

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Вебдодаток для пошуку та співпраці з незалежними виконавцями»

Виконав студент групи ІПЗ-21-2
ОЧКОВСЬКА Катерина Олександрівна

Керівник роботи:
ЧИЖМОТРЯ Олена Геннадіївна

Рецензент:
ВАКАЛЮК Тетяна Анатоліївна

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«14» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ОЧКОВСЬКА Катерина Олександрівна**

Керівник роботи: **ЧИЖМОТРЕ Олена Геннадіївна**

Тема роботи: **«Вебдодаток для пошуку та співпраці з незалежними виконавцями»**,

затверджена Наказом закладу вищої освіти від «14» лютого 2025 р., №61/с

Вихідні дані для роботи: предметом дослідження є підходи, інструменти та технології веброзробки; об'єктом дослідження є процес розробки вебдодатку для пошуку та взаємодії з незалежними виконавцями.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.Г.	14.02.2025р.	14.02.2025р.
2	Чижмотря О.Г.	14.03.2025р.	14.03.2025р.
3	Чижмотря О.Г.	14.04.2025р.	14.04.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з програмного продукту «Вебдодаток для пошуку та співпраці з незалежними виконавцями» та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи бакалавра містить текстову частину, викладену на 77 сторінках друкованого тексту, та має 3 додатки. Список використаних джерел містить 20 найменувань. У роботі наведено 22 рисунки та 13 таблиць.

Мета цієї роботи полягає у створенні сучасного та функціонального вебдодатку, який дозволить ефективно знаходити виконавців та замовлення відповідно до актуальних потреб користувачів. Основним завданням є розробка вебплатформи, що забезпечуватиме зручний механізм пошуку за різними критеріями, швидку та точну обробку запитів, а також зрозумілий інтерфейс для комфортної співпраці з незалежними виконавцями.

У рамках кваліфікаційної роботи було проведено ґрунтовний аналіз, ретельне проєктування та повномасштабна розробка вебдодатку, призначеного для пошуку та співпраці з незалежними виконавцями. Визначено та реалізовано архітектуру системи, а також ключові функціональні можливості, які забезпечують зручну та ефективну взаємодію між користувачами.

На основі зібраних даних і встановлених вимог розроблено вебдодаток, що відповідає сучасним стандартам зручності та функціональності. Його реалізація стала важливим кроком у створенні комфортного середовища для взаємодії фрилансерів із замовниками, а також заклала основу для подальшого розвитку платформи та впровадження нових можливостей.

КЛЮЧОВІ СЛОВА: ВЕБДОДАТОК, СПІВПРАЦЯ, ВИКОНАВЕЦЬ, ЗАМОВНИК, ФРИЛАНС, ПРОЄКТ

					ІПЗ.КР.Б – 121 – 25 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Вебдодаток для пошуку та співпраці з незалежними виконавцями	Літ.	Арк.	Аркушів
Розроб.		Очковська К.О.						
Керівник		Чижмоторя О.Г.					3	77
Рецензент		Вакалюк Т.А.				Житомирська політехніка, група ІПЗ		
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The bachelor's qualification thesis consists of a software product «Web Application for Finding and Collaborating with Independent Contractors» and an accompanying explanatory note. The explanatory note includes a textual component presented on 77 pages of printed text and contains 3 appendices. The list of references comprises 20 sources. The work also includes 22 illustrations and 13 tables.

The goal of this project is to develop a modern and functional website that enables users to efficiently find freelancers or project opportunities according to current user needs. The primary objective is the development of a web platform that provides a convenient search mechanism based on multiple criteria, fast and accurate request processing, and an intuitive interface for seamless collaboration with independent contractors.

As a part of this qualification thesis, a thorough analysis, careful design, and full-scale development of the website for freelancer-client collaboration were carried out. The system architecture was defined and implemented, along with the key functional features that enable convenient and efficient user interaction.

Based on the collected data and identified requirements, a website was created that meets modern standards of usability and functionality. Its implementation represents an important step in building a user-friendly environment for collaboration between freelancers and clients, and it lays the groundwork for the platform's future development and the introduction of new features.

KEYWORDS: WEBAPPLICATION, COLLABORATION, CONTRACTOR, CLIENT, FREELANCE, PROJECT

					ІІІ3.КР.Б – 121 – 25 – ІІ3	Арк.
						4
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	7
ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБДОДАТКУ ДЛЯ СПІВПРАЦІ З НЕЗАЛЕЖНИМИ ВИКОНАВЦЯМИ.....	11
1.1. Опис проєкту та постановка задачі	11
1.2. Дослідження та порівняння функціональних можливостей аналогів програмного продукту	12
1.3. Вибір архітектури вебдодатку	17
1.4. Обґрунтування вибору інструментальних засобів для розробки вебдодатку	20
Висновки до першого розділу.....	21
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБДОДАТКУ ДЛЯ СПІВПРАЦІ З НЕЗАЛЕЖНИМИ ВИКОНАВЦЯМИ.....	23
2.1. Визначення варіантів використання та структури системи.....	23
2.2. Розробка бази даних вебдодатку	27
2.3. Реалізація функціоналу роботи вебдодатку для співпраці між незалежними виконавцями.....	34
Висновки до другого розділу	39
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБДОДАТКОМ ДЛЯ ПОШУКУ ТА СПІВПРАЦІ З НЕЗАЛЕЖНИМИ ВИКОНАВЦЯМИ	41
3.1. Порядок налаштування та встановлення системи.....	41
3.2. Структура інтерфейсу та порядок взаємодії з вебдодатком для пошуку та співпраці з незалежними виконавцями.....	45
3.3. Тестування роботи розробленого вебдодатку.....	54
Висновки до третього розділу.....	58

ВИСНОВКИ.....	60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	62
ДОДАТКИ.....	64

					ПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

JWT – JSON Web Token

NFR – non-functional requirements

SQL – Structured Query Language

UML – Unified Modeling Language

СУБД – система управління базами даних

					ІІЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми: попри складну економічну ситуацію та загальну кризу в країні, сфера фрилансу продовжує не лише утримувати свої позиції, а й динамічно розвиватися. Світовий досвід підтверджує, що співпраця з незалежними виконавцями дозволяє компаніям оптимізувати витрати, водночас надаючи фрилансерам гнучкі умови праці, можливість самостійно обирати напрямок діяльності та регулювати власний робочий графік.

Швидкий розвиток сучасних технологій, особливо в сфері інформаційних та комунікаційних рішень, сприяє зміні традиційних підходів до зайнятості. Це зумовлює зростання вимог до фахівців і поступове зменшення ролі стандартних форм працевлаштування. Як наслідок, концепція фіксованого робочого місця втрачає свою актуальність, поступаючись місцем більш гнучким моделям співпраці.

З огляду на ці тенденції, розробка вебдодатку для пошуку та співпраці з незалежними виконавцями є надзвичайно актуальною. Така платформа сприятиме ефективному об'єднанню замовників і фрилансерів, спрощуючи процеси пошуку, відбору та комунікації. Вона дозволить компаніям швидко знаходити кваліфікованих спеціалістів без необхідності довгострокових зобов'язань, а виконавцям – отримувати стабільний потік замовлень відповідно до своїх навичок і вподобань.

Окрім спрощення процесу пошуку та комунікації, цей вебдодаток надасть користувачам широкий спектр можливостей. Виконавці можуть створювати детальні профілі, додавати портфоліо, зазначати навички та отримувати відгуки від замовників, що сприятиме їхній професійній репутації. Замовники, в свою чергу, отримають зручні інструменти для розміщення проєктів, підбору кандидатів та управління співпрацею. Все це значно підвищить зручність та прозорість взаємодії між усіма учасниками процесу. І, таким чином, створення такого вебдодатку є важливою ланкою у розвитку сучасного цифрового ринку праці.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

Метою кваліфікаційної роботи є створення функціонального та сучасного вебдодатку, призначеного для ефективного пошуку виконавців та замовлень, який відповідатиме актуальним потребам користувачів.

Процес створення зручної та надійної платформи для співпраці замовників та виконавців є поступовим вирішенням низки важливих задач, які визначають головні виклики та ключові пріоритети його реалізації.

Для досягнення поставленої мети визначено наступні завдання:

1. Розробити систему реєстрації та авторизації для замовників та виконавців;
2. Створити особисті кабінети для наповнення та редагування профілю;
3. Реалізувати пошук виконавців з розширеною системою фільтрації (за сферою діяльності, вартістю, рейтингом, регіоном);
4. Забезпечити можливість замовникам переглядати профіль виконавця (його портфоліо та відгуки);
5. Додати функціонал для створення замовлень (опис завдання, бюджет, терміни виконання роботи) та управління ними;
6. Впровадити механізм для подачі заявок виконавцями та вибору найкращої пропозиції замовником;
7. Забезпечити ефективну взаємодію в ході співпраці (внутрішній чат для обговорення деталей проєкту);
8. Реалізувати механізм сповіщень про нові повідомлення чи будь-які змін щодо замовлення;
9. Запровадити можливість оцінювання виконавців та замовників після завершення замовлення (відгуки та коментарі);
10. Реалізувати збереження історії всіх дій щодо проєкту (початок роботи, внесення змін, статуси);
11. Розробити панель адміністратора для модерації співпраць, вирішення конфліктів та обробки скарг.

Об'єктом дослідження є процес проєктування, розробки та впровадження вебдодатку для організації пошуку і взаємодії з незалежними виконавцями.

					ІПЗ.КР.Б – 121 – 25 – ІЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

Предметом дослідження є підходи, інструменти та технології веброзробки, що застосовуються для створення вебдодатку з пошуку та організації співпраці з незалежними виконавцями.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБДОДАТКУ ДЛЯ СПІВПРАЦІ З НЕЗАЛЕЖНИМИ ВИКОНАВЦЯМИ

1.1. Опис проєкту та постановка задачі

Експерти ринку праці прогнозують, що в найближчі роки Україна може стати однією з провідних європейських країн за кількістю фрилансерів, і їхня частка в загальній структурі зайнятості може зрости до 15–20% населення. Це свідчить про подальший розвиток дистанційної роботи та зростаючий попит на незалежних виконавців як серед вітчизняних, так і міжнародних компаній.

Проте, попри численні переваги фрилансу, взаємодія між замовниками та виконавцями часто супроводжується певними труднощами, такими як пошук надійних партнерів, ризик невиконання зобов'язань або складнощі з організацією комунікації і тд. Саме тому виникає потреба у створенні платформи, яка допоможе автоматизувати та спростити процес співпраці між сторонами.

Головною задачею цього проєкту є розробка вебдодатку, який має забезпечити зручний механізм пошуку, комунікації та оцінки роботи, що сприятиме підвищенню ефективності та безпеки фриланс-взаємодії.

Реалізація цього основного функціоналу дозволить розробити ефективний вебдодаток для співпраці роботодавців з незалежними виконавцями. Усі завдання мають на меті спростити взаємодію між замовниками та фрилансерами, а також забезпечити найкращий досвід використання платформи.

Також не менш важливими є нефункціональні вимоги, що мають великий вплив на загальну якість роботи системи та її ефективність. NFR (non-functional requirements) або нефункціональні вимоги – це набір специфікацій, які описують робочі характеристики та обмеження системи, що мають на меті покращення її загальної ефективності. Вони визначають, наскільки коректно працюватиме продукт.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

Нефункціональні вимоги, які повинні бути обов'язково враховані при реалізації вебдодатку, включають:

1. Система повинна забезпечувати стабільну роботу та постійну доступність, що є ключовими характеристиками її надійності.
2. Інтерфейс вебдодатку має бути простим, інтуїтивно зрозумілим і зручним для користувачів, що відповідає вимозі високої зручності використання (usability).
3. Швидкість реакції системи на дії користувача повинна бути максимально короткою, зокрема під час завантаження сторінок або обробки транзакцій, що визначає рівень її продуктивності.

Таким чином, проведений аналіз проєкту та виокремлення основних задач дозволяє визначити чіткий напрямок та хід робіт для реалізації вебдодатку для співпраці з незалежними виконавцями, що забезпечить його ефективне функціонування, відповідність усім потребам користувачів та можливість подальшого масштабування.

1.2. Дослідження та порівняння функціональних можливостей аналогів програмного продукту

Розробка власного програмного продукту потребує ретельного аналізу вже існуючих аналогів на ринку. Щоб створити конкурентоспроможний продукт, варто дослідити та порівняти усі функціональні можливості аналогічних платформ, визначити їхні переваги та недоліки. Все це дозволить знайти якомога кращі рішення для користувача та врахувати їх при реалізації власного програмного продукту.

Дослідження ринку бірж фрилансу, окрім знаходження необхідного функціоналу та доступних можливостей, дозволить також уникнути вже допущених помилок. Таким чином, створений вебдодаток для співпраці незалежних виконавців буде не черговим типовим рішенням, а матиме кращий рівень зручності, якості та комфорту для користувачів.

					ІПЗ.КР.Б – 121 – 25 – ІЗ	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

Для проведення дослідження та порівняння функціональних можливостей існуючих аналогів програмного продукту було обрано дві українські платформи: Freelancehunt, Freelance.ua – та одну міжнародну: Upwork.

Freelancehunt (рис. 1.1) – одна з найпопулярніших та найбільших українських платформ для фрилансерів, що була заснована у 2006 році [3]. З нею співпрацюють найприбутковіші українські, і не тільки, бізнеси, серед яких Brain, ЛУН, AUTO.RIA, Mastercard та інші.

Біржа працює за класичним принципом роботи. Клієнт (замовник) розміщує проєкт з описом задач, які необхідно виконати, та додає вартість даної роботи. Виконавець, в свою чергу, може запропонувати свою кандидатуру, якщо його зацікавила пропозиція. На Freelancehunt є можливість підібрати собі спеціаліста з різних галузей, зокрема перекладу, розробки програмного забезпечення, просування, реклами, а також дизайну.

На платформі присутня комісія, що становить 9-10 % від суми замовлення. Вона може сплачуватися як замовником, так і фрилансером.

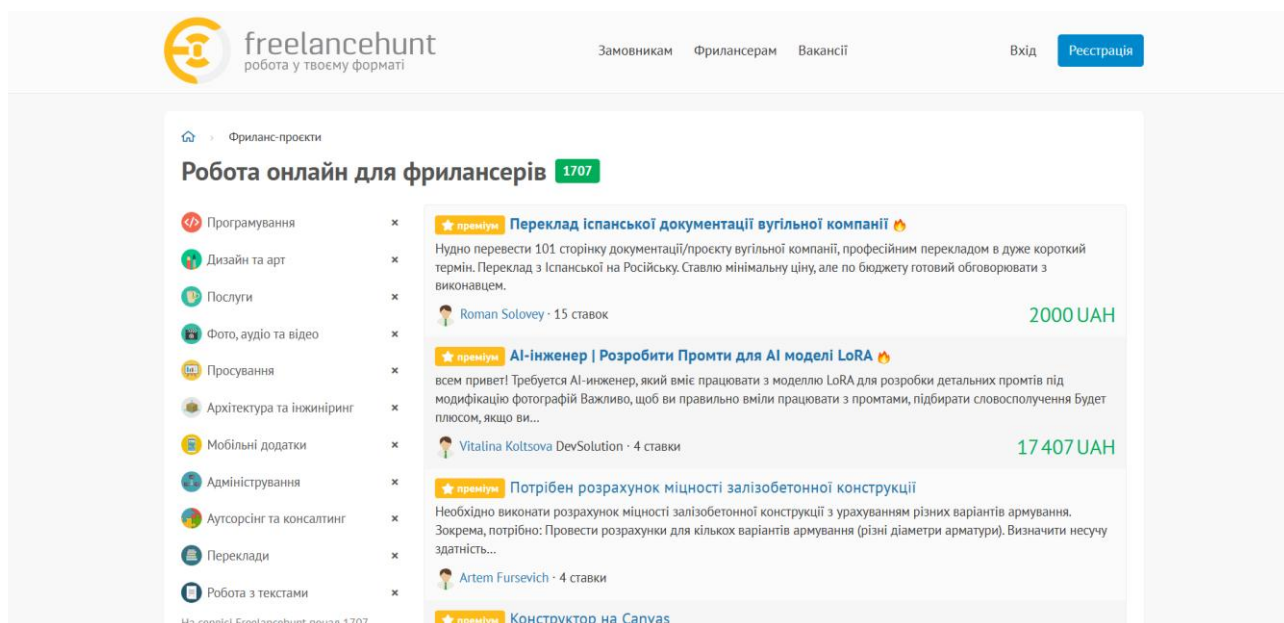


Рисунок 1.1 – Сторінка пошуку фриланс-проєктів на сайті Freelancehunt

Водночас платформа має і певні недоліки. Зокрема, висока конкуренція може ускладнити отримання перших замовлень для новачків. Фрилансери також повинні пройти платну верифікацію профілю.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		13

Окрім системи рейтингу, активності та якісного виконання замовлень, для підвищення видимості профілю та більшої кількості проєктів доведеться придбати підписку Plus. Коштує така опція: 399 грн – на 1 місяць або 3990 грн – на 12 місяців.

Freelance.ua (рис. 1.2) – також не менш популярна онлайн-платформа для пошуку співпраць та виконання фриланс-замовлень [4]. Серед спеціалізацій, з яких можна знайти фахівців, є такі, як розробка сайтів, оптимізація, мережі, інформаційні системи, анімація та багато інших.

Основний функціонал сервісу базується на такому ж принципі, що й інші аналоги: замовник публікує завдання, а виконавці подають заявки на його реалізацію. Також передбачена комісія сервісу у розмірі 7,5% від вартості проєкту (що менше, ніж у згаданій раніше платформі), і її сплачує замовник.

Головні переваги Freelance.ua – це швидка та легка реєстрація, інтуїтивно зрозумілий інтерфейс і зручна система оформлення профілю. До того ж платформа дозволяє фрилансерам обирати формат роботи, наприклад, спеціалізуватися лише на разових завданнях або довгострокових проєктах.

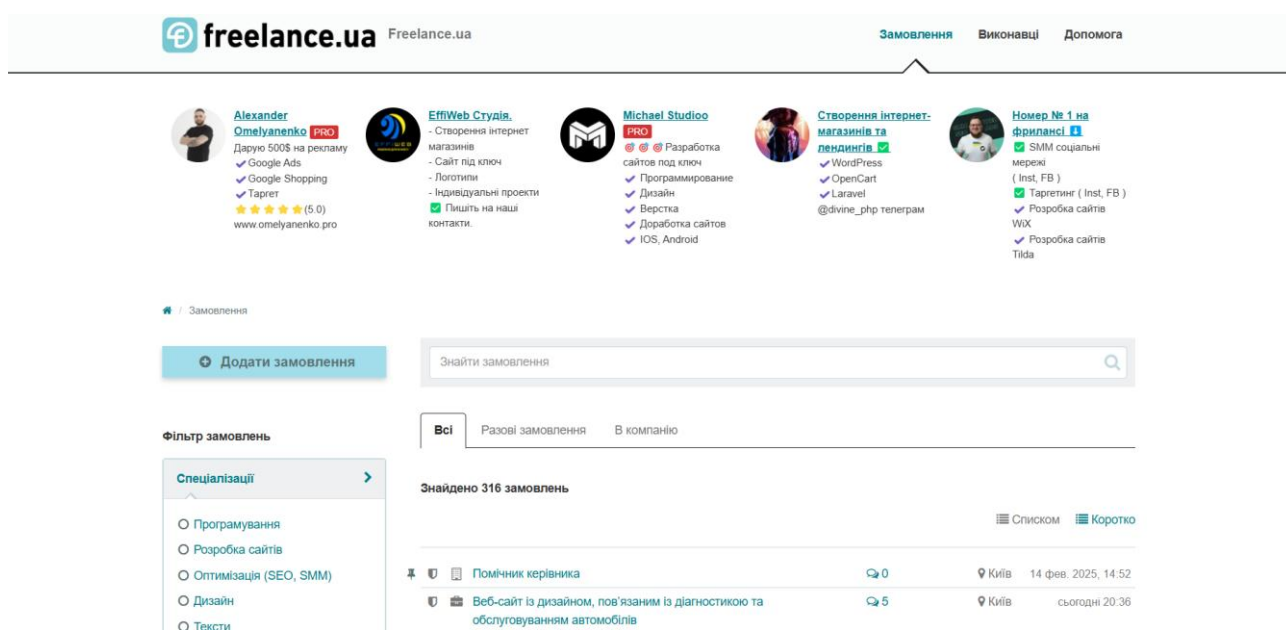


Рисунок 1.2 – Пошук замовлень на сторінці платформи Freelance.ua

Як і на більшості відомих бірж для співпраць з незалежними виконавцями, у Freelance.ua є свої недоліки. Одним із головних є висока конкуренція.

					ІПЗ.КР.Б – 121 – 25 – ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		14

Початківцям непросто отримати перші замовлення, оскільки рейтинг і відгуки мають значний вплив на вибір замовників. Через це заявки фрилансерів без досвіду та відгуків можуть залишитися непоміченими.

І так само, як на попередньому сервісі, для вирішення таких проблем є преміум підписка Pro. Вона гарантує підвищену видимість профілю та пріоритет у відгуках на замовлення. Крім того, без платного профілю неможливо подавати заявки на проєкти вартістю понад 400 грн.

Серед міжнародних платформ Upwork (рис. 1.3), заснована ще у 1999 році, по праву вважається одним із провідних вебдодатків для фрилансу [12]. Завдяки цьому Upwork налічує мільйони фахівців з різних галузей, які шукають як проєктну, так і постійну роботу. Фрилансери, використовуючи цю платформу, мають можливість знайти варіанти зайнятості, що відповідають їхньому рівню кваліфікації, досвіду, тарифам і графіку.

Звісно, порівняно з українськими аналогами, у цій системі комісія вища – 10%. Але водночас тут є можливість поспівпрацювати з такими відомими на весь світ компаніями, як Dropbox, Microsoft, Airbnb та ін. Платформа Upwork пропонує великий вибір сфер діяльності. Серед основних напрямів – розробка програмного забезпечення та ІТ, штучний інтелект, графічний дизайн і креативні послуги, маркетинг і продажі, а також інжиніринг та архітектурне проєктування.

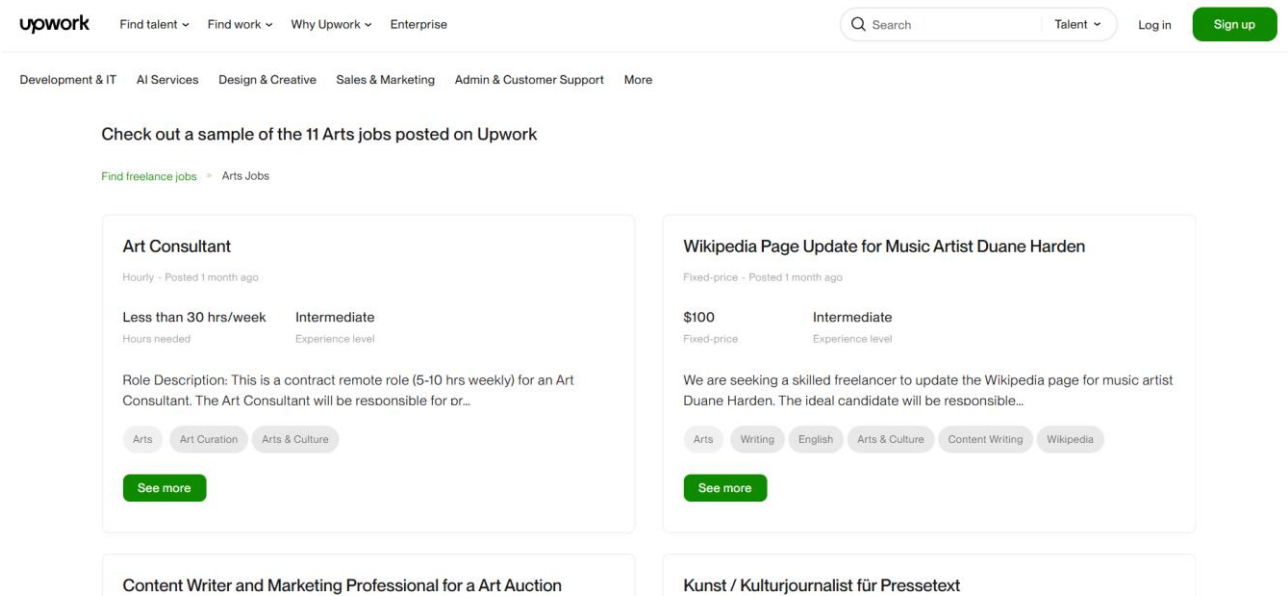


Рисунок 1.3 – Сторінка для пошуку проєктів на платформі Upwork

Втім, за цими ж перевагами ховаються й певні недоліки. Завдяки популярності та активному просуванню Upwork може створювати враження, що платформа відкриває безмежні можливості для професійного розвитку та співпраці з великими компаніями. Проте на практиці багато таких компаній використовують сервіс переважно для розміщення короткострокових чи малобюджетних проєктів, а не для масштабних довготривалих контрактів.

Отже, розробка власного програмного продукту вимагає детального аналізу існуючих рішень на ринку, щоб створити конкурентоспроможний і ефективний продукт.

Аналіз аналогів показав, що кожна з платформ має свої сильні та слабкі сторони (табл. 1.1). Водночас, жодна з них не є ідеальним рішенням для всіх категорій користувачів.

Таблиця 1.1

Порівняльна характеристика аналогів програмного продукту та власного вебдодатку

Характеристика	Freelancehunt	Freelance.ua	Upwork	Власний вебдодаток
Вибір вакансій	великий	середній	великий	великий
Комісія	9%	7,5%	10%	відсутня
Підписка	+	+	+	-
Конкуренція	висока	висока	висока	низька
Зручність	середня	середня	висока	висока

Саме тому розробка власного програмного продукту може стати не лише актуальним кроком, а й перспективним напрямом у цій сфері. Особливо якщо він здатен запропонувати користувачам певні переваги, які вигідно вирізнятимуть його серед існуючих рішень. Йдеться, зокрема, про створення більш привабливих умов для новачків, що тільки розпочинають свою діяльність і потребують підтримки на старті. Також важливо передбачити можливість

оптимізації процесу пошуку замовлень і вдосконалення механізмів оцінювання фрилансерів, що дозволить зробити співпрацю більш прозорою та ефективною. Ще одним значущим фактором є зменшення комісійних зборів, які часто стають бар'єром для залучення нових користувачів. Інтеграція сучасних технологій, спрямованих на покращення взаємодії між замовниками та виконавцями, може суттєво підвищити якість сервісу загалом і зміцнити позиції платформи на ринку.

Таким чином, створення нового вебдодатку для пошуку та співпраці з незалежними виконавцями зможе запропонувати користувачам кращий рівень зручності, якості та комфорту, враховуючи сильні сторони конкурентів і уникаючи їхніх недоліків.

1.3. Вибір архітектури вебдодатку

При розробці вебдодатку для співпраці між незалежними виконавцями та замовниками було обрано архітектуру на основі контейнеризації з використанням Docker [2]. Такий підхід забезпечує гнучкість, масштабованість і спрощене керування залежностями, що є критично важливим для ефективної роботи платформи. Завдяки контейнерам можна ізолювати різні частини системи, що сприяє стабільності та полегшує оновлення окремих компонентів без впливу на загальну працездатність сервісу.

Для реалізації клієнтської частини використовується React, оскільки цей фреймворк забезпечує швидкий рендеринг, зручну роботу з компонентами та високий рівень продуктивності. Крім того, React дозволяє створювати інтерактивний інтерфейс користувача з ефективним оновленням даних без необхідності перезавантаження сторінки [6]. Для стилізації застосовується Chakra UI, що забезпечує гнучкість у налаштуванні вигляду та адаптивність для різних пристроїв. Реалізація маршрутизації відбувається через React Router, що дозволяє створювати багатосторінкові додатки з динамічним оновленням контенту. Axios використовується для виконання HTTP-запитів, що спрощує взаємодію з бекендом через REST API. Крім того, застосування Socket.IO Client

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						17
Змн.	Арк.	№ докум.	Підпис	Дата		

дозволяє реалізувати функціонал реального часу, наприклад, миттєві сповіщення або оновлення даних без необхідності оновлення сторінки.

Серверна частина побудована на Node.js у поєднанні з Express. Це рішення було обране через його асинхронну природу та здатність обробляти велику кількість одночасних запитів, що критично важливо для динамічних вебдодатків. Express є легковаговим вебфреймворком, який забезпечує швидку обробку HTTP-запитів і дозволяє легко створювати масштабовані API. Його використання сприяє спрощенню розробки за рахунок вбудованих middleware-функцій та можливості розширення через численні модулі.

Додатково використовується Nginx як зворотний проксі-сервер для балансування навантаження та оптимізації продуктивності. Це дозволяє рівномірно розподіляти навантаження між сервісами, зменшуючи затримки при обробці запитів і підвищуючи загальну стійкість системи. Nginx також ефективно кешує статичний контент, що зменшує кількість запитів до серверів і покращує швидкість завантаження сторінок.

Backend реалізує REST API для комунікації з frontend-частиною, а також підтримує WebSocket для забезпечення реального часу. Впровадження WebSocket дозволяє реалізувати такі функції, як миттєві повідомлення, оновлення статусів проєктів і зворотний зв'язок між виконавцями та замовниками без необхідності перезавантаження сторінки.

Реалізація авторизації здійснюється за допомогою JSON Web Token (JWT), що забезпечує безпечне зберігання та передачу аутентифікаційних даних. JWT є безпечним методом аутентифікації, оскільки він дозволяє передавати дані у зашифрованому вигляді, що знижує ризик несанкціонованого доступу до облікових записів користувачів.

База даних MySQL керується за допомогою ORM-бібліотеки Sequelize, яка спрощує роботу з даними та дозволяє використовувати міграції для зручного керування схемою бази даних. Sequelize надає розширений інструментарій для побудови запитів та автоматизації управління таблицями, що полегшує процеси

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						18
Змн.	Арк.	№ докум.	Підпис	Дата		

зберігання та обробки інформації. Використання міграцій дозволяє підтримувати структуру бази даних в актуальному стані та легко розгортати зміни без порушення роботи системи.

Крім того, використання Swagger.jsdoc та Swagger-UI-Express полегшує документування API, що сприяє зручності його розширення та підтримки. Завдяки цьому розробники можуть легко переглядати та тестувати кінцеві точки API, що значно покращує процес інтеграції та взаємодії з іншими сервісами.

Одним із ключових аспектів архітектури є використання Docker, що дозволяє запускати кожен компонент системи в окремому контейнері. Наприклад, фронтенд працює у власному контейнері на основі Nginx, що забезпечує ефективну доставку статичних файлів і підвищує продуктивність. Backend також розміщується в окремому контейнері, що спрощує його розгортання, тестування та оновлення. База даних MySQL розгортається як окремий сервіс, що дозволяє ізолювати зберігання даних і спростити керування ними. Завдяки такому підходу можна легко масштабувати систему шляхом додавання нових контейнерів при зростанні навантаження.

Контейнеризація також спрощує процес розгортання та підтримки вебдодатку, оскільки всі залежності та налаштування зберігаються у вигляді конфігураційних файлів, що гарантує стабільність роботи незалежно від середовища [20]. Це особливо важливо для команди розробників, оскільки дозволяє швидко відтворювати однакове середовище на різних пристроях без потреби в ручному налаштуванні кожного компонента.

Завдяки використанню такої архітектури вебдодаток отримує високу продуктивність, гнучкість у розширенні функціоналу та можливість швидкого масштабування. Реалізація через контейнеризацію дозволяє забезпечити стабільну роботу сервісу та значно спрощує його підтримку, що є важливим аспектом для сучасних вебплатформ. Таким чином, обраний підхід до архітектури вебдодатку забезпечує всі необхідні умови для ефективного функціонування платформи та її розвитку в майбутньому.

					ІПЗ.КР.Б – 121 – 25 – ІЗ	Арк.
						19
Змн.	Арк.	№ докум.	Підпис	Дата		

1.4. Обґрунтування вибору інструментальних засобів для розробки вебдодатку

У розробці вебдодатку для фрилансерів було використано ряд ключових інструментальних засобів, кожен з яких відіграє важливу роль у забезпеченні його ефективності, безпеки та продуктивності.

Одним із центральних елементів серверної частини є Node.js у поєднанні з Express. Node.js обрано завдяки його асинхронній природі та подієвому механізму обробки запитів, що дозволяє ефективно працювати з великою кількістю одночасних з'єднань [14]. Express, як мінімалістичний вебфреймворк, спрощує створення маршрутизації і загальну організацію серверної логіки.

Для оптимізації продуктивності та балансування навантаження використовується Nginx. Цей зворотний проксі-сервер дозволяє рівномірно розподіляти запити між серверами, кешувати статичний контент і покращувати швидкодію додатку.

Щоб забезпечити безперервну взаємодію користувачів у реальному часі, реалізовано підтримку WebSocket через бібліотеку Socket.IO. Це рішення дозволяє встановлювати двосторонній зв'язок між клієнтом і сервером, що є особливо корисним для функціоналу чатів, повідомлень про статус замовлень та інших інтерактивних елементів.

Для роботи з базою даних використовується MySQL у поєднанні з ORM-бібліотекою Sequelize. Завдяки Sequelize можна зручно керувати взаємозв'язками між таблицями, створювати та виконувати міграції, що спрощує підтримку та масштабування бази даних.

Реалізація авторизації здійснюється через JSON Web Token (JWT), що забезпечує безпечне зберігання та передачу аутентифікаційних даних. Це дозволяє користувачам залишатися авторизованими без необхідності повторного введення паролів при кожному запиті до сервера.

Окрему увагу приділено документації API, яка реалізується через Swagger.jsdoc та Swagger-UI-Express. Це полегшує розробку, тестування та

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

інтеграцію API-endpoint, що є важливим аспектом для зручності розширення та підтримки системи.

Використання Docker та контейнеризації дозволяє стандартизувати середовище розгортання додатку, ізолюючи кожен компонент у власному контейнері. Це значно покращує гнучкість системи, спрощує масштабування та полегшує оновлення окремих модулів без впливу на роботу всієї платформи.

Таким чином, сукупність цих технологій забезпечує високу продуктивність, безпеку та зручність у використанні вебдодатку для фрилансерів, що робить його ефективним інструментом для співпраці замовників та виконавців.

Висновки до першого розділу

У першому розділі було проаналізовано сучасні підходи до розробки вебдодатків для взаємодії з фрилансерами. Розглянуто ключові аспекти проєктування, вимоги до функціональності таких платформ та їх вплив на ринок дистанційної зайнятості.

На основі проведеного аналізу визначено, що ефективні вебдодатки для співпраці з фрилансерами мають забезпечувати зручний інтерфейс, широкий функціонал для взаємодії між замовниками та виконавцями, а також високий рівень безпеки та довіри між учасниками. Особливу увагу слід приділяти адаптивності інтерфейсу та стабільності роботи системи.

Також розглянуто основні технологічні підходи, що використовуються для створення подібних вебплатформ, включаючи мікросервісну архітектуру, використання API та забезпечення безперебійної роботи системи. Виявлено, що важливу роль у розробці відіграє правильний вибір стеку технологій та оптимізація процесів взаємодії між клієнтом і сервером.

Таким чином, перший розділ закладає теоретичну основу та підготовчу складову для подальшого дослідження процесу створення вебдодатку, що

					ІПЗ.КР.Б – 121 – 25 – ІЗ	Арк.
						21
Змн.	Арк.	№ докум.	Підпис	Дата		

відповідатиме сучасним вимогам ринку та забезпечуватиме ефективну співпрацю між виконавцями та замовниками.

					ПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБДОДАТКУ ДЛЯ СПІВПРАЦІ З НЕЗАЛЕЖНИМИ ВИКОНАВЦЯМИ

2.1. Визначення варіантів використання та структури системи

Перед тим як переходити до безпосереднього проєктування системи, необхідно визначити її призначення, функціональні можливості та загальну структуру. Це дозволить сформувати цілісне бачення очікуваного результату як для подальшої розробки, так і для кінцевих користувачів. Визначення варіантів використання та структури вебсистеми є важливим наступним етапом після попереднього дослідження та аналізу вебдодатків для взаємодії з незалежними виконавцями, представленого в першому розділі.

Необхідно трансформувати виявлені бізнес-потреби в конкретні функціональні та нефункціональні вимоги до вебдодатку. Це надасть змогу не лише структурувати функціонал, але й закласти основу для технічного проєктування та реалізації ключових сценаріїв взаємодії між користувачами.

Зокрема, важливо ідентифікувати найголовніші бізнес-цілі платформи: забезпечення зручного вебдодатку для пошуку, комунікації, укладення угод та управління співпрацею між замовниками та незалежними виконавцями. Система має підтримувати інструменти для реєстрації користувачів, створення профілів, розміщення та пошуку проєктів, відправки пропозицій, а також збирання відгуків і рейтингів.

Крім того, для повноцінного уявлення про функціонування системи, необхідно розробити діаграми варіантів використання (use case diagrams), які демонструють взаємодію між користувачами та системою у контексті окремих сценаріїв. Це сприятиме формуванню цілісної архітектурної картини та дасть змогу перейти до побудови логічної моделі системи.

Умовно систему можна поділити на три логічні блоки відповідно до типу користувача: «Виконавець», «Замовник» та «Адміністратор». Кожен блок реалізує окрему підсистему прав доступу, що гарантує безпечну і

					ІПЗ.КР.Б – 121 – 25 – ІЗ	Арк.
						23
Змн.	Арк.	№ докум.	Підпис	Дата		

цілеспрямовану взаємодію користувача з функціоналом. Для «Виконавця» акцент зроблено на пошуку завдань, подачі заявок і веденні комунікації з замовником. «Замовник», зі свого боку, зосереджений на формуванні завдань, керуванні заявками та оцінюванні результатів роботи. «Адміністратор» відповідає за підтримання цілісності даних, модерацію контенту, налаштування параметрів системи та опрацювання звернень.

Кожен тип користувача виконує унікальні функції, які зумовлюють набір доступних йому варіантів використання. Саме тому необхідною є побудова окремих use case-діаграм, які візуально демонструють відносини між акторами та їх сценаріями взаємодії із системою.

Unified Modeling Language (UML) – це стандартизована мова візуального моделювання, що використовується в галузі програмної інженерії для опису, проєктування та документування програмних систем. Вона надає засоби для створення різноманітних типів діаграм, що відображають як статичну структуру системи, так і її динамічну поведінку.

Діаграма варіантів використання (use case diagram) належить до поведінкових діаграм UML і служить зручним інструментом для моделювання зовнішньої точки зору на систему – тобто того, як користувачі взаємодіють із нею. Ці діаграми зазвичай містять акторів (користувачів або інші системи), варіанти використання (use cases) та зв'язки між ними. Таким чином, визначення варіантів використання та побудова відповідних діаграм є не лише інструментом опису поточних функцій, а й стратегічним кроком у напрямку забезпечення довготривалої підтримуваності та технологічної гнучкості вебсистеми.

Вимоги користувачів:

Користувач з правами «Виконавець» має наступні можливості (рис. 2.1):

- додавання інформації та редагування власного профілю;
- змінити пароль;
- завантажити резюме;
- пошук та фільтрація існуючих замовлень;

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						24
Змн.	Арк.	№ докум.	Підпис	Дата		

- відгукнутися на замовлення;
- перегляд профілю замовника;
- обговорення деталей в чаті із замовником;
- відмовитися від замовлення;
- залишити відгук про співпрацю з роботодавцем.

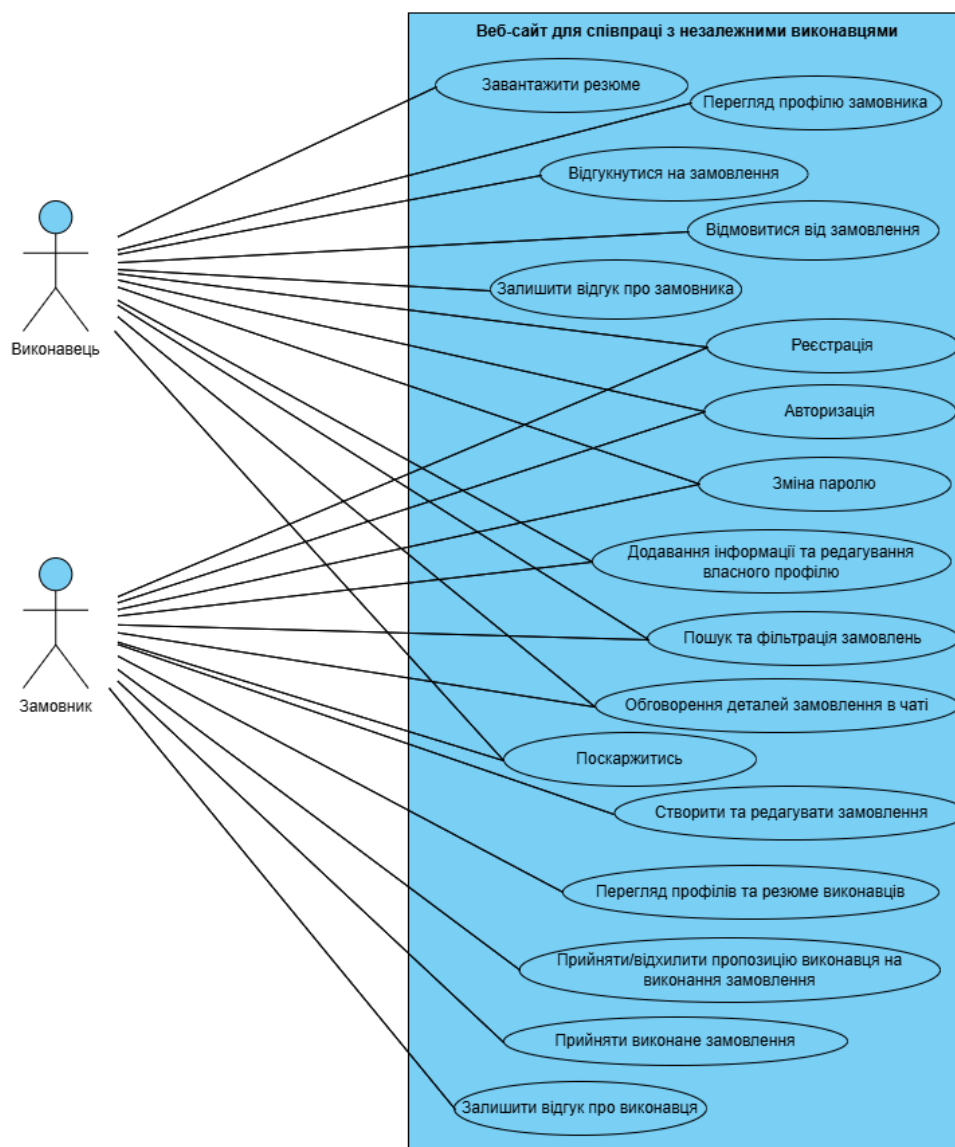


Рисунок 2.1 – Діаграма варіантів використання для виконавця та замовника

Користувач з правами «Замовник» має наступні можливості (див. рис. 2.1):

- додавання інформації та редагування власного профілю;
- змінити пароль;
- розміщення замовлення;
- перегляд профілів виконавців;

Змн.	Арк.	№ докум.	Підпис	Дата

- перегляд резюме виконавців;
- прийняття/відхилення запиту на виконання замовлення;
- обговорення деталей в чаті із виконавцем;
- прийняти виконане замовлення;
- оцінити роботу виконавця.

Користувач з правами «Адміністратор» має наступні можливості (рис. 2.2):

- пошук та фільтрація замовлень;
- перегляд списку користувачів та видалення їх;
- додавання/редагування/видалення регіонів;
- додавання/редагування/видалення навичок;
- перегляд списку замовлень та видалення їх;
- перегляд скарг від користувачів.

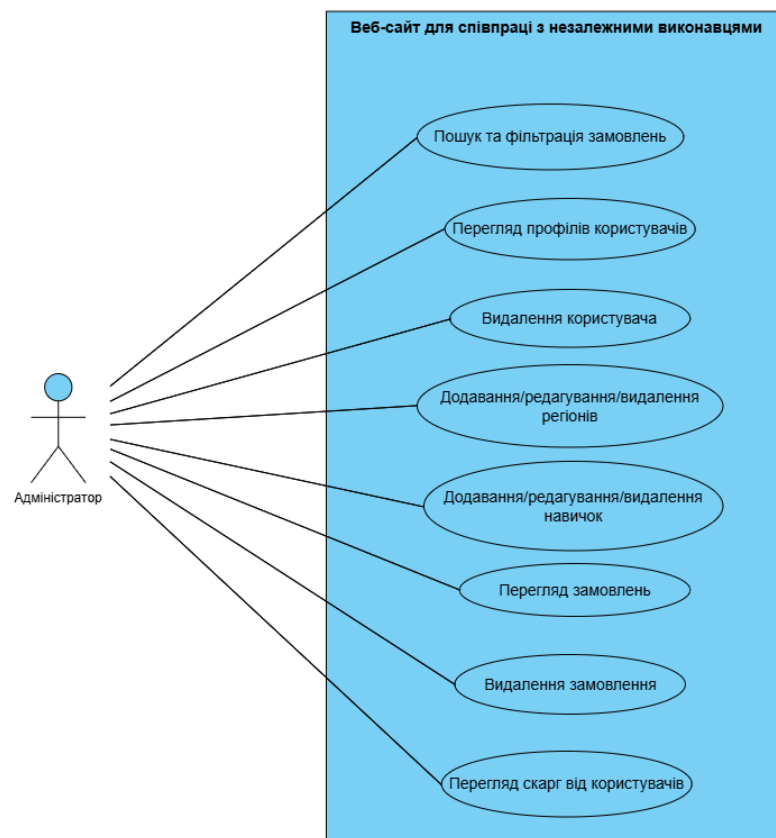


Рисунок 2.2 – Діаграма варіантів використання для адміністратора системи

Такий комплексний огляд функціональних і структурних характеристик вебдодатку, призначеного для організації взаємодії між фрілансерами та замовниками, та побудова діаграм варіантів використання (use case diagrams)

					ІПЗ.КР.Б – 121 – 25 – ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		26

дали змогу чітко окреслити основні сценарії взаємодії користувачів із системою, що, у свою чергу, дозволило структурувати функціональні можливості платформи відповідно до ролей. Завдяки використанню цих діаграм стало можливим візуально представити ключові процеси, що суттєво полегшує розуміння логіки роботи системи та взаємозв'язків між її компонентами.

2.2. Розробка бази даних вебдодатку

Одним із ключових принципів ефективного проєктування бази даних є забезпечення логічних взаємозв'язків між усіма інформаційними сутностями. Саме завдяки цій взаємозалежності база даних набуває цілісності й системності, що відрізняє її від простого набору розрізнених таблиць. Інформація у реляційній моделі повинна бути чітко структурована таким чином, щоб підтримувати узгодженість та можливість ефективного доступу до пов'язаних даних.

Не менш важливою концепцією є нормалізація – процес оптимізації структури таблиць із метою усунення логічних суперечностей, запобігання аномаліям і забезпечення цілісності даних. Серед типових проблем, які вирішує нормалізація, можна назвати дублювання даних, неконсистентність та труднощі з оновленням або видаленням записів [17]. Суть нормалізації полягає в розділенні великих таблиць на кілька логічно пов'язаних, що дозволяє уникнути надмірності інформації та забезпечити правильне функціонування бази в умовах реальної експлуатації.

У процесі підготовки до розробки системи було прийнято рішення використовувати реляційну модель бази даних, реалізовану за допомогою системи управління базами даних MySQL [9]. Такий вибір зумовлений високою надійністю цієї СУБД, її активною підтримкою спільноти, зручністю в обробці транзакцій, а також широкими можливостями інтеграції з ORM-інструментами, зокрема з бібліотекою Sequelize, яка застосовується для реалізації серверної логіки вебдодатку.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						27
Змн.	Арк.	№ докум.	Підпис	Дата		

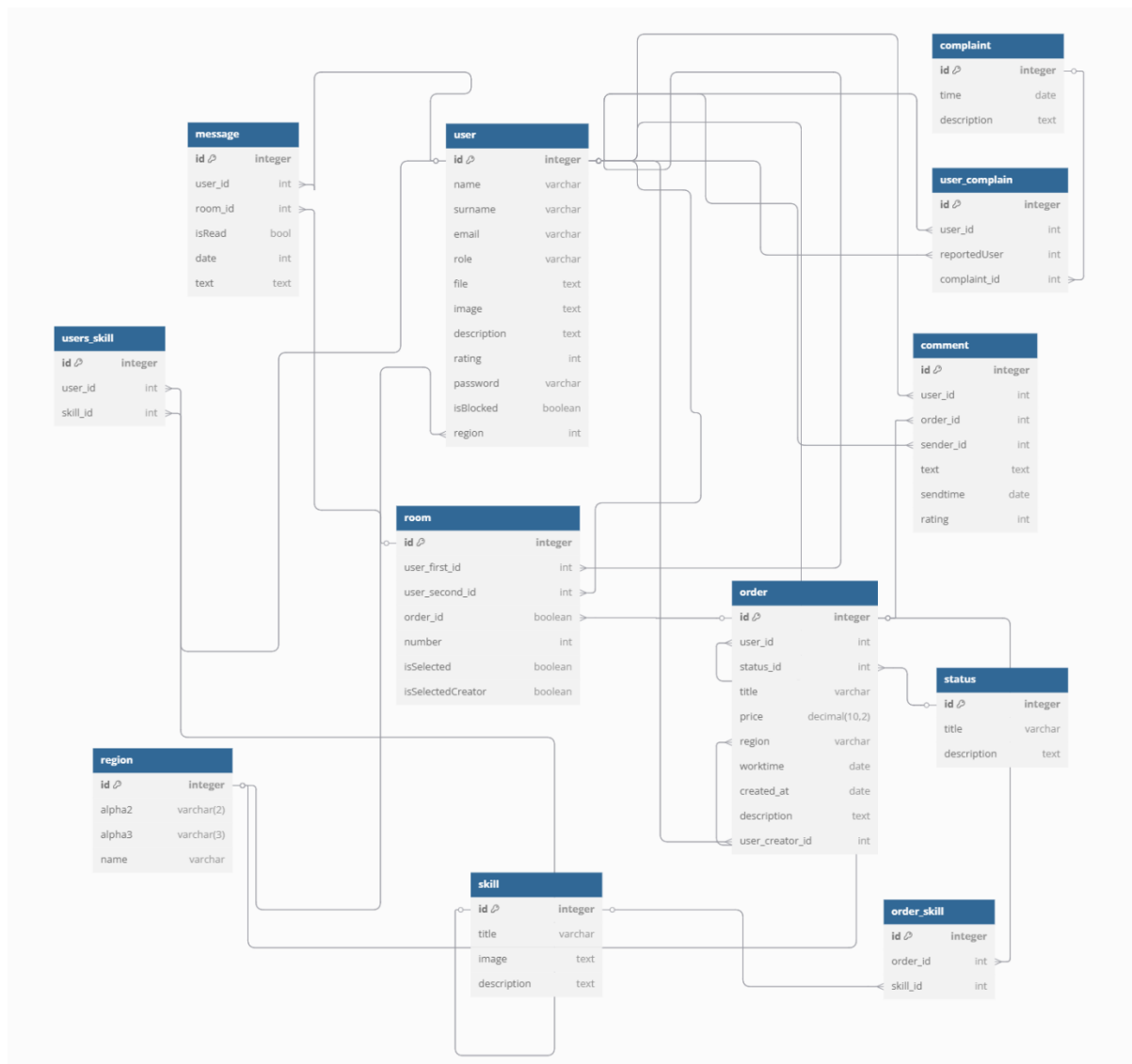


Рисунок 2.3 – Діаграма бази даних вебдодатку

Таблиця User (табл. 2.1) містить основну інформацію про всіх зареєстрованих користувачів: ім'я, прізвище, електронну адресу, роль (виконавець, замовник або адміністратор), опис профілю, рейтинг, а також шляхи до файлів і зображень. Крім того, таблиця включає поле регіону, пароль і прапорець блокування користувача.

Таблиця 2.1

Таблиця User

Поле	Тип даних	ПК	ЗК	Опис
1	2	3	4	5
id	int	+	-	id користувача

name	varchar(255)	-	-	ім'я
surname	varchar(255)	-	-	прізвище
1	2	3	4	5
email	varchar(255)	-	-	електронна адреса
role	varchar(255)	-	-	роль
file	text	-	-	шлях до файлу
image	text	-	-	шлях до зображення
description	text	-	-	опис профілю
rating	int	-	-	рейтинг
password	varchar(255)	-	-	пароль
isBlocked	boolean	-	-	блокування користувача
region	int	-	+	регіон

Таблиця Room (табл. 2.2) використовується для зберігання інформації про активні чати між користувачами в контексті конкретного замовлення. Вона містить посилання на обох співрозмовників, замовлення, у межах якого ведеться переписка, а також статус підтвердження запиту на надсилання повідомлень.

Таблиця 2.2

Таблиця Room

Поле	Тип даних	ПК	ЗК	Опис
id	int	+	-	id чату
user_first_id	int	-	+	id першого співрозмовника
user_second_id	int	-	+	id другого співрозмовника
order_id	int	-	+	id замовлення
number	int	-	-	номер
isSelected	boolean	-	-	підтвердження
isSelectedCreator	boolean	-	-	підтвердження замовником

Таблиця Message (табл. 2.3) фіксує всі повідомлення, що передаються в системі. До її структури входять ідентифікатори користувачів й кімнати (чату), текст повідомлення, дата та ознака прочитаності.

Таблиця 2.3

Таблиця Message

Поле	Тип даних	ПК	ЗК	Опис
id	int	+	-	id повідомлення
user_id	int	-	+	id користувача
room_id	int	-	+	id чату
isRead	boolean	-	-	чи прочитане повідомлення
date	datetime	-	-	дата відправки повідомлення
text	text	-	-	текст повідомлення

Таблиця Users_skill (табл. 2.4) є проміжною і реалізує зв'язок «багато-до-багатьох» між користувачами та навичками, якими вони володіють. Це дозволяє вказувати кілька навичок для кожного користувача.

Таблиця 2.4

Таблиця Users_skill

Поле	Тип даних	ПК	ЗК	Опис
id	int	+	-	id запису прив'язки користувача та вміння
user_id	int	-	+	id користувача
skill_id	int	-	+	id вміння

Таблиця Order_skill (табл. 2.5) виконує аналогічну функцію для замовлень, дозволяючи прив'язати до кожного завдання відповідні навички, необхідні для його виконання.

Таблиця 2.5

Таблиця Order_skill

Поле	Тип даних	ПК	ЗК	Опис
------	-----------	----	----	------

Продовження таблиці 2.5

id	int	+	-	id запису прив'язки замовлення та вміння
order_id	int	-	+	id замовлення
skill_id	int	-	+	id вміння

Таблиця Order (табл. 2.6) є центральною в моделі й зберігає інформацію про опубліковані замовлення: заголовок, опис, ціну, регіон, терміни виконання, дату створення, статус виконання, а також ідентифікатори виконавця та замовника.

Таблиця 2.6

Таблиця Order

Поле	Тип даних	ПК	ЗК	Опис
1	2	3	4	5
id	int	+	-	id замовлення
user_id	int	-	+	id користувача
status_id	int	-	+	id статусу
title	varchar(255)	-	-	заголовок
price	decimal(10,2)	-	-	ціна
region_id	int	-	+	id регіону
worktime	date	-	-	час на виконання
1	2	3	4	5
created_at	date	-	-	дата створення замовлення
description	text	-	-	опис
user_creator_id	int	-	+	id користувача, що створив

Таблиця Status (табл. 2.7) виконує роль довідника і містить назви можливих статусів замовлень та їхній текстовий опис.

Таблиця 2.7

Таблиця Status

Поле	Тип даних	ПК	ЗК	Опис
------	-----------	----	----	------

id	int	+	-	id статусу
title	varchar(255)	-	-	назва
description	text	-	-	опис

Таблиця Comment (табл. 2.8) використовується для зберігання відгуків між користувачами після завершення співпраці. Вона включає текст коментаря, рейтинг, дату, ідентифікатори відправника, отримувача та замовлення, до якого належить відгук.

Таблиця 2.8

Таблиця Comment

Поле	Тип даних	ПК	ЗК	Опис
id	int	+	-	id відгуку
user_id	int	-	+	id користувача
order_id	int	-	+	id замовлення
sender_id	int	-	+	id користувача
text	text	-	-	текст відгуку
sendtime	date	-	-	дата відправки
rating	int	-	-	рейтинг

Таблиця Skill (табл. 2.9) є довідником навичок. Вона містить назву, опис, шлях до зображення вміння.

Таблиця 2.9

Таблиця Skill

Поле	Тип даних	ПК	ЗК	Опис
id	int	+	-	id вміння
title	varchar(255)	-	-	назва
image	text	-	-	шлях до зображення
description	text	-	-	опис

Таблиця Complaint (табл. 2.10) реєструє скарги користувачів, зберігаючи опис проблеми та дату її подання.

Таблиця 2.10

Таблиця Complaint

Поле	Тип даних	ПК	ЗК	Опис
id	int	+	-	id скарги
time	datetime	-	-	дата та час відправки скарги
description	text	-	-	опис

Таблиця Region (табл. 2.11) містить перелік регіонів, які можуть використовуватись у профілях користувачів або замовленнях як додатковий критерій пошуку.

Таблиця 2.11

Таблиця Region

Поле	Тип даних	ПК	ЗК	Опис
id	int	+	-	id регіону
name	varchar(255)	-	-	регіон

Таблиця User_complain (табл. 2.12) є проміжною таблицею для зв'язку користувачів із відповідними скаргами. Вона дозволяє зафіксувати, хто подав скаргу, проти кого вона була спрямована, і з якою скаргою вона пов'язана.

Таблиця 2.12

Таблиця User_complain

Поле	Тип даних	ПК	ЗК	Опис
id	int	+	-	id прив'язки користувачів та скарги
user_id	int	-	+	id користувача
reportedUser	int	-	+	id користувача, що залишив скаргу
complaint_id	int	-	+	id скарги

Загалом, така структура бази даних забезпечує повну підтримку всіх функціональних вимог системи, дозволяє гнучко працювати з контентом, фільтрувати дані, підтримувати цілісність зв'язків і гарантувати стабільну

роботу вебдодатку в умовах реального навантаження.

2.3. Реалізація функціоналу роботи вебдодатку для співпраці між незалежними виконавцями

У рамках функціональності вебдодатку для співпраці між незалежними виконавцями важливою складовою є реалізація можливості роботи з файлами. Це стосується, зокрема, завантаження резюме, аватарів, зображень портфоліо та інших матеріалів, які користувачі можуть прикріплювати до своїх профілів або замовлень. Для зберігання таких файлів використовується локальне об'єктне сховище MinIO, яке є сумісним з API Amazon S3. Таке рішення забезпечує гнучкість, безпеку доступу та масштабованість обробки файлів у межах інфраструктури вебдодатку.

Ключовою функцією, що реалізує завантаження файлів до сховища, є метод `uploadFile()`. Він працює з об'єктом моделі (наприклад, профілем користувача), до якого прив'язується файл. Функція формує унікальну назву файлу, використовуючи префікс, ідентифікатор моделі та часову мітку, після чого, за потреби, видаляє попередній файл і зберігає новий у бакеті. Код реалізації виглядає наступним чином:

```
async uploadFile({
  modelInstance,
  file,
  fileNamePrefix,
  fileField,
  forcedExtension = null,
  bucketName = 'files'
}) {
  try {
    if (!modelInstance) {
      throw new Error('Сутність не знайдено');
    }
    const fileExtension = forcedExtension ||
path.extname(file.originalname);
    const fileName =
`${fileNamePrefix}_${modelInstance.id}_${Date.now()}${fileExtension}`;
    await this.deleteExistingFile(modelInstance[fileField]);
    const params = {
      Bucket: bucketName,
      Key: fileName,
      Body: file.buffer
    };
```

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						34
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    };
    await this.s3.putObject(params).promise();
    await modelInstance.update({ [fileField]: fileName });
    return fileName;
  } catch (error) {
    console.error('Помилка завантаження файлу:', error);
    throw error;
  }
}

```

Ще одним важливим компонентом є метод `getFileName()`, який дозволяє згенерувати тимчасове підписане посилання на файл. Це посилання дійсне протягом заданого часу, після чого воно стає недоступним. Такий підхід дозволяє контролювати доступ до файлів без необхідності надання відкритих прямих посилань:

```

async getFileName(fileName, bucketName = 'files', expirationInSeconds =
3600) {
  try {
    if (!fileName) {
      throw new Error('Назву файлу не вказано');
    }
    const params = {
      Bucket: bucketName,
      Key: fileName,
      Expires: expirationInSeconds
    };
    const signedUrl = await this.s3.getSignedUrlPromise('getObject',
params);
    return signedUrl.replace('minio:9000', 'localhost:9000');
  } catch (error) {
    console.error('Помилка отримання URL файлу:', error);
    throw error;
  }
}

```

Для безпосереднього отримання вмісту файлу зі сховища (наприклад, для обробки або передачі через HTTP) реалізовано метод `getFile()`. Він повертає вміст файлу у вигляді буфера:

```

async getFile(fileName, bucketName = 'files') {
  try {
    if (!fileName) {
      throw new Error('Назву файлу не вказано');
    }
    const params = {
      Bucket: bucketName,
      Key: fileName
    };
    const data = await this.s3.getObject(params).promise();
    return data.Body;
  } catch (error) {

```

```

        throw error;
    }
}

```

З метою уникнення надлишковості та підтримання актуальності контенту використовується функція `deleteExistingFile()`, яка видаляє попередній файл із бакету, якщо такий існує:

```

async deleteExistingFile(fileName, bucketName = 'files') {
    if (!fileName) return;
    const params = {
        Bucket: bucketName,
        Key: fileName
    };
    try {
        await this.s3.deleteObject(params).promise();
    } catch (error) {
    }
}

```

Наступною було реалізовано можливість створення кімнати для спілкування між замовником і виконавцем у процесі взаємодії щодо конкретного замовлення. Цей функціонал забезпечує основу для обміну повідомленнями в режимі реального часу за допомогою технології WebSocket (на базі бібліотеки Socket.IO). Створення кімнати ініціюється з клієнтської частини, після чого обробка події відбувається на сервері.

Ключовий елемент цього процесу – обробник події `'create_room'`, який реєструється на сервері при ініціалізації з'єднання. Його завдання – створити нову кімнату в базі даних, відправити перше повідомлення та повідомити другого учасника про появу нового чату. Код обробника виглядає наступним чином:

```

socket.on('create_room', async ({ order_id, message }, callback) => {
    try {
        const response = await axios.post('/rooms', { order_id, message }, axiosConfig);
        const recipientSocketId =
connectedUsers.get(response.data.user_second_id);
        if (recipientSocketId) {
            io.to(recipientSocketId).emit('new_room', response.data);
        }

        callback({ success: true });
    } catch (error) {
        callback({ success: false, message: 'Помилка при створенні чату'
});
    }
}

```

```
});
```

Також важливим сценарієм у процесі обміну повідомленнями є приєднання користувача до конкретної кімнати (чату), що дає змогу відображати лише релевантну історію переписки та забезпечити коректне надсилання і приймання повідомлень у межах відповідного замовлення. Цей процес реалізується за допомогою WebSocket-технології через обробку події `join_room`.

Метод, що відповідає за приєднання до кімнати, обробляє подію `'join_room'`, яка ініціюється клієнтською стороною із зазначенням ідентифікатора кімнати:

```
socket.on('join_room', async (roomId) => {
  try {
    const room = await axios.get(`/rooms/number/${roomId}`,
  axiosConfig);
    const rooms = Array.from(socket.rooms);
    rooms.forEach((room) => {
      if (room !== socket.id) {
        socket.leave(room);
      }
    });
    socket.join(roomId);
    await axios.patch(`/messages/room/${room.data.id}`, {},
  axiosConfig);
    const messages = await
  axios.get(`/messages/notification/${room.data.id}`, axiosConfig);
    socket.emit('notification', messages.data);
  } catch (error) {
    console.log(error);
  }
});
```

На першому етапі обробки сервер надсилає GET-запит до API `/rooms/number/:roomId`, щоб отримати повну інформацію про відповідну кімнату за її номером. Далі відбувається оновлення списку підключених кімнат для поточного сокета: з метою уникнення одночасного перебування користувача в кількох кімнатах, сокет закриває всі попередні кімнати, крім тієї, яка є його поточним ідентифікатором.

Після очищення підключень користувач приєднується до заданої кімнати через метод `socket.join(roomId)`. Це забезпечує маршрутизацію повідомлень виключно в межах потрібного чату. Одразу після цього система надсилає

					ПІЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						37
Змн.	Арк.	№ докум.	Підпис	Дата		

PATCH-запит до маршруту /messages/room/:id, що оновлює статус усіх повідомлень у кімнаті, позначаючи їх як прочитані або активовані для перегляду.

На завершальному етапі сервер виконує запит до /messages/notification/:id, який повертає список повідомлень, що підлягають відображенню користувачеві у вигляді сповіщення. Отримані дані передаються клієнту за допомогою події 'notification', дозволяючи відобразити актуальні повідомлення без необхідності перезавантаження інтерфейсу.

Функція обміну повідомленнями забезпечує безперервну комунікацію між замовником і виконавцем у межах конкретного замовлення. Серверна частина обробляє подію 'send_message', що ініціюється користувачем при відправленні нового повідомлення:

```
socket.on('send_message', async (data) => {
  try {
    if (!data.roomId || !data.message || !data.userId) {
      throw new Error('Відсутні обов'язкові поля');
    }
    const roomResponse = await
axios.get(`/rooms/number/${data.roomId}`, axiosConfig);
    const roomData = roomResponse.data;

    const messageData = {
      user_id: Number(data.userId),
      room_id: roomData.id,
      text: data.message,
      date: new Date().toISOString(),
    };

    const messagesResponse = await axios.post('/messages',
messageData, axiosConfig);
    io.to(data.roomId).emit('receive_message',
messagesResponse.data);

    await axios.patch(`/messages/room/${roomData.id}`, {},
axiosConfig);
    const messages = await
axios.get(`/messages/notification/${roomData.id}`, axiosConfig);

    const recipientSocketId =
connectedUsers.get(messages.data.userId);
    if (recipientSocketId) {
      io.to(recipientSocketId).emit('notification', messages.data);
    }
  } catch (error) {
    console.log(error);
  }
});
```

});

У першому кроці відбувається валідація вхідних даних. Якщо хоча б одне з обов'язкових полів (ідентифікатор кімнати, повідомлення або користувача) відсутнє, генерується помилка. Це забезпечує захист від некоректних запитів ще до початку обробки.

Далі за допомогою GET-запиту до маршруту `/rooms/number/:roomId` сервер отримує інформацію про кімнату, у межах якої відбувається спілкування. На основі отриманих даних формується об'єкт повідомлення, який включає ідентифікатор користувача, ідентифікатор кімнати, текст повідомлення та поточну дату і час.

Сформоване повідомлення зберігається в базі даних через POST-запит до `/messages`. Після цього нове повідомлення надсилається всім учасникам відповідної кімнати за допомогою виклику `io.to(roomId).emit('receive_message', data)`, що забезпечує його миттєве відображення у чаті на клієнтській стороні.

Наступним кроком сервер оновлює статус повідомлень у цій кімнаті за допомогою PATCH-запиту. Це дозволяє маркувати їх як прочитані або видимі. Потім виконується додатковий запит до маршруту `/messages/notification/:id`, який повертає актуальний список повідомлень для відображення у вигляді сповіщень іншому користувачу.

Завдяки такій реалізації досягається повна інтеграція повідомлень у реальному часі, що забезпечує оперативну комунікацію між фрілансером та замовником, високу швидкодію інтерфейсу й синхронізацію даних у базі.

Висновки до другого розділу

У другому розділі було здійснено всебічне проектування вебсистеми, призначеної для організації взаємодії між незалежними виконавцями та замовниками. Початковим кроком стало визначення сценаріїв використання системи та її логічної структури. Це дозволило чітко класифікувати функціональні можливості згідно з типами користувачів – виконавців, замовників і адміністраторів.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						39
Змн.	Арк.	№ докум.	Підпис	Дата		

Розробка діаграм варіантів використання у рамках UML-моделювання забезпечила візуалізацію всіх ключових сценаріїв взаємодії, що дозволило сформулювати цілісне бачення функціоналу кожної ролі. Завдяки такому підходу було створено основу для подальшої реалізації системи з чітко структурованими зв'язками між об'єктами та користувачами, а також визначено перелік операцій, які система має підтримувати для повноцінного функціонування.

Наступним етапом стало проектування бази даних, що охоплює всі інформаційні сутності, необхідні для підтримки сценаріїв співпраці. Реляційна модель, реалізована на базі MySQL, дозволила врахувати всі логічні зв'язки між таблицями, забезпечити узгодженість даних і високий рівень масштабованості. Проміжні таблиці, що реалізують зв'язки типу «багато-до-багатьох», забезпечують гнучкість при формуванні зв'язків між користувачами, навичками, замовленнями та скаргами.

Реалізація прикладної логіки взаємодії користувача із системою засвідчила ефективне використання сучасних технологій: REST API, WebSocket, а також асинхронної обробки запитів. Зокрема, реалізовано інфраструктуру роботи з файлами, яка дозволяє безпечно завантажувати, зберігати та надавати доступ до персоналізованого контенту користувачів через локальне S3-сумісне сховище. Було також впроваджено логіку обробки подій реального часу – створення чатів, приєднання до них та обмін повідомленнями – що дозволяє досягнути високої інтерактивності платформи та зручності спілкування між сторонами.

Особливу увагу приділено захисту даних, ідентифікації користувачів та контролю їхніх дій у системі. Сценарії взаємодії детально продумані таким чином, щоб забезпечити збереження логіки бізнес-процесів та узгодженість між окремими модулями системи. Усе це заклало міцну технічну основу для подальшої розробки, тестування та впровадження вебдодатку, що відповідає актуальним вимогам галузі дистанційної праці та ринку цифрових сервісів у сфері фрилансу.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБДОДАТКОМ ДЛЯ ПОШУКУ ТА СПІВПРАЦІ З НЕЗАЛЕЖНИМИ ВИКОНАВЦЯМИ

3.1. Порядок налаштування та встановлення системи

У сучасному світі веброзробки контейнеризація стала невідемним елементом для досягнення гнучкості, масштабованості та ефективного управління залежностями. Завдяки використанню Docker, розробники можуть створювати ізольовані середовища для різних компонентів своїх застосунків, що значно спрощує процес їх розгортання та підтримки [7]. Контейнеризоване середовище значно покращує управління сервісами, робить оновлення простішими та забезпечує масштабованість. Це особливо важливо для вебплатформ, які орієнтовані на співпрацю незалежних виконавців.

Вебсередовище побудоване на складній архітектурі, де кожен елемент виконує свою роль. Вебдодаток складається з кількох логічних модулів, кожен з яких працює самостійно та виконує свої конкретні завдання. Фронтенд, створений на React.js, відповідає за формування сучасного користувацького інтерфейсу та інтерактивну взаємодію з сервером. Для управління залежностями та створення оптимізованого об'єднання системи, що потім передається кінцевому користувачу, використовується Node.js.

Серверна частина, побудована на Node.js з використанням Express, зосереджена на обробці API-запитів, взаємодії з базою даних через ORM (Sequelize) та реалізації важливих функцій безпеки за допомогою бібліотек, таких як jsonwebtoken та Argon2. Крім того, реалізація модуля WebSocket за допомогою Socket.io забезпечує постійне з'єднання та обмін даними в реальному часі, що є критично важливим для даного вебдодатку, а саме чату між замовником та виконавцем.

Підготовка до створення розгорнутого середовища починається з установки Docker і Docker Compose. Docker дозволяє створювати ізольовані контейнери з чітко визначеними конфігураціями, що забезпечує незалежність від

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						41
Змн.	Арк.	№ докум.	Підпис	Дата		

середовища виконання. Docker Compose, в свою чергу, автоматизує запуск, налаштування та взаємодію між кількома контейнерами, що значно спрощує управління застосунком.

Наступний крок – це налаштування Backend API. Node.js використовується, щоб реалізувати основну серверну. У процесі також задіяні безліч бібліотек та інструментів, таких як Argon2 для безпечного хешування паролів, CORS для підтримки крос-доменної взаємодії, dotenv для зберігання конфігурацій, Express для організації структури RESTful API, JWT для авторизації, а також Sequelize разом із MySQL для роботи з базою даних. Усі залежності та конфігурації прописані в Dockerfile модуля api/, що забезпечує правильну збірку контейнера.

Окрему функціональність у системі виконує WebSocket-сервер, що реалізований на Node.js з використанням Socket.io. Він забезпечує двосторонню комунікацію між клієнтом і сервером у реальному часі. Конфігурація WebSocket-модуля передбачає використання Express як базового сервера, бібліотеки CORS, dotenv для зберігання змінних середовища, а також Sequelize і MySQL для зберігання даних про сесії або повідомлення. Всі необхідні залежності описані у Dockerfile папки websocket/.

Сервер Nginx відіграє важливу роль у загальній архітектурі, адже він функціонує як зворотний проксі. Він перенаправляє запити до відповідних сервісів, таких як Backend API, WebSocket-сервер і Frontend, а також допомагає балансувати навантаження та кешувати ресурси. Конфігурація Nginx задається в окремому файлі, який додається до контейнера через Dockerfile.

Заключним етапом є формування конфігураційного файлу docker-compose.yml, який виконує роль інтегратора всіх сервісів у одну єдину систему [1]. Файл docker-compose.yml ретельно описує, як саме контейнеризовані сервіси взаємодіють один з одним у рамках спільної інфраструктури. Він містить визначення версії синтаксису Docker Compose, що використовується для забезпечення сумісності, а також описує різноманітні контейнери, такі як API,

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						42
Змн.	Арк.	№ докум.	Підпис	Дата		

WebSocket, Frontend та Nginx, кожен з яких виконує свою специфічну функцію у системі. У цьому файлі вказано місце розташування Dockerfile для кожного сервісу, що дозволяє правильно збудувати образи, а також налаштовано відкриття портів для забезпечення зовнішнього доступу до контейнерів:

```
services:
  frontend:
    build:
      context: ./frontend
      dockerfile: ../docker/frontend/Dockerfile
    command: ["sh", "-c", "npm install && npm run dev -- --host 0.0.0.0"]
    volumes:
      - ./frontend:/app
    networks:
      - network

  api:
    build:
      context: ./api
      dockerfile: ../docker/api/Dockerfile
    command: [ "sh", "-c", "npm install && npm run dev" ]
    volumes:
      - ./api:/app
    networks:
      - network

  websocket:
    build:
      context: ./websocket
      dockerfile: ../docker/websocket/Dockerfile
    command: ["sh", "-c", "npm install && npm run dev"]
    volumes:
      - ./websocket:/app
    networks:
      - network

  nginx:
    image: nginx:1.17
    volumes:
      - ./configs/nginx:/etc/nginx/conf.d
    ports:
      - ${NGINX_PORT}:80
    networks:
      - network

  mysql:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_DATABASE: ${MYSQL_DATABASE}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
```

					ПІЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						43
Змн.	Арк.	№ докум.	Підпис	Дата		

```

ports:
  - "3306:3306"
volumes:
  - mysql_data:/var/lib/mysql
networks:
  - network

minio:
  image: minio/minio:latest
  container_name: minio
  ports:
    - "9000:9000"
    - "9001:9001"
  environment:
    MINIO_ACCESS_KEY: "rootroot"
    MINIO_SECRET_KEY: "rootroot"
  command: server /data --console-address ":9001"
  volumes:
    - minio_data:/data
  networks:
    - network

networks:
  network:

volumes:
  mysql_data:
  minio_data:
  driver: local

```

Використання контейнеризації під час розгортання вебзастосунку для пошуку та співпраці з незалежними виконавцями не лише допомогло структурувати систему, але й забезпечило стабільну роботу в реальному середовищі. Кожен компонент – від інтерфейсу користувача, створеного на React.js, до окремого WebSocket-сервера, що відповідає за обмін повідомленнями між замовником і виконавцем – розгортається в ізольованому контейнері. Це спрощує оновлення модулів, дозволяє локалізувати помилки та виключає випадкові конфлікти між залежностями. Така архітектура дозволяє швидко додавати нові функції в майбутньому, наприклад, розширити чат, впровадити систему рейтингів або автоматизовану перевірку профілів. Усі ці можливості є критично важливими для підтримки довготривалого інтересу користувачів, особливо в умовах зростаючої конкуренції серед фриланс-бірж.

					ІПЗ.КР.Б – 121 – 25 – ІЗ	Арк.
						44
Змн.	Арк.	№ докум.	Підпис	Дата		

3.2. Структура інтерфейсу та порядок взаємодії з вебдодатком для пошуку та співпраці з незалежними виконавцями

Під час розробки інтерфейсу користувача для вебдодатку співпраці з незалежними виконавцями було важливо враховувати унікальні потреби та очікування користувачів у цій сфері. Було спроектовано та створено інтерфейс, який сприяє продуктивній взаємодії між замовниками та виконавцями фриланс-проектів на вебплатформі.

Аналіз потреб користувачів визначив основні завдання, які вони виконуватимуть у вебдодатку. Це охоплює створення та пошук проектів, взаємодію з іншими користувачами, управління проектами та спілкування. Важливо було забезпечити легкий доступ до всіх функцій, а також інтуїтивно зрозумілу навігацію та простоту використання.

Головна сторінка (рис. 3.1), куди потрапляє незареєстрований користувач, містить меню з популярними напрямки роботи та основні розділи, залежно від ролі користувача на сайті (замовник / виконавець).

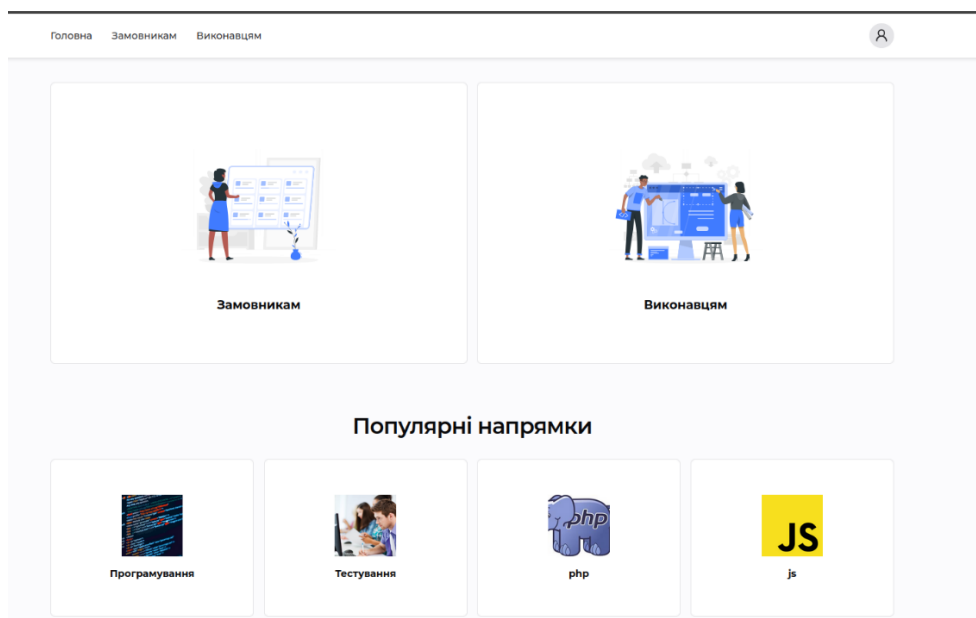


Рисунок 3.1 – Головна сторінку вебдодатку

Форма реєстрації (рис. 3.2) була створена з урахуванням принципів максимальної зручності для користувачів, які вперше знайомляться з вебдодатком і хочуть зареєструватись. Інтерфейс цієї форми виконаний у

					ПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						45
Змн.	Арк.	№ докум.	Підпис	Дата		

простому та зрозумілому стилі, містить лише найнеобхідніші поля, що дозволяє легко ввести потрібну інформацію та швидко зареєструватись. Також є зручне розділення форми для двох категорій користувачів: фрілансер та замовник. Отже, розроблена форма для реєстрації є важливим елементом користувацького досвіду, адже її інтуїтивність і простота роботи сприяють комфортному та швидкому проходженню процесу реєстрації для кожного нового відвідувача сайту.

Рисунок 3.2 – Форма реєстрації для нових користувачів

Форма авторизації (рис. 3.3) була реалізована з урахуванням потреб постійних користувачів платформи, які повертаються для подальшої роботи із сервісом. Її інтерфейс розроблено у єдиному стилі з формою реєстрації, що забезпечує візуальну узгодженість і загальну зручність навігації. У формі присутні лише два поля – для введення електронної пошти та паролю, – що мінімізує час входу до системи та не створює зайвих перешкод у взаємодії з вебплатформою.

Задля покращення досвіду користувача передбачено механізм обробки типових помилок, зокрема валідацію неправильного введення даних, підсвічування полів з помилками та інформативні підказки. Також додано

функцію «Показати/приховати пароль», що дає змогу перевірити коректність введення.

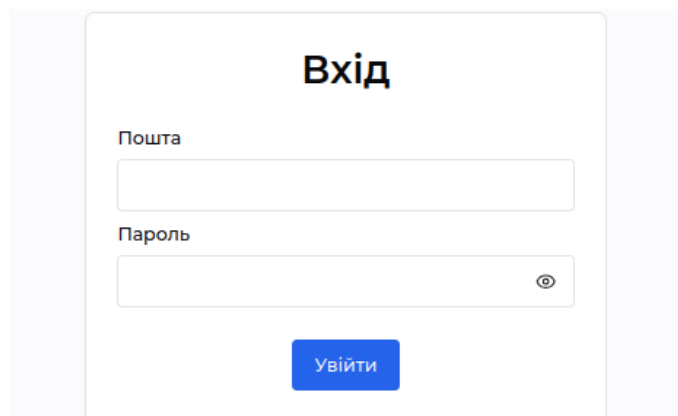


Рисунок 3.3 – Форма авторизації

На рисунку 3.4 зображено сторінку, де відображається перелік проєктів, доступ до якої здійснюється через кнопку «Виконавцям». На даній сторінці користувач бачить форму пошуку, що дозволяє вводити ключові слова для знаходження проєктів за їхніми назвами. Крім того, існує можливість використання фільтрів для сортування проєктів за розмірами бюджету навичками для виконання чи регіонами.

Сама сторінка надає перелік проєктів з базовою інформацією – назвою, коротким описом, бажаними навичками та бюджетом – і передбачає наявність переходу до розгорнутої сторінки, де надається більш детальна інформація про кожне замовлення.

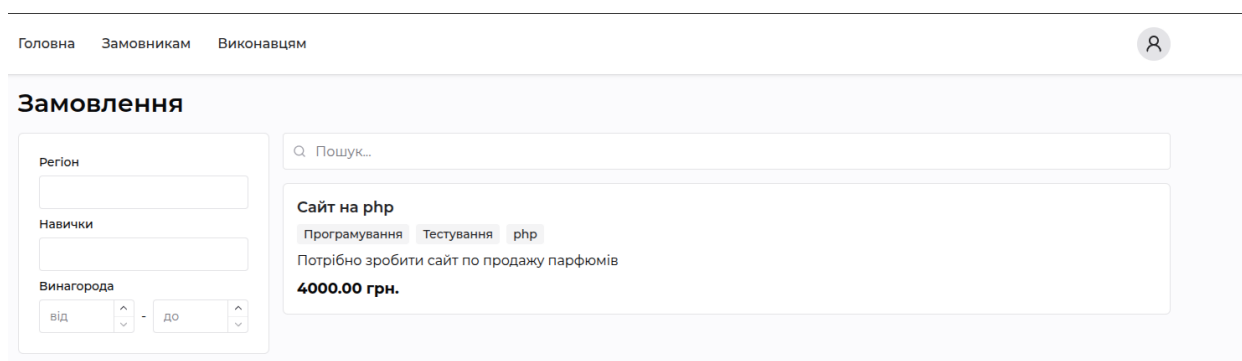


Рисунок 3.4 – Сторінка перегляду списку проєктів

Інтерфейс сторінки з детальним описом проєкту (рис. 3.5) містить повну інформацію щодо його основних характеристик: назви, бюджету, навички, поточного статусу, детального опису та дати публікації. Важливою особливістю

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		47

є функціонал, доступний лише для фрилансерів – вони можуть відгукнутися, щоб запропонувати свою кандидатуру для виконання роботи. Для замовників така опція недоступна.

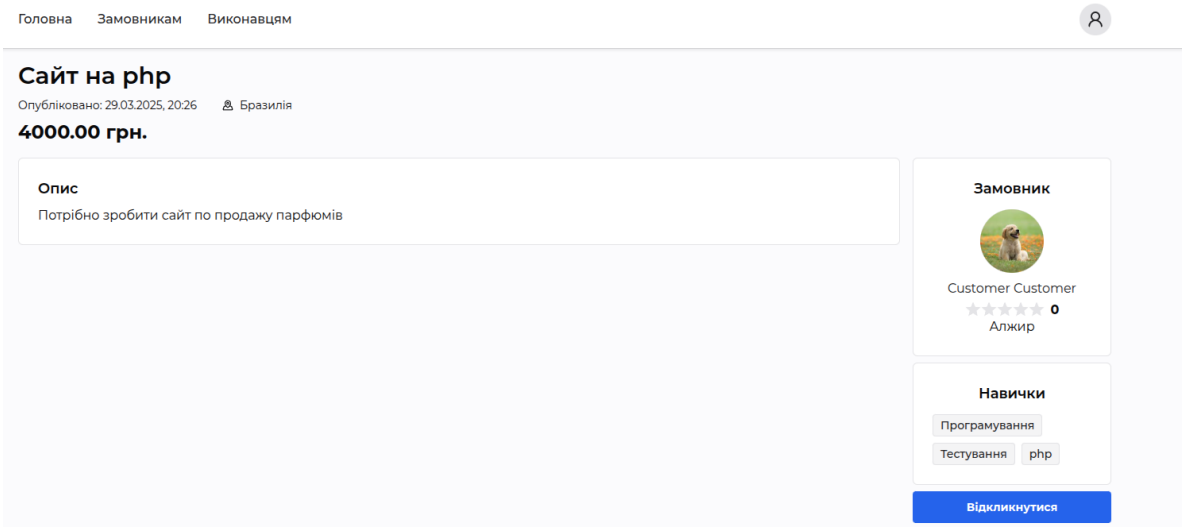


Рисунок 3.5 – Сторінка перегляду деталей замовлення

На рисунку 3.6 представлено сторінку профілю фрілансера, де зібрана ключова інформація про користувача, зокрема його повне ім'я, тип акаунту, регіон та фотографія профілю. На цій же сторінці також розміщено стислий опис користувача, а також відображається його рейтинг і відгуки від роботодавців, що дозволяє швидко оцінити професійний рівень виконавця.

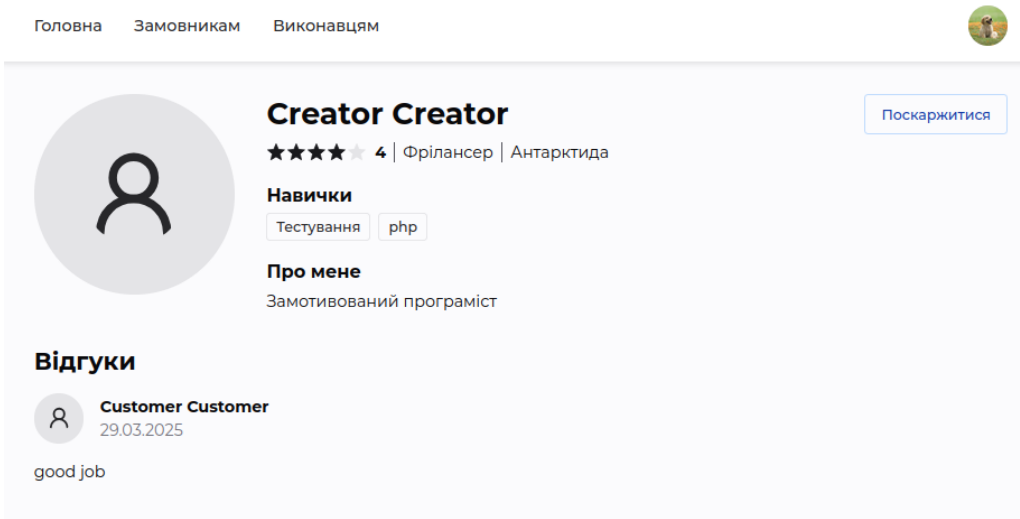


Рисунок 3.6 – Сторінка перегляду профілю користувача

Якщо це перегляд власного профілю, то завдяки кнопці «Редагувати» користувач має можливість переглянути та змінити дані свого профілю (рис. 3.7).

Редагування акаунту

×

Ім'я

creator

Прізвище

creator

Країна

Антарктида

Про себе

Замотивований програміст

Зберегти

Рисунок 3.7 – Сторінка редагування акаунту користувача

Після того, як фрилансер відгукнувся на сторінці проєкту, то замовник отримує сповіщення. На сторінці «Замовлення» можна переглянути усіх кандидатів (рис. 3.8), що відгукнулися для виконання роботи, прийняти або відхилити пропозицію, перейти до чату з виконавцем.

Головна

Замовникам

Виконавцям

+

Створити

Замовлення

Сайт на php

Розміщено

Створено: 29.03.2025

Кандидати

Creator Creator

До чату

✓

✗

Рисунок 3.8 – Сторінка редагування акаунту користувача

При відкритті чату (рис. 3.9) замовник бачить повідомлення, відправлене виконавцем, яке може містити уточнюючі питання, опис його вмінь, пропозиції щодо бюджету або просто сигнал про бажання взятися за виконання проєкту. Зі свого боку, власник замовлення має можливість прийняти запропоновану кандидатуру або відхилити її, що дозволяє оперативно визначитися з подальшими діями.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						49
Змн.	Арк.	№ докум.	Підпис	Дата		

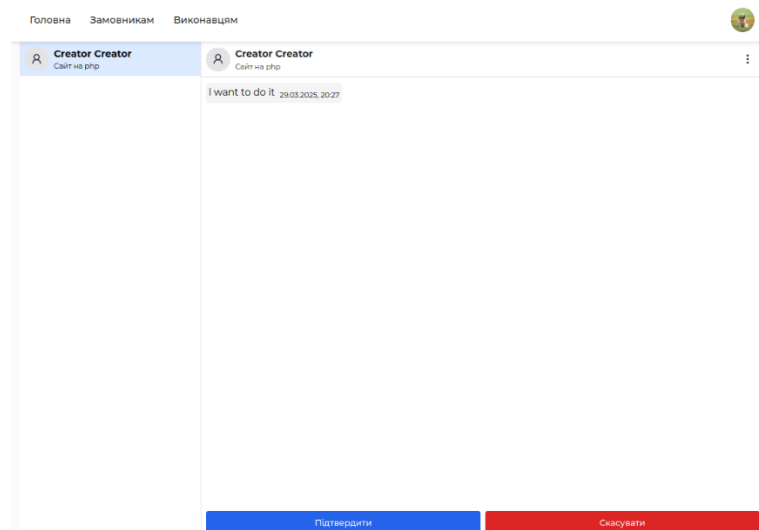


Рисунок 3.9 – Повідомлення від кандидата на виконання замовлення

Використання чату (рис. 3.10) у вебдодатку для співпраці з незалежними виконавцями є ефективним інструментом для забезпечення зручної та оперативної комунікації між користувачами платформи. Інтегрована безпосередньо у вебсистему, ця функціональність створює оптимальне середовище для обміну текстовими повідомленнями, що сприяє більш продуктивній взаємодії між учасниками. Такий підхід забезпечує комфортний обмін інформацією, сприяє швидкому вирішенню питань та покращує організацію спільної роботи. Крім того, функціонал збереження історії чату дозволяє зберігати важливі діалоги для подальшого аналізу, що забезпечує відстеження ходу обговорень та збереження критичних відомостей.

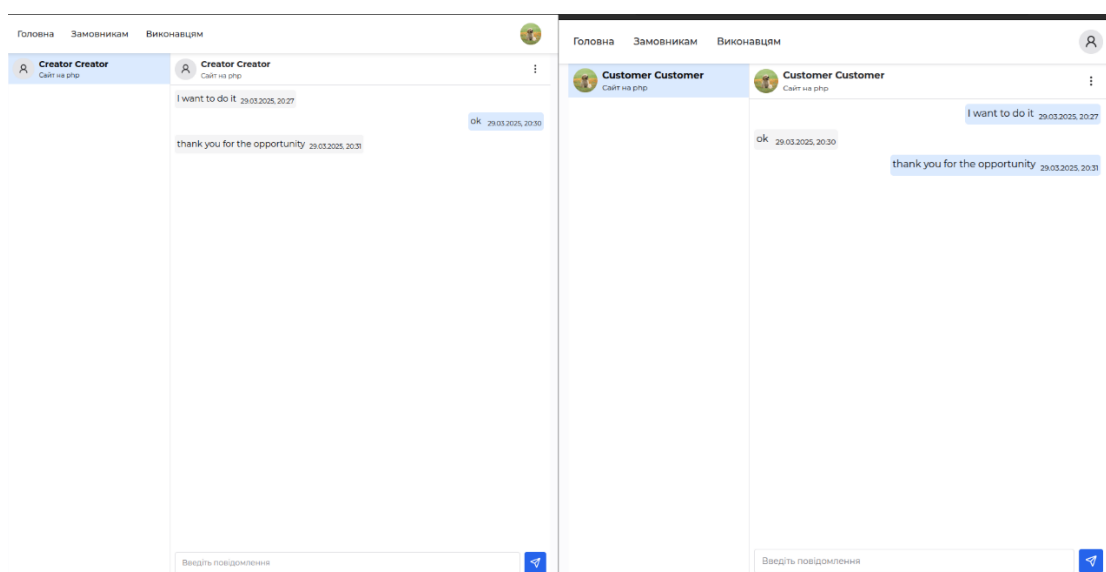


Рисунок 3.10 – Чат між замовником та виконавцем

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						50
Змн.	Арк.	№ докум.	Підпис	Дата		

Після успішного завершення роботи за замовленням, обидві сторони – і виконавець, і замовник – отримують можливість надати зворотній зв’язок, що є важливою складовою процесу співпраці. Оцінювання реалізовано за допомогою спливаючого модального вікна (рис. 3.11), через яке користувачі можуть швидко внести оцінку у вигляді рейтингу та залишити детальний коментар про якість. Такий підхід дозволяє миттєво зафіксувати враження, що є ключовим для формування довіри на платформі та сприяє відкритій комунікації між учасниками.

Конструктивні відгуки відіграють важливу роль у виявленні сильних і слабких сторін роботи, що, в свою чергу, допомагає підвищити стандарти обслуговування як для окремих користувачів, так і для системи в цілому. Отже, цей інтерфейс зворотного зв’язку стає ключовим елементом у підтримці якості та прозорості взаємодії у вебдодатку.

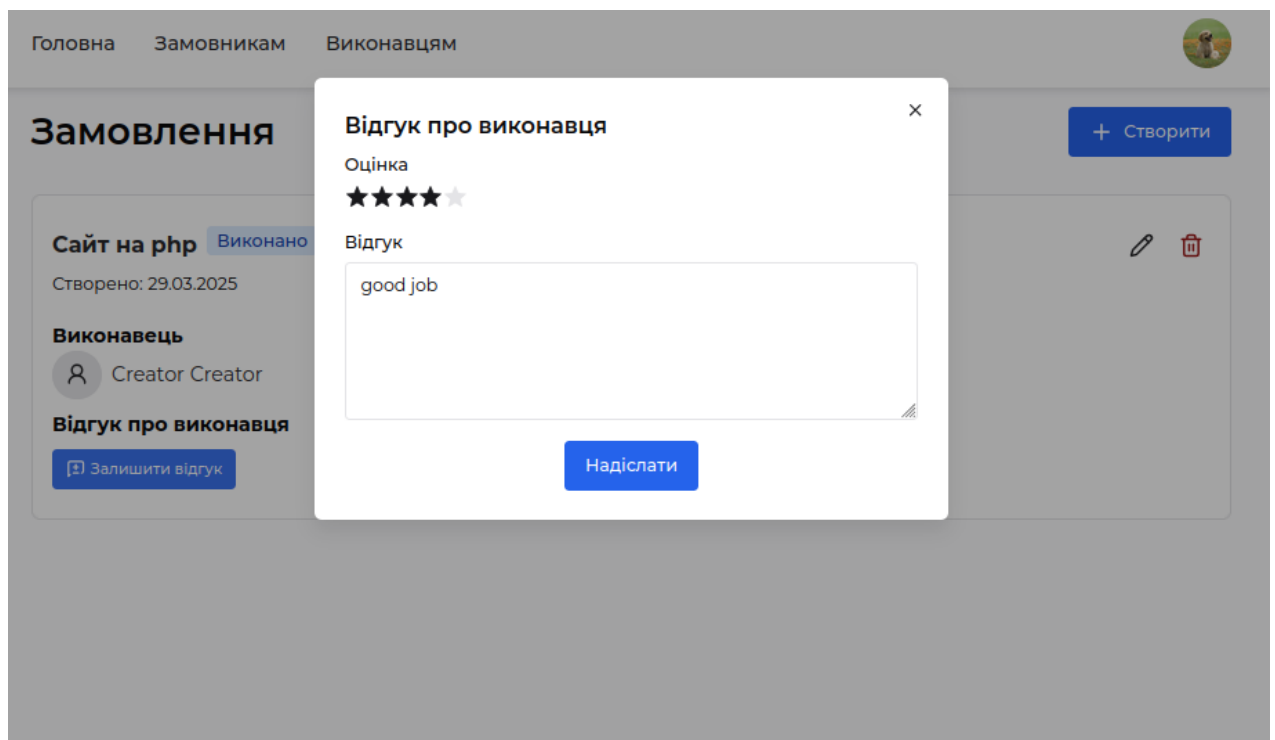


Рисунок 3.11 – Модальне вікно для оцінки співпраці

Користувач має можливість подати скаргу на іншого учасника платформи (рис. 3.12). Це дає змогу швидко повідомити про можливі порушення або неприйнятну поведінку, які можуть заважати нормальному процесу взаємодії. Скарга автоматично надсилається адміністратору для подальшого розгляду.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		51

Рисунок 3.12 – Модальне вікно для подачі скарги на користувача

Адміністратор уважно розглядає кожну скаргу (рис. 3.13), перевіряє надані докази та оцінює ситуацію, щоб ухвалити правильні рішення. У залежності від результатів аналізу можуть бути вжиті різні заходи, такі як попередження, тимчасове блокування або навіть постійне видалення користувача з системи. Цей механізм покликаний забезпечити безпеку та комфорт під час взаємодії, що, в свою чергу, допомагає підтримувати високий рівень довіри та стандартів поведінки на платформі.

Рисунок 3.13 – Сторінка перегляду списку скарг

Окрім перегляду скарг, адміністратор має всебічний доступ до всіх розділів сайту, що дозволяє йому здійснювати повноцінний моніторинг та керування інформацією на платформі. Адміністратор відповідає за формування та оновлення довідників (рис. 3.14), таких як перелік навичок та регіонів, що є критично важливими для правильної навігації користувачів та точного формування профілів і замовлень. Це завдання включає не лише додавання

нових записів, але й редагування існуючих даних для забезпечення їх актуальності й точності.

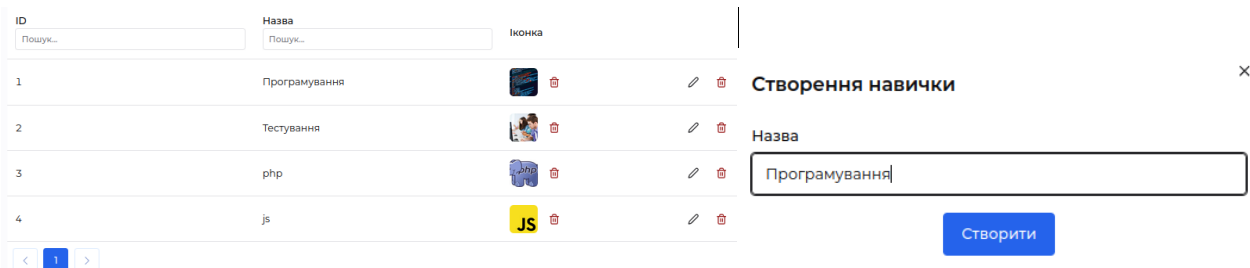


Рисунок 3.14 – Оновлення довідника «Навички»

Крім того, адміністратор має можливість виконувати різноманітні управлінські дії щодо створених замовлень та користувачів (рис. 3.15). Це включає перегляд, редагування, блокування або видалення записів, якщо виявлено порушення або для покращення роботи системи. Завдяки цим повноваженням адміністратор забезпечує стабільне та безпечне функціонування системи, що, в свою чергу, допомагає підтримувати високий рівень довіри серед користувачів вебдодатку.

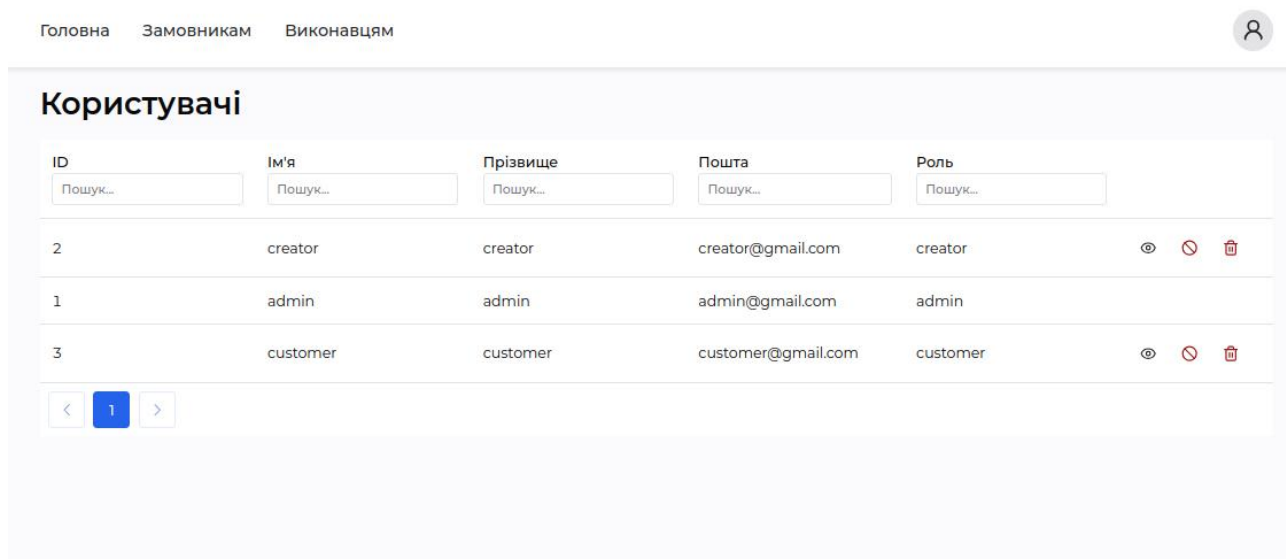


Рисунок 3.15 – Сторінка керування акаунтами користувачів

Таким чином, інтерфейс розробленого вебдодатку для взаємодії між фрілансерами та замовниками демонструє глибоке розуміння потреб кінцевих користувачів. Завдяки попередньому аналізу цільової аудиторії та визначенню ключових функціональних вимог, була створена зручна та логічно структурована платформа з інтуїтивно зрозумілою навігацією. Вона охоплює всі

основні сценарії використання – від реєстрації та авторизації до пошуку, перегляду проєктів і спілкування через вбудований чат. Особливу увагу приділено інструментам зворотного зв'язку. Крім того, розширені можливості для адміністратора системи та вбудовані механізми подання скарг допомагають підтримувати високий рівень безпеки та сервісу, що позитивно впливає на формування довіри до платформи з боку користувачів.

3.3. Тестування роботи розробленого вебдодатку

Ручне тестування, яке передбачає виконання тестових сценаріїв без використання автоматизованих засобів, залишається одним із найпопулярніших способів перевірки функціональності вебзастосунків. Його основна мета – виявлення помилок, невідповідностей з очікуваною поведінкою та перевірка зручності використання застосунку [8]. У цьому процесі тестувальники користуються документованими сценаріями, які оформлені у вигляді структурованих таблиць. У них зазначаються цілі тесту, вхідні дані, послідовність кроків, очікувані та фактичні результати, а також підсумковий висновок про проходження тесту [15].

Для розробленого вебдодатку тестування проводилося відповідно до реальних сценаріїв взаємодії користувачів. Основною метою було перевірити, як працюють усі функціональні модулі платформи – від реєстрації та авторизації до перегляду проєктів, коментування, обміну повідомленнями через чат, а також управління адміністраторським кабінетом і відображення інтерфейсів для замовників і фрілансерів. Такий підхід допоміг забезпечити високу якість роботи системи та підтвердити її відповідність вимогам користувачів.

1. Головна сторінка вебдодатку повинна забезпечувати перше позитивне враження та виконувати роль навігаційного центру, особливо для нових або незареєстрованих користувачів. У процесі перевірки особлива увага приділялась наявності меню, актуальних розділів, тематичних блоків з напрямками проєктів, а також загальному дизайну сторінки. Важливо, щоб усі ці елементи були не

					ІПЗ.КР.Б – 121 – 25 – ІЗ	Арк.
						54
Змн.	Арк.	№ докум.	Підпис	Дата		

лише видимі, а й функціонували відповідно до задуманої логіки. Було перевірено адаптивність під дві основні ролі – замовника і виконавця. Усі елементи інтерфейсу завантажуються коректно, функціонують належним чином, візуально відповідають дизайну та не викликають складнощів у користуванні.

2. Реєстрація користувача протестована на предмет зручності, зрозумілості та швидкодії. Після натискання відповідної кнопки відкривається форма з поділом на дві ролі, що дозволяє обрати відповідний тип облікового запису ще до заповнення даних. Поля введення не перевантажені, містять лише найнеобхідніше (ім'я, email, пароль, роль), що сприяє легкому входу в систему. Система успішно обробляє введені дані, реєстрація завершується без затримок, а новий користувач отримує підтвердження про створення акаунта. Створені облікові записи зберігаються в базі даних, і доступні в подальшому для авторизації.

3. Авторизація користувача є важливою частиною взаємодії з системою, адже саме вона дозволяє увійти до персоналізованого простору. Була перевірена форма входу, яка має мінімум необхідних полів (email і пароль) для швидкого доступу. Після введення правильних даних система переадресовує користувача на його профіль без помітних затримок. Не виявлено проблем із розпізнаванням ролей – як фрилансери, так і замовники потрапляють до відповідного функціоналу, без необхідності подальшої ідентифікації. Уся логіка авторизації працює стабільно, без збоїв або неочікуваних поведінкових сценаріїв.

4. Перегляд списку створених замовлень забезпечує зручну навігацію та пошук потрібного проєкту. Користувач може відкрити сторінку з доступними завданнями та скористатися пошуковим рядком або фільтрами, щоб швидше знайти релевантні пропозиції. Випробовувались фільтрація за бюджетом, ключовими словами, необхідними навичками та регіонами. Всі ці функції працюють очікувано – результати змінюються миттєво згідно з заданими параметрами. Перелік проєктів завантажувється швидко, а відображена інформація є актуальною та структурованою.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						55
Змн.	Арк.	№ докум.	Підпис	Дата		

5. Детальна сторінка проєкту є логічним продовженням після перегляду загального списку завдань. На цій сторінці перевірялась наявність повної інформації про обраний проєкт: назва, опис, бюджет, терміни, вимоги до виконавця, дата публікації та статус. Якщо користувач має роль «виконавець», йому відображається активна кнопка для відгуку. Всі ці елементи виводяться відповідно до ролі користувача та налаштувань проєкту. Інтерфейс завантажується без затримок, і користувач одразу розуміє, як діяти далі.

6. Перегляд та редагування профілю дає можливість користувачам ознайомитись із власною інформацією або внести зміни. Перевірка охопила наявність основних полів профілю (ім'я, тип акаунта, регіон, фото, опис, рейтинги, відгуки) та їх коректне відображення після входу. Редагування виконується натисканням відповідної кнопки, після чого відкривається форма з можливістю оновити окремі поля або завантажити нове фото. Усі зміни зберігаються коректно, не викликаючи помилок або перезавантаження сторінки. Нові дані оновлюються як в особистому кабінеті, так і на публічній сторінці профілю.

7. Чат між замовником та виконавцем виконує роль основного каналу зв'язку всередині платформи. Після початку взаємодії (відгук або повідомлення) відкривається чат, у якому можна миттєво обмінюватись повідомленнями. Повідомлення доставляються через WebSocket без помітних затримок, і збереження історії забезпечує можливість повернутись до будь-якої переписки. Тестування показало стабільну роботу чату, що важливо для довгострокової співпраці користувачів. Жодних функціональних або технічних проблем не виявлено.

8. Надсилення зворотного зв'язку реалізовано через спливаюче модальне вікно, яке з'являється після завершення замовлення. Користувач має змогу залишити рейтинг (у вигляді зірочок) і коментар, який буде збережено в системі та відображатиметься у профілі виконавця або замовника. Важливо, що обидві сторони мають право на зворотний зв'язок, що підвищує прозорість і довіру на

					ІПЗ.КР.Б – 121 – 25 – ІЗ	Арк.
						56
Змн.	Арк.	№ докум.	Підпис	Дата		

платформі. Усі введені дані обробляються коректно, зберігаються в базі та оновлюють відповідний розділ автоматично.

9. Подання скарги реалізовано як окрема дія, доступна з профілю іншого користувача. Натискання кнопки відкриває форму, де можна описати проблему або некоректну поведінку. Після заповнення даних скарга надсилається в адмінпанель, де вона доступна для перегляду і подальших дій. Цей механізм дозволяє оперативно реагувати на порушення, сприяючи підтримці безпечного середовища для всіх користувачів. Усі дії виконуються стабільно, без помилок, з підтвердженням про надсилання скарги.

10. Адміністрування системи перевірялось через спеціальну панель, доступ до якої мають лише адміністратори. Основна увага приділялась роботі з довідниками (наприклад, навички, регіони), перегляду профілів користувачів і проєктів, можливості додавання або редагування записів. Усі зміни, які адміністратор вносить, миттєво зберігаються та відображаються у відповідних розділах платформи. Функціонал адміністрування працює згідно з описом, не виходячи за межі ролей, доступних у системі.

Також були проведені перевірки, що спрямовані на тестування реакції вебдодатку на запити до неіснуючих URL-адрес. Під час цих тестів здійснювалися навігаційні спроби до неіснуючих роутів, щоб переконатися, що вебсистема коректно розпізнає помилкові запити та автоматично відображає розроблену сторінку з помилкою 404 (рис. 3.16).

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						57
Змн.	Арк.	№ докум.	Підпис	Дата		

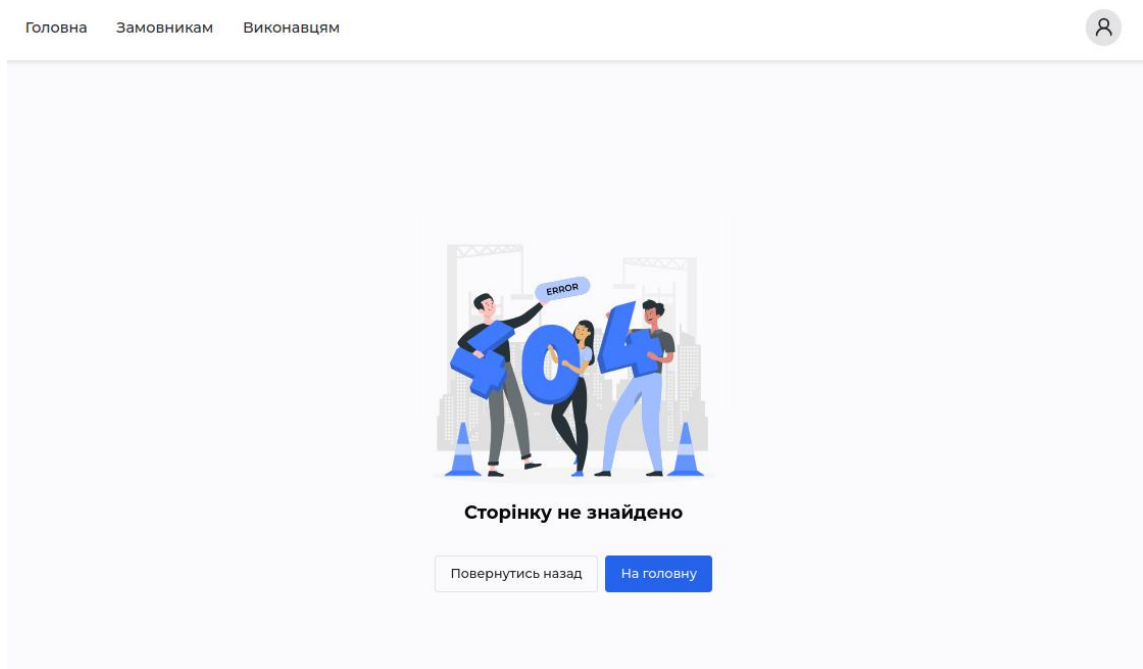


Рисунок 3.16 – Сторінка для відображення помилки 404

Цей підхід не тільки дозволив перевірити, чи правильно налаштована маршрутизація, але й допоміг з'ясувати, чи з'являється повідомлення, яке підказує користувачу, як діяти в разі помилки. Правильне оформлення сторінки помилки 404 є важливим елементом покращення користувацького досвіду, адже воно дає змогу швидко повернутися назад або перейти на головну сторінку вебдодатку.

Тестування вебдодатку дало змогу ретельно перевірити всі важливі функціональні елементи. Документування кожного сценарію тестування не лише допомогло виявити дефекти на етапах розробки, але й сприяло подальшому вдосконаленню інтерфейсу вебсистеми. Отримані результати підвищують якість обслуговування користувачів, забезпечують належний рівень безпеки та зручності використанні вебдодатку для співпраці.

Висновки до третього розділу

У ході роботи було розроблено та реалізовано функціональний вебдодаток для пошуку та співпраці з незалежними виконавцями. Архітектура платформи побудована на логічному поділі компонентів, що забезпечує її гнучкість у масштабуванні, підтримці та розширенні. Однією з ключових особливостей

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						58
Змн.	Арк.	№ докум.	Підпис	Дата		

рішення є впровадження контейнеризації за допомогою Docker та Docker Compose, що гарантує ізольоване розгортання кожного модуля.

Інтерфейс вебзастосунку спроектовано на основі аналізу цільової аудиторії та її сценаріїв взаємодії. Він охоплює всі етапи роботи користувача: від реєстрації та авторизації до створення, перегляду проєктів, комунікації через повідомлення, керування профілем та надання відгуків. Окрема увага приділена ролям: фрілансери можуть подавати заявки та спілкуватися із замовниками, а адміністративна панель забезпечує модерацію, обробку скарг та управління даними платформи.

Особливо важливим етапом стала ручна перевірка системи, в рамках якої були створені структуровані сценарії тестування всіх ключових сторінок і функцій. Це дало змогу виявити та усунути дрібні недоліки, а також підтвердити, що платформа відповідає технічним і функціональним вимогам. Значну увагу було приділено доступності функцій за ролями (фрилансер, замовник, адміністратор), коректності маршрутизації та точності передачі даних.

У процесі тестування були перевірені всі ключові модулі: реєстрація, авторизація, сторінки проєктів, профілі користувачів, чат, система оцінювання, адміністративний розділ та обробка помилок. Успішне проходження тестових сценаріїв підтвердило стабільність роботи платформи. Основні функції виконують свої завдання, а інтерфейс виявився інтуїтивно зрозумілим для всіх категорій користувачів.

У підсумку, розроблений вебзастосунок повністю відповідає запланованим цілям. Він забезпечує комфортну взаємодію між користувачами, сприяє ефективному веденню замовлень, обміну повідомленнями та підтримує високу прозорість завдяки системі оцінювання та механізму подання скарг. вебдодаток є завершеним продуктом з високим рівнем якості.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						59
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У межах виконання кваліфікаційної роботи було проведено глибокий аналіз, ретельне проєктування та повноцінна розробка вебдодатку для пошуку та співпраці з незалежними виконавцями.

Першочергово було здійснено аналіз сучасного стану ринку фрилансу, що включав дослідження існуючих платформ, їхніх особливостей, популярних механізмів взаємодії та основних викликів, з якими стикаються користувачі. У рамках цього дослідження було розглянуто провідні сервіси для співпраці між виконавцями та замовниками, зокрема Freelancehunt, Freelance.ua та Upwork. Дослідження їхнього функціоналу, організації роботи та підходів дозволило визначити найбільш важливі аспекти, які варто врахувати при створенні власного вебдодатку.

Одним із ключових висновків аналізу стало розуміння необхідності забезпечення широкого функціоналу, який дасть змогу користувачам легко публікувати бажані замовлення, знаходити відповідних виконавців, спілкуватися та управляти своїми замовленнями. Важливим аспектом є можливість гнучкого налаштування платформи для різних категорій користувачів, що дозволяє адаптувати процес роботи під їхні потреби.

Було створено архітектуру та визначено ключові функціональні можливості вебдодатку, що забезпечують зручну та ефективну взаємодію між користувачами. Використання діаграм варіантів використання дозволило чітко структурувати функціональність системи та відобразити логічні взаємозв'язки між її компонентами, що сприяло вдосконаленню архітектурного проєкту та оптимізації процесів розробки.

Особливу увагу було приділено проєктуванню бази даних, яка забезпечує ефективне зберігання та управління інформацією. Було визначено зв'язки між таблицями та типи даних, що допомагає підтримувати цілісність інформації, забезпечити та оптимізувати взаємодію між елементами системи.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						60
Змн.	Арк.	№ докум.	Підпис	Дата		

Для створення вебдодатку для співпраці з незалежними виконавцями були обрані сучасні технології, які забезпечують високу продуктивність. Використання Docker дало змогу створити контейнеризоване середовище для розгортання застосунку, що забезпечує ізоляцію окремих модулів і спрощує процес налаштування. Nginx використовується як вебсервер для ефективного управління запитами та балансування навантаження. Node.js та Express стали основою бекенд-розробки, забезпечуючи швидку розробку та подальшу надійну роботу. React.js застосовується для створення динамічного, зручного та адаптивного фронтенду, що покращує взаємодію користувачів із системою. WebSocket на базі Socket.io реалізовано для підтримки обміну повідомленнями в реальному часі в чаті, що є критично важливим для спілкування по ключовим моментам між фрилансерами та замовниками.

На основі усіх отриманих та зібраних даних, визначених вимог було створено вебдодаток для пошуку та співпраці з незалежними виконавцями, який не тільки відповідає сучасним стандартам зручності та функціональності, але й гарантує високий рівень безпеки та конфіденційності даних. Реалізація цього проекту стала важливим кроком у створенні ефективного середовища для роботи та співпраці між фрилансерами та замовниками, а також заклала основу для подальшого вдосконалення та розширення функціоналу.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						61
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Docker Compose. Manuals [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.docker.com/compose/>
2. Docker: Простими словами про контейнеризацію [Електронний ресурс] – Режим доступу до ресурсу: <https://dan-it.com.ua/uk/blog/docker-prostymi-slovami-o-kontejnerizaczii/>
3. Freelancehunt [Електронний ресурс] – Режим доступу до ресурсу: <https://freelancehunt.com/ua>
4. Freelance.ua [Електронний ресурс] – Режим доступу до ресурсу: <https://freelance.ua/>
5. JWT [Електронний ресурс] – Режим доступу: <https://jwt.io>
6. Learn React. Describing the UI [Електронний ресурс] – Режим доступу до ресурсу: <https://react.dev/learn/describing-the-ui>
7. Miell I. Docker in Practice / I. Miell and A.H. Sayers. – Manning, 2019. – 384 p.
8. Myers G.J. The Art of Software Testing / G.J. Myers, C. Sandler, T. Badgett. – Wiley, 2011. – 256 p.
9. MySQL. Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://dev.mysql.com/doc/>
10. Nichter D. Efficient MySQL Performance: Best Practices and Techniques / O'Reilly Media, 2022. – 335 p.
11. Schwarzmüller M. React Key Concepts: An in-depth guide to React's core features / Packt Publishing, 2022. – 590 p.
12. Upwork [Електронний ресурс] – Режим доступу до ресурсу: <https://www.upwork.com/>
13. Wieruch R. The Road to React: The React.js / Independently published, 2025. – 410 p.
14. Аквіно К. Front-end. Клієнтська розробка для професіоналів. Node.js, ES6, REST/ К. Аквіно, Т. Ганді, 2017. – 512 с.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						62
Змн.	Арк.	№ докум.	Підпис	Дата		

15. Блек Р. Ключові процеси тестування. Планування, підготовка, проведення, удосконалення / Лорі, 2017. – 544 с.
16. Браун І. Веброзробка із застосуванням Node і Express / PRINT2PRINT, 2021. – 336 с.
17. Гайна Г.А. Основи проектування баз даних: навчальний посібник / Г.А. Гайна. – К.: КНУБА, 2008. – 200 с.
18. Граф М.С., Кузьменко О.В. Веборієнтовані системи і технології. Навчальний посібник. Житомир: «Житомирська політехніка», 2024. – 180 с.
19. Дейт К. Дж. Введення у системи баз даних / К. Дж. Дейт. 8-ме вид. – Київ: Діалектика, 2020. – 1328 с.
20. Мікросервісна архітектура для початківців [Електронний ресурс] – Режим доступу до ресурсу: <https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-one/>

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						63
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						64
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг коду для реалізації чату

```

const { Server } = require('socket.io');
const dotenv = require('dotenv');
const axios = require('axios');

dotenv.config();

const io = new Server({
  cors: {
    origin: "*",
    credentials: true
  }
});

const connectedUsers = new Map();

io.on('connection', async (socket) => {
  const authToken = socket.handshake.headers.authorization;
  const axiosConfig = {
    headers: {
      'Authorization': authToken
    },
    baseURL: `${process.env.API}/api`,
  };
  let userId = null;

  try {
    const user = await axios.get(`/users/me`, axiosConfig);
    connectedUsers.set(user.data.id, socket.id);
    userId = user.data.id;
  } catch (err) {
    socket.disconnect();
  }

  socket.on('create_room', async ({ order_id, message }, callback) => {
    try {
      const response = await axios.post('/rooms', { order_id,
message }, axiosConfig);
      const recipientSocketId =
connectedUsers.get(response.data.user_second_id);
      if (recipientSocketId) {
        io.to(recipientSocketId).emit('new_room',
response.data);
      }

      callback({ success: true });
    } catch (error) {
      callback({ success: false, message: 'Помилка при створенні
чату' });
    }
  });
});

```

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						65
Змн.	Арк.	№ докум.	Підпис	Дата		

```

socket.on('read_messages', async (roomId) => {
  await axios.patch(`/messages/room/${roomId}`, {}, axiosConfig);
});

socket.on('join_room', async (roomId) => {
  try {
    const room=await axios.get(`/rooms/number/${roomId}`,
axiosConfig);
    const rooms = Array.from(socket.rooms);
    rooms.forEach((room) => {
      if (room !== socket.id) {
        socket.leave(room);
      }
    });
    socket.join(roomId);
    await axios.patch(`/messages/room/${room.data.id}`,
    {}, axiosConfig);

    const messages=await
    axios.get(`/messages/notification/${room.data.id}`, axiosConfig);

    socket.emit('notification', messages.data);
  } catch (error) {
    console.log(error);
  }
});

socket.on('send_message', async (data) => {
  try {
    if (!data.roomId || !data.message || !data.userId) {
      throw new Error('Відсутні обов'язкові поля');
    }
    const roomResponse = await
    axios.get(`/rooms/number/${data.roomId}`, axiosConfig);
    const roomData = roomResponse.data;
    const messageData = {
      user_id: Number(data.userId),
      room_id: roomData.id,
      text: data.message,
      date: new Date().toISOString(),
    };
    const messagesResponse = await axios.post('/messages',
messageData, axiosConfig);
    io.to(data.roomId).emit('receive_message',
messagesResponse.data);

    await axios.patch(`/messages/room/${roomData.id}`, {},
    axiosConfig);

    const messages = await
    axios.get(`/messages/notification/${roomData.id}`, axiosConfig);
    const recipientSocketId =
    connectedUsers.get(messages.data.userId);

```

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						66
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        if (recipientSocketId) {
            io.to(recipientSocketId).emit('notification',
messages.data);
        }
    } catch (error) {
        console.log(error);
    }
});

socket.on('delete_room', async ({ roomNumber, roomId }) => {
    try {
        const { data: room } = await axios.get(`/rooms/${roomId}`,
axiosConfig);

        if (!room) {
            throw new Error('Відсутня кімната');
        }

        await axios.delete(`/rooms/${roomId}`, axiosConfig);
        socket.rooms.delete(roomNumber);
        const recipientSocketId1 =
connectedUsers.get(room.user_first_id);
        const recipientSocketId2 =
connectedUsers.get(room.user_second_id);

        [recipientSocketId1, recipientSocketId2].forEach(socketId => {
            if (socketId) {
                io.to(socketId).emit('remove_room', roomId);
            }
        });

    } catch (error) {
        socket.emit('delete_chat_error', error.message);
    }
});

socket.on('disconnect', () => {
    connectedUsers.delete(userId);
});

const PORT = process.env.SOCKET_PORT || 3001;
io.listen(PORT);
console.log(`Socket.IO server running on port ${PORT}`);

```

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						67
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг коду FileService

```

const path = require('path');
const { models } = require('../models');
const AWS = require('aws-sdk');

class FileService {
  constructor() {
    this.errorMessages = {
      entityNotFound: {
        user: 'Користувача не знайдено',
        skill: 'Навичку не знайдено'
      },
      fileNotUploaded: 'Файл не завантажено'
    };

    this.s3 = new AWS.S3({
      endpoint: 'http://minio:9000',
      accessKeyId: 'rootroot',
      secretAccessKey: 'rootroot',
      s3ForcePathStyle: true,
      signatureVersion: 'v2',
      sslEnabled: false
    });

    this.initializeBucket();
  }

  async initializeBucket(bucketName = 'files') {
    try {
      const params = {
        Bucket: bucketName
      };

      try {
        await this.s3.headBucket(params).promise();
      } catch (error) {
        if (error.code === 'NotFound' || error.code === 'NoSuchBucket') {
          await this.s3.createBucket(params).promise();

          const publicReadPolicy = {
            Version: '2012-10-17',
            Statement: [
              {
                Effect: 'Allow',
                Principal: '*',
                Action: ['s3:GetObject'],
                Resource: [`arn:aws:s3:::${bucketName}/*`]
              }
            ]
          };

          await this.s3.putBucketPolicy({
            Bucket: bucketName,

```

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						68
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        Policy: JSON.stringify(publicReadPolicy)
      }).promise();
    } else {
      throw error;
    }
  }
} catch (error) {
  console.error('Error initializing bucket:', error);
  throw error;
}
}

async uploadFile({
  modelInstance,
  file,
  fileNamePrefix,
  fileField,
  forcedExtension = null,
  bucketName = 'files'
}) {
  try {
    if (!modelInstance) {
      throw new Error('Сутність не знайдено');
    }

    const fileExtension = forcedExtension ||
path.extname(file.originalname);
    const fileName =
`${fileNamePrefix}_${modelInstance.id}_${Date.now()}${fileExtension}`;

    await this.deleteExistingFile(modelInstance[fileField]);

    const params = {
      Bucket: bucketName,
      Key: fileName,
      Body: file.buffer
    };

    await this.s3.putObject(params).promise();
    await modelInstance.update({ [fileField]: fileName });

    return fileName;
  } catch (error) {
    console.error('Помилка завантаження файлу:', error);
    throw error;
  }
}

async getFileUrl(fileName, bucketName = 'files', expirationInSeconds =
3600) {
  try {
    if (!fileName) {
      throw new Error('Назву файлу не вказано');
    }
  }
}

```

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						69
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        const params = {
            Bucket: bucketName,
            Key: fileName,
            Expires: expirationInSeconds
        };

        const signedUrl = await this.s3.getSignedUrlPromise('getObject',
params);
        return signedUrl.replace('minio:9000', 'localhost:9000');
    } catch (error) {
        console.error('Помилка отримання URL файлу:', error);
        throw error;
    }
}

async getFile(fileName, bucketName = 'files') {
    try {
        if (!fileName) {
            throw new Error('Назву файлу не вказано');
        }

        const params = {
            Bucket: bucketName,
            Key: fileName
        };

        const data = await this.s3.getObject(params).promise();
        return data.Body;
    } catch (error) {
        throw error;
    }
}

async deleteExistingFile(fileName, bucketName = 'files') {
    if (!fileName) return;

    const params = {
        Bucket: bucketName,
        Key: fileName
    };

    try {
        await this.s3.deleteObject(params).promise();
    } catch (error) {
    }
}

async getModelInstance(modelName, id) {
    const model = await models[modelName].findOne({ where: { id } });
    if (!model) {
        throw new
Error(this.errorMessages.entityNotFound[modelName.toLowerCase()]);
    }
    return model;
}

```

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

```

async uploadImage(userId, file) {
    const user = await this.getModelInstance('User', userId);
    return this.uploadFile({
        modelInstance: user,
        file,
        fileNamePrefix: 'user',
        fileField: 'image'
    });
}

async getUserImage(userId) {
    const user = await this.getModelInstance('User', userId);
    if (!user.image) {
        return null;
    }
    return this.getFileUrl(user.image);
}

async deleteImage(userId) {
    const user = await this.getModelInstance('User', userId);
    if (user.image) {
        await this.deleteExistingFile(user.image);
        await user.update({ image: null });
    }
}

async uploadPDF(userId, file) {
    const user = await this.getModelInstance('User', userId);
    return this.uploadFile({
        modelInstance: user,
        file,
        fileNamePrefix: 'user',
        fileField: 'file',
        forcedExtension: '.pdf'
    });
}

async getUserPDF(userId) {
    const user = await this.getModelInstance('User', userId);
    if (!user.file) {
        return null;
    }
    return this.getFileUrl(user.file);
}

async deletePDF(userId) {
    const user = await this.getModelInstance('User', userId);
    if (user.file) {
        await this.deleteExistingFile(user.file);
        await user.update({ file: null });
    }
}

async uploadSkillImage(skillId, file) {

```

					ПІЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						71
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        const skill = await this.getModelInstance('Skill', skillId);
        return this.uploadFile({
            modelInstance: skill,
            file,
            fileNamePrefix: 'skill',
            fileField: 'image'
        });
    }

    async getSkillImage(skillId) {
        const skill = await this.getModelInstance('Skill', skillId);
        if (!skill.image) {
            return null;
        }
        return this.getFileUrl(skill.image);
    }

    async deleteSkillImage(skillId) {
        const skill = await this.getModelInstance('Skill', skillId);
        if (skill.image) {
            await this.deleteExistingFile(skill.image);
            await skill.update({ image: null });
        }
    }
}

module.exports = new FileService();

```

					ПІЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						72
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістингу коду Search

```

const express = require('express');
const router = express.Router();
const { Op, Sequelize } = require('sequelize');
const { models } = require('../models');
const getPaginationParams = require("../utils/pagination");
const FileService = require("../services/FileService");

router.get('/users', async (req, res) => {
  try {
    const { query, region, rating, role, skills } = req.query;
    const { page, limit, offset } = getPaginationParams(req.query);

    const whereCondition = {
      role: { [Op.ne]: 'admin' }
    };

    if (query) {
      whereCondition[Op.or] = [
        { name: { [Op.like]: `%${query}%` } },
        { surname: { [Op.like]: `%${query}%` } }
      ];
    }

    if (role && ['creator', 'customer'].includes(role)) {
      whereCondition.role = role;
    }

    if (rating) {
      try {
        const ratingValues = JSON.parse(rating);
        if (Array.isArray(ratingValues) && ratingValues.length > 0) {
          const validRatings = ratingValues.every(r =>
            !isNaN(parseFloat(r)) &&
            parseFloat(r) >= 0 &&
            parseFloat(r) <= 5
          );

          if (!validRatings) {
            return res.status(400).json({
              message: 'All ratings must be numbers between 0 and 5'
            });
          }

          whereCondition.rating = {
            [Op.in]: ratingValues.map(r => parseFloat(r))
          };
        } else {
          return res.status(400).json({
            message: 'Rating must be a non-empty array of numbers'
          });
        }
      } catch (e) {

```

					ПІЗ.КР.Б – 121 – 25 – ПІЗ	Арк.
						73
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        return res.status(400).json({
            message: 'Invalid rating format. Expected JSON array of
numbers'
        });
    }
}

const queryOptions = {
    where: whereCondition,
    attributes: { exclude: ['password'] },
    limit,
    offset,
    distinct: true,
    include: [{
        model: models.Region,
        as: 'userRegion',
        attributes: ['id', 'name']
    }]
};

if (region) {
    try {
        const regionIds = JSON.parse(region);
        if (Array.isArray(regionIds) && regionIds.length > 0 &&
!regionIds.some(isNaN)) {
            queryOptions.include[0].where = {
                id: {
                    [Op.in]: regionIds
                }
            };
        } else {
            return res.status(400).json({
                message: 'Invalid region format. Expected JSON array of region
IDs'
            });
        }
    } catch (e) {
        return res.status(400).json({
            message: 'Invalid region format. Expected JSON array'
        });
    }
}

let skillsInclude = {
    model: models.UsersSkill,
    as: 'userSkills',
    include: [{
        model: models.Skill,
        as: 'skill'
    }]
};

if (skills) {
    let skillIds;
    try {

```

					ПІЗ.КР.Б – 121 – 25 – ПІЗ	Арк.
						74
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        skillIds = JSON.parse(skills);
    } catch (e) {
        return res.status(400).json({ message: 'Invalid skills format' });
    }

    if (Array.isArray(skillIds) && skillIds.length > 0 &&
!skillIds.some(isNaN)) {
        queryOptions.where = {
            ...whereCondition,
            id: {
                [Op.in]: Sequelize.literal(`
                SELECT user_id
                FROM users_skill
                WHERE skill_id IN (${skillIds.join(',')})
                GROUP BY user_id
                HAVING COUNT(DISTINCT skill_id) = ${skillIds.length}
                `)
            }
        };
    } else {
        return res.status(400).json({ message: 'Invalid skills format' });
    }
}

queryOptions.include.push(skillsInclude);

const { count, rows: users } = await
models.User.findAndCountAll(queryOptions);
const usersWithImages = await Promise.all(users.map(async (user) =>
{
    const imageUrl = await FileService.getUserImage(user.id);
    return { ...user.toJSON(), image: imageUrl };
})));

res.status(200).json({
    total: count,
    page,
    limit,
    totalPages: Math.ceil(count / limit),
    users: usersWithImages
});
} catch (error) {
    console.error('Error in users search:', error);
    res.status(400).json({ message: error.message });
}
});

router.get('/orders', async (req, res) => {
    try {
        const { query, minPrice, maxPrice, region, worktime, skills } =
req.query;
        const { page, limit, offset } = getPaginationParams(req.query);

        const whereConditions = {
            status_id: { [Op.notIn]: [3, 4] }

```

					ПІЗ.КР.Б – 121 – 25 – ПІЗ	Арк.
						75
Змн.	Арк.	№ докум.	Підпис	Дата		

```

};
const queryOptions = {
  where: whereConditions,
  limit,
  offset,
  distinct: true,
  subQuery: false,
  include: []
};

if (query) {
  whereConditions.title = { [Op.like]: `%${query}%` };
}
if (minPrice || maxPrice) {
  whereConditions.price = {};
  if (minPrice) {
    whereConditions.price[Op.gte] = parseFloat(minPrice);
  }
  if (maxPrice) {
    whereConditions.price[Op.lte] = parseFloat(maxPrice);
  }
}
if (worktime) {
  whereConditions.worktime = worktime;
}

if (region) {
  try {
    let regionIds = JSON.parse(region);
    if (Array.isArray(regionIds) && regionIds.length > 0 &&
!regionIds.some(isNaN)) {
      whereConditions.region = {
        [Op.in]: regionIds
      };
    } else {
      return res.status(400).json({ message: 'Invalid regions
format' });
    }
  } catch (e) {
    return res.status(400).json({ message: 'Invalid regions format' });
  }
}

queryOptions.include = [
  {
    model: models.Region,
    as: 'orderRegion',
    attributes: ['id', 'name']
  },
  {
    model: models.OrdersSkill,
    as: 'orderSkills',
    required: false,
    include: [{
      model: models.Skill,

```

					ПІЗ.КР.Б – 121 – 25 – ПІЗ	Арк.
						76
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        as: 'skill',
        attributes: ['id', 'title']
    }]
    }
    ];

    if (skills) {
    let skillIds;
    try {
    skillIds = JSON.parse(skills);
    } catch (e) {
    return res.status(400).json({ message: 'Invalid skills format' });
    }

    if (Array.isArray(skillIds) && skillIds.length > 0 &&
!skillIds.some(isNaN)) {
    whereConditions.id = {
    [Op.in]: Sequelize.literal(`(
    SELECT DISTINCT order_id
    FROM order_skill
    WHERE skill_id IN (${skillIds.join(',')})
    )`)
    };
    } else {
    return res.status(400).json({ message: 'Invalid skills format' });
    }
    }

    const { count, rows: orders } = await
models.Order.findAndCountAll(queryOptions);

    res.status(200).json({
    total: count,
    page,
    limit,
    totalPages: Math.ceil(count / limit),
    orders
    });
    } catch (error) {
    console.error('Error in orders search:', error);
    res.status(400).json({ message: error.message });
    }
    });

module.exports = router;

```