

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

на тему:

**«Розробка платформи для організації та
виконання завдань за винагороду з вбудованим
чатом»**

Виконав студент групи ПЗ-20-1
ВЕРБОВСЬКИЙ Олександр Юрійович

Керівник роботи:
ЛОКТИКОВА Тамара Миколаївна

Рецензент:
ЄФРЕМОВ Юрій Миколайович

Житомир – 2024

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

«ЗАТВЕРДЖУЮ»
Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк
«23» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ВЕРБОВСЬКИЙ Олександр Юрійович**
Керівник роботи: **ЛОКТИКОВА Тамара Миколаївна**
Тема роботи: **«Розробка платформи для організації та виконання завдань за винагороду з вбудованим чатом»**,
затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р., №74/с**
Вихідні дані для роботи: **об'єктом дослідження є процеси для організації та виконання завдань за винагороду, а предметом дослідження є засоби, алгоритми та методи, що забезпечують ефективну роботу платформи, а також взаємодію між користувачами під час створення, виконання та оплати завдань.**
Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Локтікова Т.М.	23.02.2024р.	23.02.2024р.
2	Локтікова Т.М.	23.03.2024р.	23.03.2024р.
3	Локтікова Т.М.	13.05.2024р.	13.05.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебдодатка платформи та пояснювальної записки. Пояснювальна записка до випускної роботи містить текстову частину, викладену на 111 сторінках друкованого тексту. Список використаних джерел містить 33 найменувань і займає 3 сторінки. У роботі наведено 48 рисунків та 41 таблиця. Також у роботі є 9 сторінок додатків. Загальний обсяг роботи – 120 сторінок.

Метою роботи є створення інтегрованої та зручної платформи, яка забезпечить ефективне виконання завдань за винагороду, сприятиме взаємодії користувачів та забезпечить безпеку та надійність фінансових операцій.

Під час аналізу можливих архітектурних рішень для платформи обміну послугами та завдань було визначено, що клієнт-серверна архітектура є оптимальним вибором. Дана архітектура забезпечує ефективну взаємодію з сервером, можливість асинхронної роботи та чітке відокремлення логіки додатка та інтерфейсу.

Для розробки платформи було використано сучасні вебтехнології та інструменти – React, Node.js, Bootstrap5 та MySQL. Вони забезпечують ефективність, масштабованість та зручність у користуванні додатка.

Застосунок пропонує зручний інтерфейс, який забезпечує простий процес створення завдань, узгодження умов їхнього виконання, проведення фінансових операцій з реальними коштами користувачів та можливість взаємодії з підтримкою.

Платформа пройшла усі етапи тестування і готова до впровадження.

КЛЮЧОВІ СЛОВА: ВЕБЗАСТОСУНОК, КЛІЄНТ-СЕРВЕР, ЧАТ, JAVASCRIPT, REACT.

					ІПЗ.КР.Б-121-24-ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Вербовський О.Ю.			Розробка платформи для організації та виконання завдань за винагороду з вбудованим чатом. Пояснювальна записка	Літ.	Арк.
Керівник		Локтікова Т.М.					3
						Аркушів	
Н. контр.		Тичина О.П.					
Зав. каф.		Вакалюк Т.А.				Житомирська політехніка, група ІПЗ-20-1	

ABSTRACT

The Bachelor's qualification work consists of a web application platform and an explanatory note. The explanatory note for the thesis contains the textual part, presented in 111 pages of printed text. The list of references includes 33 titles and occupies 3 pages. The work contains 48 figures and 41 tables. There are also 9 pages of appendices in the work. The total volume of the work is 120 pages.

The goal of the work is to create an integrated and user-friendly platform that ensures effective task completion for rewards, promotes user interaction, and guarantees the security and reliability of financial transactions.

During the analysis of possible architectural solutions for the service and task exchange platform, it was determined that the client-server architecture is the optimal choice. This architecture ensures effective interaction with the server, the possibility of asynchronous operation, and a clear separation of application logic and interface.

Modern web technologies and tools were used to develop the platform – React, Node.js, Bootstrap 5, and MySQL. They ensure the efficiency, scalability, and ease of use of the application.

The application offers a user-friendly interface, featuring a simple process for creating tasks, agreeing on their terms, conducting financial transactions with real user funds, and interacting with support.

The product has passed all stages of testing and is ready for market launch.

KEYWORDS: CHAT, CLIENT-SERVER, JAVASCRIPT, REACT, WEB APPLICATION.

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1 АНАЛІЗ НАПРЯМІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ПЛАТФОРМИ.....	9
1.1 Постановка задачі	9
1.2 Аналіз аналогів програмного продукту	11
1.3 Вибір архітектури веборієнтованої платформи	15
1.4 Обґрунтування вибору засобів реалізації та вимоги до апаратного забезпечення	19
Висновки до першого розділу.....	23
РОЗДІЛ 2 ПРОЄКТУВАННЯ ПЛАТФОРМИ.....	24
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи	24
2.2 Розробка бази даних системи	47
2.3 Проєктування та реалізація алгоритмів системи	58
2.4 Реалізація платформи	66
Висновки до другого розділу.....	86
РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ ДЛЯ ОРГАНІЗАЦІЇ ТА ВИКОНАННЯ ЗАВДАНЬ.....	87
3.1 Порядок встановлення та налаштування параметрів платформи	87
3.2 Структура інтерфейсу платформи.....	90
3.3 Тестування платформи	102
Висновки до третього розділу	104
ВИСНОВКИ	106
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	108
ДОДАТКИ.....	111

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

АПЗ – архітектура програмного забезпечення

API – інтерфейс застосунків

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

MVC – архітектурний шаблон, який розділяє додаток на три основні компоненти: Model, View, Controller

MVVM – архітектурний шаблон, який розділяє додаток на три основні компоненти: Model, View, ViewModel

CRUD – основні операції, які використовуються для управління даними (створення, читання, оновлення, видалення)

РРЧ – режим реального часу

SSD – Solid State Drive

HDD – Hard Disk Drive

SPA – Single Page Applications

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми пов'язана зі стрімким зростанням популярності вебтехнологій, які стимулюють створення різних цікавих платформ, наприклад, для ефективної організації та виконання завдань за винагороду. Це особливо актуально у сучасному світі, де люди зіткнулися з проблемами у виконанні звичних завдань, які потребують фізичної присутності у конкретних місцях, через вимушену міграцію або інші проблеми, пов'язані з війною чи постпандемійним періодом.

Платформа, яка дозволяє виконувати завдання за винагороду, буде дуже популярною, оскільки вона буде сприяти виконанню завдань різного характеру та складності, а також наданню дистанційної підтримки. Цей підхід ефективно зберігатиме час і ресурси, забезпечуючи при цьому високий рівень здачі задач.

Така платформа стимулюватиме підтримку та співпрацю між людьми, незалежно від їхнього місця перебування, що буде спрощувати життя та допомагатиме зберегти важливі соціальні зв'язки, навіть коли вони фізично неможливі.

Метою даної розробки є створення сучасної та зручної платформи, що дозволить користувачам виконувати різноманітні завдання за винагороду, забезпечуючи при цьому надійну та ефективну комунікацію між виконавцями та замовниками. Важливим аспектом є створення умов для безпечного та прозорого виконання фінансових транзакцій, а також підтримка високого рівня безпеки даних користувачів. Розробка платформи також включає впровадження інтуїтивно зрозумілого інтерфейсу, що сприятиме комфортній роботі користувачів та оптимізуватиме процес взаємодії на платформі.

Для реалізації поставленої мети потрібно виконати наступні **завдання**:

1. Провести аналіз аналогів платформи, дослідити наявні рішення, їх функціональність, переваги та недоліки.

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

2. Здійснити вибір засобів проектування, ретельно оцінюючи доступні інструменти та технології.

3. Розробити платформу з використанням вибраних засобів, створити архітектуру та основні компоненти системи, інтегрувати всі необхідні функції та модулі для забезпечення повної працездатності платформи.

4. Провести тестування платформи різними методиками для зменшення ймовірності помилки.

Об’єктом дослідження є процеси для організації та виконання завдань за винагороду, які включають різноманітні функції, такі як створення завдань, їх прийняття до виконання, узгодження умов, вирішення конфліктів та проведення фінансових транзакцій. Особлива увага приділяється тому, як користувачі взаємодіють із платформою, управлінню завданнями, комунікації між ними, а також забезпеченню надійності та безпеки системи.

Предметом дослідження є засоби, алгоритми та методи, які забезпечують ефективну роботу платформи, а також взаємодію між користувачами під час створення, виконання та оплати завдань. Це включає дослідження способів оптимізації алгоритмів обробки даних, забезпечення швидкості та надійності фінансових транзакцій, а також методів підтримки високого рівня безпеки та захисту даних користувачів. Крім того, предметом дослідження є розробка і впровадження зручного та інтуїтивного інтерфейсу, який дозволить користувачам легко взаємодіяти з платформою та отримувати максимальну користь від її використання.

За темою випускної кваліфікаційної роботи бакалавра були подані та прийняті до публікації тези: Вербовський О.Ю., Локтікова Т.М., Лисогор Ю.І. Платформа для організації та виконання завдань з вбудованим чатом. Всеукраїнська наукова-практична on-line конференція здобувачів вищої освіти і молодих учених, присвячена Дню науки, 13-14 травня 2024 року.

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1 АНАЛІЗ НАПРЯМІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ПЛАТФОРМИ

1.1 Постановка задачі

На сьогодні, в умовах війни, постпандемійного періоду та інших ситуацій, які впливають на рішення людей щодо зміни місця перебування на тривалий термін, дуже складно виконати якісь прості дії самотужки, бо для цього потрібно витратити багато сил, коштів, енергії та інших ресурсів, щоб дібратись до пункту призначення. Наприклад, складно привітати свою подругу чи друга з якоюсь подією в червні, подарувавши їм букетик ромашок та кавун, без допомоги друзів, які знаходяться в одному місті з цією особою.

Цей проєкт має на меті полегшити та зробити більш доступними такі види комунікації та взаємодії в умовах глобальної нестабільності. Дана платформа дозволить людям як віддалено обмінюватися реальними матеріальними подарунками, так і просто: співати гарно пісні, розповідати виразно вірші чи передавати пакет продуктів рідній бабусі, яку вже не бачив декілька років, не маючи змоги приїхати та допомогти їй.

Тож одним з найважливіших моментів продукту є забезпечення можливість віртуальної присутності та активної участі у подіях, навіть якщо фізично люди знаходяться далеко від потрібної локації. Така платформа сприятиме відновленню та зміцненню зв'язків між людьми в умовах сучасної нестабільності та важливості віддаленої спільності.

Крім того, враховуючи сучасні технології та потреби користувачів, інтерфейс продукту повинен бути інтуїтивним та ефективним для користувачів будь-якої вікової категорії та будь-якого досвіду у використанні гаджетів та Інтернету, який дозволить навіть бабусям та дідусям, у похилому віці, передавати своїм онукам приємні враження та висловлювати теплі слова через віртуальну присутність.

Також, в контексті розробки проєкту, важливим аспектом є впровадження чату у РРЧ з розширеними можливостями надсилання

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

різноманітного контенту. Це обґрунтовано необхідністю встановлення надійного та швидкого зв'язку між замовником та виконавцем завдання. Реальний час дозволяє інтерактивно обмінюватися думками, реакціями та динамічно втручатися у виконання завдання на різних етапах, та змінювати його напрям, тривалість і якість, тим самим надаючи повний контроль замовнику над задачею створюючи відчуття, наче він сам і є виконавцем.

Ще одним важливим аспектом чату є можливість надсилення фото, відео та інших файлів в межах чату, адже це є необхідним для розширення спектра виразних можливостей та покращення комунікації. Відправлення зображень та відео дозволяє виконавцям краще розуміти, що відбувається і що від них вимагається, а замовникам – переконуватись у якості виконання задачі, детальнішому поглибленні у якусь непередбачену проблему та бути “тут і зараз” в будь-якій точці світу та о будь-якій годині доби.

Крім цього, у проєкті акцентується увага на технічних інноваціях, спрямованих на забезпечення найвищого рівня зручності та функціональності для користувачів. Платформа побудована з використанням передових технологій, що дозволяють забезпечити надійний та швидкий обмін повідомленнями, надійне виконання операцій з коштами, та збереження даних для подальшого відтворення у зв'язку з різними причинами.

Також однією з найважливіших складових платформи є система коментарів як виконавців, так і замовників, оскільки вона сприяє взаємодії, довірі та взаєморозумінні між користувачами. Крім цього, система коментарів надає можливість користувачам висловлювати свої думки та враження від співпраці. Замовники, переглядаючи відгуки, можуть отримати об'єктивне уявлення про професіоналізм та надійності виконавців. Рейтинг та відгуки формують довіру, яка є критичним аспектом в прийнятті рішення, адже на основі цього у замовника та виконавця формується перша про опонента.

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

1.2 Аналіз аналогів програмного продукту

На момент розробки платформи проведено ретельний аналіз та порівняння з наявними аналогами, які спрямовані на подібне призначення, функціонал та мають схожу цільову аудиторію замовників. Цими програмами виявились Upwork, Fiverr та TaskRabbit, які вже зарекомендували себе на ринку як визнані лідери у сфері фрілансу та надання послуг.

Upwork – це додаток, визнаний своєю орієнтацією на фріланс та надає можливість замовникам шукати виконавців для виконання різноманітних завдань, починаючи від програмування та дизайну до письмових послуг та перекладу [1]. Платформа дозволяє створювати проєкти, розміщувати завдання та взаємодіяти з фрілансерами через вбудований чат. Оплата за виконану роботу здійснюється через систему електронного платежу, яка забезпечує безпечні та ефективні транзакції між замовниками та фрілансерами.

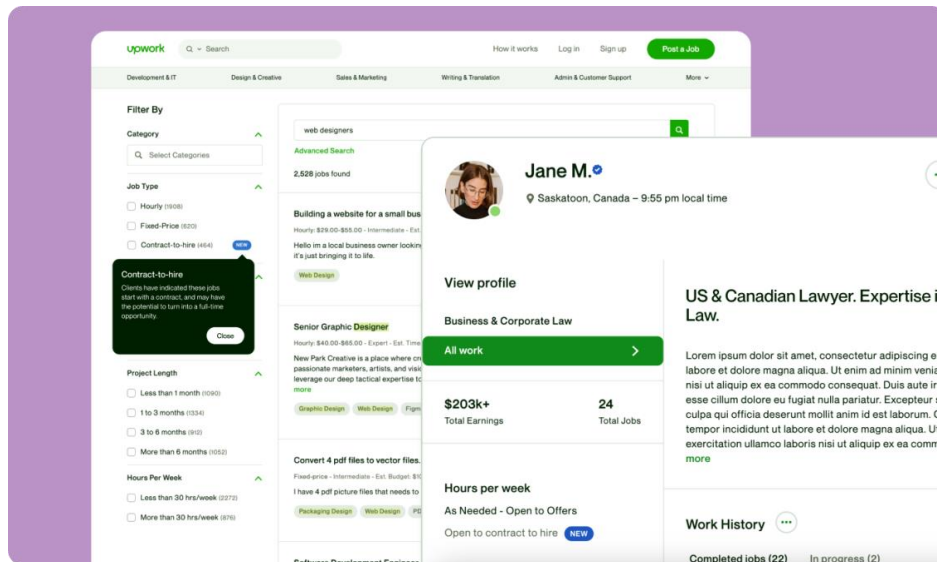


Рисунок 1.1 - Робоча сторінка додатка Upwork

Fiverr – це онлайн-платформа, спрямована на надання послуг у різних областях [2]. Замовники можуть знаходити фрілансерів, готових виконати конкретні завдання або надати конкретні послуги. Fiverr відомий своєю простою та інтуїтивно зрозумілою платформою, де користувачі можуть

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

замовляти різноманітні послуги, від графічного дизайну до відеомонтажу. Оплата за послуги здійснюється через вбудовану систему оплати, забезпечуючи зручний та безпечний обмін коштами між замовниками та фрілансерами. Комунікація між клієнтом та виконавцем здійснюється через вбудовані інструменти спілкування, включаючи чатову систему та можливість обміну повідомленнями. Це створює ефективний та прозорий зв'язок, дозволяючи обговорювати деталі проєкту, уточнювати вимоги та отримувати зворотний зв'язок для досягнення найкращого результату. Також на платформі є прозорий механізм відгуків, який допомагає іншим користувачам обирати виконавців на основі їхньої репутації та якості послуг. Крім того, важливою є можливість перегляду портфоліо виконавців, щоб забезпечити відповідність їхніх навичок та стилю вимогам конкретного проєкту.

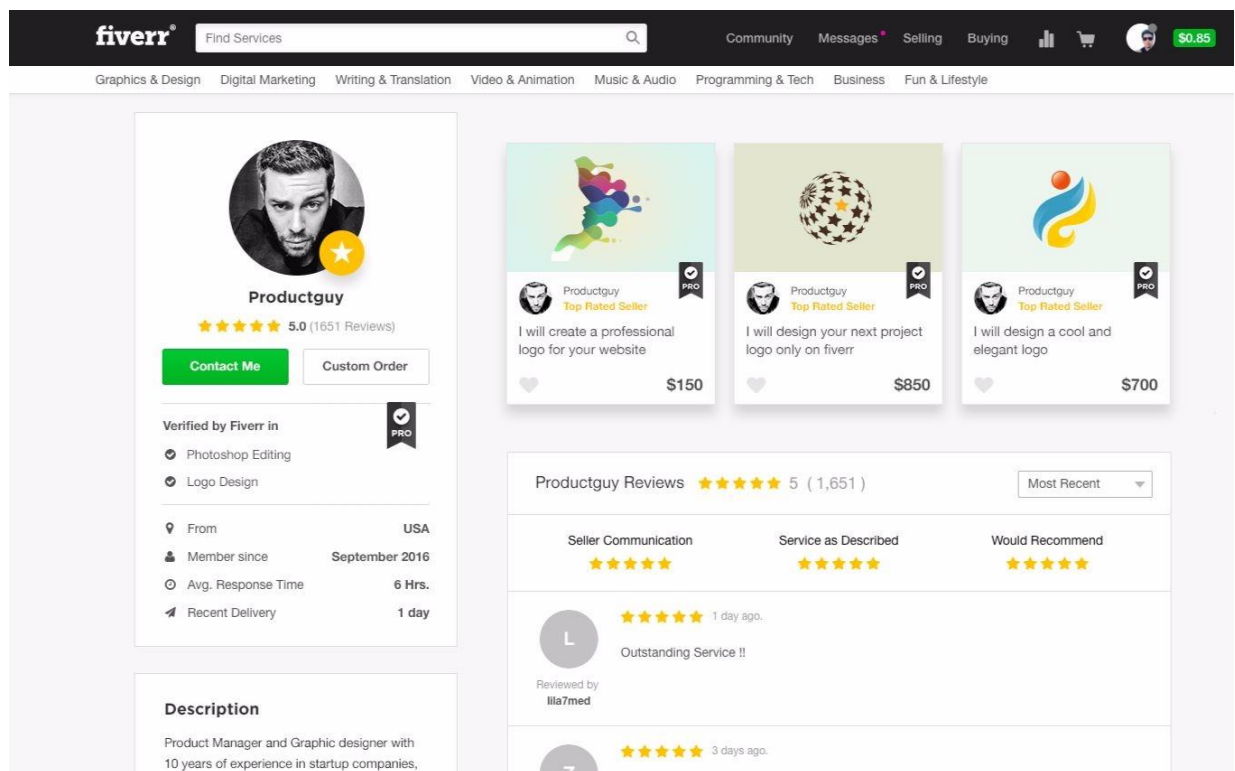


Рисунок 1.2 – Робоча сторінка додатка Fiverr

TaskRabbit – це платформа, орієнтована на надання послуг у сфері реального життя та допомоги у вирішенні різноманітних повсякденних завдань [3]. Замовники можуть шукати фахівців, готових виконати різні

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

завдання, такі як прибирання, ремонт, переїзд та інші ділові послуги. Платформа дозволяє легко знаходити та спілкуватися з фахівцями, а оплата за виконані послуги також проводиться через систему електронного платежу, щоб забезпечити прозорий та безпечний процес транзакцій між сторонами. TaskRabbit визначається своєю акцентуацією на реальних потребах користувачів і підтримкою вирішення їхніх повсякденних завдань. Комунікація на платформі відбувається через вбудований чат, який дозволяє сторонам взаємодіяти, обговорювати деталі завдань, узгоджувати умови та надавати необхідні пояснення. Цей механізм сприяє ясному розумінню вимог та забезпечує ефективний обмін інформацією для успішного виконання завдань на платформі TaskRabbit. Крім цього, на платформі існує система оглядів та оцінок, яка грає важливу роль у виборі виконавця для конкретного завдання.

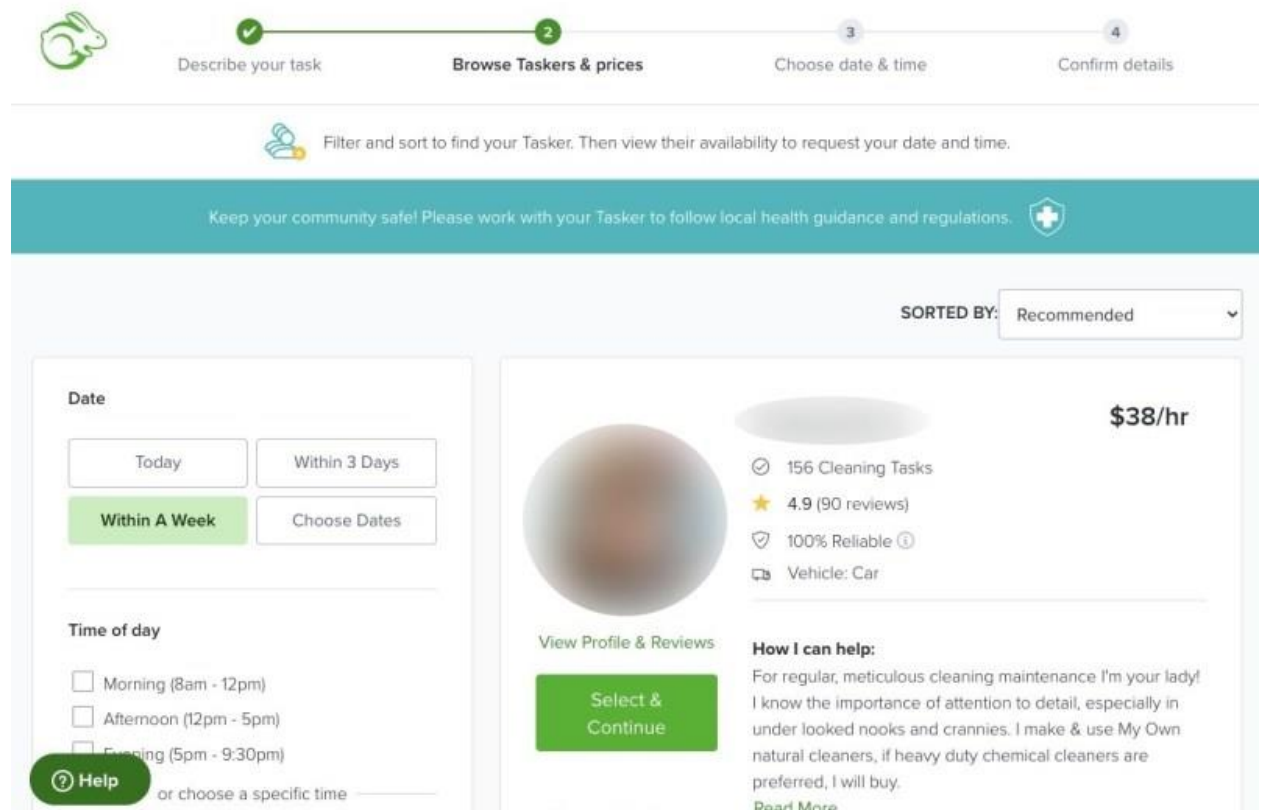


Рисунок 1.3 - Робоча сторінка додатка TaskRabbit

Загальна характеристика і порівняння всіх досліджуваних сайтів наведена нижче (табл. 1.1).

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

Порівняльна характеристика аналогів програмного продукту

Параметр	Upwork	Fiverr	TaskRabbit
Підсилюючі функції	Рейтинг, глобальна аудиторія	Рейтинг, чітка категоризація послуг	Локальна орієнтація, рейтинг
Тип виконавців	Глобальні фрілансери з різних країн	Індивіди та агентства, різноманітні категорії	Локальні виконавці для реальних завдань
Категорії послуг	Програмування, дизайн, копірайтинг	Графічний дизайн, відеоролики, письменницькі послуги	Допомога в різних сферах життя
Система оцінок та відгуків	Є	Є	Є
Глобальне охоплення	Так	Так	Ні (локально орієнтований)
Інтерфейс	Сучасний, зручний	Простий, з чіткою категоризацією	Легкий та зрозумілий
Адаптивність інтерфейсу	Так	Так	Так
Можливість поширення контенту	Файли	Фото, відео, файли	Фото, відео, файли
Вартість послуг	Різнноманітна, визначається виконавцем	Різнноманітна, визначається виконавцем	Залежить від виду та обсягу послуг
Зручність використання	Так	Так	Так

Аналіз аналогів за ключовими характеристиками дозволив визначити наступні особливості для розроблюваної платформи: зручний і адаптивний інтерфейс, який забезпечує легкий доступ з будь-яких пристроїв, підтримка різних форматів повідомлень у чаті, інтеграція системи відгуків для забезпечення високої якості послуг, а також гнучка адміністративна частина, яка дозволить допомагати користувачам платформи.

1.3 Вибір архітектури веборієнтованої платформи

Оскільки платформа повинна бути надійною, безпечною, конкурентоспроможною, легкою та зрозумілою для користувачів різного віку, а також націлена на існування протягом тривалого часу, то вона повинна бути гарно побудована на архітектурному рівні. Це необхідно, бо відразу зробити додаток у найкращій його версії неможливо. Завжди потрібні додаткові тести, перевірки та новий функціонал. Щоб помилки швидко виправлялись, а нові опції швидко додавались до платформи, потрібно правильно підібрати архітектуру програмного забезпечення.

Архітектура програмного забезпечення – це високорівневе структурне представлення програмної системи, яке визначає її ключові компоненти, їхню взаємодію та організацію для досягнення визначених функціональних і якісних характеристик. Архітектура програмного забезпечення визначає фундаментальні аспекти системи, такі як структура, поведінка, взаємодія компонентів, а також розподіл ресурсів, забезпечуючи основу для подальшого розвитку, тестування і супроводження програмного продукту.

Під час аналізу можливих АПЗ [4, 5, 6] було зроблено висновок, що для платформи, спрямованої на обмін послуг та завдань між замовниками та виконавцями, використання клієнт-серверної АПЗ має ряд вагомих переваг.

По-перше, взаємодія із сервером. Клієнт-серверна архітектура дозволяє ефективно взаємодіяти із сервером, вимагаючи або передаючи дані лише тоді, коли це необхідно. Це дозволяє зменшити навантаження на мережу та забезпечити більш ефективний обмін інформацією.

По-друге, асинхронність. Клієнт-серверна архітектура може працювати в асинхронному режимі, що є важливим для сучасних вебзастосунків. Це дозволяє обробляти багато запитів одночасно, покращуючи продуктивність та відзначаючи систему як більш масштабовану.

По-третє, відокремлення інтерфейсу та логіки. Клієнт-серверна архітектура дозволяє чітко відокремити логіку додатка на стороні сервера від

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		15

інтерфейсу користувача на стороні клієнта. Це розділення полегшує розробку, тестування та супровід, оскільки зміни в одній частині системи майже не впливають на іншу.

По-четверте, сприяння масштабованості. Клієнт-серверна архітектура забезпечує гнучкість у розгортанні, це сприяє масштабованості системи. Можна легко додавати нові клієнти чи розширювати функціонал сервера без значних змін в коді інших компонентів.

По-п'яте, більша безпека. Через те, що логіка додатка зберігається на сервері, це дозволяє ефективніше керувати безпекою та обробкою даних. Чутливі операції можна виконувати тільки на сервері, це зменшує ризик витоку чутливої інформації на клієнтську сторону.

По-шосте, підтримка різних клієнтів. Клієнт-серверна архітектура дозволяє реалізувати різноманітні клієнтські додатки для різних платформ (веб, мобільні пристрої, настільні додатки), обслуговуючи різні потреби користувачів.

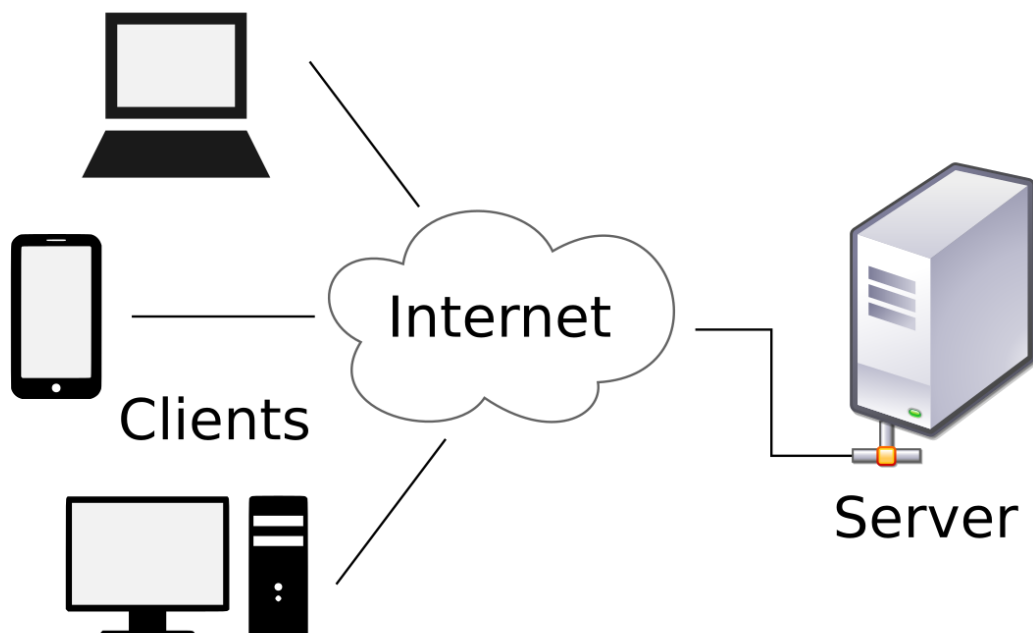


Рисунок 1.4 - Візуалізації клієнт-серверної архітектури

Аналізуючи [7], було акцентовано увагу на двох найбільш використовуваних архітектурних паттернах, а саме: на MVC та MVVM. У [8]

наведено чітке порівняння між цими паттернами, тому було прийнято рішення взяти саме MVC за основу, бо цей патерн простіший у реалізації та підтримці та не вимагає спеціальних бібліотек для реалізації, тобто є менш об’ємним, оскільки має меншу кількість компонентів.

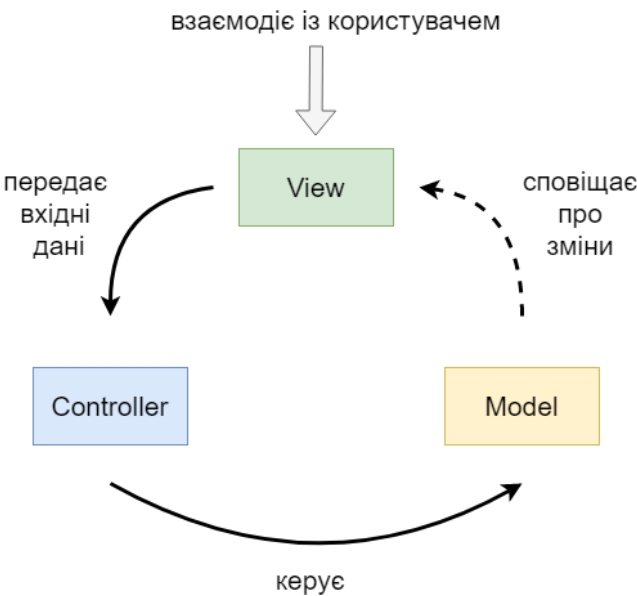


Рисунок 1.5 - Візуалізації MVC патерну

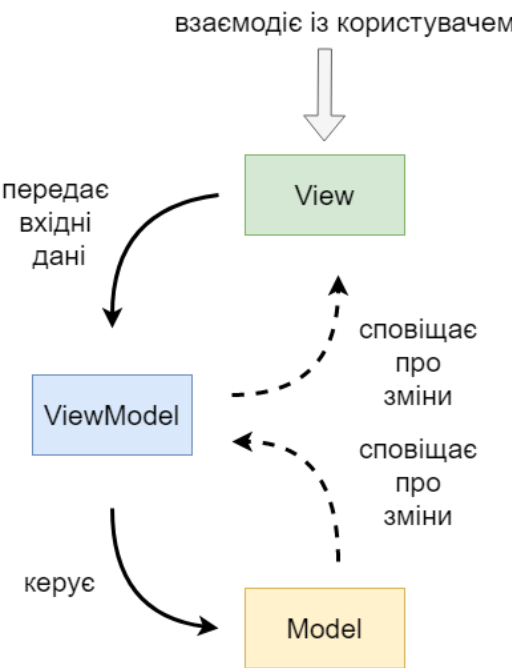


Рисунок 1.6 - Візуалізації MVVM патерну

Крім цього, патерн MVC є дуже доцільним для проекту платформи обміну послуг та завдань. Ось кілька конкретних причин, чому цей патерн підходить для реалізації платформи:

- розділення відповідальностей: MVC визначає три основні компоненти: модель (model), вигляд (view) та контролер (controller). Це дозволяє чітко визначити та розділити відповідальності між різними частинами додатка. Модель відповідає за логіку додатка та роботу з даними, Вигляд – за представлення інформації користувачеві, а Контролер – за обробку введених користувачем дій;

- легке розширення та зміни: одна з переваг MVC полягає в тому, що зміни в одній частині системи майже не впливають на інші. Це забезпечує гнучкість та легкість у внесенні змін або розширенні функціонала;

- покращена організація коду: використання MVC дозволяє розділити код на логічні шари, що полегшує організацію та управління кодовою базою;

- підтримка різних інтерфейсів: контролер дозволяє легко взаємодіяти з різними видами інтерфейсів, такими як вебсторінки, мобільні додатки, API і так далі. Модель і вигляд можуть залишатися практично незмінними, що спрощує розробку та супровід.

Для виконання етапу кодування обрано середовище розробки Visual Studio Code (VS Code), в якому буде проводитись подальша розробка додатка, використовуючи інструменти та мови програмування, такі як Node.js, React та MySQL. Visual Studio Code відомий своєю легкістю використання та розширюваністю.

VS Code – це інтегроване середовище розробки, яке надає інтелектуальний редактор з можливостями аналізу коду, автоматизованими засобами виявлення та виправлення помилок, а також широким спектром розширень для підтримки різних мов та технологій. Згідно з [9], до програми було додано деякі розширення, це зробило її ще зручнішою для розробки.

Проект розроблюватиметься за допомогою Node.js для серверної частини, React для клієнтського інтерфейсу та MySQL для управління базою даних. Використання VS Code сприяє зручній та ефективній розробці завдяки його інтуїтивному інтерфейсу та широкому спектру можливостей для роботи з кодом.

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		18

1.4 Обґрунтування вибору засобів реалізації та вимоги до апаратного забезпечення

Обґрунтування вибору засобів реалізації та вимоги до апаратного забезпечення є критичним етапом у розробці будь-якого програмного продукту. Доцільність вибору конкретних технологій, мов програмування та апаратного забезпечення визначається широким спектром факторів, таких як функціональні вимоги продукту, потреби користувачів, вимоги до продуктивності, ефективність розробки та підтримки системи.

У даному проєкті, буде використано Node.js для серверної частини, оскільки він є відкритою та ефективною платформою. Основною перевагою є здатність виконувати JavaScript на сервері, що робить його ідеальним вибором для розробки масштабованих вебдодатків.

Крім того, Node.js використовує асинхронність, що дозволяє виконувати багато операцій одночасно без очікування завершення кожної з них. Це особливо корисно для вебсерверів, які повинні ефективно обслуговувати багато запитів від користувачів одночасно. Для проєкту це означає швидший обмін даними між клієнтом і сервером, зменшуючи час очікування для кінцевого користувача.

Також Node.js здатний легко масштабуватись горизонтально, додаванням нових серверів для обробки збільшеного навантаження. Це дозволяє підтримувати високу доступність та швидкість при зростанні кількості користувачів. Це дає можливість легко розширювати систему з ростом її популярності та користувацької бази.

Ще одним плюсом Node.js є велика та активна спільнота розробників, яка забезпечує підтримку та постійний розвиток платформи. Широкий вибір пакетів (за допомогою npm – менеджера пакетів Node.js) спрощує використання готових рішень та бібліотек для розширення функціональності проєкту.

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		19

Ще однією перевагою цієї мови є наявність чіткої документації до самої мови, так і до бібліотек, що часто використовуються, наприклад, `express.js`, документації до зовнішніх API платформ, які працюють з платежами, а також до бібліотеки, яка взаємодіє з сокетом [10, 11, 12, 13, 14].

Також, ґрунтуючись на [15], завдяки використанню JavaScript, Node.js робить можливим використання однієї мови програмування як на клієнтській, так і на серверній частині. Це спрощує взаємодію між розробниками, зменшує поріг входження в проєкт та зменшує потребу у використанні великої кількості розробників для підтримки коду та проєкту у майбутньому.

Ще одним важливим аспектом є зручність реалізації патерну MVC за допомогою Node.js. Це підтверджується [16], де чітко вказано, що перевагою є простота.

Для клієнтської частини цього проєкту буде використано React. React – це бібліотека JavaScript для розробки інтерфейсів користувачів, яка дозволяє створювати високопродуктивні та ефективні односторінкові додатки.

Однією з переваг React є його компонентна архітектура. Платформа, яка дозволяє людям допомагати один одному в різних задачах за грошові нагороди, може включати різноманітні функції та елементи інтерфейсу. React дозволяє розділити ці функції на компоненти, які можуть бути незалежно розроблені, протестовані та підтримувані. Це полегшує масштабування та утримання коду.

Також React може похвалитись своєю швидкодією, адже він використовує віртуальний DOM та ефективний алгоритм оновлення, що робить його особливо швидким для SPA. Користувачі матимуть швидкий та плавний досвід взаємодії з платформою.

Крім цього, React є однією з найпопулярніших та найбільш підтримуваних бібліотек для розробки інтерфейсів. Його активна спільнота

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		20

забезпечує багатство ресурсів, документацію та готові рішення для різних аспектів розробки.

На додачу, React має унікальну документацію [17] у своїй реалізації. Вона дозволяє покроково розвиватись та розбиратись у цій бібліотеці. Ще одним плюсом є наявність великої кількості бібліотек та статей до бібліотек, які часто використовуються у парі з React. Наприклад, це документація до використання сокетів у React та використання систем оплат [18, 19, 20].

Також у React є свій синтаксис. JSX полегшує опис інтерфейсу, надаючи можливість використовувати синтаксис, схожий на HTML, для створення компонентів. Це робить код більш зрозумілим та легким для розробки. На додачу у React дуже зручна взаємодія з серверною частиною, написаною на Node.js. Це забезпечить гармонійну взаємодію між клієнтом та сервером на обох частинах вашого проєкту.

Крім того, React дуже зручно комбінувати з бібліотекою Bootstrap5, яка має зручну документацію як для компонентів, так і для іконок, які мають гарний вигляд на сайтах, реалізованих з її використанням [21, 22].

Для збереження даних, даний проєкт буде використовувати MySQL, оскільки вона є однією з найпоширеніших та добре вивчених реляційних систем управління базами даних. Вона вже довгий час успішно використовується у великих проєктах та корпоративних системах, що свідчить про її стабільність та надійність.

MySQL має широку спільноту користувачів, а також детальну документацію [23]. Ця документація гарно доповнюється іншими статтями, які забезпечують необхідними ресурсами для розробки, оптимізації та управління базою даних [24, 25]. Стаття [26] допомогла розібратися з користю та сенсом використання підзапитів при розробці.

Крім цього, одним з найважливіших моментів є те, що MySQL є вільною та відкритою системою, що робить її доступною для використання без великих витрат. Це може бути важливим фактором, особливо для початкових етапів розробки та стартапів.

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

На додачу, MySQL добре інтегрується з серверними застосунками, написаними на Node.js, завдяки наявності бібліотек та драйверів для Node.js, таких як "mysql" або "sequelize". Ці бібліотеки дозволяють виконувати SQL-запити, отримувати та взаємодіяти з даними швидко, зручно і безпечно. Це забезпечує можливість в майбутньому оновлювати та покращувати функціонал платформи.

Вимог до апаратного забезпечення небагато, але вони є, оскільки навіть з добре оптимізованим кодом додаток може просто стати набором символів, який не приносить жодної користі та задоволення.

Перш за все, для платформи потрібен чотириядерний або потужніший процесор. Такий процесор часто використовується для обробки паралельних запитів та оптимізації роботи серверної частини додатка. Це забезпечує ефективну роботу, особливо в умовах великого обсягу одночасних з'єднань.

Ефективна робота серверної частини додатка вимагає значної кількості оперативної пам'яті. Оперативна пам'ять використовується для тимчасового зберігання та швидкого доступу до активних процесів та даних. Використання обсягу оперативної пам'яті у розмірі 8 ГБ, або більше, дозволить серверу зручно та швидко обробляти великі обсяги інформації, таким чином покращуючи загальну продуктивність додатка. Також важливо мати на увазі, що сервер, який оптимально використовує ресурси оперативної пам'яті, може обслуговувати більше одночасних запитів та швидше реагувати на зміни в навантаженні.

Також на сервері має бути SSD жорсткий диск. Використання SSD є ключовим елементом для оптимізації швидкості операцій з даними на сервері. Оперативна система та серверні програми можуть швидше зчитувати та записувати дані на SSD, оскільки він не має рухомих частин, які спричиняють затримки у порівнянні з традиційним жорстким диском HDD. SSD також забезпечує високий рівень надійності та тривалості роботи, оскільки він менше схильний до механічних поломок. Це робить його ідеальним вибором для систем, які працюють без перерв, таких як сервери.

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

Пришвидшення швидкості доступу до даних сприяє зменшенню часу очікування для клієнтів та покращенню загального користувацького досвіду. Платформа буде зберігати велику кількість даних, фото та відеоматеріалів, тому найкраще, щоб розмір диска був більшим за 500 ГБ.

Висновки до першого розділу

У першому розділі бакалаврської роботи було проведено детальний аналіз особливостей сучасних платформ, схожих за ціллю та функціоналом до розроблюваної. Мета дослідження була чітко визначена, а також поставлені завдання проєкту були чітко сформульовані.

Також було визначено середовище програмного забезпечення для розробки платформи та набір інструментів, які будуть використовуватися. Зокрема, вибір був зроблений на користь Node.js для серверної частини проєкту та React для клієнтської. Цей вибір був обґрунтований їхньою ефективністю, швидкодією та можливістю легкої масштабованості.

Була проведена глибока робота над архітектурними аспектами платформи. Увага була зосереджена на визначенні ефективної та добре структурованої архітектури, яка відповідає потребам проєкту. З цією метою було обрано патерн проєктування MVC, який дозволяє чітко розділити логічні компоненти системи.

Таким чином, у першому розділі описані ключові принципи та фундаментальні рішення для подальшої розробки та реалізації платформи, визначаючи її стратегічний напрям та основні технічні параметри.

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

РОЗДІЛ 2 ПРОЄКТУВАННЯ ПЛАТФОРМИ

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

Платформа надає вебінтерфейс, через який користувачі можуть створювати завдання, вказуючи в них очікуваний час виконання та ціну, яку вони можуть заплатити. Крім того, вони вказують локацію, де потрібно виконати завдання, описують особливості виконання та умови завершення, за якими виконавець розумітиме, за що саме буде платити йому замовник. Для того, щоб зробити пропозицію на виконання, робітники мають відповідну форму, де вказують необхідний час для виконання завдання та ціну, яку вони хочуть отримувати за годину своєї роботи. Також платформа надає можливість вибору, тобто кожен замовник може отримати декілька пропозицій на виконання завдання, обговорити з виконавцями деталі через внутрішній чат і після того, вирішити, яка з них є для нього найбільш привабливою та підтвердити її. Після цього платформа надає функціонал для відстеження та зміни поточного статусу завдання, комунікації між користувачами додатком та залученням адміністратора у вирішення суперечки. Своєю чергою, адміністратори мають зручний функціонал для вирішення суперечок, де вони можуть переглядати чати між користувачами, у яких виник конфлікт (усі зміни та видалення також відображаються у візуально-зрозумілому вигляді), вести комунікацію з кожним учасником конфлікту та переглядати інформації про них, таку як статистика та деякі деталі, включаючи пошту, біографію та місце роботи.

Для кращого візуального розуміння взаємодії між елементами додатка була створена діаграма варіантів використання (рис 2.1).

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		24



Рисунок 2.1 - Діаграма варіантів використання

Опис діаграми варіантів використання наведено в табл. 2.1 – 2.19.

Таблиця 2.1

Опис прецеденту «Вхід»

Назва варіанту використання	Вхід
Основні актори	Неавторизований користувач
Короткий опис	Користувачі можуть авторизуватись і залежно від їхніх ролей на платформі виконувати певні операції

Таблиця 2.2

Опис прецеденту «Реєстрація»

Назва варіанту використання	Реєстрація
Основні актори	Неавторизований користувач
Короткий опис	Користувачі можуть реєструватись для створення нових задач, заробляння коштів, або виконання обов'язків адмінів

Таблиця 2.3

Опис прецеденту «Редагування профілю»

Назва варіанту використання	Редагування профілю
Основні актори	Користувач
Короткий опис	Користувачі можуть дозаповнювати інформацію про свій профіль, або змінювати її, залежно від актуальності

Таблиця 2.4

Опис прецеденту «Поповнення балансу»

Назва варіанту використання	Поповнення балансу
Основні актори	Користувач
Короткий опис	Користувачі можуть поповнювати свій баланс на платформі для виконання внутрішніх фінансових операцій

Таблиця 2.5

Опис прецеденту «Виведення коштів з акаунта»

Назва варіанту використання	Виведення коштів з акаунта
Основні актори	Користувач
Короткий опис	Користувачі можуть виводити кошти на зручні для них платформи

Таблиця 2.6

Опис прецеденту «Створення задач»

Назва варіанту використання	Створення задач
Основні актори	Користувач
Короткий опис	Користувачі можуть створювати задачі, які їм потрібно, щоб виконали інші

Таблиця 2.7

Опис прецеденту «Підтвердження пропозиції»

Назва варіанту використання	Підтвердження пропозиції
Основні актори	Користувач
Короткий опис	Користувач може підтвердити, що пропозиція йому підходить і автор пропозиції розпочинає її виконувати

Таблиця 2.8

Опис прецеденту «Зміна статусу виконання задачі»

Назва варіанту використання	Зміна статусу виконання задачі
Основні актори	Користувач
Короткий опис	Користувач може змінити статус виконання задачі залежно від його ролі в задачі та поточного стану виконання. Замовник може підтвердити успішне виконання задачі, а виконавець надіслати запит на підтвердження

Таблиця 2.9

Опис прецеденту «Створення суперечок»

Назва варіанту використання	Створення суперечок
Основні актори	Користувач
Короткий опис	При виникненні недовомовленості під час виконання задачі, користувачі можуть звернутись до адміністратора для вирішення конфлікту

Таблиця 2.10

Опис прецеденту «Перегляд суперечки»

Назва варіанту використання	Перегляд суперечки
Основні актори	Адміністратор
Короткий опис	Адміністратор має можливість переглядати детальну інформацію при виникненні суперечки, перед тим як братись за її вирішення

Таблиця 2.11

Опис прецеденту «Перегляд чату суперечки»

Назва варіанту використання	Перегляд чату суперечки
Основні актори	Адміністратор
Короткий опис	Адміністратор має можливість переглядати чат між користувачами, які є учасниками суперечки. Це дозволить адміністратору краще зрозуміти ситуацію і розібратись в ній

Таблиця 2.12

Опис прецеденту «Закріплення суперечки»

Назва варіанту використання	Закріплення суперечки
Основні актори	Адміністратор
Короткий опис	При закріпленні суперечки, адміністратор позначається в ній як виконавець і інші адміністратори не можуть взятись за її вирішення

Таблиця 2.13

Опис прецеденту «Перегляд задачі»

Назва варіанту використання	Перегляд задачі
Основні актори	Користувач
Короткий опис	Користувачі можуть переглядати інформацію про задачі, інформацію про власника, його статистику і так далі, щоб впевнитись чи їм підходять умови для успішного виконання

Таблиця 2.14

Опис прецеденту «Вирішення суперечки»

Назва варіанту використання	Вирішення суперечки
Основні актори	Адміністратор
Короткий опис	При вирішенні суперечки, адміністратор може обрати хто є має рацію у ній і чи отримає власник зарезервовані кошти, чи все-таки виконавець забере зароблене

Таблиця 2.15

Опис прецеденту «Перегляд профілю користувача»

Назва варіанту використання	Перегляд профілю користувача
Основні актори	Адміністратор, користувач
Короткий опис	Адміністратор та користувач мають можливість переглядати профілі інших користувачів для вирішення чи варто з ними мати справу, для аналізу коментарів та іншої статистики

Таблиця 2.16

Опис прецеденту «Надсилання повідомлень»

Назва варіанту використання	Надсилання повідомлень
Основні актори	Адміністратор, користувач
Короткий опис	Адміністратор та користувач мають можливість відправляти повідомлення іншим користувачам з метою покращення комунікації

Таблиця 2.17

Опис прецеденту «Надсилання коментаря виконавцю»

Назва варіанту використання	Надсилання коментаря виконавцю
Основні актори	Користувач
Короткий опис	Користувачі можуть оцінювати один одного як виконавців різних задач

Таблиця 2.18

Опис прецеденту «Надсилання коментаря замовнику»

Назва варіанту використання	Надсилання коментаря замовнику
Основні актори	Користувач
Короткий опис	Користувачі можуть оцінювати один одного як замовників різних задач

Таблиця 2.19

Опис прецеденту «Перегляд запитів на виконання задач»

Назва варіанту використання	Перегляд запитів на виконання задач
Основні актори	Користувач
Короткий опис	Користувачі можуть переглядати пропозиції на виконання їхніх задач

Основою метою розробки цієї платформи є створення онлайн-середовища, яке сприяє ефективній комунікації та співпраці між користувачами, які мають завдання для виконання, та тими, хто готовий їх виконати. Платформа спрямована на забезпечення зручного та надійного механізму для створення, призначення та виконання завдань, а також на сприяння обміну ідеями та знаннями у чаті. Основні функціональності платформи включають можливість створювати завдання, встановлювати винагороду за їхнє виконання, спілкуватися у чаті для обговорення деталей та узгодження умов, відстеження прогресу та завершення завдань, а також вирішення суперечок.

Основна мета платформи полягає у створенні ефективного середовища для співпраці між людьми, які мають завдання, що потребують допомоги у їхньому виконанні, та тими, хто готовий виконати ці завдання за певну винагороду. Платформа надає можливість розміщення різноманітних завдань, від рутинних до складніших, та забезпечує прозорий і безпечний процес співпраці між замовниками та виконавцями. Крім того, вона створює можливості для активної комунікації, обміну ідеями та навичками між користувачами, що сприяє якісній і швидкій реалізації завдань. Такий підхід дозволяє підтримувати розвиток і ефективне виконання проєктів у різних сферах діяльності.

Вимоги користувачів:

Внутрішній користувач – користувач:

1. Реєстрація або авторизація для доступу до функціонала.
2. Редагування профілю.
3. Поповнення балансу.
4. Виведення коштів на зручну платформу.
5. Створення завдання.
6. Перегляд пропозицій виконання певного завдання.
7. Обговорення пропозицій з виконавцями.
8. Перегляд статистики користувача.

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

9. Створення суперечки в завданні.
10. Запит на здачу завдання.
11. Підтвердження успішного завершення завдання.
12. Прийняття та відхилення пропозицій.
13. Перегляд доступних завдань для виконання.
14. Вибір та прийняття завдань для виконання.
15. Виконання завдань та надсилання результатів.
16. Отримання винагороди за успішно виконані завдання.
17. Перегляд історії виконаних завдань та отриманих винагород.
18. Написання коментарів з оцінкою виконавцю.
19. Написання коментарів з оцінкою замовнику.
20. Написання коментарів до задачі.

Внутрішній користувач – адміністратор:

1. Можливість керування користувачами та їхніми правами доступу.
2. Модерація завдань та винагород за їхнє виконання.
3. Вирішення конфліктних ситуацій.
4. Перегляд статистики платформи.

Функціональні вимоги:

1. Реєстрація користувачів: можливість реєстрації користувачів з обов'язковими полями, такими як ім'я, електронна пошта, пароль.
2. Авторизація користувачів: авторизація зареєстрованих користувачів з використанням електронної пошти та пароля.
3. Редагування профілю: можливість користувачів редагувати свої профілі, включаючи фотографію, контактну інформацію та навички.
4. Створення та перегляд замовлень: можливість роботодавців створювати замовлення для виконавців з описом завдань, бюджетом і терміном виконання. Виконавці можуть переглядати список доступних замовлень.

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

5. Чат для спілкування: вбудований чат для взаємодії між роботодавцями та виконавцями щодо уточнень, деталей та обговорення завдань.

6. Оцінювання та відгуки: можливість користувачів залишати відгуки та оцінки після виконання завдань.

7. Система суперечок: можливість розглядати суперечки між замовниками та виконавцями та вирішувати їх.

Нефункціональні вимоги:

1. Вимоги до продуктивності:

1.1. Час відгуку системи: час відгуку для типових завдань повинен бути не більше 15 секунд, а для складних завдань – не більше 30 секунд.

1.2. Підтримка одночасних користувачів: система повинна підтримувати мінімум 40 одночасно активних користувачів, які взаємодіють із загальною базою даних.

2. Можливості експлуатації:

2.1. Масштабування: система повинна мати можливість масштабуватися, щоб збільшувати продуктивність і забезпечувати зручну роботу з ростом кількості користувачів.

2.2. Забезпечення безпеки даних: захист особистих даних користувачів та даних замовлень є важливою складовою.

Системні вимоги:

1. Вимоги до середовища виконання: платформа повинна задовольняти вимогам на пристроях, які знаходяться в наступній мінімальній комплектації:

1.1. Мінімум 2 ГБ оперативної пам'яті для забезпечення плавної роботи.

1.2. Процесор з тактовою частотою не менше 1.4 GHz для швидкого виконання завдань.

1.3. Доступ до мережі Інтернет для забезпечення з'єднання з платформою.

2. Вимоги до сервера інформаційної системи:

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

2.1. У ядрі системи повинна бути встановлена реляційна база даних, яка забезпечує надійне зберігання та організацію даних.

2.2. Сервер повинен підтримувати технології Node.js для ефективної обробки запитів та забезпечення відповідності роботи платформи інформаційним потребам користувачів.

Аналіз функціональних вимог дозволив виділити такі сутності, які забезпечать програмну реалізацію додатка. Вони представлені у вигляді діаграми основних класів платформи, зображеної на рис. 2.2.

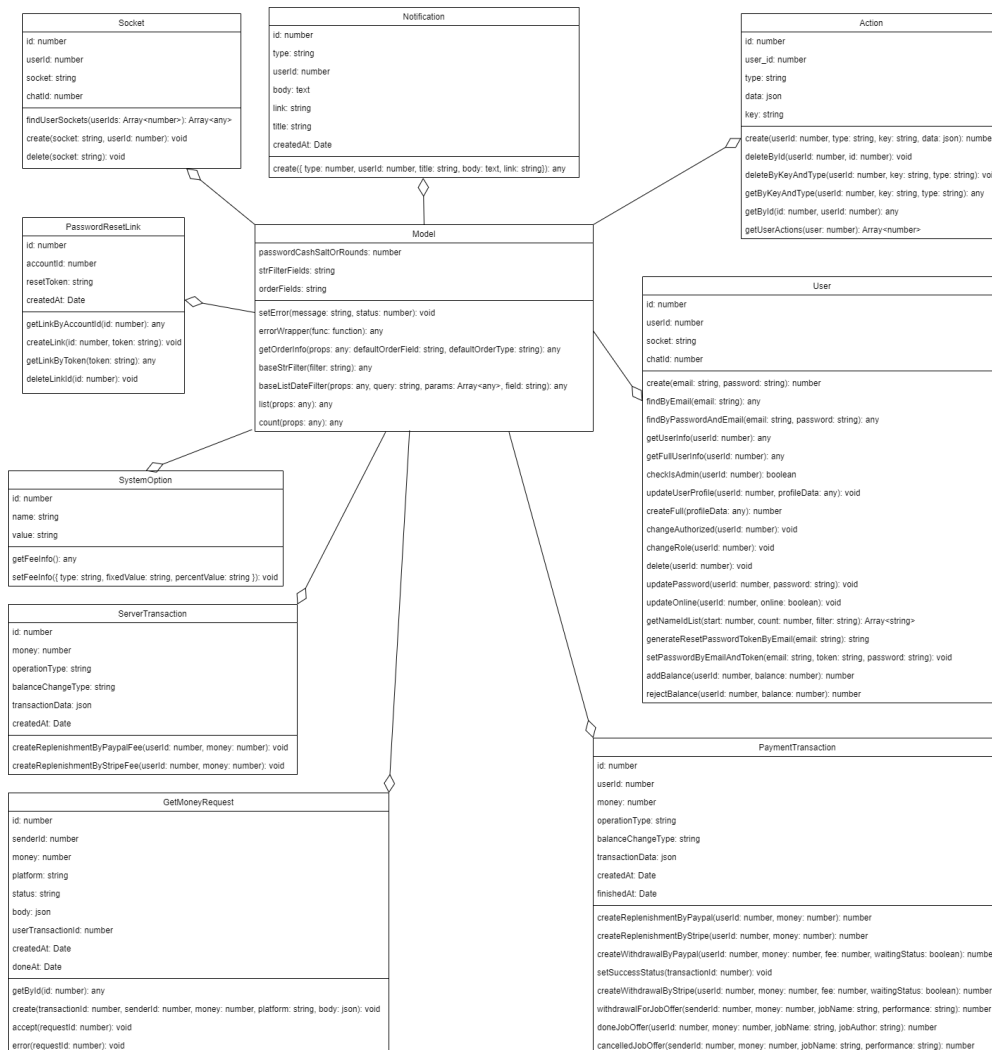


Рисунок 2.2 - Діаграма основних класів

Model – це батьківський клас, який містить декілька загальних методів, на основі яких було спрощено різні операції, що були подібними у всіх нащадкових класах. Крім того, клас містить деякі базові дані, а саме: довжину примісу для шифрування пароля, або інших секретних даних,

список полів, за якими можна робити сортування, та список полів, за якими можна робити рядковий пошук.

Нижче наведено основні методи класу:

- `errorWrapper` – це представник патерну “Декоратор” для всіх методів класів, які успадковують. Його основною метою є відловлення помилок та повернення коректної інформації до контролера;
- `setError` – генерує помилку для контролера та припиняє виконання методу моделі;
- `getOrderInfo` – це метод, який генерує частину запиту, відповідальну за сортування даних, отриманих з бази даних;
- `baseStrFilter` – це метод, який генерує частину запиту, відповідальну за фільтрацію полів з типом “string” чи “text” у вибірці даних, отриманих з бази даних;
- `baseListDateFilter` – метод, який генерує частину запиту, відповідальну за фільтрацію полів з типом “timestamp”, тобто дат, у вибірці даних, отриманих з бази даних;
- `list` – метод, який отримує список даних на основі заданих умов;
- `count` – метод, який отримує кількість даних на основі заданих умов.

User – це клас, що представляє користувачів системи. Він містить інформацію про користувачів, таку як їхній ідентифікатор, електронну адресу, пароль, ім'я (nick), адресу, аватар, координати на мапі (широту і довготу), прапорці авторизації профілю, прапорці адміністратора та дату створення облікового запису користувача. Клас містить наступні методи:

- `create` – це метод, який створює користувача лише за поштою та паролем;
- `findByEmail` – це метод, який повертає дані користувача за поштою;
- `findByPasswordAndEmail` – це метод, який повертає дані користувача за поштою та паролем;
- `getUserInfo` – це метод, який повертає дані користувача за його ідентифікатором;

- `getFullUserInfo` – це метод, який повертає повний перелік даних користувача за його ідентифікатором;
- `checkIsAdmin` – це метод, який перевіряє чи є користувач адміністратором;
- `updateUserProfile` – це метод, який оновлює дані про певного користувача;
- `createFull` – це метод, який створює користувача на основі усіх даних про нього;
- `changeVerification` – це метод, який змінює статус верифікованості користувача;
- `changeRole` – це метод, який змінює роль користувача на платформі;
- `delete` – це метод, який видаляє користувача;
- `updatePassword` – це метод, який задає новий пароль користувачу;
- `updateOnline` – це метод, який оновлює статус про онлайн користувача;
- `getNameldList` – це метод, який повертає перелік залежностей ідентифікатор-ім'я усіх користувачів;
- `addBalance` – це метод, який збільшує баланс користувача;
- `rejectBalance` – це метод, який зменшує баланс користувача.

Socket – це клас, що відповідає за роботу з сокетом, які використовуються для спілкування в режимі реального часу між користувачами. Він містить інформацію про ідентифікатор сокету, ідентифікатор користувача, ідентифікатор чату, до якого під'єднано сокет.

Клас містить наступні методи:

- `findUserSockets` – це метод, який відповідає за отримання усіх сокетів для шуканих користувачів. Необхідно для дій, пов'язаних з діями у РРЧ. Наприклад створення нотифікацій чи чатингу;
- `create` – це метод, який зберігає інформацію про сокет користувача, який зайшов на платформу. Вихід в мережу Інтернет;

- delete – це метод, який видаляє інформацію про сокет користувача, який вийшов з платформи.

Action – це клас, який використовується для ведення журналу тривалих дій користувачів. Він містить інформацію про ідентифікатор дії, ідентифікатор користувача, тип дії, додаткові дані та ключ.

Нижче наведено основні методи класу:

- create – метод, який створює запис про тривалу дію, яку робить користувач на платформі даний момент;
- deleteById – метод, який стирає запис про тривалу дію, яку робив користувач по ідентифікатору операції;
- deleteByKeyAndType – метод, який стирає запис про тривалу дію, яку робив користувач по унікальному ключу операції та типу операції;
- getByKeyAndType – метод, який повертає інформацію про тривалу дію користувача, по унікальному ключу операції та типу операції;
- getById – метод, який повертає інформацію про тривалу дію користувача, по ідентифікатору операції;
- getUserActions – метод, який список усіх поточних тривалих дій користувача.

PasswordResetLink – це клас, що відповідає за управління посиланнями для скидання пароля користувачів. Він містить інформацію про ідентифікатор посилання, ідентифікатор облікового запису користувача, токен скидання пароля та час створення посилання.

Клас містить наступні методи:

- getLinkByAccountId – це метод, який повертає інформацію про запит на скидання пароля по ідентифікатору користувача;
- createLink – це метод, який зберігає інформацію про користувача та токену для скидання пароля на профілі;
- getLinkByToken – це метод, який повертає інформацію про запит на скидання пароля по токену;
- deleteLinkId – це метод, який видаляє запит на скидання пароля.

Notification – це клас, що відповідає за нотифікації користувача. Він містить інформацію про ідентифікатор нотифікації, ідентифікатор отримувача, тип нотифікації, заголовок, основний текст, посилання, куди має перекидати нотифікація та час створення нотифікації.

Нижче наведено основні методи класу:

- create – це метод, який зберігає дані про нотифікацію в базі даних.

SystemOption – це клас, що відповідає за основні налаштування платформи. Він містить різні ключові дані, такі як податок при виведенні коштів з платформи та тип податку.

Клас містить наступні методи:

- getFeeInfo – це метод, який повертає інформацію про податки для виводу коштів;
- setFeeInfo – це метод, який зберігає інформацію про податки для виводу коштів.

ServerTransaction – це клас, що представляє інформацію про всі фінансові операції, в яких був залучений сервер. Клас містить інформацію про ідентифікатор транзакції, тип операції, з якою сумою було проведено операцію, тип зміни балансу, додаткові дані транзакції та дату виконання операції.

Нижче наведено основні методи класу:

- createReplenishmentByStripeFee – це метод, який створює запис про виведення коштів з гаманця на Stripe;
- createReplenishmentByPaypalFee – це метод, який створює запис про виведення коштів з гаманця на Paypal.

PaymentTransaction – це клас, який представляє інформацію про всі фінансові операції, в яких був залучений певний користувач. Клас містить інформацію про ідентифікатор транзакції, ідентифікатор користувача, який був учасником операції, з якою сумою було проведено операцію, тип зміни балансу, тип операції, додаткові дані транзакції, дату створення операції та дату завершення операції.

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

Клас містить наступні методи:

- `createReplenishmentByPaypal` – це метод, який створює запис про поповнення балансу через Paypal;
- `createReplenishmentByStripe` – це метод, який створює запис про поповнення балансу через Stripe;
- `createWithdrawalByPaypal` – це метод, який створює запис про виведення коштів через Paypal;
- `createWithdrawalByStripe` – це метод, який створює запис про виведення коштів через Stripe;
- `setSuccessStatus` – це метод, який позначає транзакцію як завершену;
- `withdrawalForJobOffer` – це метод, який зберігає інформацію про зменшення балансу внаслідок підтвердження початку виконання задачі;
- `doneJobOffer` – це метод, який зберігає інформацію про поповнення балансу внаслідок завершення задачі;
- `cancelledJobOffer` – це метод, який зберігає інформацію про поповнення балансу внаслідок скасування виконання задачі.

GetMoneyRequest – це клас, який представляє інформацію про всі запити на виведення коштів з акаунтів. Клас створений для випадків, коли система через якусь проблему не може відразу вивести кошти клієнту на платформу. Цей клас дозволяє ефективно керувати процесом виведення коштів з акаунтів користувачів на платформі та веде історію усіх операцій, щоб у користувача не виникало зайвих питань про зміну балансу. Клас містить інформацію про ідентифікатор запиту, ідентифікатор користувача, суму виводу, платформу на яку виводять кошти, статус, додаткові дані запиту, ідентифікатор транзакції користувача, яка очікує завершення, дату створення операції та дату завершення операції.

Нижче наведено основні методи класу:

- `getById` – метод, який повертає інформацію про запит на виведення коштів по його ідентифікатору;
- `create` – метод, який створює запит на виведення коштів;

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

- **assert** — метод, який позначає запит на виведення коштів, як завершений;
- **error** — метод, який позначає запит на виведення коштів, як провалений.

Платформа є об'ємною і має різні модулі. Одним з основних модулів платформи є її чат. Сутності, які реалізують цей модуль, зображені у вигляді діаграми класів (рис. 2.3).

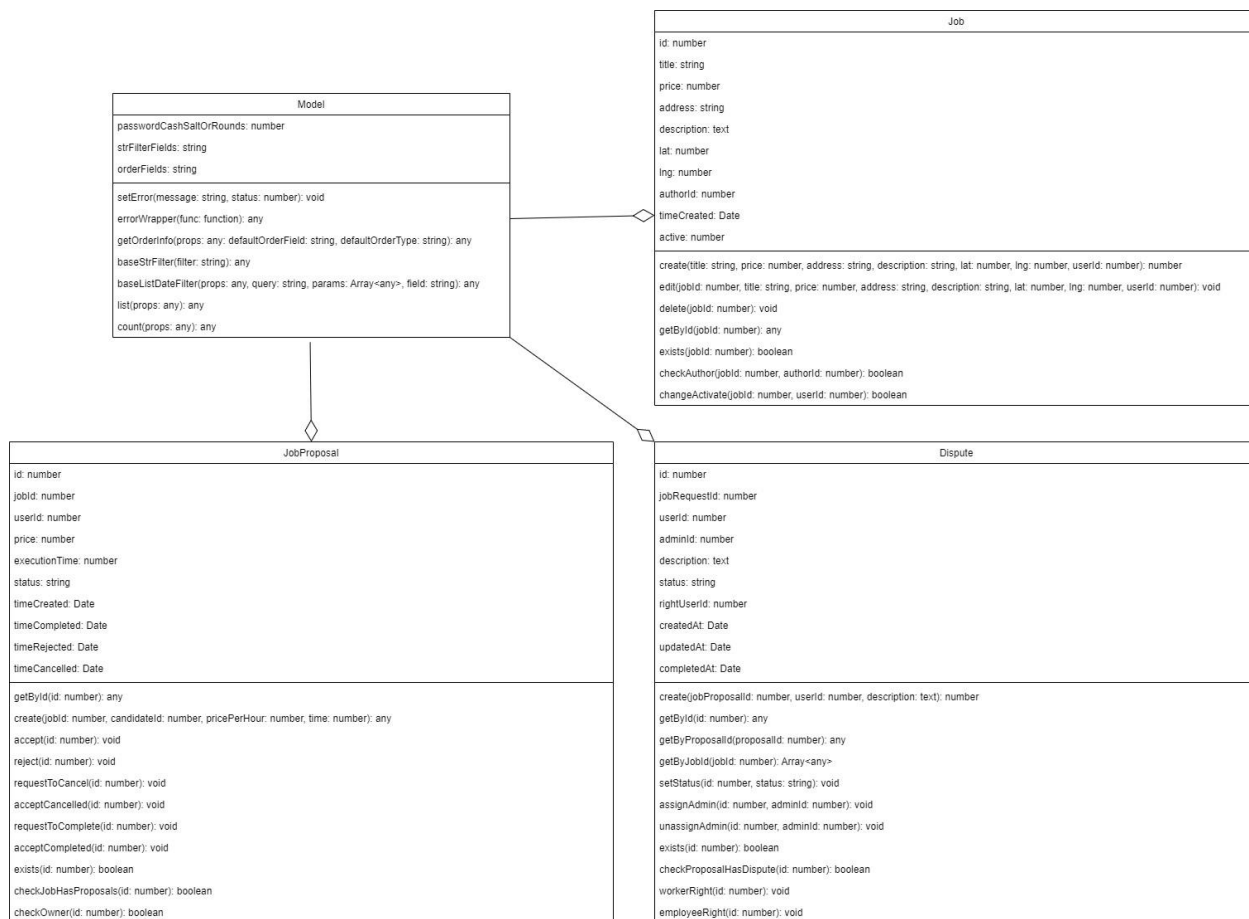


Рисунок 2.3 - Діаграма класів, пов'язаних з завданнями

Job — це клас, що представляє роботи або завдання, які користувачі можуть створювати та виконувати. Він містить таку інформацію про завдання: ідентифікатор, назва, ціна за годину, адреса, опис, координати місця, дані про замовника та дату створення.

Клас містить наступні методи:

- **create** — створює нове завдання у базі даних;
- **edit** — метод, який оновлює інформацію про завдання;

- delete – метод, який видаляє завдання;
- getById – метод, який отримує інформацію про завдання за ідентифікатором;
- exists – метод, який перевіряє існування роботи за ідентифікатором;
- checkAuthor – метод, який перевіряє, чи користувач є автором роботи;
- changeActive – метод, який змінює статус актуальності задачі. Якщо цей статус має значення негативне, то більше ніхто не зможе відправити запит на створення пропозиції на виконання її.

JobProposal – це клас для роботи з запитами на виконання робіт. Він містить інформацію про ідентифікатор запиту, ідентифікатор роботи, інформацію про виконавця, ціну за годину роботи, час для виконання, статус та дату створення запиту.

Клас містить наступні методи:

- getById – метод, який отримує інформацію про пропозицію за ідентифікатором;
- create – створює нову пропозицію для роботи;
- accept – підтверджує початок виконання завдання користувачем;
- reject – відхилення пропозиції на виконання завдання користувачем;
- requestToCancel – створює запит на скасування виконання завдання.

Потрібний, якщо користувачі дійшли до якоїсь незгоди та хочуть завершити замовлення зберігши кошти та репутацію один одного, без втручання адміністратора;

- acceptCancelled – підтверджує скасування виконання завдання.

Потрібний, щоб власники не могли скасувати виконання завдання самостійно в кінці, або в процесі виконання;

- requestToComplete – створює запит на підтвердження завершення завдання;

- acceptCompleted – підтверджує завершення виконання завдання.

Потрібний, щоб виконавці не могли забирати кошти після виконання

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

завдання самостійно в кінці, без доказів успішного завершення для замовника;

- exists – перевіряє наявність пропозиції виконання завдання;
- checkJobHasProposals – перевіряє чи завдання має пропозиції на виконання;
- checkOwner – перевіряє чи є користувач виконавцем завдання.

Dispute – це клас, який використовується для управління суперечками щодо виконання робіт. Він містить інформацію про ідентифікатор суперечки, ідентифікатор запиту на виконання роботи, ідентифікатор користувача, що створив суперечку, ідентифікатор адміністратора, який взявся за вирішення суперечки, опис суперечки, статус, дату створення та оновлення.

Нижче наведено методи класу:

- create – метод, який створює нову суперечку для пропозиції до роботи із зазначеним користувачем та описом;
- getById – метод, який отримує інформацію про суперечку за ідентифікатором спору;
- getByProposalId – метод, який отримує інформацію про суперечку, пов'язану із пропозицією до роботи за ідентифікатором пропозиції;
- getByJobId – метод, який отримує інформацію про суперечку, пов'язану із завданням за ідентифікатором завданням;
- setStatus – метод, є одним з представників реалізації патерну “Фасад”. На основі [27] можна сказати, що патерн використовується для надання простого інтерфейсу до складної системи, щоб спростити її використання. У такому випадку це методи, які змінюють статус суперечки за ідентифікатором та слугує основою для методів workerRight та employeeRight;
- assignAdmin – метод, який призначає адміністратора відповідальним для суперечки за ідентифікатором суперечки;
- unassignAdmin – метод, який скасовує відповідальність адміністратора для суперечки;

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		42

- exists – метод, який перевіряє наявність суперечки;
- checkProposalHasDispute – метод, який перевіряє наявність суперечки у пропозиції;
- workerRight – метод, який обирає робітника, як того, хто правий, у суперечці;
- employeeRight – метод, який обирає замовника, як того, хто правий, у суперечці.

У Додатку А наведено діаграму класів на пряму пов’язаних з чатом.

Дана структура класів є реалізацією патерну “Компонувальник”. Після аналізу [28] стало зрозуміло, що цей патерн дозволяє змінювати окремі частини об’єкта, не змінюючи його повністю. В одному чаті може бути багато користувачів, повідомлень та версій повідомлень, тому логічним було застосування цього патерну для моделювання взаємодії між цими об’єктами у чаті. Наприклад, клас чату виступає у ролі великого компонента, який може містити багато малих компонентів, таких як користувачі та повідомлення. Кожен користувач може бути окремим об’єктом користувача, який може містити повідомлення як підкомпоненти. Повідомлення також можуть мати свої версії, які можна розглядати як підкомпоненти повідомлення. Використання патерну “Компонувальник” дозволяє легко маніпулювати всією структурою чату, незалежно від того, чи ми працюємо з окремим користувачем, повідомленням або версією повідомлення. Це спрощує додавання нових, видалення та редагування елементів чату та виконання різних операцій з ними.

Chat – це основний клас даної структури. Він містить усі основні методи для роботи з чатом:

- hasUserAccess – метод, який перевіряє чи є у користувача доступ до чату;
- getById – метод, який отримує скорочену інформацію про чат за його ідентифікатором;

- lastReadMessageIdByUser – метод, який отримує ідентифікатор останнього прочитаного повідомлення користувачем у чаті;
- create – метод, який створює новий чат;
- addUser – метод, який додає користувача до чату;
- getChatRelations – метод, який отримує повний список користувачів чату;
- addManyUsers – метод, який додає список користувачів до чату;
- deleteUserFromChat – в метод, який видаляє користувача з чату;
- setMainAdminRole – метод, який передає права власника чату користувачу;
- setAdminRole – метод, який надає права адміністратора чату користувачу;
- unsetAdminRole – метод, який забирає права адміністратора чату користувачу;
- addContentToMessage – метод, який додає нову версію контенту повідомлення;
- createMessage – метод, який створює повідомлення в чаті;
- hideMessage – метод, який приховує повідомлення в чаті для всіх користувачів;
- hasPersonal – метод, який перевіряє чи є персональний чат між двома користувачами;
- createPersonal – метод, який створює персональний чат для двох користувачів;
- createGroup – метод, який створює групові чати;
- createSystemChat – метод, який створює системний чат для користувача;
- getAllMessageContents – метод, який отримує увесь список змін повідомлення;
- getMessageContent – метод, який отримує поточний контент повідомлення.

- getAllChats – метод, який отримує усі чати користувача;
- getAllUserSystemChats – метод, який отримує усі системні чати користувачів;
- getUsersToNewAdminChat – метод, який отримує інформацію про користувачів чату адміністраторів платформи;
- getUsersToNewNormalChat – метод, який отримує інформацію про користувачів чату;
- getUsersSocketToSend – метод, який отримує дані про сокети користувачів для розсилки інформації за їхніми ідентифікаторами;
- getMessageByld – метод, який отримує повну інформацію про повідомлення за його ідентифікатором;
- getChatinfo – метод, який отримує повну інформацію про чат за його ідентифікатором;
- getChatMessagesInfo – метод, який отримує повну інформацію про повідомлення чат за ідентифікатором чату;
- getChatMessages – метод, який отримує скорочену інформацію про повідомлення чат за ідентифікатором чату;
- getUnreadChatMessagesCount – метод, який отримує кількість непрочитаних повідомлень в чаті користувача;
- selectChat – метод, який позначає чат як той, що переглядається користувачем і повертає повну інформацію про чат;
- selectSystemChat – метод, який позначає системний чат як той, що переглядається користувачем і повертає повну інформацію про чат;
- getChatUsers – метод, який повертає список користувачів чату;
- getUserSocketsFromChat – метод, який повертає список сокетів користувачів чату;
- getUserChatRole – метод, який повертає роль користувача в чаті;
- deactivateUserChatRelation – метод, який деактивує зв'язок користувача з чатом(видаляє його);

- `setTyping` – метод, який зберігає інформацію про те чи друкує користувач повідомлення в чаті, чи вже закінчив;
- `setLastIdMessage` – метод, який зберігає інформацію про останній переглянуте користувачем повідомлення у чаті;
- `setOwnerByFirstPriority` – метод, який передає права власника чату найпріоритетнішому користувачу. Актуальне при виході власника групи з чату;
- `getUserSystemChatInfo` – метод, який отримує інформацію про системний чат користувача.

ChatsUsers – це клас, який відповідає за збереження даних про зв'язки між користувачами та чатами.

Message – це клас, який відповідає за збереження даних повідомлень, які відправляються в чатах.

MessageContent – цей клас, який представляє вміст повідомлень, який може бути текстовим або мультимедійним.

Ще одним важливим модулем платформи є коментарі. Їхні класи теж представлено у вигляді відповідної діаграми (рис. 2.4).

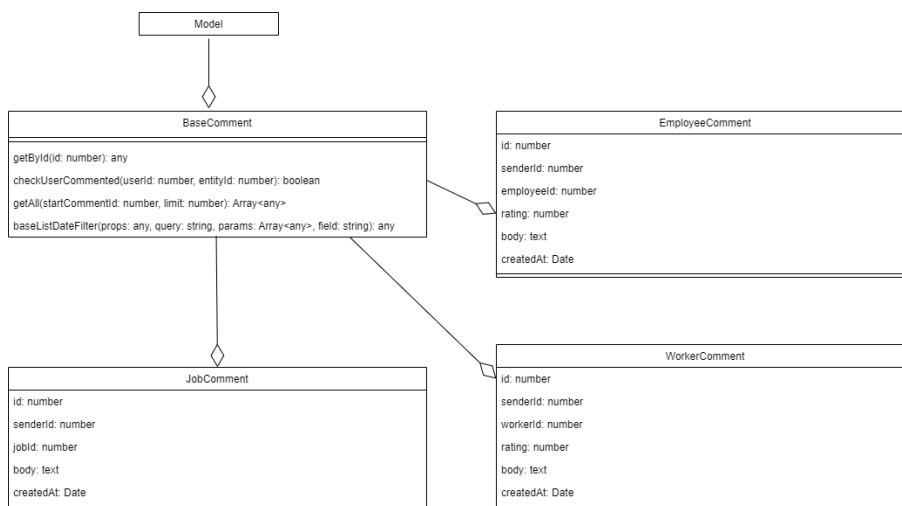


Рисунок 2.4 - Діаграма класів, пов'язаних з коментарями

Для реалізації даної структури класів було використано патерн “Міст”. Після ознайомлення з [29] було визначено переваги цього патерну, що дало можливість покращити розробку. Найдоцільніше було його використати саме

тут, бо для продавця, виконавця та замовлення будуть доступні додавання та відповідання на коментарі, а також оцінювання цих сутностей. Тому логічним буде той факт, що зберігання, відображення та обробка будуть подібні, але матимуть різні поля, різні значення і приховану. Наприклад, оцінка або рейтинг буде лише в коментарях робітника чи виконавця, а в завданні – не буде, адже це непотрібне поле для даного класу.

BaseComment – базовий клас, який є батьківським для всіх класів пов’язаних з коментарями.

Клас містить наступні методи:

- **getById** – метод, який відповідає за отримання інформації про коментар по його ідентифікатору;
- **checkUserCommented** – метод, який перевіряє чи робив користувач прямий коментар до сутності раніше;
- **baseListDateFilter** – метод, який генерує частину sql-запиту на вибірку коментарів у певному часовому діапазоні;
- **getAll** – метод, який дозволяє отримувати коментарі пачками, тобто реалізовує нескінченну пагінацію.

JobComment – клас, який відповідає за роботу з коментарями до завдань.

EmployeeComment – клас, який відповідає за роботу з коментарями до замовників.

WorkerComment – клас, який відповідає за роботу з коментарями до виконавців завдань.

2.2 Розробка бази даних системи

Для збереження інформації у додатку використовується MySQL. Це система управління базами даних, яка базується на реляційній моделі. Файли та всі медіадані зберігаються на сервері, у відповідному каталозі.

База даних системи складається з двадцяти однієї таблиці:

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

1. migrations – таблиця, яка містить інформацію про міграції.
2. system_options – таблиця, яка містить інформацію про системні параметри, які є доступними для швидкого редагування адміністратором.
3. users – таблиця, яка містить дані про користувачів.
4. password_reset_links – таблиця, яка містить інформацію про токени за якими користувачі можуть скинути паролі до своїх акаунтів.
5. notifications – таблиця, яка містить дані про нотифікації.
6. user_actions – таблиця, яка містить інформацію про незавершені дії користувача, наприклад – це поширення файлів через сокети.
7. jobs – таблиця, яка містить інформацію про задачі.
8. job_requests – таблиця, яка містить інформацію про запити на виконання задач.
9. disputes – таблиця, яка містить інформацію про суперечки до задач.
10. sockets – таблиця, яка зберігає дані про сокети користувачів.
11. get_money_requests – таблиця, яка зберігає запити до адміністраторів на зняття коштів з балансів.
12. payment_transactions – таблиця, яка зберігає дані про усі операції в результаті яких було змінено баланс.
13. server_transactions – таблиця, яка містить інформацію про операції, де проміжним етапом переводу був акаунт власника додатку.
14. chats – таблиця, яка містить дані про усі створені чати в системі.
15. chats_users – таблиця, яка містить дані про зв'язки між чатами та користувачами.
16. messages – таблиця, яка зберігає інформацію про створені повідомлення в чатах.
17. message_contents – таблиця, яка зберігає інформацію про контенти повідомлень, дозволяє реалізовувати редагування повідомлень.
18. worker_comments – таблиця, яка містить інформацію про коментарі робітникам.

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		48

19. job_comments – таблиця, яка містить інформацію про коментарі задачам.

20. employee_comments – таблиця, яка містить інформацію про коментарі замовникам.

21. reply_comments – таблиця, яка містить інформацію про коментарі, які є відповідями на інші коментарі.

Опис кожної з таблиць бази даних наведений у табл. 2.20 – 2.40.

Таблиця 2.20

Опис таблиці “migrations”

Поле	Тип	Призначення
id	int	Ідентифікатор
name	varchar(255)	Назва міграції
run_on	datetime	Час запуску міграції

Таблиця 2.21

Опис таблиці “user_actions”

Поле	Тип	Призначення
id	int	Ідентифікатор
user_id	int	Ідентифікатор користувача
type	text	Тип події
data	text	Додаткові дані події
key	text	Ключ події

Таблиця 2.22

Опис таблиці “system_options”

Поле	Тип	Призначення
id	int	Ідентифікатор
name	varchar(255)	Назва параметру
value	varchar(255)	Значення параметру

Таблиця 2.23

Опис таблиці “jobs”

Поле	Тип	Призначення
id	int	Ідентифікатор
author_id	int	Ідентифікатор автора
title	varchar(255)	Заголовок завдання
price	double	Максимальна ціна
address	text	Адреса початку задачі
description	text	Опис завдання
lat	double	Широта точки початку
lng	double	Широта точки кінця
active	tinyint(1)	Актуальна задача
time_created	timestamp	Час створення

Таблиця 2.24

Опис таблиці “notifications”

Поле	Тип	Призначення
id	int	Ідентифікатор нотифікації
user_id	int	Ідентифікатор користувача
type	varchar(255)	Тип події, яка створила нотифікацію
body	text	Основний текст нотифікації
link	varchar(255)	Посилання нотифікації
title	text	Заголовок нотифікації
created_at	timestamp	Час відправки нотифікації

Опис таблиці “users”

Поле	Тип	Призначення
id	int	Ідентифікатор користувача
email	varchar(255)	Електронна пошта
password	varchar(255)	Захешований пароль
nick	varchar(255)	Псевдонім користувача на платформі
address	varchar(255)	Адреса виконання задач
avatar	varchar(255)	Фото профілю
lat	double	Широта користувача
lng	double	Довгота користувача
profile_verified	tinyint(1)	Перевірений користувач
admin	tinyint(1)	Адміністратор
online	tinyint(1)	Відповідає за відображення приступності користувача в системі
balance	double	Баланс акаунта користувача на платформі
activity_radius	double	Радіус підбору задач
biography	text	Коротка біографія
linkedin_url	varchar(255)	Посилання на лінкедін користувача
instagram_url	varchar(255)	Посилання на інстаграм користувача
phone	varchar(255)	Телефон
time_created	timestamp	Час реєстрації на платформі
time_updated	timestamp	Час останнього оновлення

Таблиця 2.26

Опис таблиці “job_requests”

Поле	Тип	Призначення
id	int	Ідентифікатор
user_id	int	Ідентифікатор виконавця
job_id	text	Ідентифікатор задачі
price	double	Ціна за годину роботи, яку хоче виконавець
execution_time	int	Робочий час, необхідний на задачу
status	text	Статус прогресу
time_created	timestamp	Час створення
time_completed	timestamp	Час успішної здачі
time_rejected	timestamp	Час відхилення запиту на виконання
time_cancelled	timestamp	Час скасування запиту на виконання

Таблиця 2.27

Опис таблиці “job_comments”

Поле	Тип	Призначення
id	int	Ідентифікатор
sender_id	int	Ідентифікатор відправника
job_id	int	Ідентифікатор завдання
body	text	Повідомлення
created_at	timestamp	Час створення коментаря

Таблиця 2.28

Опис таблиці “chats”

Поле	Тип	Призначення
id	int	Ідентифікатор
type	varchar(255)	Тип чату
name	varchar(255)	Назва
avatar	varchar(255)	Основне фото чату

Таблиця 2.29

Опис таблиці “disputes”

Поле	Тип	Призначення
id	int	Ідентифікатор
user_id	int	Ідентифікатор автора суперечки
job_request_id	int	Ідентифікатор запиту виконання завдання
admin_id	int	Ідентифікатор відповідального
right_user_id	int	Ідентифікатор переможця
description	text	Опис проблеми
status	varchar(255)	Статус прогресу
created_at	timestamp	Час створення суперечки
updated_at	timestamp	Час останнього оновлення
completed_at	timestamp	Час вирішення суперечки

Таблиця 2.30

Опис таблиці “password_reset_links”

Поле	Тип	Призначення
id	int	Ідентифікатор
account_id	int	Ідентифікатор юзера
reset_token	varchar(255)	Токен скидання паролю
created_at	timestamp	Час створення

Таблиця 2.31

Опис таблиці “payment_transactions”

Поле	Тип	Призначення
id	int	Ідентифікатор
user_id	int	Ідентифікатор користувача
money	int	Сума зміни балансу
operation_type	varchar(255)	Тип операції
balance_change_type	varchar(255)	Тип зміни балансу
transaction_data	text	Додаткові дані
created_at	timestamp	Час початку операції
finished_at	timestamp	Час завершення операції

Таблиця 2.32

Опис таблиці “message_contents”

Поле	Тип	Призначення
id	int	Ідентифікатор
message_id	int	Ідентифікатор повідомлення
content	varchar(255)	Контент повідомлення
time_edited	timestamp	Час додання вмісту

Таблиця 2.33

Опис таблиці “messages”

Поле	Тип	Призначення
id	int	Ідентифікатор
chat_id	int	Ідентифікатор чату
sender_id	int	Ідентифікатор відправника
type	varchar(255)	Тип повідомлення
hidden	tinyint(1)	Приховане
time_created	timestamp	Час відправки

Таблиця 2.34

Опис таблиці “chats_users”

Поле	Тип	Призначення
id	int	Ідентифікатор
chat_id	int	Ідентифікатор чату
user_id	int	Ідентифікатор користувача
last_viewed_message_id	int	Останнє переглянуте повідомлення
role	varchar(255)	Роль користувача у цьому чаті
typing	tinyint(1)	Чи друкує користувач повідомлення у поточний момент
time_created	timestamp	Час додавання користувача до чату
delete_time	timestamp	Час видалення користувача з чату

Таблиця 2.35

Опис таблиці “employee_comments”

Поле	Тип	Призначення
id	int	Ідентифікатор
sender_id	int	Ідентифікатор відправника
employee_id	int	Ідентифікатор замовника
rating	int	Оцінка
body	text	Повідомлення
created_at	timestamp	Час створення

Таблиця 2.36

Опис таблиці “get_money_requests”

Поле	Тип	Призначення
id	int	Ідентифікатор
sender_id	int	Ідентифікатор користувача
user_transaction_id	int	Ідентифікатор транзакції
money	double	Сума виводу
platform	varchar(255)	Ключ сокету
status	varchar(255)	Статус прогресу
body	text	Додаткові дані, необхідні для виведення
created_at	timestamp	Час створення запиту на виведення коштів
done_at	timestamp	Час виконання запиту на виведення коштів

Таблиця 2.37

Опис таблиці “worker_comments”

Поле	Тип	Призначення
id	int	Ідентифікатор
sender_id	int	Ідентифікатор відправника
worker_id	int	Ідентифікатор робітника
rating	int	Оцінка
body	text	Додатковий опис до коментаря
created_at	timestamp	Час створення коментаря

Таблиця 2.38

Опис таблиці “reply_comments”

Поле	Тип	Призначення
id	int	Ідентифікатор
sender_id	int	Ідентифікатор відправника
parent_id	int	Ідентифікатор елемента до якого дали відповідь
reply_comment_id	int	Ідентифікатор батьківського коментаря
parent_type	varchar(255)	Тип коментаря на який зробили відповідь
body	text	Додатковий опис до коментаря
created_at	timestamp	Час створення коментаря

Таблиця 2.39

Опис таблиці “server_transactions”

Поле	Тип	Призначення
id	int	Ідентифікатор
money	int	Сума зміни балансу
operation_type	varchar(255)	Тип операції
balance_change_type	varchar(255)	Тип зміни балансу
transaction_data	text	Додаткові дані
created_at	timestamp	Час початку операції

Таблиця 2.40

Опис таблиці “sockets”

Поле	Тип	Призначення
id	int	Ідентифікатор
user_id	int	Ідентифікатор юзера
chat_id	int	Ідентифікатор чату який переглядає користувач
socket	varchar(255)	Ключ сокету

Таким чином, дана структура бази даних описує усі дані, які зберігаються в проєкті.

2.3 Проєктування та реалізація алгоритмів системи

Проєктування та реалізація алгоритмів системи відповідають за автоматичні перевірки, обробку даних та автоматичне прийняття рішень системою, що забезпечує ефективність та надійність у виконанні завдань користувачами. Ця платформа створена для надання користувачам можливості створювати задачі, виконувати їх, отримувати винагороду та спілкуватися у вбудованому чаті. Алгоритми системи забезпечують надійне

управління завданнями, безпечність фінансових транзакцій та захист платформи від потенційних загроз та шахрайств.

Діаграма активності для повної реєстрації акаунту з його авторизацією наведена на рис. 2.5. Ця діаграма відображає послідовність дій користувача для реєстрації у додатку та заповнення додаткових даних для верифікації та можливості взаємодіяти з платформою. Процес реєстрації не є складним. Він складається лише з двох етапів: власне реєстрації та заповнення додаткових даних. Процес розбитий на два підпроцеси, тому що користувачам дуже часто потрібно спочатку розібратись з тим чи потрібна їм платформа, а вже потім виконувати операції іншого характеру, тому на діаграмі (див. рис. 2.5) можна побачити здебільшого перевірки та перенаправлення користувача.

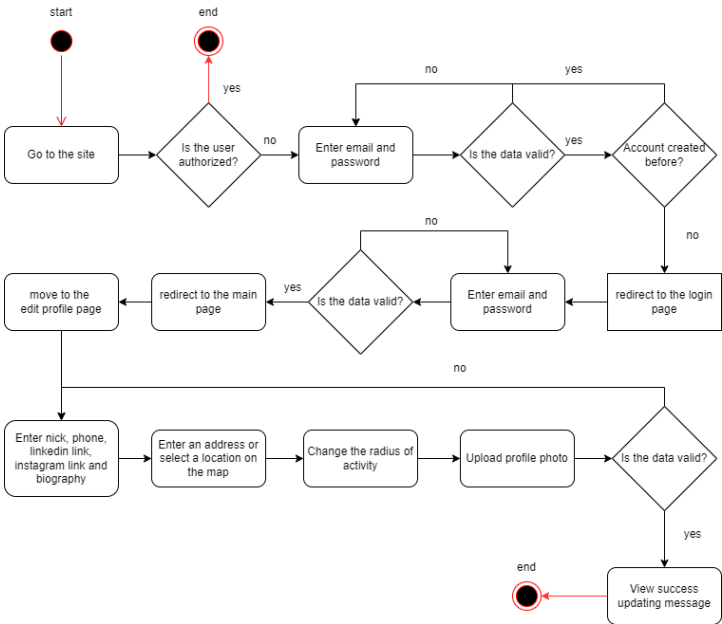


Рисунок 2.5 - Діаграма активності для повної реєстрації акаунту

Діаграма активності для створення задачі представлена на рис. 2.6. Ця діаграма відображає послідовність дій авторизованого користувача для створення нової задачі. В основному процес складається з перевірок, єдиною особливістю є відсутність модерації зі сторони адміністратора. Це є ключовим моментом, бо задачі іноді потрібно вирішувати швидко, а ручна модерація є досить тривалим процесом.

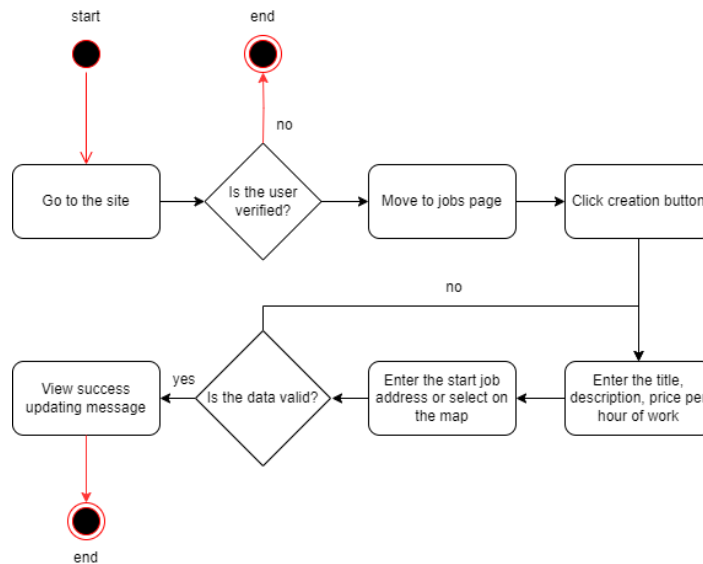


Рисунок 2.6 - Діаграма активності для створення задачі

Діаграма активності для створення пропозиції виконання завдання наведена на рис. 2.7. Ця діаграма відображає послідовність дій авторизованого користувача для створення пропозиції щодо виконання завдання. Платформа є простою у використанні, тому користувачу достатньо просто знайти привабливе для нього завдання, після чого йому достатньо просто натиснути на кнопку для надсилання запиту на виконання і дочекатись відповіді від користувача.

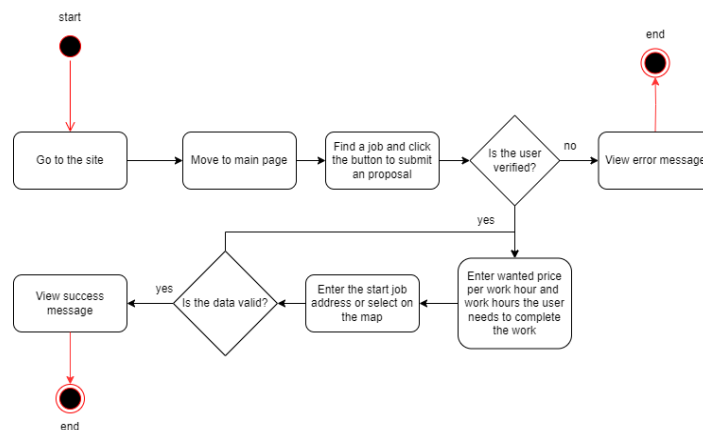


Рисунок 2.7 - Діаграма активності для створення пропозиції виконання задачі

Діаграма активності для прийняття пропозиції виконання задачі представлена на рис. 2.8. Ця діаграма відображає послідовність дій авторизованого користувача для прийняття пропозиції на виконання свого завдання. Процес створення запиту на виконання задачі складається лише з

натискання на одну кнопку, тому і процес для підтвердження виконання теж складається лише з натискання однієї кнопки. Користувачу достатньо переглянути пропозицію, можливо, зв'язатись з виконавцем і вирішити, чи підходять йому умови виконання і чи симпатизує йому виконавець.

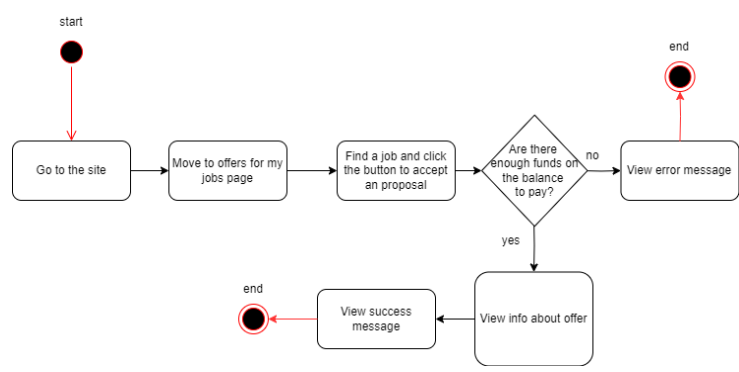


Рисунок 2.8 - Діаграма активності для прийняття пропозиції виконання задачі

Діаграма активності для зміни статусу виконання власної задачі наведена на рис. 2.9. Ця діаграма відображає послідовність дій користувача для зміни статусу виконання власного завдання. Даний процес є важливим, оскільки він надає користувачам розуміння того, що саме відбувається з задачею і фактично підтверджує статус прогресу. Наприклад, після створення задачі виконавець вже завершив виконання і хоче закінчити все та отримати кошти. Для цього він не має чекати, коли користувач дізнається про це, зайде на платформу і виконає оплату. Він може просто натиснути на кнопку зміни статусу для завершення, замовник побачить зміну і вже прийме або відхилить запит.

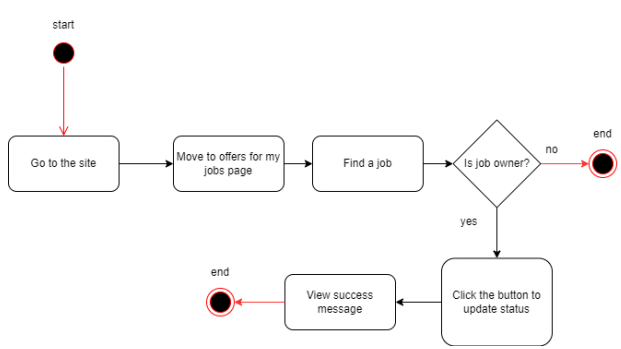


Рисунок 2.9 - Діаграма активності для зміни статусу виконання власної задачі

Діаграма активності для зміни статусу виконання задачі, виконавцем якої є цей користувач, представлена на рис. 2.10. Ця діаграма відображає

послідовність дій авторизованого користувача для зміни статусу завдання, виконавцем якого є цей користувач. Вона включає авторизацію, перегляд списку завдань, вибір конкретного завдання, зміну його статусу, підтвердження та збереження змін.

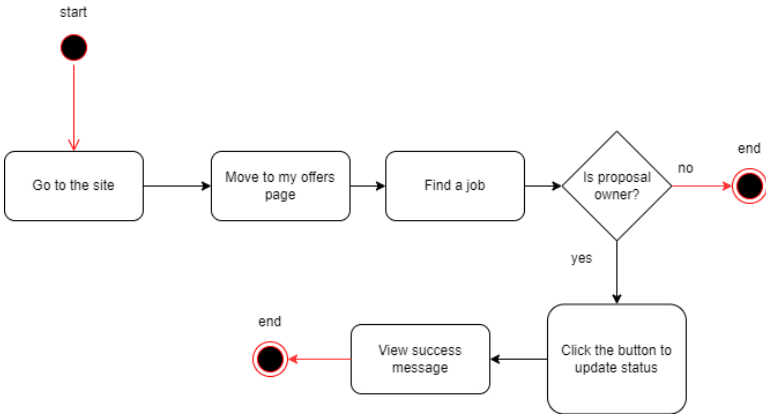


Рисунок 2.10 - Діаграма активності для зміни статусу виконання задачі, виконавцем якої є цей користувач

Діаграма активності для вирішення конфлікту наведена на рис. 2.11. Ця діаграма відображає послідовність дій адміністратора для вирішення конфлікту. Процес конфлікту технічно складається з двох етапів: створення конфлікту користувачами та його вирішення адміністратором. Решта блоків – це додаткові можливості для адміністратора, щоб краще розібратись у ситуації та прийняти правильне рішення.

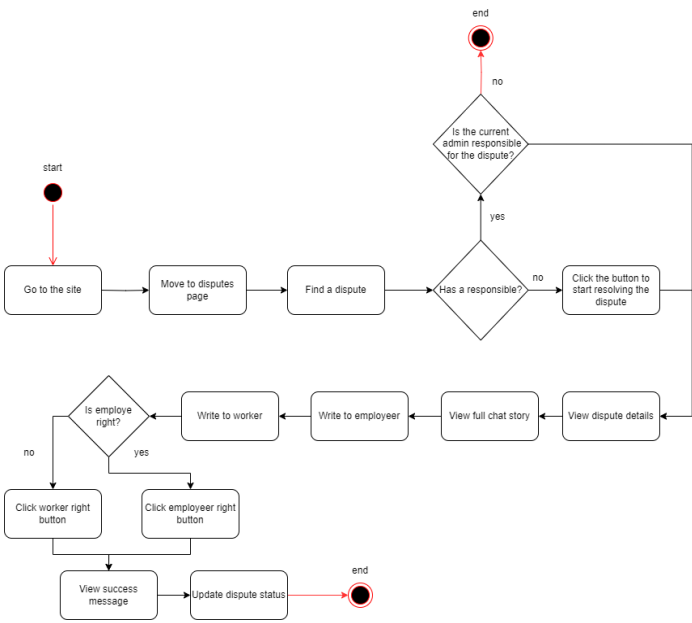


Рисунок 2.11 - Діаграма активності для вирішення конфлікту

Діаграма активності для виведення коштів з платформи представлена на рис. 2.12. Ця діаграма відображає послідовність дій верифікованого користувача для виведення коштів з платформи.

Для того, щоб вивести кошти, користувачу достатньо перейти на сторінку, де є ця операція, ввести суму для виводу і підтвердити операцію, вказавши, куди саме потрібно надіслати кошти. Після цього буде змінено баланс акаунту та надіслано кошти на реальний фінансовий носій через деякий час.

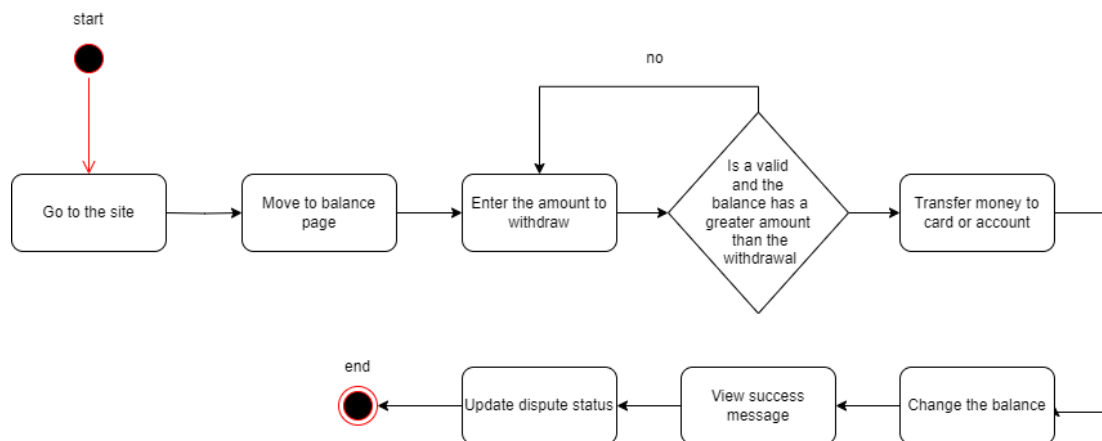


Рисунок 2.12 - Діаграма активності для виведення коштів з платформи

Діаграми активності, які були згадані, допомогли зрозуміти послідовність дій та взаємодію користувачів з системою.

Також важливими складовими проєктування є діаграми послідовності. Вони є корисними, оскільки забезпечують чітке уявлення про те, як відбуваються взаємодії між користувачами та системою протягом виконання певних процесів. Вони дозволяють візуалізувати порядок виклику методів і передачу повідомлень між об'єктами, що допомагає виявити можливі помилки або неефективності в логіці роботи платформи.

Перевага використання діаграм послідовності полягає в тому, що вони допомагають краще розуміти взаємодії між компонентами та користувачами системи. Вони забезпечують наочність та деталізацію, які важко досягти за допомогою простого текстового опису. Тому було прийнято рішення реалізувати ці діаграми для основних процесів.

Платформа реалізована для спрощення різних процесів. Одним з таких процесів є виконання задачі. Користувачі повинні відправляти запити на виконання задачі, після чого замовники підтверджують її. Далі надсилається нотифікація виконавцю, і він розпочинає виконання, змінюючи статус залежно від прогресу. Цей процес відтворено у формі діаграми послідовності, наведеної на рис. 2.13.

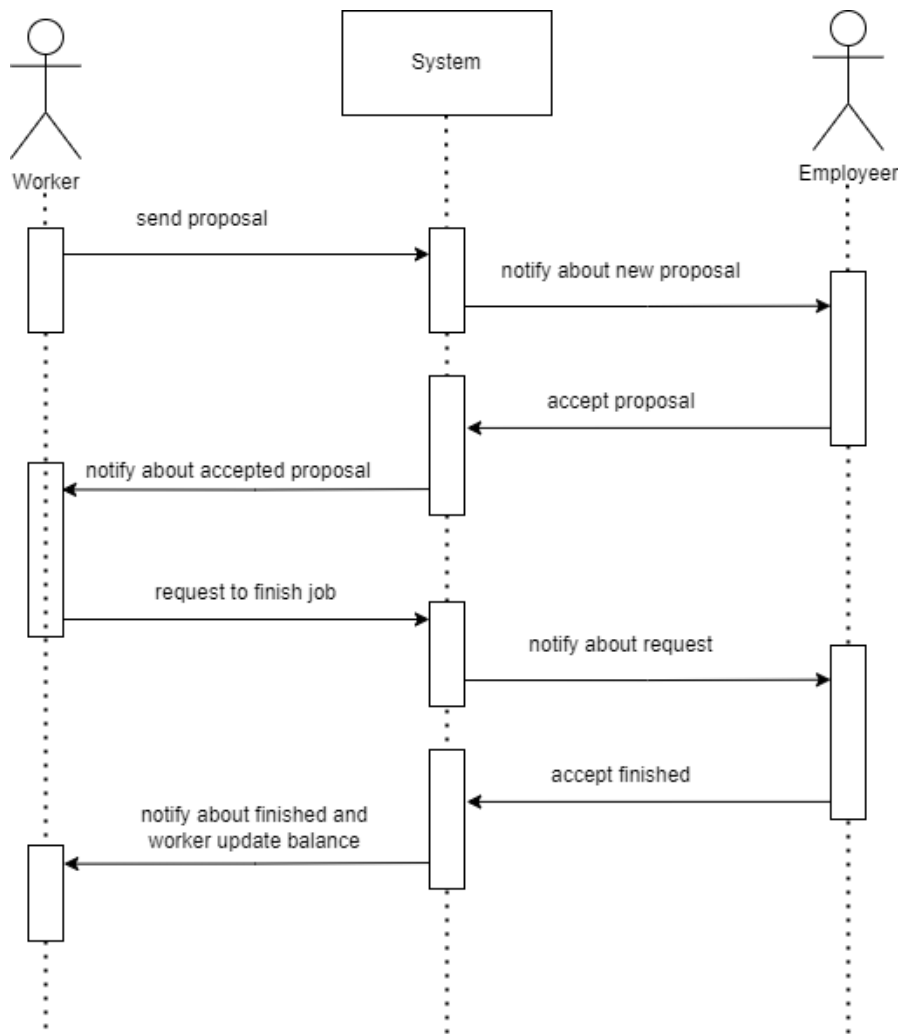


Рисунок 2.13 - Діаграма послідовності виконання задачі

Ще одним ключовим процесом платформи є вирішення суперечок. Як тільки виникає ситуація, коли користувачі не можуть самостійно прийняти рішення, вони можуть звернутися за допомогою до адміністраторів. У такому випадку адміністратор розглядає ситуацію і вирішує, що робити з коштами. Цей процес візуалізовано у формі діаграми послідовності, представленої на рис. 2.14.

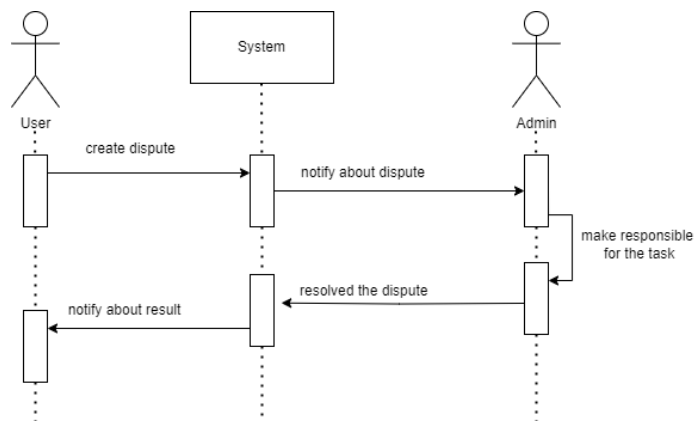


Рисунок 2.14 - Діаграма послідовності вирішення суперечок

Виведення коштів з системи є не менш важливим процесом платформи. Робота з коштами – це надзвичайно важливе завдання платформи, тому було б безглуздом не реалізувати ручну обробку переводень у випадках, коли під час автотранзакцій виникають помилки. Тому, при виведенні коштів, користувачу вони можуть прийти не відразу, а через деякий час. Транзакція буде відображатись зі статусом у процесі до тих пір, поки адміністратори не розв’яжуть проблему і не виконають транзакцію вручну. Процес також візуалізовано у формі діаграми послідовності, наведеній на рис. 2.15.

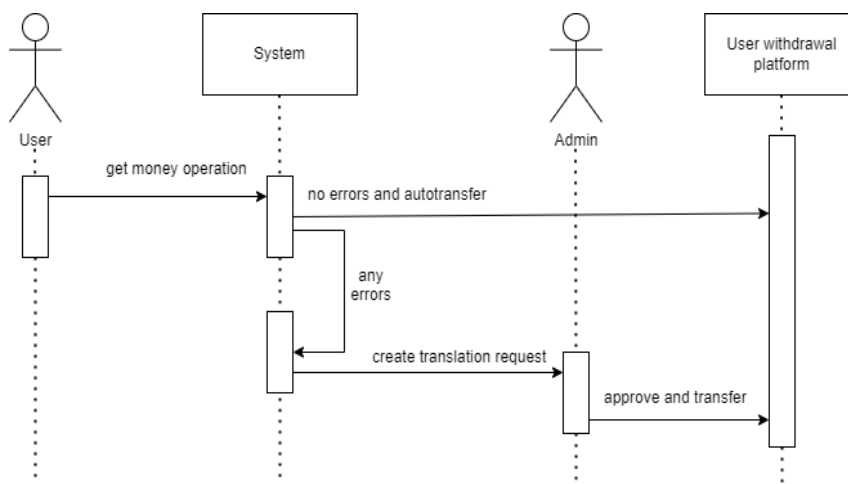


Рисунок 2.15 - Діаграма послідовності виведення коштів з системи

Діаграми послідовності допомогли з виявленням та розв’язуванням потенційних проблем ще до того, як вони проявилися в реальному середовищі. Вони дозволили ідентифікувати зайві або пропущені кроки у процесах, оптимізувати взаємодії та забезпечити більш ефективне використання ресурсів системи.

2.4 Реалізація платформи

Дана платформа складається з серверної та клієнтської частин, тому одним з основних етапів є авторизація користувача на ній. Ключовими фрагментами є логіка самої авторизації, надсилання запитів на сервер та визначення відправника запиту.

Розглянемо, для початку, функцію, яка відправляє запит авторизації користувача на сервер.

```
const handleSignIn = async () => {
  const { token, user } = await request({
    url: login.url(),
    type: login.type,
    data: { email, password, rememberMe },
    convertRes: login.convertRes,
  });

  localStorage.setItem("token", token);
  setSuccess("Logged in successfully");
  setSessionUser(user);
  redirect("/");
};
```

Як видно з коду, функція використовує додаткову асинхронну функцію `request`. Ця функція приймає об'єкт, що містить наступні поля: `url`, `data` та `convertRes`, їх розглянемо пізніше. Ці поля були отримані з об'єкта `login`, який зберігається у файлі `requests.js` теки `src` клієнтської частини.

Сам об'єкт має наступний вигляд:

```
login = {
  url: () => "login",
  type: "post",
  convertRes: (res) => {
    const token = res.headers.authorization.split(" ")[1];
    return { token, user: { ...res.data.user } };
  }
};
```

Як можна побачити, об'єкт має приблизний формат, який приймає метод `requests`. Більшість функцій, які відповідають за надсилання запитів на сервер, використовують подібні об'єкти. Вони зібрані в одному файлі для зручної взаємодії та швидкого коригування їх. Структура об'єкта-запиту складається з наступних полів:

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

- url – функції для генерації шляху запиту. Функція була розроблена для генерації посилань з різними параметрами, наприклад ідентифікаторів елементів чи інформацій про параметри таблиць в адміністраторів;
- type – це тип запиту на сервер, зазвичай або post, або get;
- convertData – це функція, яка приймає дані та на основі них генерує об’єкт, для надсилання на сервер у тілі цього запиту;
- convertRes – це функція яка обробляє результат та повертає дані у потрібному форматі для частини, яка викликала функцію.

У об’єкті login можна звернути увагу, що функція convertRes, отримуючи відповідь, аналізує заголовок та витягує звідти токен, який повертається разом з даними користувача. Далі, вже функція, яка викликала запит на сервер, зберігає токен в локальному сховищі.

Повернемося до функції request. Це одна з основних функцій клієнтської частини, оскільки усі запити на сервер виконуються саме за її участю. Вона отримується за допомогою хука useAjaxRequest, який лише приймає функцію для обробки помилок – onError.

Нижче наведено реалізацію хука useAjaxRequest.

```
const useAjaxRequest = ({ onError }) => {
  return async function ({
    url,
    type = "get",
    data = {},
    onSuccess = () => {},
    convertRes = () => {},
  }) {
    try {
      let res = type === "get" ?
        await axios.get(`${config.API_URL}/${url}`) :
        await axios.post(`${config.API_URL}/${url}`, data);
      const convertedData = convertRes ? convertRes(res) : res;
      onSuccess(convertedData);
      return convertedData;
    } catch (err) {
      const res = err.response;
      onError(res && res.status && res.data && res.data.error ?
        res.data.error : err.message);
      throw new Error(err.message);
    }
  };
};
```

Функція, яку повертає хук, слугує обгорткою для усіх запитів до сервера. Як можна побачити, у жодному рядку хука немає витягування токена зі сховища. Натомість, є виклик функції `axios`, яка зберігається в допоміжних функціях додатка.

Розглянемо вміст файлу з допоміжною функцією `axios`.

```
import axios from "axios";
import mainConfig from "config";
const api = axios.create({
  baseURL: mainConfig.CLIENT_URL,
  withCredentials: true,
  headers: {Accept: "application/json"},
});
api.interceptors.request.use((config) => {
  const token = localStorage.getItem("token");
  if (token) config.headers.Authorization = `Bearer ${token}`;
  return config;
});
export default api;
```

Як можна побачити, уся логіка відбувається саме тут. Спочатку створюється інстанція `Axios` для роботи з API. З налаштувань видно, що кожен запит, зроблений через створений екземпляр, буде мати базову URL-адресу, приймати JSON-відповіді та надсилати куки. Далі, до об'єкта додається інтерцептор для обробки запитів. Інтерцептор – це алгоритм, який виконується перед надсиланням запиту. У цьому випадку він перевіряє наявність токена в локальному сховищі. Якщо токен знайдений, він додається до заголовків запиту для авторизації.

Таким чином відбувається генерування та обробка усіх HTTP-запитів від клієнта до сервера.

Розглянемо тепер логіку серверної частини. Починається вона з файлу `index.js`, де відбувається підключення та налаштування усіх модулів додатка – бази даних, маршрутизаторів та сокетів. Код файлу наведено у Додатку Б.

Серверна частина побудована на патерні MVP, який робить написання коду гнучким та легким у розумінні та розділенні відповідальності функцій. Наприклад, запит надсилається на посилання `"/login"`, тому його перехоплює метод `login` контролера `UserController`. Основною умовою успішного

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		68

виконання цього запиту є неавторизований браузер з якого користувач намагається увійти на платформу. Для виконання цієї перевірки слугує міدلвар, який відпрацьовує до передачі до методу login. Міدلвар генерується за допомогою функції generateIsNotAuth, яка нічого не приймає.

Нижче наведено реалізацію функції generateIsNotAuth.

```
function generateIsNotAuth() {  
  return function (request, response, next) {  
    const authorization = request.headers.authorization;  
    if (!authorization) return next();  
    const token = authorization.split(" ")[1];  
    const userId = validateToken(token);  
    if (userId)  
      return response.status(403).json({message: "Forbidden"});  
    return next();  
  };  
}
```

Як можна побачити, метод перевіряє наявність авторизаційного токenu у запиті. За його наявності відбувається його дешифрування та отримання інформації про ідентифікатор користувача. Якщо токен валідний і ідентифікатор користувача в ньому наявний, то повертається помилка зі статусом заборони доступу, в іншому випадку алгоритм переходить до наступної функції, яка прив'язана до посилання.

Метод login контролера користувачів використовує декоратор errorHandler. Даний декоратор використовується у всіх методах, які приймають та обробляють дані від клієнтської частини без проміжних методів контролера. Цей декоратор перехоплює усі можливі помилки та повертає їх на сторону клієнта у вигляді статусу і зрозумілого повідомлення про помилку, а не кладе сервер, як це буває у випадку неочікуваних помилок.

Метод login зчитує параметри, які йому надійшли, визначає тривалість існування токenu на основі наявності поля rememberMe та його значення, після цього отримує інформацію про користувача за логіном та паролем, генерує новий токен користувача і зберігає інформацію про вхід в акаунт та надсилає нотифікацію про вхід. У відповідь повертає дані про користувача та його авторизаційний токен в заголовках.

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		69

Нижче наведено реалізацію методу login.

```
login = (req, res) =>
  this.errorWrapper(res, async () => {
    const { email, password, rememberMe } = req.body;
    const duration = rememberMe
      ? process.env.JWT_REMEMBER_ACCESS_LIFETIME
      : process.env.JWT_DEFAULT_ACCESS_LIFETIME;

    const user = await this.userModel.findByPasswordAndEmail(email, password);
    const userId = user.id;

    const token = jwt.sign({ userId }, process.env.SECRET_KEY, {
      expiresIn: duration,
    });

    this.createLoginNotification(userId);
    res.set("Authorization", `Bearer ${token}`);

    return this.sendResponseSuccess(
      res,
      "User authorized successfully",
      { userId },
      200
    );
  });
```

У даному методі виконано звернення до моделі userModel, а саме до її методу findByPasswordAndEmail. Цей метод виконує підключення до бази даних, де шукає користувача, а потім, перевіряє збіг пароля, який надійшов з клієнтської частини та захешованого паролю користувача, котрий зберігається в базі та на основі цього повертає дані про користувача, або генерує помилку, яка перехоплюється подібним декоратором, як в контролері, і перекидає цю помилку далі – контролеру. Даний декоратор був дуже корисним саме на етапі розробки, бо при отриманні помилки на стороні клієнта, можна було зайти в файл журналу та переглянути детальнішу інформацію, який метод викинув помилку і в якій моделі. У останніх версіях проєкту, логування відсутнє, бо цей модуль перестав бути актуальним. Лістинг методу findByPasswordAndEmail наведено в Додатку В.

Будь-яка задача, будь-які зусилля завжди вимагають плати, тому платформа надає можливість взаємодії з реальними коштами. Реалізувати

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		70

поповнення балансу можна було кількома варіантами, і обрано найпростіший для реалізації. Суть алгоритму полягає в тому, що стороння фінансова платформа бере участь лише для поповнення балансу та виведення коштів з платформи, усі інші фінансові операції виконуються саме в додатку. Це дозволило значно спростити логіку та зменшити кількість помилок на етапі розробки. Система дозволяє поповнювати та виводити кошти за допомогою Stripe та PayPal.

Загальний алгоритм для обох операцій подібний для обох платформ, тому достатньо навести по одному прикладу алгоритму взаємодії з фінансовою платформою.

Для початку розглянемо алгоритм поповнення балансу через Stripe. Користувач заповнює форму з картковими даними Stripe, після чого відправляється запит на оплату в Stripe. Після успішної оплати Stripe генерує токен оплати, який надсилається на сервер разом з іншими даними про оплату. На сервері відбувається перевірка отриманих даних, і якщо оплата відбулася успішно, то баланс користувача на платформі збільшується на відповідну суму.

Нижче наведено реалізацію методу `handleSubmit`.

```
const handleSubmit = async (event) => {
  if (!amount || isNaN(Number(amount))) {
    setAmountError("Invalid field");
    return null;
  }
  const cardElement = elements.getElement(CardElement);
  const { token, error } = await stripe.createToken(cardElement);
  if (error) return;
  const newBalance = await request({
    url: stripeCharge.url(),
    type: stripeCharge.type,
    data: stripeCharge.convertData(amount, token),
    convertRes: stripeCharge.convertRes,
  });

  setSuccess("Operation successful");
  setSessionUser((data) => ({ ...data, balance: newBalance }));
  onComplete();
  setAmount("");
};
```

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		71

У даній функції використовується об'єкт stripe, який є результатом виклику хука useStripe, бібліотеки @stripe/react-stripe-js. Цей об'єкт відповідає за взаємодію зі Stripe.

Формою, яку заповнює користувач, є результат інтеграції бібліотеки в платформу. При переглянуті згенерованого html, можна помітити, що Stripe вбудовує в код платформи фрейм, який виконує операції безпосередньо з фінансовою платформою. Це забезпечує надійність та конфіденційність інформації, оскільки дані не обробляються та не зберігаються через розроблену систему, а прямо передаються до Stripe.

Нижче наведено реалізацію інтеграцію компонентів Stripe.

```
<Elements stripe={stripePromise}>
  <form onSubmit={handleSubmit}>
    <div className="card mb-0">
      <div className="card-body">
        <CardElement
          options={...}
        />
      </div>
    </div>

    <div className="w-100 mt-4">
      <div className="card mb-0">
        <div className="card-body">
          <Input
            type="text"
            label="Money to replenishment"
            placeholder="Enter money to replenishment"
            value={amount}
            onChange={(e) => amountChange(e.target.value)}
            error={amountError}
          />
          <button
            className="w-100 btn btn-primary"
            type="submit"
            disabled={!stripe || loading}
          >
            {loading ? "Loading..." : "Pay"}
          </button>
        </div>
      </div>
    </div>
  </form>
</Elements>
```


У даному фрагменті показано реалізацію відображення форми для оплати. Змінна `stripePromise` – це об’єкт, який створюється при підключенні Stripe через метод `loadStripe` бібліотеки `@stripe/stripe-js`. Цей метод приймає публічний ключ від акаунту Stripe, на який надходять кошти від користувачів під час поповнення балансу. Компоненти `Elements` та `CardElement` представляють собою елементи бібліотеки `@stripe/react-stripe-js`, які відповідають за відображення фрейму форми оплати. При виклику методу оплати дані з фрейму зчитуються за допомогою `elements.getElement`. Проте прямого доступу до них немає, тому вони передаються в наступну функцію – `stripe.createToken`, яка проводить оплату.

Після надсилання запиту на сервер відбувається оновлення форми, даних про користувача і відображення повідомлення про успішне виконання.

На сервері виконується метод `stripeBalanceReplenishment`, який перевіряє валідність токена та ціну яку заплатив користувач, для поповнення балансу на відповідну суму. Потім ця сума змінюється, а результат з оновленим балансом відправляється назад.

Нижче наведено реалізацію методу `stripeBalanceReplenishment`.

```
stripeBalanceReplenishment = async (req, res) =>
  this.errorWrapper(res, async () => {
    const { userId } = req.userData;
    const { amount, token } = req.body;
    const charge = await stripe.charges.create({
      amount: amount,
      currency: "usd",
      source: token.id,
      description: "Example Charge",
    });

    const newBalance = await this.userModel.addBalance(userId, amount);

    await this.paymentTransactionModel.createReplenishmentByStripe(
      userId,
      Number(amount).toFixed(2)
    );

    return this.sendResponseSuccess(res, "Balance updated success-fully",
      {newBalance});
  });
```

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		73

Метод використовує об'єкт stripe, який є результатом виклику функції stripe цієї ж бібліотеки. Цей метод приймає лише приватний ключ від акаунта через Stripe, на який відбуваються всі оплати. Сам же метод виконує перевірку, після чого поповнює баланс і створює новий запис в історії оплат на платформі. Після цього він повертає на сторону клієнта інформацію про оновлений баланс користувача.

Наступним варто розглянути логіку виводу коштів з платформи. Цього разу буде використано PayPal для прикладу. Для здійснення цієї операції користувачу достатньо ввести суму, яку він хоче вивести, а також ідентифікатор, пошту чи номер телефону акаунту PayPal, на який повинні надійти кошти. Після натискання відповідної кнопки вся дія виконується на сервері, а клієнтська частина додатка просто оновлює дані та виводить повідомлення про успішну операцію. Раніше було згадано, що будь-яке виконання завдання має свою ціну, тому виведення коштів супроводжується комісією, про яку користувача попереджають під час виведення коштів з платформи.

Таким чином, спочатку метод на сервері розраховує комісію, яку потрібно відняти від суми, яку знімає користувач. Потім він списує зазначену користувачем суму, враховуючи комісію, з акаунту PayPal власника платформи і переводить її до користувача. Якщо під час операції виникає якась помилка, то вона фіксується, і створюється запит на розв'язання проблеми вручну. За це вже відповідають адміністратори, які вирішують запит шляхом ручного втручання, поповнення балансу акаунта власника (якщо проблема полягає в надлишку коштів), або змінюють дані PayPal отримувача і повторно пробують виконати операцію.

Створення задачі є простою операцією і схожою на заповнення профілю користувача, тому наступним розглянутим фрагментом коду будуть етапи виконання задачі.

Першим кроком є надсилання запиту на виконання роботи. Користувач заповнює форму, де вказує необхідні дані, та після натискання кнопки

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		74

відправляє їх на сервер. Він зберігає пропозицію до задачі та сповіщає замовника про неї.

Нижче наведено реалізацію методу create.

```
create = async (req, res) =>
  this.errorWrapper(res, async () => {
    const { jobId, price, time } = req.body;

    const userId = req.userData.userId;
    const createdRequest = await this.jobProposalModel.create(
      jobId, userId, price, time
    );

    const user = await this userModel.getFullUserInfo(userId);
    const job = await this.jobModel.getById(jobId);

    this.sendProposalNotification({...});

    return this.sendResponseSuccess(res, "Request created success-fully",
      {requestId: createdRequest.id});
  });
```

Наступним кроком є підтвердження чи відхилення пропозиції замовником. Для цього використовується метод асепт контролера jobProposalController, який використовує базовий метод __changeStatus.

Нижче наведено реалізацію методу __changeStatus.

```
__changeStatus = async (req, res, validationType, validationCallback) =>
  this.errorWrapper(res, async () => {
    const { proposalId } = req.body;

    const proposal = await this.jobProposalModel.getById(proposalId);
    const jobId = proposal.jobId;

    const userId = req.userData.userId;
    const validation =
      validationType == "job-owner"
        ? this.__jobOwnerCheck
        : this.__proposalOwnerCheck;
    const resValidation = await validation(res, proposalId, jobId,
      userId);

    if (resValidation) return resValidation;
    return await validationCallback(proposalId);
  });
```

Метод приймає тип доступу, який повинен мати користувач до задачі.

Якщо це власник задачі, тоді тільки він зможе виконати запит, який

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		75

надійшов, робітник же – не зможе нічого зробити, щоб якимось чином змінити статус задачі на бажаний. Після валідації викликається вже функція, яка була передана як зворотний виклик.

Метод асерт передає функцію, яка обчислює суму, яку потрібно зарезервувати у користувача, та віднімає її з балансу. У випадку нестачі коштів користувача, він повідомляє його про це. Після цього змінюється статус пропозиції, і виконавець отримує про це сповіщення. Після виконання операцій метод повертає клієнту інформацію про оновлений баланс та угоду.

Нижче наведено метод асерт.

```
accept = async (req, res) =>
  this.__changeStatus(req, res, "job-owner", async (proposalId) => {
    ...

    const newBalance = await this.userModel.rejectBalance(
      jobAuthorId,
      Number(pricePerHour * hours).toFixed(2)
    );

    await this.jobProposalModel.accept(proposalId);
    const performance = await
    this.userModel.getFullUserInfo(proposal.userId);

    ...

    return this.sendResponseSuccess(res, "Proposal accepted success", {
      proposal,
      newUserBalance: newBalance,
    });
  });
```

Метод reject просто змінює статус та дає нотифікацію користувачу, який відправив пропозицію. Нижче наведено реалізацію цього методу.

```
reject = async (req, res) =>
  this.__changeStatus(req, res, "job-owner", async (proposalId) => {
    const proposal = await this.jobProposalModel.reject(proposalId);

    const userId = req.userData.userId;
    const user = await this.userModel.getFullUserInfo(userId);

    this.rejectJobProposal(
      {
        proposalId: proposal.id,
        senderNick: user.nick,
```

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		76

```

        senderEmail: user.email,
        jobTitle: proposal.title,
      },
      proposal.userId
    );

    return this.sendResponseSuccess(res, "Proposal rejected success", {
      proposal,
    });
  });
});

```

Платформа надає ще чотири методи для зміни статусів. Для замовника – `requestToCancel` та `acceptCancelled`, а для виконавця – `requestToComplete` та `acceptCompleted`. Користувачам достатньо просто натискати кнопки та підтверджувати дії, а серверна частина буде виконувати методи, подібні до `reject`, для зміни статусів та оповіщень.

На останок варто розглянути методи роботи чату. Надсилання повідомлень є головною задачею будь-якого чату, тому сфокусуємося саме на цій операції.

Самим простим варіантом є звичайне надсилання повідомлення, де клієнт просто вводить текст та різні смайлики. Для текстового поля було вибрано елемент `div` з атрибутом `contentEditable`, який дозволяє змінювати внутрішній контент блоку. Перевагою такого підходу є простота візуального налаштування блоку, адже звичні текстові поля не завжди можна налаштувати так, як потрібно.

Обирати емоції користувач може після натискання смайлика на панелі, який відповідає за надсилання повідомлень. Після цього з'являється вікно з можливістю вибору символу, який потрібно додати до текстового поля.

Для вставки символу на місце курсора потрібні були додаткові операції, тому функція вставки виглядає трохи складнішою, ніж просто додання тексту до `innerHTML` блоку.

Нижче наведено реалізацію функції `handleEmojiClick`.

```

const handleEmojiClick = (emoji) => {
  const { startContainer, startOffset, endContainer, endOffset } =
    savedSelection;

```

```

const range = document.createRange();
range.setStart(startContainer, startOffset);
range.setEnd(endContainer, endOffset);
range.deleteContents();

const textNode = document.createTextNode(emoji);
range.insertNode(textNode);

range.setStartAfter(textNode);
range.setEndAfter(textNode);

const selection = window.getSelection();
selection.removeAllRanges();
selection.addRange(range);

if (selection.rangeCount) {
  const range = selection.getRangeAt(0);
  setSavedSelection({
    startContainer: range.startContainer,
    startOffset: range.startOffset,
    endContainer: range.endContainer,
    endOffset: range.endOffset,
  });
}
};

```

Спочатку відбувається перевірка наявності курсора або виділеного тексту в полі. Якщо збережений курсор або виділення існує, витягуються межі (початкова і кінцева позиції) збереженого вибору. Перевіряється, чи знаходяться елементи збереженого вибору в документі. Якщо вони відсутні (наприклад, видалені), емодзі також додається в кінець текстового поля. Створюється новий об'єкт Range (діапазон), і встановлюються його межі відповідно до збереженого вибору. Видаляється вміст, який знаходиться у межах збереженого вибору. Створюється новий текстовий вузол з емодзі та вставляється на місце видаленого вмісту. Оновлюються межі діапазону так, щоб вони були після вставленого емодзі. Поточний вибір на сторінці оновлюється, очищується, і встановлюється новий вибір з оновленими межами. Збережений вибір оновлюється новими межами, щоб у майбутньому знову можна було вставляти емодзі у правильне місце.

Повідомлення зберігається у форматі блоків, тому видалення зайвих пробільних символів теж має особливу реалізацію.

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		78

Нижче наведено реалізацію функції `handleSendClick`.

```
const handleSendClick = () => {
  let messageContent = textRef.current.innerHTML;
  let prevLength = 0;

  do {
    prevLength = messageContent.length;
    messageContent = messageContent
      .trim()
      .replace(/^(\\s*<div>(?:&nbsp;|<br>|<br\\/>|\\s)*<\\/div>\\s*)*/ , "")
      .replace(/(\\s*<div>(?:&nbsp;|<br>|<br\\/>|\\s)*<\\/div>\\s*)*$/ , "")
      .replace(/^(\\s*&nbsp;\\s*)*/ , "")
      .replace(/(\\s*&nbsp;\\s*)*$/ , "")
      .trim();
  } while (prevLength > messageContent.length);

  if (!messageContent.length) {
    return;
  }

  handleSendTextMessage(messageContent);
  textRef.current.innerHTML = "";
};
```

Спочатку текстове повідомлення видаляє крайні секції з пробілами, або переходами на новий рядок. Потім видаляє пробільні частини, які залишились, або переходи на нові рядки. Після цього очищення метод визначає, чи можна відправляти повідомлення в чат, і очищає текстове поле.

Функція `handleSendTextMessage`, що використовується в самому кінці функції, генерується в батьківському компоненті, а саме на сторінці `Chat` і передається через контекст до необхідного компонента.

Нижче наведено реалізацію функції `handleSendTextMessage`.

```
const handleSendTextMessage = (message) => {
  if (editMessageId) {
    if (message !== editMessageContent) editMessage(editMessageId, message);
    unsetEditMessage();
  } else {
    const dop = { tempKey: randomString() };

    if (activeChat.chatType === "personal") {
      dop["chatId"] = activeChat?.chatId;
      dop["getterId"] = activeChat.userId;
    }
  }
}
```

```

        sendMessage(activeChat.chatId, "text", message, activeChat.chatType,
dop);
    }
};

```

У цій функції визначається, чи потрібно відредагувати повідомлення, чи відправити нове. У випадку нового – створюється додатковий локальний ключ, за яким статус повідомлення буде змінено з “Надсилається” на “Надіслано”. Далі викликається функція `sendMessage`, яка генерується в результаті виклику хука `useChatInit`. Цей хук працює з сокетом та відповідає за отримання і надсилання повідомлень на сервер. При ініціалізації він приймає об’єкт `io`, необхідний для надсилання та отримання повідомлень, а також різні функції, які повинні виконатися при отриманні певних повідомлень від серверної частини.

Нижче наведено реалізацію функції `sendMessage`.

```

const sendMessage = (chatId, typeMessage, content, chatType, dop) => {
    const dataToSend = {chatId, typeMessage, content, chatType, ...dop};

    const dataToInsert = { chatId: chatId,
        type: typeMessage,
        content,
        chatType,
        userId: sessionUser.id,
        inProcess: true,
        timeSended: new Date().toISOString(),
        ...dop,
    };

    io.emit("send-message", dataToSend);
    onGetMessageForSockets(dataToInsert);
};

```

Після надсилання повідомлення користувачу домальовується тимчасове відображення надсилання до того моменту, доки користувач не отримає відповідь від сервера про успішне надсилання повідомлення.

Для отримання повідомлень хук `useChatInit` використовує хук `useEffect`, в якому прив’язуються реакції на повідомлення. Код хука `useEffect` наведено у Додатку Д.

На серверній частині додатка, подія `send-message` перехоплюється та оброблюється функцією `onSendMessage`. Прив’язка відбувається за

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		80

допомогою однієї з функцій вищого порядку `bindFuncToEvent`. Її особливість полягає в тому, що вона використовується у всіх подіях сокета для спрощення коду.

Нижче наведено реалізацію функції `bindFuncToEvent`.

```
const bindFuncToEvent = (event, func) => {
  socket.on(event, async (data) => {
    try {
      await func(data, {
        socket,
        userId,
        user,
      });
    } catch (e) {
      sendError(e.message);
    }
  });
};
```

Як можна побачити, функція не є складною. Вона приймає назву події та функцію, яка має відпрацювати на подію, огортає її в блок `try catch` і при виникненні помилки відправляє інформацію про це.

Метод `onSendMessage` викликає метод створення повідомлення у контролері чату. Після успішного створення відбувається розсилка повідомлення всім користувачам, крім відправника. Відправник отримує іншу подію, що підтверджує успішну операцію, яка замінює статус повідомлення з “Надсилається” на “Надіслано” на стороні клієнта. Лістинг методу `onSendMessage` наведено у Додатку Е, а лістинг методу `createMessage` контролера `chatController` у Додатку Є.

Ще одним варіантом надсилання повідомлення є поширення файлу, або запис відео чи аудіо повідомлення. Цей варіант особливий, оскільки розміри файлів можуть бути дуже великими, і передавати разово ці дані від клієнта до сервера може бути проблематично. Тому було розроблено алгоритм, який розбиває файли на масив `Blob`, які передаються поступово на сервер. Клієнтська частина зберігає повідомлення з масивом `Blob` та згенерованим випадковим ключем. Цей ключ використовується для визначення, яке саме повідомлення треба оновити і який наступний фрагмент потрібно надіслати

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		81

на сервер, або повністю завершити операцію. Також цей ключ важливий для операції скасування надсилання, оскільки за його допомогою можна перервати та видалити більше непотрібний файл.

Функція, яка відповідає за розбиття файлів та генерації масиву для надсилання:

```
async function createMediaActions(data, dataType, filetype, dop) {
  const tempFileKey = randomString();
  let arr = null;
  if (dataType == "media")
    arr = await splitBlob(data, config["BLOB_CHUNK_SIZE"], data.type);
  else if (dataType == "notmedia")
    arr = splitDataIntoChunks(data, config["BLOB_CHUNK_SIZE"], data.type);

  mediaActionsRef.current[tempFileKey] = {
    data: arr,
    percent: 0,
    inQueue: [...arr],
    filetype,
    dop: { ...dop },
  };

  const blobToSend = mediaActionsRef.current[tempFileKey]["inQueue"][0];
  const last = mediaActionsRef.current[tempFileKey]["inQueue"].length ==
1;

  return {
    tempKey: tempFileKey,
    type: filetype,
    data: blobToSend,
    last,
    ...dop,
  };
}
```

Раніше було зазначено, що функція приймає дані, їх тип (медіа чи інше), розширення файлу та інші додаткові дані, якщо вони потрібні при надсиланні. Також всередині генерується тимчасовий ключ, про який було згадано раніше.

Розбиття файлу на масив Blob відбувається в методі, який відповідає за надсилання даних на сервер. Відразу після розбиття відбувається надсилання першого шматка файлу. Після цього, система починає очікувати

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		82

підтвердження збереження і передачі наступного шматка, якщо надсилання ще не завершилося.

Нижче наведено реалізацію функції `sendMedia`.

```
const sendMedia = async (data, dataType, filetype, dop, filename) => {
  const dataToSend = await createMediaActions(data, dataType, filetype,
dop);
  const messageType = indicateMediaTypeByExtension(filetype);
  const content = messageType == "file" ? filename : data;
  const dataToInsert = {
    chatId: dataToSend["chatId"],
    type: messageType,
    content: content,
    chatType: dataToSend["chatType"],
    userId: sessionUser.id,
    inProcess: true,
    timeSended: new Date().toISOString(),
    tempKey: dataToSend["tempKey"],
  };

  onGetMessageForSockets(dataToInsert);
  io.emit("file-part-upload", { ...dataToSend });
};
```

Лістинг коду, який відповідає за надсилання наступного фрагмента файлового повідомлення наведений нижче.

```
io.on("file-part-uploaded", async ({ tempKey, message = null }) => {
  const nextPartData = await onSuccessSendBlobPart(tempKey);

  if (!nextPartData) return;

  if (nextPartData == "success saved" && message) {
    onGetMessageForSockets(message);
    return;
  }
  onUpdateMessagePercent({ tempKey, percent: nextPartData["percent"] });
  setTimeout(() => io.emit("file-part-upload", { ...nextPartData }), 1);
});
```

Щодо серверної частини, метод, який перехоплює інформацію про новий файл, просто викликає метод `__uploadToFile` для оновлення даних контролера чату, визначає, який результат повернути користувачу, і кому відправити інформацію про зміни. Лістинг методу наведено у Додатку Ж.

Нижче наведено реалізацію функції `__uploadToFile`.

```
__uploadToFile = async (userId, key, data, type) => {
  const info = await this.actionModel.getByKeyAndType(
```

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		83

```

        userId,
        key,
        "sending_file"
    );
    let filename = randomString() + "." + type;

    if (!info || !info.data) {
        this.__createFolderIfNotExists(this.__message_folder);
        fs.writeFileSync(this.__message_folder + "/" + filename, data);
        const actionInfo = JSON.stringify({filename});
        await this.actionModel.create(userId, "sending_file", key, actionInfo);
    } else {
        const resParsed = JSON.parse(info.data);
        filename = resParsed.filename;
        fs.appendFileSync(this.__message_folder + "/" + filename, data);
    }

    return filename;
};

```

Метод отримує дані про відправлений файл користувачем. Якщо раніше такої дії не було, то створюється новий файл з отриманими даними. Якщо файл вже існує, то до нього додається нова частина даних. Після цього метод повертає назву згенерованого файлу.

Записані з платформи відео та аудіо повідомлення мають формат медіафайлів, тому логіка з їхнім надсиланням на сервер та розбиттям на масив – однакова. Важливим є сам запис повідомлення.

Починається метод з виклику функції `handleStartRecording`, яка отримується від хука `useRecorder`. Цей хук надає лише один метод, в який передається згенерований файл.

Нижче наведено реалізацію функції `handleStartRecording`.

```

const handleStartRecording = async () => {
    const name = randomString();
    const afterRecording = (blob) => {
        const src = window.URL.createObjectURL(blob);
        const file = { src };
        file["type"] = recordingMediaType === "audio" ? "mp3" : "mp4";
        file["name"] = name + "." + file["type"];
        setFile(file);
    };
    const stopper = await startRecording(
        recordingMediaType,
        () => {

```

```

        setRecording(true);
        timerRef.current = setInterval(
            () => setRecordingTime((prev) => (prev += 100)), 100
        );
        setRecorder(() => stopper);
    }, afterRecording
  );
};

```

У методі `handleStartRecording` викликається функція `startRecording`, яка повертає іншу функцію, що буде виконана при натисканні кнопки зупинки запису користувачем. Функція `startRecording` отримує тип медіа запису, функцію, яка буде виконана після початку запису, та функцію, яка буде виконана після завершення запису.

Для початку функція ініціалізує об'єкт `constraints`. Цей об'єкт використовується потім як параметр для налаштувань доступів програми. Таким чином, якщо тип запису – це відео, то програма запитує ще доступ до відео, а не лише до мікрофона. За отримання медіапотoku відповідає функція `getMedia`, яка просто звертається до методу `navigator.mediaDevices.getUserMedia` і повертає його результат. Після цього створюється об'єкт `MediaRecorder` з отриманим медіапотокom для запису та починається запис за допомогою методу `mediaRecorder.start()`. Після початку відразу ж викликається передана параметром функція `afterStartRecording`. Далі встановлюється обробник події `dataavailable` для об'єкта `mediaRecorder`. І кожного разу, коли стає доступною нова частина даних, масив `voice` очищується і додаються нові дані. Потім встановлюється обробник події `stop` для об'єкта `mediaRecorder`. Коли запис зупиняється, то спочатку зупиняються всі треки медіапотoku. Потім створюється об'єкт `Blob` з типом `audio/mp3`, якщо записувалося тільки аудіо, або `video/mp4`, якщо записувалося відео. Далі отриманий об'єкт `Blob` зберігається у змінну `resBlob`. Після цього викликається функція `afterRecording` з переданим об'єктом `resBlob`, щоб обробити записані дані. Створюється функція `stopper`, яка зупиняє запис, викликаючи метод `mediaRecorder.stop()`, і встановлює `mediaRecorder` в `null`.

Повертається функція stopper, щоб її можна було викликати для завершення запису і відображення вікна з відео чи аудіо для підтвердження надсилання.

Таким чином, було реалізовано основні фрагменти коду платформи для виконання завдань за винагороду.

Висновки до другого розділу

Під час розробки платформи для організації та виконання завдань за винагороду було проведено аналіз всіх можливих сценаріїв використання платформи з урахуванням вимог програмного забезпечення. На основі цього аналізу було спроектовано структуру платформи, яка включає моделювання взаємодії між всіма її складовими частинами.

Особлива увага була приділена проєктуванню бази даних, яка відповідає за надійне зберігання інформації про користувачів, завдання, транзакції та інші важливі дані. Це забезпечує стабільну роботу платформи та швидкий доступ до необхідної інформації для всіх її компонентів. У процесі проєктування також були детально описані алгоритми роботи додатка, які включають послідовні кроки використання функцій платформи.

Основний функціонал додатка передбачає створення завдань користувачами, їхнє розміщення на платформі для виконання іншими користувачами, обговорення умов завдання та вирішення можливих конфліктів. Додатково розглянуто основні функції авторизації, взаємодії користувачів із завданнями, поповнення балансу та основні можливості чату.

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		86

РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ ДЛЯ ОРГАНІЗАЦІЇ ТА ВИКОНАННЯ ЗАВДАНЬ

3.1 Порядок встановлення та налаштування параметрів платформи

Дана платформа складається з двох основних компонентів: серверної частини та клієнтської. Серверна частина відповідає за обробку запитів, управління базою даних та розміщення файлового сховища, тоді як клієнтська частина забезпечує інтерфейс користувача та взаємодію з сервером.

Серверна частина включає не лише серверний код, але й файлове сховище та базу даних, для зменшення обсягу роботи. Це дозволяє централізувати управління всіма основними ресурсами в одному місці.

Таким чином, для розгортання платформи буде достатньо двох серверів. Один сервер буде обслуговувати фронтенд-частину, яка розроблена на основі JavaScript-бібліотеки React. Цей сервер забезпечуватиме доступ користувачів до інтерфейсу. Другий сервер буде відповідати за бекенд-частину, яка розроблена на платформі Node.js з використанням фреймворку Express. Цей сервер оброблятиме всі запити від клієнтів, управлятиме базою даних та забезпечуватиме доступ до файлового сховища.

Взаємодія між клієнтською та серверною частинами виконується через HTTPS-запити та сокети. HTTPS-запити використовуються для стандартних операцій, таких як отримання даних з бази або відправлення даних на сервер. Використання HTTPS забезпечує захищене з'єднання, що гарантує безпеку переданих даних. Сокети використовуються для реалізації функцій, які вимагають реального часу, таких як миттєві повідомлення або оновлення даних у РРЧ. Завдяки цьому платформа може оперативно реагувати на зміни та забезпечувати більш інтерактивний досвід для користувачів.

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		87

Такий підхід дозволяє спростити структуру та забезпечити ефективну взаємодію між компонентами платформи, а також забезпечити гнучкість системи при додаванні нових функцій.

Для візуального відображення архітектури платформи було створено діаграму розгортання, яка представлена на рис. 3.1.

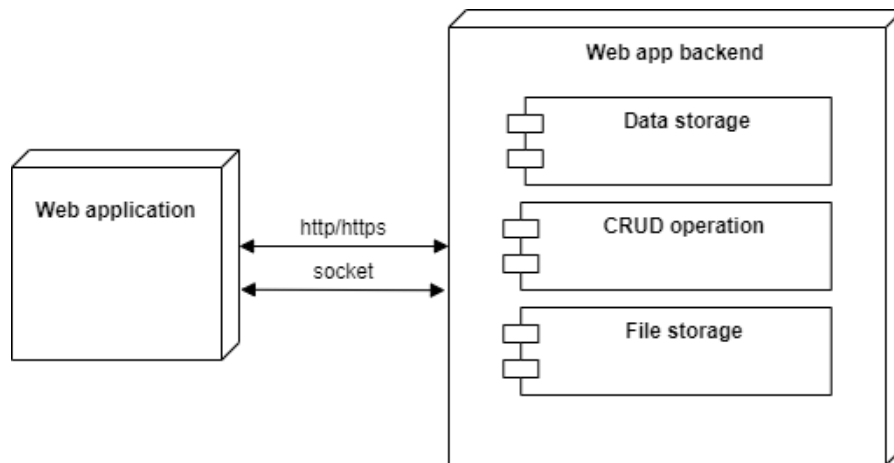


Рисунок 3.1 - Діаграма розгортання платформи

Налаштування параметрів платформи включає заповнення файлу .env для бекенд частини та файлу src/config.js для фронтенд-компонента.

До переліку змінних .env-файлу бекенду-частини належать:

- PORT – порт, на якому буде розгорнуто бекенд-частину проєкту;
- CLIENT_URL – посилання фронтенд-частини додатку, з якої будуть надсилатись API-запити користувача;
- DB_HOST – хост, за яким сервер буде підключатись до бази;
- DB_USER – користувач, від імені якого, сервер буде підключатись до бази;
- DB_PASSWORD – пароль користувача, від імені якого, сервер буде підключатись до бази;
- DB_DATABASE – ім'я бази даних, до якої сервер буде підключатись;
- DB_CHARSET – тип кодування у базі даних;
- SECRET_KEY – випадковий ключ, який використовується для хешування паролів;

- DEFAULT AJAX_COUNT_USERS_TO_CHATTING – кількість користувачів, яка довантажується при пошуку в списку чатів;
- DEFAULT AJAX_COUNT_CHAT_MESSAGES – кількість повідомлень, яка довантажується при пошуку в чаті;
- MAIL_SERVICE – назва поштового сервісу або провайдера, який використовується для надсилання електронної пошти;
- MAIL_EMAIL – електронна адреса, яка використовується як адреса відправника електронної пошти;
- MAIL_FROM – ім'я або псевдонім, який використовується як відправника електронної пошти;
- MAIL_PASSWORD – пароль для облікового запису електронної пошти;
- JWT_REMEMBER_ACCESS_LIFETIME – тривалість життя токена для користувача, який вибрав “remember me” при авторизації;
- JWT_DEFAULT_ACCESS_LIFETIME – тривалість життя токена для користувача, який не вибрав “remember me” при авторизації;
- STRIPE_PUBLIC_KEY – публічний ключ Stripe, використовується для ідентифікації облікового запису на Stripe під час обробки платежів;
- STRIPE_PRIVATE_KEY – приватний ключ Stripe, використовується для автентифікації сервера на Stripe під час створення платежів;
- PAYPAL_CLIENT_ID – ідентифікатор клієнта PayPal, використовується для ідентифікації вашого клієнта або додатку під час взаємодії з API PayPal;
- PAYPAL_SECRET_KEY – секретний ключ PayPal, використовується для забезпечення безпеки та автентифікації на серверній стороні під час взаємодії з API PayPal.

До переліку змінних конфігураційного файлу фронтенду належать:

- API_URL – URL адреса, за якою здійснюється доступ до API;
- CLIENT_URL – це URL адреса, фронтенд-частини платформи;

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		89

- BLOB_CHUNK_SIZE – розмір частини BLOB-файлу, який може передаватись сокетом на сервер;
- UNBLOB_CHUNK_SIZE – розмір частини не BLOB, який може передаватись сокетом на сервер;
- JOB_STATUSES – об’єкт, який зберігає дані про можливі статуси прогресу задачі;
- COMMENT_TYPES – об’єкт, який зберігає дані про можливі типи коментарів у додатку;
- CHAT_ROLES – об’єкт, який зберігає дані про можливі типи ролей у чаті додатка;
- CHAT_OWNER_ROLES_SELECT – об’єкт, який зберігає дані про можливі типи ролей, які можуть задавати власники чатів іншим користувачам;
- CHAT_ADMIN_ROLES_SELECT – об’єкт, який зберігає дані про можливі типи ролей, які можуть задавати адміни чатів іншим користувачам;
- MAP_KEY – ключ, який відповідає за роботу з API гугл-карт;
- STRIPE_PUBLIC_KEY – публічний ключ Stripe, використовується для ідентифікації облікового запису на Stripe під час обробки платежів;
- PAYPAL_CLIENT_ID – ідентифікатор клієнта PayPal, використовується для ідентифікації вашого клієнта або додатку під час взаємодії з API PayPal.

Таким чином, було розглянуто розгортання та налаштування компонентів розробленої платформи.

3.2 Структура інтерфейсу платформи

Тепер розглянемо інтерфейс і функціональні можливості платформи.

Його було реалізовано з урахуванням стандартних вимог UX та загальних норм розробки та вигляду сайтів [30, 31, 32, 33]. На знімках буде показано, як користувачі можуть створювати завдання, брати їх до роботи,

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		90

контролювати прогрес, комунікувати та інше. Детальні описи до кожного знімка підкреслять основні особливості інтерфейсу.

Для початку розглянемо основу будь-якої платформи, а саме форму для авторизації. Її продемонстровано у вигляді рис. 3.2.

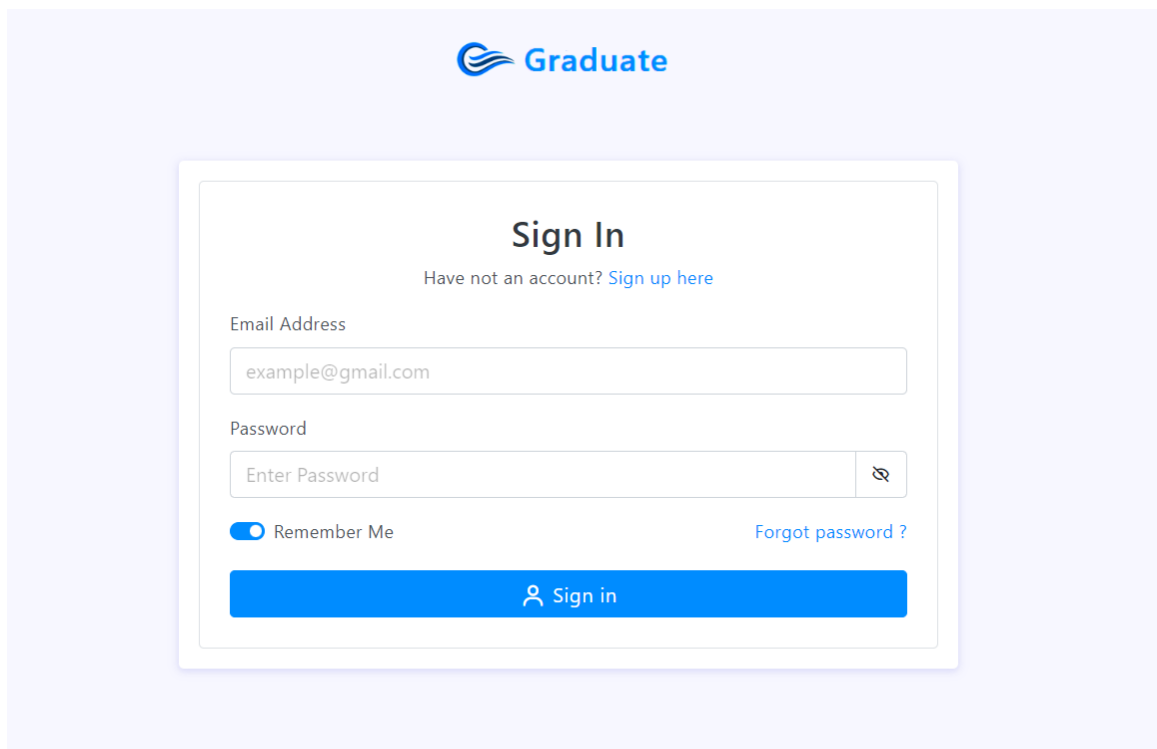


Рисунок 3.2 - Форма авторизації

Дана сторінка є реалізацією звичайної форми авторизації. Користувач вводить свою пошту та пароль, і якщо дані збігаються, то користувач отримує доступ до платформи. Тривалість авторизації залежить від того, чи активовано опцію запам'ятання, чи ні. Також форма має можливості перемикавання на форму для реєстрації, призначену для нових користувачів, та форму скидання паролю, призначену для давніх користувачів, які не можуть згадати свій пароль.

Після авторизації користувач бачить задачі відповідно до його збереженого розташування в налаштуваннях профілю. Демонстрацію наведено у вигляді рис. 3.3.

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		91

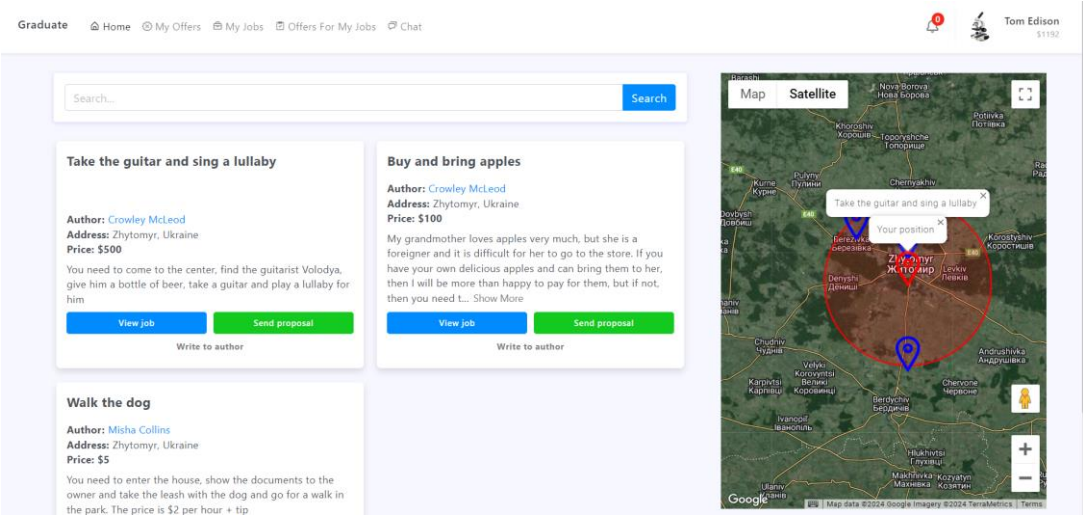


Рисунок 3.3 - Основна сторінка користувача

На сторінці користувач бачить задачі, які він може виконати. Вони відображаються у порядку найближчих до користувача. Червоним кругом позначено ділянку як найпріоритетнішу для користувача. Для перевірки локації задачі достатньо натиснути на позначку і побачити її назву. Потім можна звіритись з картками та переглянути умови. Після того, як користувач розуміє, яка задача йому підходить, він може написати власнику щодо додаткових умов чи питань у внутрішньому чаті системи. Демонстрацію чату наведено у вигляді рис. 3.4.

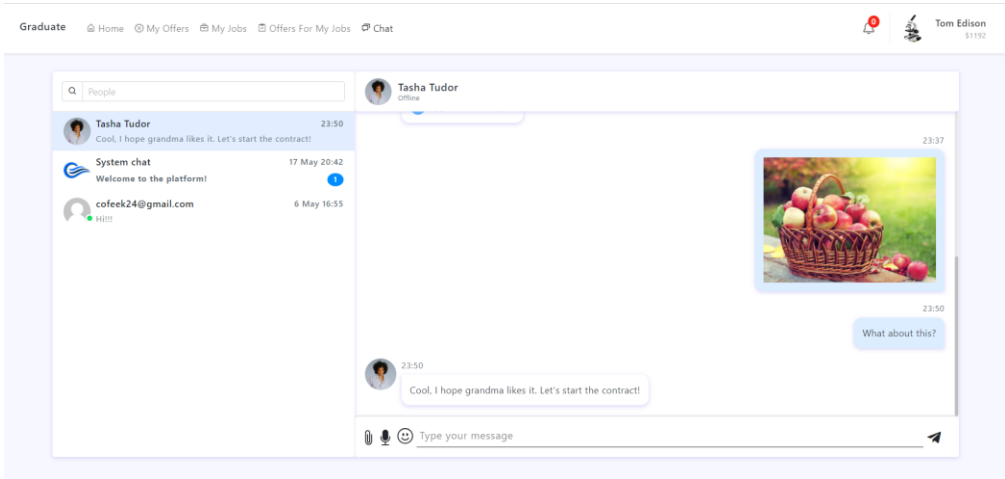


Рисунок 3.4 - Внутрішній чат платформи

Як можна побачити, чат має дві секції: список з чатами користувача та список повідомлень чату, які переглядаються наразі. У користувача є чотири варіанти надсилання повідомлень – текстом, файлом, або записавши відео чи

аудіоповідомлення. Приклади використання зображені на рис. 3.5 – 3.8 відповідно.

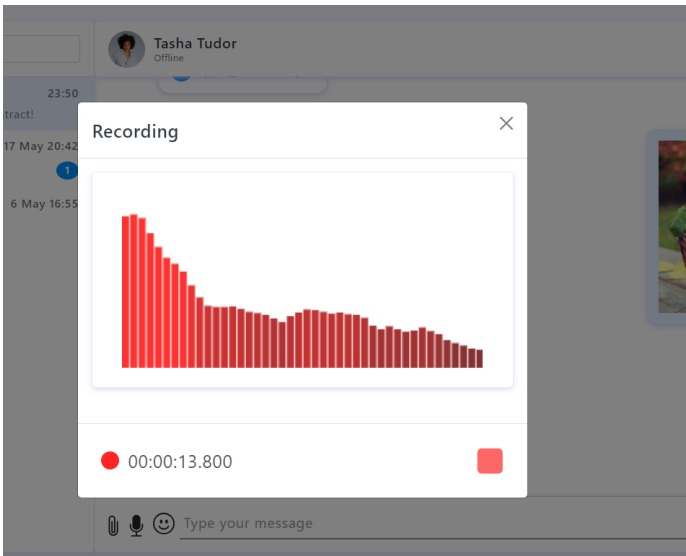


Рисунок 3.5 - Запис аудіо-повідомлення

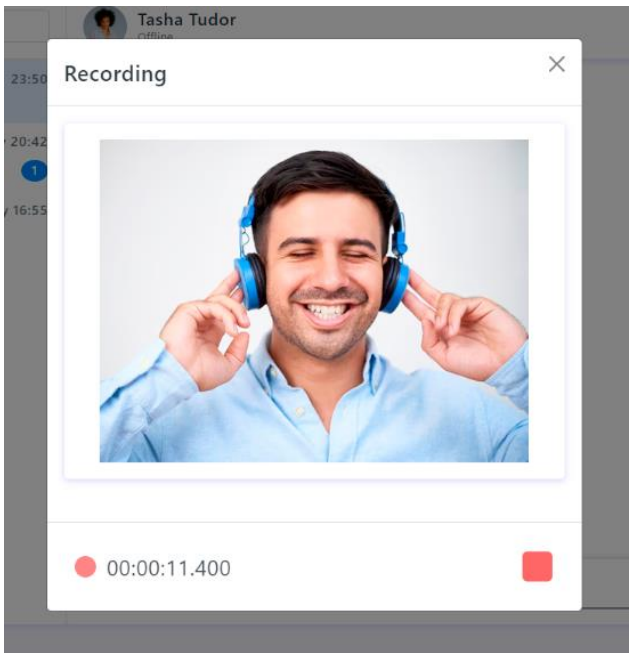


Рисунок 3.6 - Запис відео-повідомлення

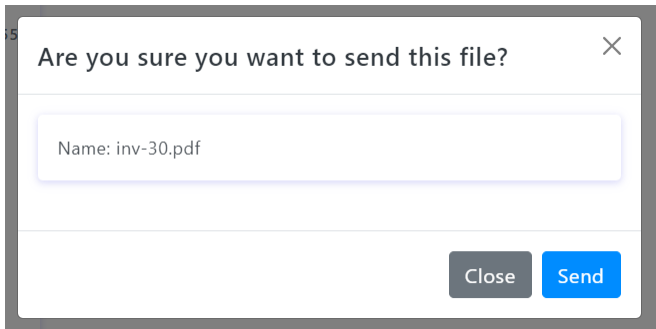


Рисунок 3.7 - Запит на підтвердження відправки файлу

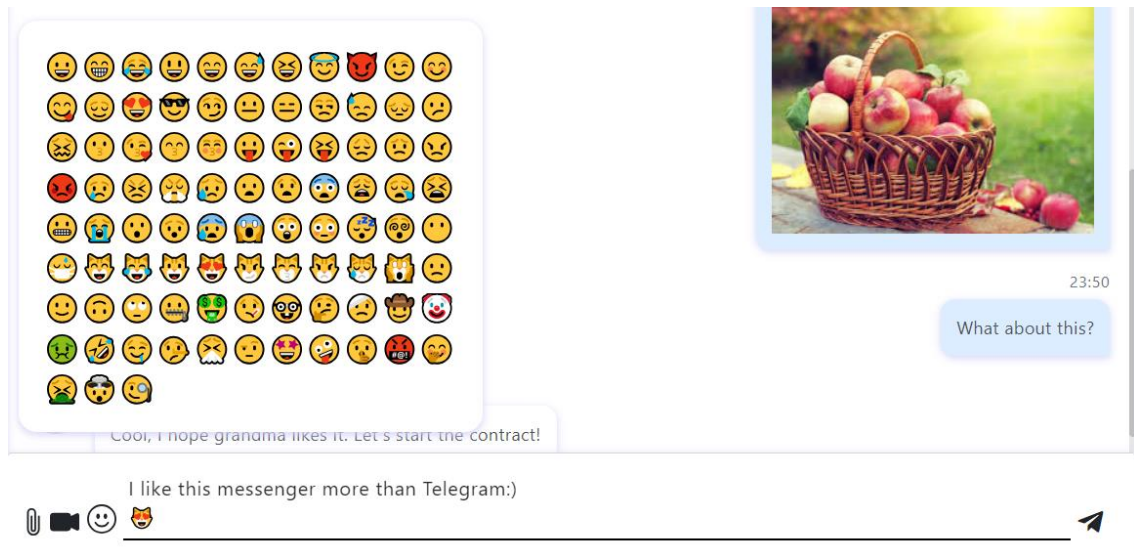


Рисунок 3.8 - Друк текстового повідомлення з емоціями

Також платформа надає зручний інтерфейс для перегляду та керування власними задачами, а також задачами, які виконує поточний користувач. Для цього на сторінці користувача наявні три вкладки на навігаційній панелі. Вкладка “My Jobs” відповідає за керування задачами, власник може редагувати їх, позначати як неактивні чи активні у випадку, якщо задача більше не потребує виконання, та переглядати приклад відображення задачі виконавцю. Сторінку продемонстровано у вигляді рис. 3.9. “Offers For My Jobs” – сторінка, де користувач може побачити пропозиції на виконання власних робіт, переглядати їхні статуси та просто контролювати. Демонстрацію наведено у вигляді рис. 3.10. “My Offers” – це сторінка для керування пропозиціями та угодами, які виконує користувач. Сторінку продемонстровано у вигляді рис. 3.11.

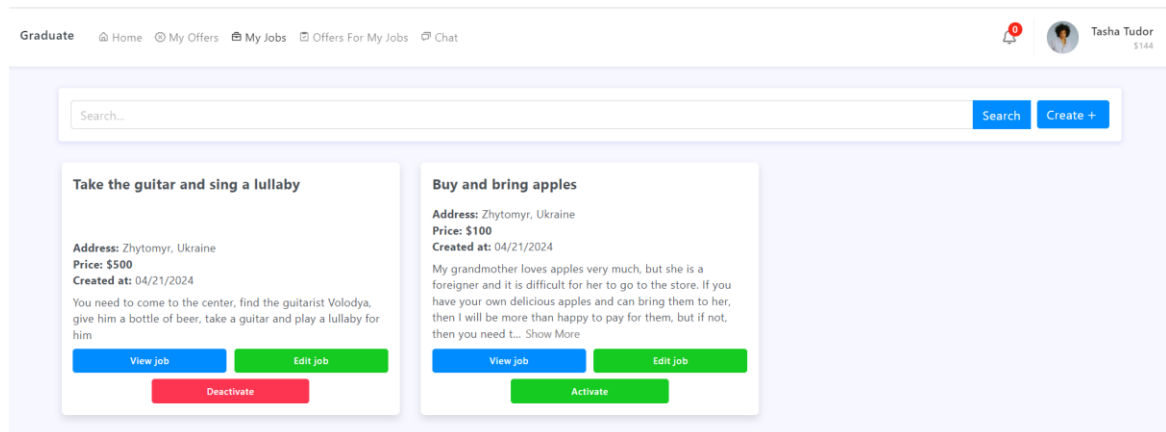


Рисунок 3.9 - Перелік задач, які створив користувач

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				94
Змн.	Арк.	№ докум.	Підпис	Дата		

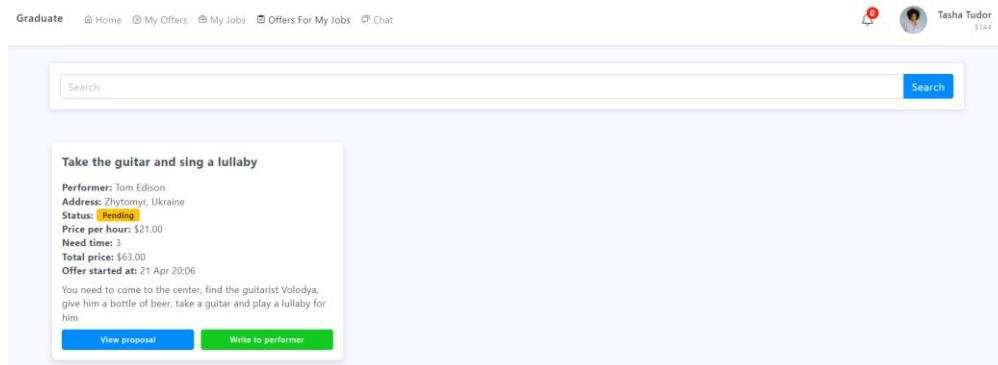


Рисунок 3.10 - Сторінка з угодами користувача, які виконуються для нього

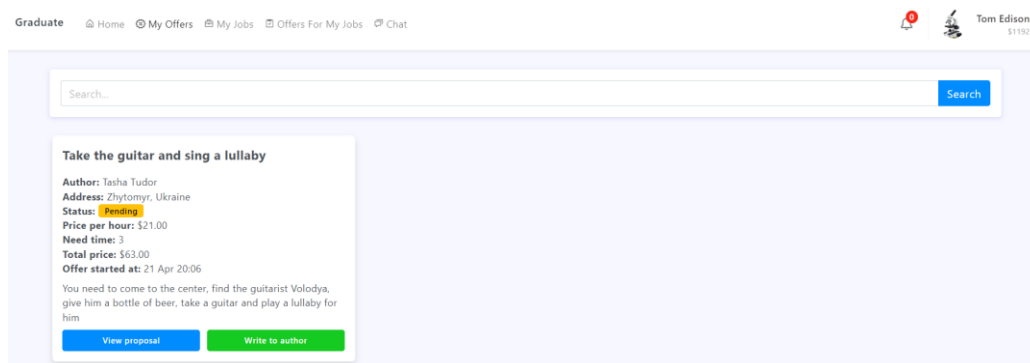


Рисунок 3.11 - Сторінка з угодами користувача, які виконуються ним

Користувачі можуть створювати запити на виконання завдань. Для цього їм потрібно просто заповнити форму на основній сторінці з переліком задач, або ж на сторінці детального перегляду інформації про задачу. Форма дозволяє виконавцям торгуватись з замовниками, вона має поле, де виконавець вказує ціну, яку він хоче мати за годину роботи, та тривалість виконання задачі. Форму продемонстровано у вигляді рис. 3.12.

×

Send proposal

Price per hour, \$

\$

10

↑

↓

Working needed time, h

📅

2

Total, \$

\$

20

Send

Рисунок 3.12 - Форма відправки запиту на виконання задачі

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		95

Замовники можуть відмовитись від послуг, якщо їм не подобається ціна, терміни не влаштовують, або якщо замовник шукає виключно високоякісного виконавця. Також вони можуть погодитись, якщо запропоновані умови підходять, і в чаті між користувачами не виникло питань. Візуальне представлення панелі керування задачею для замовника представлено у вигляді рис. 3.13.

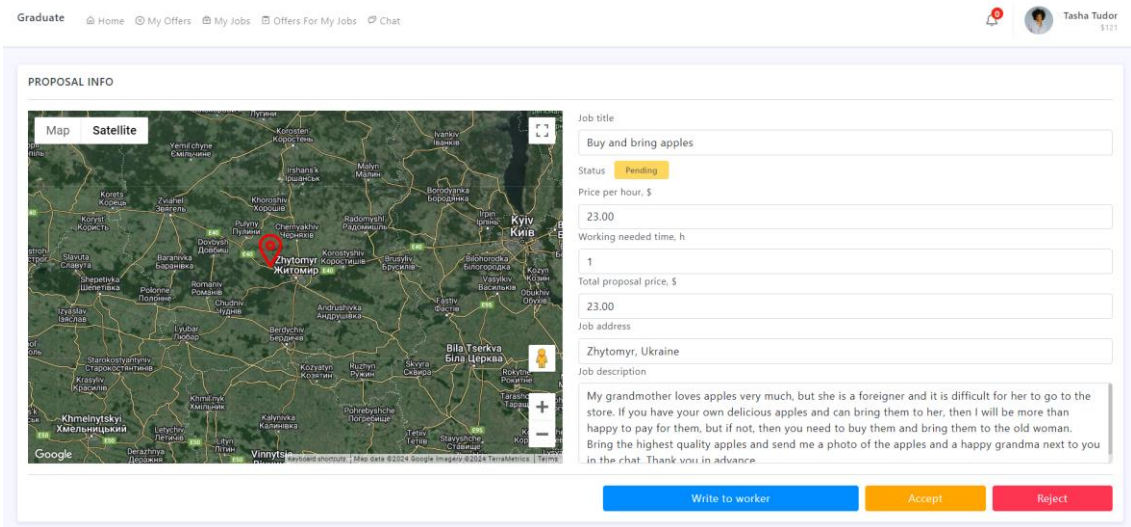


Рисунок 3.13 - Перегляд запиту виконання завдання

Після початку виконання замовлення виконавець може відправити запит на завершення роботи, написати замовнику для уточнення питань, або створити суперечку, якщо виникла якась необговорена ситуація, і користувачі не можуть вирішити її власними силами. Після цього користувач може написати замовнику або почати суперечку. Приклад відправки запиту на підтвердження завершення задачі зображено на рис. 3.14.

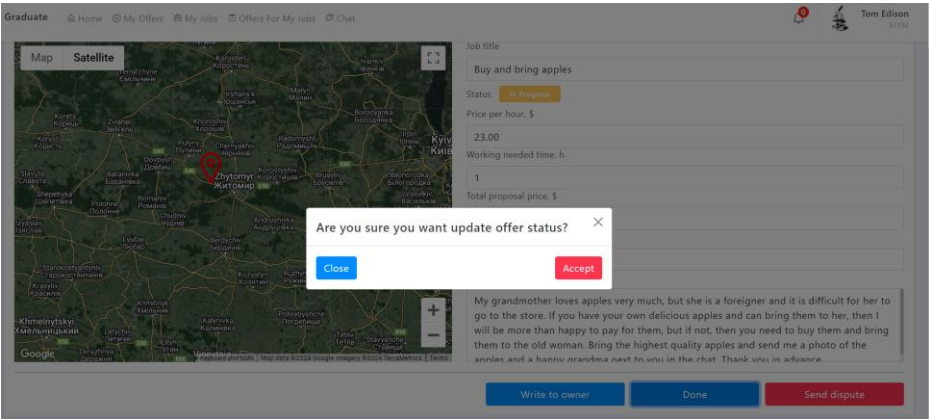


Рисунок 3.14 - Відправка виконавцем запиту на підтвердження виконання

Замовник, отримавши нотифікацію про запит на розв’язання задачі, може спочатку з’ясувати, чи задача була завершена, чи все відбулося так, як того хотів замовник. Після цього він може підтвердити завершення. Якщо якийсь з етапів залишився незавершеним, користувач може скасувати виконання задачі, після чого виконавець може або підтвердити це, і задача закриється, а власник отримає кошти назад, або створити суперечку з участю адміністратора. Приклад зображено на рис. 3.15.

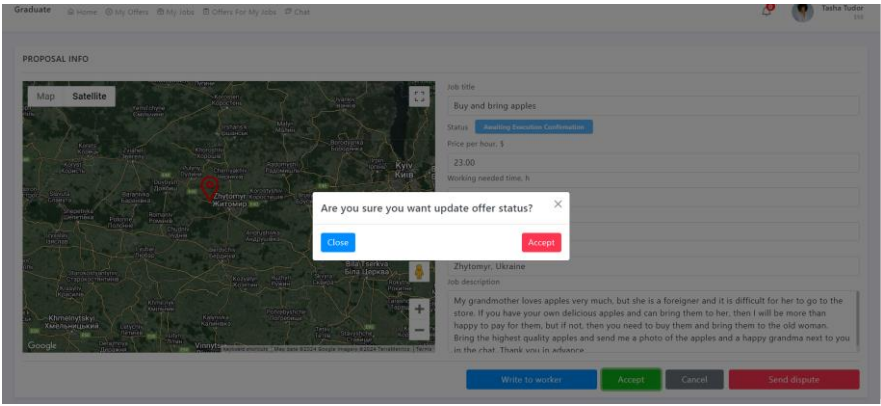


Рисунок 3.15 - Підтвердження виконання задачі

На більшість подій система пропонує оповіщення. Наприклад, на авторизацію з якогось пристрою, на надходження нового повідомлення в чаті, на зміну статусу задачі. Приклад нотифікації, яка повідомляє про успішне завершення задачі, зображено на рис. 3.16.

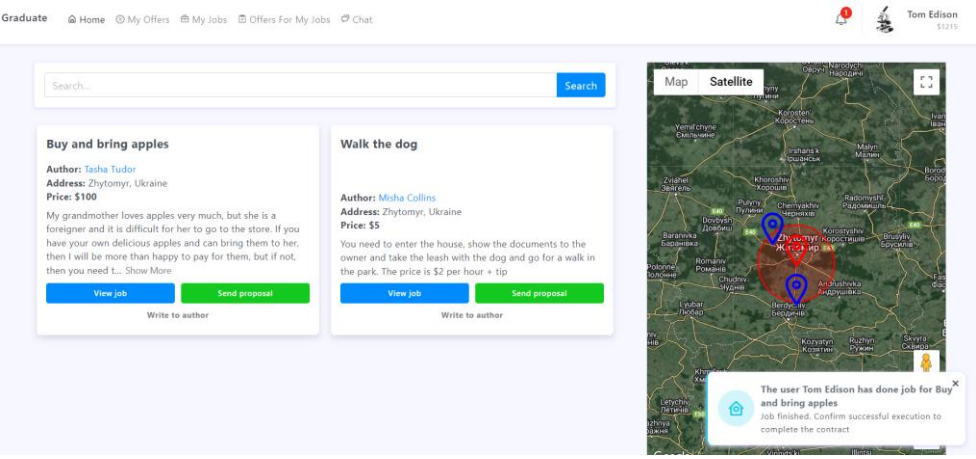


Рисунок 3.16 - Нотифікація про успішне завершення задачі

Ще однією перевагою платформи є взаємодія з реальними коштами. Таким чином, було розроблено сторінку для поповнення внутрішнього

балансу коштами з інших платформ, та для виведення їх з платформи. Вона складається з трьох вкладок. Вкладка з базовою інформацією про поточний баланс продемонстрована на рис. 3.17, вкладка для поповнення балансу зображена на рис. 3.18, та вкладка для виведення коштів відображена на рис. 3.19.

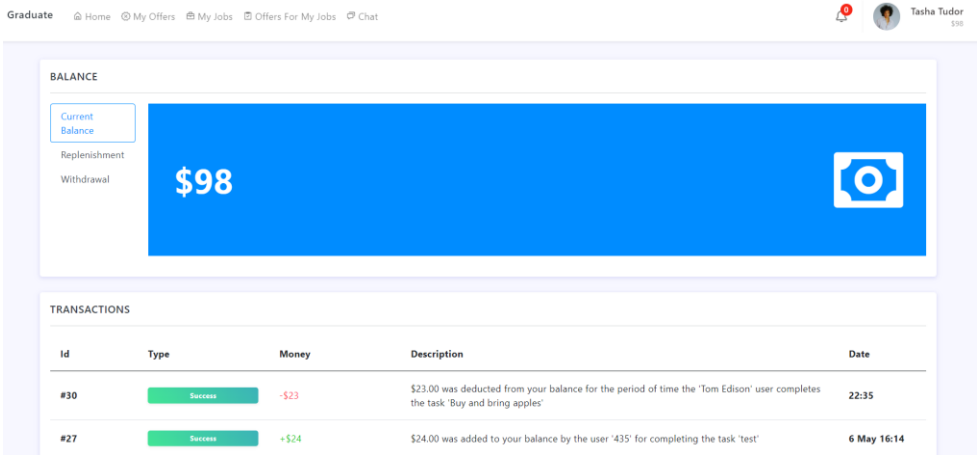


Рисунок 3.17 - Сторінка поповнення балансу користувача

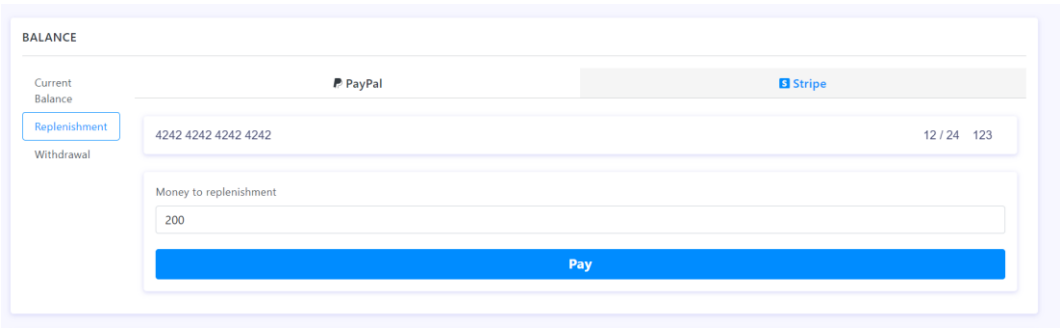


Рисунок 3.18 - Приклад поповнення балансу через страйп

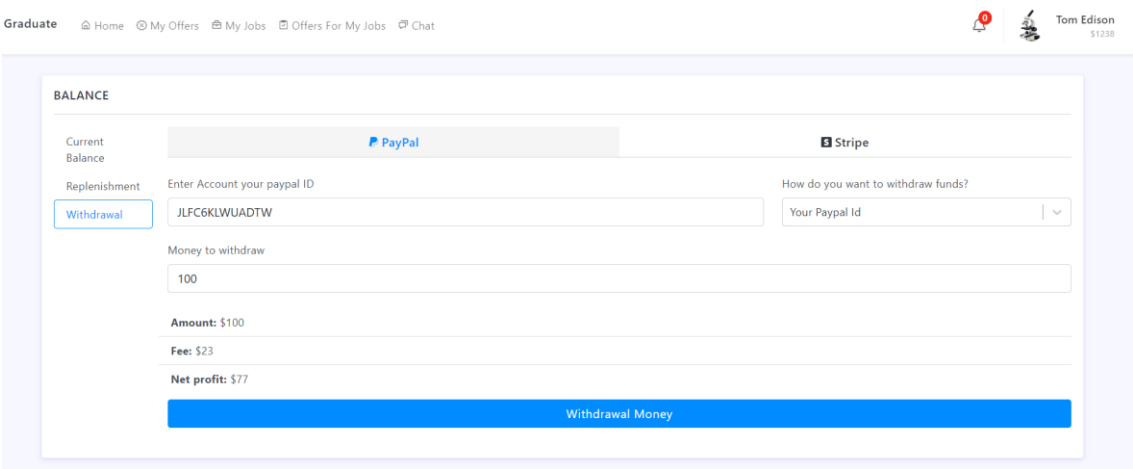


Рисунок 3.19 - Приклад виведення коштів з балансу на акаунт пейпалу

Як було згадано раніше, адміністратор має вирішувати суперечки, які не можуть вирішити користувачі самі. Таким чином, для нього було розроблено інтерфейс для цієї операції. Адміністратор може переглядати суперечки та закріплювати за собою ті, які він готовий вирішити. Таким чином, було розроблено таблицю для пошуку суперечок, які може вирішити адміністратор, представлену на рис. 3.20.

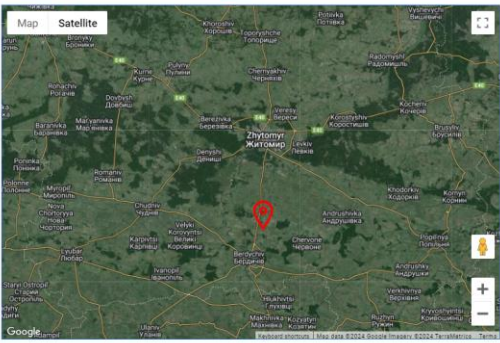
DISPUTES								
				May 9, 2024 to May 24, 2024		Search...		
Id	Job	Admin	Sender	Price per hour	Need Hours	Status	createdAt	Actions
#8	Find the car and drive it to...	ipz201_voyu@st...	edison412@gma...	12	2	Resolved	6 May 21:23	🔍
#7	Walk the dog	ipz201_voyu@st...	test@gmail.com	11	0	In Progress	16 Oct 2023 20:25	🔍
<div>← 1 →</div>								

Рисунок 3.20 - Список суперечок у адміністратора

Для вирішення суперечок адміністратор повинен мати можливість переглядати інформацію про задачу, пропозицію, суперечку та про користувачів. Основну частину інформації про суперечку, де відображені умови задачі та опис проблеми, продемонстровано у вигляді рис. 3.21.

PROPOSAL INFO

Map Satellite



Google

Dispute description

sgdgtf fg ghfag

Job title

Walk the dog

Status

In Progress

Dispute Status

In Progress

Price per hour, \$

11.00

Working needed time, h

1

Total proposal price, \$

11.00

Job address

Zhytomyr, Ukraine

Job description

You need to enter the house, show the documents to the owner and take the leash with the dog and go for a walk in the park. The price is \$2 per hour + tip

Рисунок 3.21 - Перегляд інформації адміністратором про суперечку

Крім інформації, на сторінці присутні кнопки для вирішення суперечки, для перегляду чату користувачів та надсилання повідомлення від імені системи кожному з учасників. Кожна з цих можливостей допомагає адміністратору розібратись з проблемою та зробити правильний вибір. Крім

того, адміністратор може відмовитись від суперечки, щоб хтось інший, досвідченіший, перехопив її та вирішив. Приклад перегляду фрагмента інформації про суперечку, яка відповідає за інформацію про користувачів та дії з суперечкою, наведено у вигляді рис. 3.22.

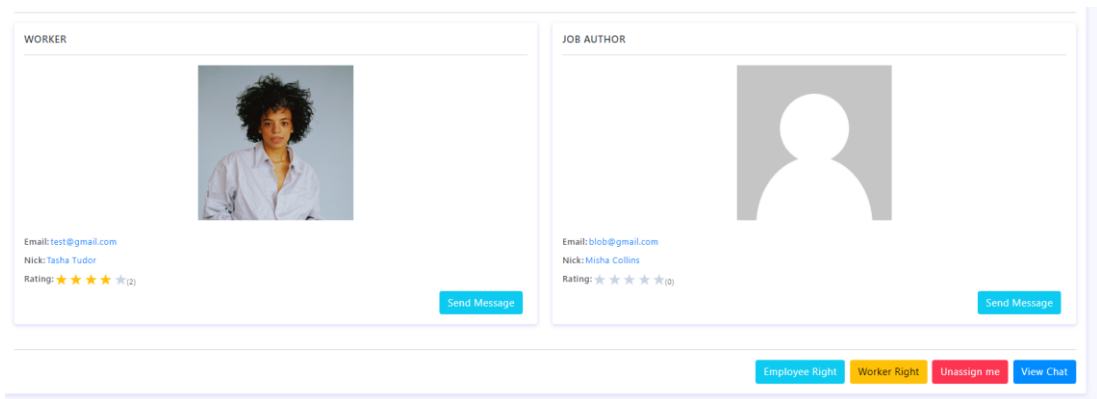


Рисунок 3.22 - Перегляд інформації про учасників суперечки

Платформа надає можливість адміністраторам переглядати чати користувачів між якими виникла суперечка. Це дозволяє поглибитись у суть суперечки та прочитати про додаткові умови, які не були обговорені в самому описі задачі. Також адміністратор бачить будь-які зміни чату. Наприклад червоним кольором підсвічено повідомлення, які були видалені, а зеленим – редаговані. Приклад чату, який переглядає адміністратор, наведено у вигляді рис. 3.23.

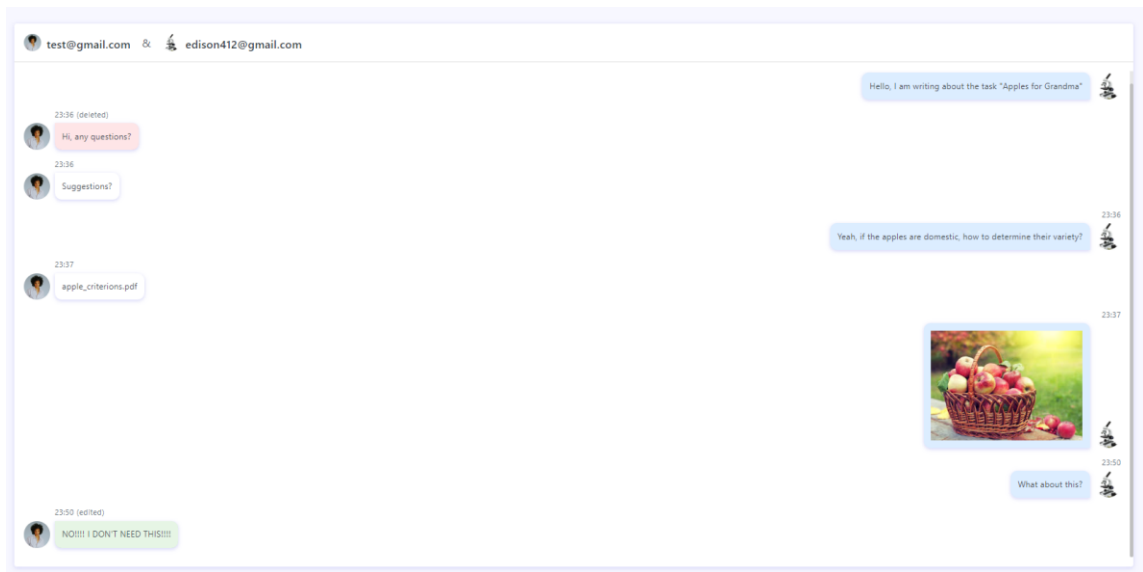


Рисунок 3.23 - Перегляд чату користувачів адміністратором

Для зручності перегляду історії змін повідомлень було створено вікно, яке активується при натисканні на слово “edited” біля повідомлення зеленого кольору. Вікно продемонстровано у вигляді рис. 3.24.

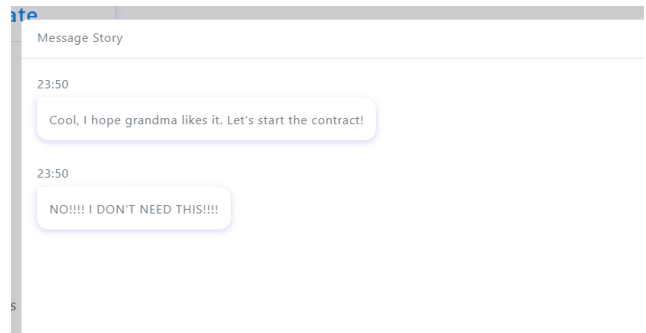


Рисунок 3.24 - Перегляд історії редагування повідомлення адміністратором

Крім того, адміністратор має можливість переглядати статистику роботи платформи. Панель адміністратора включає статистику контролю фінансів. На діаграмі відображається сума, яка надійшла за певний період до платіжної системи акаунту та була виведена. Також система включає статистику відвідуваності користувачами платформи, статистику збільшення зареєстрованих користувачів у системі, статистику по задачах, наприклад, скільки було виконано і скільки було створено у цей же період, та статистику суперечок – скільки було створено і вирішено за певний проміжок часу. Візуалізацію статистики відвідування платформи наведено у вигляді рис. 3.25.



Рисунок 3.25 - Перегляд статистики адміністратором

Також додаток пропонує адаптивний дизайн, який виглядає досить гарним та зручним на телефонах. Прикладом є перегляд основних сторінок з телефону iPhone SE, зображеному на рис. 3.26. Як можна побачити, платформа у такому розширенні екрана виглядає досить зручною і не втрачає основного функціоналу.

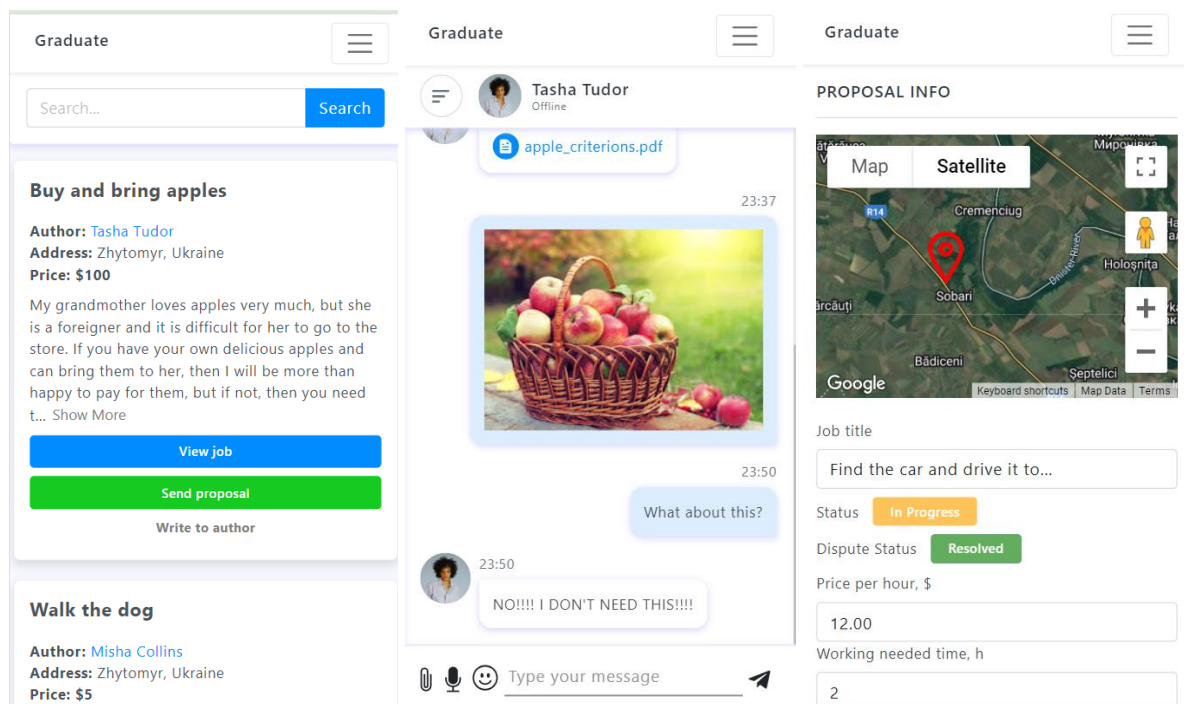


Рисунок 3.26 - Адаптивність основних сторінок платформи

3.3 Тестування платформи

Тестування є одним з ключових аспектів розробки програмного забезпечення, який гарантує якість, надійність та безпеку додатка. Воно допомагає виявити та виправити помилки на ранніх етапах, що значно знижує витрати на виправлення дефектів у майбутньому. Завдяки тестуванню можна забезпечити відповідність додатка вимогам користувачів і бізнесу, що підвищує його конкурентоспроможність і задоволеність клієнтів. Крім того, тестування сприяє підтримці високих стандартів продуктивності та сумісності з різними платформами та пристроями. Тестування платформи відбувалося у двох форматах, що дозволило охопити всі аспекти функціональності та забезпечити високу якість кінцевого продукту.

Першим є ручне регресійне тестування. За допомогою нього відбувалася перевірка на усіх етапах розробки. Після кожного глобального оновлення коду відбувалася перевірка не лише модулів, яких гарантовано зачіпали зміни, а й усіх інших. Це дозволило уникнути помилок різної складності на початкових етапах їхнього утворення.

Другим форматом є юніт-тести. Ними було покрито усі моделі бекендової частини платформи. Вони дозволяли визначати, чи не впливають поточні зміни бази даних на виконання коду.

Однією з переваг юніт-тестів була можливість перевіряти, як відпрацьовують методи різних моделей з реальною базою. Це дозволило перевіряти коректність міграцій перед використанням їх на реальних серверах. Також завдяки такому підходу тестування можна було визначати критичні методи і якимось їх оптимізувати. Приклад виконаних тестів з попередженням про надмірний час виконання зображено у вигляді знімку консолі на рис. 3.27.

```
User Model
create
  ✓ should insert a new user and return the insertId (114ms)
  ✓ should throw an error if email is already registered (109ms)
findByEmail
  ✓ should return user by email
  ✓ should return undefined if user is not found by email
findByPasswordAndEmail
  ✓ should return user if email and password are correct (116ms)
  ✓ should throw an error if email is incorrect
  ✓ should throw an error if password is incorrect (128ms)
getUserInfo
  ✓ should return user info by userId
  ✓ should return undefined if user is not found by userId
getFullUserInfo
  ✓ should return full user info by userId
  ✓ should return undefined if user is not found by userId
checkIsAdmin
  ✓ should return false if user is not an admin
  ✓ should return false if user is not found
updateUserProfile
  ✓ should update user profile data
createFull
  ✓ should insert a new user with full profile data
changeAuthorized
  ✓ should toggle user authorization status
changeRole
  ✓ should toggle user role between admin and non-admin
delete
  ✓ should delete user
updatePassword
  ✓ should update user password (107ms)

19 passing (826ms)
```

Рисунок 3.27 - Приклад виконаних тестів

Як було сказано раніше, юніт-тестами було покрито лише моделі платформи, бо саме в моделях відбувається основна логіка обробки даних. Моделі є центральним компонентом архітектури додатка, оскільки вони відповідають за взаємодію з базою даних та маніпулювання даними. Таким чином, тестами було покрито найважливіші фрагменти коду, що забезпечило перевірку коректності виконання ключових функцій та методів. Це дало можливість виявити потенційні помилки в логіці обробки даних та виправити їх до впровадження у виробництво.

У результаті, тестування допомогло забезпечити стабільність та надійність роботи платформи. Це дозволило охопити всі аспекти функціональності та забезпечити високу якість кінцевого продукту. Юніт-тести забезпечували швидке виявлення помилок під час кожного циклу розробки, тоді як ручне тестування допомогло перевірити додаток у різних сценаріях використання. Такий підхід до тестування забезпечив можливість всебічної перевірки та гарантував високу якість кінцевого продукту.

Висновки до третього розділу

Було розглянуто важливі аспекти функціонування системи та її інтерфейсу. Під час аналізу параметрів системи було досліджено спектр налаштувань, які розробник платформи може швидко змінювати та залежно від цього впливати на різні системно важливі процеси, такі як фінансові операції, взаємодія бекенду та фронтенду.

Також було розглянуто ще один важливий етап розробки, а саме – структура інтерфейсу. Основна увага була зосереджена на спрощенні користувацького досвіду. Було розглянуто різні аспекти інтерфейсу, зокрема, розташування елементів, їхній дизайн та інтерактивність. Головною метою було створення зручного, адаптивного та інтуїтивно зрозумілого інтерфейсу, який би сприяв ефективній роботі користувачів.

Крім того, було розглянуто такий важливий етап розробки, як тестування, а саме види тестування, які було використано до проєкту. Результати тестування були детально проаналізовані, а виявлені проблеми та помилки були виправлені. Також було приділено увагу на особливості кожного виду тестування і їхній вплив на розробку програмного продукту. Цей етап дозволив переконатися в якості та надійності нашої системи перед завершенням її створення.

ВИСНОВКИ

Розробка платформи передбачала глибокий аналіз потреб користувачів та характеристик ринку, тому було проведено дослідження різних платформ, які надають подібні послуги, щоб визначити основні вимоги до платформи. На основі проведеного огляду й аналізу була обрана клієнт-серверна архітектура платформи і вибрані технології для її реалізації, а саме: Node.js для серверної частини, React – для клієнтської, а MySQL – для збереження даних.

Далі було докладно розглянуто різні способи використання платформи, а також її можливості та способи взаємодії користувачів. Було описано основний функціонал, зосереджуючись на процесі створення та розміщення завдань для виконання. Крім того, акцент був зроблений на можливості обговорення умов та вирішення можливих конфліктів між користувачами. Під час розгляду цих аспектів було докладно проаналізовано кожен можливість та перевагу, яку платформа може запропонувати.

Наступним етапом стала розробка структури платформи та забезпечення взаємодії між її компонентами. Цей етап дозволив покроково вдосконалювати платформу та оптимізувати її процеси. Також була розроблена база даних для зберігання інформації про користувачів, завдання, транзакції та інші важливі дані, з урахуванням потреб платформи та забезпеченням доступності даних для всіх компонентів системи. База даних складається з двадцяти однієї таблиці.

Крім цього, належну увагу було приділено процесам авторизації, а також взаємодії користувачів з чатом та іншими ключовими модулями платформи. Після завершення етапу було зроблено висновок, що розробка забезпечує максимальний рівень зручності та функціональності для користувачів платформи.

В подальшому було ретельно розглянуто схему розгортання платформи та описано взаємодію між її компонентами. Також на цьому етапі було

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		106

представлено детальний опис інтерфейсу програмного продукту для різних категорій користувачів, таких як адміністратори та звичайні користувачі. Це дозволило забезпечити максимальний рівень зручності та доступності для всіх користувачів платформи. На завершення було проведено фінальний етап – тестування. Під час цього етапу було докладно проаналізовано різні види тестування та їхні результати.

Робота над кожним етапом розробки платформи керувалася метою створення надійного та ефективного інструменту для її користувачів. Після завершення проєкту можна зробити висновок, що пропонуваний програмний продукт відповідає вимогам сучасного ринку і готовий до впровадження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Is Upwork Worth It? Pros, Cons, and Top Tips from Freelancers [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://www.upwork.com/resources/is-upwork-worth-it>.
2. The Pros and Cons of Working on Fiverr [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://forcreators.com/working-on-fiverr/>.
3. 15 Pros and Cons of Working for Taskrabbit [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://www.becomeopedia.com/pros-and-cons-of-working-for-taskrabit/>.
4. Взаємодія клієнт-сервер [Електронний ресурс]. – 2024.– Режим доступу до ресурсу: <https://ua5.org/technol/2094-vzayemodiya-kliiyent-server.html>.
5. Монолітна архітектура ПЗ [Електронний ресурс]. – Режим доступу до ресурсу: <https://qalight.ua/baza-znaniy/shho-take-monolitna-arhitektura/>.
6. Мікросервісна архітектура ПЗ. [Електронний ресурс]. – Режим доступу до ресурсу: <https://qalight.ua/baza-znaniy/shho-take-mikroservisna-arhitektura-pz/>.
7. Про архітектуру додатків [Електронний ресурс]. – 2024.– Режим доступу до ресурсу: <https://foxminded.ua/arkhitektura-zastosunku/>.
8. MVC проти MVVM – різниця між ними [Електронний ресурс] . – 2024. – Режим доступу до ресурсу: <https://www.guru99.com/uk/mvc-vs-mvvm.html>.
9. Використання розширення Visual Studio Code [Електронний ресурс] . – 2024. – Режим доступу до ресурсу: <https://learn.microsoft.com/uk-ua/power-pages/configure/vs-code-extension>.
10. Документація Node.js [Електронний ресурс]. – Режим доступу до ресурсу: <https://nodejs.org/en/docs/>.
11. Документація express.js [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: <https://expressjs.com/en/5x/api.html>.

12. Початок використання PayPal REST APIs [Електронний ресурс]. – Режим доступу до ресурсу: <https://developer.paypal.com/api/rest/>.
13. Stripe Docs [Електронний ресурс]. – Режим доступу до ресурсу: <https://docs.stripe.com/get-started/development-environment?lang=node>.
14. Як використовувати Socket.io у Node.js [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://socket.io/get-started/chat/>.
15. Навіщо розуміти backend-частину та чому варто обрати Node.js [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://robotdreams.cc/uk/blog/241-zachem-ponimat-backend-chast-i-pochemu-stoit-vybrat-node-js>.
16. Створення та структурування програми Node.js MVC [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://blog.logrocket.com/building-structuring-node-js-mvc-application/>.
17. Покроковий підручник React.js [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://reactjs.org/docs/getting-started.html>.
18. Як використовувати Socket.io у React [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://socket.io/how-to/use-with-react>.
19. React PayPalButtons [Електронний ресурс]. – Режим доступу до ресурсу: <https://paypal.github.io/react-paypal-js/?path=/docs/example-paypalbuttons--default>.
20. React Stripe [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.stripe.com/stripe-js/react>.
21. Документація Bootstrap 5 українською [Електронний ресурс]. – Режим доступу до ресурсу: <https://bootstrap21.org/uk/docs/5.0/getting-started/introduction/>.
22. Документація використання піктограми Bootstrap 5 українською [Електронний ресурс]. – Режим доступу до ресурсу: <https://bootstrap21.org/uk/docs/5.0/extend/icons/>.
23. SQL в Access: основні поняття, глосарій і синтаксис [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: <http://surl.li/tydde>.

24. Пишемо вражаюче швидкі MySQL запити [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: <http://surl.li/tyddp>.

25. Генерація SQL-запиту засобами MySQL-сервера [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: <https://dou.ua/lenta/articles/sql-query-mysql-server/>.

26. SQL Підзапити [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: <https://support.microsoft.com/en-us/topic/run-a-sql-query-d41fc0b1-1c88-40d4-bbd1-951de6e94e2a>.

27. Патерн “Фасад” [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://refactoring.guru/uk/design-patterns/facade>.

28. Патерн "Компонувальник" [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://refactoring.guru/uk/design-patterns/composite>.

29. Патерн "Міст" [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://refactoring.guru/uk/design-patterns/bridge>.

30. Верстка сайтів [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://webtune.com.ua/statti/web-rozrobka/verстка-sajtiv/>.

31. Види верстки сайтів [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://impulse-design.com.ua/ua/chtotakoe-verстка-sajta.html>.

32. Що таке адаптивний дизайн та для чого його застосовують [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://onlinemedia.company/blog/sho-take-adaptivnij-dizajn/>.

33. 15 Правил UI/UX Дизайна [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: <https://topuser.pro/ux-ui-dizajn-osnovi-pravila-web/>.

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		110

ДОДАТКИ

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				111
Змн.	Арк.	№ докум.	Підпис	Дата		

Діаграма класів на пряму пов'язаних з чатом



Рисунок А.1 - Діаграма класів на пряму пов'язаних з чатом

Код файлу index.js

```

require("dotenv").config();

const express = require("express");
const cors = require("cors");
const bodyParser = require("body-parser");
const mysql = require("mysql");
const mainRoutes = require("./routes/main");
const socketIo = require("socket.io");
const { Chat: ChatSocketController } = require("./sockets");

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

const app = express();
app.use(bodyParser.json());
app.use(express.static("uploads"));

app.use(
  cors({
    credentials: true,
    exposedHeaders: "Authorization",
    origin: "*",
    origin: [process.env.CLIENT_URL, "https://www.sandbox.paypal.com"],
  })
);

const db = mysql.createConnection({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_DATABASE,
  charset: process.env.DB_CHARSET,
});

db.connect((err) => {
  if (err) {
    console.log("this db error: ");
    console.error(err);
  } else {
    console.log("Connected to MySQL database");
  }
});

const server = app.listen(PORT, () => {
  console.log("Server started on port " + PORT);
});

const io = socketIo(server, {

```

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ІЗ	Арк.
		Локтікова Т.М.				113
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    cors: {
      credentials: true,
    },
  });

  mainRoutes(app, db, io);

  new ChatSocketController(io, db);

```

Додаток В

Код методу findByPasswordAndEmail

```

findByPasswordAndEmail = async (email, password) => {
  await this.errorWrapper(async () => {
    const authError = () => this.setError("Invalid email or password",
401);

    const findUserRes = await this.dbQueryAsync(
      `SELECT * FROM users WHERE email = ?`,
      [email]
    );

    if (!findUserRes.length) {
      authError();
    }

    const user = findUserRes[0];
    const isValid = await bcrypt.compare(password, user.password);

    if (!isValid) {
      authError();
    }

    const res = {
      id: user.id,
      email: user.email,
      address: user.address,
      lat: user.lat,
      lng: user.lng,
      nick: user.nick,
      avatar: user.avatar,
      admin: user.admin,
    };

    return res;
  });
}

```

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				114
Змн.	Арк.	№ докум.	Підпис	Дата		

Код методу paypalGetMoneyToUser

```

paypalGetMoneyToUser = async (req, res) =>
  this.errorWrapper(res, async () => {
    const { userId } = req.userData;
    const { amount, type, typeValue } = req.body;

    const feeInfo = await this.systemOptionModel.getFeeInfo();
    const fee = calculateFee(feeInfo, amount);
    const moneyWithoutFee = Number(amount) - fee;

    const result = await sendMoneyToUser(
      type,
      typeValue,
      moneyWithoutFee.toFixed(2),
      "USD"
    );

    const payStory = async (waitingStatus = false) => {
      const newBalance = await this.userModel.rejectBalance(userId,
amount);

      const createdId =
        await this.paymentTransactionModel.createWithdrawalByPaypal(
          userId,
          Number(amount).toFixed(2),
          fee,
          waitingStatus
        );

      return { createdId, newBalance };
    };

    if (result.error) {
      if (
        result.error.toLowerCase() ==
        "Sender does not have sufficient funds. Please add funds and re-
try.".toLowerCase()
      ) {
        const { createdId, newBalance } = await payStory(true);

        await this.getMoneyRequestModel.create(
          createdId,
          userId,
          moneyWithoutFee.toFixed(2),
          "paypal",
          { type, typeValue }
        );
      }
    }
  });

```

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		115

```

        return this.sendResponseSuccess(
            res,
            "Operation completed successfully",
            { newBalance }
        );
    } else {
        return this.sendResponseError(res, "Operation error", 402);
    }
} else {
    const { newBalance } = await payStory();

    await this.serverTransactionModel.createReplenishmentByPaypalFee(
        userId,
        moneyWithoutFee.toFixed(2)
    );

    return this.sendResponseSuccess(
        res,
        "Operation completed successfully",
        { newBalance }
    );
}
});

```

Додаток Д

Код хука useEffect в якому прив'язуються реакції на повідомлення

```

useEffect(() => {
    if (!io) return;

    io.on("created-chat", (data) => onGetNewChat(data));
    io.on("created-group-chat", (data) => {
        onGetNewChat({
            chatId: data.chatId,
            type: data.message.type,
            chatType: data.message.chatType,
            content: data.message.content,
            chatAvatar: data.avatar,
            chatName: data.name,
            timeSended: data.message.timeSended,
            deleteTime: null,
        });
    });

    io.on("success-sended-message", (data) => {
        onGetMessageForSockets(data.message);
    });

    io.on("get-message", (data) => onGetMessageForSockets(data.message));

```

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		116

```

io.on("get-message-list", (data) =>
  data.messages.forEach((message) => onGetMessageForSockets(message))
);

io.on("success-deleted-message", (data) => onDeleteMessageForSockets(data));
io.on("deleted-message", (data) => onDeleteMessageForSockets(data));
io.on("success-updated-message", (data) => onUpdateMessageForSockets(data));
io.on("updated-message", (data) => onUpdateMessageForSockets(data));
io.on("typing", (data) => changeTypingForSockets(data, true));
io.on("stop-typing", (data) => changeTypingForSockets(data, false));
io.on("online", (data) => changeOnlineForSockets(data, true));
io.on("offline", (data) => changeOnlineForSockets(data, false));

io.on("chat-kicked", (data) => {
  onGetMessageForSockets({ chatId: data.chatId, ...data.message });
  deactivateChat(data.chatId, data.time);
});

io.on("file-part-uploaded", async ({ tempKey, message = null }) => {
  const nextPartData = await onSuccessSendBlobPart(tempKey);

  if (!nextPartData) return;
  if (nextPartData == "success saved" && message) {
    onGetMessageForSockets(message);
    return;
  }

  onUpdateMessagePercent({ tempKey, percent: nextPartData["percent"]
});
  setTimeout(() => io.emit("file-part-upload", { ...nextPartData }),
1);
});

io.on("message-cancelled", async ({ tempKey }) =>
  onCancelledMessage({ tempKey })
);
}, [io]);

```

Додаток Е

Код методу onSendMessage

```

onSendMessage = async (data, sessionInfo) => {
  const userId = sessionInfo.userId;
  const sender = sessionInfo.user;

  const message = await this.chatController.__createMessage(data,
userId);

```

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		117

```

    if (data.chatType == "personal") {
        let chatId = data.chatId;

        if (!data.chatId) {
            await this.__onCreateNewPersonalChat(message, data, userId);
            chatId = await this.chatController.chatModel.hasPersonal(...);
        }

        this.socketController.sendSocketMessageToUsers(...);

        this.chatController.sendMessageNotification(...);
    }

    if (data.chatType == "group" || data.chatType == "system") {
        const chatUsers = await
this.chatController.__getChatUsers(data.chatId);
        const chatUserIds = chatUsers.map((chat) => chat.userId);
        const usersToGetMessage = chatUserIds.filter((id) => id != userId);

        if (usersToGetMessage.length) {
            this.socketController.sendSocketMessageToUsers(...);

            const chat = await
this.chatController.chatModel.getById(data.chatId);

            const relations = await
this.chatController.chatModel.getChatRelations(
                data.chatId
            );

            relations.forEach((relation) => {
                if (relation.userId == userId) {
                    return;
                }

                this.chatController.sendMessageNotification(...);
            });
        } else if (data.chatType == "system") {
            this.socketController.sendSocketMessageToAdmins(...);
        }
    }

    message["tempKey"] = data["tempKey"];
    message["getterId"] = data["getterId"];

    this.socketController.sendSocketMessageToUsers(
        [userId, "success-sended-message", {message}]
    );
};

```

Код методу createMessage

```

__createMessage = async (data, userId) => {
  const localSend = async (chatId, userId) => {
    const messageId = await this.chatModel.createMessage(
      chatId, userId, data.typeMessage, data.content
    );
    return await this.chatModel.getMessageById(messageId);
  };

  if (data["chatType"] == "personal") {
    let chatId = await this.chatModel.hasPersonal(data["getterId"],
userId);

    if (chatId) {
      return await localSend(chatId, userId);
    } else {
      chatId = await this.chatModel.create("personal");
      const users = [
        { id: data["getterId"], role: "member" },
        { id: userId, role: "member" },
      ];
      await this.chatModel.addManyUsers(chatId, users);
      const message = await localSend(chatId, userId);

      message["getterInfo"] = await this.userModel.getUserInfo(
        data["getterId"]
      );
    }

    return message;
  }
  else if (data["chatType"] == "system") {
    const chatId = data["chatId"];
    const userInfo = await this.userModel.getUserInfo(userId);
    const hasUserAccess = await this.chatModel.hasUserAccess(chatId,
userId);

    if (hasUserAccess) {
      return await localSend(chatId, userId);
    } else {
      if (userInfo["admin"]) {
        return await localSend(chatId, userId);
      }

      return null;
    }
  }
  else {
    return await localSend(data["chatId"], userId);
  }
};

```

		Вербовський О.Ю.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Локтікова Т.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		119

Код методу onFilePartUpload

```

onFilePartUpload = async (data, sessionInfo) => {
  const userId = sessionInfo.userId;
  const sender = sessionInfo.user;
  const { tempKey: tempKey, data: fileBody, type, last } = data;

  const filename = await this.chatController.__uploadToFile(
    userId, tempKey, fileBody, type
  );

  if (!last) {
    this.socketController.sendSocketMessageToUsers(...);
    return;
  }

  const dataToSend = {...};

  const message = await this.chatController.__createMessage(
    dataToSend,
    userId
  );

  if (!data.chatId && data.chatType == "personal")
    await this.__onCreateNewPersonalChat(message, dataToSend, userId);

  await this.chatController.__deleteFileAction(userId, tempKey);

  message["tempKey"] = tempKey;

  if (data.chatType == "group") {
    const chatUsers = await
this.chatController.__getChatUsers(data.chatId);
    const chatUserIds = chatUsers.map((chat) => chat.userId);
    const usersToGetMessage = chatUserIds.filter((id) => id != userId);
    this.socketController.sendSocketMessageToUsers(...);
  } else {
    this.socketController.sendSocketMessageToUsers(...);
    message["getterId"] = data.getterId;
  }

  this.socketController.sendSocketMessageToUsers(...);
};

```