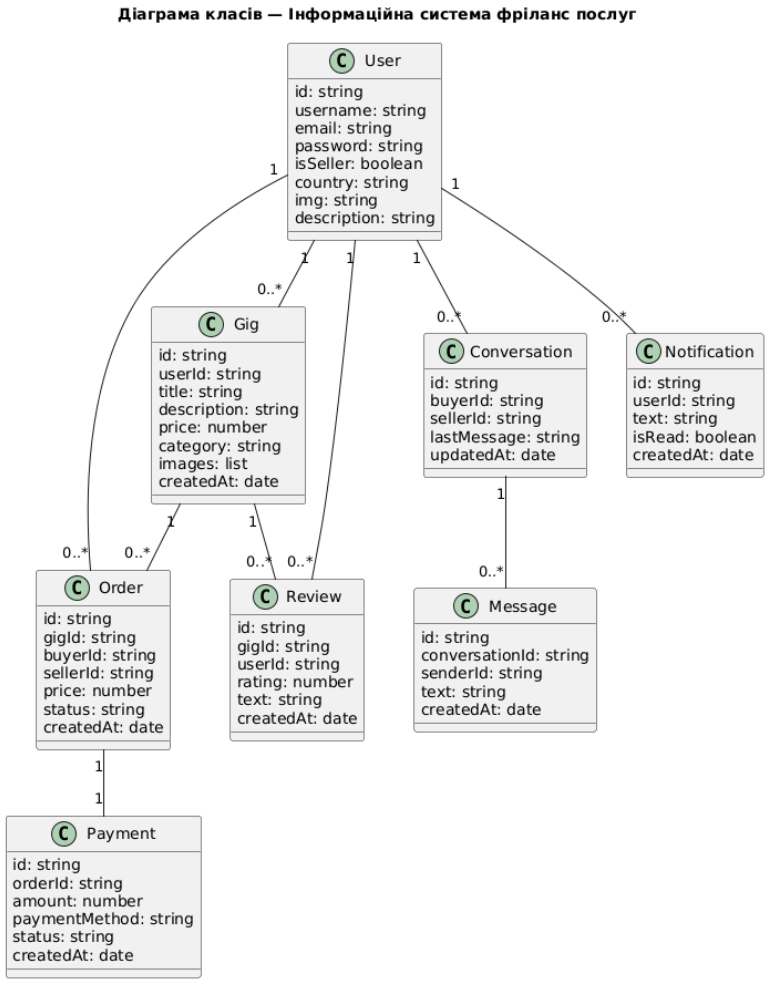

Рисунок 2.1 – Діаграма класів основної програми App

Архітектура фронтенд-частини інформаційної системи для фріланс-послуг спроєктована таким чином, щоб забезпечити зручність розширення та подальшого супроводу. Завдяки логічному поділу застосунку на автономні компоненти й сторінки з чітко окресленими функціональними обов’язками, підтримка коду та реалізація нових можливостей відбуваються швидко й ефективно. Наприклад, модулі керування профілем користувача, створення та перегляду послуг (Gig), опрацювання замовлень (Order), взаємодії через повідомлення (Message) або сповіщення (Notification) мають ізольовану реалізацію у вигляді відповідних сторінок та компонентів.

Такий підхід значно полегшує як впровадження нових функцій, так і технічне обслуговування вже реалізованого функціоналу. Основу структури фронтенд-частини фріланс-платформи становить модульність, яка забезпечує поділ логіки на

окремі, самостійні елементи. Кожен компонент реалізує окремий елемент користувацького інтерфейсу. або функціональності, що сприяє ізольованій розробці, підвищенню повторного використання коду та зменшенню залежностей. Компонент Navbar реалізує тільки навігаційну панель, тоді як AddGig відповідає за додавання нових фріланс-послуг, а Orders — за управління замовленнями. Це забезпечує високу читабельність, підтримуваність та гнучкість коду. Архітектура застосунку побудована таким чином, що забезпечує зручне масштабування функціональності. Завдяки логічному поділу на окремі компоненти та сторінки, платформа легко адаптується до нових вимог. Це дозволяє впроваджувати додаткові модулі чи розділи без необхідності змінювати вже реалізовані частини системи, зберігаючи стабільність і чистоту коду. Наприклад, при потребі у впровадженні нового модуля — як-от Підтримка клієнтів

Інтерфейс платформи побудований відповідно до принципів зручності та інтуїтивної взаємодії користувача. Основні елементи навігації дозволяють оперативно перемикатися між ключовими функціональними блоками системи, зокрема такими як «Послуги» (Gigs), «Замовлення» (Orders), «Повідомлення» (Messages) та «Сповіщення» (Notifications). Процес авторизації та реєстрації реалізований у зрозумілому форматі, що забезпечує легкий вхід у систему для нових користувачів — як фрілансерів, так і замовників.

2.4 Розробка серверної частини фріланс-платформи

Для розробки серверної частини інформаційної системи з надання фріланс послуг було використано платформу Node.js. Проєкт було ініціалізовано за допомогою команди npm , після чого були встановлені всі необхідні залежності за допомогою менеджера пакунків npm.

Основні пакети, що були використані:

- Express — для створення REST API та обробки HTTP-запитів.

- Mongoose — для підключення до бази даних MongoDB, створення схем колекцій, таких як *users*, *gigs*, *orders*, *messages*, *conversations* та *notifications*.
- bcrypt — для хешування паролів користувачів перед збереженням у БД.
- jsonwebtoken (JWT) — для реалізації механізму автентифікації та авторизації користувачів.

Для централізованого зберігання конфіденційних налаштувань було створено конфігураційний файл .env, у якому розміщено змінні середовища — зокрема, рядок підключення до бази даних (MONGO_URI) та секретний ключ для генерації JWT (JWT_SECRET). Для завантаження цих змінних у середовище виконання застосовано пакет dotenv, що імпортується у головному серверному файлі.

Підключення до бази даних MongoDB здійснюється за допомогою mongoose.connect(), де використовується рядок підключення з .env. Це дозволяє ізолювати конфіденційні дані від основного коду програми та полегшує налаштування середовища.

У БД були налаштовані та створені такі основні колекції:

- Users — зберігання інформації про користувачів (замовників та фрілансерів).
- Gigs — послуги, які пропонують фрілансери.
- Orders — замовлення, які створюють замовники.
- Messages — повідомлення між користувачами у межах платформи.
- Conversations — діалоги між користувачами.
- Notifications — система сповіщень щодо активності користувачів.

З метою оптимізації продуктивності запитів до бази даних було активовано автоматичне створення індексів у MongoDB (рис. 2.2). Це дало змогу суттєво пришвидшити виконання операцій фільтрації, пошуку та сортування даних у відповідних колекціях.

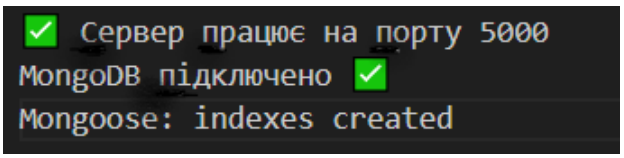

Рисунок 2.2 – Підключення до бази даних MongoDB

Домашня сторінка веб-платформи для надання фріланс послуг відіграє ключову роль у створенні першого враження у користувачів. Саме з головної сторінки починається знайомство потенційного клієнта або фрілансера з функціональними можливостями платформи.

У структурі фронтенду компонент Home представлений як окрема сторінка (pages/Home.jsx), яка об'єднує кілька важливих складових інтерфейсу.

– Featured блок

– Банер з підсвічуванням

Розміщено додатковий інформаційний банер з коротким мотиваційним текстом (див. рис. 1).

```
<div className="highlight-banner">
  <div className="container">
    <h2>🚀 Почни свою подорож із найкращими фахівцями!</h2>
    <p>Знаходь професіоналів, готових підняти твій бізнес на новий рівень.</p>
  </div>
</div>
```

Рисунок 2.3 – Банер

– Слайдер з категоріями послуг

Компонент <Slide> використовується для зручного відображення списку категорій у вигляді горизонтальної каруселі. Всередині каруселі рендеряться елементи CatCard, які містять інформацію про категорії послуг.

```
<Slide slidesToShow={4} arrowsScroll={4}>
  {cards.map((card) => (
    <CatCard key={card.id} card={card} />
  ))}
</Slide>
```

Рисунок 2.3 – Лістинг компонента

Домашня сторінка платформи спроєктована з урахуванням принципів адаптивного дизайну, що забезпечує коректне відображення контенту як на десктопних, так і на мобільних пристроях. Використання інтерактивних слайдерів та сіткової структури дозволяє створити сучасний та інтуїтивно зрозумілий вигляд (див. рис.21).

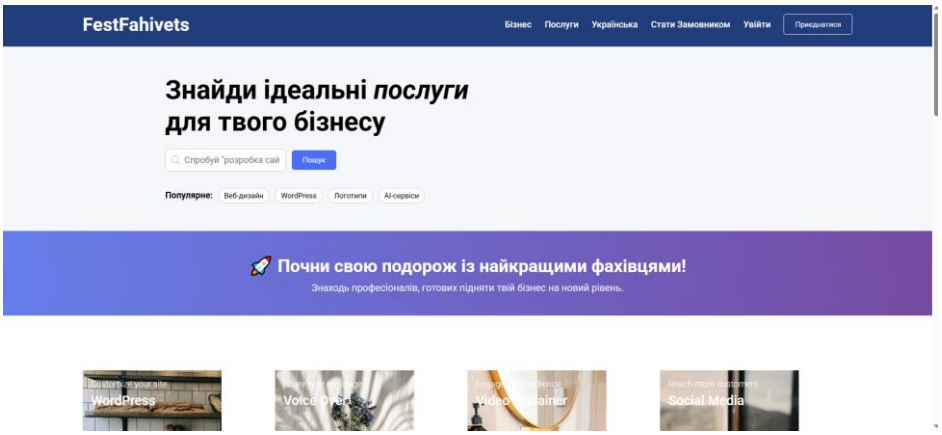

Рисунок 2.4 – Вигляд сторінки

Одним із основних функціональних можливостей платформи фріланс-послуг є створення нового замовлення (пропозиції послуги) з боку виконавця (фрілансера). Цей процес реалізовано на основі React на фронтенді та Express + MongoDB (Mongoose) на бекенді. Для забезпечення ефективної взаємодії клієнтської частини з сервером у проєкті застосовано бібліотеку React Query, яка оптимізує процес отримання, кешування та оновлення даних, спрощуючи управління асинхронними запитами у React-додатку.

У компоненті використовується хук useReducer з ред'юсером gigReducer, що дозволяє ефективно керувати станом форми:

```
const [state, dispatch] = useReducer(gigReducer, INITIAL_STATE);
```

Рисунок 2.5 – Лістинг

При введенні даних у поля форми використовується функція обробки подій

handleChange, яка змінює стан відповідно до введеного значення:

```
const handleChange = (e) => {
  dispatch({
    type: "CHANGE_INPUT",
    payload: { name: e.target.name, value: e.target.value },
  });
};
```

Рисунок 2.6 – Функції обробки подій

Завантаження медіа-даних:

При додаванні зображень використовується окрема асинхронна функція handleUpload, яка завантажує обрані зображення на сервер (за допомогою утиліти upload), а потім оновлює стан:

```
const handleUpload = async () => {
  setUploading(true);
  try {
    const cover = await upload(singleFile);

    const images = await Promise.all(
      [...files].map(async (file) => {
        const url = await upload(file);
        return url;
      })
    );
    setUploading(false);
    dispatch({ type: "ADD_IMAGES", payload: { cover, images } });
  } catch (err) {
    console.log(err);
  }
};
```

Рисунок 2.7 – Функції

Додавання додаткових характеристик

Для можливості додавання особливих характеристик послуги (features), реалізовано окрему форму та функцію обробки:

```
const handleFeature = (e) => {
  e.preventDefault();
  dispatch({
    type: "ADD_FEATURE",
    payload: e.target[0].value,
  });
  e.target[0].value = "";
};
```

Рисунок 2.8 – Функції обробки

Надсилання замовлення на сервер

Після заповнення форми користувач натискає кнопку Create, що викликає функцію handleSubmit. У ній використовується React Query (useMutation) для здійснення POST-запиту до бекенду:

```
const mutation = useMutation({
  mutationFn: (gig) => {
    return newRequest.post("/gigs", gig);
  },
  onSuccess: () => {
    queryClient.invalidateQueries(["myGigs"]);
  },
});

const handleSubmit = (e) => {
  e.preventDefault();
  mutation.mutate(state);
};
```

Рисунок 2.9 – Функції кнопки

У тілі запиту передаються всі введені дані (стан форми state), включаючи:

– title — заголовок послуги

– shortTitle — короткий заголовок

– desc — опис

– shortDesc — короткий опис

– category — категорія

– deliveryTime — термін виконання

– revisionNumber — кількість ревізій

– features — масив особливих характеристик

– cover, images — URL-адреси зображень

– price — ціна послуги

На рисунку 2.10 представлено фрагмент графічного інтерфейсу, що реалізує форму для створення нового замовлення у веб-платформі для надання фріланс послуг.

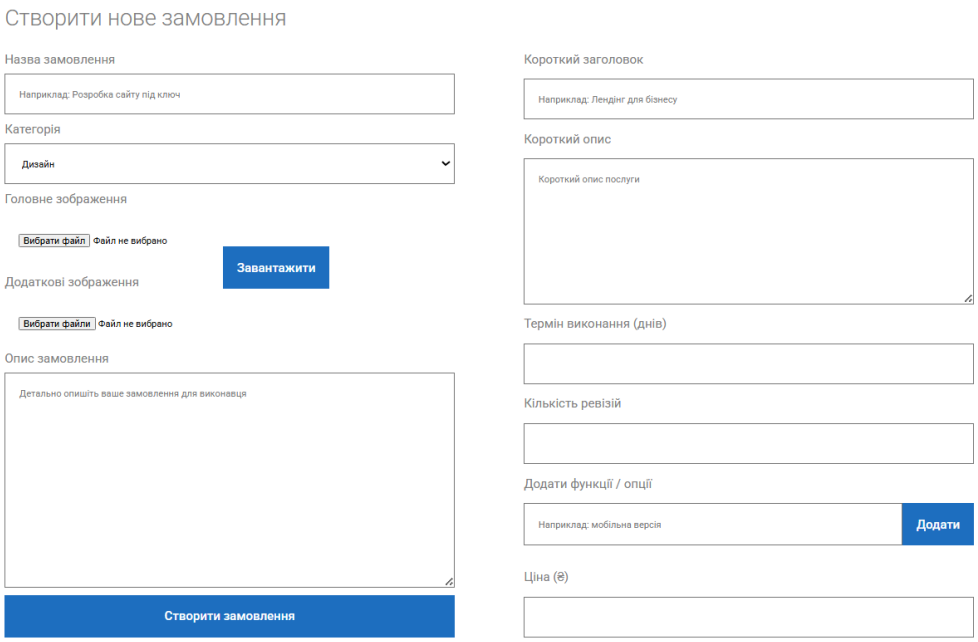

Рисунок 2.10 – Сторінка створення замовлення

Функціонал проведення оплати за замовлення

Для реалізації функціоналу оплати за замовлення на платформі було інтегровано платіжний сервіс Stripe, що дозволяє здійснювати безпечні транзакції між замовниками та фрілансерами.

В основі реалізації лежить компонент Pay.jsx (див. додаток А), який використовує Stripe API для створення платіжного наміру (payment intent) та обробки введення платіжних даних.

Основні кроки процесу:

– При переході користувача на сторінку оплати формується payment intent на інформаційній системі. Для цього використовується endpoint /orders/create-payment-intent/:id, куди передається ID замовлення. Запит виконується у useEffect після завантаження сторінки. Після отримання відповіді сервер надсилає у відповідь clientSecret, який потрібен для ініціалізації Stripe Elements (див. рис.2.11).

```
const res = await newRequest.post(`/orders/create-payment-intent/${id}`);
setClientSecret(res.data.clientSecret);
```

Рисунок 2.11 – Лістинг endpoint

На клієнті використовується бібліотека @stripe/react-stripe-js, яка забезпечує готові UI-компоненти для відображення платіжної форми (див. рис.2.12).

```
{clientSecret && (
    <Elements options={options} stripe={stripePromise}>
      <CheckoutForm />
    </Elements>
)}
```

Рисунок 2.12 – Виконання бібліотеки

Компонент CheckoutForm (підключається у Pay.jsx) містить форму для введення карткових даних та підтвердження платежу

2.5 Тестування платформи для надання фріланс послуг

Перший етап ручного тестування платформи розпочався з перевірки процесу реєстрації нового користувача. Усі обов’язкові поля форми — ім’я, електронна адреса та пароль — були заповнені. Після підтвердження дії через кнопку «Зареєструватися» система коректно обробила запит, створивши новий обліковий запис користувача, про що було повідомлено за допомогою відповідного повідомлення на екрані (див. рис. 2.13).

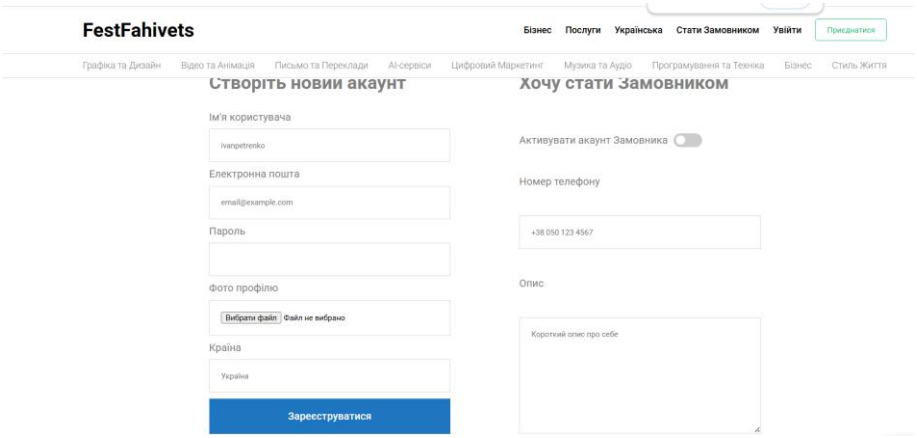

Рисунок 2.13 – Реєстрація нового користувача

Після створення облікового запису було проведено тестування процедури авторизації. У відповідній формі введено адресу електронної пошти та пароль, які були задані під час реєстрації. Після натискання кнопки «Увійти» система успішно здійснила вхід, відобразивши сторінку профілю з актуальними персональними даними користувача та надавши доступ до інших функціональних розділів веб-платформи (див. рис. 2.14).

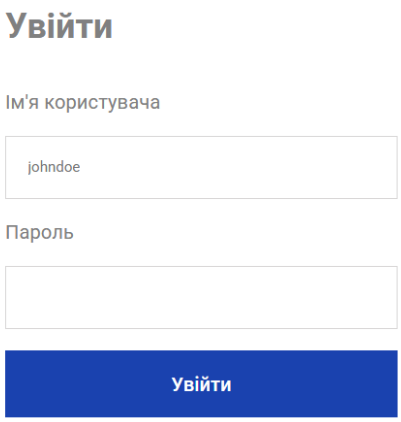

Рисунок 2.14 – Сторінка входу в систему

Інтерфейс головної сторінки платформи представлено на рисунку 2.15.

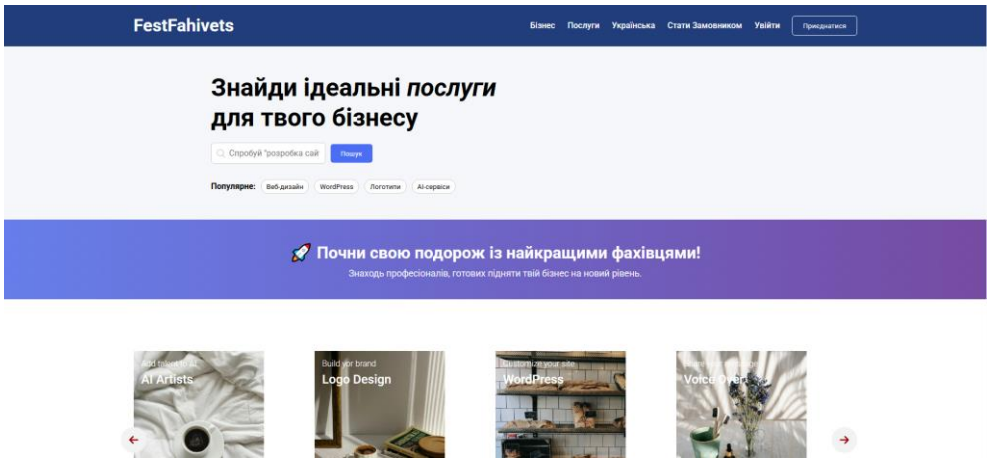

Рисунок 2.15 – Головна сторінка

Також виконано тестування ключового функціоналу — створення нового замовлення. У відповідній формі було заповнено дані: заголовок замовлення,

короткий опис, термін виконання, ціна та прикріплено зображення (див.рис.2.16).

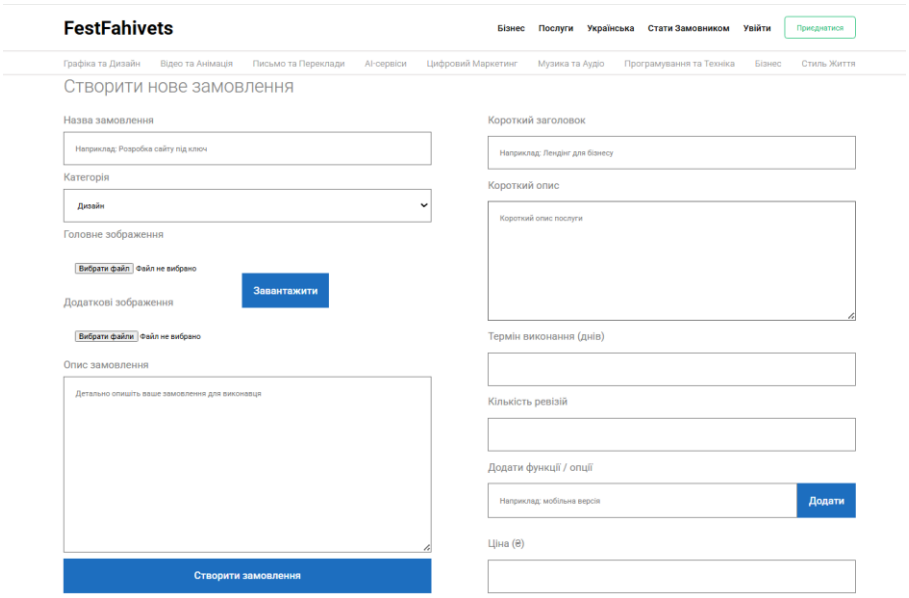

Рисунок 2.16 – Процес створення замовлення. Заповнення форми

Після заповнення форми натиснуто кнопку «Створити», замовлення було успішно додано у відповідний розділ платформи.

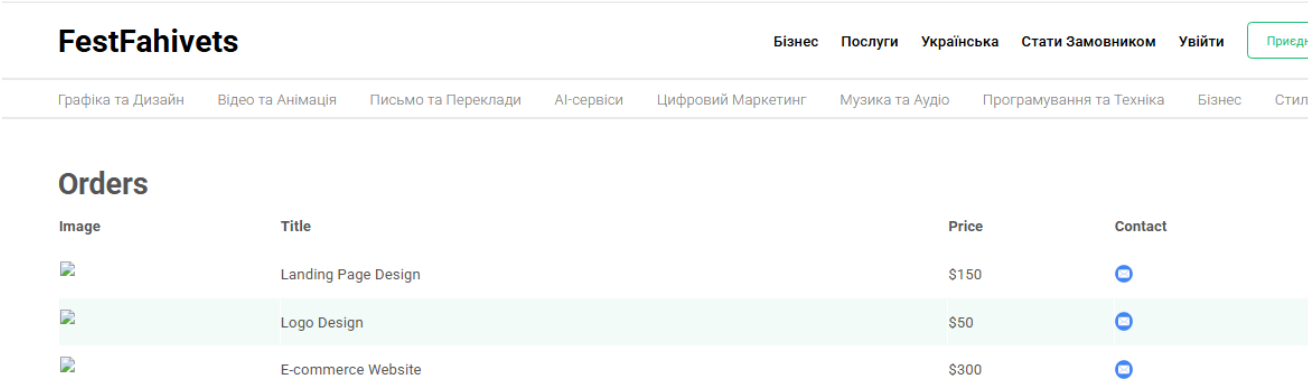

Рисунок 2.17 — Створене замовлення у списку замовлень

Також перевірено процес оплати за замовлення. Замовник після підтвердження результату перейшов на сторінку оплати. Було успішно створено платіжний намір через інтеграцію Stripe. Після введення платіжних даних транзакцію підтверджено, статус замовлення оновився на «оплачено».

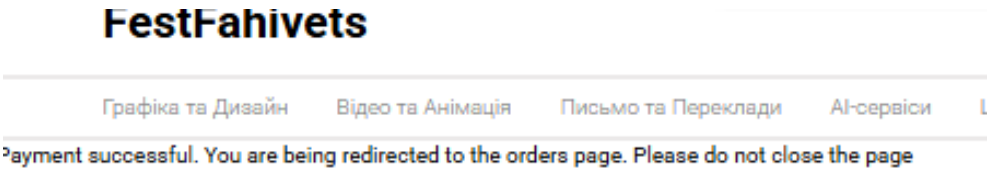

Рисунок 2.18 — Проведення оплати за замовлення

Таким чином, ключові сценарії використання платформи — реєстрація, авторизація, створення замовлення, виконання замовлення, комунікація в чаті, проведення оплати та отримання сповіщень — були успішно протестовані та працюють відповідно до вимог.

Метою автоматизованого тестування було забезпечити перевірку стабільності та правильності роботи механізму входу до системи за логіном і паролем на фріланс-платформі. Для реалізації тестів використовувалося REST API у поєднанні з Postman, що дало змогу автоматизувати тестові сценарії, спростити процес перевірки та зручно аналізувати результати [9].

Підготовчі кроки:

- Встановлено необхідні інструменти для роботи з REST API: Postman (для ручного тестування) та налаштовано середовище для надсилання HTTP-запитів.
- Налаштовано середовище з обліковими даними користувачів (фрілансери, замовники).
- Створено POST-запит на endpoint /api/auth/login (авторизація користувача), з передачею параметрів email та password у тілі запиту (формат JSON).
- Реалізовано перевірку відповідей сервера у вкладці Tests Postman, включаючи:
  - Перевірку статус-коду відповіді.
  - Валідацію наявності контенту у відповіді.
  - Перевірку типу контенту відповіді.

Проведено автоматизовані серії тестів для різних сценаріїв: із валідними та невалідними обліковими даними.

Під час автоматизованого тестування було проведено тест з використанням різних наборів даних. Тестування проходило у кілька етапів, результати наступні:

Таким чином, автоматизоване тестування дозволило виявити як позитивні моменти (наявність відповіді від сервера, робота базових механізмів взаємодії через API) (див. рис. 2.9).

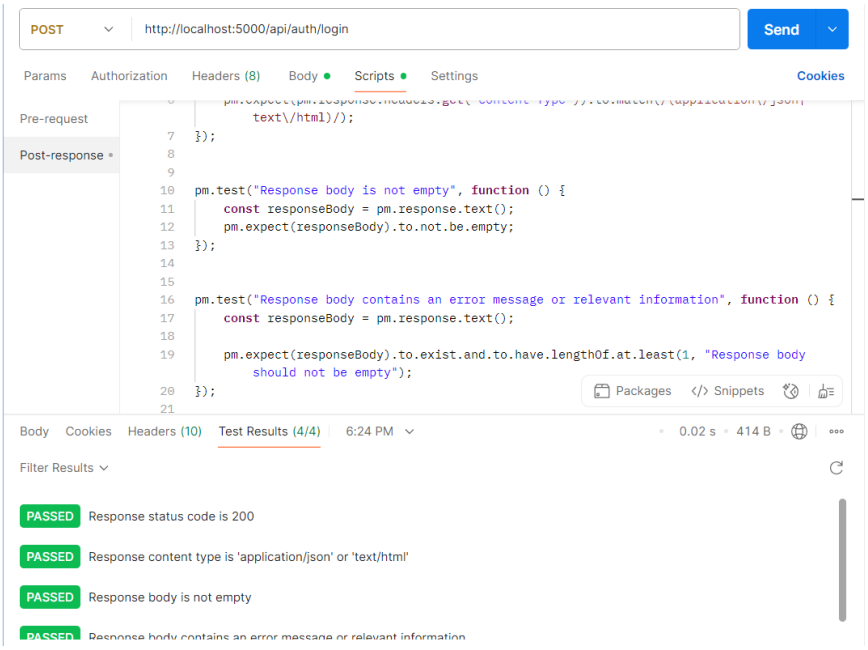

Рисунок 2.19 — Проведення оплати за замовлення

Успішне виконання всіх тестових сценаріїв підтвердило готовність функціоналу авторизації до реального використання. Система забезпечує належний рівень безпеки, зручність доступу та стабільність роботи в різноманітних умовах. Завдяки впровадженню автоматизованого тестування вдалося оперативно виявити й усунути потенційні недоліки, що суттєво підвищило надійність функціоналу входу та підготувало його до повноцінної експлуатації [8].

3.БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Порядок надання домедичної допомоги постраждалим при раптовій зупинці серця

Раптова зупинка серця (РЗС) — це стан, при якому серце раптово припиняє ефективно перекачувати кров через організм, що призводить до припинення кровопостачання життєво важливих органів, насамперед мозку. Без негайного надання допомоги смерть може настати вже через 4–6 хвилин через ішемічне ушкодження мозку. Статистика свідчить, що у разі оперативного надання домедичної допомоги шанси на виживання потерпілого значно зростають [14].

Ключовими етапами надання допомоги є:

1. Оцінка ситуації та забезпечення безпеки. Перш ніж наблизитися до постраждалого, рятівник повинен переконатися, що місце події безпечне для нього та інших осіб. Це дозволяє уникнути вторинних інцидентів.

2. Перевірка стану постраждалого. Необхідно визначити наявність свідомості та нормального дихання. Для цього слід голосно звернутися до людини, злегка потрясти її за плечі. За відсутності реакції слід перевірити дихання — візуально оцінити підйоми грудної клітки, прислухатися до дихання, відчути повітря на щоці. Якщо дихання відсутнє або є агональним (рідке, судомне), потрібно негайно розпочати СЛР.

3. Виклик бригади екстреної медичної допомоги. Необхідно викликати «103» або «112», чітко вказавши місцезнаходження, стан потерпілого та кількість людей на місці події.

4. Проведення серцево-легеневої реанімації (СЛР). Рекомендовано негайно розпочати компресії грудної клітки:

- частота: 100–120 натискань на хвилину;
- глибина компресій: 5–6 см;
- співвідношення компресій до вдихів — 30:2 (для підготовлених рятівників); якщо штучна вентиляція не можлива — безперервні компресії [4].

5. Використання автоматичного зовнішнього дефібрилятора (АЗД). За його наявності АЗД слід негайно застосувати. Пристрій самостійно аналізує серцевий ритм і за необхідності подає дефібриляційний розряд. Роль АЗД у порятунку життя є доведеною: своєчасне використання дефібрилятора протягом перших 3–5 хвилин збільшує шанси на виживання до 70 %.

6. Продовження реанімаційних заходів. Компресії слід виконувати до прибуття бригади медиків або до відновлення самостійного дихання та кровообігу.

Відповідно до Закону України «Про охорону праці» та чинних нормативів, роботодавці зобов’язані організовувати навчання персоналу з питань домедичної допомоги, а також обладнувати приміщення підприємств засобами першої допомоги, включаючи аптечки та дефібрилятори [17].

Практичні аспекти впровадження СЛР на підприємствах:

- регулярне навчання та тренування персоналу;
- встановлення дефібриляторів у громадських місцях та великих офісних будівлях;
- проведення тренувань із моделювання ситуацій РЗС.

Таким чином, наявність системного підходу до підготовки персоналу та оснащення підприємств засобами першої допомоги є критично важливим для зниження летальності при раптовій зупинці серця.

3.2 Протипожежні заходи на підприємстві, в офісі

Забезпечення пожежної безпеки — це ключова складова системи управління охороною праці підприємства. Пожежа може призвести до значних людських втрат, матеріальних збитків та порушення виробничих процесів. Тому організація ефективної системи протипожежного захисту має бути одним із пріоритетних завдань керівництва будь-якого підприємства чи офісу [17].

Основні компоненти системи пожежної безпеки:

1 Нормативно-правова база. Основою є Правила пожежної безпеки в Україні (Наказ МВС №1417 від 30.12.2014 р.) а також інструкції з пожежної безпеки, що розробляються на підприємстві відповідно до специфіки діяльності [18].

2 Організаційні заходи. У структурі підприємства призначається відповідальна особа за пожежну безпеку. Розробляється план евакуації, що розміщується у доступних для персоналу місцях.

3 Навчання персоналу. Всі працівники зобов’язані проходити інструктажі з пожежної безпеки:

- первинний;
- повторний (не рідше одного разу на 6 міс.);
- позаплановий;
- цільовий [19].

Практичні тренування з евакуації та застосування засобів пожежогасіння повинні проводитися щорічно. Підготовленість персоналу безпосередньо впливає на ефективність дій у разі виникнення пожежі.

4 Технічні заходи:

- встановлення систем автоматичної пожежної сигналізації (АПС);
- системи оповіщення та управління евакуацією (відповідно до ДБН В.1.1-7-2016) [20];
- забезпечення приміщень первинними засобами пожежогасіння згідно з ДСТУ EN ISO 7010:2019.[19];
- регулярний контроль стану електромереж та електрообладнання.

5 Шляхи евакуації. Повинні бути:

- вільними від будь-яких перешкод;
- позначені світловими покажчиками;
- двері мають відчинятися у напрямку евакуації.

6 Контроль та відповідальність. Постійний контроль за виконанням вимог пожежної безпеки здійснюється як внутрішніми службами підприємства, так і

державними органами пожежного нагляду.

Особливості впровадження протипожежних заходів в офісах IT-компаній та стартапів:

- офісна техніка (сервери, системи безперервного живлення) є джерелом потенційної пожежної небезпеки;
- обов’язкове використання сертифікованих джерел живлення та якісного електрообладнання;
- організація системного моніторингу температурного режиму серверних кімнат.

Висновок: ефективна система протипожежного захисту підприємства базується на трьох складових:

- нормативному регламентуванні;
- постійному навчанні персоналу;
- сучасному технічному оснащенні.

Завдяки цим заходам підприємства та офіси можуть значно знизити ризики виникнення пожежі та забезпечити захист життя і здоров’я своїх працівників.

ВИСНОВКИ

Розробка веб-платформи для надання фріланс-послуг із використанням технологічного стеку MERN (MongoDB, Express.js, React, Node.js) була успішно завершена. В процесі виконання дипломного проєкту було досягнуто всіх поставлених цілей і виконано повний цикл розробки сучасної веб-системи: від аналізу вимог до впровадження готового застосунку та тестування.

На початковому етапі було обґрунтовано актуальність створення подібної платформи. У сучасному світі онлайн-фріланс є одним з найбільш динамічних сегментів цифрової економіки. Попит на якісні майданчики для взаємодії між замовниками та виконавцями постійно зростає, а наявні рішення не завжди відповідають вимогам зручності, безпеки та функціональності. Тому створення власної платформи дозволяє закрити ці потреби та запропонувати конкурентоспроможний продукт.

У процесі розробки було детально опрацьовано функціональні вимоги системи. Платформа дозволяє проходити процес реєстрації та автентифікації користувачів, створювати профілі, додавати власні послуги, здійснювати пошук по послугах, оформлювати нові замовлення, переглядати замовлення, взаємодіяти у чаті між замовником і виконавцем, проводити оплату замовлень через Stripe, залишати відгуки після завершення робіт, а також отримувати сповіщення про ключові події. Усі ці функції були успішно реалізовані.

Нефункціональні вимоги включали забезпечення безпеки даних користувачів (використання JWT для авторизації, шифрування паролів, захищене з’єднання HTTPS), високу продуктивність (швидкий рендеринг сторінок завдяки React), масштабованість архітектури (використання NoSQL бази даних MongoDB, REST API) та зручний користувацький інтерфейс, що відповідає сучасним стандартам UX/UI.

Було розроблено чітку структуру клієнтської частини на базі React. Для бекенду використовувався Node.js у поєднанні з Express.js. Інтерактивність

інтерфейсу забезпечувалася компонентною архітектурою React, що дозволяє легко масштабувати й підтримувати проєкт у майбутньому. База даних MongoDB забезпечила гнучке зберігання даних, що особливо важливо для проєкту з динамічною структурою (наприклад, послуги можуть мати різні параметри, а замовлення — різні сценарії виконання).

Платформа пройшла комплексне тестування, яке включало ручні тести користувацького інтерфейсу, перевірку функціоналу для різних ролей користувачів (гості, зареєстровані замовники, фрілансери), а також автоматизовані тести REST API для перевірки стабільності серверної частини. Особливу увагу було приділено тестуванню процесу створення замовлень, їх оплати та подальшого виконання, що є ключовою бізнес-логікою даної платформи.

У результаті виконаної роботи було створено повнофункціональну сучасну веб-платформу для надання фріланс-послуг, яка відповідає всім визначеним вимогам і забезпечує якісний сервіс для кінцевих користувачів. Система готова до розширення та масштабування в разі зростання аудиторії та додавання нового функціоналу.

Отриманий досвід роботи з React, Node.js, Express.js, MongoDB, інтеграцією платіжних систем, а також із побудовою повноцінної архітектури веб-застосунку є надзвичайно корисним для подальшої професійної діяльності у сфері веб-розробки. Виконання даного проєкту дозволило закріпити знання щодо проєктування, реалізації та тестування сучасних веб-платформ і розвинути навички створення конкурентоспроможного програмного продукту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev

2. Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org

3. MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com

4. Express.js [Електронний ресурс] – Режим доступу до ресурсу: https://expressjs.com

5. JWT (JSON Web Token) [Електронний ресурс] – Режим доступу до ресурсу: https://jwt.io/introduction

6. PlantUML [Електронний ресурс] – Режим доступу до ресурсу: https://plantuml.com

7. Agile Manifesto [Електронний ресурс] – Режим доступу до ресурсу: https://agilemanifesto.org

8. Postman API Platform [Електронний ресурс] – Режим доступу до ресурсу: https://www.postman.com

9. Методичні вказівки до виконання дипломної роботи освітнього рівня - бакалавр студентами усіх форм навчання для напряму підготовки 121 – Інженерія програмного забезпечення/ Укладачі : Петрик М.Р.,Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2016 – 28 с. Смілянський, В. В. Основи роботи з базою даних MongoDB. Київ: Видавництво «Техніка», 2019. 320 с.

10. Гуменюк, І. В. REST API: розробка та впровадження. Київ: Видавництво «Либідь», 2018. 180 с.

11. Сидоренко, Н. П. Вступ до React: практичні приклади. Одеса: Видавництво «Маяк», 2020. 290 с.

12. Данилюк, В. В. Програмування на Node.js: повний курс. Харків: Видавництво «Ранок», 2018. 400 с.

13. Жидецький, В. Ц. Охорона праці користувачів комп’ютерів. Львів:

Видавництво «Афіша», 2020. 176 с.

14. Безпека життєдіяльності та охорона праці: підручник. Харків: ХНУВС, 2021. 308 с.

15. Закон України "Про охорону праці". Київ: Верховна Рада України, 2002 (зі змінами і доповненнями).

16. Андрейчук, Н. І. Охорона праці. Львів: Видавництво «Львівська політехніка», 2021. 276 с.

17. Правила пожежної безпеки в Україні. Наказ МВС №1417 від 30.12.2014 р.

18. ДСТУ EN ISO 7010:2019. Протипожежний захист. Знаки безпеки. Київ: Держспоживстандарт України, 2007. 20 с.

19. ДБН В.1.1-7-2016. Пожежна безпека об’єктів будівництва. Київ: Мінрегіон України, 2016. 64 с.

20. Пістун, І. П. Практикум з безпеки життєдіяльності. Суми: Видавництво «Університетська книга», 2023. 560 с.

21. Дистанційний курс «Кваліфікаційні роботи бакалаврів» сайту дистанційного навчання ТНТУ [Електронний ресурс]. – Режим доступу: URL: https://dl.tntu.edu.ua/bounce.php?course=5329

ДОДАТКИ

Додаток А Лістинг серверної частини програми

Лістинг файлу server.js:

```
import express from "express";
import mongoose from "mongoose";
import dotenv from "dotenv";
import cors from "cors";
import cookieParser from "cookie-parser";

import authRoute from "./routes/auth.route.js"; //

dotenv.config();

const app = express();

app.use(cors({ origin: "http://localhost:5173", credentials: true }));
app.use(express.json());
app.use(cookieParser());

const connect = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI, {
      ssl: true,
      tlsInsecure: true,
    });
    console.log("✅ MongoDB підключено");
  } catch (err) {
    console.error("❌ Помилка підключення до MongoDB", err);
  }
};


app.use("/api/auth", authRoute);

app.use((err, req, res, next) => {
  const errorStatus = err.status || 500;
  const errorMessage = err.message || "Щось пішло не так!";
  return res.status(errorStatus).send(errorMessage);
});

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {
  connect();
  console.log(`✅ Сервер працює на порту ${PORT}`);
});
```

Лістинг файлу user.model.js

```
import mongoose from "mongoose";
const { Schema } = mongoose;

const userSchema = new Schema({
  username: {
    type: String,
```

```
    required: true,
    unique: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
  },
  password: {
    type: String,
    required: true,
  },
  img: {
    type: String,
    required: false,
  },
  country: {
    type: String,
    required: true,
  },
  phone: {
    type: String,
    required: false,
  },
  desc: {
    type: String,
    required: false,
  },
  isSeller: {
    type: Boolean,
    default:false
  },
},{
  timestamps:true
});

export default mongoose.model("User", userSchema)
```

Лістинг файлу order.model.js

```
import mongoose from "mongoose";
const { Schema } = mongoose;


const OrderSchema = new Schema(
  {
    gigId: {
      type: String,
      required: true,
    },
    img: {
      type: String,
      required: false,
```

```
    },
    title: {
      type: String,
      required: true,
    },
    price: {
      type: Number,
      required: true,
    },
    sellerId: {
      type: String,
      required: true,
    },
    buyerId: {
      type: String,
      required: true,
    },
    isCompleted: {
      type: Boolean,
      default: false,
    },
    payment_intent: {
      type: String,
      required: true,
    },
  },
  {
    timestamps: true,
  }
);

export default mongoose.model("Order", OrderSchema);
```

Лістинг файлу auth.route.js

```
import express from "express";
import { register, login, logout } from
"../controllers/auth.controller.js";

const router = express.Router();
```

```
router.post("/register", register)
router.post("/login", login)
router.post("/logout", logout)

export default router;
```

Лістинг файлу order.route.js

```
import express from "express";
import { verifyToken } from "../middleware/jwt.js";
import { getOrders, intent, confirm } from
"../controllers/order.controller.js";

const router = express.Router();

// router.post("/:gigId", verifyToken, createOrder);
router.get("/", verifyToken, getOrders);
router.post("/create-payment-intent/:id", verifyToken, intent);
router.put("/", verifyToken, confirm);

export default router;
```

Лістинг файлу user.model.js

```
import mongoose from "mongoose";
const { Schema } = mongoose;

const userSchema = new Schema({
  username: {
    type: String,
    required: true,
    unique: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
  },
  password: {
    type: String,
    required: true,
  },
  img: {
    type: String,
```

```
    required: false,
  },
  country: {
    type: String,
    required: true,
  },
  phone: {
    type: String,
    required: false,
  },
  desc: {
    type: String,
    required: false,
  },
  isSeller: {
    type: Boolean,
    default:false
  },
},{
  timestamps:true
});

export default mongoose.model("User", userSchema)
```

Лістинг файлу auth.controller.js

```
import User from "../models/user.model.js";
import createError from "../utils/createError.js";
import bcrypt from "bcrypt";
import jwt from "jsonwebtoken";

export const register = async (req, res, next) => {
  try {
    const hash = bcrypt.hashSync(req.body.password, 5);
    const newUser = new User({
      ...req.body,
      password: hash,
    });

    await newUser.save();
    res.status(201).send("User has been created.");
  } catch (err) {
    next(err);
  }
};
export const login = async (req, res, next) => {
  try {
    const user = await User.findOne({ username: req.body.username });

    if (!user) return next(createError(404, "User not found!"));

    const isCorrect = bcrypt.compareSync(req.body.password, user.password);
    if (!isCorrect)
      return next(createError(400, "Wrong password or username!"));

    const token = jwt.sign(
      {
        id: user._id,
```

```
        isSeller: user.isSeller,
      },
      process.env.JWT_KEY
    );

    const { password, ...info } = user._doc;
    res
      .cookie("accessToken", token, {
        httpOnly: true,
      })
      .status(200)
      .send(info);
  } catch (err) {
    next(err);
  }
};

export const logout = async (req, res) => {
  res
    .clearCookie("accessToken", {
      sameSite: "none",
      secure: true,
    })
    .status(200)
    .send("User has been logged out.");
};
```

Лістинг файлу order.controller.js

```
import createError from "../utils/createError.js";
import Order from "../models/order.model.js";
import Gig from "../models/gig.model.js";
import Stripe from "stripe";

export const intent = async (req, res, next) => {
  const stripe = new Stripe(process.env.STRIPE);

  const gig = await Gig.findById(req.params.id);

  const paymentIntent = await stripe.paymentIntents.create({
    amount: gig.price * 100,
    currency: "usd",
    automatic_payment_methods: {
      enabled: true,
    },
  });

  const newOrder = new Order({
    gigId: gig._id,
    img: gig.cover,
    title: gig.title,
    buyerId: req.userId,
    sellerId: gig.userId,
    price: gig.price,
    payment_intent: paymentIntent.id,
  });
```

```
  await newOrder.save();

  res.status(200).send({
    clientSecret: paymentIntent.client_secret,
  });
};

export const getOrders = async (req, res, next) => {
  try {
    const orders = await Order.find({
      ...(req.isSeller ? { sellerId: req.userId } : { buyerId: req.userId
}),
      isCompleted: true,
    });

    res.status(200).send(orders);
  } catch (err) {
    next(err);
  }
};
export const confirm = async (req, res, next) => {
  try {
    const orders = await Order.findOneAndUpdate(
      {
        payment_intent: req.body.payment_intent,
      },
      {
        $set: {
          isCompleted: true,
        },
      }
    );

    res.status(200).send("Order has been confirmed.");
  } catch (err) {
    next(err);
  }
};
```

Додаток Б Лістинг фронтенд частини програми

Лістинг файлу Add.jsx

```
import React, { useReducer, useState } from "react";
import "./Add.scss";
import { gigReducer, INITIAL_STATE } from "../../reducers/gigReducer";
import upload from "../../utils/upload";
import { useMutation, useQueryClient } from "@tanstack/react-query";
import newRequest from "../../utils/newRequest";
import { useNavigate } from "react-router-dom";

const Add = () => {
  const [singleFile, setSingleFile] = useState(undefined);
  const [files, setFiles] = useState([]);
  const [uploading, setUploading] = useState(false);

  const [state, dispatch] = useReducer(gigReducer, INITIAL_STATE);

  const handleChange = (e) => {
    dispatch({
      type: "CHANGE_INPUT",
      payload: { name: e.target.name, value: e.target.value },
    });
  };

  const handleFeature = (e) => {
    e.preventDefault();
    dispatch({
      type: "ADD_FEATURE",
      payload: e.target[0].value,
    });
    e.target[0].value = "";
  };

  const handleUpload = async () => {
    setUploading(true);
    try {
      const cover = await upload(singleFile);

      const images = await Promise.all(
        [...files].map(async (file) => {
          const url = await upload(file);
          return url;
        })
      );
      setUploading(false);
      dispatch({ type: "ADD_IMAGES", payload: { cover, images } });
    } catch (err) {
      console.log(err);
    }
  };

  const navigate = useNavigate();
  const queryClient = useQueryClient();

  const mutation = useMutation({
```

```
  mutationFn: (gig) => {
    return newRequest.post("/gigs", gig);
  },
  onSuccess: () => {
    queryClient.invalidateQueries(["myGigs"]);
  },
});

const handleSubmit = (e) => {
  e.preventDefault();
  mutation.mutate(state);
  // navigate("/mygigs")
};

return (
  <div className="add">
    <div className="container">
      <h1>Створити нове замовлення</h1>
      <div className="sections">
        <div className="info">
          <label htmlFor="">Назва замовлення</label>
          <input
            type="text"
            name="title"
            placeholder="Наприклад: Розробка сайту під ключ"
            onChange={handleChange}
          />
          <label htmlFor="">Категорія</label>
          <select name="cat" id="cat" onChange={handleChange}>
            <option value="design">Дизайн</option>
            <option value="web">Веб-розробка</option>
            <option value="animation">Анімація</option>
            <option value="music">Музика</option>
          </select>
          <div className="images">
            <div className="imagesInputs">
              <label htmlFor="">Головне зображення</label>
              <input
                type="file"
                onChange={(e) => setSingleFile(e.target.files[0])}
              />
              <label htmlFor="">Додаткові зображення</label>
              <input
                type="file"
                multiple
                onChange={(e) => setFiles(e.target.files)}
              />
            </div>
            <button onClick={handleUpload}>
              {uploading ? "Завантаження..." : "Завантажити"}
            </button>
          </div>
          <label htmlFor="">Опис замовлення</label>
          <textarea
            name="desc"
            id=""
            placeholder="Детально опишіть ваше замовлення для виконавця"
```

```
            cols="0"
            rows="16"
            onChange={handleChange}
          ></textarea>
          <button onClick={handleSubmit}>Створити замовлення</button>
        </div>
        <div className="details">
          <label htmlFor="">Короткий заголовок</label>
          <input
            type="text"
            name="shortTitle"
            placeholder="Наприклад: Лендінг для бізнесу"
            onChange={handleChange}
          />
          <label htmlFor="">Короткий опис</label>
          <textarea
            name="shortDesc"
            onChange={handleChange}
            id=""
            placeholder="Короткий опис послуги"
            cols="30"
            rows="10"
          ></textarea>
          <label htmlFor="">Термін виконання (днів)</label>
          <input type="number" name="deliveryTime"
onChange={handleChange} />
          <label htmlFor="">Кількість ревізій</label>
          <input
            type="number"
            name="revisionNumber"
            onChange={handleChange}
          />
          <label htmlFor="">Додати функції / опції</label>
          <form action="" className="add" onSubmit={handleFeature}>
            <input type="text" placeholder="Наприклад: мобільна версія" />
            <button type="submit">Додати</button>
          </form>
          <div className="addedFeatures">
            {state?.features?.map((f) => (
              <div className="item" key={f}>
                <button
                  onClick={() =>
                    dispatch({ type: "REMOVE_FEATURE", payload: f })
                  }
                >
                  {f}
                  <span>X</span>
                </button>
              </div>
            ))}
          </div>
          <label htmlFor="">Ціна (₴)</label>
          <input type="number" onChange={handleChange} name="price" />
        </div>
      </div>
    </div>
```

```
  );
};

export default Add;
```

```
const mutation = useMutation({
  mutationFn: (gig) => {
    return newRequest.post("/gigs", gig);
  },
  onSuccess: () => {
    queryClient.invalidateQueries(["myGigs"]);
  },
});

const handleSubmit = (e) => {
  e.preventDefault();
  mutation.mutate(state);
};
```

Лістинг файлу home.jsx

```
import React from "react";
import "./Home.scss";
import Featured from "../../components/featured/Featured";
import TrustedBy from "../../components/trustedBy/TrustedBy";
import Slide from "../../components/slide/Slide";
import CatCard from "../../components/catCard/CatCard";
import ProjectCard from "../../components/projectCard/ProjectCard";
import { cards, projects } from "../../data";

function Home() {
  return (
    <div className="home">
      <Featured />
      <div className="highlight-banner">
        <div className="container">
          <h2>🚀 Почни свою подорож із найкращими фахівцями!</h2>
          <p>Знаходь професіоналів, готових підняти твій бізнес на новий рівень.</p>
        </div>
      </div>

      <Slide slidesToShow={4} arrowsScroll={4}>
        {cards.map((card) => (
          <CatCard key={card.id} card={card} />
        ))}
      </Slide>

      <div className="features alt">
        <div className="container">
          <div className="item">
            <h1>Чому обирають нас?</h1>
            <ul className="features-list">
```

```
              <li><img src="./img/check.png" alt="check" /> Прозоре
ціноутворення</li>
              <li><img src="./img/check.png" alt="check" /> Перевірені
виконавці</li>
              <li><img src="./img/check.png" alt="check" /> Безпечні
транзакції</li>
              <li><img src="./img/check.png" alt="check" /> Підтримка
24/7</li>
            </ul>
          </div>
          <div className="item">
            <video src="./img/video.mp4" controls className="feature-video"
/>
          </div>
        </div>
      </div>

      <div className="explore alt">
        <div className="container">
          <h1>Ознайомся з категоріями</h1>
          <div className="items-grid">
            {[
              "Графіка та Дизайн",
              "Цифровий Маркетинг",
              "Написання та Переклад",
              "Відео та Анімація",
              "Музика та Аудіо",
              "Програмування та Технології",
              "Бізнес",
              "Стиль життя",
              "Дані",
              "Фотографія",
            ].map((title, i) => (
              <div className="item" key={i}>
                <img
src={`https://source.unsplash.com/100x100/?${title.split(" ")[0]}`}
alt={title} />
                <span>{title}</span>
              </div>
            ))}
          </div>
        </div>
      </div>

      <Slide slidesToShow={3} arrowsScroll={3}>
        {projects.map((card) => (
          <ProjectCard key={card.id} card={card} />
        ))}
      </Slide>
    </div>
  );
}

export default Home;
```

Лістинг файлу Register.jsx

```
import React, { useState } from "react";
import upload from "../../utils/upload";
import "./Register.scss";
import newRequest from "../../utils/newRequest";
import { useNavigate } from "react-router-dom";

function Register() {
  const [file, setFile] = useState(null);
  const [user, setUser] = useState({
    username: "",
    email: "",
    password: "",
    img: "",
    country: "",
    isSeller: false,
    desc: "",
  });

  const navigate = useNavigate();

  const handleChange = (e) => {
    setUser((prev) => {
      return { ...prev, [e.target.name]: e.target.value };
    });
  };

  const handleSeller = (e) => {
    setUser((prev) => {
      return { ...prev, isSeller: e.target.checked };
    });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();

    const url = await upload(file);
    try {
      await newRequest.post("/auth/register", {
        ...user,
        img: url,
      });
      navigate("/");
    } catch (err) {
      console.log(err);
    }
  };

  return (
    <div className="register">
      <form onSubmit={handleSubmit}>
        <div className="left">
          <h1>Створіть новий акаунт</h1>
          <label htmlFor="">Ім'я користувача</label>
          <input
            name="username"
            type="text"
            placeholder="ivanpetrenko"
            onChange={handleChange}
```

```
          />
          <label htmlFor="">Електронна пошта</label>
          <input
            name="email"
            type="email"
            placeholder="email@example.com"
            onChange={handleChange}
          />
          <label htmlFor="">Пароль</label>
          <input name="password" type="password" onChange={handleChange} />
          <label htmlFor="">Фото профілю</label>
          <input type="file" onChange={(e) => setFile(e.target.files[0])}
/>
          <label htmlFor="">Країна</label>
          <input
            name="country"
            type="text"
            placeholder="Україна"
            onChange={handleChange}
          />
          <button type="submit">Зареєструватися</button>
        </div>
        <div className="right">
          <h1>Хочу стати Замовником</h1>
          <div className="toggle">
            <label htmlFor="">Активувати акаунт Замовника</label>
            <label className="switch">
              <input type="checkbox" onChange={handleSeller} />
              <span className="slider round"></span>
            </label>
          </div>
          <label htmlFor="">Номер телефону</label>
          <input
            name="phone"
            type="text"
            placeholder="+38 050 123 4567"
            onChange={handleChange}
          />
          <label htmlFor="">Опис</label>
          <textarea
            placeholder="Короткий опис про себе"
            name="desc"
            cols="30"
            rows="10"
            onChange={handleChange}
          ></textarea>
        </div>
      </form>
    </div>
  );
}

export default Register;
```

Додаток В Диск з роботою