МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри_____ Є. О. Давиденко

підпис

«___» _____2024р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Вебзастосунок взаємодії фрілансерів та замовників

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 409.22010920

Здобувач _____А. Ю. Самокиш

підпис

«__» _____ 2024 р.

Керівник канд. техн. наук, доцент _____Г. В. Горбань

підпис

«__» _____ 2024 р.

Консультант канд. техн. наук, доцент _____ А. О. Алексєєва

підпис

«__» _____ 2024 р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ

Зав. кафедри _____ Є. О. Давиденко

«_____» _____ 2024 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи бакалавра

Видано здобувачу групи 409 факультету комп'ютерних наук

_____Самокиш Артем Юрійович_____

(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи

Вебзастосунок взаємодії фрілансерів та замовників

Затверджена наказом по ЧНУ від «22» _____грудня_____ 2023 р. № ___269_____

2. Строк представлення кваліфікаційної роботи «____» _____ 20__ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

Очікуваним результатом є вебзастосунок взаємодії фрілансерів та замовників

4. Перелік питань, що підлягають розробці:

- дослідження предметної області та аналіз існуючих аналогів;
- формування специфікації вимог до програмного забезпечення;
- визначення архітектури для проєктування програмного забезпечення;
- моделювання та проєктування програмного забезпечення;
- розробка програмного забезпечення;
- здійснення тестування роботи програмного забезпечення;

5. Перелік графічних матеріалів

Презентація

6. Завдання до спеціальної частини

__Аналіз охорони праці на робочих місцях фахівців з інформаційних технологій__

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Алексєєва А. О.	Кафедра екології	Спеціальна частина охорони праці

Керівник роботи _____канд. техн. наук, доцент Горбань Г. В._____

(посада, прізвище, ім'я, по батькові)

_____

(підпис)

Завдання прийнято до виконання

_____Самокиш А. Ю._____

(прізвище, ім'я, по батькові)

_____

(підпис)

Дата видачі завдання «____» _____ 2024р.

КАЛЕНДАРНИЙ ПЛАН
виконання бакалаврської кваліфікаційної роботи

Тема: «Вебзастосунок взаємодії фрілансерів та замовників»

№	Найменування роботи	Початок	Закінчення	Примітки
1	Розробка та затвердження завдання на виконання КРБ	15.01.2024	17.01.2024	Виконано
2	Огляд літератури за темою роботи	18.01.2024	29.01.2024	Виконано
3	Складання календарного плану КРБ	01.02.2024	19.02.2024	Виконано
4	Аналіз предметної області	20.02.2024	24.02.2024	Виконано
5	Розробка проєктних рішень	25.02.2024	01.03.2024	Виконано
6	Моделювання та конструювання	02.03.2024	10.03.2024	Виконано
7	Кодування ПЗ	11.03.2024	20.05.2024	Виконано
8	Тестування та апробація розробленого ПЗ	21.05.2024	26.05.2024	Виконано
9	Розробка керівництва користувача	27.05.2024	31.05.2024	Виконано
10	Розробка спеціальної частини з охорони праці	01.06.2024	02.06.2024	Виконано
11	Оформлення КРБ та презентації	03.06.2024	04.06.2024	Виконано
12	Відгук керівника КРБ	03.06.2024	03.06.2024	Виконано
13	Попередній захист	04.06.2024	04.06.2024	Виконано
14	Рецензування	13.06.2024	13.06.2024	Виконано
15	Завершення оформлення КРБ та презентації	15.06.2024	15.06.2024	Виконано
16	Захист кваліфікаційної роботи	26.06.2024	26.06.2024	Виконано

Розробив здобувач Самокиш А. Ю._____ _____

(прізвище, ім'я, по батькові) *(підпис)*

«24» січня_____2024 р.

Керівник роботи канд. техн. наук, доцент Горбань Г. В.

(посада, прізвище, ім'я, по батькові)

_____

(підпис)

«24»січня 2024 р.

АНОТАЦІЯ

до кваліфікаційної роботи бакалавра

«Вебзастосунок взаємодії фрілансерів та замовників»

Здобувач 409 гр.: Самокиш Артем Юрійович

Керівник: канд. техн. наук, доцент Горбань Г. В.

Кваліфікаційна робота присвячена розробці та впровадженню вебзастосунку, орієнтованого на спрощення процесу спільної роботи між фрілансерами та їх замовниками.

Актуальність теми полягає у необхідності оптимізації процесів пошуку, найму та співпраці між фрилансерами та замовниками. Зростання кількості фрилансерів та замовників свідчить про значний попит на такі рішення. Створення вебзастосунків для спільної роботи та комунікації може покращити продуктивність робочих процесів та зменшити витрати часу та ресурсів для обох сторін.

Об'єктом кваліфікаційної роботи є процеси взаємодії фрілансерів та замовників у веброзробці.

Предметом кваліфікаційної роботи є функціональні можливості та інтерфейс вебзастосунку для полегшення співпраці фрілансерів та замовників.

Метою кваліфікаційної роботи є створення вебплатформи, спрямованої на полегшення та оптимізацію процесу взаємодії між фрілансерами та замовниками.

Перший розділ містить аналіз предметної області, аналіз готових рішень та специфікацію вимог. Другий розділ містить спроєктовані діаграми варіантів використання, класів, послідовностей та мокап системи. У третьому розділі представлено вибір технологій розробки, опис структури бази даних та розроблений мокап системи. У четвертому розділі представлено реалізацію розробки вебзастосунку.

КРБ викладена на 94 сторінки, вона містить 4 розділи, 40 ілюстрацій, 26 джерел в переліку посилань

Ключові слова: вебзастосунок, фрілансери, замовники, спільна робота, управління проєктами, комунікація, ефективність.

ABSTRACT

to the qualification work of the bachelor

"Web application for the interaction of freelancers and customers"

Student 409 gr.: Artem Yuriyovych Samokysh

Supervisor: candidate technical of Sciences, associate professor G. V. Horban

The qualification work is devoted to the development and implementation of a web application aimed at simplifying the process of joint work between freelancers and their customers.

The relevance of the topic lies in the need to optimize the processes of search, hiring and cooperation between freelancers and customers. The increase in the number of freelancers and customers indicates a significant demand for such solutions. Creating web applications for collaboration and communication can improve workflow productivity and reduce time and resource costs for both parties.

The object of the qualification work is the process of interaction between freelancers and customers in web development.

The subject of the qualification work is the functionality and interface of the web application to facilitate cooperation between freelancers and customers.

The purpose of the qualification work is to create a web platform aimed at facilitating and optimizing the process of interaction between freelancers and customers.

The first section contains the analysis of the subject area, the analysis of ready-made solutions and the specification of requirements. The second section contains designed diagrams of use cases, classes, sequences and system mockups. The third section presents a selection of development technologies, a description of the database structure, and a developed mockup of the system. The fourth chapter presents the implementation of web application development.

BQW is laid out on 94 pages, it contains 4 sections, 40 illustrations, 26 sources in the list of references

Keywords: web application, freelancers, customers, joint work, project management, communication, efficiency.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API	–	Application Programming Interface
XML	–	eXtensible Markup Language
HTML	–	HyperText Markup Language
DOM	–	Document object model
SPA	–	Single Page Application
JVM	–	Java Virtual Machine
Java SE	–	Java Standard Edition
GUI	–	Graphical user interface
AOP	–	Aspect-Oriented Programming
MVC	–	Model-View-Controller
HTTP	–	HyperText Transfer Protocol
JPA	–	Java Persistence API
ORM	–	Object-Relational Mapping
JMS	–	Java Message Service
RMI	–	Remote Method Invocation
UML	–	Unified Modeling Language
JWT	–	JSON Web Token

ВСТУП

У сучасному світі ринку праці, де розвиток технологій та зміна підходів до організації робочого процесу стають все більш пріоритетними, важливою стає тема ефективної співпраці між фрілансерами та замовниками. Зростання популярності фрілансу як форми праці та невпинний розвиток вебтехнологій відкривають нові можливості для спільної роботи та спілкування в цьому сегменті. Створення вебзастосунків, спрямованих на полегшення взаємодії між фрілансерами та замовниками, є актуальною задачею сьогодення.

Актуальність теми полягає у необхідності оптимізації процесів пошуку, найму та співпраці між фрилансерами та замовниками. Зростання кількості фрилансерів та замовників свідчить про значний попит на такі рішення. Створення вебзастосунків для спільної роботи та комунікації може покращити продуктивність робочих процесів та зменшити витрати часу та ресурсів для обох сторін.

Об'єктом кваліфікаційної роботи є способи взаємодії фрілансерів та замовників у веброзробці для полегшення комунікації та управління проєктами через Інтернет.

Предметом роботи є функціональні можливості та інтерфейс вебзастосунку, спрямованого на полегшення співпраці та комунікації між цими двома групами учасників.

Метою кваліфікаційної роботи є створення вебплатформи, спрямованої на полегшення та оптимізацію процесу взаємодії між фрілансерами та замовниками.

Шляхом розробки цього вебзастосунку планується підвищити ефективність співпраці, зменшити часові витрати на пошук та вибір підходящих кандидатів, а також сприяти покращенню якості виконання завдань за рахунок зручного та ефективного інструментарію для комунікації та спільної роботи.

Зокрема, це передбачає створення зручного та інтуїтивно зрозумілого інтерфейсу для пошуку та відбору фахівців, управління проєктами та комунікації між усіма учасниками процесу.

Для досягнення цієї мети визначено наступні завдання:

1) проведення аналізу сучасного стану ринку фрілансу та існуючих вебзастосунків для співпраці фрілансерів із замовниками;

2) розробка архітектури вебзастосунку та визначення основних функцій;

3) розробка діаграм варіантів використання, класів, послідовностей дій і діяльності;

4) вибір технологій розробки та опис бази даних;

5) реалізація інтерфейсу користувача, функціональних можливостей програми та тестування.

Аналіз сучасного стану ринку фрілансу та існуючих вебзастосунків свідчить про необхідність створення нового програмного забезпечення, орієнтованого на покращення співпраці між фрілансерами та замовниками. Існуючі рішення не завжди відповідають потребам учасників процесу та можуть бути недостатньо ефективними для оптимальної організації робочих потоків.

Основні проєктні рішення включають в себе розробку зручного та зрозумілого інтерфейсу, механізми вибору фахівців та управління проєктами, а також засоби комунікації між усіма учасниками процесу.

Результати даної роботи можуть бути використані в різних галузях діяльності, де існує потреба у співпраці між фрілансерами та замовниками, зокрема в інформаційних технологіях, маркетингу, дизайні тощо.

Отже, розробка вебзастосунку для взаємодії фрілансерів та замовників є актуальною задачею, що має значний потенціал для покращення співпраці та продуктивності в цьому сегменті ринку праці.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

Аналіз предметної області «Вебзастосунок взаємодії фрілансерів та замовників» відкриває перед нами широкий спектр аспектів, які потребують уважного розгляду та дослідження. У цій області взаємодія між фрілансерами, які надають послуги, та замовниками, які шукають фахівців для виконання різноманітних завдань, є критично важливою.

Фрілансери, як правило, шукають можливості для роботи над проєктами, що відповідають їхнім навичкам та інтересам. Для них важливо мати доступ до різноманітних проєктів, які вони можуть виконати віддалено. Основними потребами фрілансерів є можливість знаходити нових клієнтів, отримувати замовлення та відправляти свої пропозиції на проєкти. Вони також цінують зручний інтерфейс та ефективну систему спілкування з замовниками, яка дозволяє чітко визначити вимоги проєкту та узгодити умови співпраці.

З іншого боку, замовники шукають надійних та кваліфікованих фахівців для виконання своїх проєктів. Вони можуть мати різноманітні потреби, починаючи від розробки вебсайтів та мобільних застосунків до написання статей та створення графічного дизайну. Для замовників важливо мати можливість швидко знайти потрібних спеціалістів, переглянути їхні портфоліо та резюме, а також укласти угоду про умови співпраці.

Оскільки взаємодія фрілансерів та замовників відбувається в онлайн-середовищі, важливою є також забезпечення безпеки та конфіденційності даних. Як фрілансери, так і замовники повинні мати можливість довіряти платформі та бути впевненими у захисті своїх особистих даних та інформації про проєкти.

Крім того, важливо враховувати різноманітність послуг та професійних навичок, які шукають фрілансери, а також різноманіття видів проєктів, які потребують замовники. Це може включати різні галузі, такі як програмування, дизайн, письменництво, маркетинг тощо. Тому важливо, щоб вебзастосунок

надавав можливість шукати та фільтрувати результати за різними критеріями, щоб забезпечити ефективну взаємодію між фрілансерами та замовниками в різних сферах діяльності.

Загалом, аналіз предметної області вебзастосунку взаємодії фрілансерів та замовників є важливим етапом у розробці системи, який дозволяє зрозуміти потреби обох сторін, визначити ключову функціональність та вимоги до системи, а також виявити можливості для покращення та інновацій.

1.2 Аналіз готових рішень

1.2.1 Upwork

Назва: Upwork [1]

Розробник: Upwork Global Inc.

Архітектура: 3-tier web application

Мова реалізації: JavaScript, Python (Back-end)

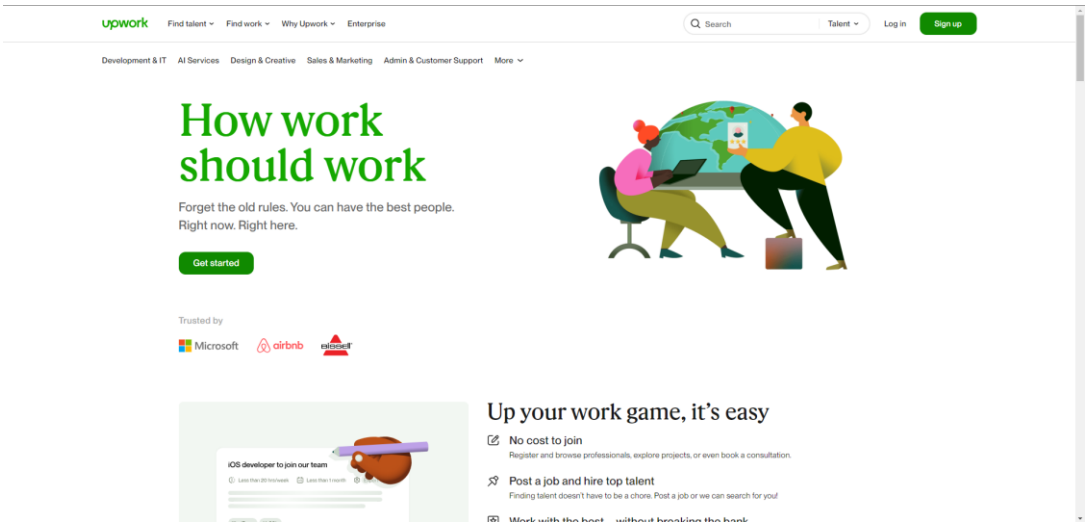


Рисунок 1.1 – Зображення Upwork

Функції:

1) створення профілю для фрілансерів та замовників;
2) розміщення проєктів та пропозицій на проєкти;
3) спілкування через вбудований чат;
4) платіжна система для здійснення оплати за роботу;

5) оцінювання фрілансерів та замовників.

Переваги:

– велика база фрілансерів різних спеціалізацій;

– зручний інтерфейс для пошуку проєктів та спілкування з клієнтами;

– вбудована система оплати та відстеження робочого часу;

– можливість оцінювання якості роботи фрілансерів.

Недоліки:

– висока комісія для обох сторін (фрілансерів та замовників);

– конкуренція серед фрілансерів може бути високою особливо для новачків;

– обмежена можливість комунікації за межами платформи.

Зображення інтерфесу Freelancer.com представлена на рис. 1.1.

1.2.2 Freelancer.com

Назва: Freelancer.com [2]

Розробник: Freelancer Technology Pty Limited

Архітектура: 3-tier web application

Мова реалізації: PHP, JavaScript

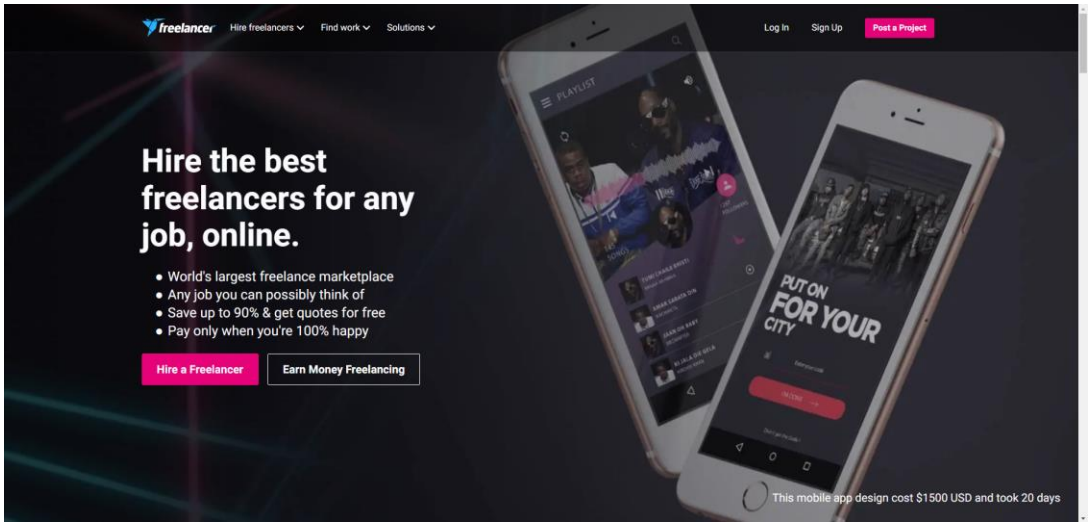


Рисунок 1.2 – Зображення Freelancer.com

Функції:

1) реєстрація як фрілансер або замовник;

2) розміщення проєктів та тендерів;

3) можливість участі у тендерах та конкурсах;

4) система оплати та відстеження робочого часу;

5) підтримка декількох мов та валют.

Переваги:

– широкий вибір проєктів різного рівня складності та обсягу;

– можливість участі у тендерах та конкурсах для здобуття проєктів;

– система гарантійного фонду для захисту платежів, доступ до широкого кола фрілансерів з різних країн.

Недоліки:

– велика комісія та додаткові витрати за використання додаткових функцій платформи;

– висока конкуренція.

Зображення інтерфесу Freelancer.com пердставлена на рис. 1.2.

1.2.3 Fiverr

Назва: Fiverr [3]

Розробник: Fiverr International Ltd.

Архітектура: 3-tier web application

Мова реалізації: Ruby, JavaScript

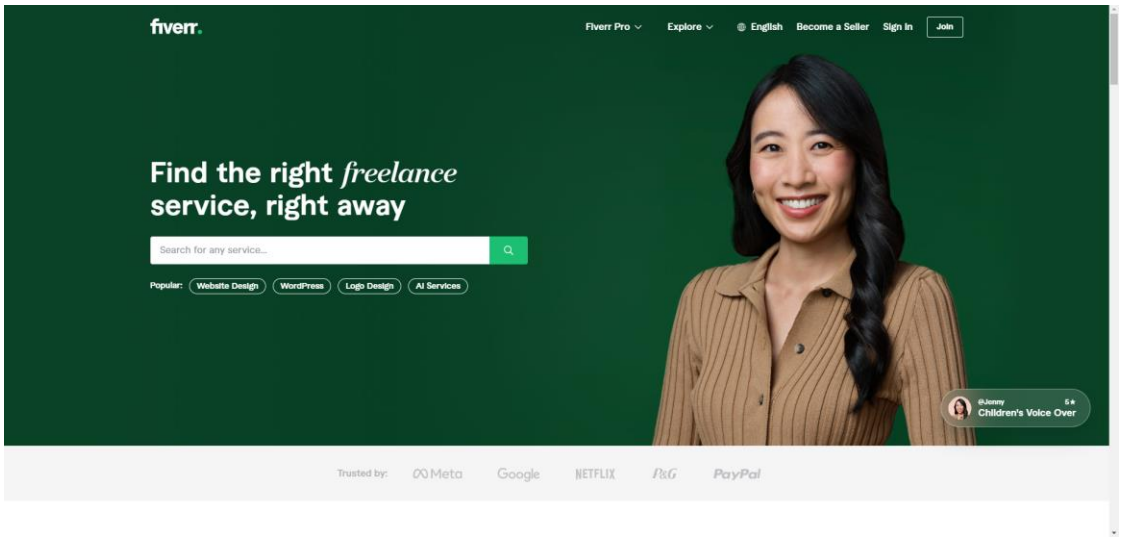


Рисунок 1.3 – Зображення Fiverr

Функції:

1) реєстрація як фрілансер або замовник;
2) розміщення послуг (ґігів) та пропозицій;
3) можливість замовлення послуг та участь у проєктах;
4) система оплати та відстеження виконання замовлень;
5) підтримка декількох мов та валют.

Переваги:

– простота використання та швидкість замовлення послуг;
– різноманітність послуг та гнучка система ціноутворення;
– доступ до глобального ринку фрілансерів та послуг.

Недоліки:

– обмежена можливість пошуку та фільтрації результатів;
– висока конкуренція серед фрілансерів.

Зображення інтерфесу Freelancer.com представлена на рис. 1.3.

1.3 Специфікація вимог

Призначення та межі проєкту

Призначення системи

Цей проєкт спрямований на створення вебзастосунку для полегшення співпраці та комунікації між фрілансерами та замовниками у сфері веброзробки.

Погодження, що ухвалені в програмній документації

Погоджень не ухвалено.

Межі проєкту ПЗ

Крайня дата завершення роботи над ПЗ – 01.06.2024р.

Загальний опис

Сфера застосування

Вебзастосунок призначений для використання фрілансерами та замовниками у різноманітних галузях та послугах для полегшення співпраці та комунікації під час виконання проєктів.

Характеристики користувачів

Користувачі цієї вебплатформи, які можуть бути фрілансерами або замовниками, можуть мати різний рівень знань та досвіду у використанні вебзастосунків, що відображається у їхніх очікуваннях щодо простоти використання та інтуїтивності інтерфейсу.

Загальна структура і склад системи

Загальна структура системи включатиме клієнтську та серверну частини. Для зберігання даних буде використана реляційна база даних.

Загальні обмеження

Вебзастосунок буде обмежений доступом для неавторизованих користувачів.

Функції системи

Функції системи вебзастосуноку для взаємодії фрілансерів та замовників

- реєстрація користувачів (фрілансерів та замовників);
- пошук і перегляд проєктів для фрілансерів;
- розміщення проєктів для замовників;
- взаємодія між фрілансерами та замовниками (обговорення деталей проєкту, уточнення вимог, обговорення ціни тощо);
- оцінювання роботи фрілансерів з боку замовників.

Джерела і зміст вхідної інформації

Джерела вхідної інформації включають дані реєстрації користувачів, інформацію про проєкти, текстові обговорення, оцінки роботи та інші метадані.

Нормативно-довідкова інформація

Вимоги до даного пункту відсутні.

Вимоги до способів організації, збереження та ведення інформації

Для системи вибрано MySQL як базу даних.

Вимоги до технічного забезпечення

Головна умова для використання системи – доступ до веббраузера на пристрої користувача

Архітектура програмної системи

Система складається з клієнтської та серверної частин, а також бази даних.

Системне програмне забезпечення

Будь-яка операційна система з встановленою Java.

Мова і технологія розробки ПЗ

Застосунок реалізовано на Java з використанням фреймворку Spring Framework для бекенду, а також на JavaScript з використанням бібліотеки React для фронтенду, з використанням бази даних MySQL.

Інтерфейс користувача

Виконання інтерфейсу користувача повинно відповідати всім дизайнерським вимогам, щоб забезпечити максимальний комфорт та ефективність роботи користувача з системою.

Апаратний інтерфейс

Апаратний інтерфейс охоплює пристрій користувача, який буде використовуватися для взаємодії з вебзастосунком.

Висновки до розділу 1

В першому розділі кваліфікаційної роботи бакалавра детально розглянуто концепцію вебзастосунку для взаємодії фрілансерів та замовників, що відкриває нові можливості для обох сторін у сфері ведення бізнесу. Застосунок спрямований на полегшення пошуку та здійснення замовлень, забезпечуючи зручний і ефективний інтерфейс для користувачів.

Описаний функціонал включає в себе реєстрацію користувачів як фрілансерів чи замовників, можливість розміщення та перегляду проєктів, спілкування між учасниками через вбудований чат.

У ході аналізу існуючих систем, таких як Upwork, Freelancer.com та Fiverr, були виявлені ключові аспекти, які можна врахувати при розробці власної платформи. Зокрема, це включає широкий функціонал, можливість гнучкої настройки вимог користувачів, а також важливість забезпечення безпеки та конфіденційності даних.

2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ ВЗАЄМОДІЇ ФРІЛАНСЕРІВ ТА ЗАМОВНИКІВ

2.1 Розробка діаграми варіантів використання

Діаграма варіантів використання – це вид діаграми у моделюванні програмного забезпечення, який описує функціональність системи з точки зору користувача або акторів, що взаємодіють з системою. Діаграма варіантів використання вебзастосунку представлена на рис. 2.1.

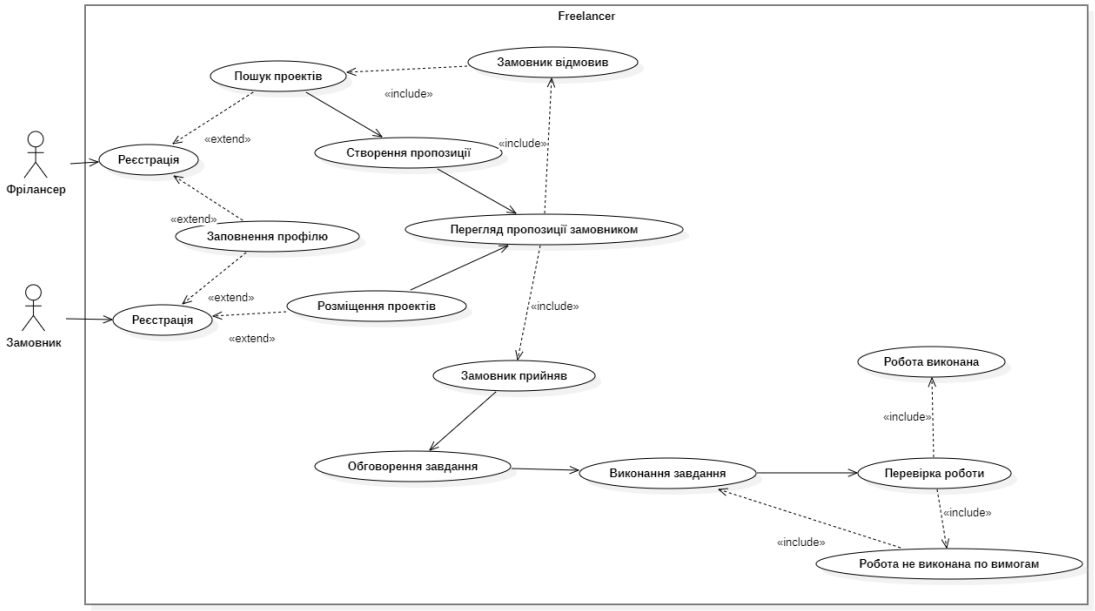


Рисунок 2.1 – Діаграма варіантів використання вебзастосунку

Ця діаграма варіантів використання показує взаємодію між фрілансерами та замовниками у вебзастосунку. Фрілансери реєструються, заповнюють профіль, шукають проєкти, створюють пропозиції, а замовники – розміщують проєкти, переглядають пропозиції, приймають їх або відхиляють, після чого обговорюють завдання, підтверджують роботу після перевірки виконаної роботи.

2.2 Розробка діаграми класів

Діаграма класів – це вид діаграми у моделюванні програмного забезпечення, який використовується для візуалізації структури класів та їх взаємозв'язків у системі.

Діаграми класів з атрибутами та методами представлені на рис. 2.2.

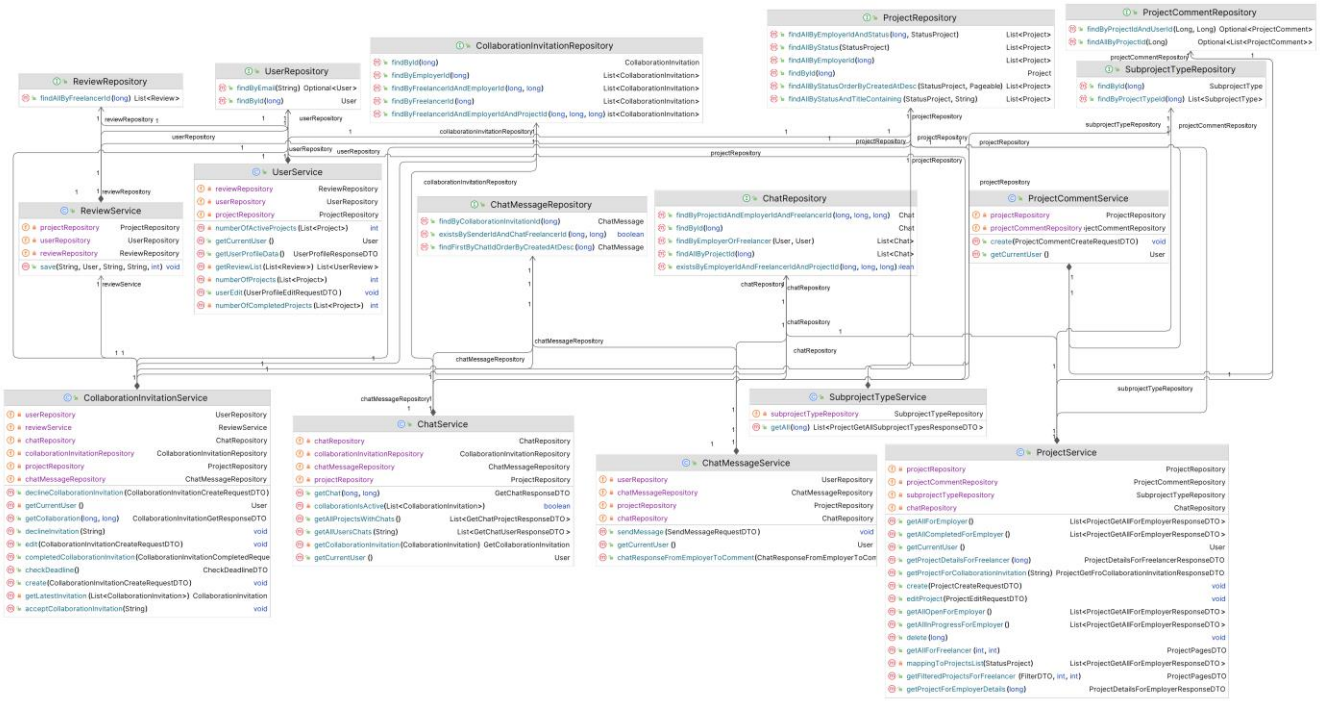


Рисунок 2.2 – Діаграма класів

Опис сервісів:

а) UserService: Цей сервіс відповідає за управління користувачами, включаючи отримання інформації про користувачів, їх профілі та проєкти.

Основні методи включають:

– numberOfActiveProjects(List<Project>), який повертає кількість активних проєктів;

– numberOfCompletedProjects(List<Project>), який повертає кількість завершених проєктів;

– getUserProfileData(), що отримує дані профілю користувача;

– getReviewList(List<Review>), що отримує список відгуків; та

– getCurrentUser(), який повертає поточного користувача. Взаємодіє з UserRepository, ReviewRepository, ProjectRepository для отримання та управління даними користувачів та проєктів.

б) ReviewService: Цей сервіс обробляє всю інформацію, пов'язану з відгуками, зокрема, збереження нових відгуків та отримання списку відгуків для конкретного фрілансера.

Основні методи включають:

– save(String, User, String, int), який дозволяє зберігати новий відгук;

– getReviewList(List<Review>), що отримує список відгуків для конкретного користувача. Взаємодіє з ReviewRepository, UserRepository, ProjectRepository для обробки даних відгуків та проєктів.

в) ProjectService: Цей сервіс відповідає за управління проєктами, включаючи створення нових проєктів, отримання інформації про поточні та завершені проєкти.

Основні методи включають:

– getAllForEmployer(), що повертає всі проєкти для замовника;

– getAllInProgressForEmployer(), що повертає всі проєкти в процесі виконання для замовника;

– getAllInProgressForFreelancer(), що повертає всі проєкти в процесі виконання для фрілансера;

– create(ProjectCreateRequestDTO), який дозволяє створювати нові проєкти. Взаємодіє з ProjectRepository, UserRepository, ChatRepository для управління даними проєктів та користувачів.

г) ChatService: Цей сервіс управляє чатами між користувачами, включаючи отримання повідомлень та проєктних чатів.

Основні методи включають:

– getChat(Long, Long), який отримує чат за його ID та ID користувача;

– getAllProjectsWithChats(), що повертає всі проєкти з чатами;

– getChatMessages(Long), який отримує всі повідомлення в чаті за його ID. Взаємодіє з ChatRepository, ChatMessageRepository, CollaborationInvitationRepository для обробки даних чатів та повідомлень.

д) CollaborationInvitationService: Цей сервіс відповідає за управління запрошеннями на співпрацю між користувачами.

Основні методи включають:

– declineCollaborationInvitation(CollaborationInvitationCreateRequestDTO), який відхиляє запрошення на співпрацю;

– acceptCollaborationInvitation(String), який приймає запрошення на співпрацю. Взаємодіє з CollaborationInvitationRepository, ReviewService, UserRepository, ProjectRepository для обробки запрошень та даних користувачів.

е) ProjectCommentService: Цей сервіс обробляє коментарі до проєктів, включаючи створення нових коментарів та отримання існуючих. Основний метод включає:

– create(ProjectCommentCreateRequestDTO), який дозволяє створювати нові коментарі до проєктів. Взаємодіє з ProjectRepository, ProjectCommentRepository для обробки даних коментарів.

ж) SubprojectTypeService: Цей сервіс управляє типами підпроєктів, дозволяючи отримувати доступ до типів підпроєктів за їх ID.

Основний метод включає:

– getAll(Long), який повертає всі типи підпроєктів для конкретного проєкту. Взаємодіє з SubprojectTypeRepository.

з) ChatMessageService: Цей сервіс управляє повідомленнями в чатах, включаючи надсилання та отримання повідомлень.

Основні методи включають:

– sendMessage(SendMessageRequestDTO), який дозволяє надсилати повідомлення;

– getChatResponseFromEmployerToComment(), який отримує відповідь замовника на коментар. Взаємодіє з ChatMessageRepository, UserRepository, ChatRepository для обробки даних повідомлень та чатів.

Репозиторії:

– UserRepository: надає доступ до даних користувачів, дозволяє знайти користувача за email або ID;

– ReviewRepository: надає доступ до даних відгуків, дозволяє знайти всі відгуки про фрілансера за його ID;

– ProjectRepository: надає доступ до даних проєктів, дозволяє знайти проєкти за статусом, ID замовника, ID фрілансера;

– ChatRepository: надає доступ до даних чатів, дозволяє знайти чати за ID проєкту та фрілансера, перевірити існування чату;

– CollaborationInvitationRepository: надає доступ до даних запрошень на співпрацю, дозволяє знайти запрошення за їх ID;

– ProjectCommentRepository: надає доступ до даних коментарів до проєктів, дозволяє знайти коментарі за ID проєкту та користувача;

– SubprojectTypeRepository: надає доступ до даних типів підпроєктів, дозволяє знайти тип підпроєкту за його ID;

– ChatMessageRepository: надає доступ до даних повідомлень у чатах, дозволяє знайти повідомлення за ID чату, сортувати за датою створення.

2.3 Розробка діаграми послідовностей дій системи

Діаграма послідовності - це вид діаграми у моделюванні програмного забезпечення, який показує послідовність взаємодій між об'єктами або компонентами системи в певному порядку часу.

Діаграма послідовності системи реєстрації та додавання відгуку представлена на рис. 2.3.

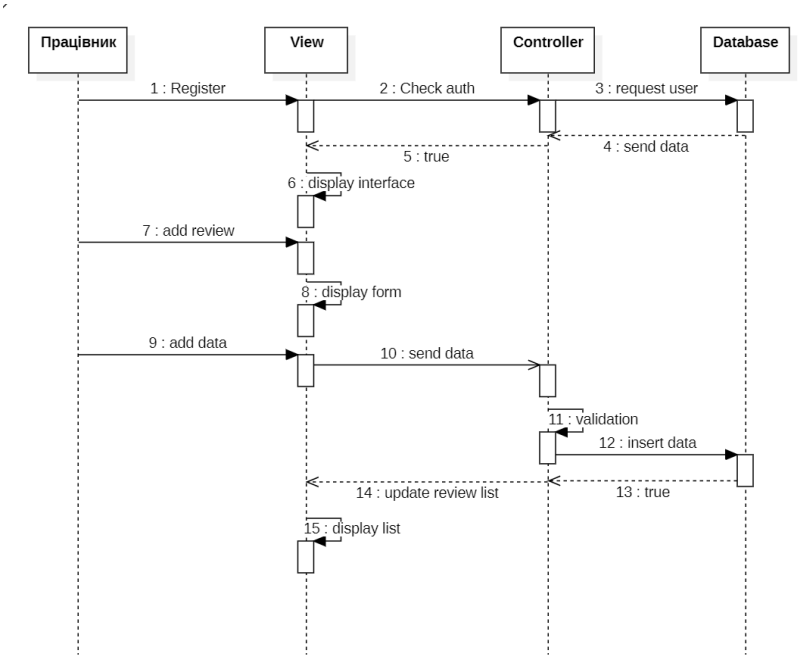


Рисунок 2.3 – Діаграма послідовності системи

Ця діаграма послідовностей ілюструє процес реєстрації та додавання відгуку працівником. Спочатку працівник реєструється, і аутентифікація перевіряється контролером. Контролер робить запит до бази даних, отримує дані і повертає результат перевірки. Інтерфейс відображається працівнику. Працівник додає відгук, і форма відображається. Після заповнення даних, дані надсилаються контролеру, який перевіряє їх, вставляє в базу даних і підтверджує успіх. Інтерфейс оновлює список відгуків і відображає його працівнику.

Діаграма послідовності системи реєстрації та створення вакансії наймачем представлена на рис. 2.4.

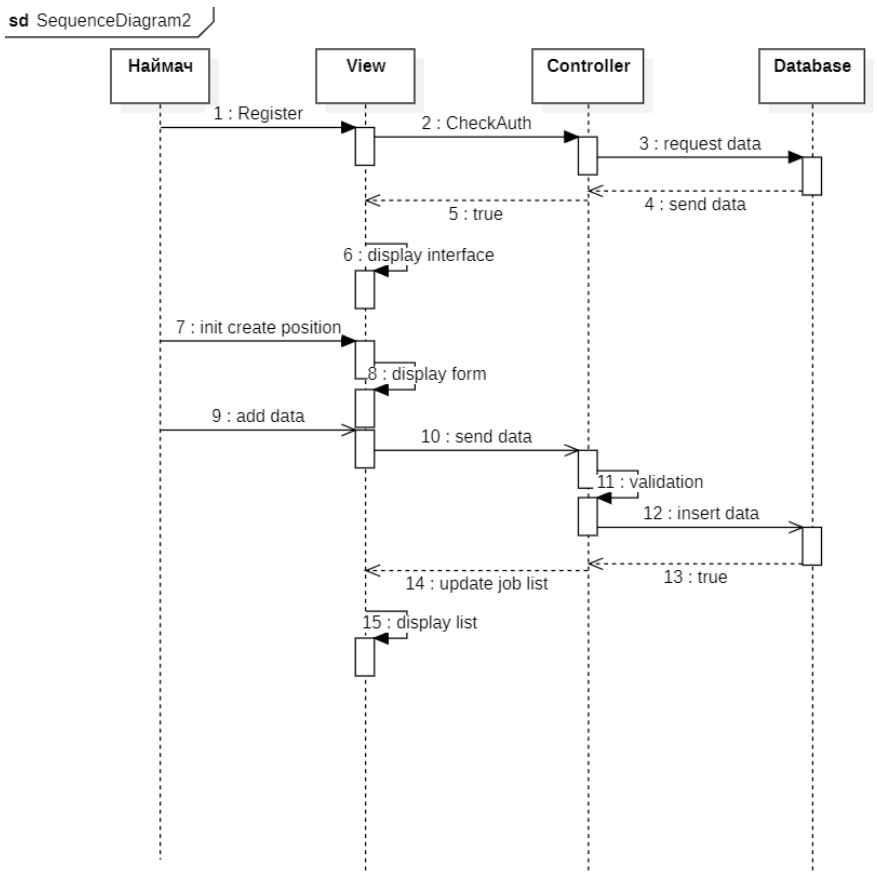


Рисунок 2.4 – Діаграма послідовності системи

Ця діаграма послідовності ілюструє процес реєстрації та створення вакансії наймачем. Спочатку наймач реєструється, і аутентифікація перевіряється контролером. Контролер робить запит до бази даних, отримує дані і повертає результат перевірки. Інтерфейс відображається наймачу. Наймач ініціює

створення нової вакансії, і форма відображається. Після заповнення даних, дані надсилаються контролеру, який перевіряє їх, вставляє в базу даних і підтверджує успіх. Інтерфейс оновлює список вакансій і відображає його наймачу.

Діаграма послідовності системи авторизації та додавання відгуку працівником представлена на рис. 2.5.

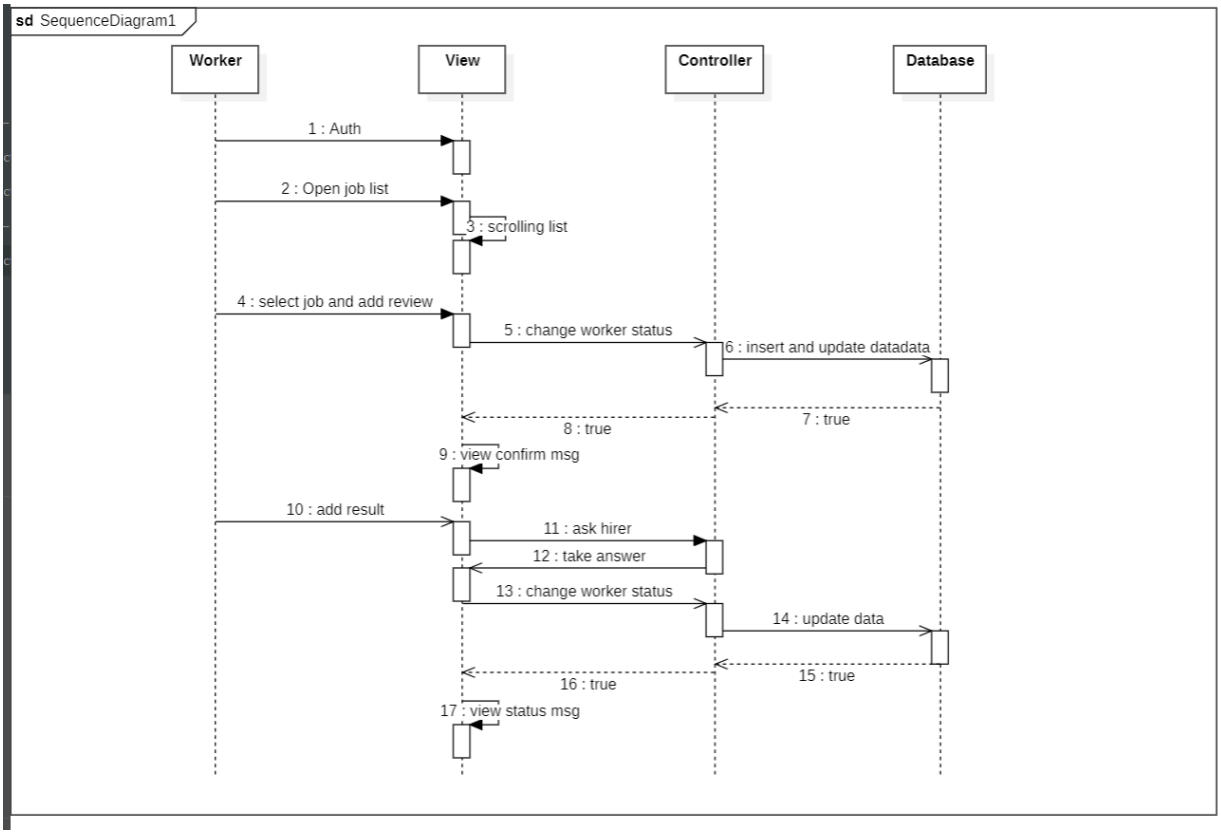


Рисунок 2.5 – Діаграма послідовності системи

Ця діаграма послідовностей ілюструє процес авторизації та додавання відгуку працівником. Спочатку працівник проходить авторизацію. Після успішної авторизації, працівник відкриває список вакансій і може прокручувати його. Вибравши вакансію, працівник додає відгук. Контролер змінює статус працівника і вставляє та оновлює дані в базі даних. Після успішного оновлення, працівник бачить підтвердження на інтерфейсі. Потім працівник може додати результат виконання роботи, і цей результат надсилається наймачу для підтвердження. Наймач відповідає, і статус працівника оновлюється знову. Інтерфейс відображає статус працівника.

2.4 Розробка діаграм діяльності

Діаграми діяльності – це тип діаграм в UML, що відображають потік роботи в системі. Вони ілюструють послідовність дій і їх взаємодію, дозволяючи зрозуміти процеси та алгоритми, що виконуються в системі.

Діаграма діяльності для розміщення замовлення замовником представлена на рис. 2.6.

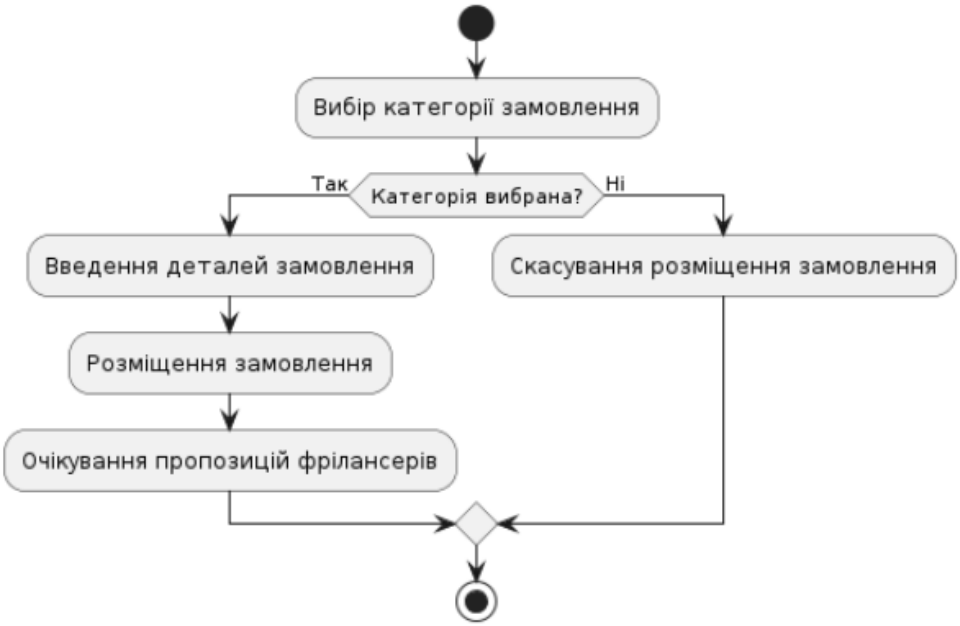


Рисунок 2.6 – Діаграма діяльності для розміщення замовлення замовником

Ця діаграма показує процес розміщення замовлення замовником. Спочатку замовник вибирає категорію замовлення. Якщо категорія вибрана, користувач вводить деталі замовлення і розміщує замовлення. Далі замовлення очікує на пропозиції від фрілансерів. У разі невибору категорії замовлення, процес завершується скасуванням розміщення замовлення.

Діаграма діяльності створення проєкту фрілансером представлена на рис. 2.7.

Рисунок 2.7 – Діаграма діяльності для створення проєкту фрілансером

Ця діаграма відображає процес створення проєкту фрілансером. Спочатку фрілансер вибирає тип проєкту. Якщо тип обрано, фрілансер вводить деталі проєкту і створює проєкт. Після цього проєкт надсилається на розгляд. У разі невибору типу проєкту, процес завершується скасуванням створення проєкту.

Діаграма діяльності реєстрації користувача представлена на рис. 2.8.

Рисунок 2.8 – Діаграма діяльності для реєстрації користувача

Ця діаграма показує процес реєстрації користувача. Спочатку користувач вводить свої дані. Якщо дані валідні, створюється обліковий запис і користувач

входить в систему. Якщо дані невалідні, відображається повідомлення про помилку.

Висновки до розділу 2

У другому розділі проведено детальний аналіз функціональних та структурних аспектів вебзастосунку для взаємодії фрілансерів та замовників. Використання діаграм варіантів використання, класів, послідовностей дій та діяльностей системи дозволило систематизувати функціональні можливості та взаємозв'язки між елементами системи.

Діаграми варіантів використання показують, як фрілансери та замовники взаємодіють з системою, зокрема реєстрацію, авторизацію, створення вакансій та додавання відгуків. Вони допомагають зрозуміти дії користувачів та підтримку системи.

Діаграми класів деталізують внутрішню структуру вебзастосунку, показуючи об'єкти системи та їх взаємозв'язки. Це допомагає уявити організацію даних, атрибути та методи об'єктів.

Діаграми послідовностей дій ілюструють порядок взаємодій між об'єктами системи у часі, показуючи процеси реєстрації, додавання відгуків, створення вакансій та інші операції.

Діаграми діяльності деталізують окремі процеси, допомагаючи зрозуміти, як здійснюються ключові операції у системі.

Таким чином, використання діаграм варіантів використання, класів, послідовностей дій та діяльностей допомогло структурувати функціональні можливості вебзастосунку, забезпечуючи глибоке розуміння взаємозв'язків між його компонентами. Це створює основу для подальшої розробки та вдосконалення системи, забезпечуючи її ефективне функціонування та зручність для користувачів.

3 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ ТА ОПИС БАЗИ ДАНИХ

3.1 Вибір технологій розробки

Для реалізації вебзастосунку обрано наступний:

- JavaScript;
- React;
- Java;
- Spring Framework;
- MySQL.

JavaScript — це динамічна, інтерпретована мова програмування, створена Бренданом Айком у 1995 році під час роботи в компанії Netscape. Відтоді JavaScript став однією з найпопулярніших мов програмування у світі, що широко використовується для розробки вебзастосунків, серверних програм, мобільних застосунків і багато іншого [4]. JavaScript відомий своєю універсальністю, гнучкістю і здатністю виконувати код на клієнтській стороні, що робить його незамінним інструментом для створення інтерактивних і динамічних вебсайтів.

Однією з ключових особливостей JavaScript є його здатність виконувати код безпосередньо в браузері, що дозволяє створювати інтерактивні елементи користувацького інтерфейсу без необхідності перезавантаження сторінки. Це робить вебсайти більш чутливими і зручними для користувачів, оскільки вони можуть взаємодіяти з елементами інтерфейсу в режимі реального часу. JavaScript підтримує подієво-орієнтоване програмування, що дозволяє реагувати на дії користувача, такі як натискання кнопок, введення даних у форми, переміщення курсора миші та багато іншого.

JavaScript є мовою з динамічною типізацією, що означає, що змінні можуть змінювати свій тип даних під час виконання програми. Це забезпечує велику гнучкість і зручність у роботі з даними, оскільки розробники можуть легко змінювати типи даних залежно від потреб програми [5]. JavaScript також підтримує функції першого класу, що дозволяє передавати функції як аргументи,

повертати їх як результати і зберігати їх у змінних. Це відкриває широкі можливості для створення складних і гнучких програмних конструкцій, таких як замикання і лямбда-вирази.

JavaScript має потужну стандартну бібліотеку, яка включає безліч вбудованих функцій і об'єктів для роботи з рядками, масивами, датами, регулярними виразами і багато іншого. Крім того, існує безліч сторонніх бібліотек і фреймворків, які розширюють можливості JavaScript і полегшують розробку застосунків. Серед найпопулярніших бібліотек можна виділити jQuery, Lodash і Moment.js, а серед фреймворків — React, Angular і Vue.js. Ці інструменти дозволяють швидко і ефективно створювати складні вебзастосунки з багатим функціоналом і зручним інтерфейсом.

JavaScript також підтримує асинхронне програмування, що дозволяє виконувати тривалі операції без блокування основного потоку виконання. Це особливо важливо для роботи з мережевими запитами, завантаженням даних, обробкою файлів та іншими операціями, які можуть займати тривалий час [6]. Асинхронні операції в JavaScript реалізовані за допомогою колбеків, промісів і async/await синтаксису, що робить код більш читабельним і легким для розуміння.

Node.js — це середовище виконання для JavaScript, яке дозволяє запускати JavaScript-код на серверній стороні. Це розширює сферу застосування JavaScript і дозволяє створювати повноцінні серверні застосунки, використовуючи одну і ту ж мову як на клієнтській, так і на серверній стороні [7]. Node.js забезпечує високу продуктивність і масштабованість завдяки своєму неблокуючому вводу/виводу і подієво-орієнтованій архітектурі. Це робить його ідеальним вибором для розробки реальних застосунків, таких як чат-системи, потокове передавання даних, API та багато іншого.

JavaScript також використовується для розробки мобільних застосунків за допомогою таких фреймворків, як React Native, Ionic і NativeScript [8]. Це дозволяє створювати кросплатформні мобільні застосунки, які працюють на різних операційних системах, таких як iOS і Android, використовуючи одну кодову базу.

Це значно скорочує час і витрати на розробку, оскільки немає потреби створювати окремі застосунки для кожної платформи.

Завдяки своїй популярності і широкому використанню, JavaScript має велику і активну спільноту розробників. Це означає наявність безлічі ресурсів для навчання, документації, форумів і бібліотек, що робить вивчення і використання JavaScript доступним і зручним для розробників будь-якого рівня. Постійні оновлення мови і поява нових інструментів і технологій забезпечують постійний розвиток і актуальність JavaScript у сучасному світі.

У підсумку, JavaScript є потужною і універсальною мовою програмування, яка забезпечує розробників інструментами для створення динамічних і інтерактивних вебзастосунків, серверних програм, мобільних застосунків і багато іншого. Його гнучкість, продуктивність і широка екосистема роблять JavaScript одним із найпопулярніших і найвикористовуваніших інструментів у світі програмування.

React — це бібліотека JavaScript з відкритим кодом, яка використовується для створення інтерфейсів користувача. Вона була розроблена компанією Facebook і забезпечує розробникам можливість створювати масштабовані та високопродуктивні вебзастосунки [9]. Ця бібліотека сприяє створенню динамічних і чутливих інтерфейсів користувача, забезпечуючи високу продуктивність і зручність використання.

Однією з ключових особливостей React є компонентний підхід, який передбачає розділення користувацького інтерфейсу на незалежні, багаторазово використовувані компоненти. Кожен компонент є ізольованим елементом інтерфейсу, який має власну логіку і стан. Це сприяє організованості коду і полегшує його обслуговування, оскільки кожен компонент може бути розроблений, протестований і змінений окремо від інших [10]. Компоненти можна комбінувати і вкладати один в одного, створюючи складні інтерфейси з простих блоків.

React використовує JSX, що дозволяє писати структуру користувацького інтерфейсу з використанням синтаксису, схожого на HTML, прямо в JavaScript-

коді. Це робить код більш читабельним і зрозумілим, оскільки розмітка і логіка компонента знаходяться в одному файлі. JSX також підтримує вбудовування JavaScript-виразів у розмітку, що дозволяє легко працювати з даними і динамічно змінювати інтерфейс залежно від стану компонента.

Віртуальний DOM є ще однією важливою особливістю React, яка сприяє підвищенню продуктивності застосунків. Замість безпосередньої роботи з реальним DOM, React використовує віртуальну копію DOM для відстеження змін у стані компонентів. Коли стан компонента змінюється, React спочатку оновлює віртуальний DOM, а потім порівнює його з реальним DOM, застосовуючи лише необхідні зміни. Це дозволяє зменшити кількість операцій з реальним DOM, які є відносно повільними, і підвищити продуктивність застосунків.

React підтримує односторонній потік даних, що означає, що дані в застосунку рухаються в одному напрямку [11]. Це робить застосунок більш передбачуваним і легшим у налагодженні, оскільки стан компонентів може змінюватися лише через явні дії (наприклад, виклик функції або зміну властивостей). Такий підхід також сприяє кращій організації коду і зменшує ймовірність виникнення помилок.

React має потужну екосистему і підтримує інтеграцію з різними інструментами і бібліотеками. Наприклад, для управління станом застосунків часто використовуються бібліотеки Redux або MobX, які забезпечують централізоване зберігання стану і спрощують роботу з великими застосунками [12]. React Router дозволяє створювати односторінкові застосунки з маршрутизацією, що робить навігацію між різними сторінками більш плавною і швидкою.

Розробка з використанням React дозволяє створювати динамічні і чутливі інтерфейси користувача з високою продуктивністю і зручністю використання. Завдяки компонентному підходу, віртуальному DOM і односторонньому потоку даних, React забезпечує високу продуктивність і передбачуваність застосунків. Потужна екосистема і підтримка різних інструментів роблять React гнучким і зручним для розробки застосунків будь-якої складності.

React широко використовується великими компаніями і розробниками у всьому світі, включаючи Facebook, Instagram, WhatsApp, Netflix, Airbnb і багато інших [13]. Це свідчить про його надійність, масштабованість і популярність серед професійних розробників. Завдяки активній спільноті і постійним оновленням, React продовжує залишатися актуальним і затребуваним інструментом для створення сучасних вебзастосунків.

Java — це об'єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems (нині належить Oracle) у 1995 році. Вона відома своєю портативністю, що досягається за допомогою Java Virtual Machine, яка дозволяє запускати Java-застосунки на будь-якому пристрої, де встановлена JVM [14]. Основною ідеєю Java є принцип «Write Once, Run Anywhere» (WORA), що означає, що код, написаний на Java, може бути виконаний на будь-якій платформі без необхідності його перекомпіляції.

Java забезпечує високу безпеку за рахунок кількох рівнів перевірок, виконуваних під час компіляції, завантаження класів і виконання. Наприклад, вона має вбудовану систему управління пам'яттю, яка допомагає уникати витоків пам'яті та збоїв через переповнення [15]. Крім того, Java надає потужні механізми обробки виключень, що дозволяють ефективно керувати помилками.

Об'єктно-орієнтований підхід Java означає, що вона підтримує створення програм із використанням об'єктів і класів. Це сприяє кращій структурованості коду, його повторному використанню та полегшує обслуговування. В Java все є об'єктом, що дозволяє використовувати потужні механізми наслідування та поліморфізму для створення гнучких і масштабованих програм.

Java має величезну стандартну бібліотеку Java SE, яка надає безліч готових до використання класів і методів для виконання різноманітних завдань: від роботи з файлами і мережевими підключеннями до розробки графічних інтерфейсів користувача (GUI). Крім стандартної бібліотеки, існує також багато сторонніх бібліотек і фреймворків, які розширюють можливості Java і дозволяють розробляти програми будь-якої складності.

Завдяки JVM Java є мовою з байт-кодом, що робить її незалежною від платформи. Це означає, що байт-код, згенерований компілятором Java, може виконуватися на будь-якій машині з установленою JVM, незалежно від апаратного забезпечення та операційної системи. Це робить Java ідеальним вибором для розробки корпоративних застосунків, вебзастосунків, мобільних застосунків (особливо для Android), а також серверних і клієнтських програм.

Java також підтримує багатонитевість, що дозволяє створювати багатозадачні програми, які можуть виконувати кілька завдань одночасно. Це особливо важливо для серверних застосунків і програм, що працюють у реальному часі. Крім того, Java має потужні засоби для мережевого програмування, що дозволяє легко створювати мережеві застосунки і клієнт-серверні системи [16].

Java широко використовується в освітніх установах для навчання програмуванню завдяки своїй простоті, строгості синтаксису і наявності великої кількості навчальних матеріалів. Це дозволяє новачкам швидко освоїти основні концепції програмування та приступити до розробки власних програм.

Однією з найважливіших особливостей Java є її велика та активна спільнота розробників [17]. Це означає, що завжди можна знайти допомогу, відповіді на питання або готові рішення для своїх завдань. Існує безліч форумів, блогів, конференцій і курсів, присвячених Java, що робить її доступною для вивчення і використання.

З моменту свого створення Java зазнала багатьох змін і покращень. Останні версії мови включають нові функції, оптимізації та інструменти, що робить її ще більш потужною і зручною для розробників [18]. Серед цих нововведень можна виділити лямбда-вирази, Streams API, модульну систему і багато іншого, що дозволяє створювати сучасні, високопродуктивні та масштабовані програми.

У підсумку, Java залишається однією з найпопулярніших і найбільш використовуваних мов програмування у світі. Її універсальність, безпека, продуктивність і багатофункціональність роблять її відмінним вибором для розробки програмного забезпечення будь-якої складності і для будь-яких галузей. Завдяки постійним оновленням і підтримці спільноти, Java продовжує залишатися

актуальною і затребуваною мовою програмування, забезпечуючи розробників потужними інструментами для створення якісних і надійних застосунків.

Spring Framework — це популярний і потужний фреймворк для розробки Java-застосунків, створений для спрощення процесу розробки складних корпоративних систем. Він був розроблений Рідом Джонсоном і з'явився у 2003 році [19]. З тих пір Spring став стандартом де-факто для розробки застосунків на платформі Java, завдяки своїй гнучкості, масштабованості та можливості легкої інтеграції з іншими технологіями.

Однією з основних концепцій Spring є Інверсія керування (Inversion of Control, IoC), яка полягає в тому, що контейнер Spring відповідає за управління життєвим циклом об'єктів і їхніми залежностями [20]. Це дозволяє знизити зв'язність між компонентами програми і робить їх більш гнучкими та легкими для тестування. Замість того, щоб об'єкти самостійно створювали свої залежності, контейнер Spring впроваджує ці залежності автоматично, використовуючи техніку під назвою Dependency Injection.

Ще однією важливою особливістю Spring є підтримка аспектно-орієнтованого програмування. AOP дозволяє розділяти різні аспекти програми, такі як логування, управління транзакціями, безпека і кешування, від основної бізнес-логіки. Це дозволяє зменшити дублювання коду і підвищити його структурованість. У Spring AOP використовується для впровадження додаткових поведінок в існуючі класи без необхідності змінювати їхній код.

Spring складається з кількох модулів, кожен з яких можна використовувати окремо або в комбінації з іншими. Наприклад, Spring MVC призначений для розробки вебзастосунків і надає потужні інструменти для роботи з HTTP-запитами, обробки форм, валідації даних і генерації відповіді. Spring Data спрощує роботу з базами даних, надаючи абстракції для взаємодії з різними системами зберігання даних, такими як реляційні бази даних, NoSQL бази даних і системи управління великими даними [21]. Spring Security надає засоби для управління безпекою застосунків, включаючи аутентифікацію, авторизацію і захист від атак.

Однією з причин популярності Spring є його легка інтеграція з іншими технологіями та фреймворками. Наприклад, Spring добре працює з ORM-фреймворками, такими як Hibernate і JPA, що дозволяє розробникам легко управляти базами даних і виконувати складні запити без необхідності написання великої кількості SQL-коду. Крім того, Spring підтримує інтеграцію з іншими популярними фреймворками, такими як JMS для обміну повідомленнями і RMI для виклику віддалених методів [22].

Spring також має потужну підтримку тестування, що дозволяє розробникам писати юніт-тести і інтеграційні тести для своїх застосунків. Spring TestContext Framework надає інструменти для налаштування тестових контекстів, управління транзакціями під час тестування і виконання тестів у різних середовищах.

Завдяки своїй модульності і гнучкості, Spring може використовуватися для розробки різноманітних застосунків, від невеликих вебсайтів до великих корпоративних систем. Він підтримує різні архітектурні стилі, включаючи мікросервіси, які стають все більш популярними завдяки своїй здатності покращувати масштабованість і надійність застосунків. Spring Cloud — це набір інструментів, розроблених для полегшення розробки мікросервісів, що дозволяє керувати конфігурацією, обробкою збоїв, балансуванням навантаження, маршрутизацією запитів і багатьма іншими аспектами розподілених систем.

Spring також підтримує сучасні підходи до розробки, такі як DevOps і безперервна інтеграція/безперервне постачання [23]. Spring Boot, наприклад, спрощує процес налаштування і розгортання застосунків, забезпечуючи готові до використання конфігурації і зменшуючи кількість необхідного конфігураційного коду. Це дозволяє розробникам швидко створювати та розгортати готові до виробництва застосунки, мінімізуючи час, витрачений на налаштування інфраструктури.

У підсумку, Spring Framework залишається одним з найпопулярніших і найбільш використовуваних фреймворків для розробки Java-застосунків. Його гнучкість, модульність і потужні інструменти роблять його ідеальним вибором для розробки застосунків будь-якої складності, від невеликих вебсайтів до великих

корпоративних систем. Завдяки постійним оновленням і активній підтримці спільноти, Spring продовжує залишатися актуальним і затребуваним фреймворком, забезпечуючи розробників потужними інструментами для створення якісних і надійних застосунків.

MySQL — це відома система управління реляційними базами даних, яка використовує SQL для доступу і управління даними. Вона була розроблена шведською компанією MySQL AB і нині підтримується Oracle [24]. З моменту свого створення MySQL стала однією з найпопулярніших баз даних у світі завдяки своїй продуктивності, надійності і простоті використання.

MySQL оптимізована для високої продуктивності і може обробляти великі обсяги даних швидко та ефективно. Вона використовує різноманітні механізми зберігання даних, такі як InnoDB і MyISAM, які забезпечують різні рівні продуктивності, надійності та функціональності. InnoDB, наприклад, підтримує транзакції, зберігає дані у вигляді кластеризованих індексів і забезпечує високу цілісність даних завдяки використанню ACID-властивостей (Atomicity, Consistency, Isolation, Durability) [25]. MyISAM, з іншого боку, забезпечує високу швидкість читання і написання даних, але не підтримує транзакції.

Однією з головних переваг MySQL є її масштабованість. Вона може використовуватися як для невеликих застосунків, так і для великих, високонавантажених систем. MySQL підтримує реплікацію, що дозволяє створювати резервні копії баз даних і розподіляти навантаження між кількома серверами. Це дозволяє забезпечити високу доступність і надійність системи, а також покращити продуктивність за рахунок розподілу навантаження.

MySQL є багатоплатформовою системою, що означає, що вона може працювати на різних операційних системах, включаючи Windows, Linux, MacOS і багато інших. Це забезпечує гнучкість і зручність використання, оскільки розробники можуть вибирати платформу, яка найкраще відповідає їхнім потребам.

MySQL підтримує різні типи даних, включаючи числові, символьні, часові та інші, що дозволяє зберігати і обробляти різноманітні дані. Вона також підтримує складні SQL-запити, включаючи вкладені запити, об'єднання таблиць,

групування даних і багато інших функцій, що дозволяє виконувати складні операції з даними і отримувати необхідну інформацію швидко і ефективно.

Безпека є ще однією важливою перевагою MySQL. Вона забезпечує кілька рівнів захисту, включаючи управління доступом на основі ролей, шифрування даних, аутентифікацію користувачів і багато іншого. Це дозволяє забезпечити надійний захист даних і запобігти несанкціонованому доступу до бази даних.

MySQL має велику і активну спільноту користувачів і розробників, що означає наявність безлічі ресурсів, документації і підтримки. Це дозволяє швидко знаходити відповіді на питання, вирішувати проблеми і отримувати поради щодо оптимального використання системи. Крім того, існує безліч сторонніх інструментів і програмного забезпечення, які розширюють можливості MySQL і дозволяють легко інтегрувати її з іншими системами і застосунками [26].

MySQL також підтримує процедури, тригери і події, що дозволяє автоматизувати різні завдання і підвищити продуктивність застосунків. Процедури дозволяють виконувати складні операції з даними, тригери автоматично виконують певні дії при настанні певних подій, а події дозволяють планувати виконання завдань у майбутньому. Це робить MySQL потужним інструментом для розробки складних і багатофункціональних застосунків.

Завдяки своїй продуктивності, надійності і масштабованості, MySQL широко використовується в різних галузях, включаючи веброзробку, електронну комерцію, аналітику даних, фінансові системи і багато інших. Вона є основою для багатьох популярних вебзастосунків і платформ, таких як WordPress, Drupal, Joomla, Magento і багато інших. MySQL також використовується великими компаніями і організаціями, включаючи Facebook, Twitter, YouTube, Google і багато інших, що свідчить про її надійність і ефективність.

У підсумку, MySQL є однією з найпопулярніших і найнадійніших систем управління реляційними базами даних у світі. Її продуктивність, масштабованість, безпека і багатофункціональність роблять її відмінним вибором для розробки застосунків будь-якої складності і для будь-яких галузей. Завдяки активній спільноті, постійним оновленням і підтримці з боку Oracle, MySQL продовжує

залишатися актуальною і затребуваною системою управління базами даних, забезпечуючи розробників потужними інструментами для зберігання, управління і обробки даних.

3.2 Опис бази даних

Дані вебзастосунку зберігаються у базі даних MySQL, фізична модель представлена на рис. 3.1.

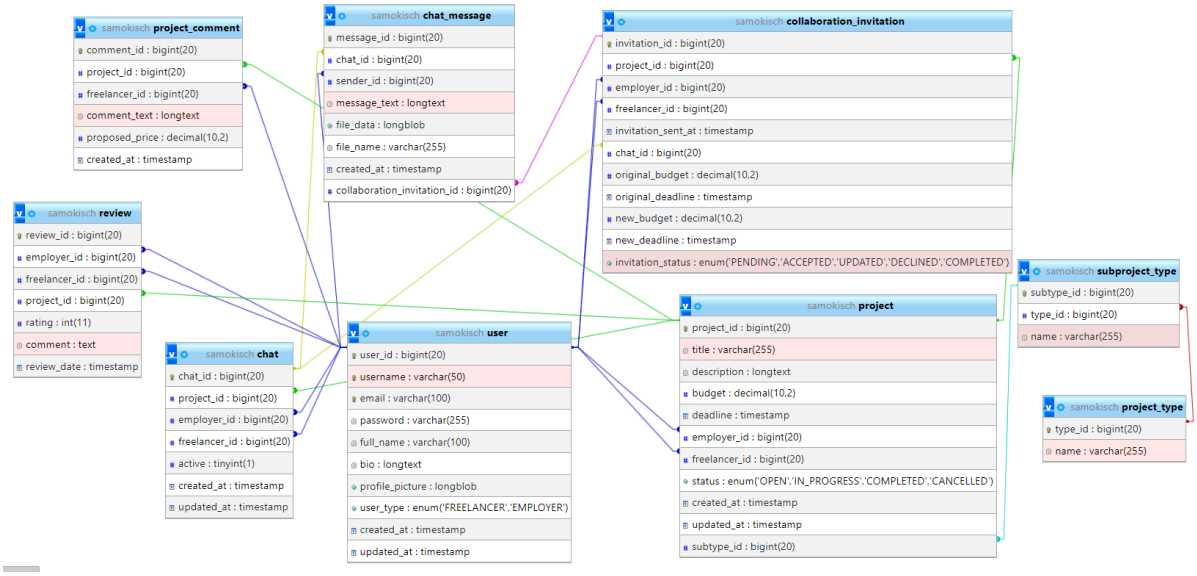


Рисунок 3.1 – Фізична модель бази даних вебзастосунку

Далі розглянемо кожну таблицю окремо.

Таблиця chat містить інформацію про чати, пов'язані з проєктами. Вона включає такі поля:

– chat_id (BIGINT, AUTO_INCREMENT, PRIMARY KEY): унікальний ідентифікатор чату;

– project_id (BIGINT, NOT NULL): ідентифікатор проєкту, до якого належить чат;

– employer_id (BIGINT, NOT NULL): ідентифікатор роботодавця, який бере участь у чаті;

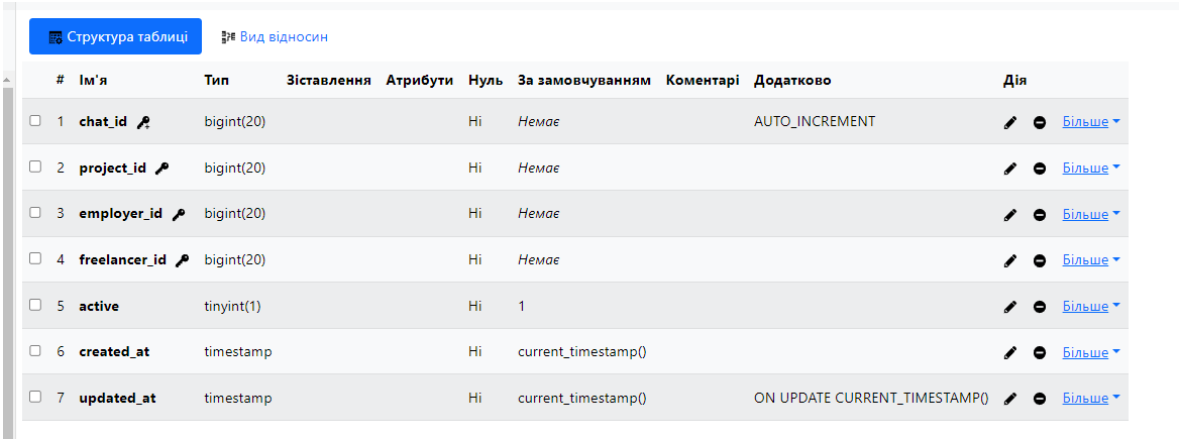
– freelancer_id (BIGINT, NOT NULL): ідентифікатор фрілансера, який бере участь у чаті;

– active (BOOLEAN, NOT NULL, DEFAULT TRUE): активність чату;

– created_at (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP, NOT NULL): час створення чату;

– updated_at (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP, NOT NULL): час останнього оновлення чату.

Ця таблиця забезпечує зберігання інформації про чати між роботодавцями і фрілансерами щодо проєктів. Структуру таблиці представлено на рис. 3.2.


Структура таблиці | Вид відносин

#	Ім'я	Тип	Зіставлення	Атрибути	Нуль	За замовчуванням	Коментарі	Додатково	Дія
1	chat_id	bigint(20)			Ні	*Немає*		AUTO_INCREMENT	Більше
2	project_id	bigint(20)			Ні	*Немає*			Більше
3	employer_id	bigint(20)			Ні	*Немає*			Більше
4	freelancer_id	bigint(20)			Ні	*Немає*			Більше
5	active	tinyint(1)			Ні	1			Більше
6	created_at	timestamp			Ні	current_timestamp()			Більше
7	updated_at	timestamp			Ні	current_timestamp()		ON UPDATE CURRENT_TIMESTAMP()	Більше

Рисунок 3.2 – Структура таблиці чату

Таблиця chat_message зберігає повідомлення у чатах. Вона включає такі поля:

– message_id (BIGINT, AUTO_INCREMENT, PRIMARY KEY): унікальний ідентифікатор повідомлення;

– chat_id (BIGINT, NOT NULL): ідентифікатор чату, до якого належить повідомлення;

– sender_id (BIGINT, NOT NULL): ідентифікатор користувача, який відправив повідомлення;

– message_text (LONGTEXT): текст повідомлення;

– file_data (LONGBLOB): додаткові файли у повідомленні;

– file_name (VARCHAR(255)): ім'я файлу;

– created_at (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP, NOT NULL): час створення повідомлення;

– collaboration_invitation_id (BIGINT): ідентифікатор запрошення до співпраці, пов'язаного з повідомленням.

Ця таблиця дозволяє зберігати історію повідомлень у чатах, включаючи текстові повідомлення та вкладення файлів. Структуру таблиці представлено на рис. 3.3.

Структура таблиці | Вид відносин

#	Ім'я	Тип	Зіставлення	Атрибути	Нуль	За замовчуванням	Коментарі	Додатково	Дія
1	message_id	bigint(20)			Ні	*Немає*		AUTO_INCREMENT	Більше
2	chat_id	bigint(20)			Ні	*Немає*			Більше
3	sender_id	bigint(20)			Ні	*Немає*			Більше
4	message_text	longtext	*utf8mb4_unicode_ci*		Так	*NULL*			Більше
5	file_data	longblob			Так	*NULL*			Більше
6	file_name	varchar(255)	*utf8mb4_unicode_ci*		Так	*NULL*			Більше
7	created_at	timestamp			Ні	current_timestamp()			Більше
8	collaboration_invitation_id	bigint(20)			Так	*NULL*			Більше

Рисунок 3.3 – Структура таблиці повідомлень у чаті

Таблиця collaboration_invitation містить інформацію про запрошення до співпраці. Вона включає такі поля:

– invitation_id (BIGINT, AUTO_INCREMENT, PRIMARY KEY): унікальний ідентифікатор запрошення;

– project_id (BIGINT, NOT NULL): ідентифікатор проєкту, для якого створено запрошення;

– employer_id (BIGINT, NOT NULL): ідентифікатор роботодавця, який відправив запрошення;

– freelancer_id (BIGINT, NOT NULL): ідентифікатор фрілансера, який отримав запрошення;

– invitation_sent_at (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP, NOT NULL): час відправлення запрошення;

– chat_id (BIGINT): ідентифікатор чату, пов'язаного з запрошенням;

– original_budget (DECIMAL(10, 2), NOT NULL): початковий бюджет проєкту;

– original_deadline (TIMESTAMP, NOT NULL): початковий крайній термін проєкту;

– new_budget (DECIMAL(10, 2), NOT NULL): новий запропонований бюджет;

– new_deadline (TIMESTAMP): новий крайній термін;

– invitation_status (ENUM('PENDING', 'ACCEPTED', 'UPDATED', 'DECLINED', 'COMPLETED'), NOT NULL, DEFAULT 'PENDING'): статус запрошення.

Ця таблиця дозволяє роботодавцям відправляти запрошення фрілансерам для співпраці над проєктами, зберігаючи інформацію про бюджет, терміни та статуси запрошень. Структуру таблиці представлено на рис. 3.4.

Структура таблиці | Вид відносин

#	Ім'я	Тип	Зіставлення	Атрибути	Нуль	За замовчуванням	Коментарі	Додатково	Дія
1	invitation_id	bigint(20)			Ні	*Немає*		AUTO_INCREMENT	Більше
2	project_id	bigint(20)			Ні	*Немає*			Більше
3	employer_id	bigint(20)			Ні	*Немає*			Більше
4	freelancer_id	bigint(20)			Ні	*Немає*			Більше
5	invitation_sent_at	timestamp			Ні	current_timestamp()			Більше
6	chat_id	bigint(20)			Так	*NULL*			Більше
7	original_budget	decimal(10,2)			Ні	*Немає*			Більше
8	original_deadline	timestamp			Ні	*Немає*			Більше
9	new_budget	decimal(10,2)			Ні	*Немає*			Більше
10	new_deadline	timestamp			Так	*NULL*			Більше
11	invitation_status	enum('PENDING', 'ACCEPTED', 'UPDATED', 'DECLINED',...	utf8mb4_unicode_ci		Ні	PENDING			Більше

Рисунок 3.4 – Структура таблиці запрошень

Таблиця project містить інформацію про проєкти. Вона включає такі поля:

– project_id (BIGINT, AUTO_INCREMENT, PRIMARY KEY): унікальний ідентифікатор проєкту;

– title (VARCHAR(255), NOT NULL): назва проєкту;

– description (LONGTEXT): опис проєкту;

– budget (DECIMAL(10, 2)): бюджет проєкту;

– deadline (TIMESTAMP): крайній термін виконання проєкту;

– employer_id (BIGINT): ідентифікатор роботодавця, який створив проєкт;

– freelancer_id (BIGINT): ідентифікатор фрілансера, який виконує проєкт;

– status (ENUM('OPEN', 'IN_PROGRESS', 'COMPLETED', 'CANCELLED'), DEFAULT 'OPEN', NOT NULL): статус проєкту;

– created_at (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP, NOT NULL): час створення проєкту;

– updated_at (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP, NOT NULL): час останнього оновлення проєкту;

– subtype_id (BIGINT): ідентифікатор підтипу проєкту.

Ця таблиця зберігає інформацію про різні проєкти, включаючи їхні назви, описи, бюджети, статуси та асоційованих користувачів (роботодавців і фрілансерів). Структуру таблиці представлено на рис. 3.5.

Структура таблиці | Вид відносин

#	Ім'я	Тип	Зіставлення	Атрибути	Нуль	За замовчуванням	Коментарі	Додатково	Дія
1	project_id	bigint(20)			Ні	*Немає*		AUTO_INCREMENT	Більше
2	title	varchar(255)	*utf8mb4_unicode_ci*		Ні	*Немає*			Більше
3	description	longtext	*utf8mb4_unicode_ci*		Так	*NULL*			Більше
4	budget	decimal(10,2)			Так	*NULL*			Більше
5	deadline	timestamp			Так	*NULL*			Більше
6	employer_id	bigint(20)			Так	*NULL*			Більше
7	freelancer_id	bigint(20)			Так	*NULL*			Більше
8	status	enum('OPEN', 'IN_PROGRESS', 'COMPLETED', 'CANCELLE...	*utf8mb4_unicode_ci*		Ні	OPEN			Більше
9	created_at	timestamp			Ні	current_timestamp()			Більше
10	updated_at	timestamp			Ні	current_timestamp()		ON UPDATE CURRENT_TIMESTAMP()	Більше
11	subtype_id	bigint(20)			Так	*NULL*			Більше

Рисунок 3.5 – Структура таблиці проєктів

Таблиця project_comment зберігає коментарі до проєктів. Вона включає такі поля:

– comment_id (BIGINT, AUTO_INCREMENT, PRIMARY KEY): унікальний ідентифікатор коментаря;

– project_id (BIGINT, NOT NULL): ідентифікатор проєкту, до якого належить коментар;

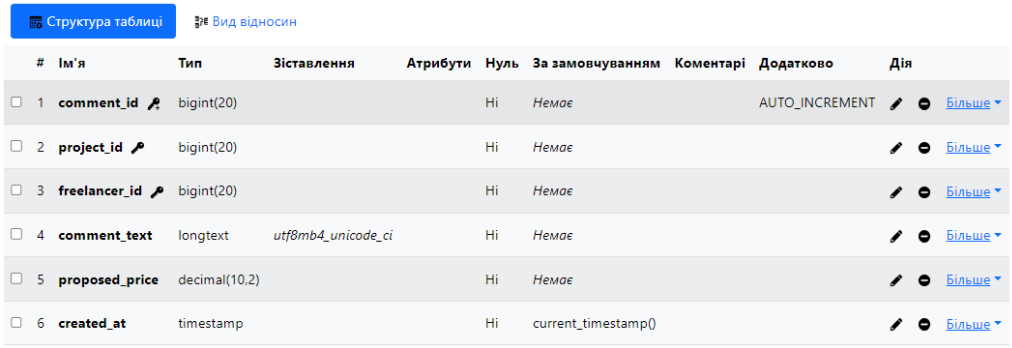
– freelancer_id (BIGINT, NOT NULL): ідентифікатор фрілансера, який залишив коментар;

– comment_text (LONGTEXT, NOT NULL): текст коментаря;

– proposed_price (DECIMAL(10, 2), NOT NULL): запропонована ціна фрілансера;

– created_at (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP, NOT NULL): час створення коментаря.

Ця таблиця зберігає коментарі до проєктів, дозволяючи фрілансерам пропонувати свої ціни та обговорювати деталі проєктів. Структуру таблиці представлено на рис. 3.6.


Структура таблиці | Вид відносин

#	Ім'я	Тип	Зіставлення	Атрибути	Нуль	За замовчуванням	Коментарі	Додатково	Дія
1	comment_id	bigint(20)			Ні	Немає		AUTO_INCREMENT	Більше
2	project_id	bigint(20)			Ні	Немає			Більше
3	freelancer_id	bigint(20)			Ні	Немає			Більше
4	comment_text	longtext	utf8mb4_unicode_ci		Ні	Немає			Більше
5	proposed_price	decimal(10,2)			Ні	Немає			Більше
6	created_at	timestamp			Ні	current_timestamp()			Більше

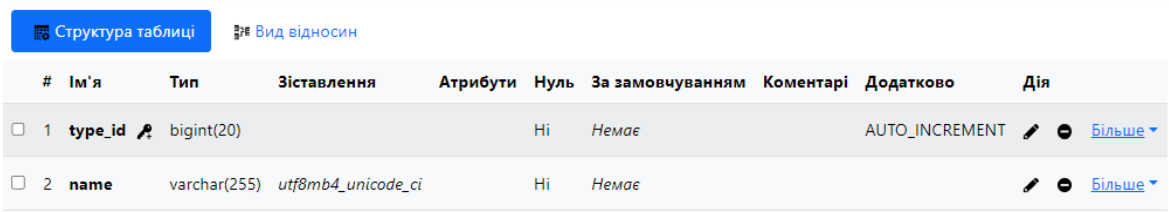
Рисунок 3.6 – Структура таблиці коментарів

Таблиця project_type містить інформацію про типи проєктів. Вона включає такі поля:

– type_id (BIGINT, AUTO_INCREMENT, PRIMARY KEY): унікальний ідентифікатор типу проєкту;

– name (VARCHAR(255), NOT NULL): назва типу проєкту.

Ця таблиця використовується для класифікації проєктів за різними типами. Структуру таблиці представлено на рис. 3.7.


Структура таблиці | Вид відносин

#	Ім'я	Тип	Зіставлення	Атрибути	Нуль	За замовчуванням	Коментарі	Додатково	Дія
1	type_id	bigint(20)			Ні	Немає		AUTO_INCREMENT	Більше
2	name	varchar(255)	utf8mb4_unicode_ci		Ні	Немає			Більше

Рисунок 3.7 – Структура таблиці типів

Таблиця review містить відгуки про виконані проєкти. Вона включає такі поля:

– review_id (BIGINT, AUTO_INCREMENT, PRIMARY KEY): унікальний ідентифікатор відгуку;

– employer_id (BIGINT, NOT NULL): ідентифікатор роботодавця, який залишив відгук;

– freelancer_id (BIGINT, NOT NULL): ідентифікатор фрілансера, про якого залишено відгук;

– project_id (BIGINT, NOT NULL): ідентифікатор проєкту, про який залишено відгук;

– rating (INT, NOT NULL, CHECK (rating >= 1 AND rating <= 5)): оцінка проєкту (від 1 до 5);

– comment (TEXT): текстовий коментар відгуку;

– review_date (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP, NOT NULL): час створення відгуку.

Ця таблиця дозволяє зберігати відгуки про проєкти, що допомагає оцінювати якість роботи фрілансерів і досвід співпраці з роботодавцями. Структуру таблиці представлено на рис. 3.8.

Структура таблиці | Вид відносин

#	Ім'я	Тип	Зіставлення	Атрибути	Нуль	За замовчуванням	Коментарі	Додатково	Дія
1	review_id	bigint(20)			Ні	Немає		AUTO_INCREMENT	Більше
2	employer_id	bigint(20)			Ні	Немає			Більше
3	freelancer_id	bigint(20)			Ні	Немає			Більше
4	project_id	bigint(20)			Ні	Немає			Більше
5	rating	int(11)			Ні	Немає			Більше
6	comment	text	utf8mb4_unicode_ci		Так	NULL			Більше
7	review_date	timestamp			Ні	current_timestamp()			Більше

Рисунок 3.8 – Структура таблиці відгуків

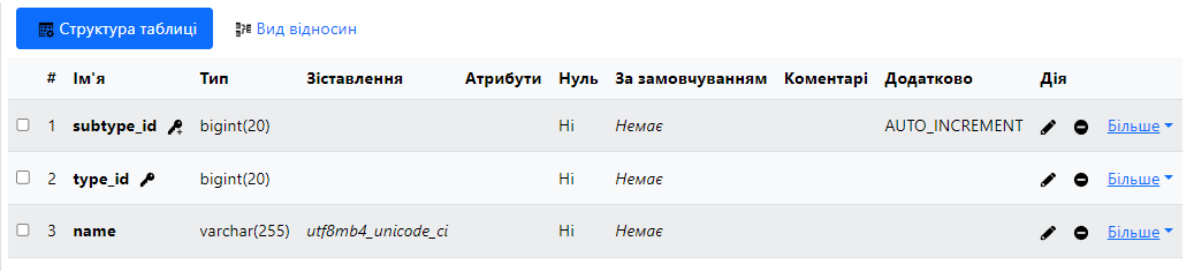
Таблиця subproject_type містить інформацію про підтипи проєктів. Вона включає такі поля:

– subtype_id (BIGINT, AUTO_INCREMENT, PRIMARY KEY): унікальний ідентифікатор підтипу проєкту;

– type_id (BIGINT, NOT NULL): ідентифікатор типу проєкту;

– name (VARCHAR(255), NOT NULL): назва підтипу проєкту.

Ця таблиця дозволяє класифікувати проєкти за підтипами в межах основних типів проєктів. Структуру таблиці представлено на рис. 3.9.


Структура таблиці | Вид відносин

#	Ім'я	Тип	Зіставлення	Атрибути	Нуль	За замовчуванням	Коментарі	Додатково	Дія
1	subtype_id	bigint(20)			Ні	*Немає*		AUTO_INCREMENT	Більше
2	type_id	bigint(20)			Ні	*Немає*			Більше
3	name	varchar(255)	*utf8mb4_unicode_ci*		Ні	*Немає*			Більше

Рисунок 3.9 – Структура таблиці підтипів

Таблиця user зберігає інформацію про користувачів системи. Вона містить такі поля:

– user_id (BIGINT, AUTO_INCREMENT, PRIMARY KEY): унікальний ідентифікатор користувача;

– username (VARCHAR(50), NOT NULL, UNIQUE): унікальне ім'я користувача;

– email (VARCHAR(100), NOT NULL, UNIQUE): унікальна електронна адреса користувача;

– password (VARCHAR(255), NOT NULL): зашифрований пароль користувача;

– full_name (VARCHAR(100)): повне ім'я користувача;

– bio (LONGTEXT): Біографія користувача;

– profile_picture (LONGBLOB): зображення профілю користувача;

– user_type (ENUM('FREELANCER', 'EMPLOYER'), NOT NULL): тип користувача (фрілансер або роботодавець) ;

– created_at (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP): час створення запису;

– updated_at (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP): час останнього оновлення запису.

Ця таблиця забезпечує зберігання основної інформації про користувачів, включаючи їхні ідентифікаційні дані, облікові записи та додаткову інформацію, що дозволяє класифікувати користувачів за типами. Структуру таблиці представлено на рис. 3.10.

Структура таблиці | Вид відносин

#	Ім'я	Тип	Зіставлення	Атрибути	Нуль	За замовчуванням	Коментарі	Додатково	Дія
1	user_id	bigint(20)			Ні	Немає		AUTO_INCREMENT	Більше
2	username	varchar(50)	utf8mb4_unicode_ci		Ні	Немає			Більше
3	email	varchar(100)	utf8mb4_unicode_ci		Ні	Немає			Більше
4	password	varchar(255)	utf8mb4_unicode_ci		Ні	Немає			Більше
5	full_name	varchar(100)	utf8mb4_unicode_ci		Так	NULL			Більше
6	bio	longtext	utf8mb4_unicode_ci		Так	NULL			Більше
7	profile_picture	longblob			Так	NULL			Більше
8	user_type	enum('FREELANCER', 'EMPLOYER')	utf8mb4_unicode_ci		Ні	Немає			Більше
9	created_at	timestamp			Так	current_timestamp()			Більше
10	updated_at	timestamp			Так	current_timestamp()		ON UPDATE CURRENT_TIMESTAMP()	Більше

Рисунок 3.10 – Структура таблиці користувачів

Таблиця user містить усю необхідну інформацію для управління користувачами та їхніми профілями в системі.

3.3 Розробка мокапу системи

Мокап системи - це візуальна модель або прототип інформаційної системи, що відображає її інтерфейс, функціональність та взаємодію з користувачем. Застосунок включатиме такі вікна як:

– «Головна»;
– «Сторінка каталогу вакансій»;
– «Сторінка створення вакансії»;
– «Форма реєстрації замовника»;
– «Форма авторизації користувача».

Після запуску застосунку буде відкриватись головна сторінка, яка представлена на рис. 3.11.

Рисунок 3.11 – Головна сторінка

Після перегляду головної сторінки, користувач може переглянути список вакансій. Мокап сторінки каталогу вакансій представлено на рис. 3.12.

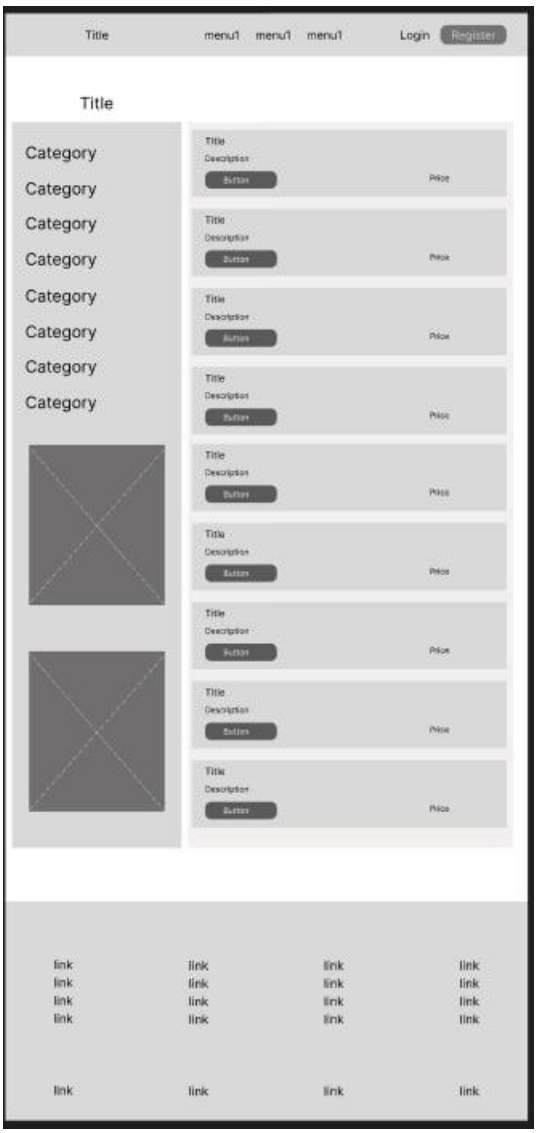


Рисунок 3.12 – Сторінка каталогу вакансій

На рис. 3.13 зображено форму авторизації, яка надає користувачеві можливість увійти в систему за допомогою своїх облікових даних. Ця форма дозволяє введення електронної адреси або імені користувача разом з паролем для підтвердження особистості.

Рисунок 3.13 – Форма авторизації користувача

На рис. 3.14 представлений макет форми, яку користувач може використати для реєстрації в системі. Ця форма містить поля для введення особистих даних, таких як ім'я, електронна адреса, пароль і контактні дані компанії.

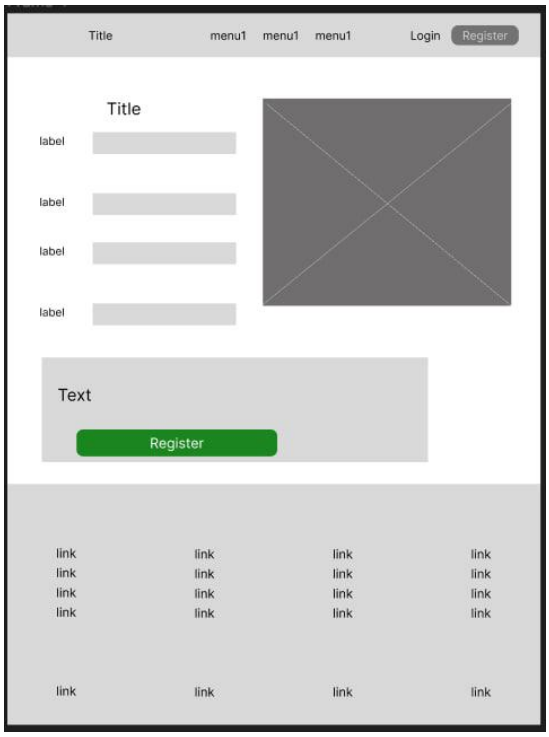


Рисунок 3.14 – Форма реєстрації користувача

На рис. 3.15 зображена сторінка, на якій роботодавець може створити нову вакансію. Ця сторінка дозволяє введення деталей вакансії, таких як назва посади, опис завдань, очікувані вимоги до кандидатів та бюджет. Після створення вакансії роботодавець може опублікувати її на платформі для залучення кандидатів.

Рисунок 3.15 – Сторінка створення вакансії

Розробка мокапів системи дозволила візуалізувати зовнішній вигляд роботи системи. Це важливо для того, щоб користувачі та зацікавлені сторони могли отримати уявлення про те, як система буде працювати в реальному середовищі.

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи представлено:

– вибір технологій розробки для системи, а саме Java, Spring Framework, React, MySQL, JavaScript. Ці технології були обрані з огляду на їхню потужність, гнучкість і популярність у розробці сучасних вебзастосунків, що забезпечує надійну і масштабовану основу для системи;

– описано структуру створеної бази даних вебзастосунку, включаючи ключові таблиці, зв'язки між ними та типи даних. Це дозволяє зрозуміти, як дані зберігаються, організовуються і взаємодіють у системі, що є критично важливим для ефективного управління даними.

Крім того, розробка мокапів системи дозволила візуалізувати зовнішній вигляд роботи системи. Це важливо для того, щоб користувачі та зацікавлені сторони могли отримати уявлення про те, як система буде працювати в реальному середовищі. Візуалізація допомагає визначити основні елементи інтерфейсу користувача, поліпшити їх зручність використання та вчасно виявити і виправити можливі проблеми з дизайном. Таким чином, мокапи слугують як інструмент комунікації між розробниками та кінцевими користувачами, сприяючи більш ефективному і точному процесу розробки.

4 РЕЗУЛЬТАТ РОЗРОБКИ ВЕБЗАСТОСУНКУ

4.1 Кодування програмного забезпечення

4.1.1 Створення сервісів аутентифікації та генерації JWT

Клас AuthenticationService забезпечує аутентифікацію користувачів та управління їхньою реєстрацією наведений у додатку А. Основні функції цього класу включають:

– реєстрація користувача (signup): метод приймає дані від користувача у вигляді об'єкта RegisterUserDto, перевіряє наявність користувача з таким самим email у базі даних і, якщо такий користувач не існує, створює новий запис у базі даних із зашифрованим паролем та іншою необхідною інформацією;

– аутентифікація користувача (authenticate): метод приймає об'єкт LoginUserDto, виконує аутентифікацію за допомогою AuthenticationManager та повертає дані користувача у випадку успішної перевірки;

– конвертація зображення в масив байтів (convertToByteArray): приватний метод для конвертації зображення профілю користувача з формату Base64 у масив байтів для зберігання в базі даних.

Клас JwtService відповідає за генерацію, валідацію та обробку JSON Web Token наведений у додатку Б. Основні функції цього класу включають:

– витягнення імені користувача (extractUsername): метод, який витягує ім'я користувача з JWT;

– генерація токенів (generateToken): методи для генерації JWT для конкретного користувача з додатковими (або без них) claims;

– перевірка дійсності токену (isTokenValid): метод перевіряє, чи є токен дійсним для даного користувача;

– перевірка, чи токен закінчився (isTokenExpired): метод перевіряє, чи закінчився термін дії токену;

– витягнення усіх claims (extractAllClaims): метод, який витягує усі claims з JWT;

– отримання ключа для підпису (getSignInKey): метод, який отримує ключ для підпису токену з секретного ключа, закодованого в Base64.

4.1.2 Створення сервісу користувача

Клас UserService забезпечує управління даними користувача, включаючи отримання профілю, редагування даних та обробку проєктів і відгуків наведений у додатку В. Основні методи цього класу:

– getUserProfileData: отримує дані профілю поточного користувача, включаючи статистику проєктів для роботодавців або список відгуків для фрілансерів. Повертає об'єкт UserProfileResponseDTO, який містить необхідну інформацію про користувача;

– getReviewList: приватний метод, який перетворює список відгуків у зручний для відображення формат UserReview, включаючи інформацію про рейтинг, коментар, дату відгуку, ім'я роботодавця, бюджет та назву проєкту;

– numberOfProjects: приватний метод, який повертає загальну кількість проєктів користувача;

– numberOfActiveProjects: приватний метод, який повертає кількість активних проєктів користувача, тобто таких, які не завершені та не скасовані;

– numberOfCompletedProjects: приватний метод, який повертає кількість завершених проєктів користувача;

– userEdit: метод для редагування профілю користувача, приймає об'єкт UserProfileEditRequestDTO, оновлює дані поточного користувача та зберігає зміни в базі даних.

4.1.3 Створення сервісу проєктів

Клас ProjectService забезпечує управління проєктами в системі, включаючи створення, редагування, видалення та отримання інформації про проєкти наведений у додатку Г. Основні методи цього класу:

– getAllForEmployer: отримує список усіх проєктів для поточного роботодавця;

– getAllOpenForEmployer: отримує список всіх відкритих проєктів для поточного роботодавця;

– getAllInProgressForEmployer: отримує список всіх проєктів у процесі для поточного роботодавця;

– getAllCompletedForEmployer: отримує список всіх завершених проєктів для поточного роботодавця;

– mappingToProjectsList: приватний метод, який мапить список проєктів у формат, зручний для відображення роботодавцем;

– create: створює новий проєкт на основі даних з ProjectCreateRequestDTO;

– getProjectForEmployerDetails: отримує детальну інформацію про проєкт для роботодавця за заданим ідентифікатором;

– getProjectDetailsForFreelancer: отримує детальну інформацію про проєкт для фрілансера за заданим ідентифікатором;

– editProject: редагує існуючий проєкт на основі даних з ProjectEditRequestDTO;

– delete: видаляє проєкт за заданим ідентифікатором;

– getAllForFreelancer: отримує список усіх відкритих проєктів для фрілансера з пагінацією;

– getFilteredProjectsForFreelancer: отримує список відфільтрованих проєктів для фрілансера з пагінацією на основі FilterDTO;

– getProjectForCollaborationInvitation: отримує інформацію про проєкт для запрошення до співпраці за заданим ідентифікатором проєкту.

4.1.4 Створення сервісу запрошень до співпраці

Клас CollaborationInvitationService забезпечує управління запрошеннями до співпраці між роботодавцями та фрілансерами в системі наведений у додатку Д. Основні методи цього класу:

– create: створює нове запрошення до співпраці на основі даних з CollaborationInvitationCreateRequestDTO;

– acceptCollaborationInvitation: приймає запрошення до співпраці за заданим ідентифікатором запрошення;

– declineInvitation: відхиляє запрошення до співпраці за заданим ідентифікатором;

– declineCollaborationInvitation: відхиляє запрошення до співпраці на основі даних з CollaborationInvitationCreateRequestDTO та створює нове запрошення зі статусом «відхилено»;

– getCollaboration: отримує інформацію про запрошення до співпраці для заданого проєкту та користувача;

– edit: редагує існуюче запрошення до співпраці на основі даних з CollaborationInvitationCreateRequestDTO;

– completedCollaborationInvitation: завершує запрошення до співпраці на основі даних з CollaborationInvitationCompletedRequestDTO;

– checkDeadline: перевіряє дедлайни для всіх поточних запрошень до співпраці поточного користувача;

– getCurrentUser: отримує поточного аутентифікованого користувача з контексту безпеки;

– getLatestInvitation: приватний метод, який повертає останнє запрошення зі списку запрошень.

4.1.5 Створення сервісу чатів та повідомлень

Клас ChatService забезпечує управління чатами між роботодавцями та фрілансерами наведений у додатку Е. Основні методи цього класу:

– getAllProjectsWithChats: отримує всі проєкти з чатами для поточного користувача, сортує їх за датою останнього оновлення;

– getAllUsersChats: отримує всі чати для конкретного проєкту, де бере участь поточний користувач, сортує їх за часом останнього повідомлення;

– getChat: отримує інформацію про конкретний чат між роботодавцем та фрілансером для заданого проєкту.

Клас ChatMessageService відповідає за керування повідомленнями в чатах між користувачами наведений у додатку Ж. Основні методи цього класу:

– chatResponseFromEmployerToComment: створює чат і додає повідомлення від роботодавця у відповідь на коментар фрілансера, базуючись на даних з ChatResponseFromEmployerToCommentRequestDTO;

– sendMessage: відправляє повідомлення в чат на основі даних з SendMessageRequestDTO, зберігаючи текстове повідомлення або файл.

4.2 Опис створеного інтерфейсу

Головна сторінка, на яку потрапляє незареєстрований або неаутентифікований користувач, представлена на рис. 4.1. Вона містить карусель зображень, та опис послуг і функцій застосунку, інформацію про компанію, відгуками користувачів, а також закликом до дії для нових користувачів.

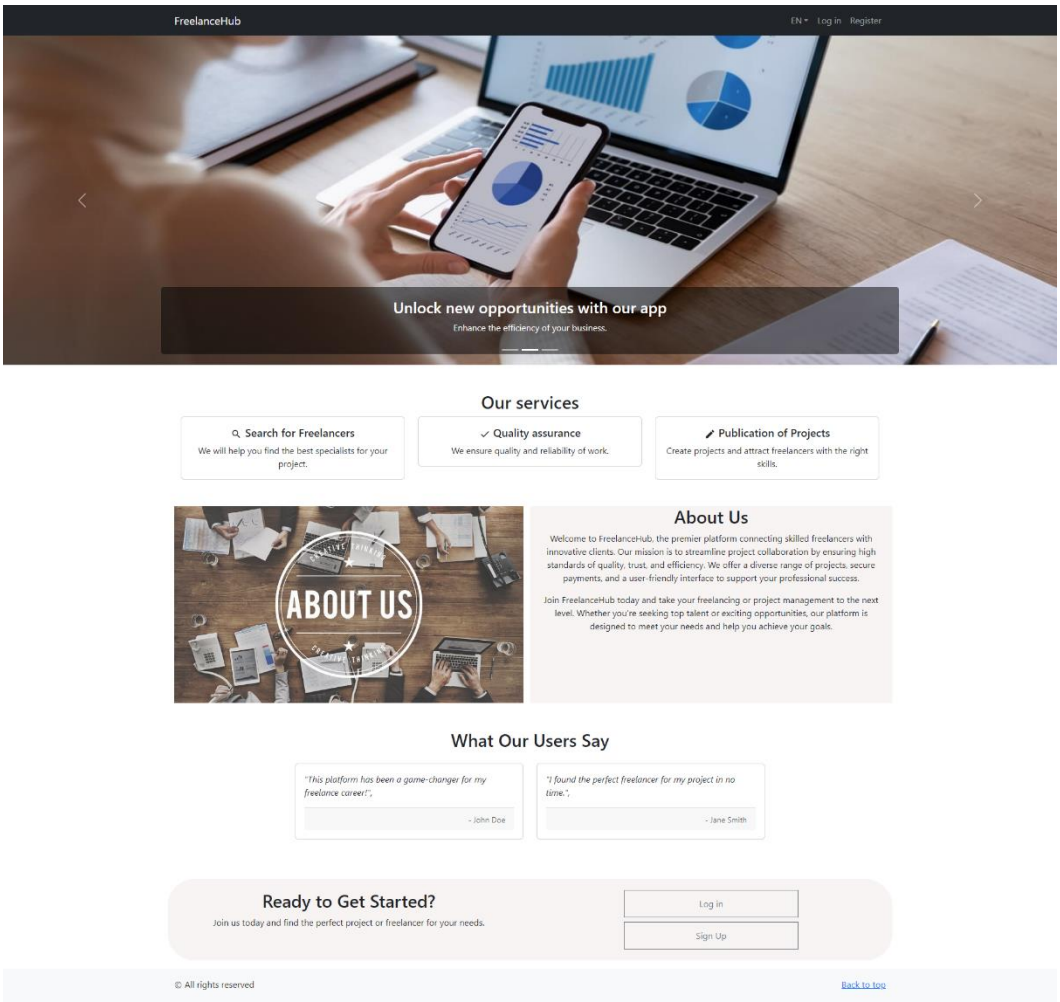


Рисунок 4.1 – Головна сторінка застосунку

На рис. 4.2 зображена сторінка з формою для реєстрації. До цієї сторінки можна потрапити через хедер або через блок із закликом до дії на головній сторінці.

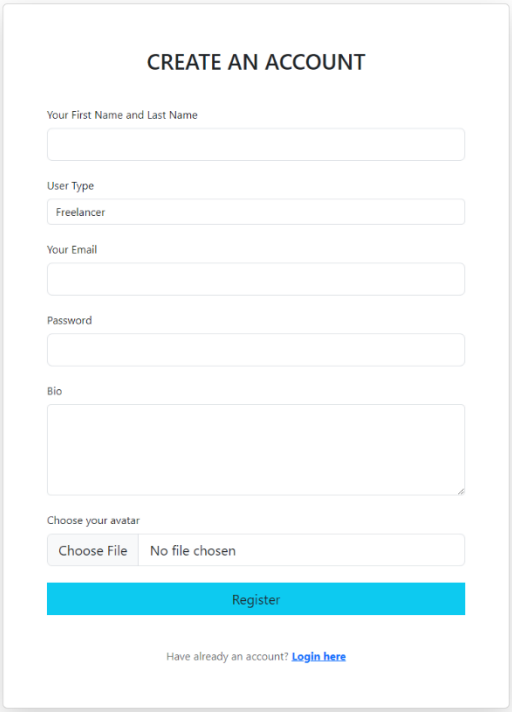


Рисунок 4.2 – Сторінка з формою реєстрації

На рис. 4.3 представлена сторінка для аутентифікації, до якої також можна потрапити через хедер або через блок із закликом до дії на головній сторінці.

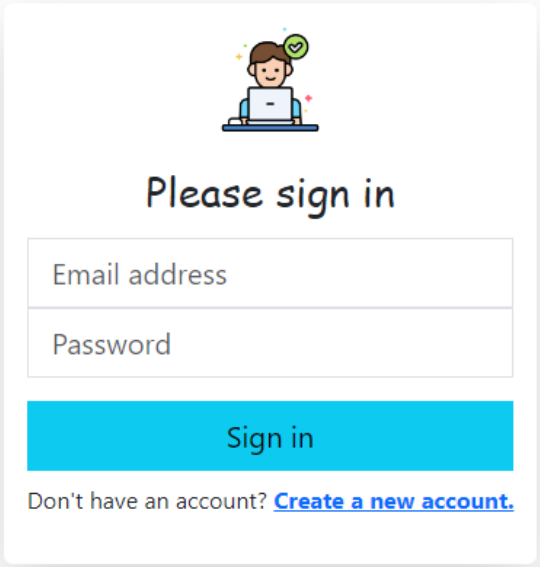


Рисунок 4.3 – Сторінка з формою аутентифікації

На рис. 4.4 зображена сторінка з переліком проєктів. На цю сторінку можна потрапити, натиснувши на кнопку «Find Jobs». На сторінці представлена форма пошуку, де користувач може ввести ключові слова для пошуку проєктів за назвою. Також є можливість застосувати різні фільтри для сортування проєктів за новизною, бюджетом або термінами виконання. Користувач може вказати мінімальний та максимальний бюджет, вибрати тип проєкту і його підтип. На сторінці відображаються проєкти з інформацією про назву, опис, термін виконання та бюджет. Кожен проєкт має кнопку для переходу на детальну сторінку з більш детальною інформацією про нього.

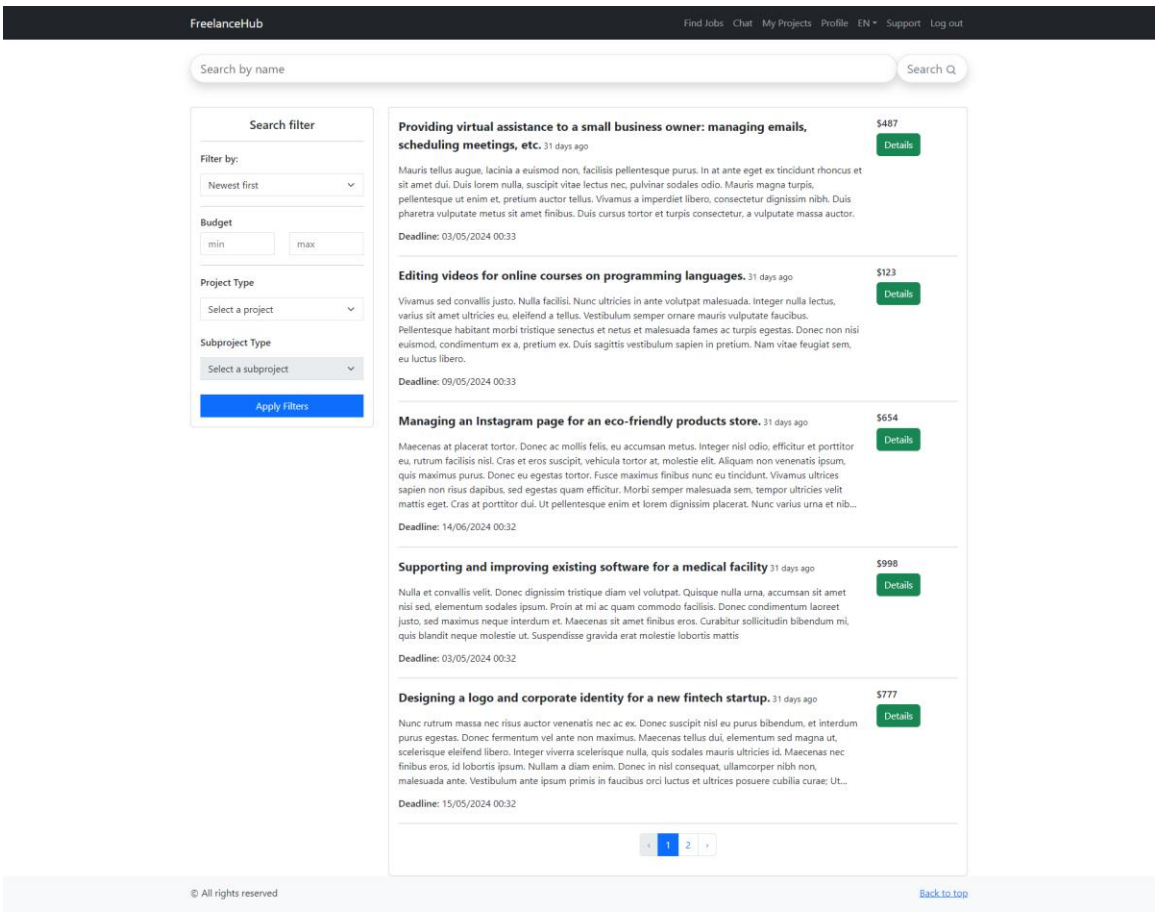


Рисунок 4.4 – Сторінка з переліком проєктів

На рис. 4.5 зображена сторінка опису проєкту. Ця сторінка містить детальну інформацію про проєкт, включаючи його назву, бюджет, тип проєкту, статус, опис, термін виконання та дату створення. собливістю є те, що для фрілансерів передбачена можливість залишити коментар до проєкту та запропонувати свою ціну, тоді як для замовників така функція відсутня. На сторінці також відображаються коментарі інших користувачів.

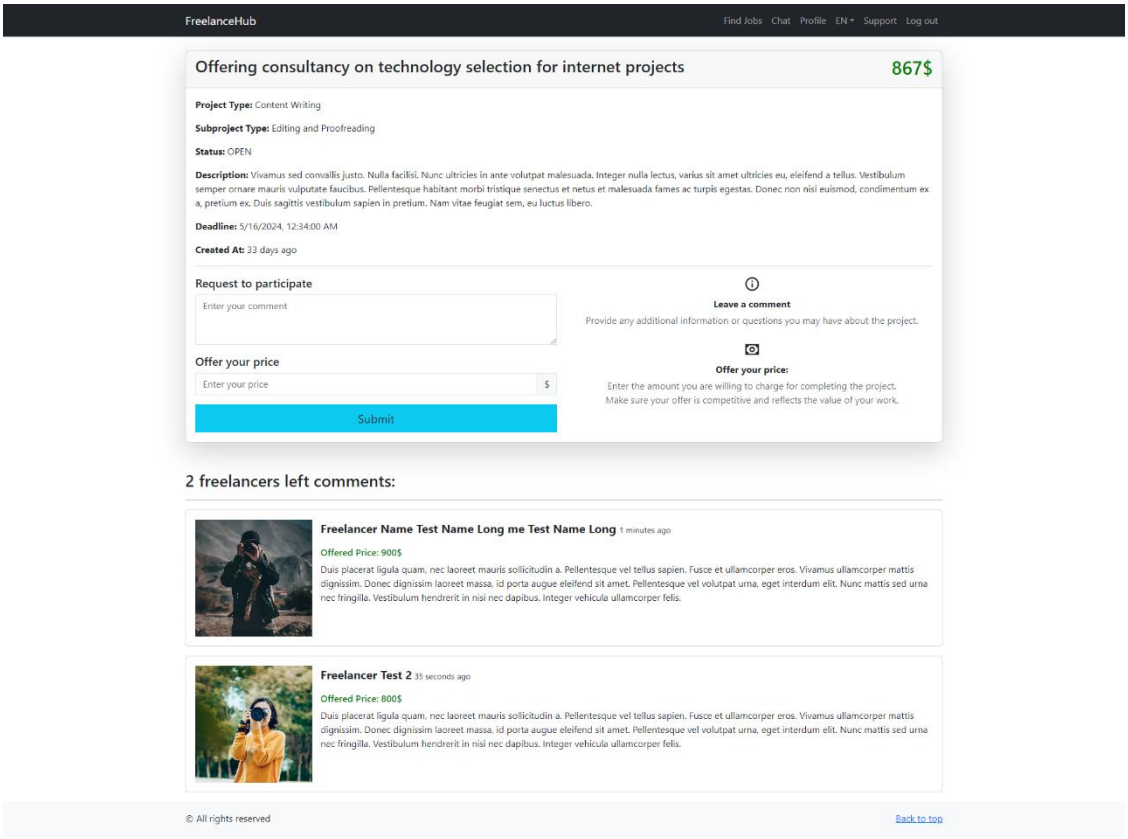


Рисунок 4.5 – Сторінка опису проєкта

На рис. 4.6 зображена сторінка профілю фрілансера. Ця сторінка включає в себе основну інформацію про користувача, таку як повне ім'я та тип аккаунту, а також фотографію профілю. Крім того, користувач може переглянути і відредагувати свій профіль за допомогою кнопки «Edit Profile». На цій сторінці також відображається короткий опис про користувача, а також показується рейтинг та коментарі від роботодавців.

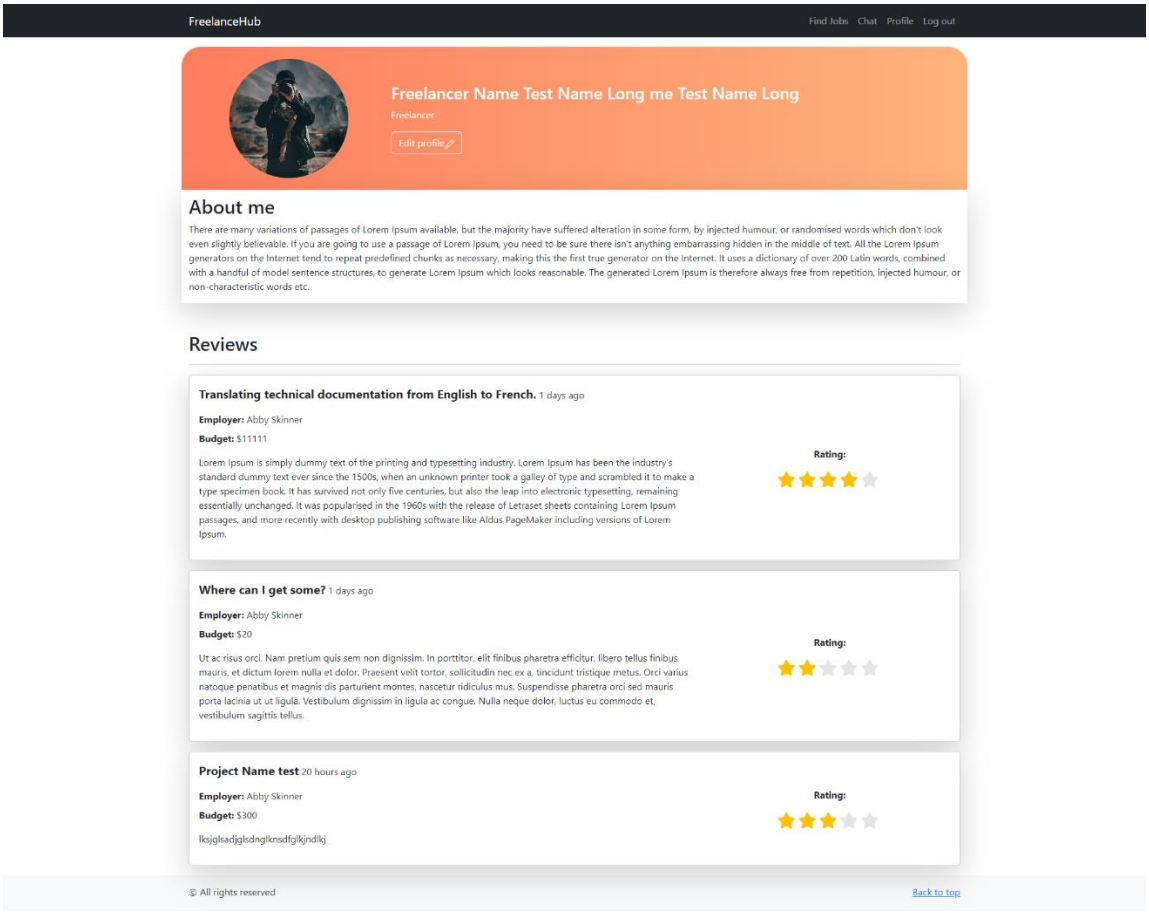


Рисунок 4.6 – Відображення сторінки профілю фрілансера

На рис. 4.7 зображена сторінка профілю роботодавця. Ця сторінка містить ту саму основну інформацію про користувача, але вказує, що він є роботодавцем. Також є можливість редагування профілю. Однак, в цьому випадку замість відгуків відображається статистика проєктів, така як загальна кількість проєктів, активні проєкти та завершені проєкти.

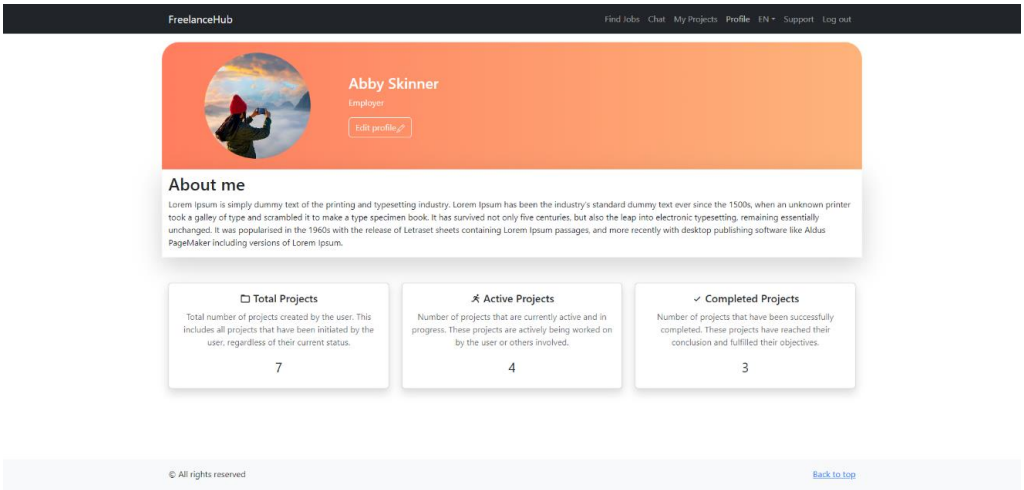


Рисунок 4.7 – Відображення сторінки профілю роботодавця

На рис. 4.8 зображена сторінка проєктів, яка доступна тільки роботодавцям. На цій сторінці роботодавець може створити новий проєкт за допомогою кнопки «Create New Project». Після цього представлений перегляд усіх проєктів у вкладках: «All», «Open», «In Progress» та «Completed». Кожна вкладка відображає відповідний стан проєктів з можливістю перегляду деталей.

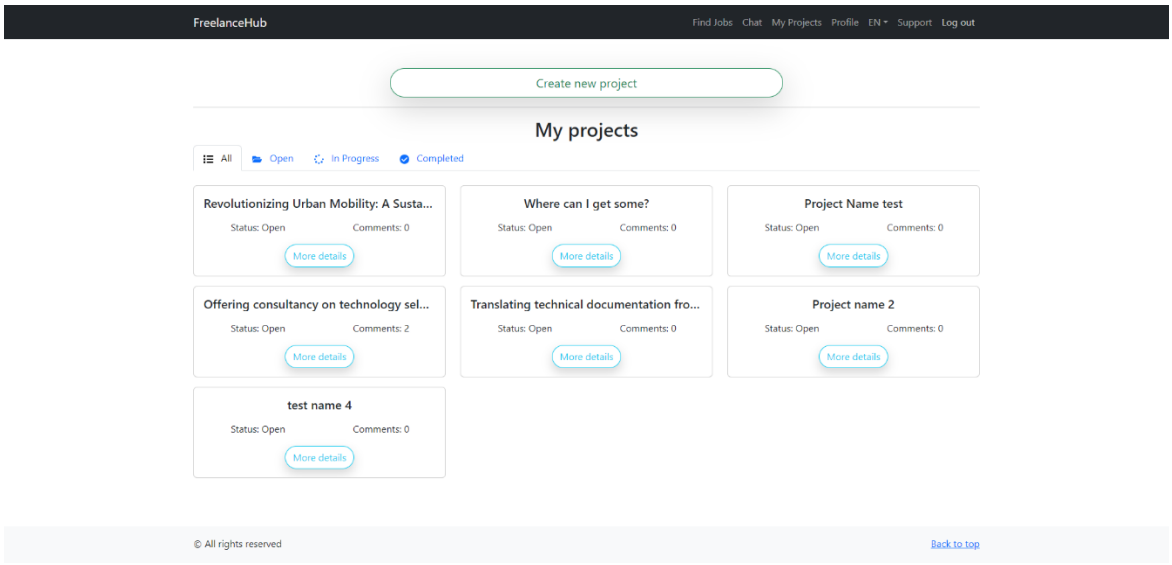


Рисунок 4.8 – Відображення сторінки

На рис. 4.9 зображена сторінка з детальною інформацією про проєкт, на яку можна потрапити через сторінку управління проєктами, натиснувши «More Details» на конкретному проєкті. На цій сторінці роботодавець може переглядати деталі проєкту, включаючи його назву, тип, підтип, опис, бюджет, термін виконання, статус, а також інформацію про фрілансера, якщо він призначений. Є можливість редагувати або видаляти проєкт за допомогою відповідних кнопок. На сторінці також є модальне вікно для підтвердження видалення проєкту. Додатково, роботодавець може переглядати коментарі, залишені фрілансерами, та взаємодіяти з ними через чат.

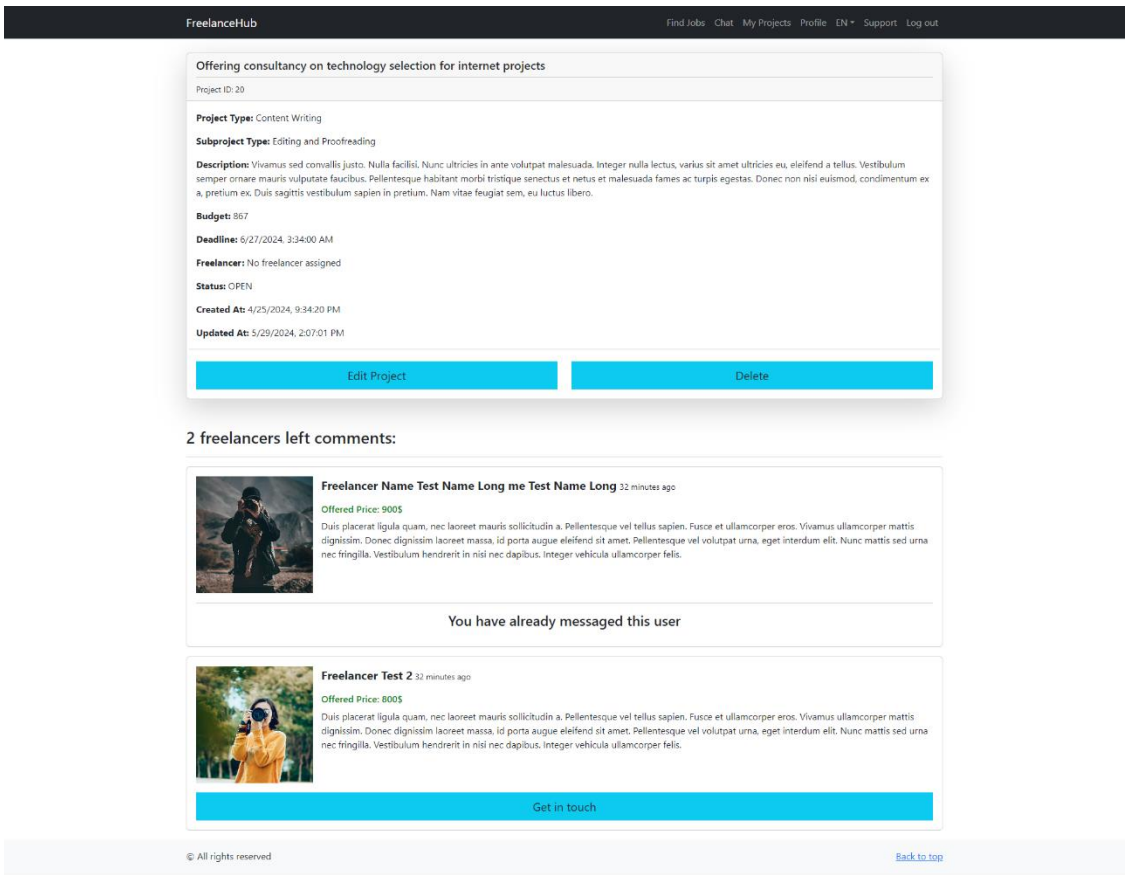


Рисунок 4.9 – Відображення сторінки з детальною інформацією про проєкт

На рис. 4.10 відображено список проєктів, для яких були створені чати. Кожен проєкт у списку має назву, статус і короткий опис. Користувачі можуть переглядати цей список, щоб знайти необхідний проєкт і отримати доступ до відповідних чатів. Це допомагає організувати спілкування і тримати всі проєкти під контролем.

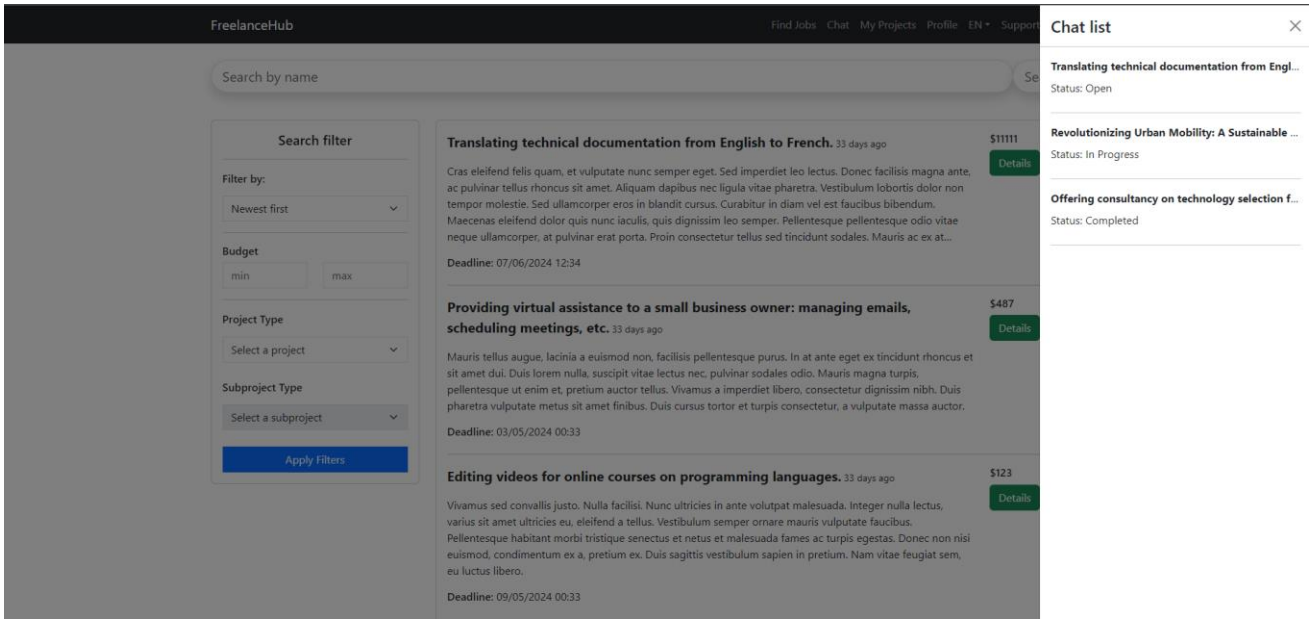


Рисунок 4.10 – Відображення списка проєктів в чаті

На рис. 4.11 показано список чатів з користувачами, які були створені для обраного проєкту. Потрапити на цю сторінку можна, натиснувши на конкретний проєкт із списку проєктів. Для кожного проєкту є свої окремі чати з фрілансерами. Користувач може вибрати потрібний чат, щоб почати або продовжити переписку з конкретним фрілансером, переглядати попередні повідомлення та взаємодіяти з ним. Якщо умова співпраці була створена, то всі чати стають неактивними, окрім того, з яким угода була створена.

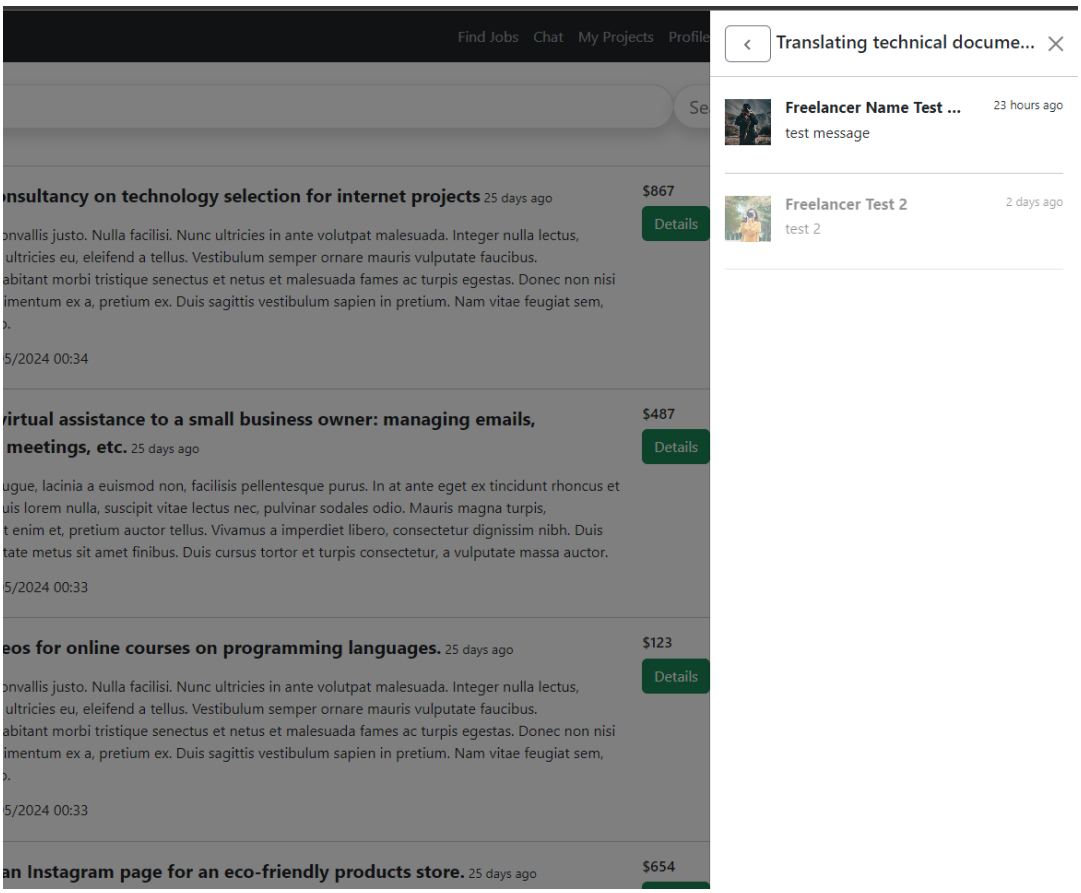


Рисунок 4.11 – Відображення списока чатів з користувачами

На рис. 4.12 представлено сам чат з конкретним користувачем. У цьому чаті можна обмінюватися повідомленнями, відправляти файли, а також, якщо роль користувача – роботодавець, то він може відправити фрілансеру запит на співпрацю. Якщо фрілансер погоджується, з'являються додаткові можливості, такі як затвердження, відхилення та редагування роботи. Усі дії відображаються у чаті.

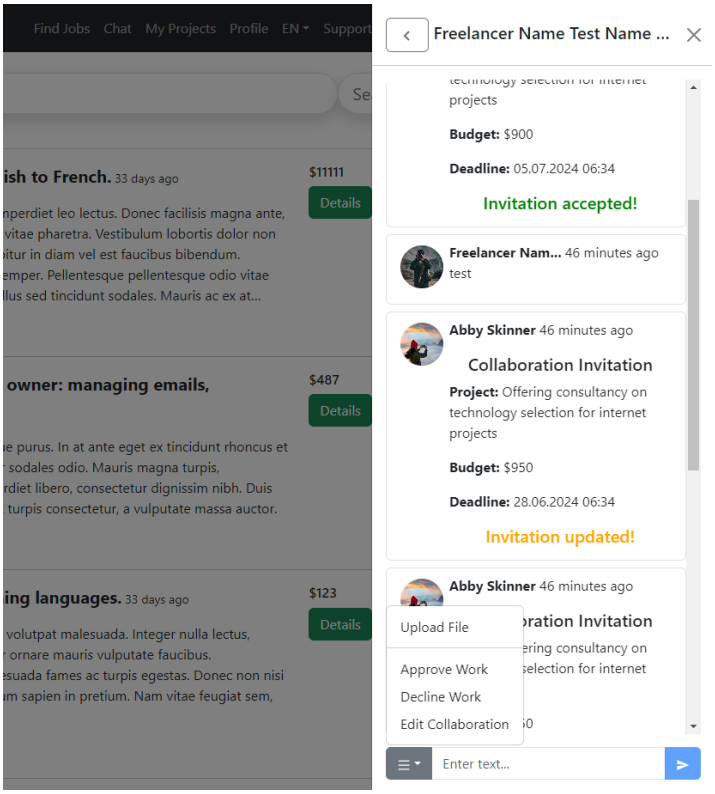


Рисунок 4.12 – Відображення чата з конкретним користувачем

Після затвердження роботи роботодавець може залишити рейтинг та коментар для фрілансера у спливаючому модальному вікні, яке можна побачити на рис. 4.13.

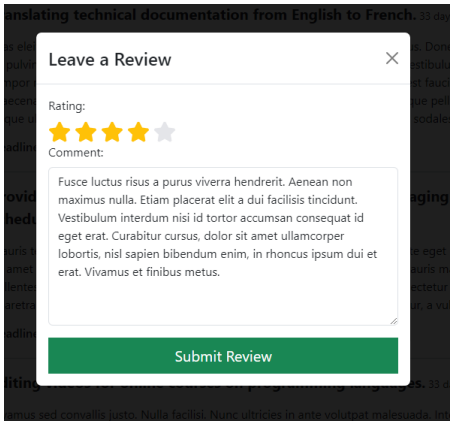


Рисунок 4.13 – Відображення модального вікна для відгука про працю фрілансера

На рис. 4.14 показано модальне вікно, яке відкривається після натискання на кнопку «Support» в хедері. У цьому вікні користувач може написати повідомлення до служби підтримки для вирішення своїх питань.

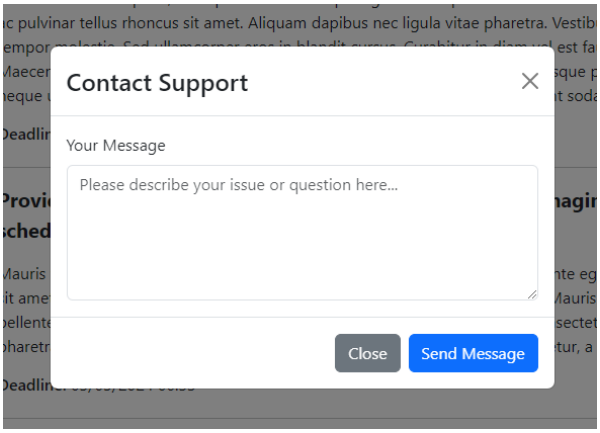


Рисунок 4.14 – Відображення модального вікна для звернення до служби підтримки

Під час розробки застосунку виконані усі поставлені задачі.

Висновки до розділу 4

У четвертому розділі кваліфікаційної роботи представлено розроблений вебзастосунок для взаємодії фрілансерів та замовників. Метою кваліфікаційної роботи було створення вебплатформи, спрямованої на полегшення та оптимізацію процесу співпраці між фрілансерами та замовниками. У результаті розробки вдалося створити інтерфейс головної сторінки для незареєстрованих або неаутентифікованих користувачів, сторінку для аутентифікації, а також детальні сторінки профілів для фрілансерів та замовників, де відображається вся необхідна інформація.

Крім того, були реалізовані функції перегляду, додавання та редагування проєктів, управління чатами для спілкування між користувачами, а також система відгуків та рейтингів для оцінки роботи фрілансерів. Усі ці елементи спрямовані на підвищення ефективності співпраці, зменшення часових витрат на пошук та вибір кандидатів, та покращення якості виконання завдань. Таким чином, при реалізації вебзастосунку виконані всі поставлені задачі, що сприяє досягненню мети кваліфікаційної роботи.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було проаналізовано, спроєктовано та реалізовано вебзастосунок для взаємодії фрілансерів та замовників. Також було розроблено звіт, який складається з чотирьох розділів.

Було проведено детальний аналіз сучасного стану ринку фрілансу та існуючих вебзастосунків для співпраці фрілансерів із замовниками. Вивчено основні платформи, такі як Upwork, Freelancer.com та Fiverr, що дозволило визначити ключові аспекти для врахування у розробці власної платформи, зокрема широкий функціонал, гнучкі налаштування для користувачів, а також високий рівень безпеки та конфіденційності даних.

Розроблено архітектуру вебзастосунку та визначено основні функції, серед яких реєстрація користувачів, можливість розміщення та перегляду проєктів, і спілкування через вбудований чат. Використання діаграм варіантів використання, класів, послідовностей дій та діяльності дозволило систематизувати функціональні можливості та взаємозв'язки між елементами системи, що сприяло кращому розумінню її архітектури та функціонування.

Для розробки системи обрано технології Java, Spring Framework, React, MySQL та JavaScript. Ці технології забезпечили потужність, гнучкість та популярність у розробці сучасних вебзастосунків. Описано структуру бази даних, включаючи ключові таблиці, зв'язки між ними та типи даних, що дало змогу зрозуміти, як дані зберігаються та взаємодіють у системі.

Розроблено інтерфейси для незареєстрованих користувачів, сторінку аутентифікації, детальні профілі для фрілансерів та замовників. Реалізовано функції перегляду, додавання та редагування проєктів, управління чатами для спілкування між користувачами, систему відгуків та рейтингів для оцінки роботи фрілансерів. Усі ці елементи сприяють підвищенню ефективності співпраці, зменшенню часових витрат на пошук та вибір кандидатів, та покращенню якості виконання завдань. Таким чином, вебзастосунок реалізовано у повній відповідності до поставлених задач.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Upwork : вебсайт. URL : https://www.upwork.com/ (Last accessed: 02.04.2024)

2. Freelancer.com : вебсайт. URL : https://www.freelancer.com/ (Last accessed: 02.04.2024)

3. Fiverr: вебсайт. URL : https://www.fiverr.com/ (Last accessed: 02.04.2024)

4. JavaScript docs : вебсайт. URL : https://developer.mozilla.org/en-US/docs/Web/JavaScript (Last accessed: 04.04.2024)

5. JavaScript Tutorial: вебсайт. URL : https://www.w3schools.com/js/ (Last accessed: 15.04.2024)

6. Crockford D. The Good Parts: підручник. Yahoo Pres, 2008. 172 с.

7. Haverbeke M. Eloquent JavaScript : підручник. No Starch Press, 2018. 472 с.

8. Svekis L. JavaScript from Beginner to Professional : підручник. Packt Publishing, 2021. 546 с.

9. Sierra K., Bates B., Gee T. Head First Java : A Brain-Friendly Guide: Bloch J. Effective java. Addison-Wesley Professional : підручник. O'Reilly Media, 2017. 752 с.

10. Schildt H., Coward D. Java: The Complete Reference, Thirteenth Edition : підручник. McGraw Hill, 2024. 1280 с.

11. Java docs : вебсайт. URL : https://docs.oracle.com/en/java/ (Last accessed: 23.04.2024)

12. Java Tutorial : вебсайт. URL : https://www.w3schools.com/java/ (Last accessed: 29.04.2024)

13. Java Tutorial : вебсайт. URL : https://www.javatpoint.com/java-tutorial (Last accessed: 01.05.2024)

14. React docs : вебсайт. URL : https://legacy.reactjs.org/docs/getting-started.html (Last accessed: 04.05.2024)

15. React Tutorial : вебсайт. URL : https://www.w3schools.com/REACT/DEFAULT.ASP (Last accessed: 08.05.2024)

16. React interactivity: Events and state : вебсайт. URL : https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_interactivity_events_state (Last accessed: 08.04.2024)

17. Wieruch R. The Road to React : підручник. Independently published, 2018. 286 с.

18. Schwarzmüller M. React Key Concepts : підручник. Packt Publishing, 2022. 590 с.

19. Spring Framework docs : вебсайт. URL : https://docs.spring.io/spring-framework/reference/index.html (Last accessed: 11.05.2024)

20. Spring Tutorial : вебсайт. URL : https://www.baeldung.com/spring-tutorial (Last accessed: 14.05.2024)

21. Spring Tutorial : вебсайт. URL : https://www.javatpoint.com/spring-tutorial (Last accessed: 21.05.2024)

22. Walls C. Spring in Action, Sixth Edition : підручник. Manning Publications, 2022. 520 с.

23. Ullenboom C. Spring Boot 3 and Spring Framework 6 : підручник. Rheinwerk Computing, 2023. 934 с.

24. MySQL docs : вебсайт. URL : https://dev.mysql.com/doc/ (Last accessed: 23.05.2024)

25. Nichter D. Efficient MySQL Performance : підручник. O'Reilly Media, 2022. 335 с.

26. Grippa V., Kuzmichev S. Learning MySQL: Get a Handle on Your Data : підручник. O'Reilly Media, 2021. 629 с.

ДОДАТОК А

Лістинг класу AuthenticationService

```
@Service
public class AuthenticationService {
   private final UserRepository userRepository;
   private final PasswordEncoder passwordEncoder;
   private final AuthenticationManager authenticationManager;
   private final static String DEFAULT_AVATAR ="
ORw0KGg…";
   public AuthenticationService(
         UserRepository userRepository,
         AuthenticationManager authenticationManager,
         PasswordEncoder passwordEncoder
   ) {
      this.authenticationManager = authenticationManager;
      this.userRepository = userRepository;
      this.passwordEncoder = passwordEncoder;
   }
   public User signup(RegisterUserDto input) {
      if (userRepository.findByEmail(input.getEmail()).isEmpty()) {
         User user = new User();
         Timestamp currentTime = new Timestamp(System.currentTimeMillis());
         user.setFullName(input.getFullName());
         user.setEmail(input.getEmail());
         user.setPassword(passwordEncoder.encode(input.getPassword()));
         user.setBio(input.getBio());

user.setUserTypeEnum(UserTypeEnum.valueOf(input.getUserType().toUpperCase()));
         if (input.getProfilePicture() != null) {
```

```
            user.setProfilePicture(convertToByteArray(input.getProfilePicture()));
        } else {
            user.setProfilePicture(convertToByteArray(DEFAULT_AVATAR));
        }
        user.setCreatedAt(currentTime);
        user.setUpdatedAt(currentTime);

        return userRepository.save(user);
    } else {
        throw new UserWithThisEmailAlreadyExistsException();
    }
  }
  public User authenticate(LoginUserDto input) {
    authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                    input.getEmail(),
                    input.getPassword()
            )
    );
    return userRepository.findByEmail(input.getEmail())
            .orElseThrow();
  }
  private byte[] convertToByteArray(String base64) {
    base64 = base64.replace("", "");
    return Base64.getDecoder().decode(base64);
  }
}
```

ДОДАТОК Б

Лістинг класу JwtService

```
@Service
public class JwtService {
    @Value("${security.jwt.secret-key}")
    private String secretKey;
    @Value("${security.jwt.expiration-time}")
    private long jwtExpiration;
    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
    }
    public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }
    public String generateToken(UserDetails userDetails) {
        return generateToken(new HashMap<>(), userDetails);
    }
    public String generateToken(Map<String, Object> extraClaims, UserDetails
userDetails) {
        return buildToken(extraClaims, userDetails, jwtExpiration);
    }
    public long getExpirationTime() {
        return jwtExpiration;
    }
    private String buildToken(
            Map<String, Object> extraClaims,
            UserDetails userDetails,
            long expiration
```

```
    ) {
        return Jwts
                .builder()
                .setClaims(extraClaims)
                .setSubject(userDetails.getUsername())
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + expiration))
                .signWith(getSignInKey(), SignatureAlgorithm.HS256)
                .compact();
    }
    public boolean isTokenValid(String token, UserDetails userDetails) {
        final String username = extractUsername(token);
        return (username.equals(userDetails.getUsername())) &&
!isTokenExpired(token);
    }
    private boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());}
    private Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration); }
    private Claims extractAllClaims(String token) {
        return Jwts
                .parserBuilder()
                .setSigningKey(getSignInKey())
                .build()
                .parseClaimsJws(token)
                .getBody();}
    private Key getSignInKey() {
        byte[] keyBytes = Decoders.BASE64.decode(secretKey);
        return Keys.hmacShaKeyFor(keyBytes);}}
```

ДОДАТОК В

Лістинг класу UserService

```
@Service
@Transactional(readOnly = true)
public class UserService {
    private final UserRepository userRepository;
    private final ProjectRepository projectRepository;
    private final ReviewRepository reviewRepository;
    @Autowired
    public UserService(UserRepository userRepository, ProjectRepository
projectRepository, ReviewRepository reviewRepository) {
        this.userRepository = userRepository;
        this.projectRepository = projectRepository;
        this.reviewRepository = reviewRepository;
    }
    public UserProfileResponseDTO getUserProfileData() {
        User currentUser = getCurrentUser();
        List<Project> projects = null;
        List<Review> reviews = null;
        if (currentUser.getUserTypeEnum() == UserTypeEnum.EMPLOYER) {
            projects = projectRepository.findAllByEmployerId(currentUser.getId());
        } else {
            reviews = reviewRepository.findAllByFreelancerId(getCurrentUser().getId());
        }
        return UserProfileResponseDTO.builder()
                .userProfileData(UserProfileData.builder()
                        .fullName(currentUser.getFullName())
                        .bio(currentUser.getBio())
                        .userType(String.valueOf(currentUser.getUserTypeEnum()))
```

```
                .profilePicture(currentUser.getProfilePicture())
                .build())
            .projectStatistics(projects != null ? UserProfileProjectStatistics.builder()
                .numberOfProjects(numberOfProjects(projects))
                .numberOfActiveProjects(numberOfActiveProjects(projects))
                .numberOfCompletedProjects(numberOfCompletedProjects(projects))
                .build() : null)
            .reviewList(reviews != null ? getReviewList(reviews) : null)
            .build();
    }
    private List<UserReview> getReviewList(List<Review> reviewList) {
        return reviewList.stream()
            .map(review -> UserReview.builder()
                .rating(review.getRating())
                .comment(review.getComment())
                .reviewDate(review.getReviewDate())
                .employerName(review.getEmployer().getFullName())
                .budget(review.getProject().getBudget())
                .projectName(review.getProject().getTitle())
                .build())
            .toList();
    }
    private int numberOfProjects(List<Project> projects) {
        return projects.size();
    }
    private int numberOfActiveProjects(List<Project> projects) {
        return (int) projects.stream().filter(project ->
!project.getStatus().equals(StatusProject.COMPLETED) &&
!project.getStatus().equals(StatusProject.CANCELLED))
```

```
            .count();
    }
    private int numberOfCompletedProjects(List<Project> projects) {
        return (int) projects.stream().filter(project ->
project.getStatus().equals(StatusProject.COMPLETED))
            .count();
    }
    @Transactional
    public void userEdit(UserProfileEditRequestDTO profileRequestDTO) {
        User currentUser = getCurrentUser();
        currentUser.setFullName(profileRequestDTO.getFullName());
        currentUser.setBio(profileRequestDTO.getBio());
        currentUser.setProfilePicture(profileRequestDTO.getProfilePicture());
        currentUser.setUpdatedAt(new Timestamp(System.currentTimeMillis()));
        userRepository.save(currentUser);
    }
    public User getCurrentUser() {
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        return (User) authentication.getPrincipal();
    }
}
```

ДОДАТОК Г

Лістинг класу ProjectService

```
@Service
@Transactional(readOnly = true)
public class ProjectService {
   private final ProjectRepository projectRepository;
   private final SubprojectTypeRepository subprojectTypeRepository;
   private final ProjectCommentRepository projectCommentRepository;
   private final ChatRepository chatRepository;
   @Autowired
   public ProjectService(ProjectRepository projectRepository,
SubprojectTypeRepository subprojectTypeRepository, ProjectCommentRepository
projectCommentRepository, ChatRepository chatRepository) {
      this.projectRepository = projectRepository;
      this.subprojectTypeRepository = subprojectTypeRepository;
      this.projectCommentRepository = projectCommentRepository;
      this.chatRepository = chatRepository; }
   public List<ProjectGetAllForEmployerResponseDTO> getAllForEmployer() {
      return mappingToProjectsList(null); }
   public List<ProjectGetAllForEmployerResponseDTO> getAllOpenForEmployer() {
      return mappingToProjectsList(StatusProject.OPEN); }
   public List<ProjectGetAllForEmployerResponseDTO>
getAllInProgressForEmployer() {
      return mappingToProjectsList(StatusProject.IN_PROGRESS); }
   public List<ProjectGetAllForEmployerResponseDTO>
getAllCompletedForEmployer() {
      return mappingToProjectsList(StatusProject.COMPLETED); }
   private List<ProjectGetAllForEmployerResponseDTO>
mappingToProjectsList(StatusProject filterBy) {
      User currentUser = getCurrentUser();
```

```
        List<Project> projects;
        if (filterBy == null) {
            projects = projectRepository.findAllByEmployerId(currentUser.getId());
        } else {
            projects =
projectRepository.findAllByEmployerIdAndStatus(currentUser.getId(), filterBy); }
        return projects.stream()
                .sorted(Comparator.comparing(Project::getUpdatedAt).reversed())
                .map(project -> ProjectGetAllForEmployerResponseDTO.builder()
                        .id(project.getId())
                        .title(project.getTitle())
                        .status(project.getStatus().toString())
.amountOfComments(projectCommentRepository.findAllByProjectId(project.getId()).
get().size()).build()).toList();}
    @Transactional
    public void create(ProjectCreateRequestDTO projectCreateRequestDTO) {
        User currentUser = getCurrentUser();
        Project newProject = new Project();
        newProject.setTitle(projectCreateRequestDTO.getTitle());
        newProject.setDescription(projectCreateRequestDTO.getDescription());
        newProject.setBudget(projectCreateRequestDTO.getBudget());
        newProject.setDeadline(projectCreateRequestDTO.getDeadline());

newProject.setSubprojectType(subprojectTypeRepository.findById(Long.parseLong(p
rojectCreateRequestDTO.getSubprojectType())));
        newProject.setEmployer(currentUser);
        projectRepository.save(newProject); }
    public ProjectDetailsForEmployerResponseDTO
getProjectForEmployerDetails(long id) {
        Project project = projectRepository.findById(id);
```

```
        ProjectDetailsForEmployer projectDetailsForEmployer =
ProjectDetailsForEmployer.builder()
                .id(project.getId())
                .title(project.getTitle())
                .description(project.getDescription())
                .budget(project.getBudget())
                .deadline(project.getDeadline())
                .freelancer(project.getFreelancer())
                .status(project.getStatus().toString())
                .projectType(project.getSubprojectType().getProjectType())
                .subprojectType(project.getSubprojectType())
                .createdAt(project.getCreatedAt())
                .updatedAt(project.getUpdatedAt()).build();
        List<ProjectCommentGetAllForProjectDetails>
projectCommentGetAllForProjectDetails =
                projectCommentRepository.findAllByProjectId(id)
                        .orElse(Collections.emptyList())
                        .stream()
                        .map(projectComment ->
ProjectCommentGetAllForProjectDetails.builder()
                                .commentText(projectComment.getCommentText())
                                .budget(projectComment.getBudget())
                                .createdAt(projectComment.getCreatedAt())
                                .userName(projectComment.getUser().getFullName())
                                .userId(projectComment.getUser().getId())
.firstMessage(chatRepository.existsByEmployerIdAndFreelancerIdAndProjectId(getC
urrentUser().getId(), projectComment.getUser().getId(),
projectComment.getProject().getId()))
                                .profilePicture(projectComment.getUser().getProfilePicture())
                                .build()).toList();
```

```
        return new ProjectDetailsForEmployerResponseDTO(projectDetailsForEmployer,
projectCommentGetAllForProjectDetails); }

    public ProjectDetailsForFreelancerResponseDTO
getProjectDetailsForFreelancer(long id) {

        Project project = projectRepository.findById(id);

        ProjectDetailsForFreelancer projectDetailsForFreelancer =
ProjectDetailsForFreelancer.builder()

                .id(project.getId())

                .title(project.getTitle())

                .description(project.getDescription())

                .budget(project.getBudget())

                .deadline(project.getDeadline())

                .status(project.getStatus().toString())

                .projectType(project.getSubprojectType().getProjectType())

                .subprojectType(project.getSubprojectType())

                .createdAt(project.getCreatedAt())

                .build();

        List<ProjectCommentGetAllForProjectDetails>
projectCommentGetAllForProjectDetails =

                projectCommentRepository.findAllByProjectId(id)

                        .orElse(Collections.emptyList())

                        .stream()

                        .map(projectComment ->
ProjectCommentGetAllForProjectDetails.builder()

                                .commentText(projectComment.getCommentText())

                                .budget(projectComment.getBudget())

                                .createdAt(projectComment.getCreatedAt())

                                .userName(projectComment.getUser().getFullName())

                                .profilePicture(projectComment.getUser().getProfilePicture())

                                .build()).toList();
```

```
        return new
ProjectDetailsForFreelancerResponseDTO(projectDetailsForFreelancer,
projectCommentGetAllForProjectDetails); }
    @Transactional
    public void editProject(ProjectEditRequestDTO editRequestDTO) {
        Project project = projectRepository.findById(editRequestDTO.getId());
        project.setTitle(editRequestDTO.getTitle());
        project.setDescription(editRequestDTO.getDescription());
        project.setBudget(editRequestDTO.getBudget());
        project.setDeadline(editRequestDTO.getDeadline());
        project.setFreelancer(editRequestDTO.getFreelancer());
        project.setSubprojectType(editRequestDTO.getSubprojectType());
        project.getSubprojectType().setProjectType(editRequestDTO.getProjectType());
        if (editRequestDTO.getStatus() != null &&
!editRequestDTO.getStatus().isEmpty())
            project.setStatus(StatusProject.valueOf(editRequestDTO.getStatus()));
        project.setUpdatedAt(new Timestamp(System.currentTimeMillis()));
        projectRepository.save(project); }
    @Transactional
    public void delete(long id) {
        projectRepository.deleteById(id);}
    public ProjectPagesDTO getAllForFreelancer(int page, int size) {
        PageRequest pageable = PageRequest.of(page, size);
        List<Project> list = projectRepository.findAllByStatus(StatusProject.OPEN);
        int numberOfProjects = list.size();
        List<ProjectGetAllForFreelancerResponseDTO> projects =
projectRepository.findAllByStatusOrderByCreatedAtDesc(StatusProject.OPEN,
pageable).stream()
                .sorted(Comparator.comparing(Project::getCreatedAt).reversed())
                .map(project -> ProjectGetAllForFreelancerResponseDTO.builder()
```

```
                .id(project.getId())
                .title(project.getTitle())
                .budget(project.getBudget())
                .description(project.getDescription())
                .deadline(project.getDeadline())
                .created_at(project.getCreatedAt())
                .build())
        .toList();
    return new ProjectPagesDTO(projects, numberOfProjects);}
  public ProjectPagesDTO getFilteredProjectsForFreelancer(FilterDTO filterDTO, int page, int size) {
    List<Project> projects;
    if (!filterDTO.getSearchString().isEmpty()) {
        projects = projectRepository.findAllByStatusAndTitleContaining(StatusProject.OPEN, filterDTO.getSearchString());
    } else {
        projects = projectRepository.findAllByStatus(StatusProject.OPEN);
    }
    List<ProjectGetAllForFreelancerResponseDTO> responseDTOList = SearchFilter.filterProjects(projects, filterDTO).stream()
        .map(project -> ProjectGetAllForFreelancerResponseDTO.builder()
                .id(project.getId())
                .title(project.getTitle())
                .budget(project.getBudget())
                .description(project.getDescription())
                .deadline(project.getDeadline())
                .created_at(project.getCreatedAt())
                .build())
        .toList();
```

```
        int totalProjects = responseDTOList.size();
        int startIndex = Math.max((page - 1) * size, 0);
        int endIndex = Math.min(startIndex + size, totalProjects);
        List<ProjectGetAllForFreelancerResponseDTO> pageResponseDTOList =
responseDTOList.subList(startIndex, endIndex);
        return new ProjectPagesDTO(pageResponseDTOList, totalProjects); }
    public ProjectGetFroCollaborationInvitationResponseDTO
getProjectForCollaborationInvitation(String projectId) {
        Project project = projectRepository.findById(Long.parseLong(projectId));
        return ProjectGetFroCollaborationInvitationResponseDTO.builder()
                .title(project.getTitle())
                .budget(project.getBudget())
                .deadline(project.getDeadline())
                .build();}
    public User getCurrentUser() {
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        return (User) authentication.getPrincipal();
    }
}
```

ДОДАТОК Д

Лістинг класу CollaborationInvitationService

```
@Service
@Transactional(readOnly = true)
public class CollaborationInvitationService {
    private final CollaborationInvitationRepository collaborationInvitationRepository;
    private final ProjectRepository projectRepository;
    private final UserRepository userRepository;
    private final ChatMessageRepository chatMessageRepository;
    private final ChatRepository chatRepository;
    private final ReviewService reviewService;
    @Autowired
    public CollaborationInvitationService(CollaborationInvitationRepository
collaborationInvitationRepository, ProjectRepository projectRepository,
UserRepository userRepository, ChatMessageRepository chatMessageRepository,
ChatRepository chatRepository, ReviewService reviewService) {
        this.collaborationInvitationRepository = collaborationInvitationRepository;
        this.projectRepository = projectRepository;
        this.userRepository = userRepository;
        this.chatMessageRepository = chatMessageRepository;
        this.chatRepository = chatRepository;
        this.reviewService = reviewService;}
    @Transactional
    public void create(CollaborationInvitationCreateRequestDTO createRequestDTO) {
        Project project =
projectRepository.findById(Long.parseLong(createRequestDTO.getProjectId()));
        CollaborationInvitation collaborationInvitation =
collaborationInvitationRepository.save(CollaborationInvitation.builder()
                .employer(getCurrentUser())
```

```
.freelancer(userRepository.findById(Long.parseLong(createRequestDTO.getFreelance
rId())))
                .project(project)
                .originalBudget(project.getBudget())
                .originalDeadline(project.getDeadline())
                .newBudget(new BigDecimal(createRequestDTO.getNewBudget()))
                .newDeadline(createRequestDTO.getNewDeadline())
                .status(InvitationStatus.PENDING)
.chat(chatRepository.findByProjectIdAndEmployerIdAndFreelancerId(Long.parseLon
g(createRequestDTO.getProjectId()), getCurrentUser().getId(),
Long.parseLong(createRequestDTO.getFreelancerId())))
                .build());
        chatMessageRepository.save(ChatMessage.builder()
                .chat(chatRepository.findById(collaborationInvitation.getChat().getId()))
                .sender(getCurrentUser())
                .messageText(collaborationInvitation.getStatus().getStatus())
                .collaborationInvitation(collaborationInvitation)
                .build());}
    @Transactional
    public void acceptCollaborationInvitation(String invitationId) {
        CollaborationInvitation invitation =
collaborationInvitationRepository.findById(Long.parseLong(invitationId));
        invitation.setStatus(InvitationStatus.ACCEPTED);
        ChatMessage chatMessage =
chatMessageRepository.findByCollaborationInvitationId(invitation.getId());
        chatMessage.setMessageText("Accepted");
        chatMessageRepository.save(chatMessage);
        long currentChatId = invitation.getChat().getId();
        List<Chat> chats =
chatRepository.findAllByProjectId(invitation.getProject().getId());
```

```
        chats.forEach(chat -> chat.setActive(chat.getId() == currentChatId));
        Project project = projectRepository.findById(invitation.getProject().getId());
        project.setFreelancer(invitation.getFreelancer());
        project.setStatus(StatusProject.IN_PROGRESS);
        projectRepository.save(project);
        chatRepository.saveAll(chats);
        collaborationInvitationRepository.save(invitation); }
    @Transactional
    public void declineInvitation(String id) {
        CollaborationInvitation collaborationInvitation =
collaborationInvitationRepository.findById(Long.parseLong(id));
        collaborationInvitation.setStatus(InvitationStatus.DECLINED);
        collaborationInvitationRepository.save(collaborationInvitation); }
    @Transactional
    public void
declineCollaborationInvitation(CollaborationInvitationCreateRequestDTO
requestDTO) {
        CollaborationInvitation latestCollaborationInvitation =
getLatestInvitation(collaborationInvitationRepository.findByFreelancerIdAndEmploye
rIdAndProjectId(Long.parseLong(requestDTO.getFreelancerId()),
getCurrentUser().getId(), Long.parseLong(requestDTO.getProjectId())));
        CollaborationInvitation newCollaborationInvitation =
CollaborationInvitation.builder()
                .employer(latestCollaborationInvitation.getEmployer())
                .freelancer(latestCollaborationInvitation.getFreelancer())
                .project(latestCollaborationInvitation.getProject())
                .originalBudget(latestCollaborationInvitation.getOriginalBudget())
                .originalDeadline(latestCollaborationInvitation.getOriginalDeadline())
                .newBudget(latestCollaborationInvitation.getNewBudget())
                .newDeadline(latestCollaborationInvitation.getNewDeadline())
                .status(InvitationStatus.DECLINED)
```

```
            .chat(latestCollaborationInvitation.getChat()).build();
        List<Chat> chats =
chatRepository.findAllByProjectId(latestCollaborationInvitation.getProject().getId());
        chats.forEach(chat -> chat.setActive(true));
        Project project =
projectRepository.findById(latestCollaborationInvitation.getProject().getId());
        project.setFreelancer(null);
        project.setStatus(StatusProject.OPEN);
        chatMessageRepository.save(ChatMessage.builder()
.chat(chatRepository.findById(latestCollaborationInvitation.getChat().getId()))
            .sender(getCurrentUser())
            .messageText("Declined")
            .collaborationInvitation(newCollaborationInvitation).build());
        projectRepository.save(project);
        chatRepository.saveAll(chats);
        collaborationInvitationRepository.save(newCollaborationInvitation); }
    public CollaborationInvitationGetResponseDTO getCollaboration(long projectId,
long userId) {
        List<CollaborationInvitation> collaborationInvitationList =
collaborationInvitationRepository.findByFreelancerIdAndEmployerIdAndProjectId(us
erId, getCurrentUser().getId(), projectId);
        if (collaborationInvitationList.isEmpty()) {
            Project project = projectRepository.findById(projectId);
            return CollaborationInvitationGetResponseDTO.builder()
                .title(project.getTitle())
                .budget(project.getBudget())
                .deadline(project.getDeadline())
                .build();
        } else {
            collaborationInvitationList = collaborationInvitationList.stream()
```

```
.sorted(Comparator.comparing(CollaborationInvitation::getCreatedAt).reversed())
                .toList();
        CollaborationInvitation latestCollaborationInvitation =
collaborationInvitationList.isEmpty() ? null : collaborationInvitationList.get(0);
        return CollaborationInvitationGetResponseDTO.builder()
                .title(latestCollaborationInvitation.getProject().getTitle())
                .budget(latestCollaborationInvitation.getNewBudget())
                .deadline(latestCollaborationInvitation.getProject().getDeadline())
                .build();}}
    @Transactional
    public void edit(CollaborationInvitationCreateRequestDTO editRequestDTO) {
        List<CollaborationInvitation> collaborationInvitationList =
collaborationInvitationRepository.
findByFreelancerIdAndEmployerIdAndProjectId(Long.parseLong(editRequestDTO.ge
tFreelancerId()), getCurrentUser().getId(),
Long.parseLong(editRequestDTO.getProjectId()));
collaborationInvitationList.sort(Comparator.comparing(CollaborationInvitation::getCr
eatedAt));
        CollaborationInvitation latestCollaborationInvitation =
collaborationInvitationList.isEmpty() ? null : collaborationInvitationList.get(0);
        CollaborationInvitation updatedCollaborationInvitation =
CollaborationInvitation.builder()
                .employer(latestCollaborationInvitation.getEmployer())
                .freelancer(latestCollaborationInvitation.getFreelancer())
                .project(latestCollaborationInvitation.getProject())
                .originalBudget(latestCollaborationInvitation.getOriginalBudget())
                .originalDeadline(latestCollaborationInvitation.getOriginalDeadline())
                .newBudget(new BigDecimal(editRequestDTO.getNewBudget()))
                .newDeadline(editRequestDTO.getNewDeadline())
                .status(InvitationStatus.UPDATED)
                .chat(latestCollaborationInvitation.getChat()).build();
```

```
        collaborationInvitationRepository.save(updatedCollaborationInvitation);
        chatMessageRepository.save(ChatMessage.builder()
.chat(chatRepository.findById(latestCollaborationInvitation.getChat().getId()))
                .sender(getCurrentUser())
                .messageText("Updated")
                .collaborationInvitation(updatedCollaborationInvitation)
                .build());
    }
    @Transactional
    public void
completedCollaborationInvitation(CollaborationInvitationCompletedRequestDTO
requestDTO) {
        CollaborationInvitation latestCollaborationInvitation =
getLatestInvitation(collaborationInvitationRepository.findByFreelancerIdAndEmploye
rIdAndProjectId(Long.parseLong(requestDTO.getFreelancerId()),
getCurrentUser().getId(), Long.parseLong(requestDTO.getProjectId())));
        CollaborationInvitation completedCollaborationInvitation =
CollaborationInvitation.builder()
                .employer(latestCollaborationInvitation.getEmployer())
                .freelancer(latestCollaborationInvitation.getFreelancer())
                .project(latestCollaborationInvitation.getProject())
                .originalBudget(latestCollaborationInvitation.getOriginalBudget())
                .originalDeadline(latestCollaborationInvitation.getOriginalDeadline())
                .newBudget(latestCollaborationInvitation.getNewBudget())
                .newDeadline(latestCollaborationInvitation.getNewDeadline())
                .status(InvitationStatus.COMPLETED)
                .chat(latestCollaborationInvitation.getChat()).build();
        collaborationInvitationRepository.save(completedCollaborationInvitation);
        chatMessageRepository.save(ChatMessage.builder()
  .chat(chatRepository.findById(latestCollaborationInvitation.getChat().getId()))
```

```
            .sender(getCurrentUser())
            .messageText("Completed")
            .collaborationInvitation(completedCollaborationInvitation).build());
        Project project =
projectRepository.findById(latestCollaborationInvitation.getProject().getId());
        project.setStatus(StatusProject.COMPLETED);
        reviewService.save(requestDTO.getFreelancerId(), getCurrentUser(),
requestDTO.getProjectId(), requestDTO.getReview().getComment(),
requestDTO.getReview().getRating());
        projectRepository.save(project);
        collaborationInvitationRepository.save(completedCollaborationInvitation);}
    private CollaborationInvitation getLatestInvitation(List<CollaborationInvitation>
list) {
        return list.stream()
       .sorted(Comparator.comparing(CollaborationInvitation::getCreatedAt).reversed())
            .toList().get(0); }
    public CheckDeadlineDTO checkDeadline() {
        User currentUser = getCurrentUser();
        List<CollaborationInvitation> invitationList;
        if (currentUser.getUserTypeEnum().equals(UserTypeEnum.EMPLOYER)) {
            invitationList =
collaborationInvitationRepository.findByEmployerId(currentUser.getId());
        } else {
            invitationList =
collaborationInvitationRepository.findByFreelancerId(currentUser.getId());}
        Map<Long, Optional<CollaborationInvitation>> latestInvitationsByProjectId =
invitationList.stream()
            .collect(Collectors.groupingBy(
                invitation -> invitation.getProject().getId(),

Collectors.maxBy(Comparator.comparing(CollaborationInvitation::getCreatedAt))));
```

```
        List<CheckDeadline> checkDeadlines =
latestInvitationsByProjectId.values().stream().filter(Optional::isPresent).map(Optional:
:get).filter(invitation -> !Arrays.asList(InvitationStatus.PENDING,
InvitationStatus.DECLINED,
InvitationStatus.COMPLETED).contains(invitation.getStatus()))
                .map(invitation -> CheckDeadline.builder()
                        .deadline(invitation.getNewDeadline())
                        .projectName(invitation.getProject().getTitle()).build()).toList();
        return CheckDeadlineDTO.builder()
                .checkDeadlines(checkDeadlines).build();}
    private User getCurrentUser() {
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        return (User) authentication.getPrincipal();
    }
}
```

ДОДАТОК Е

Лістинг класу ChatService

```
@Service
@Transactional
public class ChatService {
    private final ChatRepository chatRepository;
    private final ChatMessageRepository chatMessageRepository;
    private final CollaborationInvitationRepository collaborationInvitationRepository;
    @Autowired
    public ChatService(ChatRepository chatRepository, ChatMessageRepository
chatMessageRepository, CollaborationInvitationRepository
collaborationInvitationRepository) {
        this.chatRepository = chatRepository;
        this.chatMessageRepository = chatMessageRepository;
        this.collaborationInvitationRepository = collaborationInvitationRepository;
    }
    public List<GetChatProjectResponseDTO> getAllProjectsWithChats() {
        List<Chat> chats =
chatRepository.findByEmployerOrFreelancer(getCurrentUser(), getCurrentUser());
        return chats.stream()
                .map(Chat::getProject)
                .distinct()
                .sorted(Comparator.comparing(Project::getUpdatedAt).reversed())
                .map(project -> GetChatProjectResponseDTO.builder()
                        .projectId(project.getId())
                        .projectName(project.getTitle())
                        .status(String.valueOf(project.getStatus()))
                        .build()).toList();
    }
```

```
public List<GetChatUserResponseDTO> getAllUsersChats(String projectId) {
    List<Chat> chats =
chatRepository.findAllByProjectId(Long.parseLong(projectId));
    User currentUser = getCurrentUser();
    long currentUserId = currentUser.getId();
    boolean isCurrentUserFreelancer =
currentUser.getUserTypeEnum().getUserType().equals(UserTypeEnum.FREELANCE
R.getUserType());
    return chats.stream()
        .filter(chat -> {
          if (isCurrentUserFreelancer) {
            return chat.getFreelancer().getId() == currentUserId;
          } else {
            return chat.getEmployer().getId() == currentUserId; }})
        .map(chat -> {
          ChatMessage lastMessage =
chatMessageRepository.findFirstByChatIdOrderByCreatedAtDesc(chat.getId());
          return GetChatUserResponseDTO.builder()
              .fullName(isCurrentUserFreelancer ?
chat.getEmployer().getFullName() : chat.getFreelancer().getFullName())
              .lastMessage(
                  lastMessage.getMessageText() != null ?
lastMessage.getMessageText() :
                      (lastMessage.getFileName() != null ?
lastMessage.getFileName() : "Collaboration Invitation"))
              .lastMessageTime(lastMessage.getCreatedAt())
              .userPicture(isCurrentUserFreelancer ?
chat.getEmployer().getProfilePicture() : chat.getFreelancer().getProfilePicture())
              .userId(isCurrentUserFreelancer ? chat.getEmployer().getId() :
chat.getFreelancer().getId())
              .active(chat.isActive()).build();})
```

```
.sorted(Comparator.comparing(GetChatUserResponseDTO::getLastMessageTime).rev
ersed()).toList();}
   private boolean collaborationIsActive(List<CollaborationInvitation> list) {
     list.sort(Comparator.comparing(CollaborationInvitation::getCreatedAt).reversed());
      for (CollaborationInvitation c : list) {
         if (c.getStatus().getStatus().equals("ACCEPTED")) {
            return true; }
         if (c.getStatus().getStatus().equals("DECLINED")) {
            return false; }}
      return false;}
   public GetChatResponseDTO getChat(long userId, long projectId) {
      User currentUser = getCurrentUser();
      Chat chat;
      boolean collaborationIsActive = false;
      if
(currentUser.getUserTypeEnum().getUserType().equals(UserTypeEnum.EMPLOYER.
getUserType())) {
         chat =
chatRepository.findByProjectIdAndEmployerIdAndFreelancerId(projectId,
currentUser.getId(), userId);
         collaborationIsActive =
collaborationIsActive(collaborationInvitationRepository.findByFreelancerIdAndEmpl
oyerIdAndProjectId(userId, currentUser.getId(), projectId));
      } else {
         chat =
chatRepository.findByProjectIdAndEmployerIdAndFreelancerId(projectId, userId,
currentUser.getId());}
      return GetChatResponseDTO.builder()
         .chatId(chat.getId())
         .collaborationIsActive(collaborationIsActive)
         .chatMessageList(chat.getMessages().stream()
```

```
                .map(message -> GetChatMessage.builder()
                        .messageId(message.getId())
                        .sender(Sender.builder()
                                .senderId(message.getSender().getId())
                                .fullName(message.getSender().getFullName())
                                .profilePicture(message.getSender().getProfilePicture())
                                .build())
                        .messageText(message.getCollaborationInvitation() == null ?
message.getMessageText() : null)
                        .fileName(message.getFileName())
                        .file(message.getFileData())
                        .createdAt(message.getCreatedAt())
                        .collaborationInvitation(message.getCollaborationInvitation() !=
null ?
getCollaborationInvitation(message.getCollaborationInvitation()) : null)
                        .build()).toList()).build();}
    private GetCollaborationInvitation
getCollaborationInvitation(CollaborationInvitation collaborationInvitation) {
        return GetCollaborationInvitation.builder()
                .id(collaborationInvitation.getId())
                .budget(collaborationInvitation.getNewBudget())
                .projectName(collaborationInvitation.getProject().getTitle())
                .status(collaborationInvitation.getStatus().getStatus())
                .createdAt(collaborationInvitation.getCreatedAt())
                .deadline(collaborationInvitation.getNewDeadline()).build();}
    public User getCurrentUser() {
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        return (User) authentication.getPrincipal();}
}
```

ДОДАТОК Ж

Лістинг класу ChatMessageService

```
@Service
@Transactional(readOnly = true)
public class ChatMessageService {
    private final ChatMessageRepository chatMessageRepository;
    private final ProjectRepository projectRepository;
    private final UserRepository userRepository;
    private final ChatRepository chatRepository;
    @Autowired
    public ChatMessageService(ChatMessageRepository chatMessageRepository,
ProjectRepository projectRepository, UserRepository userRepository, ChatRepository
chatRepository) {
        this.chatMessageRepository = chatMessageRepository;
        this.projectRepository = projectRepository;
        this.userRepository = userRepository;
        this.chatRepository = chatRepository; }
    @Transactional
    public void
chatResponseFromEmployerToComment(ChatResponseFromEmployerToCommentRe
questDTO chatDTO) {
        User employer = getCurrentUser();
        User freelancer = userRepository.findById(chatDTO.getFreelancerId());
        Project project = projectRepository.findById(chatDTO.getProjectId());
        Chat chat = new Chat();
        chat.setProject(project);
        chat.setEmployer(employer);
        chat.setFreelancer(freelancer);
        chat.setActive(true);
        chatRepository.save(chat);
```

```
        ChatMessage message = new ChatMessage();
        message.setChat(chat);
        message.setSender(employer);
        message.setMessageText(chatDTO.getMessage());
        if (chat.getMessages() == null) {
            chat.setMessages(new ArrayList<>());}
        chat.getMessages().add(message);
        chatRepository.save(chat); }
    @Transactional
    public void sendMessage(SendMessageRequestDTO requestDTO) {
        chatMessageRepository.save(ChatMessage.builder()
                .chat(chatRepository.findById(requestDTO.getChatId()))
                .messageText(!requestDTO.getTextMessage().isEmpty() ?
requestDTO.getTextMessage() : null)
                .fileData(requestDTO.getFile() != null ? requestDTO.getFile() : null)
                .fileName(!requestDTO.getFileName().isEmpty() ?
requestDTO.getFileName() : null)
                .sender(getCurrentUser())
                .build());}
    public User getCurrentUser() {
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        return (User) authentication.getPrincipal();
    }
}
```