МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

____ _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-професійної програми «Інформаційні технології проектування»

на тему: Вебплатформа підтримки діяльності фрілансерів

Здобувача групи ІТ-03-2 Борьби Вадима Ігоровича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Вадим БОРЬБА
(підпис)

Керівник к.т.н., доцент, Анна НЕНЯ _____
(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ

В.о. зав. кафедри ІТ

_ Ващенко С.М.

«_____» _ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА СТУДЕНТУ

Борьбі Вадиму Ігоровичу

1 Тема роботи Вебплатформа підтримки діяльності фрілансерів

керівник роботи *Неня Анна Вікторівна, к.т.н., доцент*

затверджені наказом по університету від «*07*» 05 2024 р. № 0482_VI

2 Строк подання студентом роботи « 26 » травня 2024 р.

3 Вхідні дані до роботи технічне завдання

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) аналіз предметної області, моделювання вебплатформи, розробка вебплатформи підтримки діяльності фрілансерів

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Актуальність, постановка задачі, аналіз сайтів-аналогів, порівняльна таблиця аналогів, функціональні вимоги, структура сторінки, діаграма бізнес-процесів в нотації IDEF0. Концептуальний рівень , декомпозиція діаграми, діаграма варіантів використання, модель бази даних, засоби реалізації, демонстрація результату, висновки

6. Консультанти розділів роботи:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7.Дата видачі завдання 8 березня 2024

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Дослідження предметної області	11.04.24 – 15.04.24	
2	Розробка технічного завдання	16.04.24 – 17.04.24	
3	Планування робіт проєкту	18.04.24 – 21.04.24	
4	Огляд останніх досліджень	22.04.24 – 25.04.24	
5	Аналіз існуючих вебплатформ	26.04.24 – 30.04.24	
6	Постановка задачі	01.05.24 – 02.05.24	
7	Вибір засобів реалізації	03.05.24 – 10.05.24	
8	Структурно-функціональне моделювання	11.05.24 – 14.05.24	
9	Розробка вебплатформи	15.05.24 – 29.05.24	
10	Оформлення документації	30.05.24 – 02.06.24	

Студент _____ Борьба В.І.
(підпис)

Керівник роботи _____ к.т.н., доц. Неня А.В.

(підпис)

АНОТАЦІЯ

Записка: 95 сторінок, 58 рисунків , 4 додатки, 25 використаних джерел та літератури.

Обґрунтування актуальності теми роботи: Сучасний ринок праці відкриває все більше можливостей для фрілансерів у сфері ІТ та інших галузях. Це обумовлено не лише зростанням популярності роботи на відстані, але й потребою в ефективному управлінні робочими процесами та оптимізації робочого часу. Розробка вебплатформи підтримки діяльності фрілансерів стане важливим інструментом для полегшення їхнього робочого процесу, підвищення продуктивності та конкурентоспроможності на ринку праці.

Мета роботи: розробка вебплатформи підтримки діяльності фрілансерів у сфері ІТ та інших галузях.

Методи дослідження: аналіз та впровадження сучасних вебтехнологій, вивчення та впровадження кращих практик управління робочим процесом.

Результат: Розроблена система призначена для підтримки діяльності фрілансерів, що забезпечує їхню ефективність, оптимізацію робочих процесів, автоматизацію завдань та полегшує взаємодію між користувачами. Використання цієї системи призведе до підвищення продуктивності фрілансерів, кращого управління робочим часом та більш ефективної організації робочих завдань.

Ключові слова: веб-платформа, фріланс, ІТ, управління робочим процесом, продуктивність, ефективність.

ЗМІСТ

ВСТУП

Актуальність. У сучасному світі, організації, які орієнтовані на бізнес-середовище, вкрай важливо забезпечити свою присутність в Інтернеті для просування власного бренду. У випадку з бізнесом, що передбачає взаємодію між клієнтами в Інтернеті, ретельно розроблені вебплатформи забезпечують досягнення подвійних переваг: присутність в Інтернеті, а також надання клієнтам можливості зручно купувати або обмінюватися послугами в Інтернеті. Саме тут відчувається потреба в ефективній вебплатформі.

Ефективна та надійна, інтернет-арена може виявитися серйозним викликом для підприємців у просуванні своїх послуг та бізнесу в Інтернеті. Оскільки впровадження вебплатформ є одним з найважливіших компонентів, необхідних для успіху електронної комерції, немає жодних сумнівів у тому, чому розробка вебплатформ вважається критично важливою. Зрештою, в сучасному бізнес-середовищі ніхто не дізнається про продукти або послуги без вебсайту або вебплатформи.

Впровадження вебплатформи є важливою складовою сучасних технологій, оскільки дозволяє компаніям та організаціям поширювати свої послуги та інформацію через Інтернет, створюючи тим самим новий потенціал для зростання, співпраці та інновацій.

Однією з найважливіших переваг впровадження вебплатформ є те, що вона дозволяє організаціям отримати доступ до більшої аудиторії. Оскільки Інтернет сьогодні використовується дуже широко, вебплатформи можуть бути доступні з будь-якої точки світу, що дозволяє компаніям охоплювати клієнтів за межами їхнього безпосереднього розташування. Це особливо вигідно для малих фірм або стартапів, які не мають можливості розвивати фізичну присутність у різних регіонах. Підприємства можуть розширити свій потенціал заробітку, створивши онлайн-додатки або сервіси, доступні для користувачів з усього світу.

Ще однією перевагою впровадження вебплатформ є те, що вони можуть покращити командну співпрацю та комунікацію. За допомогою вебплатформ члени команди можуть отримувати доступ до одних і тих самих проектів і працювати над ними з різних місць, що полегшує спілкування та обмін ідеями. Це особливо зручно для компаній з віддаленими командами або людей, які працюють вдома. Використовуючи онлайн-додатки, компанії можуть гарантувати, що всі члени команди мають доступ до однієї і тієї ж інформації та можуть безперешкодно співпрацювати незалежно від їхнього місцезнаходження.

Вебплатформи також надають користувачам різні переваги. Вебплатформи, як правило, прості в отриманні та використанні, потребують лише підключення до Інтернету та веб-браузера. Як наслідок, вони ідеально підходять для користувачів, які перебувають у дорозі або не мають доступу до десктопного програмного забезпечення. Вебплатформи також доступні з різних пристроїв, включаючи смартфони та планшети, що дозволяє користувачам залишатися на зв’язку та працювати продуктивно незалежно від місця розташування.

Вебплатформи також дешевші у використанні, ніж традиційні десктопні програми, оскільки вони потребують менше апаратного та програмного забезпечення. Це особливо корисно для невеликих організацій і приватних осіб, які можуть не мати ресурсів, щоб інвестувати в дороге програмне та апаратне забезпечення. Вебплатформи також часто легше підтримувати та оновлювати, оскільки їх можна оновлювати централізовано, а не на пристрої кожного користувача.

Впровадження вебплатформ − це життєво важливий компонент сучасних технологій, який пропонує численні переваги як для організацій, так і для клієнтів. Вебплатформи можуть допомогти організаціям розширюватися і процвітати в сучасній конкурентній економіці, дозволяючи їм отримати доступ до ширшої аудиторії, кращої співпраці та комунікації, отримати важливу інформацію про поведінку клієнтів і багато іншого.

Мета. Розробка вебплатформи підтримки діяльності фрілансерів, що дозволить останнім знаходити проєкти для виконання, а також дозволить замовникам знаходити кваліфікованих спеціалістів для власних проєктів.

Декомпозиція мети наступна:

- проведення детального аналізу ринку фрілансу та огляд останніх публікацій в даній сфері;
- визначення критеріїв функціоналу та аналіз існуючих аналогів веб-додатку;
- розробка алгоритму підтримки зв’язку між фрілансером та замовником;
- визначення стеку технології та сховища даних для розробки веб-додатку.
- виконання структурно-функціонального моделювання з декомпозицією процесу;
- реалізація клієнтської та серверної частини вебплатформи.

Об’єкт дослідження. Процес пошуку та відслідковування процесу виконання та звітування прийнятих в роботу фріланс-проектів з урахуванням специфіки взаємодії фрілансерів та клієнтів всередині та поза межами платформи.

Предмет дослідження: Вебплатформа для замовників та фрілансерів, яка дозволяє розміщувати, знаходити та звітування про виконання проєкту.

Практична цінність: Розробка вебплатформи для фрілансерів на базі стеку технологій MERN допоможе ефективно вирішувати завдання по пошуку проєктів та виконавців. Використання даного стеку дозволить створити масштабовану та гнучку систему, яка буде відповідати потребам користувачів даної системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд останніх досліджень і публікацій

Зі зростанням стартапів та бізнесу зростає потреба у гнучких та надійних талантах. Знайти потрібних людей для розширення команди може бути справжнім викликом, відкладаючи в сторону бюджет, необхідний для їхнього залучення до роботи. Робота з фрілансерами зводить накладні витрати до мінімуму і створює можливість для організацій працювати з багатьма талановитими людьми над конкретними проектами. Завдяки дистанційній роботі та цифровому кочівництву стало можливим отримати доступ до багатьох талановитих людей з усього світу. Але як знайти фрілансерів з саме тим набором навичок, який вам потрібен?

Для фрілансерів пошук клієнтів може бути не менш складним завданням. Деякі створюють веб-сайти, налаштовують свої профілі в соціальних мережах і починають просувати себе. У цьому підході є багато недоліків, оскільки існує ймовірність, що він може взагалі не спрацювати [1].

Використовуючи онлайн-платформи для фрілансерів, такі як Upwork, Freelancer, Win A Talent та Fiverr, фрілансери та компанії, розташовані в різних куточках світу, можуть налагодити зв’язок та співпрацювати один з одним. Деякі з цих платформ є більш галузевими, наприклад, 99designs для дизайнерів та ілюстраторів.

В публікації Andri Sunardi «MVC Architecture: A Comparative Study Between Laravel Framework and Slim Framework in Freelancer Project Monitoring System Web Based» описується доцільність розробки системи моніторингу та управління проектами фрілансерів [2]. Фрілансери затребувані на ринку праці через їх гнучкості і здатності працювати без прив’язки до конкретного роботодавця або місця роботи. Однак відсутність

системи моніторингу та управління часто викликає проблеми з управлінням завданнями та проектами.

На думку автора, використання PHP Laravel та Slim framework для розробки вебплатформ для фрілансерів на основі архітектури MVC відповідає цим потребам. MVC (Model-View-Controller) дозволяє перетворити вашу програму на модель (дані), подання (інтерфейс) та контролер (процес). За допомогою Laravel та Slim framework можна швидко та ефективно розробляти веб-програми різного рівня складності.

Крім того, автор звернув увагу на порівняльний аналіз продуктивності за допомогою Apache JMeter, за допомогою якого можна знайти відповідну платформу (Laravel або Slim) для цієї веб-програми. Результати показують важливість тонкої структури для швидкості та ефективності, що може бути важливим фактором при виборі методу для розробки вебплатформ для фрілансерів.

У статті Rabeya Sultana «Do IT freelancers increase their entrepreneurial behavior and performance by using IT self-efficacy and social capital?» представлено дослідження, яке використовує теорію підприємництва та перспективу соціального капіталу для покращення розуміння IT-фрілансу [3]. У дослідженні стверджується, що на підприємницьку поведінку IT-фрілансерів впливає як IT-самодостатність, так і соціальний капітал, отриманий через соціальні мережі, що, в свою чергу, впливає на їхню продуктивність. Результати дослідження свідчать про те, що IT-фрілансери, які володіють як IT-самодостатністю, так і соціальним капіталом, є більш продуктивними, ніж ті, кому бракує жодного з цих факторів.

Автори дослідження зібрали первинні дані з oDesk.com (зараз Upwork.com), одного з провідних онлайн-майданчиків для фрілансерів. Емпіричні результати підтверджують основну тезу дослідження, що має вирішальне значення для розробки вебплатформ для фрілансерів.

Результати переконливо свідчать про те, що фрілансери можуть значно підвищити свою продуктивність, використовуючи такі підприємницькі

характеристики, як креативність, працьовитість та схильність до ризику. У заяві підкреслюється важливість соціального капіталу, який можна отримати через соціальні мережі, для розвитку підприємницької діяльності.

Стаття Hsieh, Jung Kuei, Hsieh, Yi Ching «Appealing to Internet-based freelance developers in smartphone application marketplaces» демонструє важливість залучення та утримання фрілансерів для власників платформ мобільних додатків, з особливим акцентом на власників смартфонів [4]. Дослідження має на меті ретельно вивчити умови, необхідні для того, щоб позаштатні розробники могли підтримувати довгострокові відносини з власником платформи, а також визначити ключові фактори, що сприяють цьому. Це дослідження впевнено вивчає фактори, які впливають на прихильність до обчислень, включаючи грошову винагороду, потенційний ринковий попит та витрати на перехід. Крім того, воно розглядає вплив афективної прихильності на якість послуг, ідентифікацію з власником платформи та внутрішню мотивацію. Дослідження також використовує дипломатичний підхід, визнаючи складність цих факторів і можливість існування різних точок зору.

Основний висновок дослідження полягає в тому, що власники платформ мобільних додатків повинні залучати та утримувати позаштатних розробників, оскільки вони відіграють вирішальну роль у розробці та підтримці додатків для цих платформ. Власникам платформ вкрай важливо визнати важливість фрілансерів і надавати їм необхідну підтримку для забезпечення подальшого успіху своєї платформи. З дослідження можна отримати інформацію, що позаштатні розробники підтримують стосунки з власником платформи через обчислювальну та емоційну прихильність. Такі фактори, як грошова винагорода та якість послуг, мають значний вплив на ці зобов’язання [5].

1.2 Переваги та недоліки вебплатформ для фрілансу

Огляд останніх досліджень та публікацій свідчить про значний попит у фріланс-секторі на інструменти для оптимізації взаємодії між фрілансерами та їхніми клієнтами. Зокрема, існують операційні процеси, які потребують автоматизації та спрощення для покращення робочих процесів та підвищення ефективності взаємодії між сторонами.

Враховуючи вищезазначене, актуальною є розробка вебплатформи для фрілансерів, яка оптимізує взаємодію між клієнтами та продавцями. Основна мета цієї платформи - надати фрілансерам зручний та ефективний інструмент для спілкування з клієнтами, управління замовленнями та швидкого і зручного доступу до інформації про товари та послуги [6].

Перелік переваг та недоліків впровадження вебплатформ в роботу фрілансерів представлено у таблиці 1.1

Таблиця 1.1 – Переваги та недоліки вебплатформ для фрілансу

Переваги	Недоліки
Доступність: сучасні додатки доступні з будь-якого пристрою, це надає гнучкість та зручність	Залежність від Інтернету: постійна потреба в підключенні до мережі інтернет
Вільний графік: можливість працювати будь-де та організовувати вільний графік відповідно до власного розпорядку дня	Проблеми з безпекою: Недостатня захищеність даних може призвести до кіберзагроз та порушень безпеки даних
Вільний вибір проєктів та клієнтів: пошук роботи відповідно до власних навичок та потреб	Висока конкуренція: через високу конкуренцію фрілансерів може ускладнитися процес

	знаходження та утримання клієнтів

Продовження таблиці 1.1

Переваги	Недоліки
Зручність тайм-менеджменту: деякі Вебплатформи надають інструменти для аналізу витрачено часу та витрат, що полегшує керування фінансами та плануванням робочих годин	Нестабільність доходів: через відсутність фіксованих цін та проєктів, можливий нестабільний дохід
Різноманітність: можливість власного розвитку та навичок через роботу з різними типами проєктів	Відсутність гарантованого найму: витрачення часу та енергії на пошук клієнтів та проєктів
Глобальний ринок клієнтів: попит клієнтів з усього світу що розширює потенційні можливості фрілансера	Відносна токсичність: більшість клієнтів вимагають швидке виконання роботи та не зважають на особисті справи фрілансера.

Джерело: розроблено автором

1.3 Аналіз програмних продуктів – аналогів

Аутсорсинг завдань фрілансерам − це високоефективна та дієва практика, яка широко застосовується в сучасних бізнес-стратегіях. Вона дозволяє компаніям співпрацювати з найкращими талантами, яких може не вистачати в штаті, включаючи розробку програмного забезпечення, ведення блогів та виконання життєво важливих адміністративних завдань.

Співпраця з незалежними професіоналами на проектній основі може значно скоротити витрати і час, а також принести свіжі та унікальні ідеї. Такий підхід свідчить про впевненість компанії у своїй здатності знаходити та залучати найкращі таланти, а також є чітким свідченням прагнення компанії до досконалості. Співпраця з фрілансерами може бути важливим кроком, особливо якщо перший досвід роботи з зовнішніми працівниками. Однак, зі зростанням економіки фрілансу, технології призвели до появи фріланс-маркетплейсів. Завдяки фріланс-платформам клієнт з конкретною задачею може зв'язатися з потрібними талантами, які можуть виконати дану роботу за фіксовану суму [7].

Для аналізу було обрано три фріланс-платформи: Fiverr, Upwork та Freelancer.com.

Fiverr − це професійна фріланс-платформа, яка пропонує широкий спектр швидких і доступних послуг для бізнесу та індивідуальних роботодавців де клієнти можуть купувати проєкти та здійснювати платежі на основі погодинній оплаті фрілансера. Важливо зазначити, що всі послуги, пропоновані на Fiverr, є легальними і відповідають умовам надання послуг платформи. Платформа надає широкий спектр навичок, від копірайтингу та дизайну WordPress до відеомонтажу та маркетингових кампаній у соціальних мережах.

Фрілансери пропонують послуги всього за $5 за проект, що робить його доступним варіантом для власників малого бізнесу. Важливо зазначити, що ціна виконання може змінюватися в залежності від якості роботи та термінів виконання.

Крім того, платформа має власну систему рангу фрілансерів на основі об'єктивних критеріїв, таких як рівень задоволеності клієнтів, час виконання та якість послуг. Платформа надає клієнтам вичерпну інформацію про фрілансера, включаючи його характеристики та рейтинги від попередніх роботодавців. Це дає змогу потенційним покупцям прийняти обґрунтоване рішення щодо найму фрілансера.

Fiverr дає можливість покупцям впевнено спілкуватися з продавцями за допомогою захищеного обміну повідомленнями та документами, забезпечуючи легкий обмін всією інформацією про проект та відповідними деталями. Ця функція сприяє створенню атмосфери співпраці та професіоналізму для успішного завершення проєкту.

На рисунку 1.1 представлено вигляд головної сторінки платформи Fiverr [8].

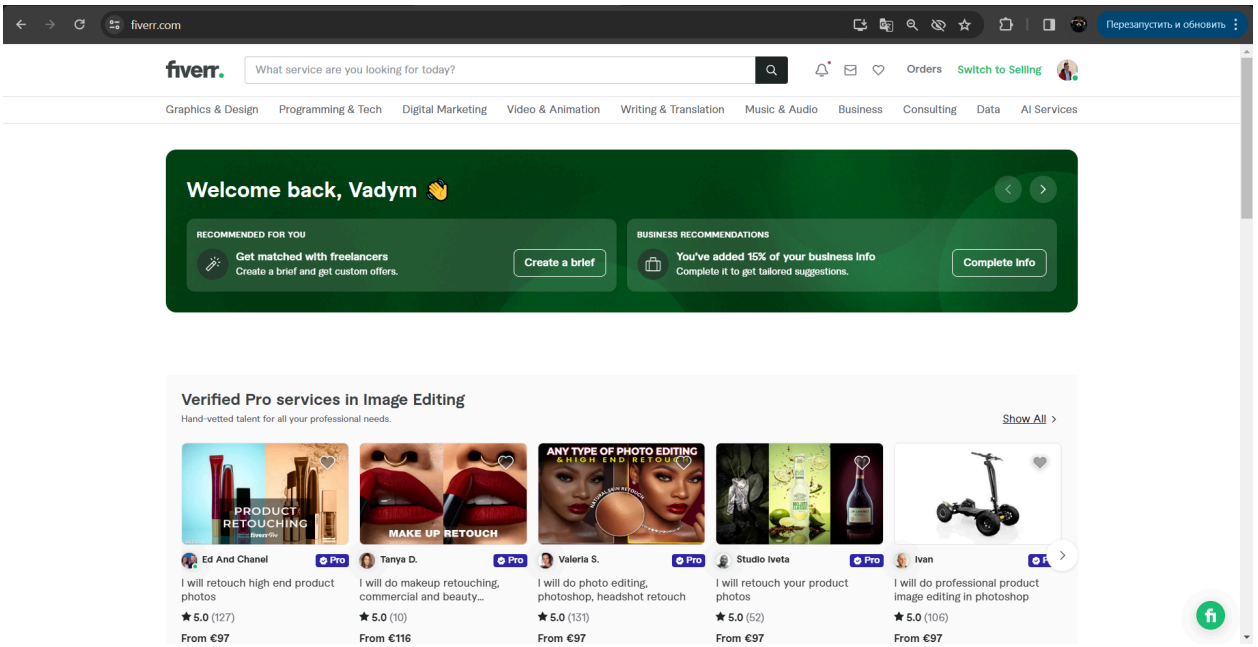

Рисунок 1.1 – Головна сторінка платформи Fiverr

Джерело: [8]

Upwork − провідна онлайн-платформа для фрілансерів з великою кількістю висококваліфікованих фрілансерів та роботодавців [9]. Платформа містить більш ніж 12 мільйонами фрілансерів і 5 мільйонами роботодавців, і щорічно публікує близько 3 мільйонів вакансій. Фрілансери на Upwork пропонують широкий спектр навичок, включаючи копірайтинг, графічний дизайн, веб-розробку та розробку програмного забезпечення.

Клієнти можуть легко розмістити на платформі свої вимоги до роботи та бюджет і бути впевненими, що знайдуть потрібного фрілансера для свого проекту. На Upwork фрілансери можуть впевнено переглядати доступні проекти та подавати пропозиції, які відповідають їхньому портфоліо чи навичкам. Проекти на Upwork можна розбивати на менші завдання, відповідно до узгодженого бюджету, що дозволяє клієнтам мати більший контроль над проектом. Унікальна платіжна система Upwork пропонує гнучкість, дозволяючи здійснювати платежі за проект або за кожен етап.

Крім того, Upwork забезпечує перевірку особистості фрілансерів, надаючи клієнтам спокій і відчуття безпеки. Крім того, Upwork регулярно оцінює фрілансерів кожні два тижні на основі результатів їхньої роботи. Це гарантує, що клієнти отримують актуальні оцінки успішності роботи фрілансерів (зі 100%) та попередні відгуки клієнтів, які можуть допомогти їм у виборі найбільш підходящого фрілансера.

Також платформа пропонує ефективні канали комунікації, які дозволяють клієнтам зв’язуватися з фрілансерами. Це дозволяє клієнтам проводити співбесіди, надавати інформацію про проект або моніторити проекти за допомогою вбудованих інструментів чату та відеоконференцій. І Upwork, і Fiverr є добре відомими платформами для фрілансерів і замовників для зв’язку та спільної роботи над проектами. У той час як Fiverr пропонує безліч функцій, Upwork виділяється як сильний вибір для завершення проектів завдяки своєму широкому спектру інструментів і ресурсів, включаючи можливість легко ділитися документами і файлами. Хоча Fiverr пропонує безліч функцій, Upwork виділяється як сильний вибір для завершення проектів завдяки своєму широкому спектру інструментів і ресурсів, включаючи можливість легко ділитися документами і файлами. З Upwork і клієнти, і фрілансери можуть бути впевнені у своїй здатності успішно завершити будь-який проект.

На рисунку 1.2 представлено вигляд головної сторінки платформи Upwork [9].

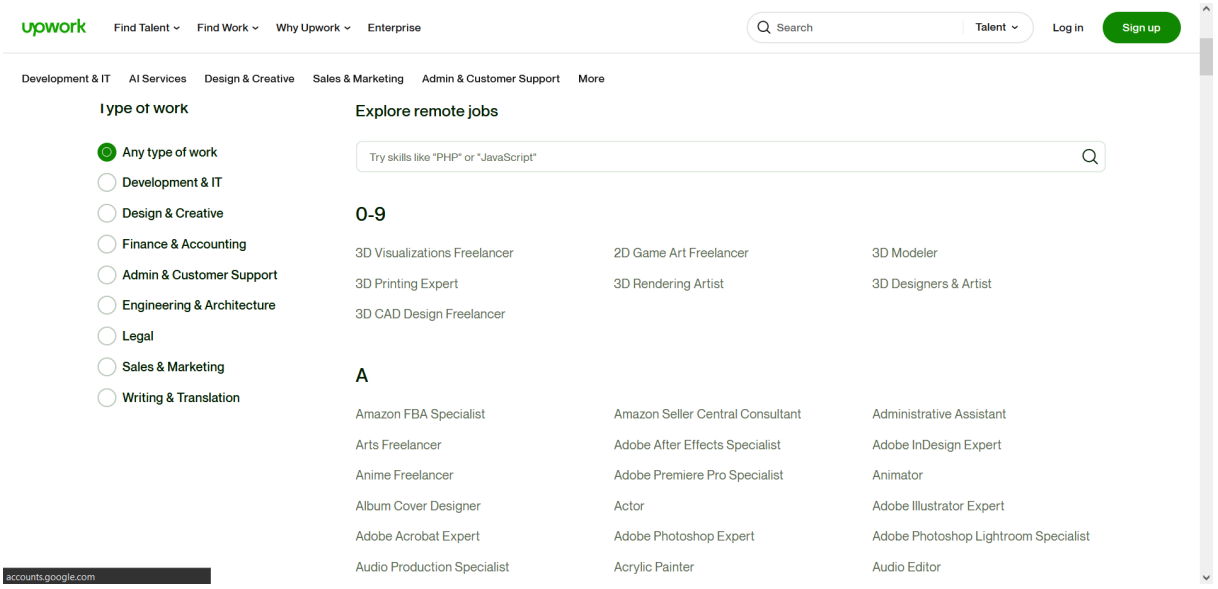

Рисунок 1.2 – Головна сторінка платформи Upwork

Джерело: [9]

Freelancer.com – провідна платформа, яка з'єднує компанії з незалежними професіоналами по всьому світу [10]. Маючи понад 25 мільйонів зареєстрованих користувачів і майже 12 мільйонів завершених проектів, це одна з 3 найкращих фріланс-платформ у світі. Freelancer.com пропонує свої послуги мільйонам клієнтів і фрілансерів у більш ніж 247 країнах, демонструючи свій досвід і авторитет у галузі.

Freelancer.com вирізняється з-поміж інших платформ тим, що дозволяє здійснювати платежі в різних місцевих валютах. Подібно до Upwork, клієнти можуть розміщувати вакансії та отримувати пропозиції від фрілансерів. Потім клієнти можуть переглянути портфоліо фрілансерів, щоб вибрати найкращого для свого проекту.

Також Freelancer.com надає клієнтам і фрілансерам можливість ефективно спілкуватися в чаті. Freelancer.com пропонує корисну функцію, яка дозволяє клієнтам відстежувати прогрес проекту на етапі співбесіди. Обидві сторони можуть спілкуватися в чаті, співпрацювати та обмінюватися файлами і документами, перебуваючи в дорозі. Freelancer.com впровадив комплексну

систему оцінювання, яка включає різні рівні, бейджі та складну систему XP для оцінки фрілансерів.

Крім того, клієнти не зобов'язані платити передоплату за проекти на Freelancer.com. Платежі будуть перераховані фрілансеру лише після задовільного завершення роботи. Крім того, платформа пропонує клієнтам можливість розбивати великі проекти на менші, більш керовані завдання і доручати їх одному або декільком фрілансерам.

На рисунку 1.3 представлено вигляд головної сторінки платформи Freelancer.com [10].

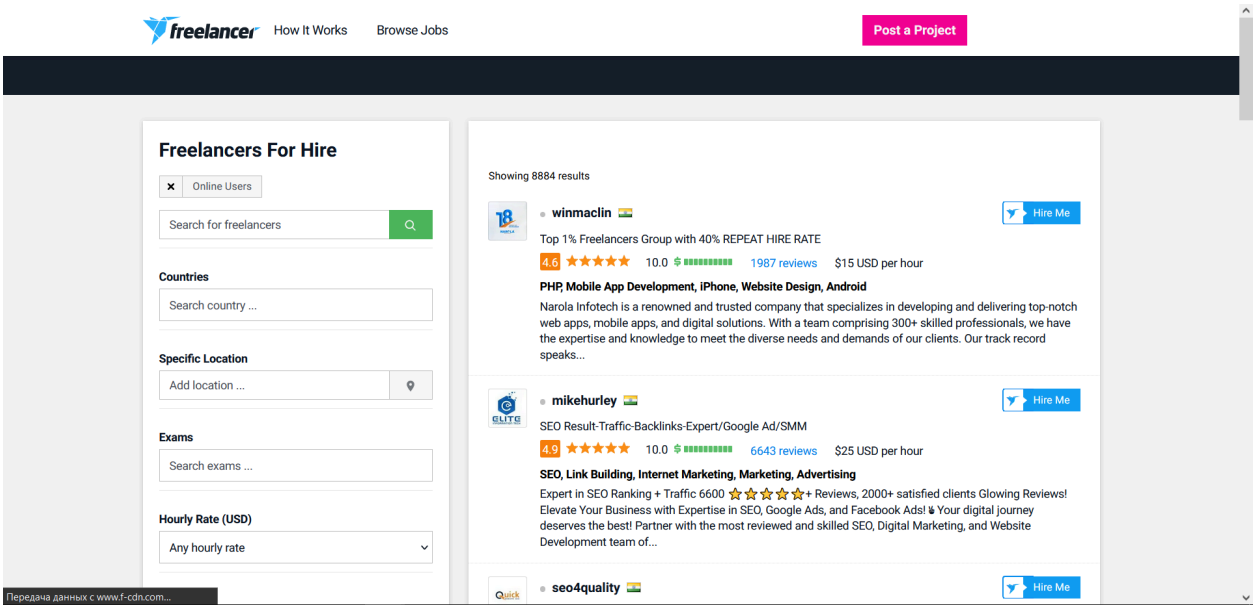

Рисунок 1.3 – Головна сторінка платформи Freelancer.com

Джерело: [10]

На сьогоднішній день існує безліч додатків-платформ які відносяться до даної тематики. Тому кожен додаток має власні переваги та недоліки. Для більш поглибленого аналізу необхідно визначити функціональні елементи які найчастіше присутні в вебдодатках. Саме для цього необхідно створити порівняльну таблицю. Результати представлені в таблиці 1.2.

Таблиця 1.2 – Порівняльна характеристика аналогів вебплатформ.

Особливості	Fiverr	Upwork	Freelancer.com	Work Wonders
Тип платформи	Мікросервісна, з фокусом на послуги	Велика ринкова платформа для проектів	Ринкова платформа для проектів	
Послуги/проекти	Графічний дизайн, копірайтинг, інше	Веб-розробка, маркетинг, адміністрування, інше	Веб-розробка, дизайн, письменництво, інше	Веб-розробка, Digital-маркетинг, UX/UI дизайн, інше
Вартість	Зазвичай низькі вартості послуг	Залежить від типу та об’єму проекту	Залежить від типу та об’єму проекту	Залежить від типу та об’єму проекту
Платформа оплати	Зазвичай PayPal або Fiverr Pay	PayPal, Payoneer, банківські перекази	PayPal, Skrill, банківські перекази	Банківські перекази
Гарантії	Грошові гарантії та політика повернення	Грошові гарантії та поетапна оплата	Грошові гарантії та поетапна оплата	Грошові гарантії та політика повернення
Рівень конкуренції	Високий, особливо для початківців	Високий, особливо у популярних категоріях	Високий, особливо у популярних категоріях	Високий, особливо у популярних категоріях
Фільтрація	Проста фільтрація за категоріями, рейтингом тощо	Детальна фільтрація за навичками, досвідом тощо	Розширена фільтрація за навичками, рейтингом тощо	Пошукова система
Вбудований чат	Модульний онлайн-чат	Повноцінний онлайн-чат	Модульний онлайн-чат	Повноцінний онлайн-чат

Джерело: розроблено автором

1.4 Обґрунтування актуальності розробки

Розробка вебплатформи для фрілансерів є дуже актуальною через необхідність персоналізованого підходу до клієнтів та ефективного

обслуговування. Такий підхід не лише підвищить задоволеність клієнтів, але й покращить загальну якість послуг, що надаються. За допомогою такого додатку фрілансери можуть адаптувати свою роботу до індивідуальних потреб клієнта, враховуючи попередні відвідування платформи, покупки та інші дії.

Однією з переваг використання вебплатформ для фрілансерів є зниження витрат на обслуговування та експлуатацію порівняно з традиційними бізнес-моделями. Це пов’язано з обмеженою потребою в матеріально-технічних ресурсах і персоналі. Такий підхід дозволяє фрілансерам економити ресурси і зосередитися на наданні якісних послуг клієнтам, підвищуючи свою конкурентоспроможність на ринку.

1.5Аналіз існуючих технологій вирішення задачі

Коли поставлена задача розробки вебплатформи слід звернути увагу на ряд популярних технологічних стеків, таких як MERN (MongoDB, ExpressJS, ReactJS, NodeJS) , MEAN (MongoDB, ExpressJS, AngularJS, NodeJS) і MEVN (MongoDB, ExpressJS, VueJS, NodeJS).

Стек MERN дуже схожий на стек MEAN, але критична відмінність полягає у використанні ReactJS замість AngularJS. Користувач взаємодіє з компонентами інтерфейсу користувача ReactJS на стороні клієнта. Back-end лежить на сервері, зазвичай за допомогою Express.js, що працює поверх Node.js [11].

Список популярних сайтів, створених за допомогою фреймворку MERN, включає Facebook, Instagram, Netflix, Airbnb тощо.

Компоненти стеку MERN

MongoDB − масштабована та гнучка крос-платформна база даних NoSQL.

Express.js − безкоштовний фреймворк вебплатформ з відкритим вихідним кодом, який працює на Node.js.

ReactJS − інтерфейсна JavaScript-бібліотека з відкритим кодом для створення користувацьких інтерфейсів на основі UI-компонентів.

Node.js − кросплатформенне середовище виконання JavaScript з відкритим вихідним кодом

Особливості стеку MERN

- єдиний скрипт кодування охоплює як бек-, так і фронтенд;
- MongoDB може працювати на різних серверах;
- весь процес стекування може відбуватися виключно на Java та JSON;
- динамічна схема, надійний графічний інтерфейс, інструменти командного рядка для прискорення;
- тестування в реальному часі за допомогою вбудованих інструментів;
- плавний рендеринг інтерфейсу.

Переваги стеку MERN

- архітектура MERN є масштабованою, оскільки дозволяє додаткам обробляти більший трафік;
- підтримує повторне використання коду;
- для створення інтерактивних інтерфейсів використовується популярний JavaScript фреймворк React.

Мінуси стеку MERN

- гнучкість MERN може призвести до непослідовних структур коду;
- підтримка та рефактор структури коду;
- він має круту криву навчання, оскільки потрібні знання декількох технологій.

MEAN – Стек технологій з відкритим вихідним кодом на основі JavaScript, що ідеально підходить для створення динамічних вебплатформ і веб-сайтів − як адаптивних, так і прогресивних. Він складається з декількох компонентів, які спрощують розробку додатків і прискорюють процес [12].

Приклади деяких з відомих веб-сайтів, побудованих на MEAN, − це IBM, The Guardian, PayPal та LinkedIn.

Компоненти стеку MEAN

Express.js − безкоштовний фреймворк вебплатформ з відкритим вихідним кодом, який працює на Node.js.

AngularJS (або Angular) − популярний JavaScript-фреймворк для розробки крос-платформних додатків.

Node.js − кросплатформенне середовище виконання JavaScript з відкритим вихідним кодом.

Можливості стеку MEAN:

- використання JavaScript для зменшення непотрібного використання пропускної здатності;
- тестування в реальному часі завдяки вбудованим інструментам;
- переваги плагінів та віджетів;
- node.js зменшує час завантаження сайту;
- можна створювати всі типи додатків ;
- має ізоморфний код: дозволяє працювати як в браузері, так і на сервері;
- дозволяє швидко переносити код на інший фреймворк зі збереженням його функцій;
- єдина мова для бекенду і фронтенду.

Переваги стеку MEAN:

- використання JavaScript у всьому стеку спрощує процес розробки;
- підтримує кросплатформені операційні системи;
- MEAN є універсальним та адаптивним.

Мінуси стека MEAN:

- JavaScript іноді може спричиняти уповільнення роботи веб-сайтів;
- погана ізоляція сервера від бізнес-логіки;
- не забезпечує такої ж функціональності, як реляційні бази даних.

MEVN − це знову ж таки веб-стек, як і MERN та MEAN, хоча він є відносно новим для його фронтенд-технології VueJS. Ось чим ці компоненти схожі та відрізняються від MERN і MEAN [13].

Компоненти стеку MEVN:

- VueJS − клієнтський фреймворк з двостороннім зв’язуванням даних, що дозволяє безперешкодно розробляти інтерфейс, MVC та інтерактивні серверні додатки;
- Node.js − кросплатформне середовище виконання JavaScript з відкритим вихідним кодом.

Особливості стеку MEVN:

- Javascript використовується на всіх рівнях розробки, від клієнтського до серверного, що спрощує процес. Це робить розробку швидшою та ефективнішою;
- незалежність від платформи;
- архітектура MVC в бекенд-версії;
- Vue.JS швидше і легше вивчати.

Переваги стеку MEVN:

- підтримує архітектуру MVC
- він наскрізь зашифрований
- MEVN простий у вивченні

Мінуси стеку MEVN:

- обмежена підтримка SEO;
- Vue.js та Node.js все ще нові у порівнянні з іншими технологіями;
- стек MEVN має відносно менше ресурсів та документації.

Для більш детального порівняння було створено таблицю аналізу 3 фреймворків – ReactJS, AngularJS та VueJS. Результати порівняння представлені в таблиці 1.3.

Таблиця 1.3 – Порівняльна таблиця фреймворків

ReactJS	AngularJS	VueJS
UI фреймворк	Повноцінний фреймворк	Користувацький фреймворк
Virtual DOM	Управління станами	MVC фреймворк
Одностороння прив’язка даних	Маршрутизація	Прив’язка даних до способу
Нативний фреймворк для мобільної розробки	Валідація та обробка форм	Немає валідації та обробки форм
Використовується мільйонами людей по всьому світу	HTTP-клієнт	Немає HTTP-клієнта

Джерело: розроблено автором

Для розробки вебплатформи підтримки діяльності фрілансерів було прийнято рішення використати стек технологій MERN.

У порівнянні з традиційними веб-додатками, додатки, розроблені за допомогою стека MERN, легше підтримувати. Архітектура MERN складається з декількох рівнів, що робить її масштабованою як для збільшення, так і для зменшення без шкоди для цілісності коду. Такий підхід не лише економить час, але й виявляється економічно вигідним рішенням.

Крім того, стек MERN набуває все більшої популярності завдяки своїй здатності поєднувати функціональність інтерфейсу, бек-енду та бази даних, що підтримується потужною спільнотою. Архітектура MERN є дуже гнучким і надійним рішенням, яке набирає популярності серед розробників [14].

1.6 Постановка завдання

Метою цього проекту є створення вебплатформи, яка підтримує фрілансерів у їхній роботі та спілкуванні з клієнтами. Платформа дозволить фрілансерам переглядати доступні завдання та проекти які були розміщені клієнтами, подавати свої пропозиції, розміщувати замовлення та взаємодіяти з адміністрацією платформи.

Платформа також надасть можливість зворотнього зв’язку клієнта з фрілансером.

Для досягнення мети проєкту необхідно виконати наступні задачі:

- визначити актуальність розробки вебплатформи для підтримки діяльності фрілансерів у сучасному ринковому середовищі;
- дослідити предметну область фріланс-платформ та специфіку їх роботи;
- провести аналіз існуючих аналогів вебплатформ для визначення їх переваг та недоліків;
- реалізувати компоненти вебплатформи на основі потреб користувачів;
- виконати структурно-функціональне моделювання для візуального розуміння процесів;
- провести тестування вебплатформи для перевірки працездатності модулів та компонентів перед релізом.

2 ПРОЕКТУВАННЯ ВЕБПЛАТФОРМИ

2.1 Структурно-функціональне моделювання

Метод IDEF0 аналізує об’єкт, досліджуючи, як він функціонує, моделюючи виявлені операційні деталі та застосовуючи методи оцінки для визначення потенційних покращень. У процесі розробки програмного забезпечення інструмент IDEF0 використовується для моделювання рішень, дій та діяльності програмної системи. Він містить необхідні нотації для підтримки розробки програмного забезпечення.

На діаграмі IDEF0 підфункції відображаються на головній діагоналі. Порядок функцій автоматично визначається, який обходить базову структуру батьківської функції. Структура обходиться зліва направо з паралельними і вибраними конструкціями, які обходяться по одній гілці за раз [15].

Вхідні дані вводяться зліва. Вони можуть надходити або з краю діаграми (зовнішні входи), або з іншої функції на діаграмі.

Керування (дані для запуску) вводяться зверху. Вони можуть надходити або з краю діаграми (зовнішні тригери), або з іншої функції на діаграмі.

Виходи виходять праворуч. Виходи можуть або з’єднуватися з іншою функцією на діаграмі, або виходити на край діаграми, або робити і те, і інше (представляючи собою вихід, який є вхідним / запускає як внутрішні, так і зовнішні функції).

Механізми (розподіл на Компоненти) вводяться внизу.

На рисунку 2.1 представлено контекстну діаграму в нотації IDEF0 для вебплатформи підтримки діяльності фрілансерів.

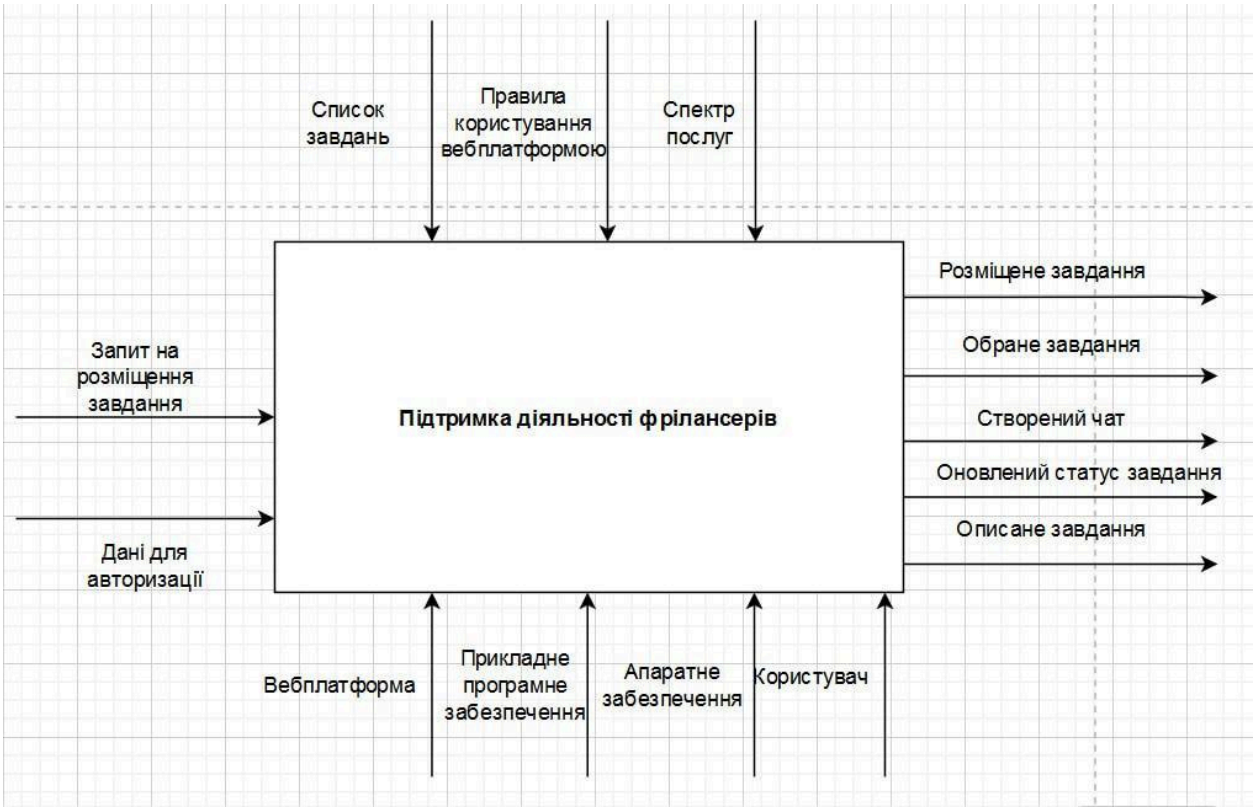

Рисунок 2.1 – Діаграма в нотації IDEF0 для вебплатформи підтримки діяльності фрілансерів

Джерело: розроблено автором

Діаграма функціональної декомпозиції − це метод моделювання, який використовується в різних сферах, зокрема в інженерії програмного забезпечення та аналізі даних фізики високих енергій. Ці діаграми допомагають розбити складні системи на менші, більш керовані компоненти. Мета полягає в тому, щоб забезпечити узгодженість і коректність поведінки під час процесу вдосконалення. Традиційні методи декомпозиції часто покладаються на ручну техніку, що може призвести до створення піддіаграм, які не є незалежними або повними. У контексті діаграм послідовності був введений новий підхід функціональної декомпозиції, що фокусується на декомпозиції глобальної системи на глобальні підсистеми, які можуть бути перекомпоновані без припущень про зв’язок [16].

Нарешті, в контексті систем реального часу ієрархічне моделювання функціональної декомпозиції підкреслює декомпозицію системи на паралельні завдання та використання потоку даних, потоку управління та діаграм переходу стану.

На рисунку 2.2 представлено декомпозицію IDEF0-діаграми для вебплатформи підтримки діяльності фрілансерів.

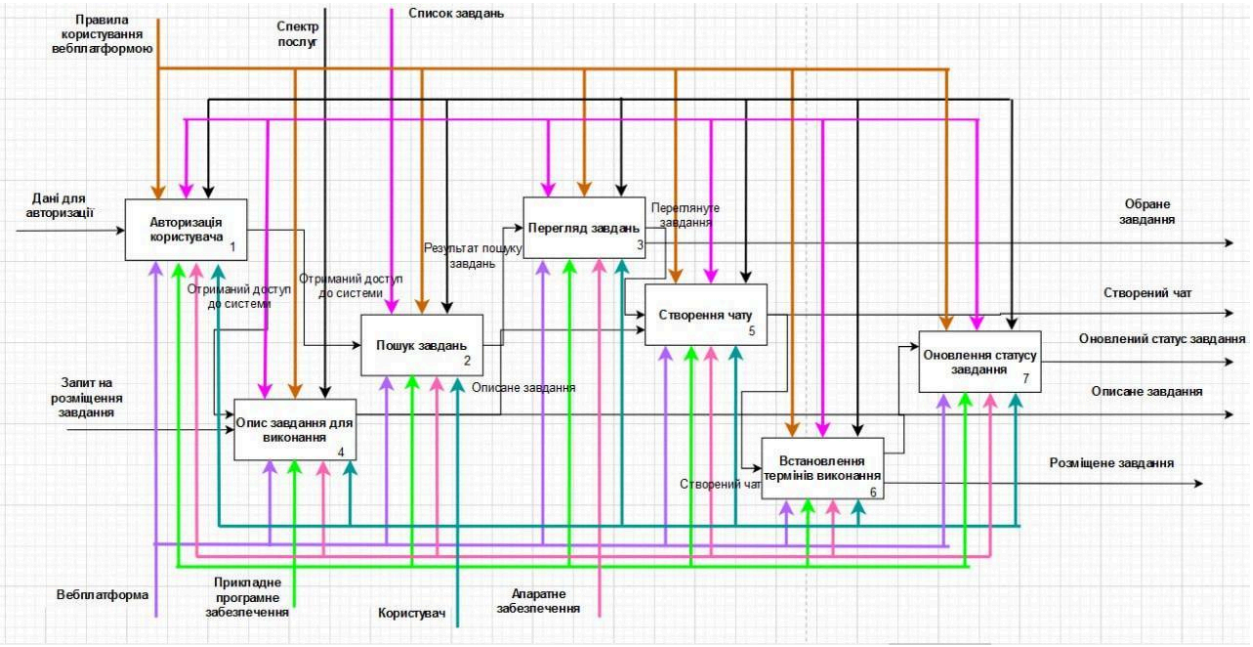

Рисунок 2.2 – Декомпозиція IDEF0-діаграми для вебплатформи підтримки діяльності фрілансерів.

Джерело: розроблено автором

2.2 Моделювання варіантів використання

Уніфікована мова моделювання (UML) − це потужна та універсальна мова, яка широко використовується у моделюванні програмної інженерії.

Вона дозволяє точно оцінювати продуктивність, ефективно відстежувати та забезпечувати надійну безпеку, а також надає чіткі та стислі інструкції для операцій. UML особливо добре підходить для передачі складної візуальної інформації про архітектуру програмного забезпечення широкій аудиторії.

Визначення пріоритетів використання інструменту планування перед початком розробки програми є дуже важливим.

UML є важливим інструментом для планування програм, оскільки він може генерувати код для налаштування UML-моделі, роблячи реалізацію програми більш ефективною та зменшуючи накладні витрати. Крім того, діаграми UML легко модифікуються, що може заощадити час і зусилля в процесі розробки. Крім того, UML полегшує налагодження, що може допомогти запобігти системним проблемам у майбутньому [17].

Діаграма варіантів використання вебплатформи підтримки діяльності фрілансерів представлена на рисунку 2.3.

2.3Проектування моделі бази даних

MongoDB − це високомасштабована та гнучка NoSQL база даних документів з відкритим вихідним кодом, яка проста у використанні та освоєнні [18].

Бази даних документів пропонують швидкі запити, структуру, що добре підходить для роботи з великими масивами даних, гнучке індексування та спрощений метод ведення бази даних. Вони ефективні для веб-додатків і були повністю інтегровані великими IT-компаніями, такими як Amazon.

Хоча бази даних SQL мають велику стабільність і вертикальну потужність, вони менш придатні для надвеликих баз даних. Для випадків використання, які вимагають негайного доступу до даних, таких як додатки

для охорони здоров'я, краще підходять документні бази даних. Бази даних документів полегшують пошук даних за допомогою тієї ж моделі документів, яка була використана при кодуванні програми [19].

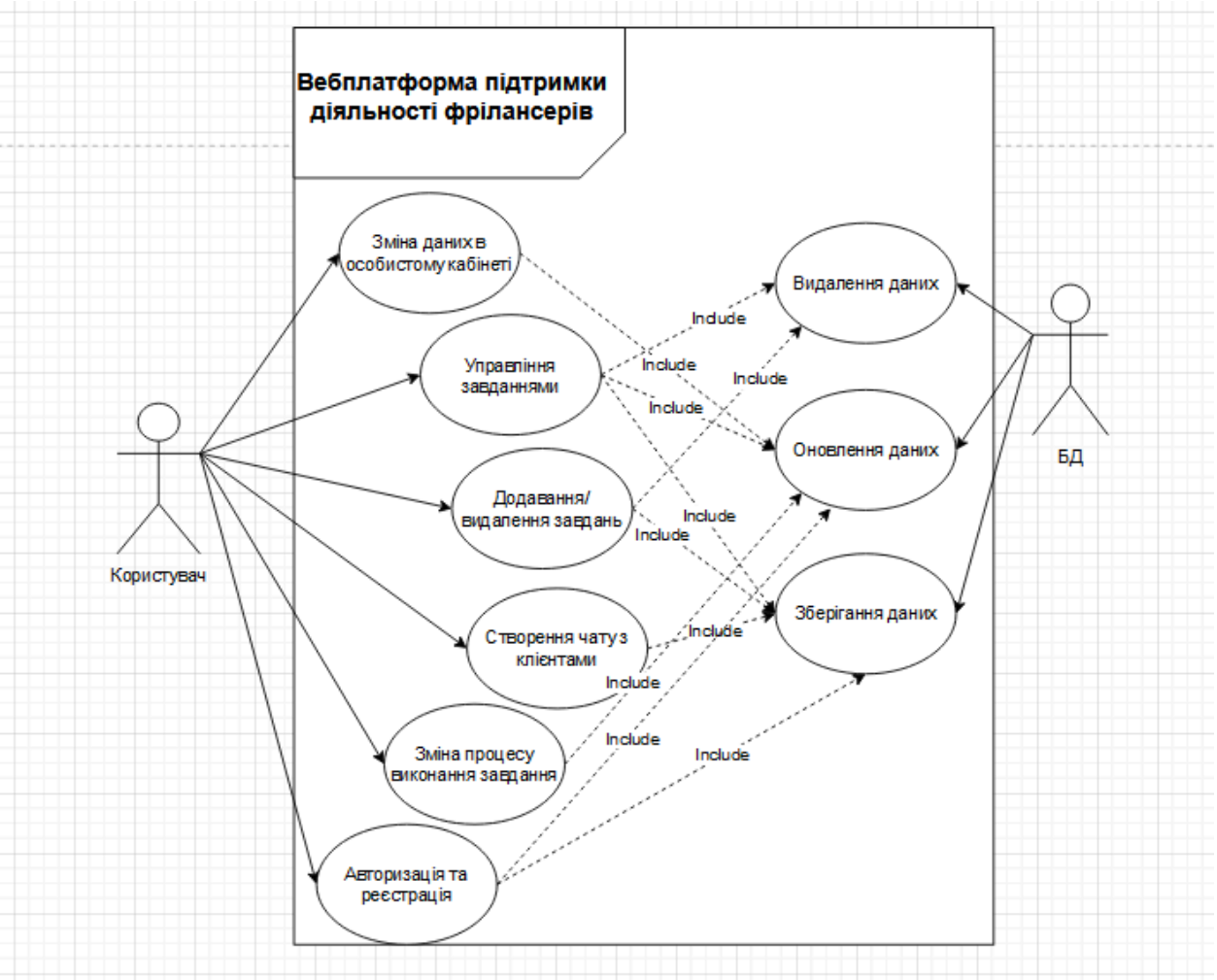

Рисунок 2.3 - Діаграма варіантів використання вебплатформи підтримки діяльності фрілансерів

Джерело: розроблено автором

Переваги

Відсутність схем. Немає обмежень у форматі та структурі зберігання даних. Це добре для збереження існуючих даних у великих обсягах і різних структурних станах, особливо в системі, що постійно трансформується.

Швидше створення та обслуговування. Після створення документа потрібно мінімальне обслуговування, яке може бути настільки простим, що достатньо лише один раз додати складний об'єкт.

Відсутність зовнішніх ключів. Завдяки відсутності динамічних зв'язків, документи можуть бути незалежними один від одного.

Відкриті формати. Чистий процес збірки, який використовує XML, JSON та інші похідні для опису документів.

Вбудоване керування версіями. Зі збільшенням розміру ваших документів може зростати і їх складність. Керування версіями зменшує кількість конфліктів.

Недоліки

Обмеження перевірки узгодженості. У наведеному вище прикладі використання бази даних книг можна було б шукати книги неіснуючого автора. Ви можете шукати в колекції книг і знаходити документи, які не пов'язані з колекцією авторів.

Кожен список може також дублювати інформацію про автора для кожної книги. Ці невідповідності не є суттєвими в деяких контекстах, але при вищих стандартах аудиту узгодженості БД вони серйозно знижують продуктивність бази даних.

Слабкі сторони атомарності. Реляційні системи також дозволяють змінювати дані з одного місця без необхідності використовувати JOIN. Всі нові запити на читання успадковують зміни, внесені до ваших даних за допомогою однієї команди (наприклад, оновлення або видалення рядка).

Для баз даних документів зміна, що стосується двох колекцій, вимагатиме виконання двох окремих запитів (для кожної колекції). Це порушує вимоги атомарності.

Безпека. Майже половина веб-додатків сьогодні активно виточують конфіденційні дані. Тому власникам баз даних NoSQL потрібно звертати особливу увагу на вразливості веб-додатків.

MongoDB використовує документно-орієнтовану модель даних і неструктуровану мову запитів. Масштабована архітектура MongoDB дозволяє розробникам створювати додатки за допомогою гнучких методів. Оскільки MongoDB є популярним вибором для створення інтернет-додатків і бізнес-додатків, компаніям важливо розуміти як переваги, так і недоліки MongoDB [20].

На рисунках 2.4 – 2.9 представлено документи збереження даних.

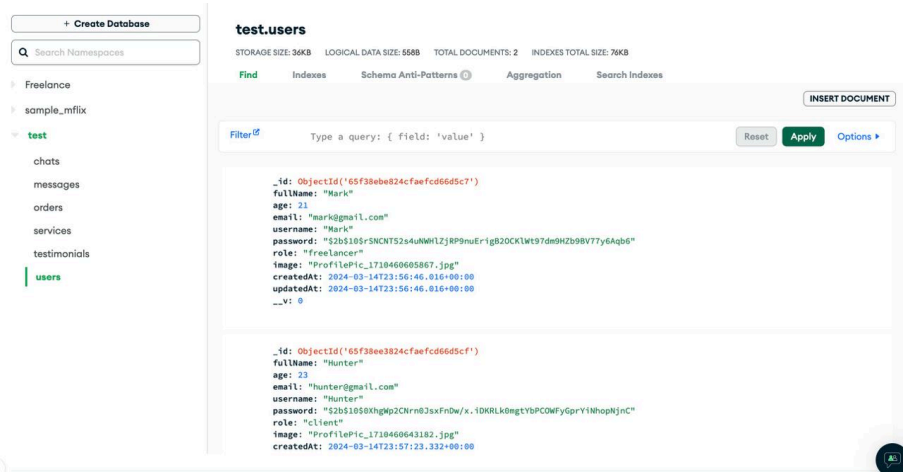

Рисунок 2.4 – Документ «users» для збереження користувачів вебплатформи

Джерело: розроблено автором

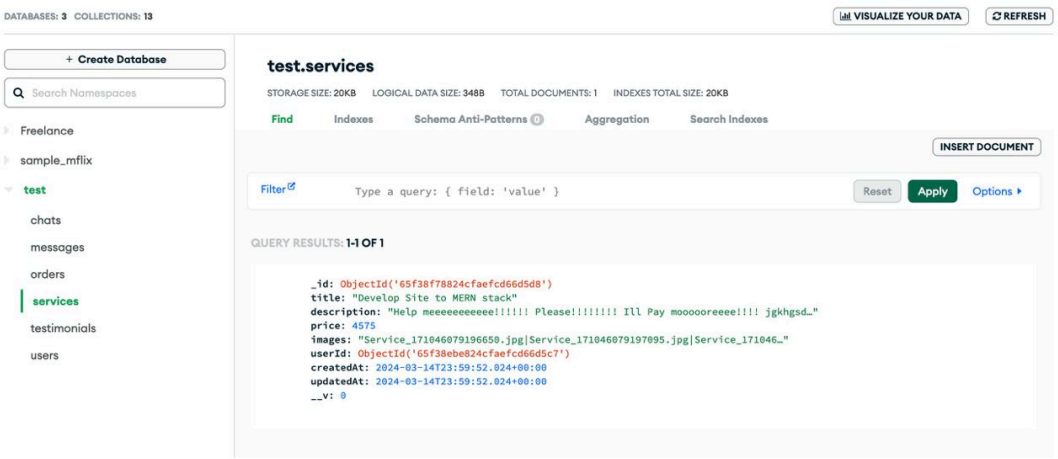

Рисунок 2.5 – Документ «services» для збереження завдань вебплатформи

Джерело: розроблено автором

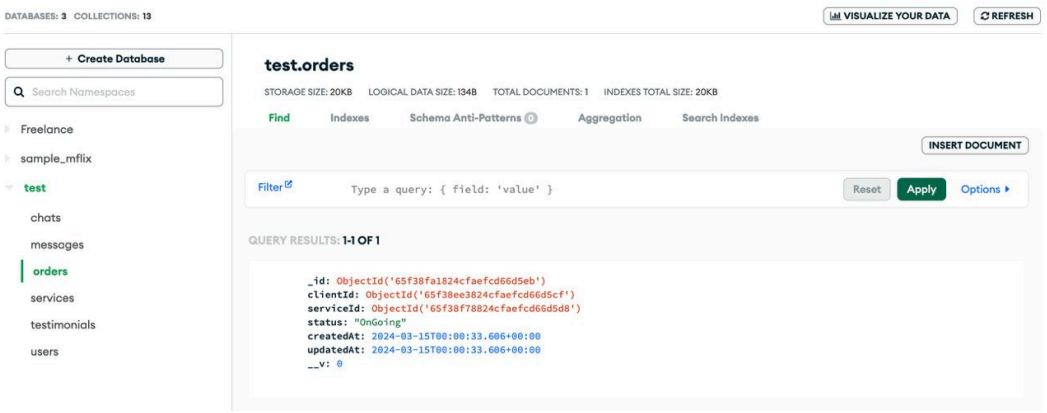

Рисунок 2.6 – Документ «orders» для збереження замовлень вебплатформи

Джерело: розроблено автором

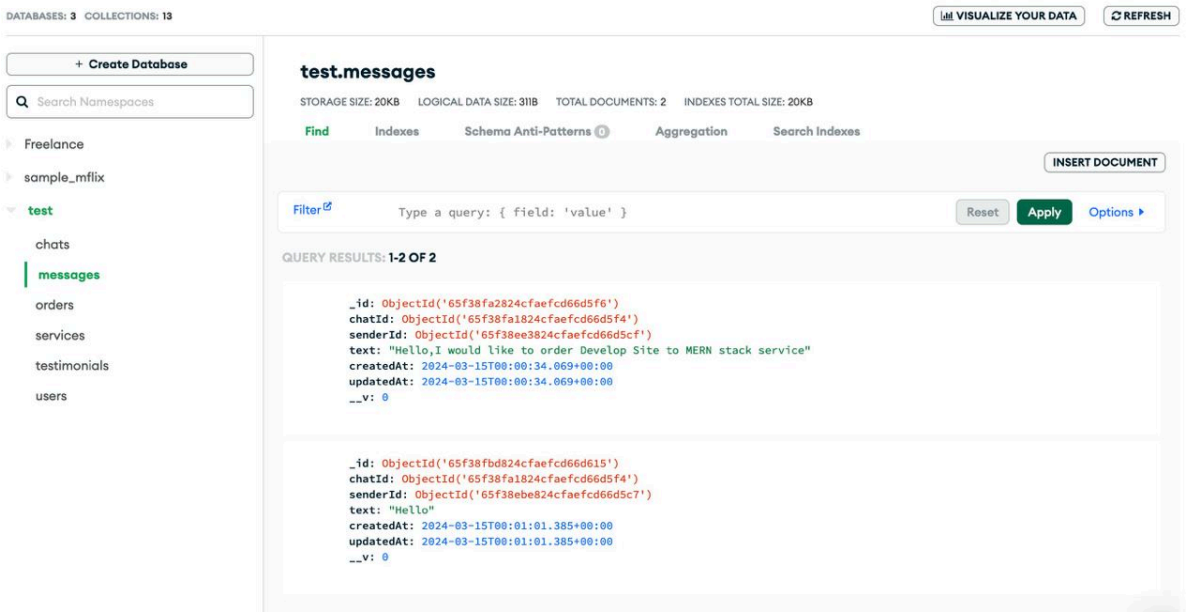

Рисунок 2.7 – Документ «messages» для збереження надісланих повідомлень вебплатформи

Джерело: розроблено автором

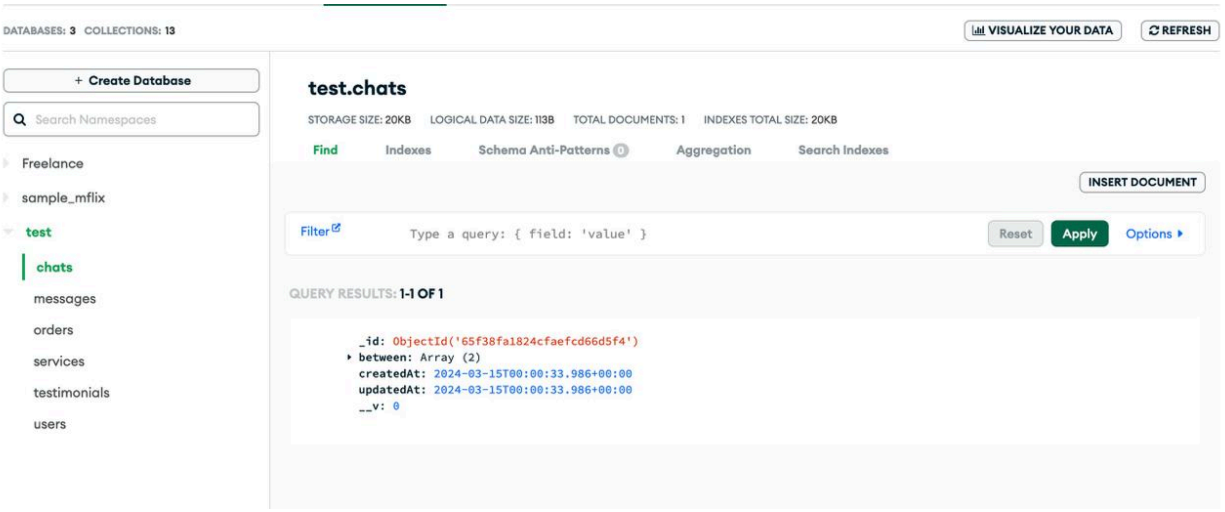

Рисунок 2.7 – Документ «chats» для збереження створених чатів вебплатформи

Джерело: розроблено автором

3. РЕАЛІЗАЦІЯ ВЕБПЛАТФОРМИ ДІЯЛЬНОСТІ ФРІЛАНСЕРІВ

3.1 Архітектура вебплатформи

Архітектура MERN дозволяє легко побудувати трирівневу архітектуру (фронтенд, бекенд, база даних) повністю за допомогою JavaScript та JSON.

Верхній рівень стеку MERN - це React.js, JavaScript-фреймворк для створення динамічних клієнтських додатків на HTML. React дозволяє створювати складні інтерфейси з простих компонентів, підключати їх до даних на внутрішньому сервері та відображати у вигляді HTML.

Наступним рівнем нижче є серверний фреймворк Express.js, який працює всередині сервера Node.js. Express.js позиціонує себе як веб-фреймворк для Node.js і це саме те, чим він є.

Express.js має потужні моделі для маршрутизації URL-адрес (зіставлення вхідної URL-адреси з функцією сервера) та обробки HTTP-запитів і відповідей.

JSON-документи, створені у фронтенді React.js, можуть бути надіслані на сервер Express.js, де вони можуть бути оброблені і збережені безпосередньо в MongoDB для подальшого пошуку.

На рисунку 3.1 представлено вигляд архітектури вебплатформи для підтримки фрілансерів.

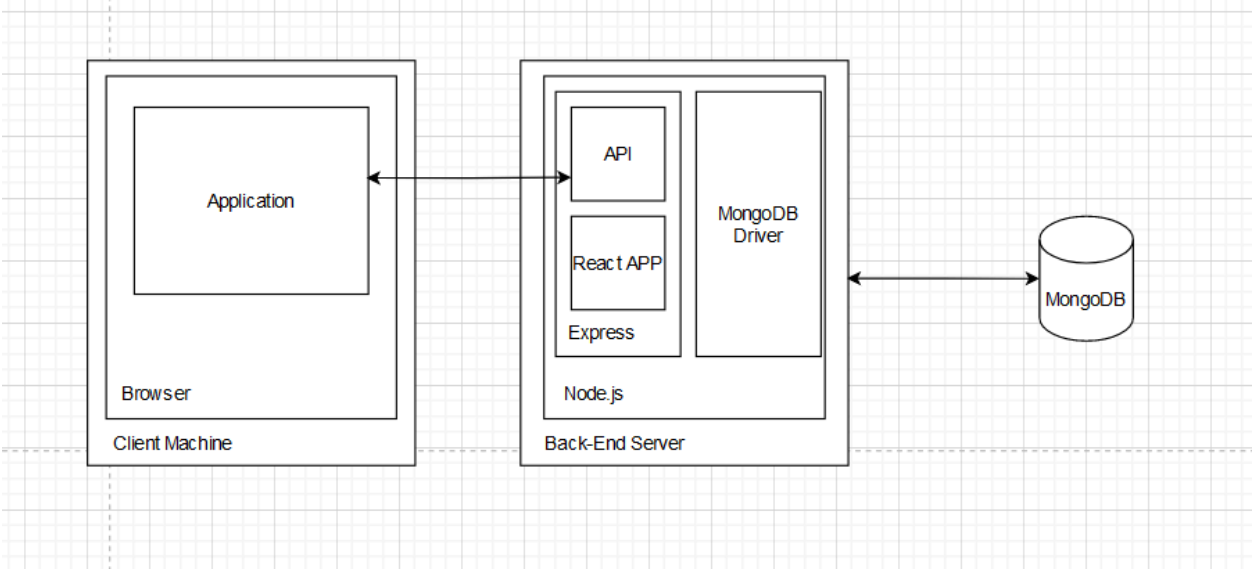

Рисунок 3.1 – Архітектура вебплатформи

Джерело: розроблено автором

3.2 Реалізація БД та її інтеграція в платформу

Для початку роботи, перше, що потрібно зробити, це створити новий проект на сайті MongoDB. Зареєструвати обліковий запис, знайти кнопку "Новий проект" і дати проєкту назву.

На рисунку 3.2 представлено вигляд створеного проєкту.

Після того, як створено проект, потрібно буде створити новий кластер. Для цього необхідно натиснути кнопку "Створити кластер" і дотримуватись підказок, щоб налаштувати новий кластер. Ви можете вибрати тип кластера, регіон та інші параметри, які найкраще відповідають потребам бази даних.

Мета кластера - забезпечити високу доступність і масштабованість для розгортання MongoDB. Але останнім часом доведеться спочатку створити базу даних в MongoDB, і можна це зробити, натиснувши на кнопку "Створити базу даних".

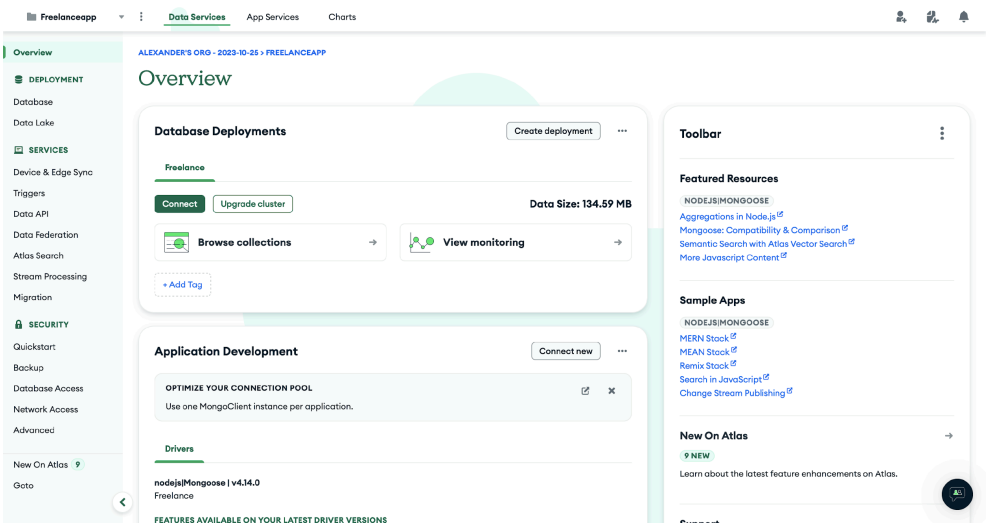

Рисунок 3.2 – Вигляд створеного проєкту

Джерело: розроблено автором

На рисунках 3.3 – 3.5 представлено результат створення кластеру та бази даних.

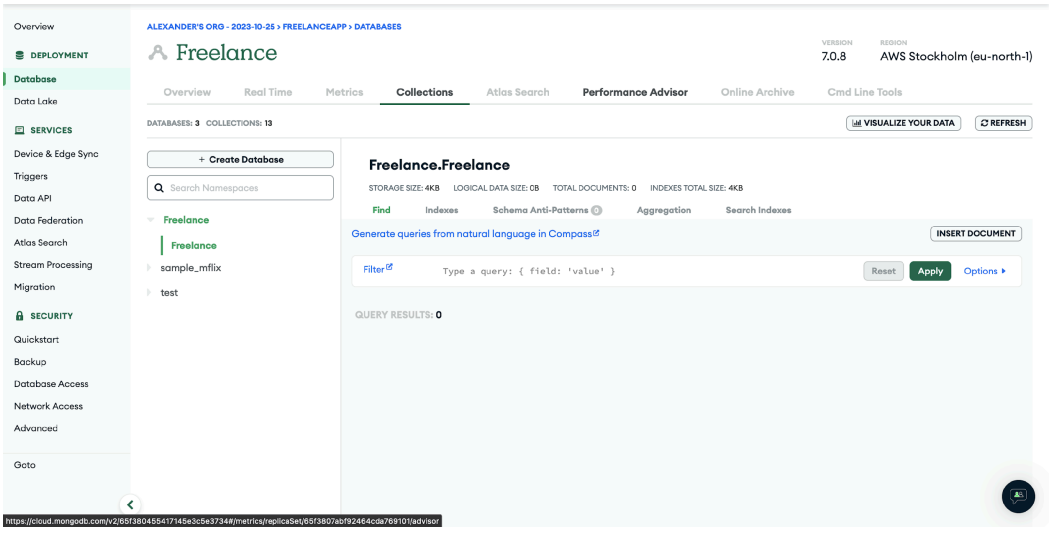

Рисунок 3.3 – Результат створення кластеру

Джерело: розроблено автором

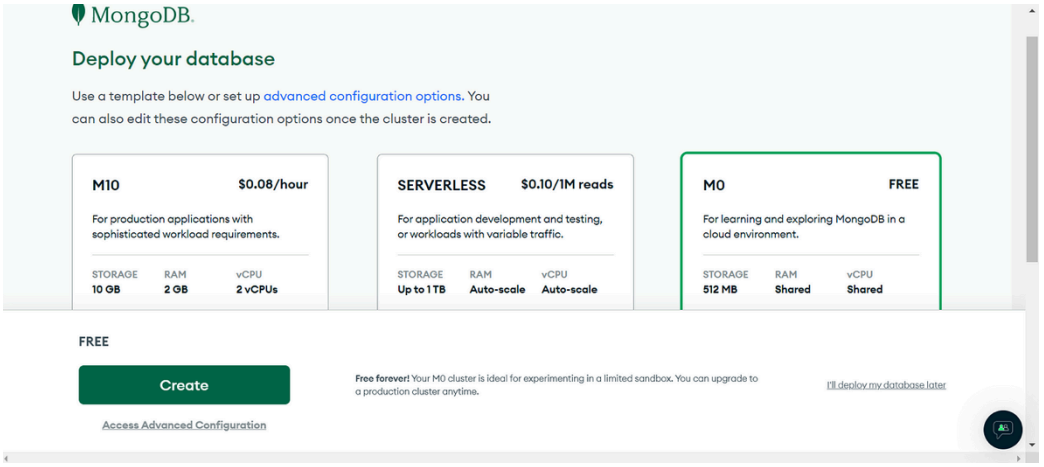

Рисунок 3.4 – Ілюстрація результату розгортання БД

Джерело: розроблено автором

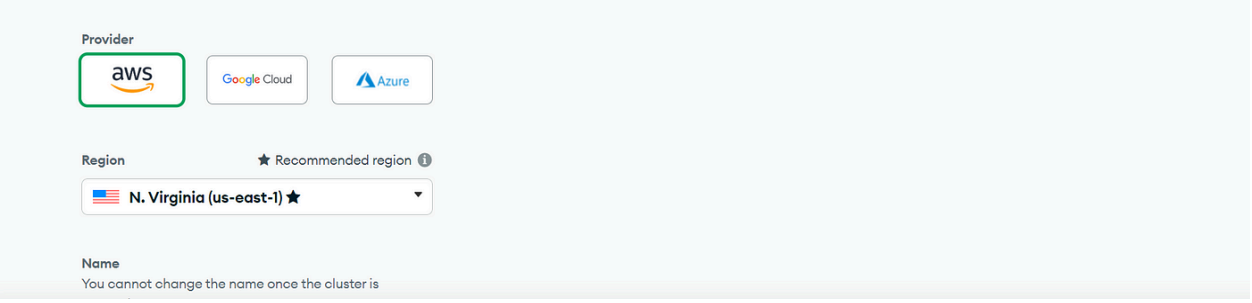

Рисунок 3.5 – Ілюстрація результату вибору провайдеру для підтримки БД

Джерело: розроблено автором

Після того як було виконано налаштування БД, необхідно задати власну IP-адресу для користування. Для цього необхідно натиснути на кнопку "Підключитися", перейти до розділу "Доступ до мережі" в розділі "Безпека" на бічній панелі, а потім натиснути "ДОДАТИ IP-АДРЕСУ".

На рисунку 3.6 представлено результат додавання власної IP-адреси.

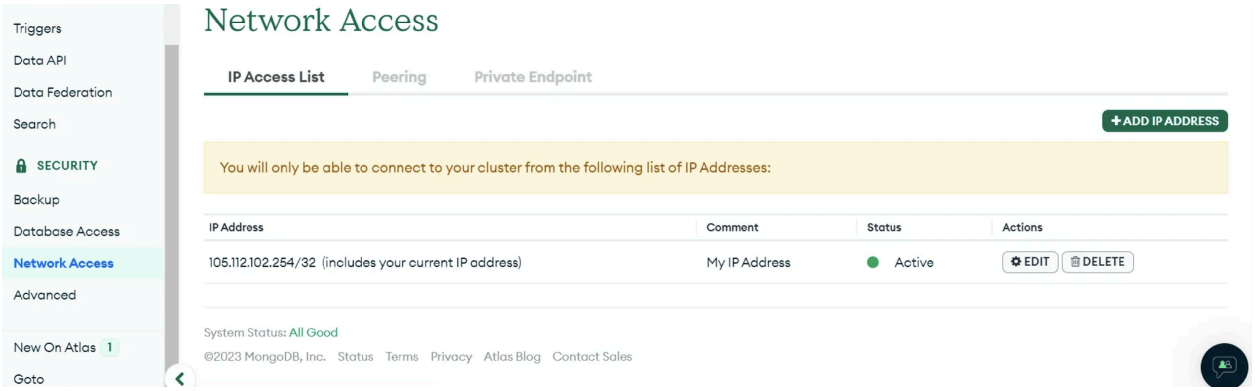

Рисунок 3.6 – Ілюстрація результату додавання IP-адреси

Джерело: розроблено автором

Після виконання всіх операцій, необхідно з'єднати створену БД з вебплатформою. Для цього неоюхідно перейти на вкладку "База даних", обрати базу даних і натиснути "Підключити". Також необхідно обрати метод підключення, який найкраще підходить для потреб, наприклад, підключення через Node.js.

На рисунку 3.7 представлено вигляд модульного вікна з підключенням БД до платформи.

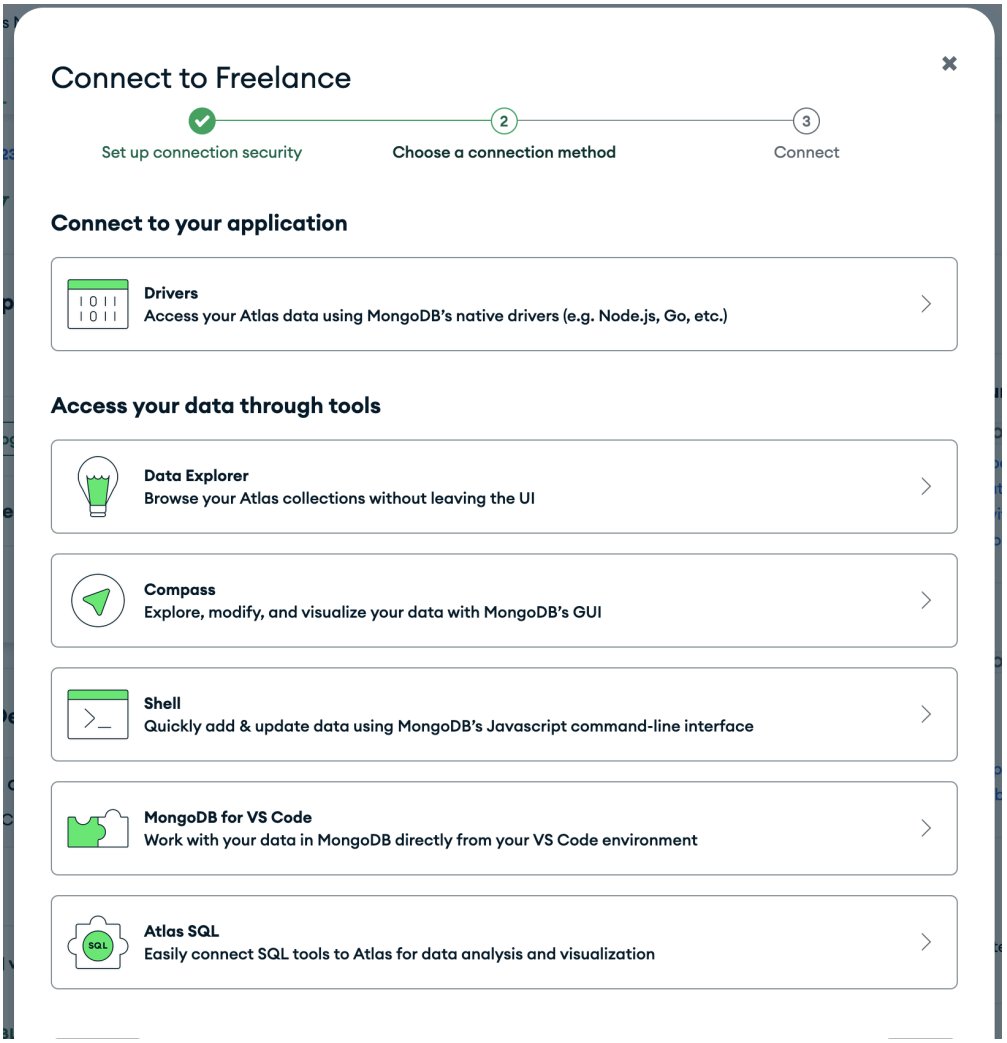

Рисунок 3.7 – Вигляд вікна підключення БД до платформи

Джерело: розроблено автором

Потім необхідно скопіювати посилання URL і вставити його безпосередньо у експрес-сервер або вставити файл .env, що вказує на MONGO_URL.

На рисунках 3.8 – 3.13 представлено створені моделі для серверної частини вебплатформи.

Рисунок 3.8 – Реалізація моделі «chatModel»

Джерело: розроблено автором

Рисунок 3.9 – Реалізація моделі «messageModel»

Джерело: розроблено автором

Рисунок 3.10 – Реалізація моделі «orderModel»

Джерело: розроблено автором

Рисунок 3.11 – Реалізація моделі «serviceModel»

Джерело: розроблено автором

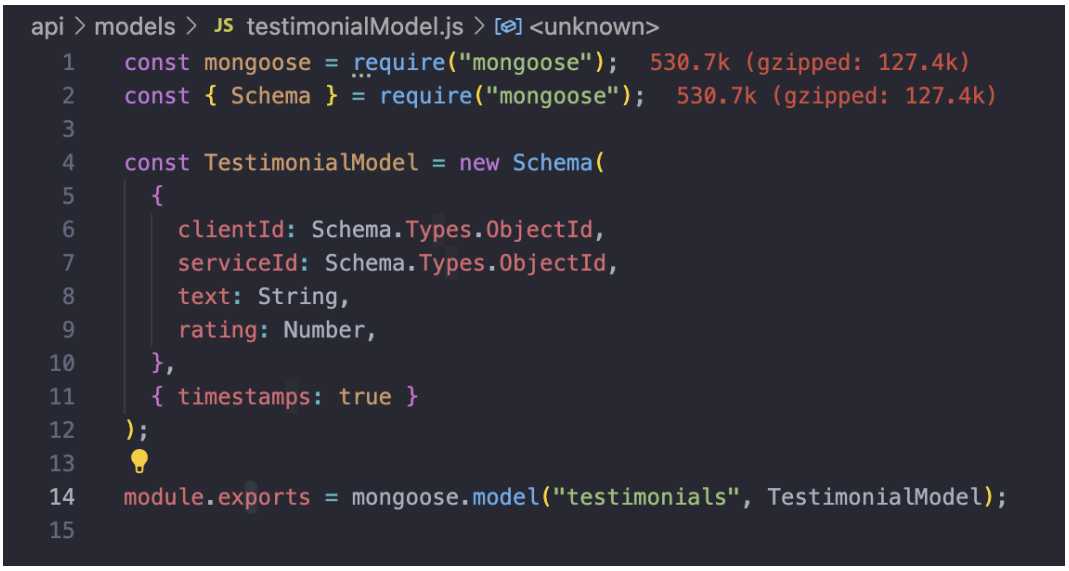

Рисунок 3.12 – Реалізація моделі «testimonialModel»

Джерело: розроблено автором

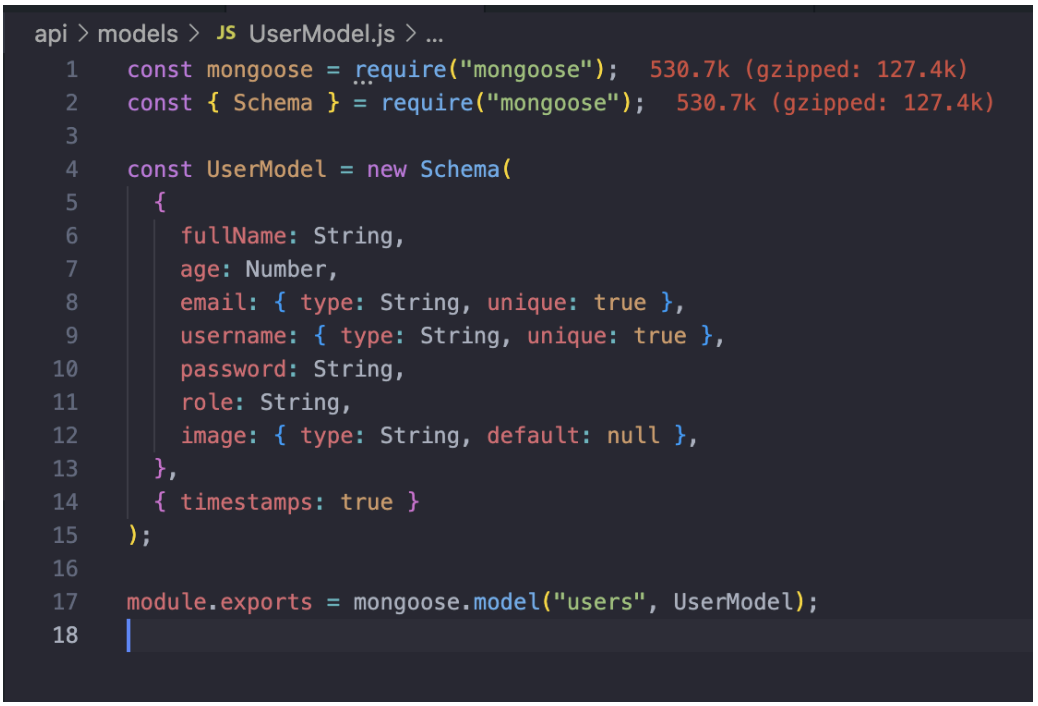

Рисунок 3.13 – Реалізація моделі «UserModel»

Джерело: розроблено автором

3.3 Реалізація серверної частини

На початку реалізації серверної частини вебплатформи було створено файл «database.js», який відповідає за підключення бази даних до сервсеру. На рисунку 3.14 представлено результат створення файлу «database.js».

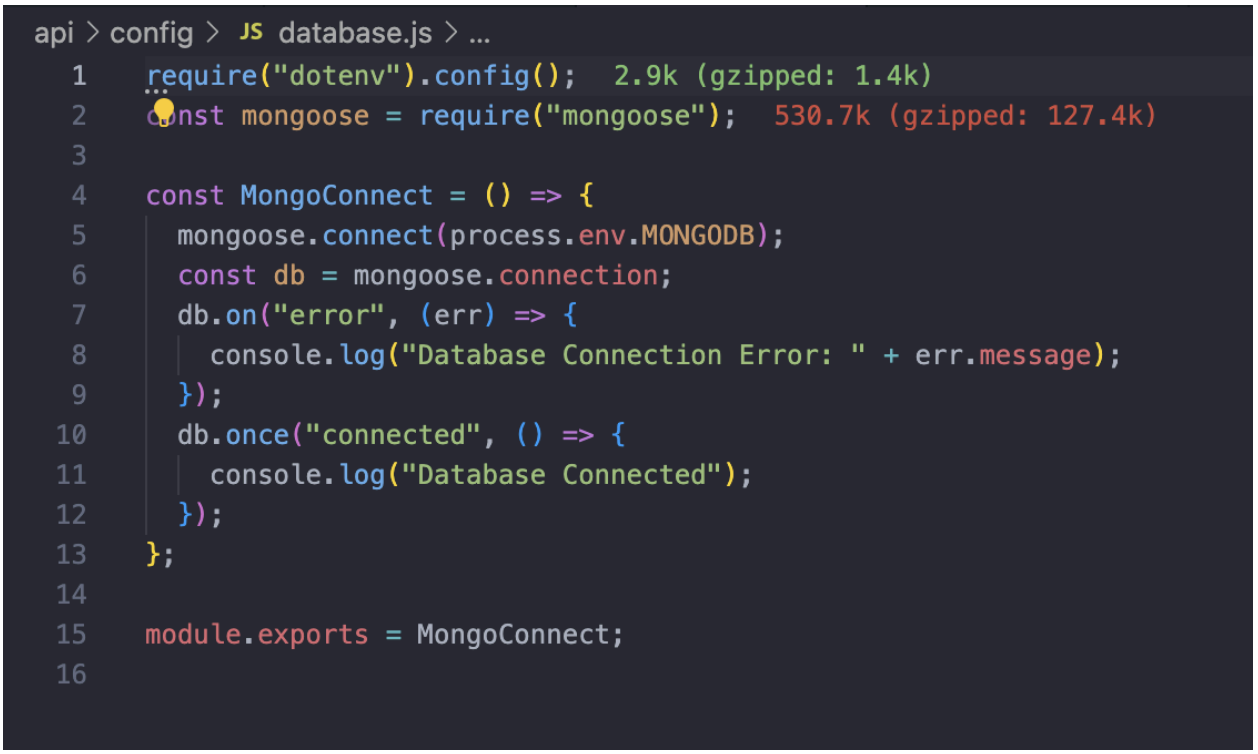

Рисунок 3.14– Результат створення файлу «database.js»

Джерело: розроблено автором

Для обробки запитів від клієнтів та виконавців було створено контролери які визначають дії, які необхідно виконати в залежності від URL запиту.

Контролери приймають дані від клієнтського інтерфейсу, обробляють їх і надсилають відповідь. Наприклад, коли користувач надсилає форму для

створення нового проекту, контролер відповідає за зберігання цього проекту в базі даних.

На рисунках 3.15 – 3.17 представлено результат створення контролерів для вебплатформи підтримки фрілансерів. Лістинги коду «ChatController», «OrdersController» та «UserController» представлені в додатку В.

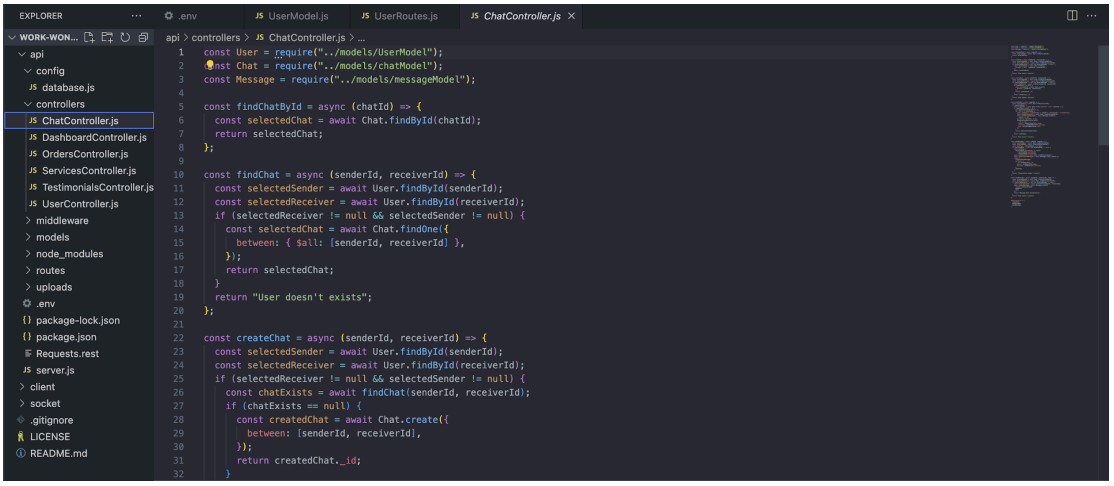

Рисунок 3.15 – Результат створення контролеру «ChatController»

Джерело: розроблено автором

Рисунок 3.16 - Результат створення контролеру «OrdersController»

Джерело: розроблено автором

Рисунок 3.17 - Результат створення контролеру «UserController»

Джерело: розроблено автором

У серверній частині вебплатформи файли маршрутів використовуються для визначення маршрутів, за якими клієнти отримують доступ до різних функцій і сторінок додатку.

Це включає в себе запити на маршрутизацію, ініціалізацію контролерів, обробку параметрів маршрутів, автентифікацію та авторизацію, обробку помилок тощо.

У контексті вебплатформи для підтримки діяльності фрілансерів, маршрути використовуються для реєстрації, входу, створення та перегляду сторінок проекту, взаємодії з користувачами, управління профілями та налаштуваннями користувачів.

На рисунках 3.18 – 3.20 представлено частину реалізованих маршрутизаторів. Всі лістинги кодів маршрутизаторів представлені в додатку Г.

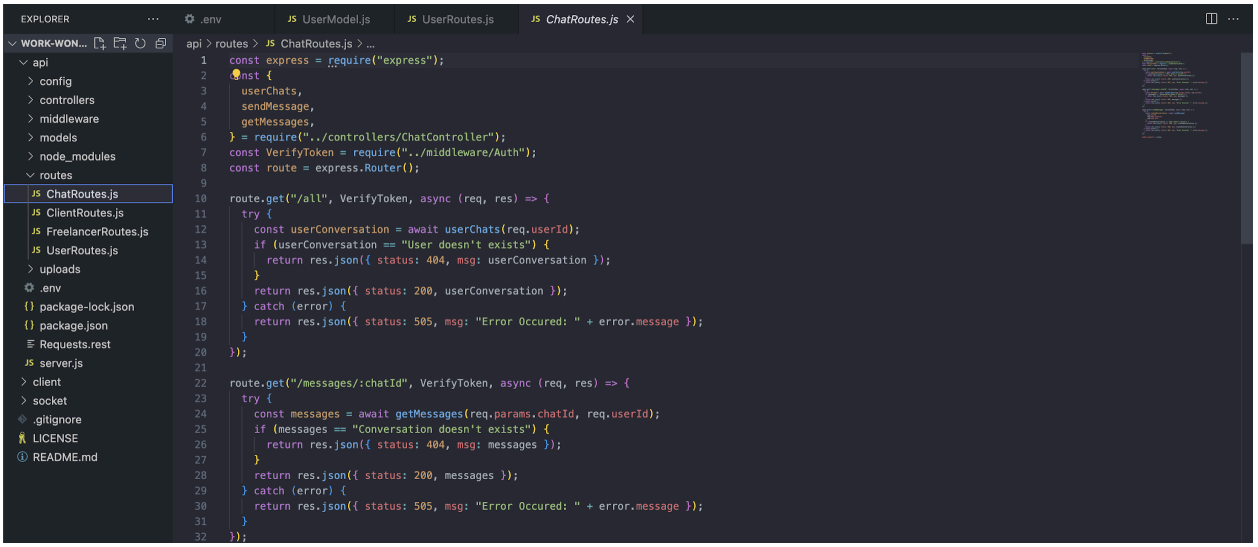

Рисунок 3.18 – Результат реалізації маршрутизатора «CharRoutes»

Джерело: розроблено автором

Рисунок 3.19 – Результат реалізації маршрутизатора «ClientRoutes»

Джерело: розроблено автором

Рисунок 3.20 - Результат реалізації маршрутизатора «UserRoutes»

Джерело: розроблено автором

3.4 Реалізація клієнтської частини

Для реалізації клієнтської частини було реалізовано компоненти, які дозволяють користувачам переглядати, фільтрувати та сортувати інформацію про проекти, спілкуватися з клієнтами, керувати завданнями, залишати відгуки та оцінювати один одного. Це полегшує співпрацю та покращує користувацький досвід.

На рисунках 3.21 – 3.23 представлено процес реалізації компонентів для клієнтської частини вебплатформи.

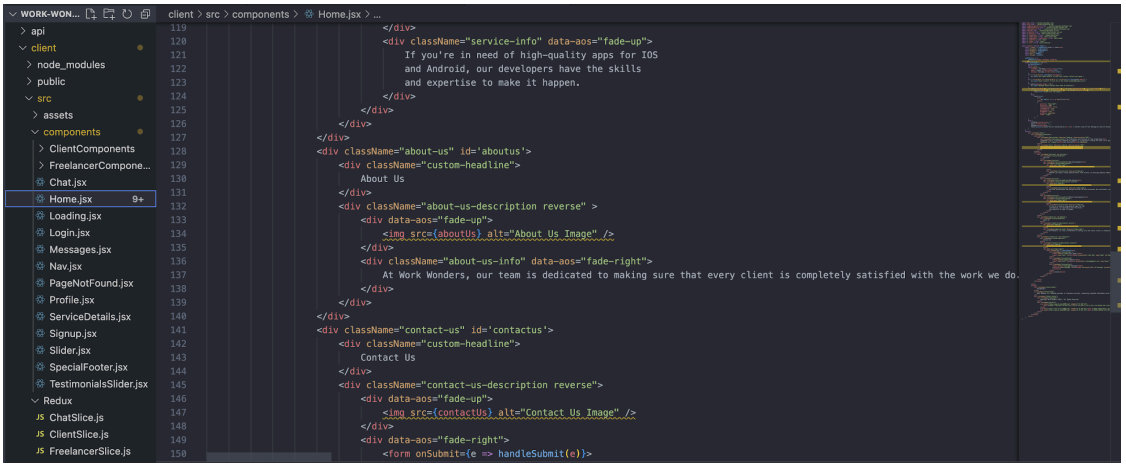

Рисунок 3.21 – Результат реалізації компоненту «Home.jsx»

Джерело: розроблено автором

Рисунок 3.22 – Результат реалізації компоненту «Login.jsx»

Джерело: розроблено автором

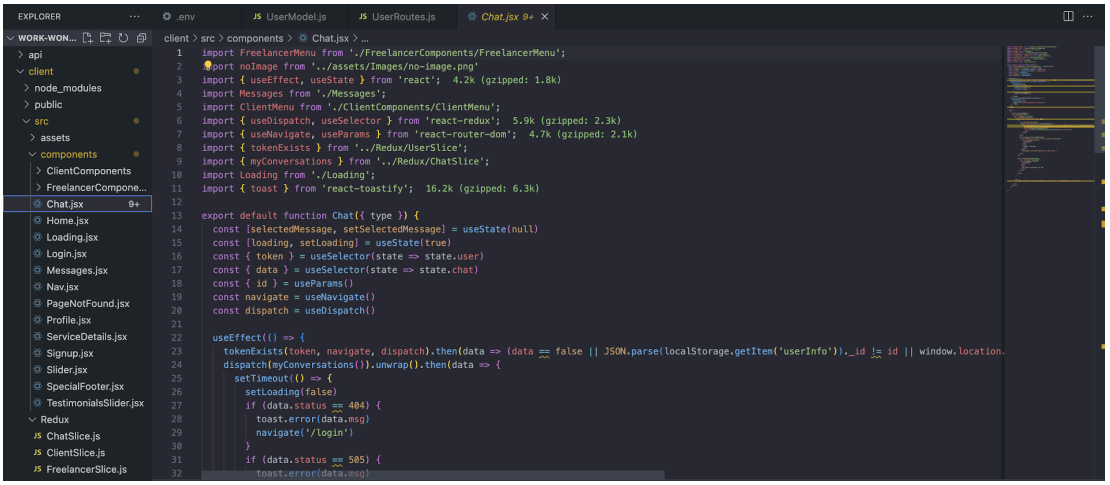

Рисунок 3.23 – Результат реалізації компоненту «Chat.jsx»

Джерело: розроблено автором

Використання Redux у клієнтській частині вебплатформи надала можливість спростити розробку, підтримку та масштабування додатку, забезпечуючи ефективне управління станом та даними. На рисунках 3.24 – 3.26 представлено результат створення redux-файлів.

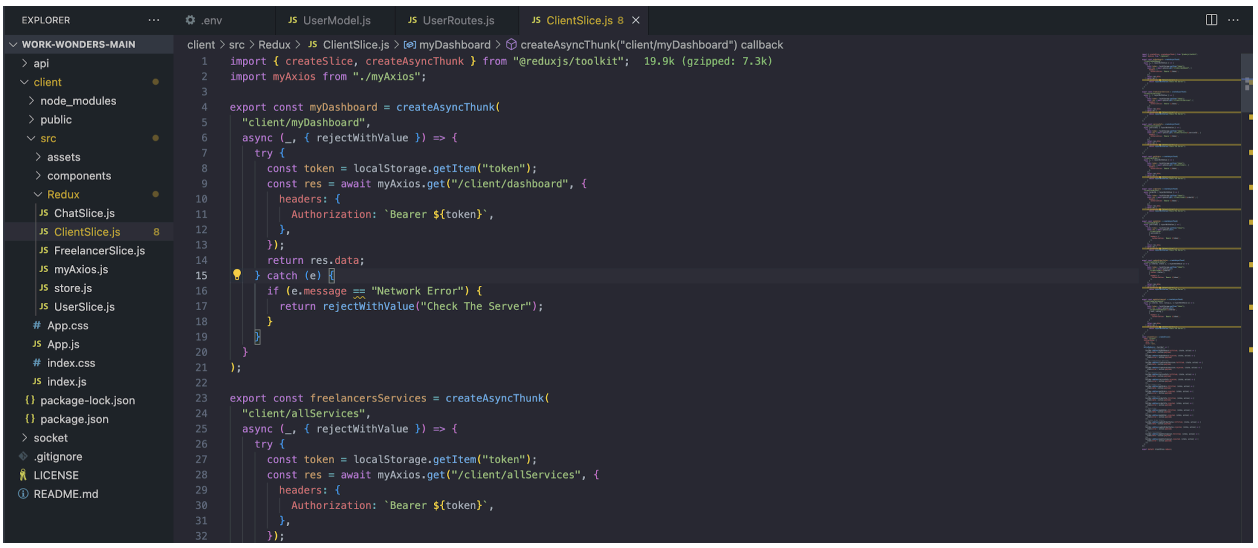

Рисунок 3.24 – Результат реалізації файлу «ClientSlice»

Джерело: розроблено автором

Рисунок 3.25 – Результат реалізації файлу «ChatSlice»

Джерело: розроблено автором

Рисунок 3.26 – Результат реалізації файлу «FreelacneSlice»

Джерело: розроблено автором

3.5 Настанови з використання

На початку користування платформою, користувач потрапляє на головну сторінку вебплатформи на якій надається основна інформація про переваги та діяльність платформи.

На рисунках 3.27 – 3.30 представлено вигляд головної сторінки вебплатформи підтримки діяльності фрілансерів.

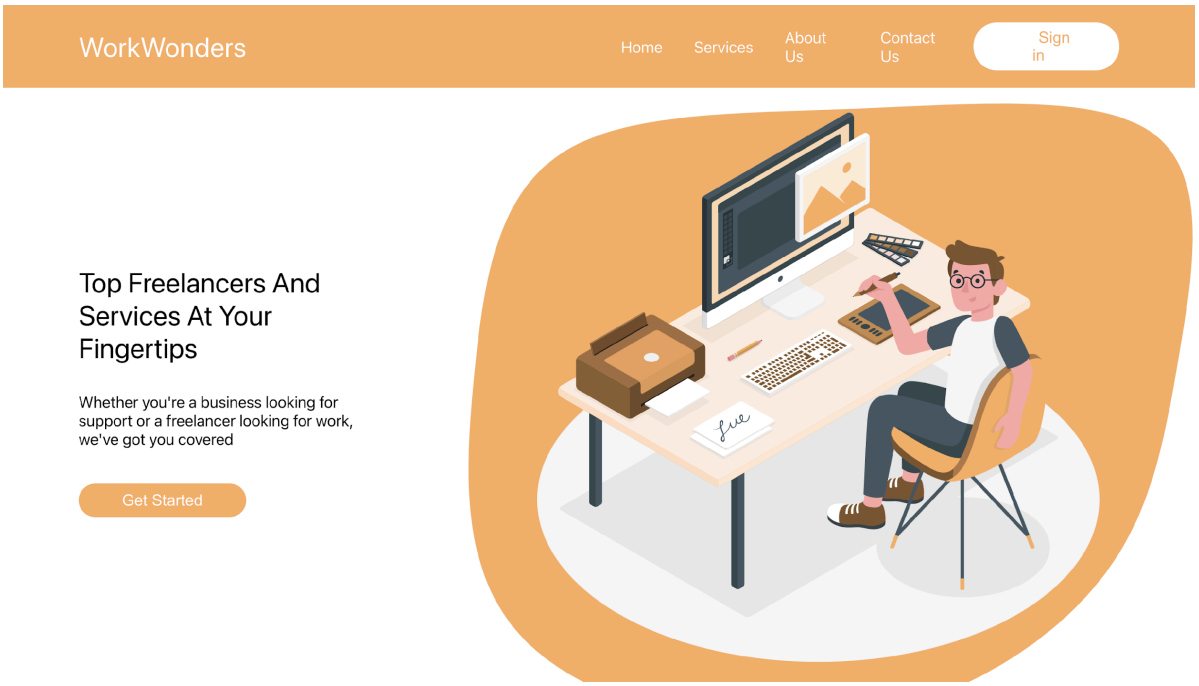

Рисунок 3.27 – Вигляд головної сторінки платформи. Частина 1

Джерело: розроблено автором

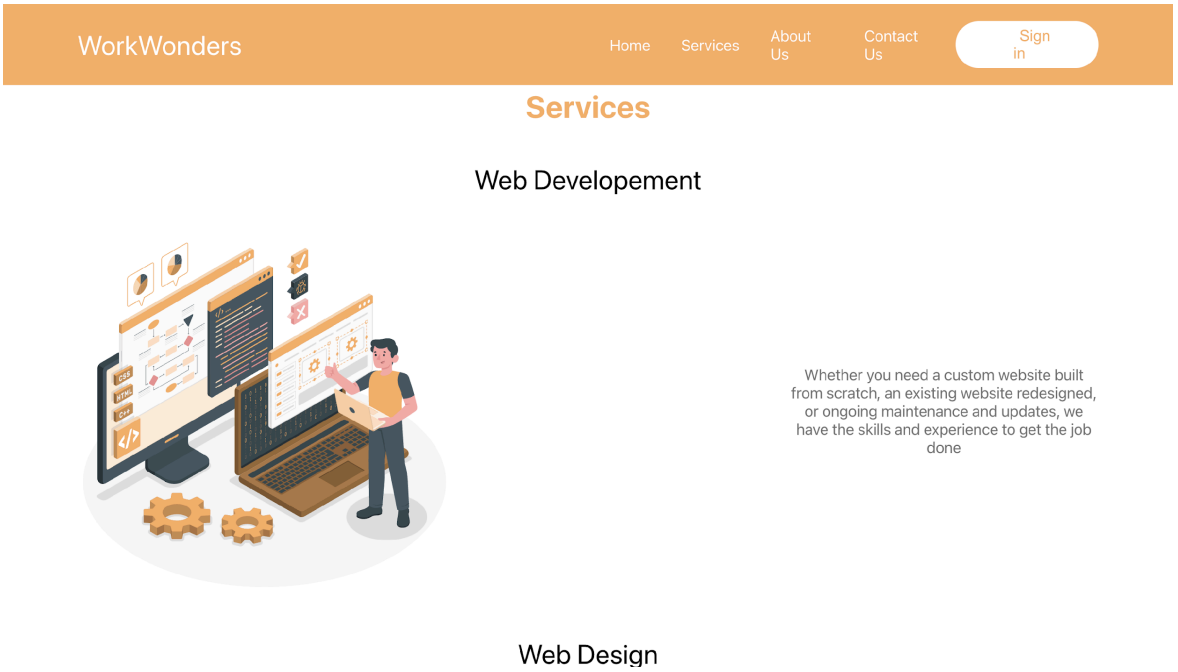

Рисунок 3.28 – Вигляд головної сторінки платформи. Частина 2

Джерело: розроблено автором

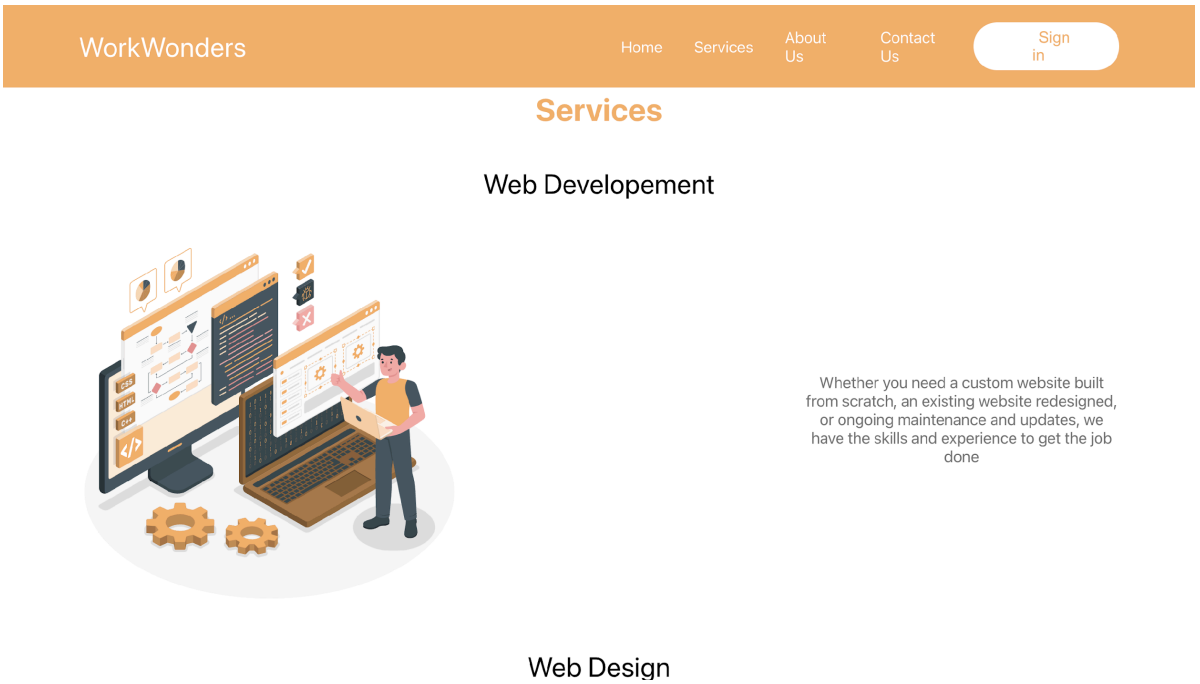

Рисунок 3.29 - Вигляд головної сторінки платформи. Частина 3

Джерело: розроблено автором

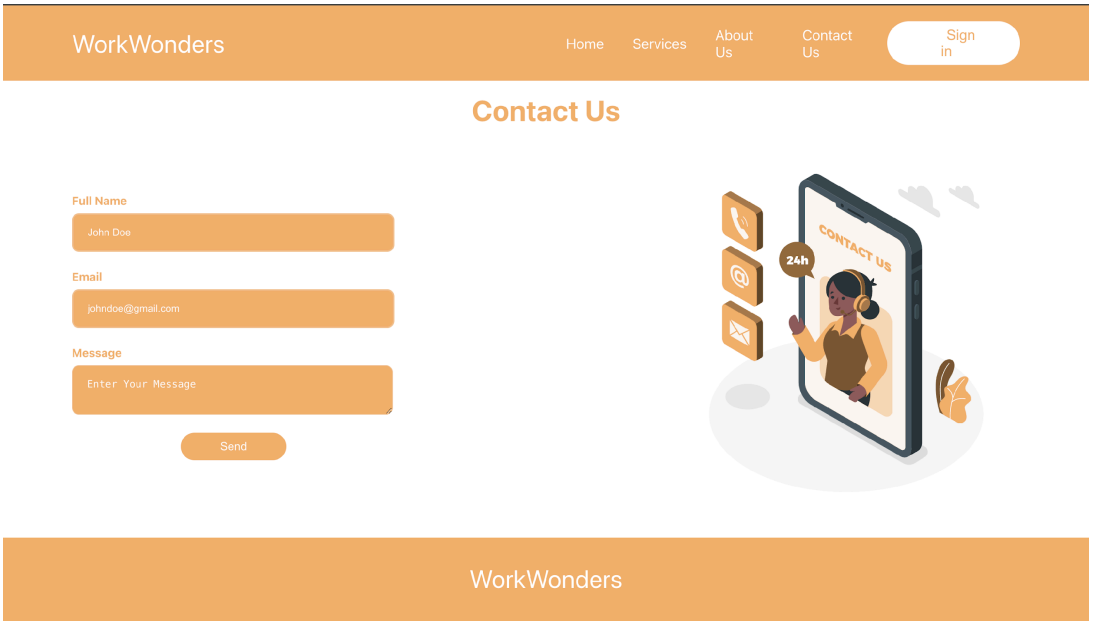

Рисунок 3.30 - Вигляд головної сторінки платформи. Частина 4

Джерело: розроблено автором

Для використання послуг, які надає вебплатформа, користувач повинен виконати операцію авторизації або реєстрації облікового запису. На рисунках 3.31 – 3.32 представлено сторінку авторизації та реєстрації.

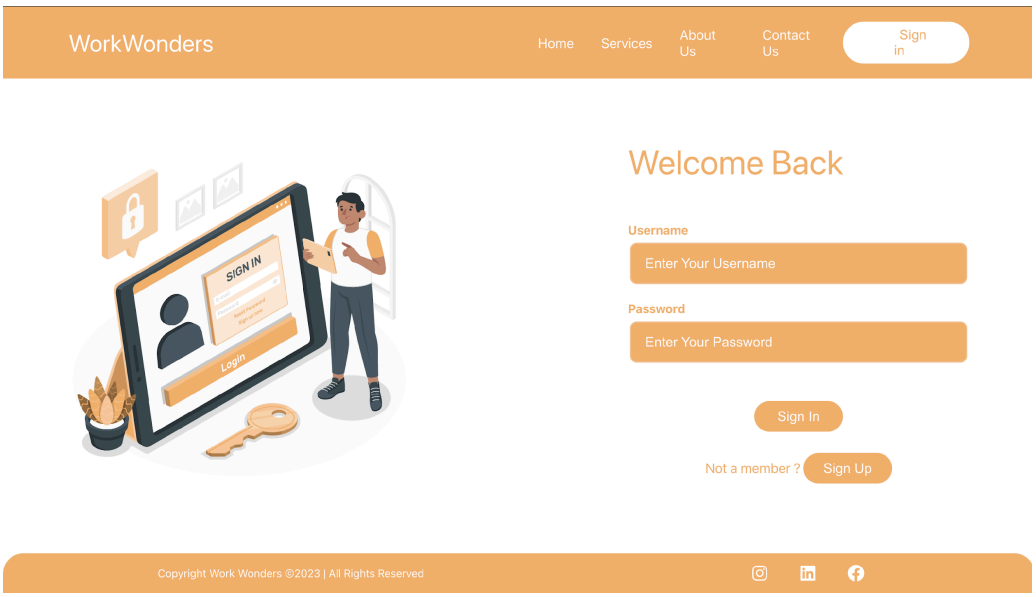

Рисунок 3.31 – Вигляд сторінки авторизації

Джерело: розроблено автором

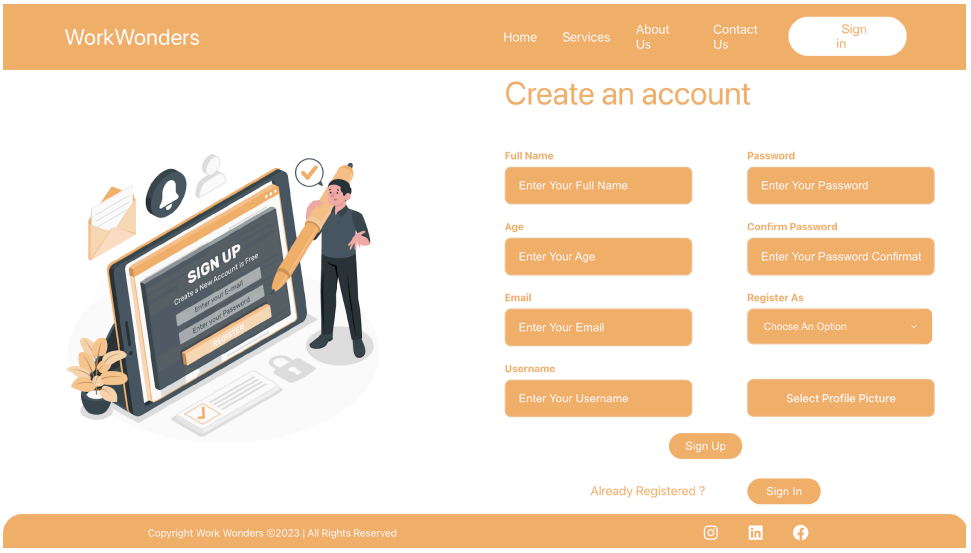

Рисунок 3.32 – Вигляд сторінки реєстрації

Джерело: розроблено автором

Після того як користувач виконав авторизацію в системі, виконується переадресація на особисту сторінку користувача. На рисунку 3.33 представлено вигляд особистої сторінки користувача.

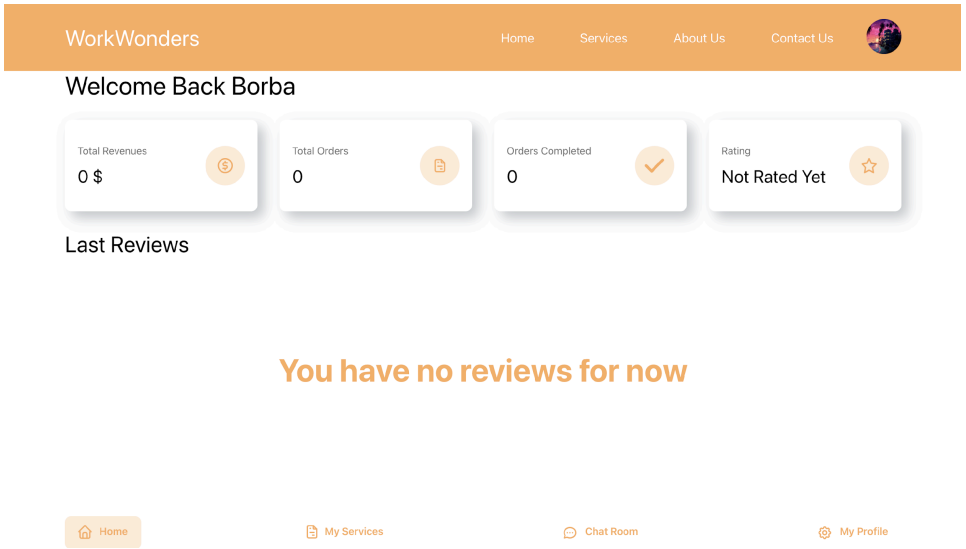

Рисунок 3.33 – Особиста сторінка користувача

Джерело: розроблено автором

Для потреби, кожен користувач може змінити особисту інформацію про себе (рис. 3.34).

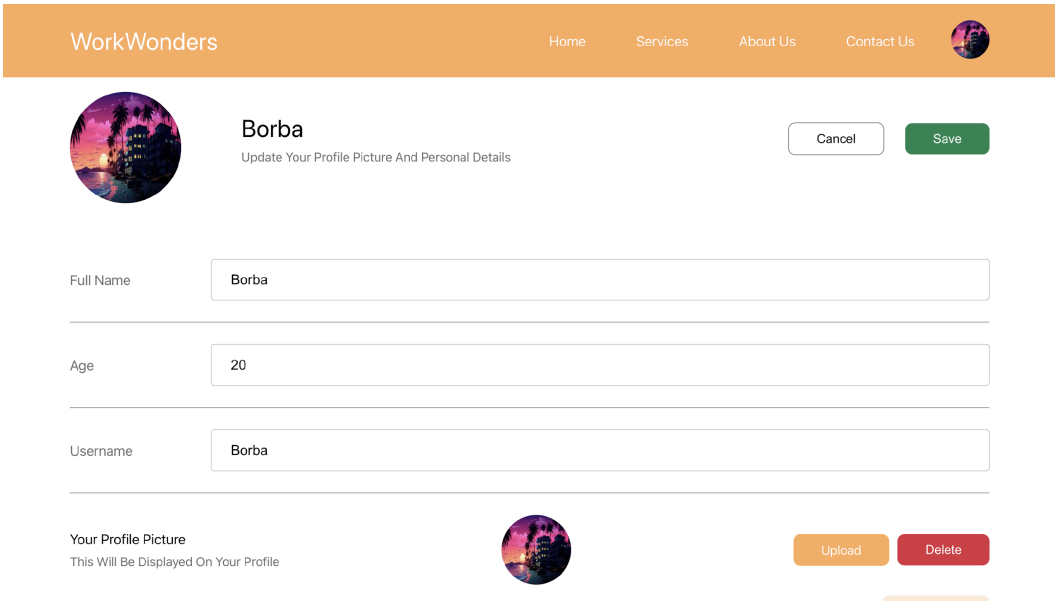

Рисунок 3.34 – Сторінка редагування особистої інформації

Джерело: розроблено автором

Також для фрілансерів доступна сторінка додавання власних послуг, за допомогою якої будь-який фрілансер може запропонувати виконання завдань будь-якого напрямку.

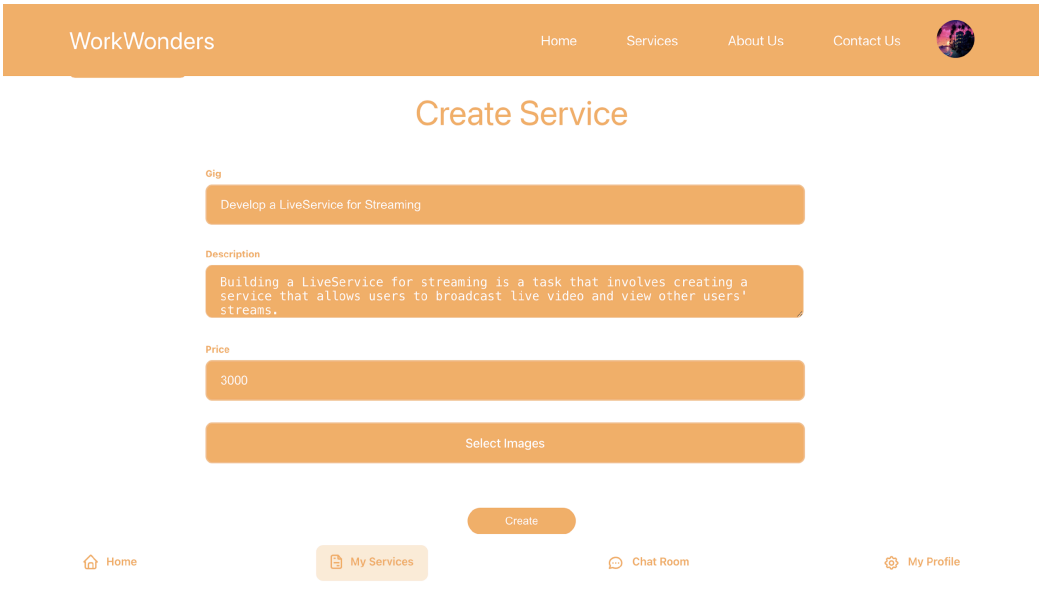

Рисунок 3.35 – Сторінка додавання власних послуг

Джерело: розроблено автором

Результат додавання послуги та сторінку редагування доданих послуг представлено на рисунках 3.36 – 3.37.

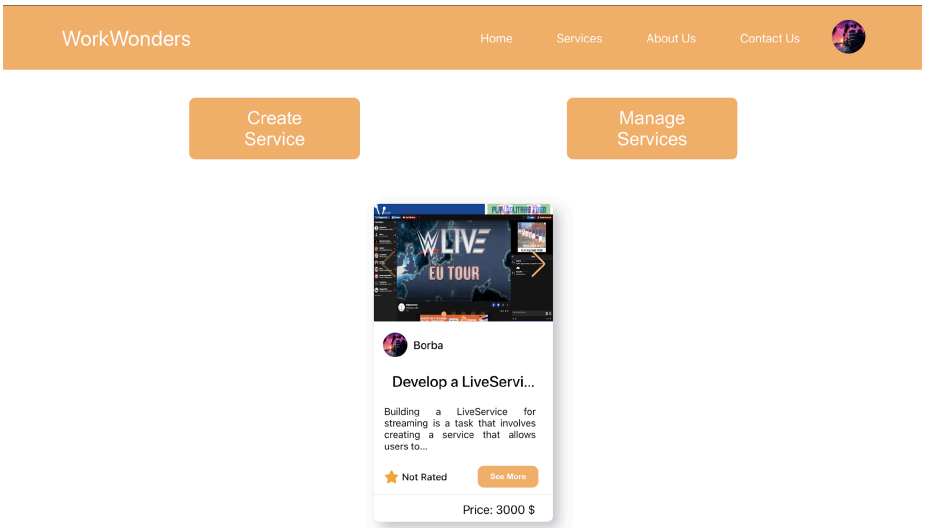

Рисунок 3.36 – Результат додавання послуги

Джерело: розроблено автором

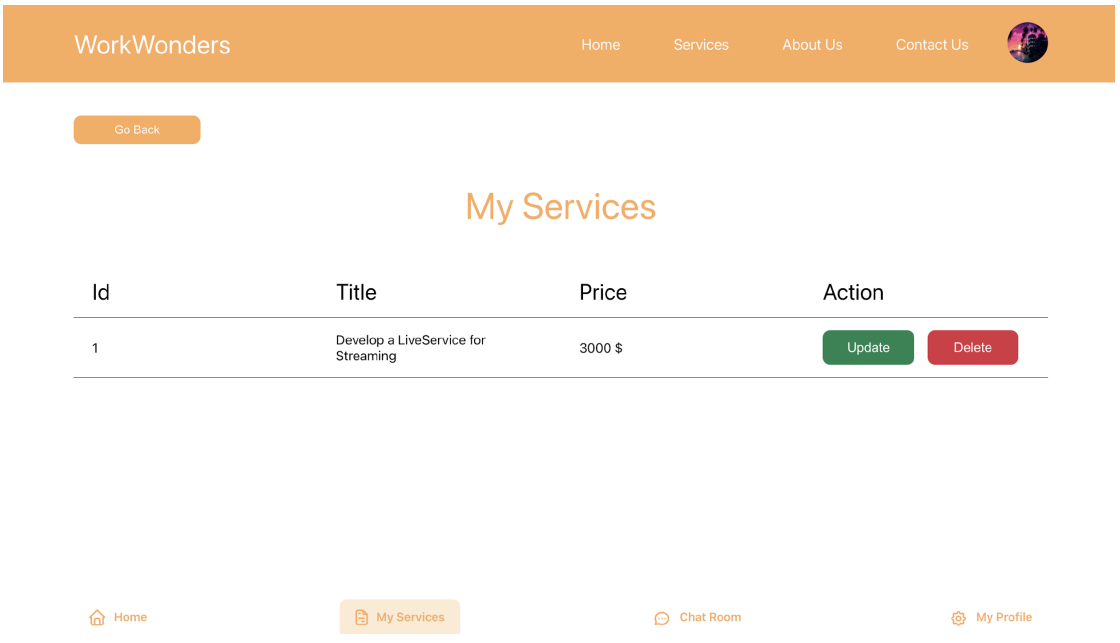

Рисунок 3.37 – Сторінка редагування інформації про послуги

Джерело: розроблено автором

Якщо користувач відчуває потребу в виконанні якогось завдання, то він може обрати послугу яка розміщена на платформі. На рисунку 3.38 представлено вигляд доступних послуг від інших користувачів.

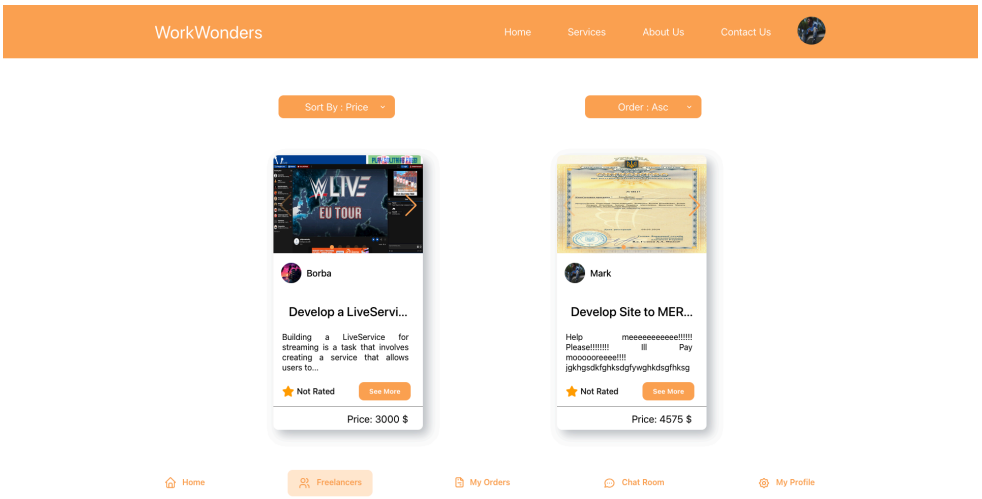

Рисунок 3.38 – Вигляд доступних послуг від інших користувачів

Джерело: розроблено автором

При натисканні на послугу, користувач потрапляє на сторінку детального опису послуги (рис. 3.39).

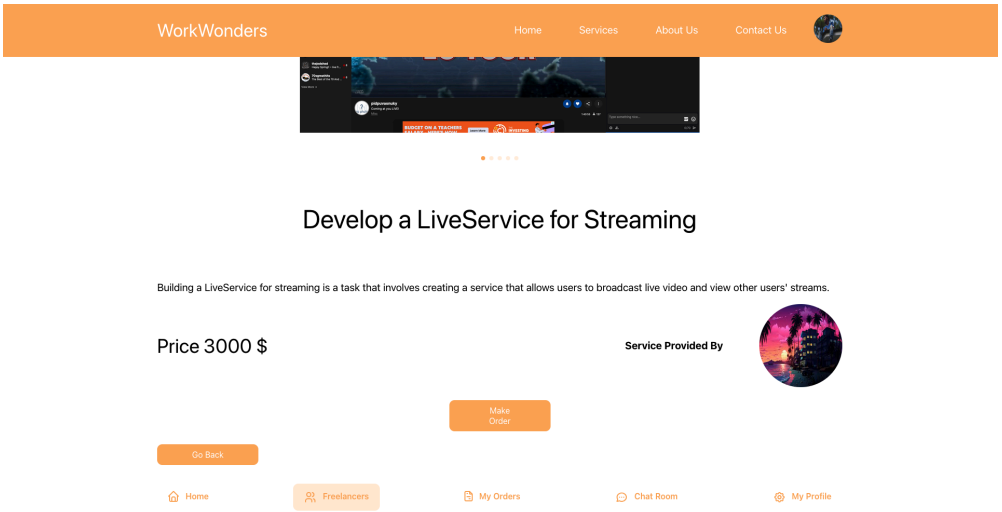

Рисунок 3.39 – Сторінка детального опису послуги

Джерело: розроблено автором

Після того як фрілансер прийняв послугу в написанні сервісу, то платформа пропонує систему чату для підтримки зв`язку з замовником. На рисунку 3.40 представлено вигляд чату фрілансера з замовником.

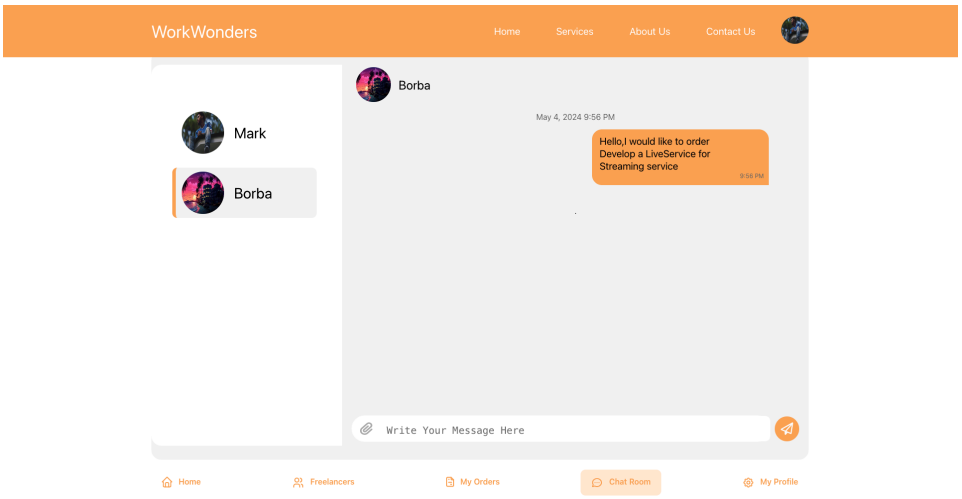

Рисунок 3.40 – Вигляд чату фрілансера з замовником

Джерело: розроблено автором

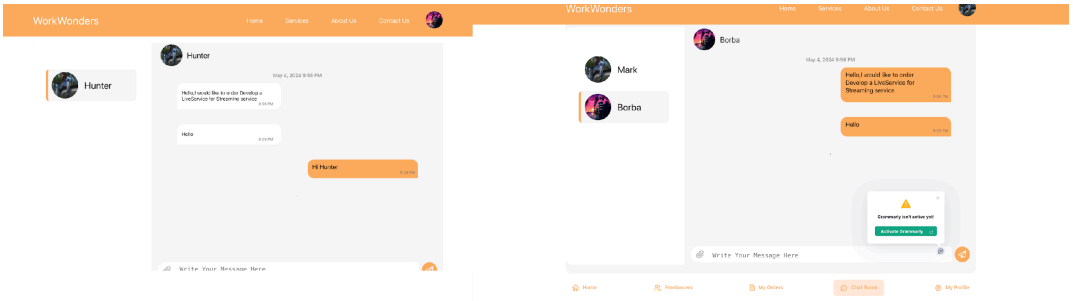

Рисунок 3.41 – Вигляд чату з різних браузерів

Джерело: розроблено автором

Після успішного виконання роботи, особиста сторінка надасть інформацію про виконання завдання (рис.3.42).

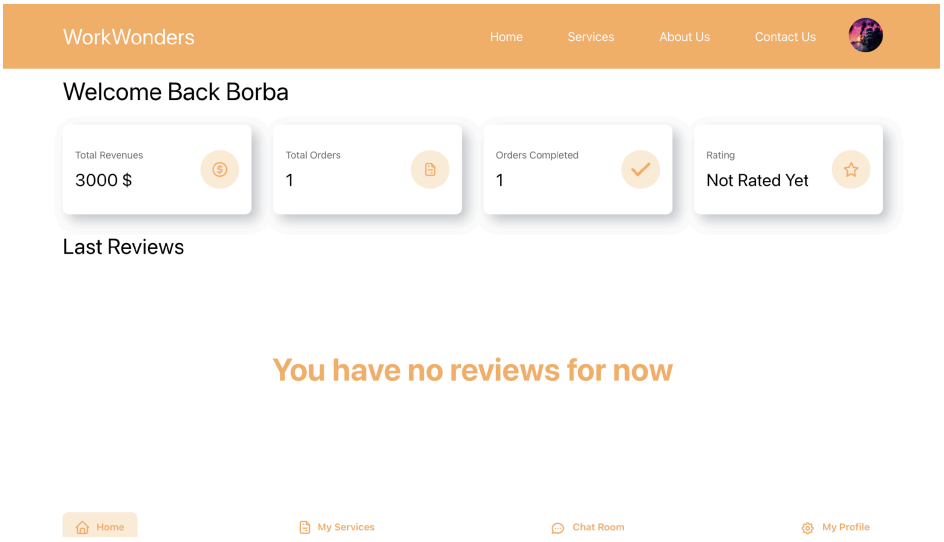

Рисунок 3.42 – Вигляд особистої сторінки

Джерело: розроблено автором

ВИСНОВКИ

Метою кваліфікаційної роботи бакалавра є розробка вебплатформи для підтримки діяльності фрілансерів. Для визначення актуальності цієї розробки було застосовано аналітичний підхід, в рамках якого досліджувалися як аналогові системи, так і предметна область, про яку йде мова.

Були визначені функціональні та нефункціональні вимоги до платформи та сформовано технічне завдання на проектування вебплатформи. Визначені основні цілі проекту та функціональні вимоги до вебплатформи.

Крім того, були визначені технології, які будуть використовуватися для реалізації цієї системи, а також база даних для обробки та зберігання даних, а саме React.js, Express.js, Node.js та MongoDB.

Планування IT проекту дозволило узгодити терміни виконання задач проектування та сформована діаграма Ганта. Для запобігання виникненню ризиків у процесі розробки було проведено роботу з управління ризиками.

Для більш детального управління процесом розробки були використані методології OBS та WBS. Організаційна структура проекту (OBS) визначила відповідальність членів команди, забезпечивши чіткий розподіл обов'язків та контроль за виконанням завдань.

Структура розподілу робіт (WBS) слугувала основою для поділу проекту на менші, більш керовані компоненти, що сприяло посиленню контролю над робочим процесом та більш ефективному використанню ресурсів.

Було виконано структурно-функціональне моделювання та розроблено діаграму варіантів використання, які надають інформацію про функціональні можливості користувача.

Впровадження вебплатформи забезпечить оптимізацію процесів для фрілансерів. Потреба у високоефективних інструментах для підвищення

продуктивності та покращення клієнтського досвіду робить цю розробку актуальною та важливою для сучасних фрілансерів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Benefits of Using Online Freelance Platforms Автор: Ladan Jafari [Електронний ресурс] – Режим доступу до ресурсу: https://winatalent.com/blog/benefits-of-using-online-freelance-platforms/ (дата звернення 22.03.2024).

2. MVC Architecture: A Comparative Study Between Laravel Framework and Slim Framework in Freelancer Project Monitoring System Web Based Andri Sunardi, Suharjito Procedia Computer Science, 157, 1 2019 (дата звернення 22.03.2024).

3. Do IT freelancers increase their entrepreneurial behavior and performance by using IT self-efficacy and social capital? Evidence from Bangladesh Rabeya Sultana, Il Im et al. Information & Management, 56, 6, 9 2019 (дата звернення 22.03.2024).

4. Appealing to Internet-based freelance developers in smartphone application marketplaces Jung Kuei Hsieh, Yi Ching Hsieh International Journal of Information Management, 33, 2, 4 2013 (дата звернення 22.03.2024).

5. Freelance Platforms vs. Independent Clients (Pros and Cons) Автор: Wayne Mullins [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@fireyourself/freelance-platforms-vs-independent-clients-pros-and-cons-19ba8570447f

6. Top 3 Freelance Platforms in 2024: Comparison of Fiverr, Upwork, and Freelancer.com [Електронний ресурс] – Режим доступу до ресурсу: https://financesonline.com/top-3-freelance-platforms/ (дата звернення 22.03.2024).

7. A Comprehensive Comparison of Freelancing Platforms: Finding the Right Fit for Your Skills [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@nathaliefreelance9/a-comprehensive-comparison-of-freelanc

ing-platforms-finding-the-right-fit-for-your-skills-4a44efd0aeef (дата звернення 22.03.2024).

8. Вебплатформа для фрілансерів Fiverr [Електронний ресурс] – Режим доступу до ресурсу: https://www.fiverr.com/ (дата звернення 22.03.2024).

9. Вебплатформа для фрілансерів upwork [Електронний ресурс] – Режим доступу до ресурсу: https://www.upwork.com (дата звернення 22.03.2024).

10. Вебплатформа для фрілансерів Freelancer.com [Електронний ресурс] – Режим доступу до ресурсу: https://www.freelancer.com (дата звернення 22.03.2024).

11. What Is the MERN Stack? Guide & Examples Автор Jeffrey Erickson [Електронний ресурс] – Режим доступу до ресурсу: https://www.oracle.com/cis/database/mern-stack/ (дата звернення 22.03.2024).

12. What is the MEAN stack? [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/mean-stack (дата звернення 22.03.2024).

13. What is MEVN stack? Everything You Need To Know [Електронний ресурс] – Режим доступу до ресурсу: https://techvify-software.com/what-is-mevn-stack/ (дата звернення 22.03.2024).

14. Mern Stack Advantages And Disadvantages [Електронний ресурс] – Режим доступу до ресурсу: https://www.skillvertex.com/blog/mern-stack-advantages-and-disadvantages/ (дата звернення 22.03.2024).

15. IDEF0 Diagram [Електронний ресурс] – Режим доступу до ресурсу: https://vitechcorp.com/resources/GENESYS/onlinehelp/desktop/Views/IDEF0.htm (дата звернення 22.03.2024).

16. What is a functional decomposition diagrams? [Електронний ресурс] – Режим доступу до ресурсу:

https://typeset.io/questions/what-is-a-functional-decomposition-diagrams-34sfugs2ta (дата звернення 22.03.2024).

17. 4 Advantages and Disadvantages of UML Diagrams for Companies [Електронний ресурс] – Режим доступу до ресурсу: https://www.pczone.co.uk/4-advantages-and-disadvantages-of-uml-diagrams-for-companies/ (дата звернення 22.03.2024).

18. Understanding the Pros and Cons of MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://www.knowledgenile.com/blogs/pros-and-cons-of-mongodb#DisadvantagesofMongoDB (дата звернення 22.03.2024).

19. MongoDB Atlas — Technical Overview & Benefits Автор: Nick Parsons [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@nparsons08/mongodb-atlas-technical-overview-benefits-9e4cff27a75e (дата звернення 22.03.2024).

20. Advantages of NoSQL Databases [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/nosql-explained/advantages (дата звернення 22.03.2024).

21. 28 SMART Goals for Smart Project Managers: A Complete Guide With Examples Автор: Kate Eby [Електронний ресурс] – Режим доступу до ресурсу:

22. 7 Benefits of Using a WBS [Електронний ресурс] – Режим доступу до ресурсу: https://tensix.com/7-benefits-of-using-a-wbs/ (дата звернення 23.03.2024).

23. Organizational Breakdown Structure (OBS) [Електронний ресурс] – Режим доступу до ресурсу: https://projectvictor.com/knowledge-base/organizational-breakdown-structure-obs/ (дата звернення 23.03.2024).

24. Discover the 5 advantages of Gantt Diagrams [Електронний ресурс] – Режим доступу до ресурсу:

https://www.wimi-teamwork.com/blog/discover-the-5-advantages-of-gantt-diagrams/ (дата звернення 23.03.2024).

25. 5 benefits of risk management [Електронний ресурс] – Режим доступу до ресурсу: https://claptek.com/blog/5-benefits-of-risk-management/ (дата звернення 23.03.2024).

ДОДАТОК А

Технічне завдання
на створення програмного продукту

«Вебплатформа підтримки діяльності фрілансерів»

ПОГОДЖЕНО

Старший викладач кафедри
інформаційних технологій

_____доц. Неня А.В.

Студент групи ІТ-03-2

_____ Борьба В.І.

2024

1 ПРИЗНАЧЕННЯ Й МЕТА СТВОРЕННЯ ПЛАТФОРМИ

1.1 Призначення вебплатформи

Вебплатформа повинна надавати можливість та сприяти ефективній комунікації між фрілансерами та потенційними клієнтами. Вона універсальною платформою для фрілансерів для демонстрації своїх послуг, ведення портфоліо проектів, отримання замовлень та відстеження своєї діяльності.

1.2 Мета створення вебплатформи

Мета створення – надання надійної та зручної платформи для розміщення замовлення та виконання фріланс проєктів. Платформа повинна сприяти зростанню ефективності праці фрілансерів та задовольняти потреби клієнтів в наданні професійних послуг.

1.3 Цільова аудиторія

До цільової аудиторії можна віднести фрілансерів різних спеціалізацій та клієнтів, які зацікавлені у наймі фахівців для виконання короткострокових або довгострокових проєктів.

2 ВИМОГИ ДО ВЕБПЛАТФОРМИ

2.1 Вимоги до структури й функціонування вебплатформи

Реалізована платформа повинна забезпечувати реєстрацію та авторизацію користувачів, пошук проєктів та фрілансерів, додавати та редагувати власні завдання. а також надавати систему для взаємодії клієнта та виконавця проєкту.

2.2 Вимоги до стилістичного оформлення платформи

Інтерфейс платформи повинен забезпечувати інтуїтивно зрозуміле уявлення про розміщену на ньому інформацію, а також надавати логічний перехід до розділів і сторінок. Посилання на сторінки повинні бути відповідати наданим заголовкам. При виборі будь-якого з пунктів меню користувачем повинна завантажуватися відповідна йому інформаційна сторінка, а в блоці меню відкриватися список підрозділів.

2.3Вимоги до розмежування доступу

Система повинна забезпечувати розмежування доступу до функціоналу залежно від ролі користувача (фрілансер, клієнт, адміністратор).

2.5 Наповнення платформи (контент)

Контент повинен включати інформацію про послуги, поради для фрілансерів та клієнтів, відгуки.

2.6 Вимоги до системи управління контентом платформи

Система управління контентом (CMS) повинна забезпечувати легке та інтуїтивно зрозуміле додавання, редагування та видалення контенту на платформі без потреби в глибоких знаннях програмування. Вона повинна підтримувати мультимедійні файли, дозволяти створювати різноманітні типи сторінок (профілі фрілансерів, проєкти, блоги), інтегрувати SEO-інструменти для оптимізації контенту.

2.7 Функціональні можливості розділів

- **Профіль фрілансера**: персоналізовані сторінки з детальною інформацією про кваліфікацію, портфоліо, відгуки клієнтів і рейтинг.
- **Проєкти**: можливість розміщення проєктів з детальним описом, ціною, термінами виконання та вимогами до фрілансерів.
- **Пошук**: розширені фільтри для пошуку фрілансерів і проєктів за різними критеріями.
- **Комунікація**: вбудовані інструменти для спілкування між фрілансерами та клієнтами через платформу.
- **Фінансові операції**: інтеграція платіжних систем для здійснення оплати за проєкти.

2.8 Вимоги до програмної платформи та коду

Реалізація платформи відбувається з використанням стеку технологій MERN (MongoDB, ExpressJS, ReactJS, NodeJS). В якості редактора коду необхідно використовувати Visual Studio Code.

ДОДАТОК Б

Планування робіт

Метою цього проекту є створення вебплатформи, яка підтримує фрілансерів у їхній роботі та спілкуванні з клієнтами. Платформа дозволить фрілансерам переглядати доступні завдання та проекти які були розміщені клієнтами, подавати свої пропозиції, розміщувати замовлення та взаємодіяти з адміністрацією платформи.

Платформа також надає можливість зворотнього зв’язку клієнта з фрілансером.

Для досягнення мети проєкту необхідно виконати наступні задачі:

- визначити актуальність розробки вебплатформи для підтримки діяльності фрілансерів у сучасному ринковому середовищі;
- дослідити предметну область фріланс-платформ та специфіку їх роботи;
- провести аналіз існуючих аналогів вебплатформ для визначення їх переваг та недоліків;
- реалізувати компоненти вебплатформи на основі потреб користувачів;
- виконати структурно-функціональне моделювання для візуального розуміння процесів;

Деталізація мети проекту методом SMART

В управлінні проектами SMART-цілі − це конкретні, вимірювані, досяжні, релевантні та обмежені в часі цілі, які керують плануванням та виконанням проекту. Цей підхід широко визнаний як ефективний спосіб управління проектами та досягнення успіху. Встановлюючи SMART-цілі, команди можуть гарантувати, що вони узгоджені і працюють над досягненням конкретних результатів проекту у встановлені терміни.

SMART-цілі, або цілі, які є конкретними, вимірюваними, досяжними, актуальними та обмеженими в часі, є широко використовуваним інструментом для керівників проектів та лідерів для встановлення чітких та вимірюваних результатів для окремих осіб та проектів. Використовуючи SMART-цілі, менеджери та керівники проектів можуть впевнено повідомляти про свої очікування і гарантувати, що всі працюють над досягненням одних і тих же цілей [21].

Результати наведені у таблиці Б.1

Таблиця Б.1 – Формалізація мети за технологією SMART

Конкретизація	Розробка вебплатформи підтримки діяльності фрілансерів
Вимірюваний	Результат допоможе скоротити час (5 днів) та ресурси (матеріальні) для організації надання послуг в різних сферах.
Досяжний	Проєкт реалізовується у відповідності до рівня досвіду та на основі затвердженого технічного завдання.
Значимість	Результат допоможе охопити більшу кількість аудиторії за допомогою спрощеної операції найму фрілансерів та надання робочих послуг.
Обмежений в часі	Проєкт виконується враховуючи встановлені на ранньому етапі обмеження в часі (травень 2024).

Джерело: розроблено автором

Work Breakdown Structure (WBS) – Ієрархічна структура робіт

Структура розподілу робіт (WBS) є стандартним будівельним блоком управління проектом. Вона є обов'язковою частиною EIA-748-B, оскільки орієнтована на продукт WBS визначає роботу, необхідну для завершення програми. WBS показує ієрархію програмних і проектних робіт і зазвичай використовується в програмах, які використовують водоспадний або прогнозований життєвий цикл проекту.

WBS розбиває ключові аспекти обсягу проекту. Структура розбиття робіт (WBS) ділить проект на менші робочі пакети або частини. Деякі гілки WBS можуть мати більше рівнів, ніж інші, за умови, що структура є внутрішньо узгодженою і зрозумілою для команди.

WBS для складних проектів може здаватися перевантаженим великою кількістю граф. При створенні WBS важливо включити супровідну документацію, яка містить додаткові деталі, такі як власник результатів, пов'язані з ними витрати та часові рамки для виконання пакету робіт. Рекомендується, щоб державні підрядники дотримувалися рекомендацій MIL-STD-881, щоб забезпечити створення всеосяжного та ефективного WBS [22].

На рисунку Б.1 представлено WBS проєкту щодо розробки web – платформи підтримки діяльності фрілансерів.

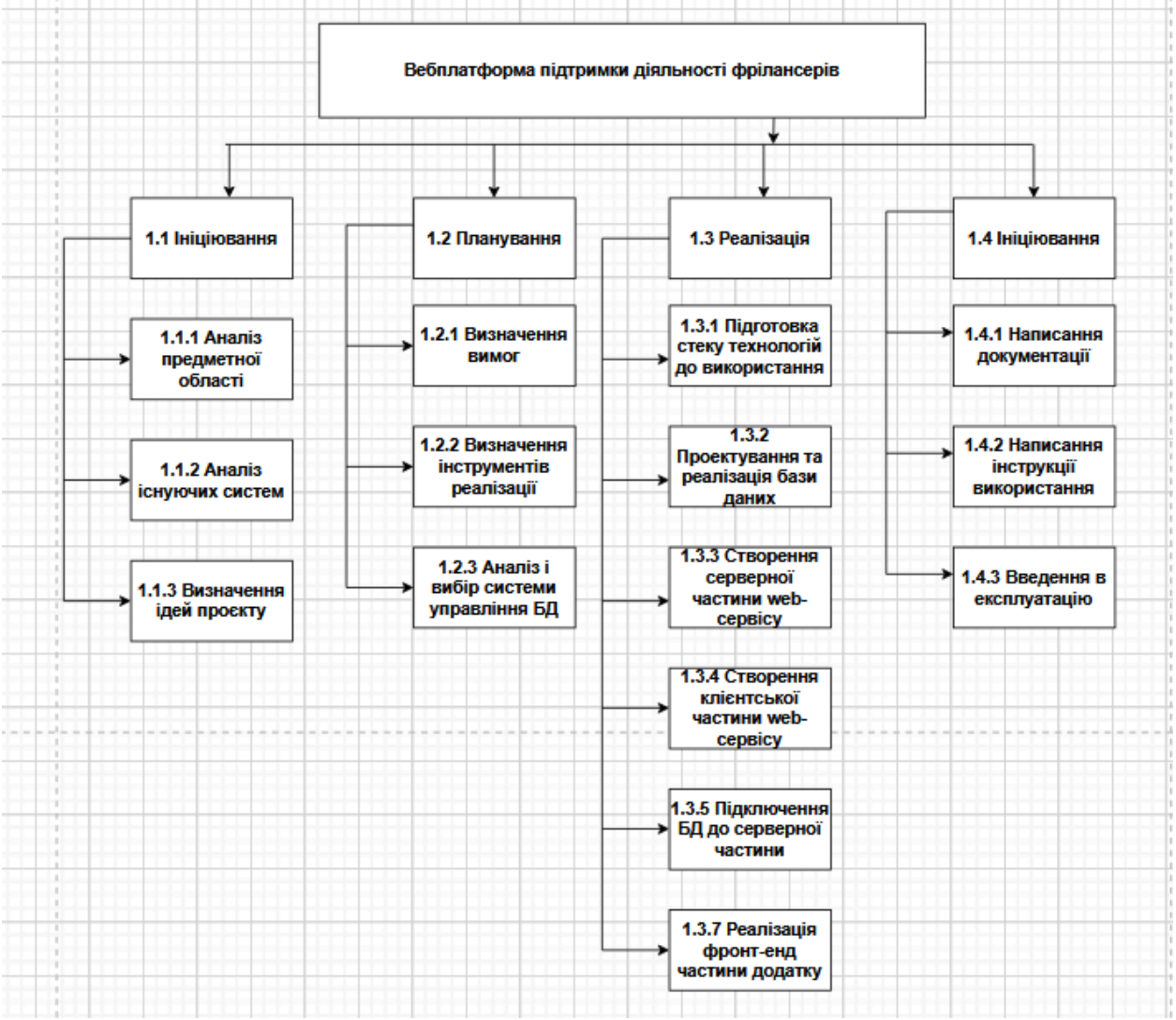

Рисунок Б.1 − WBS-структура робіт проекту

Джерело: розроблено автором

Організаційна структура робіт (OBS)

Структура OBS являє собою процес, який використовується для організації заходів і того, як вони допоможуть у повідомленні про проблеми, пов’язані з будівельним проектом. Це організаційна структура, заснована на відповідальності різних зацікавлених сторін проекту.

Будівельний проект - це складна операція, яка включає в себе різні сегменти, що складаються з різних видів діяльності. Структура розподілу

робіт (WBS) - це розбивка і сегментація всього будівельного проекту. Роботи, призначені в цих сегментах, називаються робочими пакетами [23].

Організаційна структура (OBS) має тісний зв’язок з WBS, оскільки вона визначає організацію, відповідальну за виконання певного сегменту в WBS. Цей взаємопов’язаний процес спирається на методологію матриці розподілу відповідальності (RAM) і допомагає у визначенні обсягу проекту.

Структура OBS ретельно розробляється керівником проекту з урахуванням цілей проекту, можливостей, наданих субпідрядникам, а також архітектурного та технічного персоналу. Зміни вносяться залежно від складності, обсягу та розміру проекту.

На рисунку Б.2 представлено OBS-структуру робіт проекту.

Діаграма Ганта

Діаграма Ганта - це інструмент управління проектами. Вона представлена візуально, щоб допомогти побачити хід виконання завдань. У стовпчику зліва ми знаходимо всі завдання, які потрібно виконати. У верхньому рядку відображаються часові одиниці проекту, про який йде мова. Це можуть бути дні, тижні або навіть місяці.

На діаграмі завдання представлені горизонтальною смугою. Положення цієї смуги на діаграмі та її довжина вказує на приблизну тривалість завдання, а також дати початку і закінчення [24].

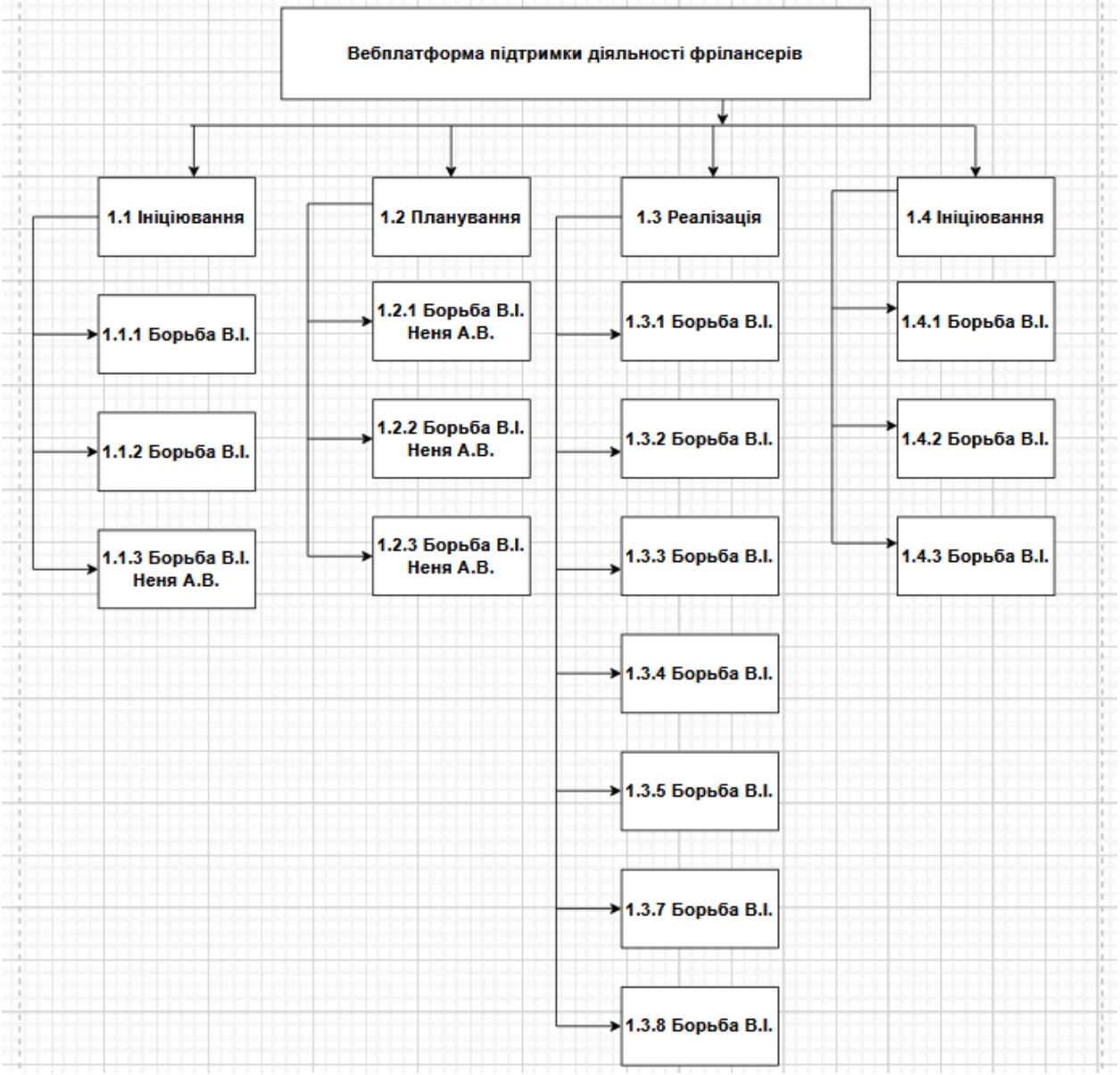

Рисунок Б.2 – OBS-структура робіт проекту

Джерело: розроблено автором

Діаграма складається з декількох основних елементів, таких як:

- Дати початку і закінчення проекту та завдання, які необхідно виконати.
- Визначені завдання та підзадачі.
- Попередній графік, щоб знати, яке завдання є пріоритетним.
- Залежності між завданнями.
- Загальний статус проекту.

Календарний план проекту показаний на рисунках Б.3 - Б.4.

Вебплатформа підтримки діяльності фрілансерів	46 дней	Пн 01.04.24	Сб 01.06.24			
Ініціалізація проекту	4 дней	Пн 01.04.24	Чт 04.04.24			
Опрацювання ТЗ	2 дней	Чт 04.04.24	Пт 05.04.24		Борьба В.І.	
Аналіз контенту	2 дней	Сб 06.04.24	Вс 07.04.24	3	Борьба В.І.	$10000
Пошук контенту	2 дней	Пн 08.04.24	Вт 09.04.24	4	Борьба В.І.	$1,116.00
Ідентифікація проекту	2 дней	Ср 10.04.24	Чт 11.04.24	5	Неня А.В.	$72.00
Планування проекту	**15 дней**	Пт 12.04.24	Чт 02.05.24	6		$72.00
Структура проекту	2 дней	Пт 12.04.24	Пн 15.04.24	7	Борьба В.І.	$72.00
Розробка дизайну та макету	2 дней	Вт 16.04.24	Ср 17.04.24	8	Борьба В.І.	$900.00
Роз календарного плану робка "Вебплатформа підтримки діяльності фрілансерів»	5 дней	Чт 18.04.24	Ср 24.04.24	9	Борьба В.І.	$1,432.00
Управління ресурсами	1 день	Чт 25.04.24	Чт 25.04.24	10	Борьба В.І.	$72.00
Управління фінансами	1 день	Пт 26.04.24	Пт 26.04.24	11	Борьба В.І.	$72.00
Управління якістю	1 день	Сб 27.04.24	Сб 27.04.24	12	Борьба В.І.	$100.00
Оптимізація плану	1 день	Вс 28.04.24	Вс 28.04.24	13	Борьба В.І.	$72.00
Затвердження плану	1 день	Пн 29.04.24	Пн 29.04.24	14	Неня А.В.	$72.00
Реалізація проекту	15 дней	Вт 30.04.24	Пн 20.05.24		Борьба В.І.	$7,020.00
Розробка вебплатформи	10 дней	Вт 30.04.24	Пн 13.05.24		Борьба В.І.	$3,250.00
Реалізація дизайну та макету	4 дней	Пн 13.05.24	Чт 16.05.24		Борьба В.І.	$1,300.00
Верстка	3 дней	Чт 16.05.24	Пн 20.05.24	18	Борьба В.І.	$1,500.00
Розробка сторінок	5 дней	Пн 20.05.24	Пт 24.05.24	19	Борьба В.І.	$350.00

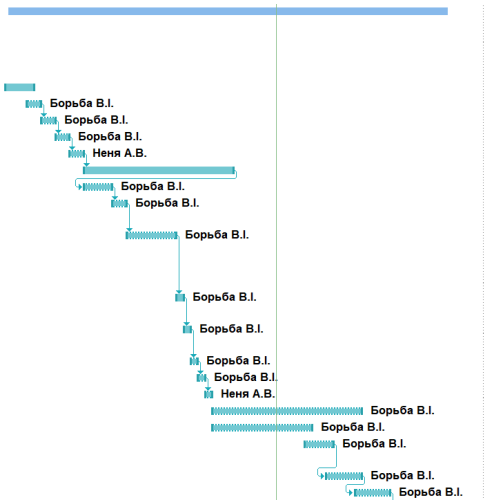

Рисунок Б.3 – Діаграма Ганта. Частина 1

Джерело: розроблено автором

Проектування БД	3 дней	Пт 24.05.24	Вт 28.05.24	20	Борьба В.І.	$100.00
Створення форм реєстрації	1 день	Вт 28.05.24	Вт 28.05.24	21	Борьба В.І.	$100.00
Створення зв'язку між користувачем та додатком	**4 дней**	Вт 28.05.24	Пт 31.05.24		Борьба В.І.	$3,270.00
Розробка модулю для додавання заявки на виконання завдання	1 день	Вт 28.05.24	Вт 28.05.24		Борьба В.І.	$270.00
Розробка модулю реєстрації нового користувача	1 день	Вт 28.05.24	Вт 28.05.24	24	Борьба В.І.	$1,500.00
Розробка модулю організації чату	2 дней	Вт 28.05.24	Ср 29.05.24	25	Борьба В.І.	$1,500.00
Реалізація зв'язку БД з додатком	2 дней	Ср 29.05.24	Чт 30.05.24	26	Борьба В.І.	$500.00
Тестування	2 дней	Чт 30.05.24	Пт 31.05.24	27	Борьба В.І.	$500.00
Завершення проекту	6 дней	Пт 31.05.24	Чт 06.06.24	28		$432.00
Створення документації	1 день	Пт 31.05.24	Пт 31.05.24	29	Борьба В.І.	$72.00
Створення інструкції користувача	2 дней	Пт 31.05.24	Сб 01.06.24	30	Борьба В.І.	$72.00
Запуск і підтримка	2 дней	Сб 01.06.24	Пн 03.06.24	31	Борьба В.І.	$72.00
Архівація проєкту	1 день	Пн 03.06.24	Пн 03.06.24	32	Борьба В.І.	$72.00

Рисунок Б.4 – Діаграма Ганта. Частина 2

Джерело: розроблено автором

Управління ризиками

Управління ризиками - це процес, спрямований на виявлення ризиків у бізнесі та створення методології для зменшення або усунення цих можливих ризиків. Ефективна система дозволяє підтримувати безпеку персоналу та захищати бізнес-активи.

Управління ризиками є фундаментальною частиною ефективного бізнес-планування, і організації повинні забезпечити захищене середовище для ефективної роботи своїх співробітників.

Управління ризиками - це основний лідерський підхід, який гарантує, що будь-які можливі ризики для успіху будуть розпізнані та керовані до того, як вони зруйнують ваш проект.

Надійний спосіб управління ризиками дозволяє членам команди з більшою ймовірністю вчасно повідомляти про труднощі в проекті. Практики управління ризиками дозволяють групі виявити проблеми задовго до їх виникнення.

Раннє усвідомлення потенційних ризиків означає, що відповідні особи можуть втрутитися, щоб пом’якшити проблему до того, як вона стане надто серйозною.

Управління ризиками до того, як вони стануть серйозними, має важливе значення для більш плавного, продуктивного та економічно ефективного ведення бізнесу [25].

Таблиця Б.2. Ймовірність виникнення і величина ризику

№	**Ризики**	**Виникнення**	**Втрати**
1	Незахищеність від SQL-ін’єкцій	2	4
2	Недостатній захист від атак типу XSS	3	3
3	Втрата доступу до даних через неавторизований доступ	3	5
4	Низька швидкість реагування на інциденти безпеки	2	3
5	Втрата даних через ненадійне резервне копіювання	4	5
6	Порушення авторських прав	2	3

Джерело: розроблено автором

Таблиця Б.3 – Матриця впливу

Вірогідність виникнення	Матриця впливу				
5			3	5	
4		4			
3		6	2		
2				1	
1					
Ступінь впливу	1	2	3	4	5

Джерело: розроблено автором

ДОДАТОК В

Лістинг коду ChatController.js

```
const User = require("../models/UserModel");
const Chat = require("../models/chatModel");
const Message = require("../models/messageModel");

const findChatById = async (chatId) => {
  const selectedChat = await Chat.findById(chatId);
  return selectedChat;
};

const findChat = async (senderId, receiverId) => {
  const selectedSender = await User.findById(senderId);
  const selectedReceiver = await User.findById(receiverId);
  if (selectedReceiver != null && selectedSender != null) {
    const selectedChat = await Chat.findOne({
      between: { $all: [senderId, receiverId] },
    });
    return selectedChat;
  }
  return "User doesn't exists";
};

const createChat = async (senderId, receiverId) => {
  const selectedSender = await User.findById(senderId);
  const selectedReceiver = await User.findById(receiverId);
  if (selectedReceiver != null && selectedSender != null) {
    const chatExists = await findChat(senderId, receiverId);
    if (chatExists == null) {
      const createdChat = await Chat.create({
        between: [senderId, receiverId],
      });
      return createdChat._id;
    }
    return chatExists._id;
  }
  return "User doesn't exists";
};

const userChats = async (userId) => {
  const selectedUser = await User.findById(userId);
  if (selectedUser) {
     const userChats = await Chat.find({ between: { $in: [userId]
} });
    if (userChats.length != 0) {
      let conversationUsersInfo = [];
      for (let i of userChats) {
         const chatWith = i.between[0] != userId ? i.between[0] :
i.between[1];
        const chatWithUserInfo = await User.findById(chatWith);
```

```
        const lastMessageWithUser = await Message.findOne({
          chatId: i._id,
        }).sort({ createdAt: -1 });
        conversationUsersInfo.push({
          _id: i._id,
          avatar: chatWithUserInfo.image,
          username: chatWithUserInfo.username,
          msg: lastMessageWithUser.text,
        });
      }
      return conversationUsersInfo;
    }
    return userChats;
  }
  return "User doesn't exists";
};

const getMessages = async (chatId, userId) => {
  const selectedUser = await User.findById(userId);
  const selectedChat = await findChatById(chatId);
  const chatTime = selectedChat.createdAt;
  if (selectedUser != null && selectedChat != null) {
    const withUser =
      selectedChat.between[0] != userId
        ? selectedChat.between[0]
        : selectedChat.between[1];
    const withUserInfo = await User.findById(withUser);
    const conversationMessages = await Message.find({ chatId });
    return {
      conversationMessages,
      withInfo: {
        _id: withUserInfo._id,
        avatar: withUserInfo.image,
        username: withUserInfo.username,
      },
      chatTime,
    };
  }
  return "Conversation doesn't exists";
};

const sendMessage = async (senderId, receiverId, text) => {
  const selectedSender = await User.findById(senderId);
  const selectedReceiver = await User.findById(receiverId);
  if (selectedReceiver != null && selectedSender != null) {
          const  conversationId  =  await  createChat(senderId,
receiverId);
    const createdMessage = await Message.create({
      chatId: conversationId,
      senderId,
      text,
    });
    return "Message Sent Successfully";
```

```
  }
  return "User doesn't exists";
};

module.exports = {
  userChats,
  getMessages,
  sendMessage,
};
```

Лістинг коду OrderController.js

```
const Order = require("../models/orderModel");
const          {          getServiceRating          }          =
require("./TestimonialsController");
const { findServiceById } = require("./ServicesController");
const { findUserById } = require("./UserController");
const { sendMessage } = require("./ChatController");

const findOrder = async (orderId) => {
  const selectedOrder = Order.findById(orderId);
  return selectedOrder;
};

const findClientOrders = async (clientId) => {
  const selectedClient = await findUserById(clientId);
  if (selectedClient) {
    if (selectedClient.role != "client") {
      return "You Don't Have Permission";
    }
     const clientOrders = await Order.find({ clientId }).sort({
updatedAt: -1 });
    if (clientOrders.length != 0) {
      let allOrdersInfo = [];
      for (let i of clientOrders) {
                                const    serviceInfo    =    await
findServiceById(i.serviceId.toString());
                               const    serviceRating    =    await
getServiceRating(i.serviceId.toString());
                              const    serviceUserInfo    =    await
findUserById(serviceInfo.userId);
        const ordersInfo = {
          serviceInfo,
          serviceRating,
          serviceUserInfo,
          status: i.status,
          _id: i._id,
        };
        allOrdersInfo.push(ordersInfo);
      }
      return allOrdersInfo;
```

```
    }
    return [];
  }
  return "User Doesn't Exists";
};

const findClientOrder = async (clientId, orderId) => {
  const selectedClient = await findUserById(clientId);
  if (selectedClient) {
    if (selectedClient.role != "client") {
      return "You Don't Have Permission";
    }
    const clientOrder = await findOrder(orderId);
    if (clientOrder) {
      const serviceInfo = await findServiceById(
        clientOrder.serviceId.toString()
      );
      const serviceRating = await getServiceRating(
        clientOrder.serviceId.toString()
      );
                          const    serviceUserInfo    =    await
findUserById(serviceInfo.userId);
      const orderInfo = {
        serviceInfo,
        serviceRating,
        serviceUserInfo,
        status: clientOrder.status,
        _id: clientOrder._id,
      };
      return orderInfo;
    }
    return "Order Doesn't Exists";
  }
  return "User Doesn't Exists";
};

const makeOrder = async (clientId, serviceId) => {
  const selectedClient = await findUserById(clientId);
  if (selectedClient) {
    if (selectedClient.role != "client") {
      return "You Don't Have Permission";
    }
    const selectedService = await findServiceById(serviceId);
    if (selectedService) {
      const orderExists = await Order.find({
        clientId: selectedClient._id,
        serviceId: selectedService._id,
        status: "OnGoing",
      });
      if (orderExists.length != 0) {
          return "You Already Have A Uncompleted Order For This
Service";
      }
```

```
              const text = `Hello,I would like to order
${selectedService.title} service`;
      sendMessage(clientId, selectedService.userId, text);
      const createdOrder = Order.create({
        clientId: selectedClient._id,
        serviceId: selectedService._id,
      });
      return "Order Made Successfully";
    }
    return "Service Doesn't Exists";
  }
  return "User Doesn't Exists";
};

const updateOrder = async (clientId, orderId, orderState) => {
  const selectedClient = await findUserById(clientId);
  if (selectedClient) {
    if (selectedClient.role != "client") {
      return "You Don't Have Permission";
    }
    const selectedOrder = await findOrder(orderId);
    if (selectedOrder) {
      if (selectedOrder.clientId.toString() != clientId) {
        return "You Don't Have Permission";
      }
            if (orderState != "Completed" && orderState !=
"Cancelled") {
        return "Order Status Unrecognized";
      }
      const updatedOrder = Order.updateOne(
        { clientId, _id: orderId, status: "OnGoing" },
        {
          status: orderState,
        }
      );
      return updatedOrder;
    }
    return "Order doesn't exists";
  }
  return "User doesn't exists";
};

module.exports = {
  findClientOrder,
  findClientOrders,
  makeOrder,
  updateOrder,
  findOrder,
};
```

Лістинг коду UserController.js

```
require("dotenv").config();
const User = require("../models/UserModel");
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");
const { existsSync, unlinkSync } = require("fs");

const userExists = async (email) => {
  const selectedUser = await User.findOne({ email });
  return selectedUser;
};
const findUsers = async () => {
  const allUsers = await User.find();
  return allUsers;
};
const findUserById = async (id) => {
  const selectedUser = await User.findById(id);
  if (selectedUser) return selectedUser;
  else return null;
};
const registerUser = async (
  fullName,
  age,
  email,
  username,
  password,
  image,
  role
) => {
  const allUsers = await findUsers();
  const userExists = allUsers.find(
    (e) => e.email == email || e.username == username
  );
  if (userExists) {
    return null;
  } else {
    const salt = await bcrypt.genSalt(10);
    const newPassword = await bcrypt.hash(password, salt);
    let newImage;
    if (image) {
      newImage = image;
    } else {
      newImage = "no-image.png";
    }

    const createdUser = await User.create({
      fullName,
      age,
      email,
      username,
      password: newPassword,
      role,
```

```
      image: newImage,
    });
    return createdUser;
  }
};
const loginUser = async (username, password) => {
  const allUsers = await findUsers();
    const userExists = allUsers.find((e) => e.username ==
username);
  if (userExists) {
           const result = await bcrypt.compare(password,
userExists.password);
    if (result) {
          const token = jwt.sign({ userId: userExists._id },
process.env.SECRET);
      return {
        token,
        user: userExists,
      };
    }
    return 1;
  } else {
    return null;
  }
};

const updateUser = async (
  userId,
  fullName,
  age,
  username,
  image,
  imageFile
) => {
  const selectedUser = await findUserById(userId);
  if (selectedUser) {
    let newImage;
    if (image == undefined && imageFile == undefined) {
      image = selectedUser.image;
        } else if (image == "no-image.png" && imageFile ==
undefined) {
                                                          if
(existsSync(`./uploads/Users_imgs/${selectedUser.image}`)) {

unlinkSync(`./uploads/Users_imgs/${selectedUser.image}`);
      }
      newImage = "no-image.png";
    } else if (image == null && imageFile != null) {
                                                          if
(existsSync(`./uploads/Users_imgs/${selectedUser.image}`)) {

unlinkSync(`./uploads/Users_imgs/${selectedUser.image}`);
      }
```

```
      newImage = imageFile;
    }
    const updatedUser = User.updateOne(
      { _id: selectedUser._id },
      {
        fullName,
        age,
        username,
        image: newImage,
      }
    );
    return updatedUser;
  } else {
    return null;
  }
};
module.exports = {
  userExists,
  findUserById,
  findUsers,
  registerUser,
  loginUser,
  updateUser,
};
```

ДОДАТОК Г

Лістинг коду ChatRoutes.js

```
const express = require("express");
const {
  userChats,
  sendMessage,
  getMessages,
} = require("../controllers/ChatController");
const VerifyToken = require("../middleware/Auth");
const route = express.Router();

route.get("/all", VerifyToken, async (req, res) => {
  try {
    const userConversation = await userChats(req.userId);
    if (userConversation == "User doesn't exists") {
      return res.json({ status: 404, msg: userConversation });
    }
    return res.json({ status: 200, userConversation });
  } catch (error) {
      return res.json({ status: 505, msg: "Error Occured: " +
error.message });
  }
});

route.get("/messages/:chatId", VerifyToken, async (req, res) =>
{
  try {
       const messages = await getMessages(req.params.chatId,
req.userId);
    if (messages == "Conversation doesn't exists") {
      return res.json({ status: 404, msg: messages });
    }
    return res.json({ status: 200, messages });
  } catch (error) {
      return res.json({ status: 505, msg: "Error Occured: " +
error.message });
  }
});

route.post("/sendMessage", VerifyToken, async (req, res) => {
  try {
    const createdConversation = await sendMessage(
      req.userId,
      req.body.receiver,
      req.body.text
    );
    if (createdConversation == "User doesn't exists") {
        return res.json({ status: 404, msg: createdConversation
});
    }
```

```
    return res.json({ status: 200, msg: createdConversation });
  } catch (error) {
      return res.json({ status: 505, msg: "Error Occured: " +
error.message });
  }
});

module.exports = route;
```

Лістинг коду ClientRoutes.js

```
const express = require("express");
const          {          clientDashboard          }          =
require("../controllers/DashboardController");
const {
  findClientOrders,
  makeOrder,
  updateOrder,
  findClientOrder,
} = require("../controllers/OrdersController");
const {
  findUsersServices,
  findServiceById,
} = require("../controllers/ServicesController");
const          {          createTestimonial          }          =
require("../controllers/TestimonialsController");
const           {           findUserById           }           =
require("../controllers/UserController");
const VerifyToken = require("../middleware/Auth");
const route = express.Router();

route.get("/dashboard", VerifyToken, async (req, res) => {
  try {
    const dashboard = await clientDashboard(req.userId);
    if (dashboard == "User doesn't exist") {
      return res.json({ status: 404, msg: dashboard });
    } else if (dashboard == "You Don't Have Permission") {
      return res.json({ status: 403, msg: dashboard });
    } else {
      return res.json({ status: 200,  dashboard });
    }
  } catch (error) {
      return res.json({ status: 505, msg: "Error Occured: " +
error.message });
  }
});

route.get("/allServices", VerifyToken, async (req, res) => {
  try {
    const allServices = await findUsersServices();
    return res.json({ allServices, status: 200 });
  } catch (error) {
```

```
      return res.json({ status: 505, msg: "Error Occured: " +
error.message });
  }
});

route.get("/service/:idService", VerifyToken, async (req, res)
=> {
  try {
                        const     selectedService     =     await
findServiceById(req.params.idService);
    if (selectedService) {
                         const    selectedServiceUser    =    await
findUserById(selectedService.userId);
      const serviceInfo = {
        ...selectedService._doc,
        userId: selectedServiceUser,
      };
         return res.json({ selectedService: serviceInfo, status:
200 });
    }
    return res.json({ msg: "Service Not Found", status: 404 });
  } catch (error) {
      return res.json({ status: 505, msg: "Error Occured: " +
error.message });
  }
});

route.get("/orders", VerifyToken, async (req, res) => {
  try {
    const clientOrders = await findClientOrders(req.userId);
    if (clientOrders == "User doesn't exists") {
      return res.json({ status: 404, clientOrders });
    }
    if (clientOrders == "You don't have permission") {
      return res.json({ status: 403, clientOrders });
    }
    return res.json({ status: 200, clientOrders });
  } catch (error) {
      return res.json({ status: 505, msg: "Error Occured: " +
error.message });
  }
});

route.get("/order/:idOrder", VerifyToken, async (req, res) => {
  try {
    const clientOrderInfo = await findClientOrder(
      req.userId,
      req.params.idOrder
    );
    if (
      clientOrderInfo == "User doesn't exists" ||
      clientOrderInfo == "Order Doesn't Exists"
    ) {
```

```
      return res.json({ status: 404, clientOrderInfo });
    }
    if (clientOrderInfo == "You don't have permission") {
      return res.json({ status: 403, clientOrderInfo });
    }
    return res.json({ status: 200, clientOrderInfo });
  } catch (error) {
      return res.json({ status: 505, msg: "Error Occured: " +
error.message });
  }
});

route.post("/order", VerifyToken, async (req, res) => {
  try {
        const createdService = await makeOrder(req.userId,
req.body.serviceId);
    if (
      createdService == "Service Doesn't Exists" ||
      createdService == "User Doesn't Exists"
    ) {
      return res.json({ status: 404, msg: createdService });
    }
    if (createdService == "You Don't Have Permission") {
      return res.json({ status: 403, msg: createdService });
    }
    if (
       createdService == "You Already Have A Uncompleted Order
For This Service"
    ) {
      return res.json({ status: 400, msg: createdService });
    }
    return res.json({ status: 200, msg: createdService });
  } catch (error) {
      return res.json({ status: 505, msg: "Error Occured: " +
error.message });
  }
});

route.put("/order/:orderId", VerifyToken, async (req, res) => {
  try {
    const updatedOrder = await updateOrder(
      req.userId,
      req.params.orderId,
      req.body.status
    );
    if (
      updatedOrder == "Service Doesn't Exists" ||
      updatedOrder == "User Doesn't Exists"
    ) {
      return res.json({ status: 404, msg: updatedOrder });
    }
    if (updatedOrder == "You Don't Have Permission") {
      return res.json({ status: 403, msg: updatedOrder });
```

```
    }
    if (updatedOrder == "Order Status Unrecognized") {
      return res.json({ status: 400, msg: updatedOrder });
    }
    if (updatedOrder.modifiedCount == 1) {
          return res.json({ status: 200, msg: "Order Updated
Successfully" });
    } else {
       return res.json({ status: 409, msg: "Cannot Update Order
Again" });
    }
  } catch (error) {
      return res.json({ status: 505, msg: "Error Occured: " +
error.message });
  }
});

route.post("/testimonial/:orderId", VerifyToken, async (req,
res) => {
  try {
    const createdTestimonial = await createTestimonial(
      req.userId,
      req.params.orderId,
      req.body
    );
    if (
      createdTestimonial == "User doesn't exists" ||
      createdTestimonial == "Service doesn't exists"
    ) {
      return res.json({ status: 404, msg: createdTestimonial });
    }
    if (createdTestimonial == "You Don't Have Permission") {
      return res.json({ status: 403, msg: createdTestimonial });
    }
    return res.json({ status: 200, msg: createdTestimonial });
  } catch (error) {
      return res.json({ status: 505, msg: "Error Occured: " +
error.message });
  }
});

module.exports = route;
```

Лістинг коду FreelanceRoutes.js

```
const express = require("express");
const { freelancerDashboard } =
require("../controllers/DashboardController");
const {
  findUserServices,
  findServiceById,
  createService,
```

```
  deleteService,
  updateService,
} = require("../controllers/ServicesController");
const VerifyToken = require("../middleware/Auth");
const          {          createServiceUpload          }          =
require("../middleware/uploadImage");
const route = express.Router();

route.get("/dashboard", VerifyToken, async (req, res) => {
  try {
    const dashboard = await freelancerDashboard(req.userId);
    if (dashboard == "User doesn't exist") {
      return res.json({ status: 404, msg: dashboard });
    } else if (dashboard == "You Don't Have Permission") {
      return res.json({ status: 403, msg: dashboard });
    } else {
      return res.json({ status: 200, dashboard });
    }
  } catch (error) {
      return  res.json({  status:  505,  msg:  "Error  Occured:  " +
error.message });
  }
});

route.get("/myServices", VerifyToken, async (req, res) => {
  try {
    const allServices = await findUserServices(req.userId);
    if (allServices) {
      return res.json({ allServices, status: 200 });
    }
     return  res.json({  msg:  "User  Doesn't  Exists",  status:  404
});
  } catch (error) {
      return  res.json({  status:  505,  msg:  "Error  Occured:  " +
error.message });
  }
});

route.get("/service/:idService",  VerifyToken,  async  (req,  res)
=> {
  try {
                       const     selectedService     =     await
findServiceById(req.params.idService);
    if (selectedService) {
      return res.json({ selectedService, status: 200 });
    }
    return res.json({ msg: "Service Not Found", status: 404 });
  } catch (error) {
      return  res.json({  status:  505,  msg:  "Error  Occured:  " +
error.message });
  }
});
```

```
route.post("/service", VerifyToken, createServiceUpload, async
(req, res) => {
  try {
    if (req.files.length == 0) {
      return res.json({
        msg: "You should select at least 3 images",
        status: 400,
      });
    }
    const images = req.files.map((image) => image.filename);
    const { title, description, price } = req.body;
    const createdService = await createService(
      title,
      description,
      price,
      req.userId,
      images
    );
    if (createdService) {
        return res.json({ msg: "Service Created Successfully",
status: 200 });
    }
    return res.json({
      msg: "You Already Have This Service Gig",
      status: 409,
    });
  } catch (error) {
      return res.json({ status: 505, msg: "Error Occured: " +
error.message });
  }
});

route.put(
  "/service/:idService",
  VerifyToken,
  createServiceUpload,
  async (req, res) => {
    try {
      if (req.files.length == 0) {
        return res.json({
          msg: "You should select at least 3 images",
          status: 400,
        });
      }
      const images = req.files.map((image) => image.filename);
      const { title, description, price } = req.body;
      const updatedService = await updateService(
        title,
        description,
        price,
        req.userId,
        images,
        req.params.idService
```

```
      );
      switch (updatedService) {
        case "User Doesn't exists":
        case "Service doesn't exists":
          return res.json({ msg: updatedService, status: 404 });
        case "This service doesn't belongs to you":
          return res.json({ msg: updatedService, status: 403 });
        case "Service gig already exists":
          return res.json({ msg: updatedService, status: 409 });
        default:
          return res.json({ msg: updatedService, status: 200 });
      }
    } catch (error) {
        return res.json({ status: 505, msg: "Error Occured: " +
error.message });
    }
  }
);

route.delete("/service/:idService", VerifyToken, async (req,
res) => {
  try {
    const deletedService = await deleteService(
      req.userId,
      req.params.idService
    );
    if (deletedService) {
      if (deletedService == 1) {
           return res.json({ status: 404, msg: "Service doesn't
exists" });
      } else if (deletedService == -1) {
        return res.json({
          status: 403,
          msg: "This service doesn't belongs to you",
        });
      }
         return res.json({ status: 200, msg: "Service Deleted
Successfully" });
    }
     return res.json({ status: 404, msg: "User Doesn't exists"
});
  } catch (error) {
      return res.json({ status: 505, msg: "Error Occured: " +
error.message });
  }
});

module.exports = route;
```

Лістинг коду UserRoutes.js

```
const express = require("express");
```

```
const route = express.Router();
const VerifyToken = require("../middleware/Auth");
const {
  registerUser,
  findUserById,
  updateUser,
  loginUser,
} = require("../controllers/UserController");
const {
  createProfileUploadImage,
  updateProfileUploadImage,
} = require("../middleware/uploadImage");

route.post("/register", createProfileUploadImage, async (req,
res) => {
  const image = req.file && req.file.filename;
   const { fullName, age, email, username, password, role } =
req.body;
  try {
    const createdUser = await registerUser(
      fullName,
      age,
      email,
      username,
      password,
      image,
      role
    );
    if (createdUser) {
          return res.json({ msg: "User Created Successfully",
status: 200 });
    }
     return res.json({ msg: "Username Or Email Already Exists",
status: 403 });
  } catch (err) {
      return res.json({ msg: "Error Occured: " + err.message,
status: 505 });
  }
});

route.post("/login", async (req, res) => {
  try {
    const { username, password } = req.body;
         const authentifiedUser = await loginUser(username,
password);
    if (authentifiedUser) {
      if (authentifiedUser == 1) {
         return res.json({ msg: "Wrong Credentials", status: 401
});
      } else {
        return res.json({
          msg: "Logged Successfully",
          token: authentifiedUser.token,
```

```
          userInfo: authentifiedUser.user,
          status: 200,
        });
      }
    }
      return res.json({ msg: "User Doesn't Exists", status: 404
});
  } catch (err) {
      return res.json({ msg: "Error Occured: " + err.message,
status: 505 });
  }
});

route.put(
  "/update",
  VerifyToken,
  updateProfileUploadImage,
  async (req, res) => {
    const imageFile = req.file && req.file.filename;
    const { fullName, age, username, image } = req.body;
    const updatedUser = await updateUser(
      req.userId,
      fullName,
      age,
      username,
      image,
      imageFile
    );
    const updatedUserInfo = await findUserById(req.userId);
    try {
      if (updatedUser) {
        return res.json({
          msg: "User Updated Successfully",
          updatedUserInfo,
          status: 200,
        });
      } else {
          return res.json({ msg: "User Doesn't Exists", status:
404 });
      }
    } catch (err) {
        return res.json({ msg: "Error Occured: " + err.message,
status: 505 });
    }
  }
);

module.exports = route;
```