МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»

Кафедра інженерії програмного забезпечення

КУРСОВА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

з дисципліни: «МОДЕЛЮВАННЯ ТА АНАЛІЗ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

на тему:

«РОЗРОБКА ТА АНАЛІЗ ПЛАТФОРМИ ДЛЯ ВИКОНАННЯ ЗАВДАНЬ ЗА НАГОРОДУ З ВБУДОВАНИМ ЧАТОМ»

студента IV курсу групи ІПЗ-20-1
спеціальності 121 «Інженерія програмного забезпечення»
Вербовський Олександр Юрійович
(прізвище, ім'я та по-батькові)

Керівник: В.Л.Левківський

Дата захисту: "_" _____ 2023р.
Національна шкала _____
Кількість балів: _____
Оцінка: ECTS _____

Члени комісії

_____ (підпис)	І.І. Сугоняк (прізвище та ініціали)
_____ (підпис)	С.М. Кравченко (прізвище та ініціали)
_____ (підпис)	В.Л.Левківський (прізвище та ініціали)

Житомир – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
Факультет інформаційно-комп'ютерних технологій
Кафедра інженерії програмного забезпечення
Освітній рівень: бакалавр
Спеціальність 121 «Інженерія програмного забезпечення»

«ЗАТВЕРДЖУЮ»
Завідувач. кафедри ІПЗ
_____ Т.А.Вакалюк
"____"_____2023 р.

ЗАВДАННЯ
НА КУРСОВУ РОБОТУ СТУДЕНТУ
Вербовському Олександру Юрійовичу

1. Тема роботи: "Розробка та аналіз платформи для виконання завдань за винагородження з вбудованим чатом", керівник роботи: Левківський В. Л.
2. Строк подання студентом: "21" грудня 2023р.
3. Вихідні дані до роботи: спроектована модель платформи
4. Зміст розрахунково-пояснювальної записки(перелік питань. Які підлягають розробці)
   - Аналіз теоретичних засад моделювання програмного забезпечення;
   - Аналіз та опис вимог користування;
   - Методи модулювання функцій та поведінки системи;
   - Проектування об'єктної структури системи;
   - Фізичне моделювання програмних комплексів;
   - Кодогенерація із моделей.

5. Перелік графічного матеріалу(з точним зазначенням обов'язкових креслень)
   1. Презентація до КР: https://gitlab.com/2020-2024/ipz-20-1/verbovskiy-alexandr/cursove_model/-/blob/master/cursova.doc.
   2. Посилання на репозиторій: https://gitlab.com/2020-2024/ipz-20-1/verbovskiy-alexandr/cursove_model/-/blob/master/cursova.ppt.
6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посади консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1,2,3	Сугоняк І.І., к.т.н., доц.		

7. Дата видачі завдання: " 14 " вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів курсового проекту	Строк виконання етапів проекту	Примітки
1	Вибір тематичного напрямку та узгодження теми курсової роботи	18.09	Виконано
2	Розробка завдання, визначення методів та засобів реалізації поставленої задачі	23.09	Виконано
3	Аналіз теоретичних матеріалів за напрямком дослідження, вивчення предметної області	01.11	Виконано
4	Розробка концептуальної та логічної моделі програмної системи	20.11	Виконано
5	Розробка фізичної моделі та реалізація прототипу програмного забезпечення	14.12	Виконано
6	Оформлення пояснювальної записку	15.12	Виконано
7	Захист		

Студент _____ Вербовський О.Ю.
(підпис) (прізвище та ініціали)

Керівник проекту _____ Левківський В.Л.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Завданням на курсовой проект (роботу) було дослідження особливостей моделювання та аналізу програмних комплексів за визначеним темою курсової роботи напрямком з використанням CASE-технологій.

Пояснювальна записка до курсового проекту (роботи) на тему «Розробка та аналіз платформи для організації та виконання завдань з вбудованим чатом» складається з вступу, чотирьох розділів, висновків, списку використаної літератури та додатку.

Текстова частина викладена на 50 сторінках друкованого тексту.

Пояснювальна записка має 28 сторінку додатків. Список використаних джерел містить 25 найменувань і займає 3 сторінки. В роботі наведено 21 рисунків. Загальний обсяг роботи – 78 сторінок.

Ключові слова: користувач, сокет, чат, повідомлення, ідентифікатор, дія, робота, суперечка, створити, оновити, видалити, перевірити, відносини, вміст, посилання, пропозиція, статус, адміністратор

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД – база даних

КР – курсова робота

CRUD – create, read, update, delete.

ІС – інформаційна система

ТЗ – технічне завдання

СУБД – система управління базою даних

РРЧ – режим реального часу

ЗМІСТ

ВСТУП

У цьому курсовому проекті буде наведено процес дослідження особливостей моделювання та аналізу програмних комплексів за визначеним темою курсової роботи напрямком з використанням CASE-технологій.

Актуальність теми.

Розвиток CASE-технологій в сучасному програмуванні прямо пов'язаний з розробкою різного програмного забезпечення. Сучасний ринок вимагає не лише швидкого інноваційного розвитку програм, але і високого рівня організації та контролю за процесами роботи. CASE-технології, забезпечуючи підтримку в розробці, дозволяють ефективно впоратися із складністю завдань та швидко і без багів внести зміни. Необхідність ефективного управління завданнями зумовлює використання цих технологій для автоматизації процесів планування, виконання, та моніторингу завдань у проекті. Відзначаючи прогрес у розвитку CASE-технологій, важливо розуміти, як вони сприяють створенню структурованих та узгоджених систем управління завданнями, що є ключовим елементом успішної розробки програмного продукту. Такий інтегрований підхід дозволяє підтримувати високий стандарт ефективності, що є важливим у вимогливих умовах ринку програмного забезпечення.

Метою роботи є дослідження особливостей моделювання та аналізу програмних комплексів за визначеним темою курсової роботи напрямком з використанням CASE-технологій.

Завданням на курсову роботу є:

- Аналіз теоретичних засад моделювання програмного забезпечення;
- Аналіз та опис вимог користування;
- Методи модулювання функцій та поведінки системи;
- Проектування об’єктної структури системи;
- Фізичне моделювання програмних комплексів;
- Кодогенерація із моделей.

Об'єктом дослідження є методи та засоби проектування програмного забезпечення та уніфікація процесу проектування.

Предметом дослідження можливості застосування CASE-засобів проектування програмного забезпечення.

РОЗДІЛ 1. ТЕОРЕТИЧНИЙ АНАЛІЗ ІНФОРМАЦІЙНИХ ПОТОКІВ ТА ОСОБЛИВОСТЕЙ ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ

1.1. Технічне завдання на розробку системи

Основні цілі:

- Розробка функціональної платформи: Основною метою проекту є створення високофункціональної онлайн-платформи з вбудованим чатом для організації та виконання різноманітних разових завдань і послуг між користувачами в режимі реального часу.

- Створення зручного середовища для користувачів: Забезпечити зручний та інтуїтивно зрозумілий інтерфейс для користувачів, який сприятиме простоті використання платформи та забезпечить позитивний користувацький досвід.

- Забезпечення безпеки і конфіденційності: Гарантувати високий рівень безпеки для всіх користувачів, включаючи перевірку ідентифікації, захист від шахрайства(у кожного юзера буде статистика як по ролі продавця так і по ролі клієнта. В ній буде інформація про кількість успішних виконань замовлень, кількість звертань до адміністраторів для вирішення спорів і т.д.) та збереження конфіденційності особистих даних.

- Монетизація та прибутки: Розробити ефективну стратегію монетизації, включаючи встановлення комісій за операції на платформі, платні функції або інші джерела доходу.

- Залучення та утримання користувачів: Привертати нових користувачів на платформу та створювати механізми для збереження і залучення постійних клієнтів.

- Географічна підв’язка: Надавати змогу користувачам знаходити оптимальні точки для виконання завдань і отримання прибутку.

Представлення проекту:

Проект полягає в розробці та впровадженні інноваційної онлайн-платформи для розміщення та виконання різноманітних разових завдань і послуг. Платформа надасть можливість роботодавцям легко знаходити осіб, готових виконати різні завдання, включаючи кур'єрські послуги, послуги перевезення, подарунки, послуги по дому і багато інших.

Основні переваги платформи включають вбудований чат для зручного спілкування між користувачами, систему рейтингів і відгуків для забезпечення довіри, а також високий рівень безпеки для всіх учасників.

Платформа покликана забезпечити якісні та зручні послуги для роботодавців і робітників, спрощуючи процес знаходження та виконання різних завдань. Команда проекту(в образі одного студента) вірить, що ця ідея має великий потенціал для успіху на ринку та може стати цінним інструментом для покращення як повсякденного життя людей, так і бізнесових можливостей для виконавців послуг.

1.2. Обґрунтування вибору засобів моделювання

Під час виконання курсової роботи був вибір використання між трьома засобами моделювання – draw.io, starUML та lucidchart.

Draw.io:

Тип інструменту: Онлайн-інструмент для створення діаграм.

Підтримка UML: Забезпечує можливість створення різних видів діаграм UML.

Інтерфейс та використання: Легкий у використанні, не вимагає встановлення.

Спільна робота: Має можливості спільної роботи та збереження у хмарному сховищі.

Вартість: Зазвичай безкоштовний для основного використання, але може вимагати плати за додаткові функції.

Підтримка кодогенерації: Відсутня або обмежена.

Можливості експорту/імпорту: Підтримка експорту діаграм у різних форматах.

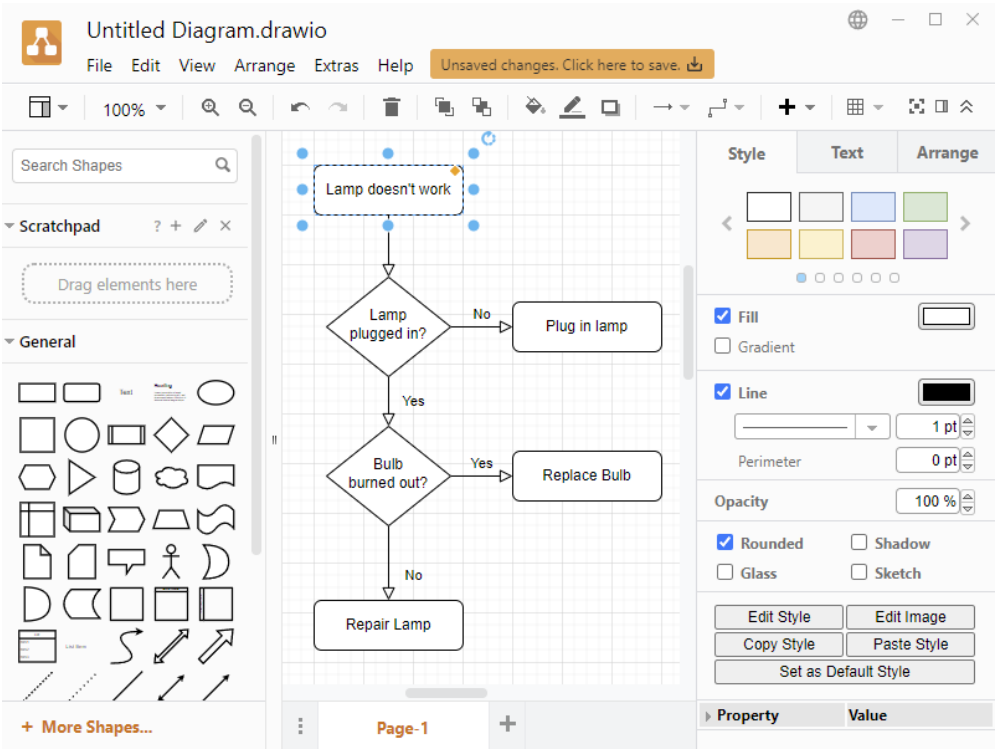

Рис. 1.1. Інтерфейс draw.io

StarUML:

Тип інструменту: Спеціалізоване програмне забезпечення для моделювання UML.

Підтримка UML: Надає розширені можливості для моделювання на мові UML, включаючи всі основні діаграми.

Інтерфейс та використання: Має потужний функціонал, але може вимагати певного часу для вивчення.

Спільна робота: Зазвичай призначений для індивідуального використання, але може підтримувати взаємодію зі сховищами коду.

Вартість: Має безкоштовну версію та платні плани для продуктивнішого використання.

Підтримка кодогенерації: Присутня, можливість генерації базового коду.

Можливості експорту/імпорту: Розширені можливості експорту та імпорту моделей для роботи у великих проектах.

Спрощення процесу реалізації: Вбудовані можливості кодогенерації спрощують перехід від моделювання до реалізації.

Розширюваність інструменту: Можливість встановлення плагінів для розширення функціоналу.

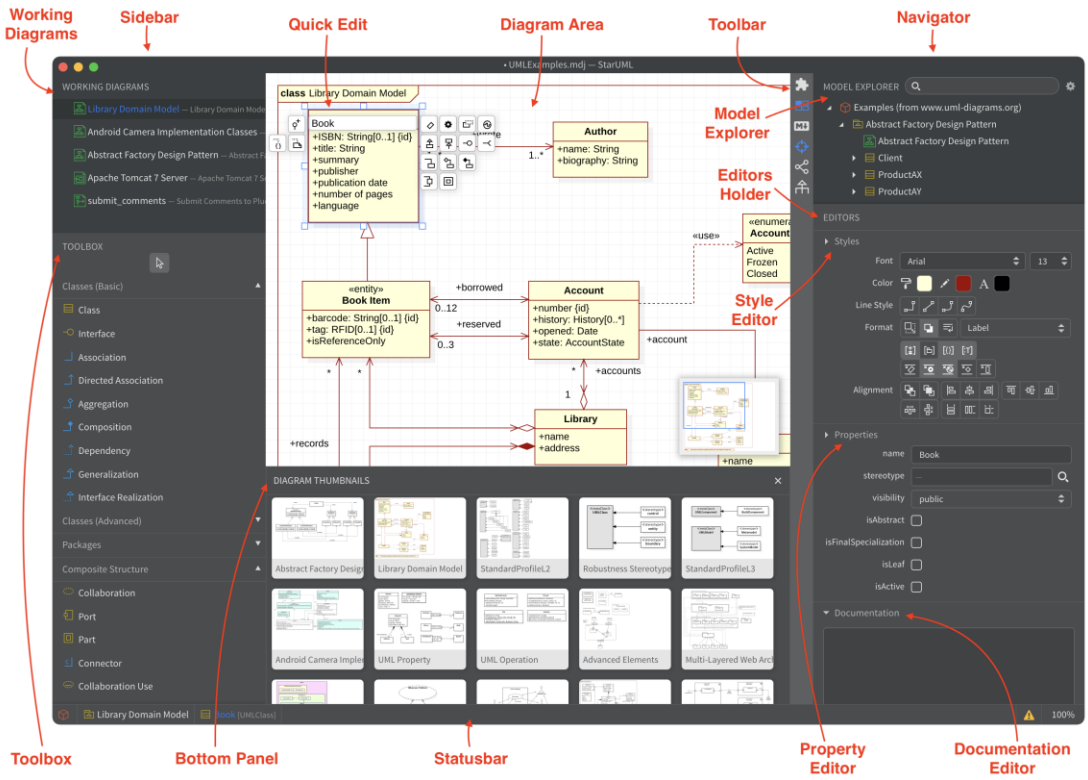

Рис. 1.2. Інтерфейс starUML

Lucidchart:

Тип інструменту: Онлайн-інструмент для створення діаграм та схем.

Підтримка UML: Має можливість створення діаграм UML та інших типів діаграм.

Інтерфейс та використання: Легкий у використанні з інтуїтивним інтерфейсом.

Спільна робота: Підтримка режиму спільної роботи для команд.

Вартість: Платний сервіс, але надає безкоштовний пробний період.

Підтримка кодогенерації: Відсутня або обмежена.

Можливості експорту/імпорту: Підтримка експорту діаграм у різних форматах.

Зручність спільної роботи: Режим спільної роботи відповідає потребам команд та дозволяє ефективно взаємодіяти.

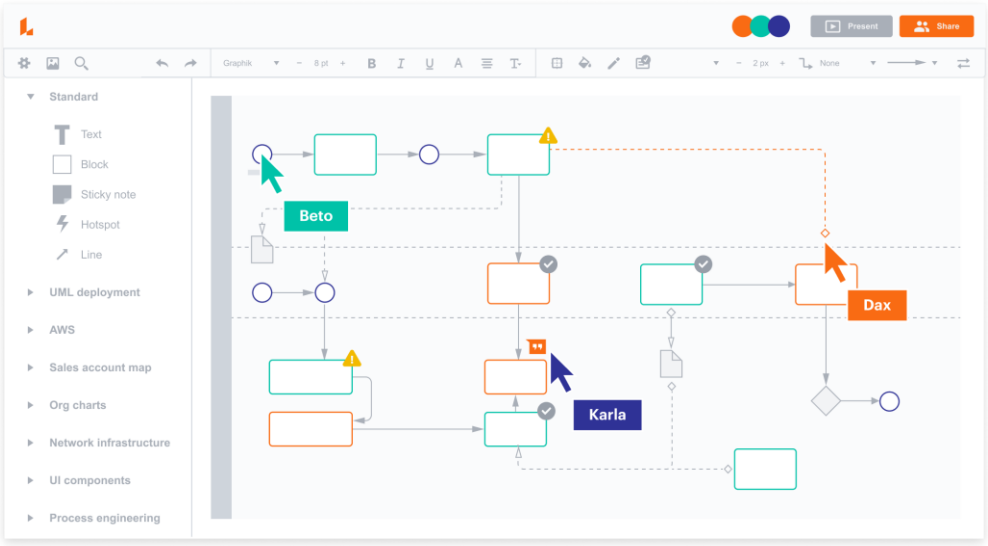

Рис. 1.3. Інтерфейс lucidchart

Так як, для деяких цілей краще підходив draw.io, а для деяких – starUML, то під час виконання, було використано обидва варіанти. Наприклад, через те, що draw.io це онлайн сервіс, то він був дуже зручним для вдосконалення діаграм в різні моменти без прив'язки до ПЗ та до певного комп'ютера. Це дуже пришвидшило розробку та скоротило простої. В той же момент StarUML мав можливість для кодогенерації, на відміну від draw.io, що допомогло пришвидшити перехід від візуального плану до конкретики.

1.3. Аналіз вимог до програмного продукту

Вимоги від роботодавців (замовників):

- Реєстрація та профіль: Роботодавці повинні мати можливість реєстрації на платформі та створення профілю, в якому вони можуть

представити себе, вказати контактну інформацію і надати докладний опис завдань.

- Створення замовлень: Роботодавці повинні мати можливість легко створювати нові замовлення, визначати вид роботи, деталі завдання, бюджет і термін виконання.

- Взаємодія з виконавцями: Вони повинні мати можливість обговорювати деталі з виконавцями через вбудований чат, надавати додаткову інформацію і встановлювати умови співпраці.

- Перегляд профілів виконавців: Роботодавці повинні мати можливість переглядати профілі виконавців, включаючи їхні навички, досвід і відгуки від інших роботодавців.

- Підтримка замовлень: Важливо, щоб роботодавці могли слідкувати за статусами своїх замовлень, відстежувати виконавців і надавати відгуки після завершення робіт.

Вимоги від виконавців (робітників):

- Реєстрація та профіль: Виконавці повинні мати можливість реєстрації на платформі і створення свого профілю, в якому вони представляють свої навички, досвід і контактну інформацію.

- Пошук та перегляд замовлень: Вони повинні мати можливість шукати доступні замовлення на платформі, переглядати їх деталі і визначати, які завдання їм цікаві.

- Взаємодія з роботодавцями: Виконавці повинні мати можливість спілкуватися з роботодавцями через вбудований чат, відповідати на запити, надавати додаткову інформацію і уточнювати деталі.

- Подання пропозицій: Вони повинні мати можливість подавати свої пропозиції на замовлення, вказуючи бюджет і обсяг робіт.

- Виконання замовлень: Важливо, щоб виконавці виконували замовлення з уважністю до деталей, вказаних у завданні, і дотримувалися умов співпраці.

- Відгуки і рейтинг: Вони повинні мати можливість отримувати відгуки від роботодавців і отримувати рейтинг за якість роботи.

- Підтримка співпраці: Важливо, щоб виконавці могли надавати підтримку під час виконання завдань і взаємодіяти з роботодавцями через чат.

Характеристика об'єкта комп'ютеризації (платформа для пошуку замовлень):

- Перегляд інформації про активні замовлення: Користувачі платформи будуть мати можливість переглядати інформацію про активні замовлення, включаючи опис завдань, бюджет і термін виконання. Це допоможе їм знайти підходящі робочі можливості.

- Пошук та фільтрація замовлень: Користувачі зможуть легко шукати замовлення за різними критеріями, такими як місцезнаходження, тип роботи, бюджет і т. д. Фільтрація допоможе їм знайти точно те, що вони шукають.

- Зручність використання додатку: Платформа повинна бути зручною в користуванні і мати інтуїтивний інтерфейс. Користувачі повинні легко розуміти, як використовувати функції додатку і швидко реагувати на замовлення.

- Простота навігації: Додаток повинен мати просту і логічну структуру навігації, щоб користувачі могли легко переходити між розділами та функціями.

- Зручний чат для взаємодії: Вбудований чат повинен бути легким у використанні і дозволяти користувачам спілкуватися між собою та обмінюватися інформацією щодо завдань.

- Можливість створення та редагування профілю: Користувачі повинні мати можливість створювати та редагувати свої профілі, включаючи контактну інформацію, навички та інші деталі.

- Зручна система оповіщень: Додаток може надсилати оповіщення користувачам про нові замовлення, повідомлення від роботодавців і виконавців, а також інші важливі події.

Функціональні та нефункціональні вимоги розміщені в Додатку А.

Глосарій розміщено в Додатку Б.

Аналіз вимог користувачів дав підстави для визначення варіантів використання додатку.

На рисунку 1.4 наведено діаграму використання програмної системи роботодавцями. На рисунку 1.5 наведено діаграму використання програмної системи робітниками. На рисунку 1.6 наведено діаграму використання програмної системи адміністраторами. Їх опис наведено у вимогах.

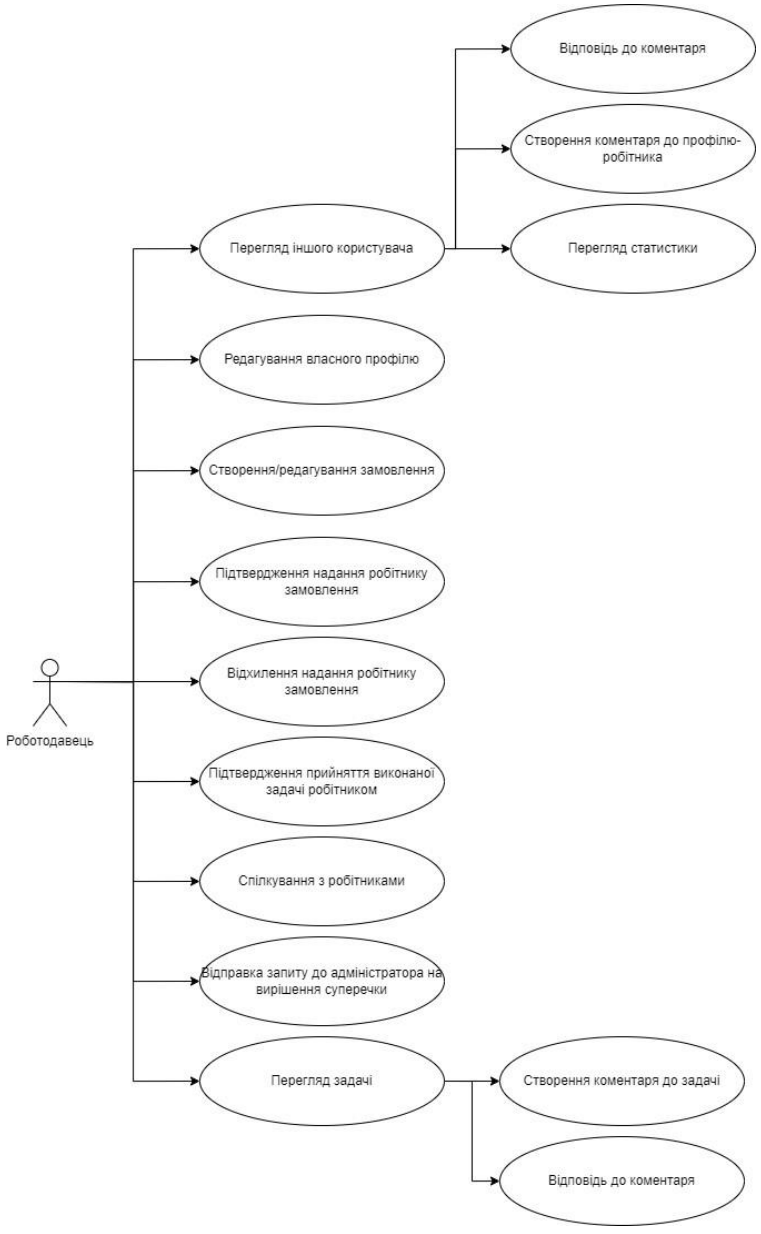

Рис. 1.4. Використання програмної системи роботодавцями

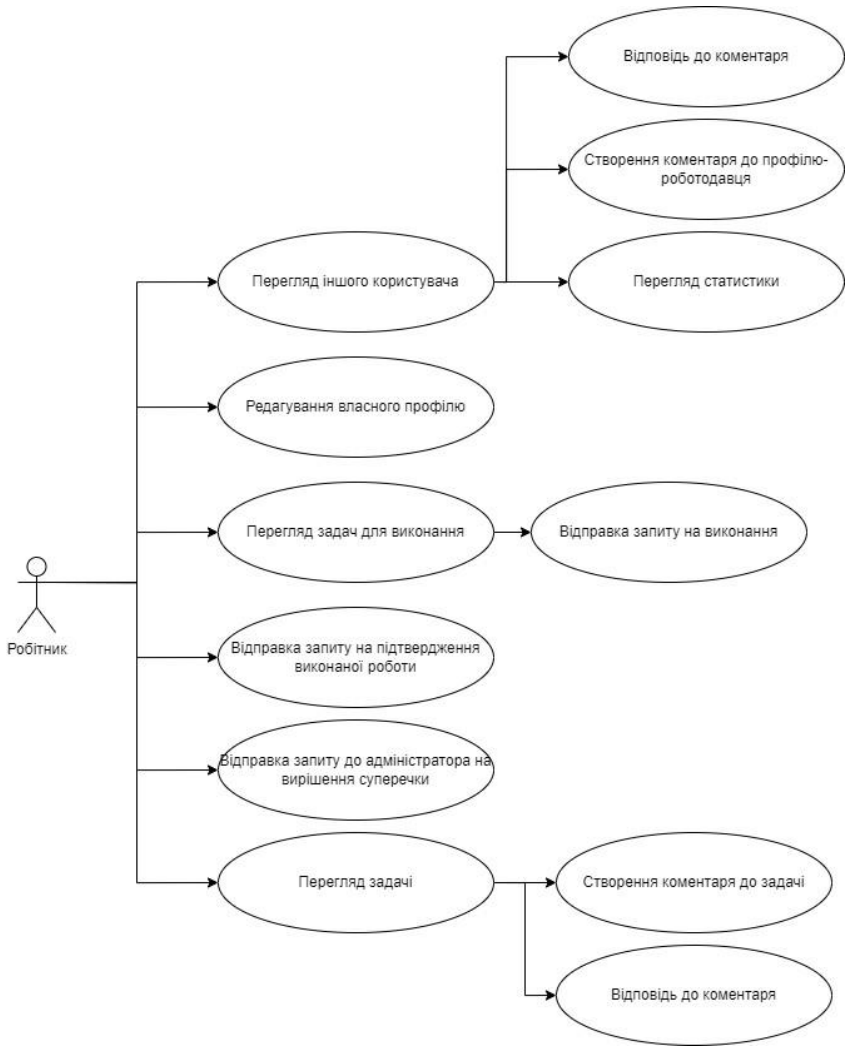

Рис. 1.5. Використання програмної системи робітниками

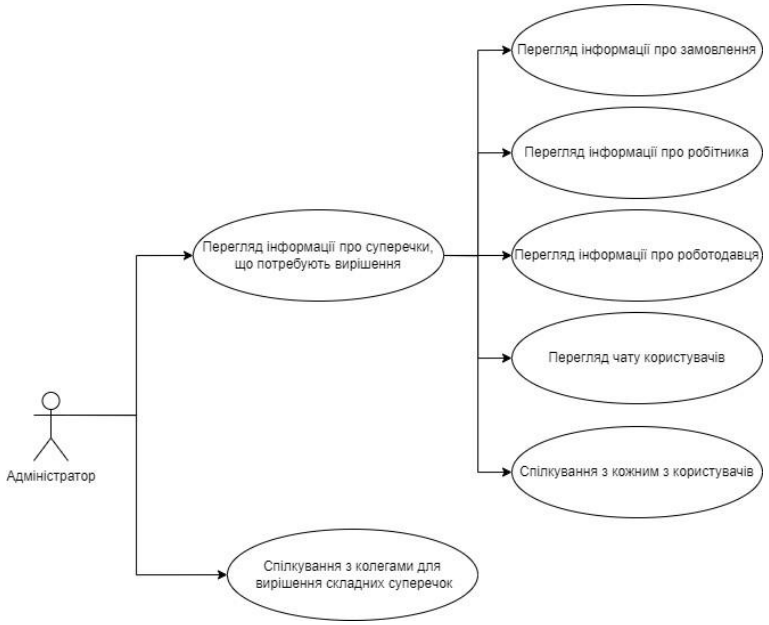

Рис. 1.6. Використання програмної системи адміністраторами

Висновок до першого розділу:

У цьому розділі ми провели аналіз вимог до програмного продукту – інноваційної онлайн-платформи для організації та виконання різноманітних завдань у режимі реального часу. Функціональні вимоги визначають, що система повинна надавати ефективні засоби реєстрації, створення профілів, управління замовленнями та забезпечувати зручну взаємодію між користувачами. Нефункціональні вимоги вкладають акцент на безпеку даних, конфіденційність, стратегію монетизації та стабільність системи. Результати аналізу визначають фундаментальні принципи розробки системи, спрямованої на створення зручного та безпечного середовища для виконання різноманітних робочих завдань.

РОЗДІЛ 2. ТЕОРЕТИЧНИЙ АНАЛІЗ ІНФОРМАЦІЙНИХ ПОТОКІВ ТА ОСОБЛИВОСТЕЙ ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ

2.1. Алгоритм роботи та стани програмної системи

Основні варіанти використання системи:

- Створення задач
- Подання заявки на роботу
- Зміна статусу заявки
- Створення суперечок
- Вирішення суперечок

Доріжки виконання(управління):

- Користувач
- Адміністратор
- Додаток
- Система

Основні стани:

- Очікування
- Контроль
- Виконання
- Операцій

Створення задач:

Очікування:

- Користувач ініціює створення нової задачі, вказуючи назву, ціну, опис та інші параметри.
- Система очікує введення користувача та готова прийняти нову задачу.

Контроль:

- Система перевіряє, чи введені дані коректні

Виконання операцій:

- Після успішного вводу та контролю, система заносить нову задачу до списку, що очікують на заявки виконання.

Подання заявки на роботу:

Очікування:

- Кандидат входить на платформу та відправляє запит на роботу.

- Кандидат заповнює очікувану почасову ціну та час, скільки йому потрібно на виконання завдання.

Контроль:

- Система перевіряє, чи коректно заповнені дані

Виконання операцій:

- Після успішного вводу та контролю, система заносить нову пропозицію до списку, що очікують на підтвердження.

- Виконавець/адмін/замовник можуть змінювати статуси відповідно до етапу пропозиції.

Зміна статусу задач:

Очікування:

- Користувач входить у систему та обирає пропозицію, для якої він хоче змінити статус.

- Система очікує на вибір користувача щодо дії зі зміною статусу пропозиції.

Контроль:

- Система перевіряє, чи користувач має відповідні права для зміни статусу пропозиції.

- Перевірка, чи обраний статус є допустимим для поточного етапу виконання пропозиції.

Виконання операцій:

- Після підтвердження користувача та успішного контролю, система змінює статус пропозиції відповідно до вибору користувача.

Зміна статусу задач:

Очікування:

- Конфлікт може виникнути через непорозуміння, різницю думок, некоректні умови чи рішення задач або інші причини.

Контроль:

- Система перевіряє чи має пропозиція якесь відношення до користувача, що дає запит на допомогу суперечки.

Виконання операцій:

- Після валідації, суперечка зберігається і відображається в адміністратора.

Створення суперечки:

Очікування:

- Конфлікт може виникнути через непорозуміння, різницю думок, некоректні умови чи рішення задач або інші причини.

Контроль:

- Система перевіряє чи має пропозиція якесь відношення до користувача, що дає запит на допомогу суперечки.

Виконання операцій:

- Після валідації, суперечка зберігається і відображається в адміністратора.

Вирішення суперечок:

Очікування:

- Адміністратор входить в систему та обирає суперечку, за описом

- Аналізує інформацію про суперечку і робить висновок

Контроль:

- Перевірка наявності необхідної ролі для зміни статусу суперечки.

- Перевірка коректності розбиття ціни(кому скільки потрібно передати) та зміни статусу задачі.

Виконання операцій:

- Після валідації, переводяться кошти на баланси користувачів і змінюється статус суперечки та пропозиції.

Далі зображено послідовність дій системи:

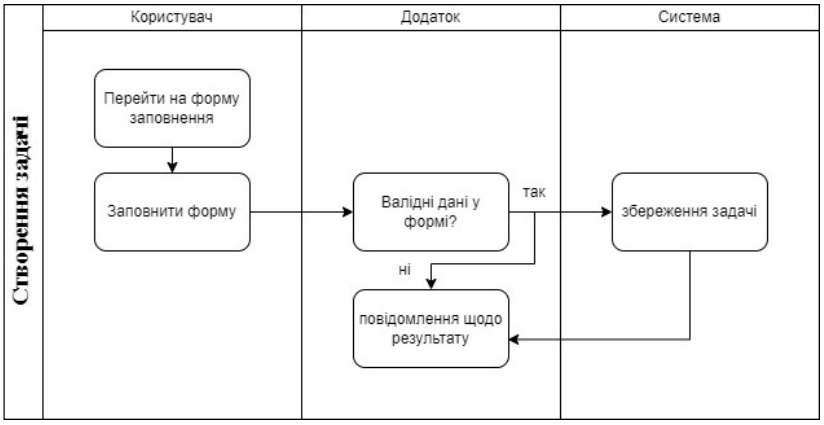

Рис. 2.1. Послідовність дій для створення задачі

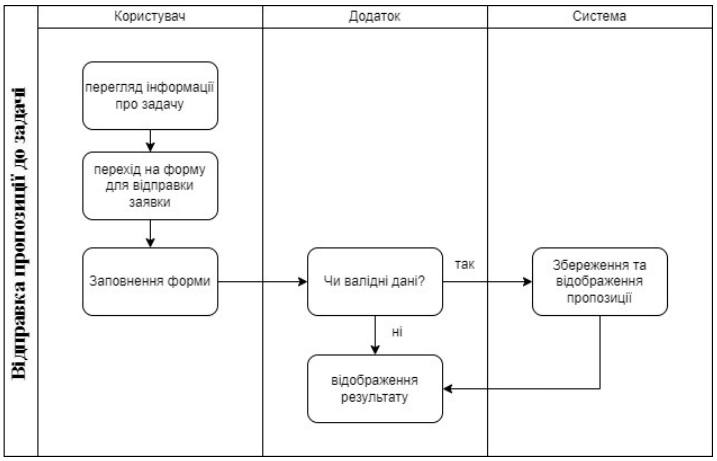

Рис. 2.2. Послідовність дій для створення пропозиції

Рис. 2.3. Послідовність дій для зміни статусу

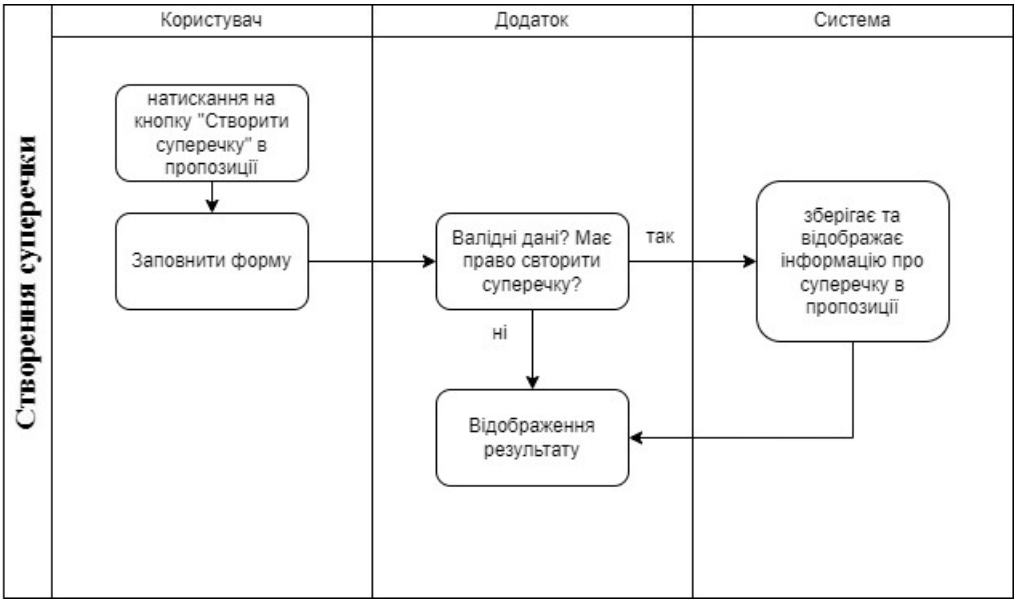

Рис. 2.4. Послідовність дій для подання суперечки

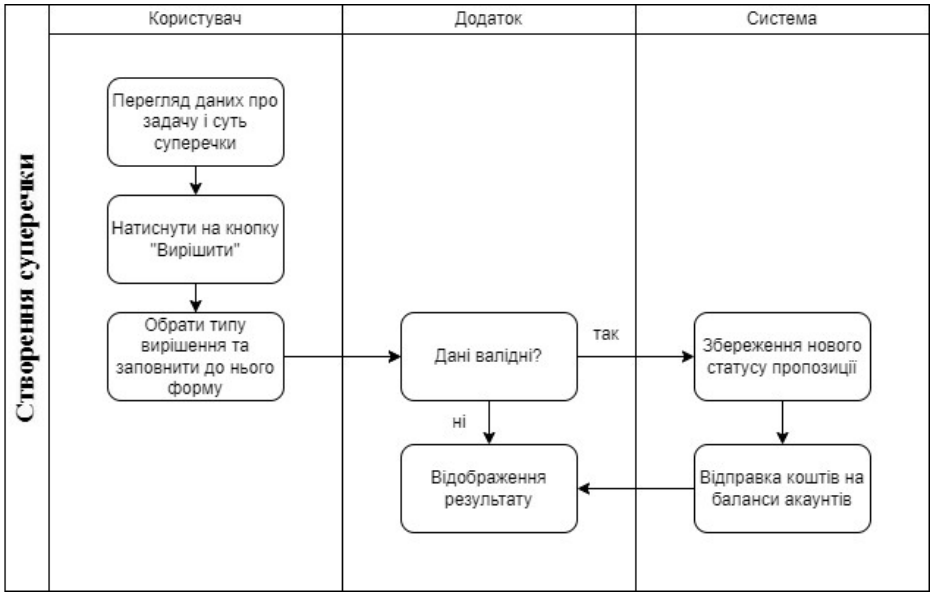

Рис. 2.5. Послідовність дій для вирішення суперечки

2.2. Об’єктно-орієнтована модель системи

Основні класи:

- **Users**: Цей клас представляє користувачів системи. Він містить інформацію про користувачів, таку як їх 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.

- **Sockets:** Цей клас відповідає за роботу з сокетами, які використовуються для спілкування в режимі реального часу між користувачами. Він містить інформацію про ідентифікатор сокету, ідентифікатор користувача, ідентифікатор чату, до якого підключено сокет.

- **Chats:** Цей клас представляє чати або групи для спілкування між користувачами. Він має ідентифікатор чату і тип чату.

- **ChatsUsers:** Клас, що відображає зв'язок між користувачами і чатами. Він містить ідентифікатор зв'язку, ідентифікатор чату та ідентифікатор користувача.

- **Messages:** Клас для роботи з повідомленнями, які відправляються в чатах. Він має ідентифікатор повідомлення, ідентифікатор чату, ідентифікатор користувача, прапорець прихованості повідомлення, тип повідомлення та час створення повідомлення.

- **MessagesContents:** Цей клас представляє вміст повідомлень, який може бути текстовим або мультимедійним. Він має ідентифікатор вмісту, ідентифікатор повідомлення, сам вміст та час останнього редагування.

- **UsersActions:** Клас, який використовується для ведення журналу дій користувачів. Він містить ідентифікатор дії, ідентифікатор користувача, тип дії, додаткові дані та ключ.

- **Jobs:** Цей клас представляє роботи або завдання, які користувачі можуть створювати та виконувати. Він містить інформацію про роботу, таку як ідентифікатор, назву, ціну, адресу, опис, координати місця, автора та час створення.

- **PasswordResetLinks:** Клас, що відповідає за управління посиланнями для скидання паролю користувачів. Він містить інформацію про ідентифікатор посилання, ідентифікатор облікового запису користувача, токен скидання паролю та час створення посилання.

- **JobRequests:** Клас для роботи з запитами на виконання робіт. Він містить інформацію про ідентифікатор запиту, ідентифікатор роботи, ідентифікатор користувача, ціну, час виконання, статус та час створення запиту.

- **JobDisputes:** Клас, який використовується для управління суперечками щодо виконання робіт. Він містить інформацію про ідентифікатор суперечки, ідентифікатор запиту на виконання роботи, ідентифікатор користувача, ідентифікатор адміністратора, опис суперечки, статус та час створення та оновлення.

Дані класи є представниками компонентів, зв'язок між якими представлений у вигляді діаграми на рисунку 12. Опис схем наведено вище.

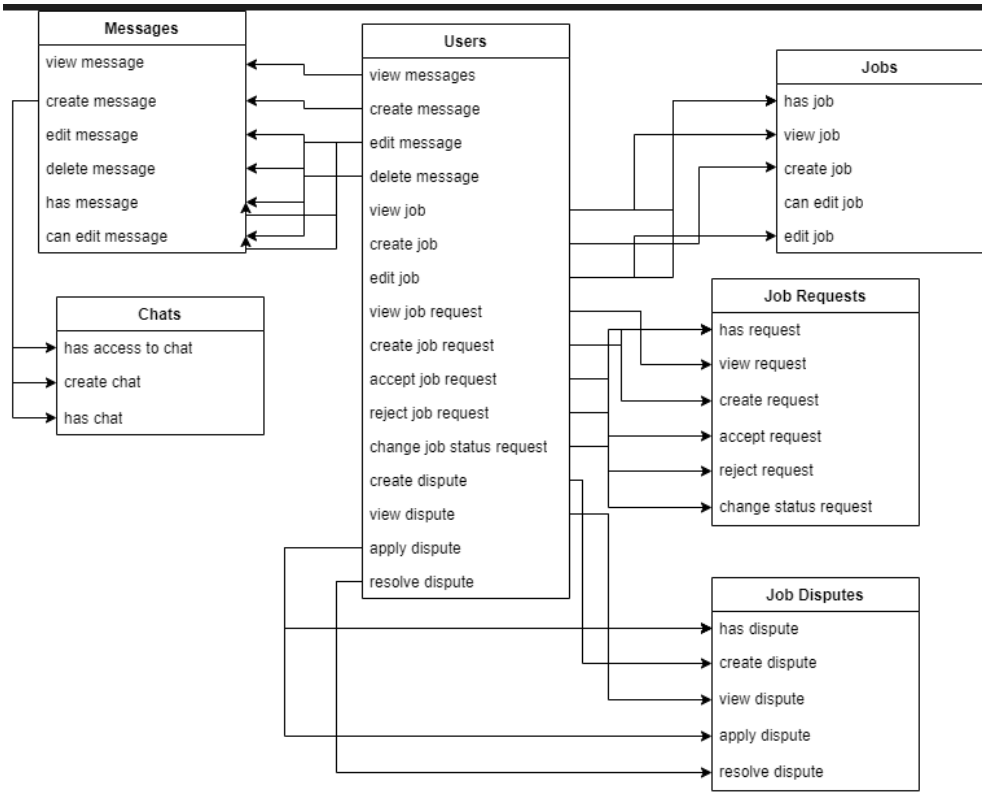

Рис. 2.6. Діаграма взаємодії компонентів

Атрибути класів:

Users:

- id: Унікальний ідентифікатор користувача.
- email: Електронна пошта користувача.
- password: Захешований пароль користувача.
- nick: Ім'я користувача або псевдонім.

- address: Адреса користувача.

- avatar: Шлях до аватара користувача.

- coords: Широта та довгота місцезнаходження користувача.

- profile_authorized: Прапор, що позначає, чи авторизований користувач в своєму профілі.

- admin: Прапор, що позначає, чи є користувач адміністратором.

- time_created: Мітка часу створення користувача.

Chats:

- id: Унікальний ідентифікатор чату.

- type: Тип чату (наприклад, "personal" для особистих чатів).

ChatsUsers:

- id: Унікальний ідентифікатор відношення між користувачами та чатами.

- chat_id: Ідентифікатор чату.

- user_id: Ідентифікатор користувача.

Messages:

- id: Унікальний ідентифікатор повідомлення.

- chat_id: Ідентифікатор чату, до якого відноситься повідомлення.

- sender_id: Ідентифікатор користувача, який надіслав повідомлення.

- hidden: Прапор, що позначає, чи приховане повідомлення.

- type: Тип повідомлення.

- time_created: Мітка часу створення повідомлення.

MessagesContents:

- id: Унікальний ідентифікатор вмісту повідомлення.

- message_id: Ідентифікатор повідомлення, до якого відноситься вміст.

- content: Вміст повідомлення.

- time_edited: Мітка часу редагування вмісту повідомлення.

UsersActions:

- id: Унікальний ідентифікатор дії користувача.

- user_id: Ідентифікатор користувача, якому належить дія.

- type: Тип дії.

- data: Додаткові дані про дію.

- key: Ключ дії.

Jobs:

- id: Унікальний ідентифікатор роботи.

- title: Назва роботи.

- price: Вартість роботи.

- address: Адреса роботи.

- description: Опис роботи.

- lat: Широта місцезнаходження роботи.

- lng: Довгота місцезнаходження роботи.

- author_id: Ідентифікатор користувача, який створив роботу.

- time_created: Мітка часу створення роботи.

PasswordResetLinks:

- id: Унікальний ідентифікатор посилання для скидання паролю.

- account_id: Ідентифікатор облікового запису, якому належить посилання.

- reset_token: Токен скидання паролю.

- created_at: Мітка часу створення посилання.

JobRequests:

- id: Унікальний ідентифікатор запиту на роботу.

- job_id: Ідентифікатор роботи, до якої відноситься запит.

- user_id: Ідентифікатор користувача, який надіслав запит.

- price: Вартість роботи, запропонована користувачем.

- execution_time: Час виконання роботи, запропонований користувачем.

- status: Статус запиту.

- time_created: Мітка часу створення запиту.

JobDisputes:

- id: Унікальний ідентифікатор суперечки.

- job_request_id: Ідентифікатор запиту на роботу, до якого відноситься суперечка.

- user_id: Ідентифікатор користувача, який висунув суперечку.

- admin_id: Ідентифікатор адміністратора, який обраний для вирішення суперечки.

- description: Опис суперечки.

- status: Статус суперечки.

- created_at: Мітка часу створення суперечки.

- updated_at: Мітка часу оновлення суперечки.

Методи класів:

Users:

- create(email, password): Створює нового користувача з вказаною адресою електронної пошти та паролем. Виконує хешування паролю перед збереженням у базі даних.

- findByPasswordAndEmail(email, password): Перевіряє, чи існує користувач з вказаною адресою електронної пошти та чи вірний пароль.

- getUserInfo(userId): Отримує інформацію про користувача за його ідентифікатором.

- checkIsAdmin(userId): Перевіряє, чи користувач є адміністратором.

- updateUserProfile(userId, profileData): Оновлює профіль користувача з вказаними даними.

- updatePassword(accountId, password): Оновлює пароль користувача.

Sockets:

- findUserSockets(usersIds): Знаходить сокети користувачів з вказаними ідентифікаторами.

- create(socket, userId): Створює запис про сокет користувача у базі даних.

- delete(socket): Видаляє запис про сокет з бази даних за ідентифікатором сокету.

- clearAll(): Очищає всю таблицю сокетів (важливо розуміти наслідки цього дії).

Chats:

- create(type): Створює новий чат з вказаним типом.

- hasUserAccess(chatId, userId): Перевіряє, чи користувач має доступ до чату.

- addUser(chatId, userId): Додає користувача до чату.

- addManyUsers(chatId, usersIds): Додає кілька користувачів до чату одночасно.

- deleteUser(chatId, userId): Видаляє користувача з чату.

- hasPersonal(userId1, userId2): Перевіряє наявність особистого чату між двома користувачами.

- createPersonal(userId2, typeMessage, contentMessage, userId1): Створює особистий чат між двома користувачами.

ChatsUsers:

- getChatRelations(chatId): Отримує всі відносини користувачів до конкретного чату.

- addManyUsers(chatId, usersIds): Додає багатьох користувачів до чату.

Messages:

- createMessage(chatId, senderId, typeMessage, contentMessage): Створює нове повідомлення в чаті.

- hideMessage(messageId): Приховує повідомлення.

- getChatMessages(chatId, lastId, count): Отримує повідомлення з чату.

MessagesContents:

- addContentToMessage(messageId, content): Додає вміст до повідомлення.

- getMessageContent(messageId): Отримує вміст конкретного повідомлення.

UsersActions:

- create(userId, type, key, data): Створює нову дію (action) у базі даних.

- deleteById(userId, id): Видаляє дію за її ідентифікатором.

- deleteByKeyAndType(userId, key, type): Видаляє дію за ключем та типом.

- getByKeyAndType(userId, key, type): Отримує дію за ключем та типом.

- getById(id, userId): Отримує дію за її ідентифікатором.

- getUserActions(userId): Отримує всі дії користувача.

Jobs:

- create(title, price, address, description, lat, lng, userId): Створює нову роботу (job) у базі даних.

- edit(jobId, title, price, address, description, lat, lng): Редагує інформацію про роботу за її ідентифікатором.

- getById(jobId): Отримує інформацію про роботу за ідентифікатором.

- getByDistance(latitude, longitude, skippedIds, filter, limit): Отримує роботи за координатами та іншими параметрами.

- getByAuthor(authorId): Отримує роботи, створені конкретним автором.

- exists(jobId): Перевіряє існування роботи за ідентифікатором.

- checkAuthor(jobId, authorId): Перевіряє, чи користувач є автором роботи.

PasswordResetLinks:

- getLinkByAccountId(accountId): Отримує посилання для скидання пароля за ідентифікатором облікового запису (account).

- createLink(accountId, resetToken): Створює нове посилання для скидання пароля із зазначеним обліковим записом і токеном скидання пароля.

- getLinkByToken(token): Отримує посилання для скидання пароля за токеном скидання пароля.

- deleteLink(linkId): Видаляє посилання для скидання пароля за його ідентифікатором.

JobRequests:

- getById(proposalId): Отримує інформацію про пропозицію за ідентифікатором.

- getForJobAuthor(userId, skippedIds, filter, limit): Отримує пропозиції, які надійшли авторові роботи.

- getForProposalAuthor(userId, skippedIds, filter, limit): Отримує пропозиції, які стосуються користувача, що робить пропозицію.

- create(jobId, candidateId, pricePerHour, time): Створює нову пропозицію для роботи.

- changeStatus(proposalId, status): Змінює статус пропозиції.

- Методи для зміни статусу пропозиції: accept, reject, requestToCancel, acceptCancelled, requestToComplete, acceptCompleted.

- exists(proposalId): Перевіряє існування пропозиції за ідентифікатором.

- checkOwner(proposalId, userId): Перевіряє, чи користувач є власником пропозиції.

- checkJobOwner(proposalId, userId): Перевіряє, чи користувач є власником роботи для пропозиції.

JobDisputes:

- create(jobProposalId, userId, description): Створює новий спір (dispute) для пропозиції до роботи із зазначеним користувачем та описом.
- getById(id): Отримує інформацію про спір за ідентифікатором спору.
- getByProposalId(proposalId): Отримує спір, пов'язаний із пропозицією до роботи за ідентифікатором пропозиції.
- getByJobId(jobId): Отримує спір, пов'язаний із роботою за ідентифікатором роботи.
- setStatus(disputeId, status): Змінює статус спору за ідентифікатором спору.
- assignAdmin(disputeId, adminId): Призначає адміністратора для спору за ідентифікатором спору.

Опис діаграм наведено перед ними.

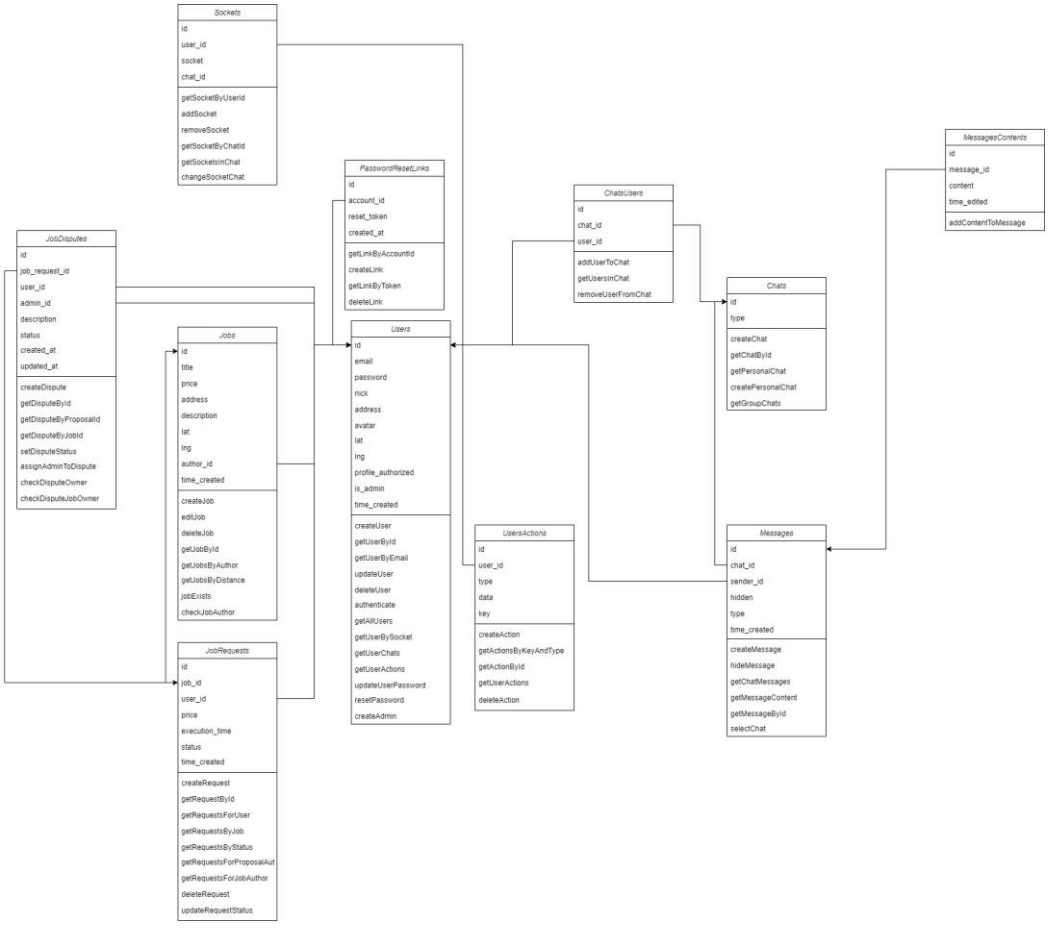

Рис. 2.7. Діаграма взаємодії класів

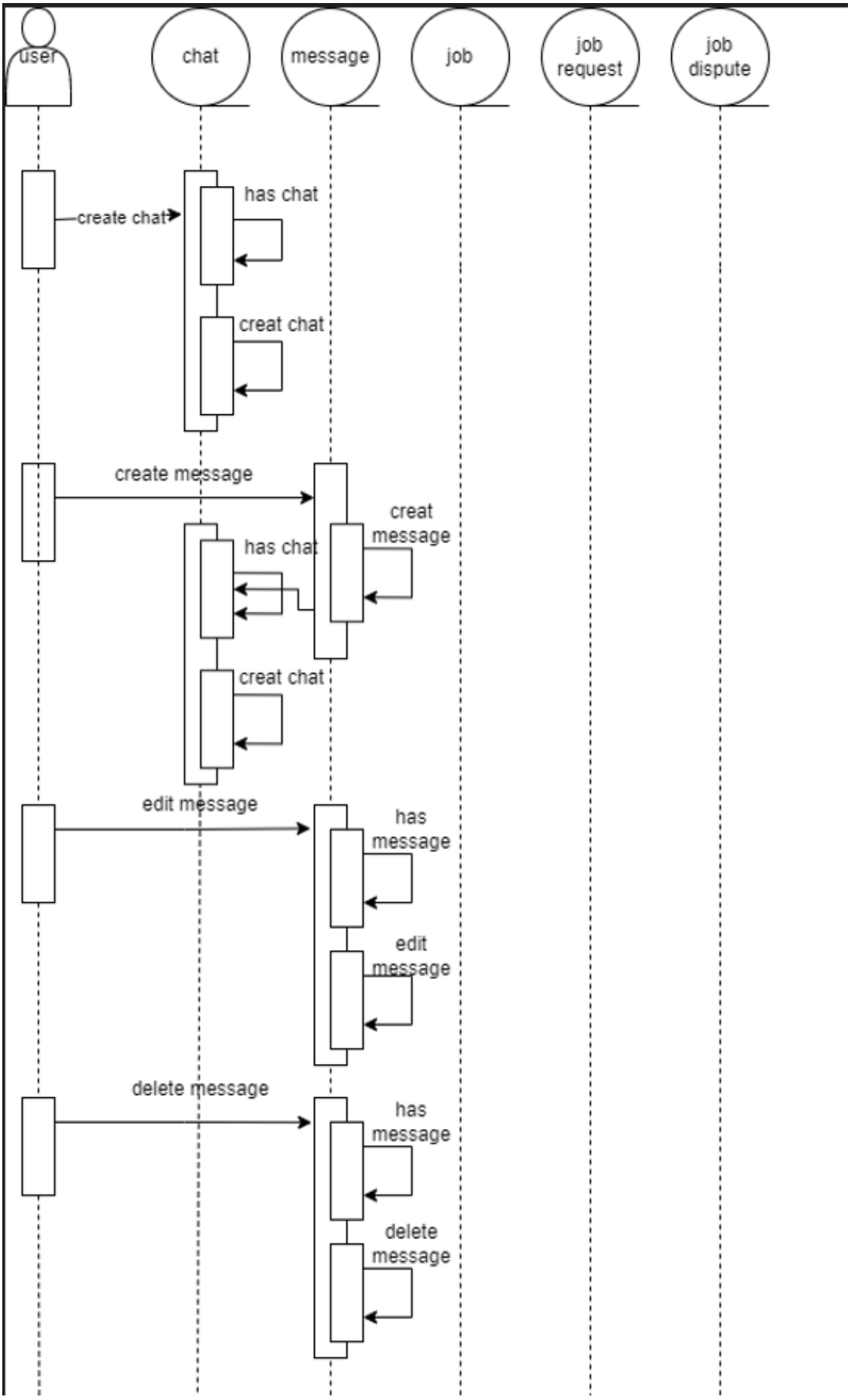

Рис. 2.8. Фрагмент 1 діаграми послідовностей

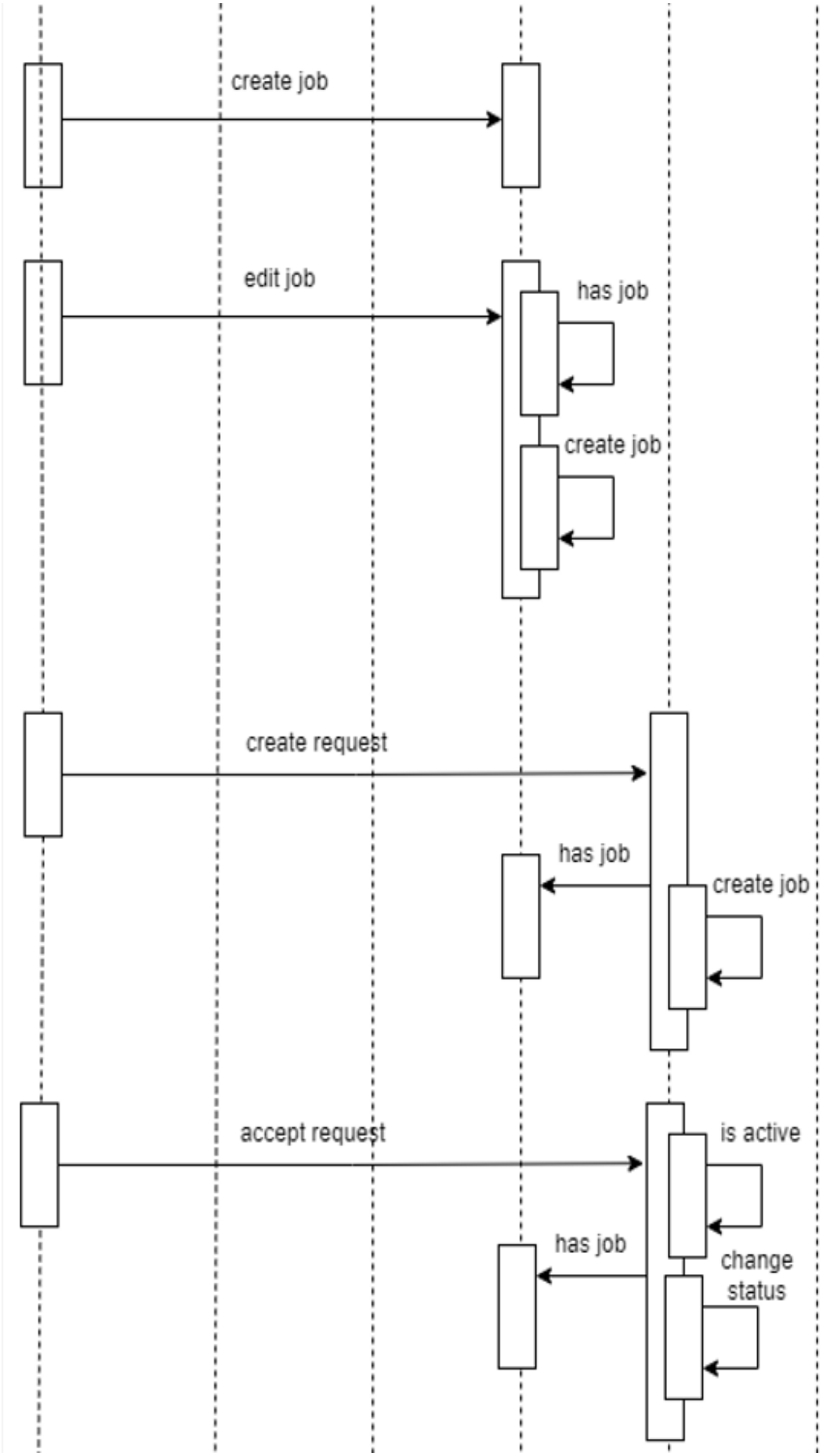

Рис. 2.9. Фрагмент 2 діаграми послідовностей

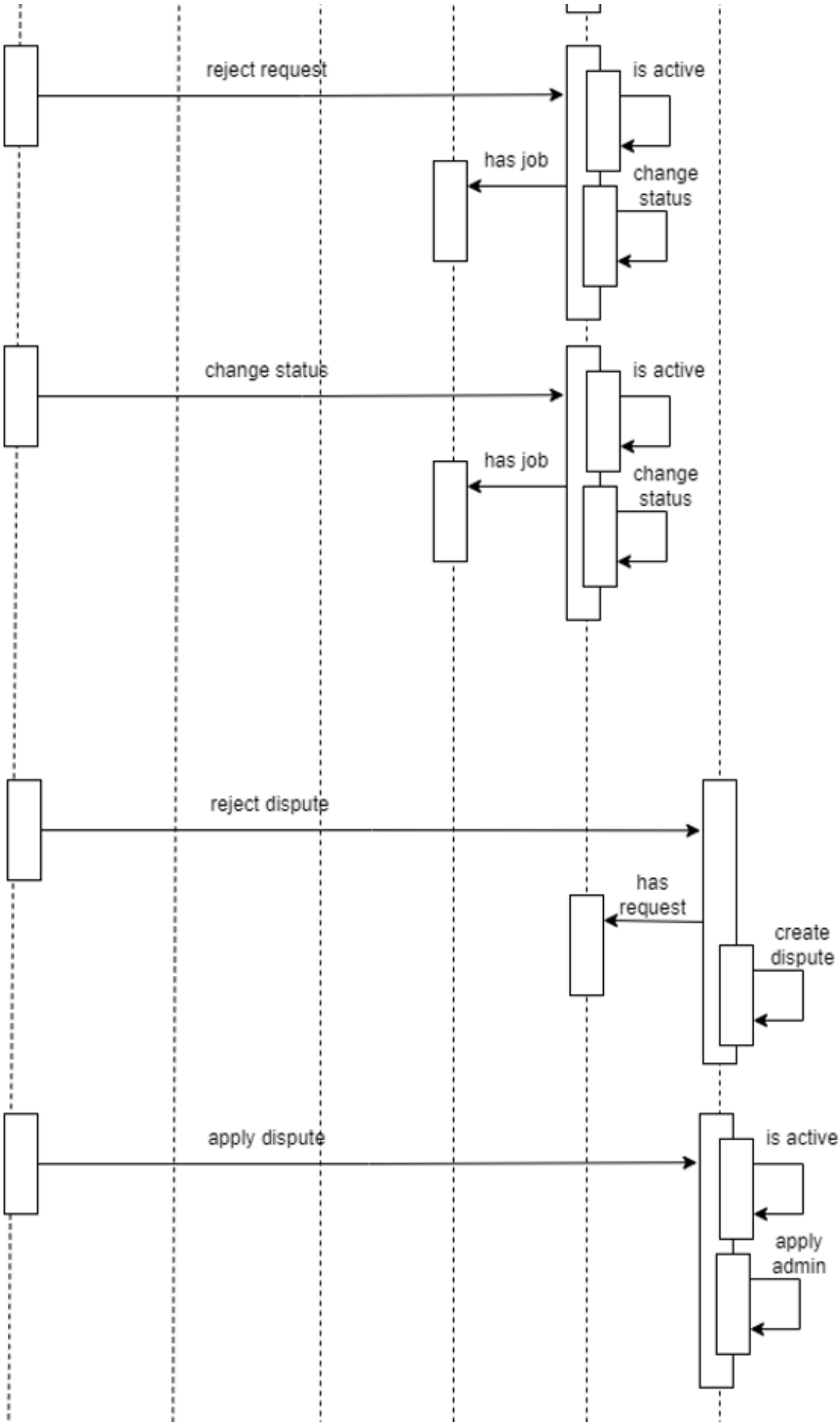

Рис. 2.10. Фрагмент 3 діаграми послідовностей

Рис. 2.11. Фрагмент 4 діаграми послідовностей

2.3. Комунікації і послідовність взаємодії об’єктів системи

Для спрощення розробки, у даному додатку буде використано наступні патерни: “Декоратор”, “MVC”, “Одинак”, “Фасад” та “Міст”.

Патерн MVC:

Серверну частину додатку побудовано на патерні “MVC”, де контролер повертає в якості view – дані(таке собі урізання). MVC (Model-View-Controller): Це архітектурний патерн, який використовується для розподілення відповідальностей у веб-додатках. Основні компоненти цього патерну включають:

Модель (Model): Представляє собою даний об'єкт або відповідає за доступ до даних та логіку бізнес-логіки.

Вид (View): Представляє інтерфейс користувача та відображає дані моделі користувачеві(в даному випадку повертаються просто дані).

Контролер (Controller): Обробляє вхідні дані від користувача, обновляє модель та відображення.

Патерн “Міст”:

Патерн “Міст” буде використано саме при розробці модулів для роботи з коментарями. Так як для продавця, виконавця та замовлення будуть доступні додавання та відповідання на коментарі, а також оцінювання цих сутностей, то лочіним буде той факт, що зберігання, відображення та буде працювати подібно, але зберігатиме різні поля, різні значення і приховану логіку буде мати різну.

Приблизний код для мосту наведено в Додатку В.

Патерн “Декоратор”:

Патерн “Декоратор” використовується доволі часто, на основі нього було розроблено унікальну систему для спрощення роботи з помилками, як у моделі, так і в контролері.

Наприклад в моделі, метод для відновлення помилок дозволяє з будь-якої точки методу закінчити його та повідомити точку, з якох було викликано метод про помилку.

Приблизний код декоратора виглядає наступним чином:

```
async errorWrapper(func) {
    try {
        return await func();
    } catch (err) {
        this.setError("Internal server error", 500);
    } finally {
        this.throwMainError();
    }
}
```

Приблизний код методу, що використовує цей декоратор:

```
getByJobId = async (jobId) =>
    await this.errorWrapper(async () => {
      const disputes = await this.dbQueryAsync(
        `SELECT ${this.__fullDisputeInfo} WHERE jobs.id = ?`,
        [jobId]
      );
      if (disputes.length > 0) return disputes[0];
      return null;
});
```

Метод errorWrapper виправляє помилки в межах свого виклику, обробляє їх та передає далі до точки виклику. Це гарний спосіб обробки помилок на рівні класу, забезпечуючи однорідність та управління помилками для всіх методів, що використовуються в межах класу Model.

Такий підхід забезпечує гнучкість та чистоту коду, оскільки логіка обробки помилок винесена в окремий метод, і вам не потрібно дублювати цю логіку у кожному методі окремо. Також це дозволяє легко впроваджувати зміни в обробку помилок в майбутньому, якщо це буде потрібно.

Наступний приклад реалізації даного патерну це метод контролера – errorWrapper.

Лістинг коду обгортки наведено в Додатку Г.

Лістинг коду, що використовує обгортку наведено в Додатку Ґ.

Як можна побачити, у методі використано метод setResponseBaseSuccess, цей метод дозволяє визначити, що задачу було виконано успішно, статус її 200 і відповідь клієнту буде успішною. errorWrapper слугує методом, що перехоплює помилки і на основі них повертає клієнту повідомлення помилки і статус помилки. Крім цього, дана логіка дозволяє дуже спростити логування та інформування адміністраторів про помилки.

Патерн “Одинак”:

Патерн Одинак гарантує, що клас матиме лише один екземпляр і надасть глобальний доступ до цього екземпляра в межах додатку.

Лістинг коду наведено в Додатку Д.

У класі Controller додається статичне поле instance, яке вказує на єдиний екземпляр класу. Далі в конструкторі класу Controller використовується перевірка, яка перевіряє, чи існує вже екземпляр класу. Якщо екземпляр вже існує, конструктор повертає його, інакше створює новий екземпляр.

Патерн “Фасад”:

Патерн "Фасад" використовується для надання простого інтерфейсу до складної системи, щоб спростити її використання. У даному випадку методи getAllPending, getAllInProgress і getAllResolved можна розглядати як фасад, який надає зручний інтерфейс для отримання списку сутностей з різним статусом.

Лістинг коду наведено в додатку Е.

Висновок до другого розділу:

У цьому розділі було представлено структуру та взаємодію об'єктів у системі. Зазначено основні класи, їх атрибути та методи, відображено об'єктно-орієнтовану модель системи. Докладно розглянуті патерни проектування, такі як "MVC", "Міст", "Декоратор", "Одинак" та "Фасад", що використовуються для створення структурованого та ефективного коду. Висвітлено роль кожного класу та методу в системі, що сприяє зрозумілій архітектурі додатку та його подальшому розвитку.

РОЗДІЛ 3. ФІЗИНА МОДЕЛЬ ТА ПРОТОТИП ПРОГРАМНОГО КОМПЛЕКСУ

3.1. Взаємодія компонентів системи

Загальна інформація про проект:

Серверна розробка виконується за допомогою Node.js. Версія 18.17.1. Весь процес сервера починається з файлу index.js де відбувається розвертання програми. Далі все залежить від запитів, що приходять на сервер(типу Http чи сокети, на основі яких сервер зберігає зміни, або віддає дані).

Сервер використовує реляційну Mysql-базу даних. Версія - 8.0.30.

Клієнтська частина розробляється за допомогою React. Версія - 18.2.0. Увесь процес клієнта починається з файлу App.jsx де відбувається розгортання головного компоненту, а далі, залежно від різних умов(маршрутів, виконаних клієнтом дій) – генеруються дочірні компоненти

Бібліотеки фреймворки та драйвера, що використовує сервер:

- Express: Фреймворк для створення сервера.
- Bcrypt: Використовується для шифрування паролів.
- Body-parser: Допоміжна бібліотека для обробки даних запиту.
- Cors: Дозволяє обробляти запити з різних джерел.
- Db-migrate та db-migrate-mysql: Використовуються для керування міграціями бази даних.
- Dotenv: Допоміжна бібліотека для завантаження змінних середовища з файлу.
- Jsonwebtoken: Використовується для генерації та перевірки JWT-токенів.
- Multer: Middleware для обробки файлових завантажень.
- Mysql: Драйвер для роботи з MySQL базою даних.

- Nodemon: Допоміжна утиліта для автоматичного перезапуску сервера при змінах у коді.

- Socket.io: Бібліотека для роботи з WebSocket для реалізації двостороннього зв'язку між сервером та клієнтом.

Бібліотеки, що використовує клієнтська частина:

- @react-google-maps/api: Бібліотека для взаємодії з Google Maps API.

- Axios:Бібліотека для здійснення HTTP-запитів.

- Chart.js та react-chartjs-2: Використовуються для створення графіків та діаграм на клієнті.

- React-bootstrap-icons: Використовується для використання Bootstrap іконок в React.

- React-contenteditable: Компонент React для редагування вмісту з HTML.

- React-geocode: Використовується для геокодування (перетворення адреси на координати та навпаки).

- React-router-dom: Для навігації між різними сторінками вашого React додатку.

- React-star-ratings: Компонент для відображення та редагування рейтингів.

- Socket.io-client: Клієнтська частина для взаємодії з Socket.io сервером.

- Wavesurfer.js: Бібліотека для візуалізації аудіоданих.

Топологія та технологія комунікацій:

HTTP (Axios):

Точки доступу та типи запитів:

На сервері визначені різні маршрути, такі як /register, /login, /reset-password, /get-job/:id та інші.

Використовуються різні HTTP-методи, такі як POST та GET, залежно від призначення маршруту.

Взаємодія сервера та клієнта за допомогою Axios:

Axios використовується для здійснення HTTP-запитів на сервер.

Наприклад, виклик API-запиту на реєстрацію: axios.post("/register", userData).

Дані що передаються: може бути як json, так і formdata залежно від умов використання.

WebSocket:

Використані бібліотеки:

Клієнтська сторона використовує socket.io-client.

Серверна частина використовує socket.io.

Використання сокетів для реального часу:

Сокети використовуються для реалізації реального часу в чаті та інших функціях.

Передається різні події, такі як created-chat, success-sended-message, success-deleted-message та інші.

Є події для сповіщення про дії користувача, такі як писання (typing), виходити з писання (stop-typing), онлайн (online) та офлайн (offline).

Спільні аспекти:

WebSocket використовується для вирішення завдань, пов'язаних з реальним часом та чатом.

Axios використовується для взаємодії з сервером для реєстрації, входу, отримання роботи та інших статичних запитів.

Візуалізація інформація про компоненти зображена на рисунку 18.

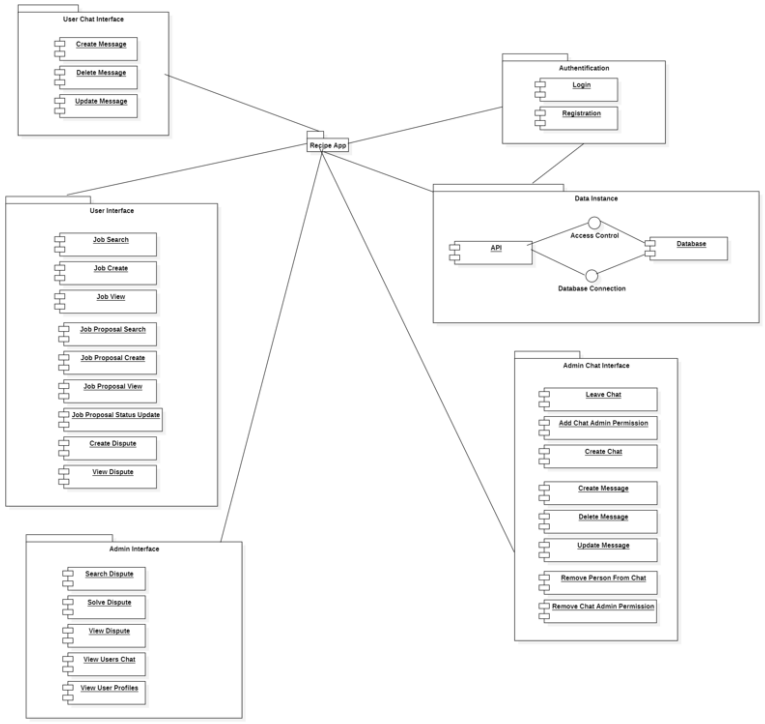

Рис. 3.1. Діаграма компонентів

3.2. Архітектура програмного комплексу та його розгортання

Є два вузли розгортання – серверний та клієнтський. Серверний, передбачає, незалежність від клієнтсьокого з перспективою розширити клієнтські додатки.

Серверний вузол (Node.js):

Версія операційної системи, на якій розгортаний серверний вузол: Windows 10

Node.js: 18.17.1

Посилання сервера: http://localhost:5000

База даних MySQL та її конфігурація:

Хост бази даних: localhost

Ім'я користувача: root

Пароль для підключення: відсутній

База даних: dyplome

Клієнтський вузол (React):

Операційна система, на якій розгортаний клієнтський вузо: Windows 10, Linux, Max

Посилання клієнта: http://localhost:3000

Браузери, які підтримуються: Google Chrome, Mozilla Firefox, Safari.

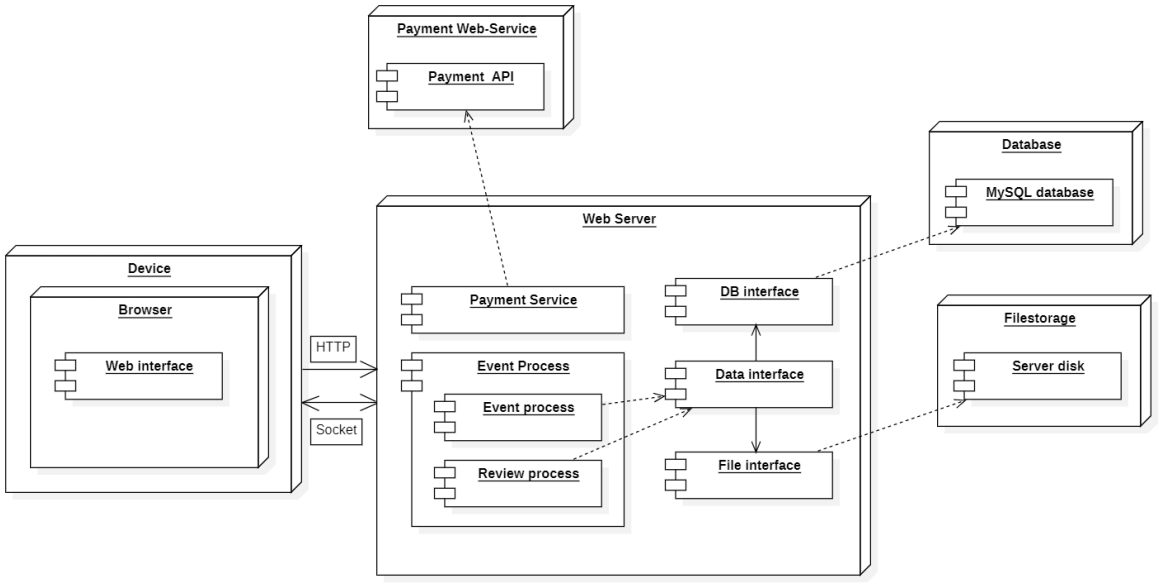

Рис. 3.2. Діаграма розгортання

3.3. Генерування програмного коду для прототипу програмного комплексу

Використовуючи раніше створену діаграму класів, можна автоматизувати генерацію коду для створення файлів класів запропонованої системи. Важливо відзначити, що кодогенерація - це процес автоматичного перетворення діаграм UML у програмний код.

Для здійснення генерації програмного коду потрібно мати діаграму класів та діаграму компонентів, де компоненти вже мають визначену мову програмного коду. Це необхідно, оскільки послідовність кроків генерації коду залежить від обраної мови реалізації.

Перед розпочатком генерації коду слід адаптувати всі елементи під обрану мову програмування, змінивши, наприклад, типи даних на відповідні для обраної мови програмування. У нашому випадку ми використовували мову програмування PHP(програму буде розроблено на js, на жаль, додаток не дає можливості генерації коду на цій мові).

Лістинг згенерованих класів наведено у Додатку Є.

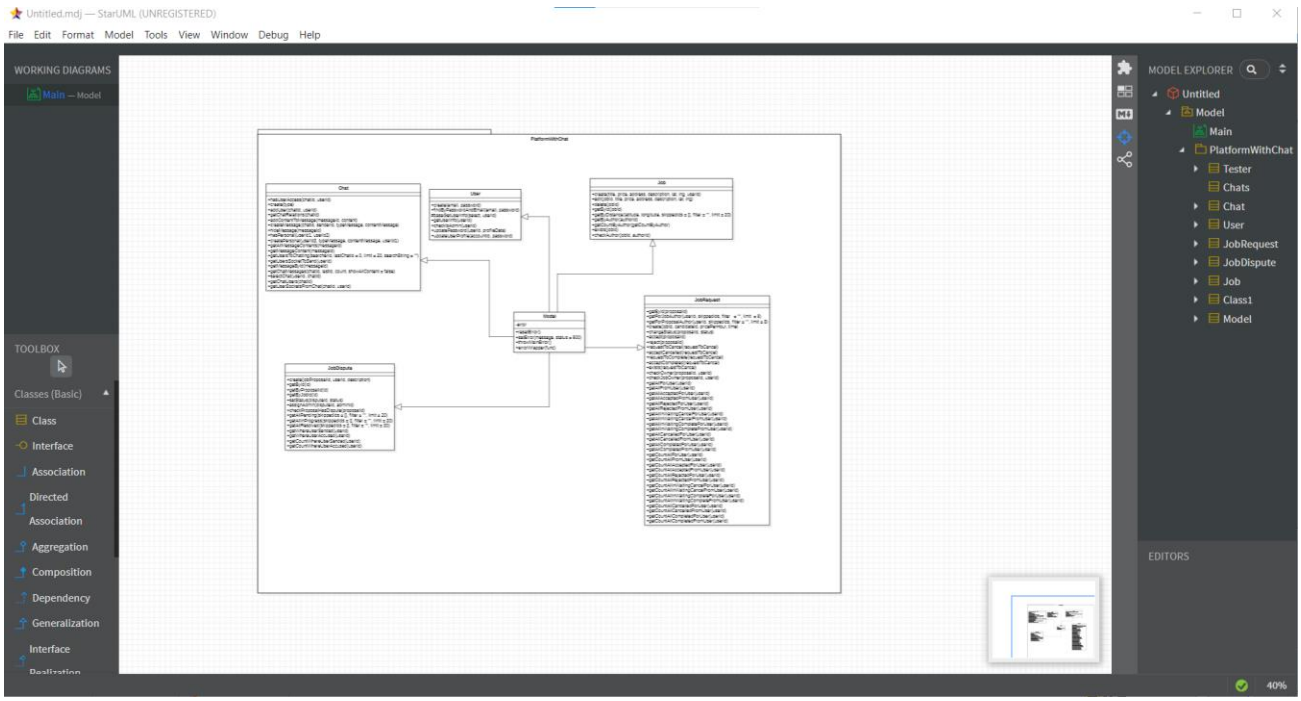

Рис. 3.3. Робочий простір програми

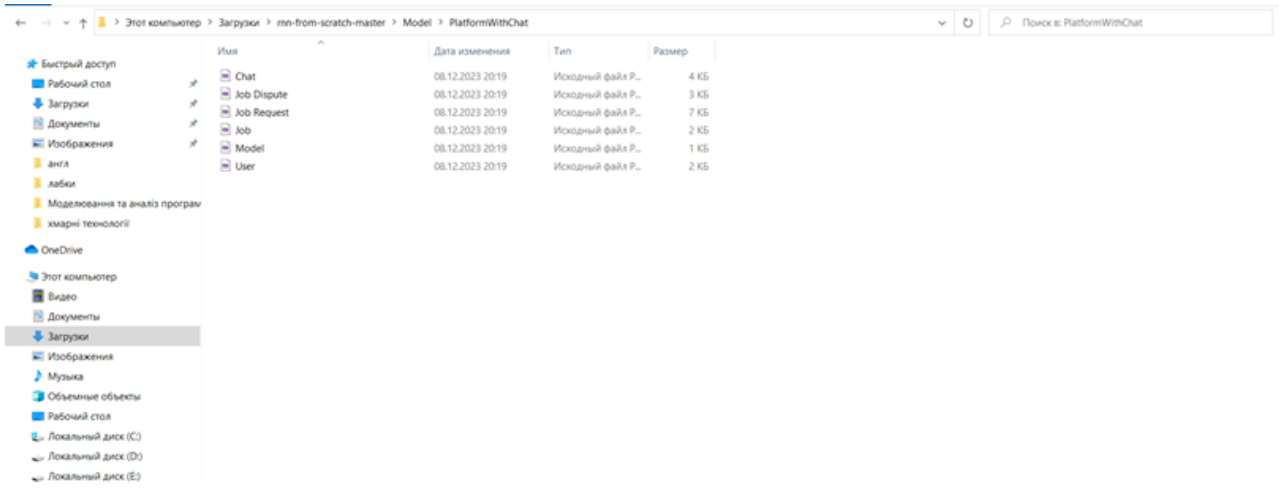

Рис. 3.4. Результат генерації коду

Висновки до третього розділу:

У данному розділі розглянута детальна інформація про взаємодію компонентів системи, в якій використовуються технології Node.js, React, та MySQL для серверної та клієнтської частин. Зазначено використані бібліотеки та фреймворки, такі як Express, Axios, Socket.io, що сприяють ефективній роботі системи. Детально описано взаємодію за допомогою протоколів HTTP та WebSocket. Діаграми компонентів та розгортання ілюструють структуру системи, а розділ про генерацію коду підкреслює автоматизацію процесу розробки. Загалом, система використовує сучасні технології, забезпечуючи ефективність та гнучкість у взаємодії та розширенні. Крім цього було виконано кодогенерацію прототипу серверних класів на мові PHP з можливістю адаптації під js.

ВИСНОВКИ

У результаті виконання даної роботи було вивчено та проаналізовано ключові аспекти створення інноваційної онлайн-платформи для для виконання завдань за нагороду. Перший розділ визначив вимоги до програмного продукту, покладаючи акцент на функціональність та нефункціональні аспекти, такі як безпека та стабільність. Другий розділ висвітлив архітектурні аспекти системи, включаючи структуру класів та використання патернів проектування. У третьому розділі було детально розглянуто взаємодію компонентів системи, використані технології та бібліотеки, а також проведено кодогенерацію.

Під час виконання роботи, було визначено, що система буде мати сучасний технічний стек, включаючи Node.js, React та MySQL, що забезпечує її ефективність та гнучкість. Структура системи визначена з огляду на принципи об'єктно-орієнтованого програмування та використання патернів проектування. Детально розглянуті взаємодія компонентів через протоколи HTTP та WebSocket підтримують реалізацію реального часу. Кодогенерація прототипу серверних класів спрощує процес розробки та адаптації системи.

Усі ці аспекти свідчать про високий рівень технічної та функціональної готовності системи, що робить її потенційно ефективним інструментом для виконання головної задачі будь-якої платформи – заробітку коштів.

СПИСОК ЛІТЕРАТУРИ

1. Загальна інформація про моделювання програмного забезпечення. [Електронне джерело] - Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Simulation_software.

2. Загальна інформація про розробку програмного забезпечення. [Електронне джерело] - Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Software_development.

3. Діаграма класів. [Електронне джерело] - Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Class_diagram.

4. Генерація коду. [Електронне джерело] - Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Code_generation_(compiler).

5. Діаграма розгортання. [Електронне джерело] - Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Deployment_diagram.

6. Загальна інформація про Draw.io. [Електронне джерело] - Режим доступу до ресурсу: https://en.wikipedia.org/wiki/StarUML

7. Загальна інформація про StarUML. [Електронне джерело] - Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Diagrams.net

8. Документакція до редактора діаграм StarUML. [Електронне джерело] - Режим доступу до ресурсу: https://docs.staruml.io/user-guide/diagram-editor.

9. Документація до діаграм класів StarUML. [Електронне джерело] - Режим доступу до ресурсу: https://docs.staruml.io/working-with-uml-diagrams/class-diagram.

10. Документація до діаграм пакетів в StarUML. [Електронне джерело] - Режим доступу до ресурсу: https://docs.staruml.io/working-with-uml-diagrams/package-diagram.

11. Документація до діаграм об’єктів в StarUML. [Електронне джерело] - Режим доступу до ресурсу: https://docs.staruml.io/working-with-uml-diagrams/object-diagram.

12. Документація до діаграм компонентів в StarUML. [Електронне джерело] - Режим доступу до ресурсу: https://docs.staruml.io/working-with-uml-diagrams/component-diagram.

13. Документація до діаграм деплойменту в StarUML. [Електронне джерело] - Режим доступу до ресурсу: https://docs.staruml.io/working-with-uml-diagrams/deployment-diagram.

14. Документація до діаграм юз-кейсів в StarUML. [Електронне джерело] - Режим доступу до ресурсу: https://docs.staruml.io/working-with-uml-diagrams/use-case-diagram.

15. Документація до встановлення розширення в StarUML. [Електронне джерело] - Режим доступу до ресурсу: https://docs.staruml.io/user-guide/managing-extensions#install-extension.

16. Кодогенерація діаграми до php. [Електронне джерело] - Режим доступу до ресурсу: https://www.robinglauser.ch/blog/2019/03/20/creating-a-uml-to-php-generator-for-staruml/.

17. Приклад діаграм в draw.io. [Електронне джерело] - Режим доступу до ресурсу: https://www.drawio.com/example-diagrams.

18. Документація до draw.io. [Електронне джерело] - Режим доступу до ресурсу:https://www.drawio.com/doc/#get-started-with-diagramsnet.

19. Інструкція побудови UML-діаграм. [Електронне джерело] - Режим доступу до ресурсу: https://dou.ua/forums/topic/40575/.

20. Документація до мови php. [Електронне джерело] - Режим доступу до ресурсу: https://www.php.net/manual/en/index.php.

21. Документація до node.js. [Електронне джерело] - Режим доступу до ресурсу: https://nodejsdev.ru/guides/webdraftt/.

22. Документація до Java Script. [Електронне джерело] - Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/JavaScript.

23. Документація до HTML. [Електронне джерело] - Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/HTML.

24. Документація до CSS. [Електронне джерело] - Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/CSS/Reference.
25. Документація до React. [Електронне джерело] - Режим доступу до ресурсу: https://uk.legacy.reactjs.org/docs/getting-started.html.

ДОДАТКИ

Функціональні вимоги:

- Авторизація та реєстрація користувачів: Можливість реєстрації користувачів з обов'язковими полями, такими як ім'я, електронна пошта, пароль. Авторизація зареєстрованих користувачів з використанням електронної пошти та пароля.

- Створення та редагування профілю: Можливість користувачів створювати та редагувати свої профілі, включаючи фотографію, контактну інформацію та навички.

- Створення та перегляд замовлень: Можливість роботодавців створювати замовлення для виконавців з описом завдань, бюджетом і терміном виконання. Виконавці можуть переглядати список доступних замовлень.

- Чат для спілкування: Вбудований чат для взаємодії роботодавців та виконавців щодо уточнень, деталей та обговорення завдань.

- Оцінювання та відгуки: Можливість користувачів залишати відгуки та оцінки після виконання завдань.

- Система спорів: Можливість розглядати спори та конфлікти між роботодавцями та виконавцями та приймати рішення.

Нефункціональні вимоги:

Вимоги до продуктивності:

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

Підтримка одночасних користувачів: Система повинна підтримувати мінімум 40 одночасно працюючих користувачів, які взаємодіють із загальною базою даних.

Можливості експлуатації:

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

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

Додаток Б

Глосарій:

Ціль Глосарію:

Словник містить опис термінів, які використовуються при проектуванні додатку-лотереї для розіграшу товарів. Визначаються основні поняття, безпосередньо пов'язані з плануванням і диспетчеризація замовлень.

Контекст Глосарію:

Словник створений в рамках платформи.

Поняття, що використовуються при плануванні та при описі вихідної інформації

- Функціональна платформа - високофункціональна онлайн-платформа із вбудованим чатом для виконання різноманітних завдань та послуг у режимі реального часу.

- Зручне середовище для користувачів - інтуїтивно зрозумілий інтерфейс, спрямований на спрощення використання платформи та забезпечення позитивного користувацького досвіду.

- Безпека і конфіденційність - гарантія високого рівня безпеки для всіх користувачів, включаючи перевірку ідентифікації та захист від шахрайства. Ведення статистики успішних виконань замовлень та звертань до адміністраторів для вирішення спорів.

- Монетизація та доходи - розробка стратегії монетизації, включаючи комісії за операції, платні функції та інші джерела доходу.

- Залучення та утримання користувачів - механізми для привертання нових та утримання постійних користувачів на платформі.

- Географічна підв'язка - забезпечення можливості користувачам знаходити оптимальні точки для виконання завдань і отримання прибутку.

- Онлайн-платформа - інтерактивна система, доступна через Інтернет, що надає користувачам можливість взаємодії та виконання різних завдань.

- Вбудований чат - засіб для спілкування користувачів у режимі реального часу, вбудований безпосередньо в платформу.

- Система рейтингів і відгуків - механізм для оцінки та залишення відгуків, спрямований на забезпечення довіри між користувачами.

Додаток В

```
// Абстракція "Коментар"
class Comment {
    constructor(storage) {
      this.storage = storage;
    }

    async create(commentData) {
      return await this.storage.create(commentData);
    }

    async getById(commentId) {
      return await this.storage.getById(commentId);
    }
  }

  // Реалізація "Коментар" для виконавця
  class ExecutorCommentStorage extends Model{
    async create(commentData) {
      const insertRes = await this.db.query(
        "INSERT INTO executor_comments (author_id, executor_id, description,
rating) VALUES (?, ?, ?, ?)",
       [commentData.authorId, commentData.executorId, commentData.description,
commentData.rating]
      );

      const commentId = insertRes.insertId;
      const comment = await this.getById(commentId);
      return comment;
    }
```

```
    async getById(commentId) {
      const comment = await this.db.query(
        "SELECT * FROM executor_comments WHERE id = ?",
        [commentId]
      );

      if (comment.length > 0) return comment[0];
      return null;
    }
  }

  // Реалізація "Коментар" для продавця
  class SellerCommentStorage extends Model {
    async create(commentData) {
      const insertRes = await this.db.query(
        "INSERT INTO seller_comments (author_id, seller_id, description, rating)
VALUES (?, ?, ?, ?)",
        [commentData.authorId, commentData.sellerId, commentData.description,
commentData.rating]
      );

      const commentId = insertRes.insertId;
      const comment = await this.getById(commentId);
      return comment;
    }

    async getById(commentId) {
      const comment = await this.db.query(
        "SELECT * FROM seller_comments WHERE id = ?",
        [commentId]
      );

      if (comment.length > 0) return comment[0];
      return null;
    }
  }

  // Реалізація "Коментар" для пропозиції
  class ProposalCommentStorage extends Model{
    async create(commentData) {
      const insertRes = await this.db.query(
        "INSERT INTO proposal_comments (author_id, proposal_id, rating) VALUES
(?, ?, ?)",
        [commentData.authorId, commentData.proposalId, commentData.rating]
      );

      const commentId = insertRes.insertId;
      const comment = await this.getById(commentId);
      return comment;
    }
```

```
    async getById(commentId) {
      const comment = await this.db.query(
        "SELECT * FROM proposal_comments WHERE id = ?",
        [commentId]
      );

      if (comment.length > 0) return comment[0];
      return null;
    }
  }

  // Реалізація "Коментар" для відповіді
  class ReplyCommentStorage extends Model{
    async create(commentData) {
      const insertRes = await this.db.query(
        "INSERT INTO reply_comments (parent_comment_id, text, respondent_id,
comment_type) VALUES (?, ?, ?, ?)",
        [commentData.parentCommentId, commentData.text, commentData.respondentId,
commentData.commentType]
      );

      const commentId = insertRes.insertId;
      const comment = await this.getById(commentId);
      return comment;
    }

    async getById(commentId) {
      const comment = await this.db.query(
        "SELECT * FROM reply_comments WHERE id = ?",
        [commentId]
      );

      if (comment.length > 0) return comment[0];
      return null;
    }
  }



  // приймає об'єкт бази даних, який має метод query
  async function init(db){
  // Приклад використання
  const executorCommentStorage = new ExecutorCommentStorage(db);
  const sellerCommentStorage = new SellerCommentStorage(db);
  const proposalCommentStorage = new ProposalCommentStorage(db);
  const replyCommentStorage = new ReplyCommentStorage(db);

  const executorComment = new Comment(executorCommentStorage);
  const sellerComment = new Comment(sellerCommentStorage);
```

```
  const proposalComment = new Comment(proposalCommentStorage);
  const replyComment = new Comment(replyCommentStorage);

  // Створення коментарів різних типів
  console.log(await executorComment.create({ authorId: 1, executorId: 2,
description: 'Great job!', rating: 5 }));

  console.log(await sellerComment.create({ authorId: 1, sellerId: 3, description:
'Excellent service!', rating: 4 }));

  console.log(await proposalComment.create({ authorId: 1, proposalId: 4, rating:
3 }));

  console.log(await replyComment.create({ parentCommentId: 1, text: 'Thank you!',
respondentId: 2, commentType: 'executor' }));
  }
```

Додаток Г

```
async errorWrapper(res, func) {
    try {
      await func();
    } catch (e) {
      console.log(e);
      const status = e.status ? e.status : 500;
      const error = e.message;
      this.setResponse(
        {
          error,
        },
        status
      );
    } finally {
      this.sendResponse(res);
    }
  }
```

Додаток Ґ

```
getAllByStatus = async (req, res) =>
    this.errorWrapper(res, async () => {
      const skippedIds = req.body.skippedIds ? req.body.skippedIds : [];
      const needCountJobs = req.body.count ? req.body.count : 20;
      const filter = req.body.filter ?? "";
      const status = req.params.status ?? "pending";

      let getFunc = null;
      if (status == "pending") getFunc = this.disputeModel.getAllPending;
      if (status == "in-progress") getFunc = this.disputeModel.getAllInProgress;
```

```
      if (status == "resolved") getFunc = this.disputeModel.getAllResolved;

      if (!getFunc)
        return this.setResponseNoFoundError("Dispute status not found");

      const disputes = await getFunc(skippedIds, filter, needCountJobs);
      this.setResponseBaseSuccess("Disputes was get successfully", {
        disputes,
      });
    });
```

Додаток Д

```
class Controller {
  constructor(db) {
    if (!Controller.instance) {
      this.userModel = new UserModel(db);
      this.passwordResetLinkModel = new PasswordResetLinkModel(db);
      this.socketModel = new SocketModel(db);
      this.chatModel = new ChatModel(db);
      this.actionModel = new ActionModel(db);
      this.jobModel = new JobModel(db);
      this.jobProposalModel = new JobProposalModel(db);
      this.disputeModel = new DisputeModel(db);

      Controller.instance = this;
    }

    return Controller.instance;
  }
}

Controller.instance = null;

module.exports = Controller;
```

Додаток Е

```
__getAllByStatus = async (status, skippedIds = [], filter = "", limit = 20) =>
    await this.errorWrapper(async () => {
      const params = [];

      let query = `SELECT cu1.chat_id, ${this.__fullDisputeInfo}

      JOIN chats_users as cu1 ON job_requests.user_id = cu1.user_id
      JOIN chats_users as cu2 ON (cu2.chat_id = cu2.chat_id AND jobs.author_id =
cu2.user_id)

      WHERE disputes.status = '${status}'`;
```

```
      const skipIdsRequest = "disputes.id NOT IN (?)";
      const filterRequest = `(jobs.title like "%${filter}%")`;

      if (skippedIds.length > 0 && filter && filter.length > 0) {
        query += ` AND ${skipIdsRequest} AND ${filterRequest}`;
        params.push(skippedIds);
      } else {
        if (skippedIds.length > 0) {
          query += ` AND ${skipIdsRequest}`;
          params.push(skippedIds);
        }

        if (filter && filter.length > 0) {
          query += ` AND ${filterRequest}`;
        }
      }

      query += ` ORDER BY created_at ASC LIMIT ?`;
      params.push(limit);
      const distributors = await this.dbQueryAsync(query, params);
      return distributors;
    });

  getAllPending = (skippedIds = [], filter = "", limit = 20) =>
    this.__getAllByStatus("Pending", skippedIds, filter, limit);

  getAllInProgress = (skippedIds = [], filter = "", limit = 20) =>
    this.__getAllByStatus("In Progress", skippedIds, filter, limit);

  getAllResolved = (skippedIds = [], filter = "", limit = 20) =>
    this.__getAllByStatus("Resolved", skippedIds, filter, limit);
```

Додаток Є

Лістинги коду класу Model:

```
<?php

declare(strict_types=1);

namespace PlatformWithChat;

class Model
{

    private  $error;

    /**
```

```
     * Default constructor
     */
    public function __construct()
    {
        // ...
    }

    /**
     *
     */
    public function resetError()
    {
        // TODO implement here
    }

    /**
     * @param  $message
     * @param  $status
     */
    public function setError( $message,  $status = 500)
    {
        // TODO implement here
    }

    /**
     *
     */
    public function throwMainError()
    {
        // TODO implement here
    }

    /**
     * @param  $func
     */
    public function errorWrapper( $func)
    {
        // TODO implement here
    }

}
```

Лістинги коду класу User:

```
<?php

declare(strict_types=1);

namespace PlatformWithChat;
```

```php
class User
{

    /**
     * Default constructor
     */
    public function __construct()
    {
        // ...
    }

    /**
     * @param  $email
     * @param  $password
     */
    public function create( $email,  $password)
    {
        // TODO implement here
    }

    /**
     * @param  $email
     * @param  $password
     */
    public function findByPasswordAndEmail( $email,  $password)
    {
        // TODO implement here
    }

    /**
     * @param  $select
     * @param  $userId
     */
    protected function baseGetUserInfo( $select,  $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getUserInfo( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
```

```
    public function checkIsAdmin( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     * @param  $profileData
     */
    public function updatePassword( $userId,  $profileData)
    {
        // TODO implement here
    }

    /**
     * @param  $accountId
     * @param  $password
     */
    public function updateUserProfile( $accountId,  $password)
    {
        // TODO implement here
    }

}
```

Лістинги коду класу Chat:

```
<?php

declare(strict_types=1);

namespace PlatformWithChat;

class Chat extends Model
{

    /**
     * Default constructor
     */
    public function __construct()
    {
        // ...
    }

    /**
     * @param  $chatId
     * @param  $userId
     */
    public function hasUserAccess( $chatId,  $userId)
```

```
    {
        // TODO implement here
    }

    /**
     * @param  $type
     */
    public function create( $type)
    {
        // TODO implement here
    }

    /**
     * @param  $chatId
     * @param  $userId
     */
    public function addUser( $chatId,  $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $chatId
     */
    public function getChatRelations( $chatId)
    {
        // TODO implement here
    }

    /**
     * @param  $messageId
     * @param  $content
     */
    public function addContentToMessage( $messageId,  $content)
    {
        // TODO implement here
    }

    /**
     * @param  $chatId
     * @param  $senderId
     * @param  $typeMessage
     * @param  $contentMessage
     */
    public function createMessage( $chatId,  $senderId,  $typeMessage,
$contentMessage)
    {
        // TODO implement here
    }
```

```
    /**
     * @param  $messageId
     */
    public function hideMessage( $messageId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId1
     * @param  $userId2
     */
    public function hasPersonal( $userId1,  $userId2)
    {
        // TODO implement here
    }

    /**
     * @param  $userId2
     * @param  $typeMessage
     * @param  $contentMessage
     * @param  $userId1
     */
    public function createPersonal( $userId2,  $typeMessage,  $contentMessage,
$userId1)
    {
        // TODO implement here
    }

    /**
     * @param  $messageId
     */
    public function getAllMessageContents( $messageId)
    {
        // TODO implement here
    }

    /**
     * @param  $messageId
     */
    public function getMessageContent( $messageId)
    {
        // TODO implement here
    }

    /**
     * @param  $searcherId
     * @param  $lastChatId
     * @param  $limit
     * @param  $searchString
```

```
     */
    public function getUsersToChatting( $searcherId,  $lastChatId = 0,  $limit =
20,  $searchString = "")
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getUsersSocketToSend( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $messageId
     */
    public function getMessageById( $messageId)
    {
        // TODO implement here
    }

    /**
     * @param  $chatId
     * @param  $lastId
     * @param  $count
     * @param  $showAllContent
     */
    public function getChatMessages( $chatId,  $lastId,  $count,  $showAllContent
= false)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     * @param  $chatId
     */
    public function selectChat( $userId,  $chatId)
    {
        // TODO implement here
    }

    /**
     * @param  $chatId
     */
    public function getChatUsers( $chatId)
    {
        // TODO implement here
```

```
    }

    /**
     * @param  $chatId
     * @param  $userId
     */
    public function getUserSocketsFromChat( $chatId,  $userId)
    {
        // TODO implement here
    }

}
```

Лістинги коду класу Job:

```
<?php

declare(strict_types=1);

namespace PlatformWithChat;

class Job
{

    /**
     * Default constructor
     */
    public function __construct()
    {
        // ...
    }

    /**
     * @param  $title
     * @param  $price
     * @param  $address
     * @param  $description
     * @param  $lat
     * @param  $lng
     * @param  $userId
     */
    public function create( $title,  $price,  $address,  $description,  $lat,
 $lng,  $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $jobId
     * @param  $title
```

```
     * @param  $price
     * @param  $address
     * @param  $description
     * @param  $lat
     * @param  $lng
     */
    public function edit( $jobId,  $title,  $price,  $address,  $description,
$lat,  $lng)
    {
        // TODO implement here
    }

    /**
     * @param  $jobId
     */
    public function delete( $jobId)
    {
        // TODO implement here
    }

    /**
     * @param  $jobId
     */
    public function getById( $jobId)
    {
        // TODO implement here
    }

    /**
     * @param  $latitude
     * @param  $longitude
     * @param  $skippedIds
     * @param  $filter
     * @param  $limit
     */
    public function getByDistance( $latitude,  $longitude,  $skippedIds = [],
$filter = "",  $limit = 20)
    {
        // TODO implement here
    }

    /**
     * @param  $authorId
     */
    public function getByAuthor( $authorId)
    {
        // TODO implement here
    }

    /**
```

```
     * @param  $getCountByAuthor
     */
    public function getCountByAuthor( $getCountByAuthor)
    {
        // TODO implement here
    }

    /**
     * @param  $jobId
     */
    public function exists( $jobId)
    {
        // TODO implement here
    }

    /**
     * @param  $jobId
     * @param  $authorId
     */
    public function checkAuthor( $jobId,  $authorId)
    {
        // TODO implement here
    }

}
```

Лістинги коду класу JobRequest:

```
<?php

declare(strict_types=1);

namespace PlatformWithChat;

class JobRequest
{

    /**
     * Default constructor
     */
    public function __construct()
    {
        // ...
    }

    /**
     * @param  $proposalId
     */
    public function getById( $proposalId)
    {
```

```
        // TODO implement here
    }

    /**
     * @param null $userId
     * @param null $skippedIds
     * @param null $filter
     * @param null $limit
     */
    public function getForJobAuthor(null $userId, null $skippedIds, null $filter
 = "", null $limit  = 8)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     * @param  $skippedIds
     * @param  $filter
     * @param  $limit
     */
    public function getForProposalAuthor( $userId,  $skippedIds,  $filter = "",
$limit = 8)
    {
        // TODO implement here
    }

    /**
     * @param  $jobId
     * @param  $candidateId
     * @param  $pricePerHour
     * @param  $time
     */
    public function create( $jobId,  $candidateId,  $pricePerHour,  $time)
    {
        // TODO implement here
    }

    /**
     * @param  $proposalId
     * @param  $status
     */
    public function changeStatus( $proposalId,  $status)
    {
        // TODO implement here
    }

    /**
     * @param  $proposalId
     */
```

```
    public function accept( $proposalId)
    {
        // TODO implement here
    }

    /**
     * @param  $proposalId
     */
    public function reject( $proposalId)
    {
        // TODO implement here
    }

    /**
     * @param  $requestToCancel
     */
    public function requestToCancel( $requestToCancel)
    {
        // TODO implement here
    }

    /**
     * @param  $requestToCancel
     */
    public function acceptCancelled( $requestToCancel)
    {
        // TODO implement here
    }

    /**
     * @param  $requestToCancel
     */
    public function requestToComplete( $requestToCancel)
    {
        // TODO implement here
    }

    /**
     * @param  $requestToCancel
     */
    public function acceptCompleted( $requestToCancel)
    {
        // TODO implement here
    }

    /**
     * @param  $requestToCancel
     */
    public function exists( $requestToCancel)
    {
```

```
        // TODO implement here
    }

    /**
     * @param  $proposalId
     * @param  $userId
     */
    public function checkOwner( $proposalId,  $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $proposalId
     * @param  $userId
     */
    public function checkJobOwner( $proposalId,  $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getAllForUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getAllFromUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getAllAcceptedForUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getAllAcceptedFromUser( $userId)
    {
```

```
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getAllRejectedForUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getAllRejectedFromUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getAllInWaitingCancelForUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getAllInWaitingCancelFromUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getAllInWaitingCompleteForUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getAllInWaitingCompleteFromUser( $userId)
    {
        // TODO implement here
    }
```

```
    /**
     * @param  $userId
     */
    public function getAllCancelledForUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getAllCancelledFromUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getAllCompletedForUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getAllCompletedFromUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getCountAllForUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getCountAllFromUser( $userId)
    {
        // TODO implement here
    }

    /**
```

```
     * @param  $userId
     */
    public function getCountAllAcceptedForUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getCountAllAcceptedFromUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getCountAllRejectedForUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getCountAllRejectedFromUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getCountAllInWaitingCancelForUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getCountAllInWaitingCancelFromUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
```

```
    public function getCountAllInWaitingCompleteForUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getCountAllInWaitingCompleteFromUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getCountAllCancelledForUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getCountAllCancelledFromUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getCountAllCompletedForUser( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getCountAllCompletedFromUser( $userId)
    {
        // TODO implement here
    }

}
```

Лістинги коду класу JobDispute:

```php
<?php

declare(strict_types=1);

namespace PlatformWithChat;

class JobDispute
{

    /**
     * Default constructor
     */
    public function __construct()
    {
        // ...
    }

    /**
     * @param  $jobProposalId
     * @param  $userId
     * @param  $description
     */
    public function create( $jobProposalId,  $userId,  $description)
    {
        // TODO implement here
    }

    /**
     * @param  $id
     */
    public function getById( $id)
    {
        // TODO implement here
    }

    /**
     * @param  $id
     */
    public function getByProposalId( $id)
    {
        // TODO implement here
    }

    /**
     * @param  $id
     */
    public function getByJobId( $id)
    {
```

```
        // TODO implement here
    }

    /**
     * @param  $disputeId
     * @param  $status
     */
    public function setStatus( $disputeId,  $status)
    {
        // TODO implement here
    }

    /**
     * @param  $disputeId
     * @param  $adminId
     */
    public function assignAdmin( $disputeId,  $adminId)
    {
        // TODO implement here
    }

    /**
     * @param  $proposalId
     */
    public function checkProposalHasDispute( $proposalId)
    {
        // TODO implement here
    }

    /**
     * @param  $skippedIds
     * @param  $filter
     * @param  $limit
     */
    public function getAllPending( $skippedIds = [],  $filter = "",  $limit = 20)
    {
        // TODO implement here
    }

    /**
     * @param  $skippedIds
     * @param  $filter
     * @param  $limit
     */
    public function getAllInProgress( $skippedIds = [],  $filter = "",  $limit =
20)
    {
        // TODO implement here
    }
```

```
    /**
     * @param  $skippedIds
     * @param  $filter
     * @param  $limit
     */
    public function getAllResolved( $skippedIds = [],  $filter = "",  $limit =
20)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getWhereUserSended( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getWhereUserAccused( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getCountWhereUserSended( $userId)
    {
        // TODO implement here
    }

    /**
     * @param  $userId
     */
    public function getCountWhereUserAccused( $userId)
    {
        // TODO implement here
    }

}
```