МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ

(підпис)

01 червня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 – Комп'ютерні науки,

освітньо-професійної програми «Інформатика»

на тему: «Інформаційне і програмне забезпечення онлайн-ринку для співпраці між замовниками та фрілансерами»

здобувача групи ІН-02 Погорєлова Данила Олександровича

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

_____ Данило ПОГОРЄЛОВ

(підпис)

Керівник

асистент кафедри комп'ютерних наук,

кандидат фізико-математичних наук Олександр ВЛАСЕНКО_____

(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
на здобуття освітнього ступеня бакалавра
зі спеціальності 122 – Комп'ютерні науки, освітньо-професійної програми «Інформатика»
здобувача групи ІН-02 Погорєлова Данила Олександровича

1. Тема роботи: «Інформаційне і програмне забезпечення онлайн-ринку для співпраці між замовниками та фрілансерами» затверджена наказом по СумДУ від *«22» квітня 2024 р. № 0414-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 01 червня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити): *1) Інформаційний огляд. 2) Вибір методів рішення задач. 3) Інформаційне та програмне забезпечення системи.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «08» квітня 2024 р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз предметної області, постановка і формування завдань*	21.04.2024	
2	*Огляд і вивчення технологій*	1.05.2024	
3	*Розробка системи програмного забезпечення*	12.05.2024	
4	*Аналіз отриманих результатів*	18.05.2024	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	24.05.2024	

Здобувач вищої освіти _____ Керівник _____

АНОТАЦІЯ

Записка: 97 стор., 21 рис., 57 додатків, 11 використаних джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною тому що зростає попит на фріланс, виникає потреба в ефективному посередництві між фрілансерами та замовниками, розвиваються технології, які забезпечують швидкодію та безпеку платформ, а також це економічно вигідно для бізнесу і має соціальну значущість для суспільства. Така платформа сприяє розвитку підприємництва, цифрових навичок і забезпечує гнучкі умови роботи.

Об'єкт дослідження – сфера взаємодії між роботодавцями та виконавцями, зокрема платформи для пошуку фрілансерів та замовлень.

Мета роботи – розробка веб-сайту для пошуку роботи в мережі Інтернет, який забезпечуватиме ефективну співпрацю між замовниками та фрілансерами.

Методи дослідження – системний аналіз для визначення вимог до функціоналу платформи та оцінки ефективності існуючих рішень; порівняльний аналіз – для вивчення та порівняння сучасних технологій та інструментів розробки, моделювання, емпіричний метод – для збору та аналізу даних про користувацький досвід та вимоги до платформи.

Результати — результатом роботи є створена інтелектуальна система, яка включає наступний функціонал: реєстрація та авторизація користувачів, пошук та перегляд доступних проєктів, подання заявок на проєкти фрілансерами, управління проєктами зі сторони замовників, включаючи прийняття та відхилення заявок. Для досягнення результатів були використані сучасні технології та інструменти, такі як Spring Boot, MySQL, Spring Security, Lombok для серверної частини, React та Material-UI для клієнтської, а також JWT, MapStruct і Flyway для забезпечення безпеки, маппінгу даних та управління версіями бази даних.

BACK-END, SPRING BOOT, FRONT-END, REACT, MATERIAL UI, JAVA, JAVASCRIPT, LOMBOK

ЗМІСТ

ВСТУП

Актуальність роботи. У світі, насиченому технологіями та постійною потребою в ефективних рішеннях, інформаційне та програмне забезпечення онлайн-ринку для співпраці між замовниками та фрілансерами виявляється надзвичайно актуальним. Ця платформа створює умови для замовників знайти відповідних фахівців для виконання різноманітних завдань та для фрілансерів знайти проєкти, що відповідають їхнім навичкам і інтересам.

Однією з ключових складових такої платформи є бекенд [1] – цечастина програмного забезпечення, яка відповідає за обробку та зберігання даних. Бекенд забезпечує коректну взаємодію між користувачами, зберігає дані про їхні проєкти та управляє логікою роботи системи. Часто використовується сполучною з frontend, ця частина допомагає створити вебінтерфейс, який користувачі можуть використовувати для взаємодії з платформою. Frontend відповідає за створення привабливого та інтуїтивно зрозумілого інтерфейсу, який би забезпечував зручну навігацію та високу функціональність.

Мета кваліфікаційної роботи полягає в розробці та реалізації комплексної системи, що сприятиме зручній та ефективній співпраці між замовниками та фрілансерами. Шляхом використання сучасних технологій та оптимального планування ресурсів ми прагнемо створити платформу, яка відповідає вимогам сучасного онлайн-ринку, забезпечуючи зручну навігацію, безпеку даних та високу швидкість роботи.

Предметом дослідження є методи та технології, що використовуються для створення онлайн платформи для співпраці між замовниками та фрілансерами.

Об'єктом дослідження є сфера взаємодії між роботодавцями та виконавцями, зокрема онлайн платформи для пошуку фрілансерів та замовлень.

Гіпотеза дослідження полягає в тому, що розробка зручного та функціонального веб-сайту для пошуку роботи в мережі Інтернет з використанням сучасних технологій (Spring Boot для бекенду та React для

фронтенду) сприятиме підвищенню ефективності співпраці між замовниками та фрілансерами.

Новизна фінального результату полягає у створенні інтегрованої системи, що забезпечує зручний інтерфейс для користувачів, високу швидкість роботи та безпеку даних, що відповідає сучасним вимогам ринку онлайн співпраці між замовниками та фрілансерами.

Структура роботи складається зі вступу, огляду сучасних технологій розробки вебзастосунків, постановки задачі, вибору мов програмування та фреймворків, практичної реалізації та огляду використання програмного додатку, висновків, списку використаних джерел та додатків.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Онлайн ринок для співпраці між замовниками та фрілансерами.

Фріланс [2] – це форма зайнятості, при якій людина працює незалежно від роботодавця, зазвичай на віддаленій основі. Фрілансери можуть виконувати різноманітні завдання: від написання текстів та дизайну до програмування та консультацій. Гнучкий графік роботи, можливість працювати з будь-якої точки світу та вибір проєктів за власним бажанням роблять фріланс привабливою формою зайнятості для багатьох людей.

Системи співпраці між замовниками та фрілансерами, як ця, мають на меті спростити пошук та вибір проєктів, а також забезпечити надійну комунікацію між сторонами. Це дозволяє замовникам знаходити професіоналів для виконання своїх завдань швидко та ефективно, тоді як фрілансерам надається можливість знайти цікаві проєкти та заробляти на них. Такі системи створюють сприятливе середовище для розвитку бізнесу, сприяють збільшенню кількості замовлень та розширенню можливостей для фахівців у різних галузях.

1.2 Існуючі рішення

На ринку фрілансу існують кілька відомих платформ, які сприяють співпраці між замовниками та фрілансерами.

Однією з найпопулярніших є Upwork. Upwork [3] – це онлайн-платформа, що надає послуги для пошуку та укладання контрактів між фрілансерами та замовниками. Заснована в 2015 році після об’єднання двох популярних платформ Elance та oDesk. Цей ресурс надає можливість фрілансерам з усього світу заробляти гроші, працюючи над проєктами різної складності та обсягу, тоді як замовники можуть знайти спеціалістів для вирішення своїх завдань.

Серед переваг Upwork можна виділити широкий вибір фахівців у різних галузях, можливість вибору з урахуванням відгуків та рейтингів, а також зручна

система платежів та забезпечення безпеки транзакцій. Крім того, платформа надає інструменти для відстеження прогресу проєктів та спілкування між сторонами.

Однак, серед недоліків Upwork можна відзначити високу конкуренцію серед фрілансерів, що може призвести до складнощів у виграші проєктів, а також наявність комісії платформи, яка зменшує дохід фахівців. Крім того, деякі користувачі скаржаться на якість послуг та підтримки клієнтів.

Upwork користується популярністю серед фрілансерів та замовників з усього світу, особливо серед компаній, які шукають віддалених співробітників для виконання різноманітних завдань. За останні роки платформа значно зросла в популярності і стала однією з провідних у своєму сегменті.

Ще однією популярною платформою є Freelancer [4], яка також пропонує широкий вибір проєктів у різних галузях. Freelancer – це одна з найпопулярніших платформ для фрілансу, яка дозволяє замовникам та фахівцям від усього світу знаходити один одного для виконання різноманітних завдань та проєктів. Заснований у 2009 році, Freelancer став місцем, де мільйони людей збираються, щоб знайти роботу або фахівців для своїх проєктів.

Однією з ключових переваг Freelancer є широкий спектр послуг та проєктів у різних галузях, від програмування до дизайну та маркетингу. Це дозволяє як замовникам, так і фахівцям знаходити роботу або фахівців, що відповідають їхнім потребам та вимогам.

Однак, серед недоліків Freelancer можна відзначити високий рівень конкуренції серед фахівців, що може зробити складним отримання проєктів, а також наявність комісії платформи, яка може зменшувати прибуток фахівців. Крім того, деякі користувачі вказують на труднощі у взаємодії з платформою та нестабільність деяких проєктів. Незважаючи на це, Freelancer залишається однією з провідних платформ у світі фрілансу і продовжує привертати увагу мільйонів користувачів як замовників, так і фахівців. Вона є важливим ресурсом для тих, хто шукає можливість працювати віддалено або знаходити кваліфікованих фахівців для виконання різних проєктів.

Крім того, варто згадати про Fiverr. Fiverr - це інтернет-платформа, що дозволяє фахівцям з усього світу пропонувати свої послуги в різних сферах, від дизайну до копірайтингу, від програмування до маркетингу. Заснована в 2010 році, Fiverr вирізняється своєю концепцією "послуг за 5 доларів", що стало одним із факторів популярності сервісу. На сьогоднішній день платформа розвинулася і стала місцем, де можна знайти фахівців на будь-які потреби та бюджети.

Однією з переваг Fiverr є широкий вибір фахівців та послуг, що дозволяє замовникам знаходити відповідних експертів для своїх проєктів [5]. Крім того, платформа має систему рейтингу та відгуків, яка допомагає користувачам приймати обґрунтовані рішення при виборі фахівця.

Проте, серед недоліків Fiverr можна відзначити обмеження на початковий ціновий поріг для послуг, що може обмежити можливості фахівців з встановленням конкурентоспроможних цін. Крім того, іноді можуть виникати проблеми з якістю послуг, оскільки платформа не завжди забезпечує контроль якості виконання. Незважаючи на це, Fiverr залишається популярним вибором для тих, хто шукає швидкі та доступні послуги в різних галузях.

Платформи, такі як Upwork, Freelancer та Fiverr, є важливими гравцями у сучасному інтернет-ринку фрілансу та послуг. Кожна з них має свої унікальні особливості, які приваблюють як фахівців, так і замовників.

Upwork відомий своєю великою базою фахівців та різноманітністю послуг, що пропонуються, разом з системою захисту та оплати. Freelancer славиться своєю широкою географічною розповсюдженістю та можливістю конкурентної ціноутворення. Fiverr привертає увагу своїми низькими вартостями послуг та широким спектром фахівців. Незважаючи на переваги, кожна платформа також має свої недоліки. Наприклад, на Upwork можуть бути високі комісійні витрати, на Freelancer - можливі проблеми з якістю та спроможність фахівців, а на Fiverr - обмеженість на початковий ціновий поріг для послуг. Однак з урахуванням цих чинників, ці платформи залишаються популярними серед користувачів, які шукають якісні та доступні послуги в онлайн-середовищі.

1.3 Постановка задачі

Постановка задачі до інформаційного та програмного забезпечення онлайн-ринку для співпраці між замовниками та фрілансерами є важливим етапом розробки проєкту. На даному етапі визначаються основні цілі та завдання, а також визначається спосіб до їх досягнення. Тому для успішної розробки слід виконати наступні кроки:

1. Аналіз потреб користувачів:

- визначення потреб та очікувань замовників та фрілансерів від платформи.
- вивчення сучасних тенденцій та вимог ринку фрілансу.

2. Визначення функціональних вимог:

- створення системи реєстрації користувачів з можливістю створення облікових записів для замовників та фрілансерів.
- розробка механізмів пошуку та фільтрації профілів фрілансерів за різними критеріями.
- впровадження системи оцінювання та відгуків для забезпечення якості роботи фрілансерів.

3. Визначення нефункціональних вимог:

- забезпечення безпеки та конфіденційності персональних даних користувачів.
- забезпечення високої швидкодії та надійності платформи навіть при великому навантаженні.

4. Розробка архітектури системи:

- вибір технологій та інструментів для реалізації проєкту.
- проєктування бекенд та фронтенд частин платформи.

5. Планування етапів розробки:

- розподіл завдань між командою розробників.
- визначення термінів виконання кожного етапу та проєкту в цілому.

Ці кроки допоможуть створити чітке і реалізоване планування проєкту, що сприятиме його успішному втіленню та подальшому розвитку.

2. ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧ

2.1 Вибір фреймворку

Мова програмування Java є однією з найпопулярніших [6] мов програмування. Її широке використання пояснюється багатьма факторами, включаючи велику кількість доступних бібліотек і фреймворків, високий рівень переносимості програмного коду між різними платформами, надійність та безпека, а також простота використання. Крім того, Java є мовою програмування, яка застосовується в широкому спектрі областей, від веброзробки до розробки мобільних додатків та корпоративних систем. Крім того, Java добре підходить для створення високопродуктивних, масштабованих та безпечних систем, що є важливими аспектами у проєкті, який передбачає обробку конфіденційної інформації та великий потік даних. Таким чином, вибір мови програмування Java є обґрунтованим і відповідає потребам проєкту.

Spring Framework [7] є одним з найпопулярніших і потужних фреймворків для розробки вебдодатків мовою програмування Java. Він надає широкий спектр можливостей, спрощуючи процес розробки та підтримки програмного забезпечення. Spring пропонує модулярну архітектуру, що дозволяє розробникам використовувати лише ті компоненти, які необхідні для їх проєктів, що гарантує гнучкість та ефективність у використанні.

Основною складовою Spring Framework є модуль Spring Core, який надає базовий функціонал контейнера IoC (Inversion of Control) та виконує управління бінами (класами або об’єктами), інжекцію залежностей та керування життєвим циклом об’єктів. Крім того, Spring має ряд допоміжних модулів, таких як Spring Boot, Spring Security, Spring Data JPA, Spring MVC, що надають додатковий функціонал для розробки різноманітних видів додатків.

Spring Boot, зокрема, є надбудовою над Spring Framework, яка дозволяє швидко створювати самостійні, автономні додатки з мінімальною конфігурацією. Це робить процес розгортання та управління додатками значно

простішим та швидшим. Spring Security забезпечує механізми автентифікації та авторизації, що є критичними для захисту даних та забезпечення безпеки додатків. Крім того, Spring Data JPA дозволяє легко взаємодіяти з базами даних за допомогою стандартів JPA (Java Persistence API), спрощуючи розробку і підтримку шару доступу до даних.

Micronaut – це модерний фреймворк для розробки мікросервісів та вебдодатків на Java, Kotlin і Groovy. Він відрізняється від інших фреймворків тим, що пропонує розробникам рішення, які спрощують роботу і підвищують продуктивність. Основними перевагами Micronaut є швидкодія та низький рівень споживання пам’яті завдяки використанню анотацій для виконання деяких завдань ще на етапі компіляції коду. Це дозволяє зменшити вартість підтримки додатків і підвищити їх масштабованість.

Однією з ключових особливостей Micronaut є його підтримка GraalVM, що дозволяє компілювати додатки в нативний код, що пришвидшує їх запуск та зменшує витрати пам’яті. Крім того, Micronaut використовує вбудовану підтримку реактивного програмування для забезпечення ефективної роботи з асинхронними операціями і обробки багатьох запитів одночасно.

Ще однією перевагою Micronaut є його інтеграція з різноманітними технологіями і фреймворками, такими як Spring, Hibernate, RxJava, Reactor і багато інших. Це дозволяє розробникам використовувати свої улюблені інструменти та технології разом із Micronaut, щоб створювати потужні та ефективні додатки.

Ktor – це легкий і гнучкий фреймворк для створення вебдодатків на Kotlin. Він відомий своєю простотою використання та високою продуктивністю. Основними перевагами Ktor є його асинхронна природа та підтримка реактивного програмування, що дозволяє ефективно обробляти багатопоточні запити та операції в реальному часі.

Ключова перевага Ktor – це його низький поріг входження для початківців. Він має простий API та добре документовану платформу, яка дозволяє швидко розпочати роботу над проєктами. Крім того, Ktor ідеально

підходить для мікросервісної архітектури, оскільки дозволяє швидко створювати невеликі та швидкі сервіси з мінімальними зусиллями.

Підсумовуючи розглянуті фреймворки, кожен з них має свої унікальні переваги та особливості. Spring Framework є стандартом для багатьох Java-розробників і надає велику кількість інструментів для швидкого розгортання складних додатків.

JavaScript [8] – це одна з найпопулярніших мов програмування, яка використовується для розробки вебдодатків. Вона є мовою сценаріїв на клієнтській стороні, що дозволяє динамічно змінювати вміст вебсторінок, взаємодіяти з користувачем та взаємодіяти з вебсерверами. Її популярність пояснюється простотою вивчення, гнучкістю та широким спектром інструментів і бібліотек, які дозволяють розробникам створювати різноманітні та інноваційні продукти. JavaScript також постійно розвивається, з’являються нові фреймворки, бібліотеки та стандарти, що дозволяє розробникам залишатися в актуальному стані та впроваджувати нові технології у свої проєкти.

React.js – це відкритий JavaScript фреймворк [9], розроблений компанією Facebook. Він використовується для створення інтерфейсів користувача, особливо вебдодатків і односторінкових додатків. React відомий своєю декларативністю та компонентною структурою, що робить його дуже зручним для розробки та підтримки складних інтерфейсів.

Однією з ключових особливостей React є використання віртуального Document Object Model, що дозволяє реалізувати ефективне оновлення інтерфейсу, забезпечуючи високу продуктивність додатків. Компоненти React можуть бути легко перевикористані та комбіновані, що сприяє створенню модульних інтерфейсів та забезпечує чистоту коду.

Ще однією перевагою React є наявність широкого екосистеми бібліотек та інструментів, що розширюють його можливості. Наприклад, Redux – це популярна бібліотека для керування станом додатків, яка часто використовується разом з React для управління складними даними. Крім того,

React має активну спільноту розробників, яка постійно вносить нові ідеї та розробляє різноманітні розширення для фреймворку.

Загалом, React є потужним інструментом для створення сучасних вебдодатків з високою швидкістю, ефективністю та розширюваністю. Він надає розробникам можливість швидко та ефективно створювати інтерфейси, які забезпечують приємний користувальницький досвід.

Angular і Vue.js – це два популярних фреймворки JavaScript, які також використовуються для розробки вебдодатків. Обидва фреймворки володіють великою кількістю функціональних можливостей і дозволяють розробникам створювати потужні, масштабовані додатки.

Angular – це фреймворк, розроблений командою Google, і відомий своєю повноцінною платформою для створення вебдодатків. Він включає в себе багато вбудованих інструментів, таких як маршрутизація, обробники подій і керування станом за допомогою Angular Service. Angular також має велику спільноту розробників та широкий набір офіційно підтримуваних бібліотек і модулів.

Vue.js – це прогресивний фреймворк JavaScript, який дозволяє створювати користувацькі інтерфейси. Він вирізняється своєю простотою використання і легкістю вивчення. Vue.js має гнучку архітектуру і дозволяє розробникам використовувати його як бібліотеку для створення інтерфейсів в невеликих проєктах або як повноцінний фреймворк для великих застосунків.

Після ретельного аналізу і порівняння фронтенд фреймворків, було прийнято рішення вибрати React.js для розробки вебзастосунку. React.js визначається своєю простотою використання, гнучкістю і широким спектром можливостей, що робить його одним з найпопулярніших інструментів у світі веброзробки.

2.3 Вибір бази даних

При виборі баз даних для проєкту інформаційного та програмного забезпечення онлайн-ринку для співпраці між замовниками та фрілансерами,

важливо розглянути різноманітні аспекти, такі як тип даних, вимоги до продуктивності, масштабованість та інтеграційні можливості. Одним з ключових виборів є вибір між реляційними та нереляційними базами даних, відомими як SQL та NoSQL відповідно [10].

Реляційні та нереляційні бази даних є двома основними типами систем для зберігання та організації даних. Реляційні бази даних (SQL) базуються на моделі зв'язків між різними таблицями, де дані представлені у вигляді таблиць з рядками і стовпцями. Цей підхід забезпечує структурованість та надійність даних, дозволяє виконувати складні запити та забезпечує додаткові можливості, такі як транзакції та індексація.

Нереляційні бази даних (NoSQL) використовують альтернативні моделі зберігання даних, такі як ключ-значення, документи, стовпці або графи. Ці системи надають більшу гнучкість у роботі з невстановленою або змінюваною структурою даних, а також дозволяють швидше масштабувати систему та виконувати розподілені операції. Вони часто використовуються в вебпроєктах, де потрібно обробляти великі обсяги даних або забезпечити високу доступність.

Обираючи між реляційними та нереляційними базами даних, важливо враховувати вимоги проєкту, типи даних, які потрібно зберігати, та потреби у масштабованості та продуктивності. Реляційні бази даних зазвичай краще підходять для проєктів з чіткою структурою даних та потребою у складних операціях, тоді як нереляційні бази даних можуть бути корисні для проєктів, які вимагають швидкості, гнучкості та масштабованості. Крім того, комбінування обох типів баз даних може бути ефективним рішенням для деяких проєктів, де різні типи даних можуть зберігатися та оброблятися відповідно до їх характеристик.

MySQL, є одним з найпопулярніших варіантів, особливо в середовищі веброзробки, відомий своєю швидкодією та надійністю, а також широким спектром функцій і можливостей. Він широко підтримується різними платформами та мовами програмування, що робить його дуже зручним для використання в різних проєктах.

Іншою альтернативою є Microsoft SQL Server, який відомий своєю масштабованістю і можливостями для великих корпоративних систем. SQL Server надає розширені функції для управління даними та безпеки, а також інтеграцію з іншими продуктами Microsoft, що робить його зручним вибором для організацій, які вже використовують інші продукти цієї компанії.

Додатковою альтернативою може бути Oracle Database, що також відома своєю надійністю та масштабованістю. Oracle Database має велику кількість функцій та можливостей, таких як розподілена база даних та великі можливості аналітики, що робить його популярним вибором для великих підприємств та організацій з великим обсягом даних.

У результаті аналізу різних SQL баз даних для проєкту було прийнято рішення обрати MySQL – одн з найпопулярніших та надійних варіантів серед SQL баз даних, особливо в контексті веброзробки.

2.4 Вибір місця зберігання коду

Зберігання коду відіграє ключову роль у процесі розробки програмного забезпечення. Платформи для зберігання коду, такі як GitHub, GitLab або Bitbucket, забезпечують централізований доступ до кодової бази для всіх членів команди розробників. Це сприяє спільній роботі, спілкуванню та взаємодії між учасниками проєкту. Кожен розробник може створювати власні гілки, вносити зміни та пропонувати їх для обговорення та злиття з основною кодовою базою.

Крім того, платформи для зберігання коду забезпечують контроль версій, що дозволяє відстежувати історію змін у коді. Це допомагає у виявленні помилок, відновленні попередніх версій та вирішенні конфліктів під час розвитку проєкту. Також це забезпечує надійність і безпеку, оскільки можна легко відновити попередні версії коду в разі потреби.

Зберігання коду на платформі також сприяє спільноті розробників. Розробники можуть співпрацювати, надавати відкритий доступ до свого коду для інших користувачів, вносити внески в відкриті проєкти та спілкуватися з іншими

членами глобальної розробницької спільноти. Такий підхід дозволяє прискорити розвиток програмного забезпечення та вирішення проблем шляхом спільної роботи та обміну досвідом.

Сервіси для зберігання коду, такі як GitHub, GitLab та Bitbucket, надають розробникам зручне та ефективне середовище для спільної роботи над проєктами. GitHub, наприклад, є найбільш популярною платформою для зберігання коду, де розробники можуть створювати репозиторії, вносити зміни, обговорювати питання та злити зміни з основною кодовою базою.

GitLab, у свою чергу, пропонує широкий набір інструментів для керування проєктами та автоматизації процесів розробки, включаючи системи звітності, CI/CD і інші.

Bitbucket, який належить компанії Atlassian, також є популярним сервісом для зберігання коду, зокрема серед команд, які використовують інші продукти Atlassian, такі як Jira або Confluence. Bitbucket надає інтеграцію з цими інструментами, що дозволяє розробникам зручно взаємодіяти з кодом та відстежувати зміни у відповідності до управління проєктами.

Загалом, сервіси для зберігання коду забезпечують розробникам місце для спільної роботи над проєктами, контролю версій, спілкування та спільного внеску до кодової бази, що допомагає прискорити розробку програмного забезпечення та підвищити продуктивність команди.

GitHub [11] обрано як місце для зберігання коду в цьому проєкті через його широкий функціонал та популярність серед розробників у всьому світі. Завдяки своїм потужним інструментам для контролю версій, керування проєктами та спільної роботи, GitHub забезпечує ефективне спілкування та співпрацю всередині команди розробників. Крім того, інтеграція з іншими популярними інструментами розробки, такими як Jira або Slack, робить GitHub зручним вибором для реалізації цього проєкту.

3. ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Побудова структури папок серверної частини проєкту

Будуємо структуру папок як показано на рисунку 3.1:

- Папка config – включатиме налаштування проєкту в тому числі безпекові;
- Папка dao – має сутності для роботи з базою даних, власний маппер за допомгою якого буде відбуватися перетворення різних класів а також DTOs, які являють собою шаблон проєктування, який використовується для передачі даних між різними частинами програми або між різними системами. DTO зазвичай не містить бізнес-логіки або методів для маніпуляції даними, а служить лише для зберігання та транспортування даних.
- Папка repository – містить інтерфейси, які відповідають за доступ до даних і виконання CRUD-операцій над сутностями. Ці інтерфейси зазвичай розширюють JpaRepository або інші спеціалізовані інтерфейси Spring Data JPA, що забезпечує автоматичну реалізацію основних методів роботи з базою даних, таких як збереження, пошук, оновлення та видалення записів.
- Папка rest – містить контроллери, які відповідають за обробку HTTP-запитів та відповідають за маршрутизацію та виконання бізнес-логіки. Ці контролери зазвичай позначені анотаціями, такими як @RestController і @RequestMapping, що дозволяє легко визначати шляхи та методи обробки запитів (GET, POST, PUT, DELETE). Контролери в цій папці забезпечують взаємодію між клієнтом та сервером, приймаючи дані від клієнта, обробляючи їх із використанням сервісного шару, та повертаючи відповідні відповіді, зазвичай у форматі JSON або XML.
- Папка service – містить сервіси, які реалізують бізнес-логіку програми. Сервісні класи зазвичай позначаються анотацією @Service і використовуються для обробки складних операцій, що потребують взаємодії з кількома репозиторіями або виконання специфічних бізнес-процесів. Вони виступають як

посередники між контролерами та репозиторіями, забезпечуючи чистоту та розділення обов'язків у коді. Використовуючи сервіси, можна легко підтримувати та розширювати логіку програми без зміни коду в інших шарах архітектури.

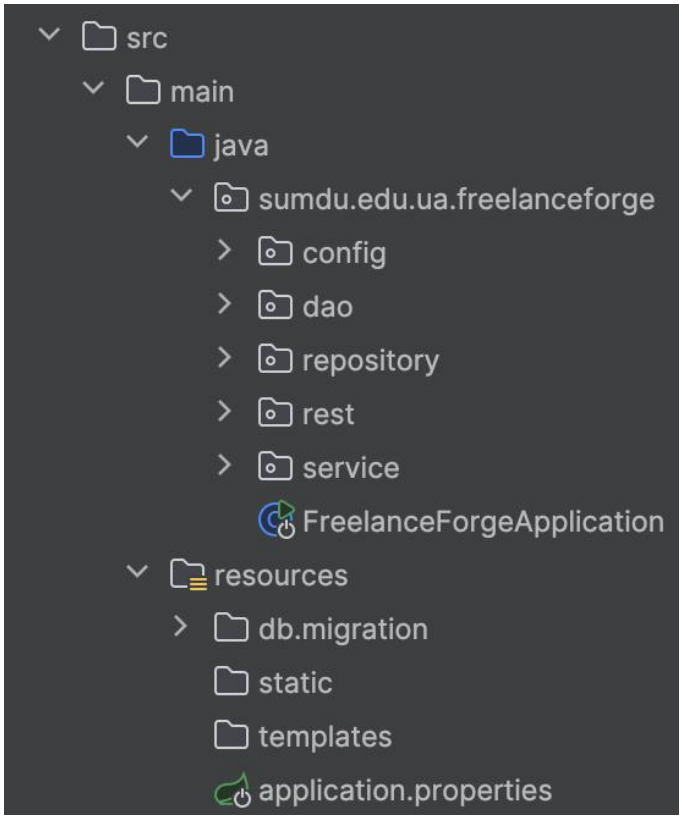

Рисунок 3.1 – Структура папок проєкту

Також маємо папку db.migration яка містить скрипти, які зчитує flyWay - інструмент для управління версіями схем баз даних, який дозволяє легко відстежувати, версіонувати та застосовувати зміни до бази даних у проєкті.

3.2 Створення головних сутностей та їх DTO об'єктів.

Спочатку створюємо сутності (рис.3.2):

- Job – сутність у проєкті, яка відображає таблицю jobs у базі даних. Вона містить поля id, client, title, description, status, createdAt та updatedAt. Поле id є унікальним ідентифікатором, який генерується автоматично. Поле client є

посиланням на об’єкт User, що вказує на клієнта, який створив роботу, реалізоване через зв’язок ManyToOne. Поля title та description зберігають заголовок та опис роботи відповідно. Поле status зберігає статус роботи, визначений через перерахування JobStatus. Поля createdAt та updatedAt зберігають час створення та останнього оновлення роботи. Використання анотацій @Entity, @Table, @Id, @GeneratedValue, @ManyToOne, @JoinColumn, @Column та @Enumerated забезпечує відповідне відображення цієї сутності у базі даних та її коректну обробку в рамках JPA.

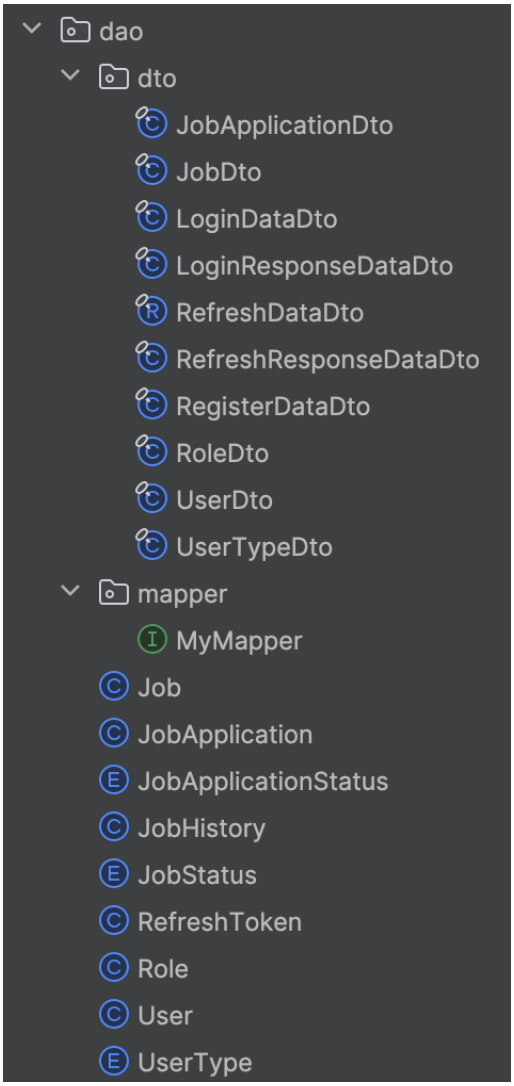

Рисунок 3.2 – Створені файли у папці dao

- JobApplication – сутність у проєкті, яка відображає таблицю job_applications у базі даних. Вона містить поля id, job, freelancer, status, createdAt

та updatedAt. Поле id є унікальним ідентифікатором, який генерується автоматично. Поле job представляє зв’язок з конкретною роботою через зв’язок ManyToOne та JoinColumn з обов’язковим значенням. Поле freelancer вказує на користувача-фрілансера, який подав заявку на роботу, також реалізоване через зв’язок ManyToOne та JoinColumn з обов’язковим значенням. Поле status зберігає статус заявки на роботу, визначений через перерахування JobApplicationStatus. Поля createdAt та updatedAt зберігають час створення та останнього оновлення заявки.

- JobApplicationStatus – перерахування (enum) у проєкті, яке визначає можливі статуси заявки на роботу. Воно містить три значення: PENDING, ACCEPTED та REJECTED. Ці статуси використовуються для відображення стану заявки на роботу, зокрема чи заявка очікує розгляду, була прийнята або відхилена. Перерахування забезпечує безпечне та зручне використання цих значень у коді, гарантуючи, що можливі лише визначені стани.

- JobHistory – сутність, яка представляє таблицю job_history у базі даних. Вона містить такі поля: id, job, user, status та createdAt. Поле id є унікальним ідентифікатором, який генерується автоматично. Поле job є посиланням на роботу і встановлює зв’язок ManyToOne з обов’язковим значенням JoinColumn. Поле user представляє користувача, пов’язаного з цією історією роботи, також реалізоване через зв’язок ManyToOne з обов’язковим значенням JoinColumn. Поле status зберігає статус роботи в цей момент часу. Поле createdAt зберігає час створення цього запису в історії. Сутність JobHistory використовується для відстеження змін статусу робіт та дій користувачів, пов’язаних з цими роботами.

- JobStatus – перерахування, яке визначає можливі статуси роботи. Воно містить чотири значення: OPEN, IN_PROGRESS, COMPLETED та CANCELED. Ці статуси використовуються для відображення поточного стану роботи, такого як чи відкрита, чи в процесі виконання, чи завершена, чи скасована. Використання перерахування забезпечує безпечне та зручне використання цих значень у коді, гарантуючи, що можливі лише визначені стани.

- RefreshToken – сутність, яка представляє таблицю refresh_tokens у базі даних. Вона містить поля id, user, token та expiryDate. Поле id є унікальним ідентифікатором, який генерується автоматично. Поле user вказує на користувача, пов’язаного з цим токеном, і встановлює зв’язок OneToOne з таблицею користувачів через JoinColumn. Поле token зберігає сам токен оновлення. Поле expiryDate вказує на дату закінчення терміну дії токена. Використання анотацій JPA дозволяє відображати цю сутність у базі даних та забезпечує легку інтеграцію з іншими складовими системи.

- Role – сутність, яка представляє таблицю roles у базі даних. Вона містить поля id та name, які відповідають ідентифікатору ролі та її назві відповідно. Поле id є унікальним ідентифікатором, який генерується автоматично. Поле name зберігає назву ролі. Ця сутність використовується для визначення ролей користувачів у системі. Вона може мати відношення багато-до-багатьох з сутністю User, що дозволяє одній ролі мати багатьох користувачів, а також кожен користувач може мати декілька ролей. Відношення встановлено через анотацію @ManyToMany, а параметр mappedBy вказує на поле roles у сутності User, яке відповідає за зв’язок між користувачем та його ролями. FetchType встановлено як LAZY, щоб уникнути надмірного завантаження даних при витягуванні ролей.

- User – сутність, яка представляє таблицю users у базі даних. Вона містить такі поля: id, username, firstName, lastName, password, userType, roles, created та updated. Поле id є унікальним ідентифікатором, який генерується автоматично. Поля username, firstName, lastName та password відповідають особистим даним користувача. Поле userType визначає тип користувача та використовується перерахуванням UserType. Поле roles відображає відношення багато-до-багатьох до сутності Role, що вказує на ролі користувача. Воно використовується для визначення ролей, які призначені конкретному користувачеві. FetchType встановлено як EAGER, щоб забезпечити вибірку ролей разом з користувачем у випадку необхідності. Поля created та updated зберігають час створення та останнього оновлення запису користувача відповідно.

MyMapper – це інтерфейс, який використовується для відображення об’єктів одного типу на об’єкти іншого типу за допомогою бібліотеки MapStruct. У цьому інтерфейсі оголошені методи для відображення об’єктів різних класів на об’єкти DTO та навпаки. Наприклад, метод registerDataDtoToUser приймає об’єкт класу RegisterDataDto і повертає об’єкт класу User. Цей метод відповідає за відображення даних з об’єкту RegisterDataDto на об’єкт User, щоб можна було використовувати ці дані для реєстрації нового користувача.

Анотація @Mapper(componentModel = "spring") вказує MapStruct на те, що цей інтерфейс має бути реалізований за допомогою Spring. Це дозволяє автоматично створювати екземпляри цього інтерфейсу та використовувати їх в інших компонентах Spring, таких як сервіси чи контролери.

MyMapper допомагає у відділенні логіки перетворення об’єктів від логіки бізнес-логіки, що полегшує розробку та підтримку коду.

3.3 Створення репозиторіїв для роботи з базою даних

Файли у репозиторії (рис.3.3) freelanceforge.repository містять інтерфейси, які відповідають за доступ до даних у базі даних. Кожен інтерфейс наслідуєся від інтерфейсу JpaRepository, що надає стандартні методи для виконання операцій над даними, таких як збереження, пошук, оновлення та видалення. Наприклад, у файлі JobApplicationRepository описаний інтерфейс, який відповідає за роботу з сутністю JobApplication. Цей інтерфейс містить методи для взаємодії з об’єктами JobApplication, такі як збереження, видалення та пошук за ідентифікатором чи іншими критеріями. Використання таких інтерфейсів дозволяє легко та ефективно взаємодіяти з даними у базі даних без написання власного SQL-коду для кожного запиту.

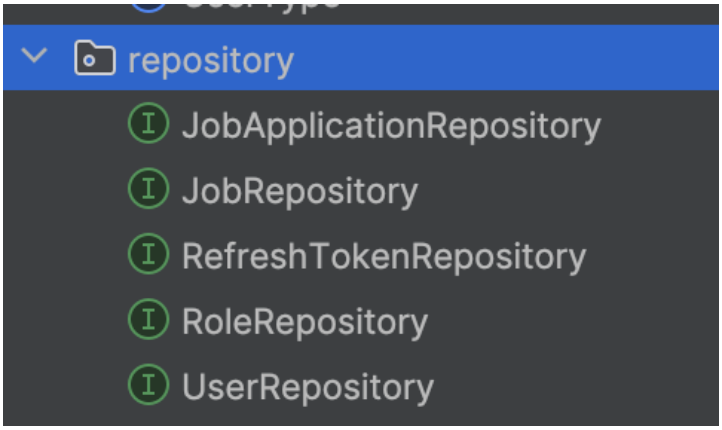

Рисунок 3.3 – Папка з репозиторіями

Ці файли є ключовими компонентами у взаємодії з базою даних у Spring Boot додатку. Вони дозволяють використовувати потужності Spring Data JPA для простого та зручного доступу до даних, спрощуючи розробку та підтримку програмного забезпечення.

3.4 Створення потрібних налаштувань проєкту

У конфігураційних файлах (рис.3.4) у пакеті sumdu.edu.ua.freelanceforge.config забезпечується налаштування аутентифікації та авторизації у додатку. Клас SecurityConfig є конфігураційним файлом для Spring Security. Він включає у себе встановлення прав доступу до різних URL-шляхів, налаштування створення сесій та встановлення фільтру для обробки запитів аутентифікації.

SecurityFilter – це фільтр, який використовується для обробки та перевірки токенів авторизації у кожному HTTP-запиті. Він перевіряє наявність токенів у заголовку Authorization, а також перевіряє їх правильність та час дії. У випадку вичерпання терміну дії токена, фільтр відповідає із статусом UNAUTHORIZED.

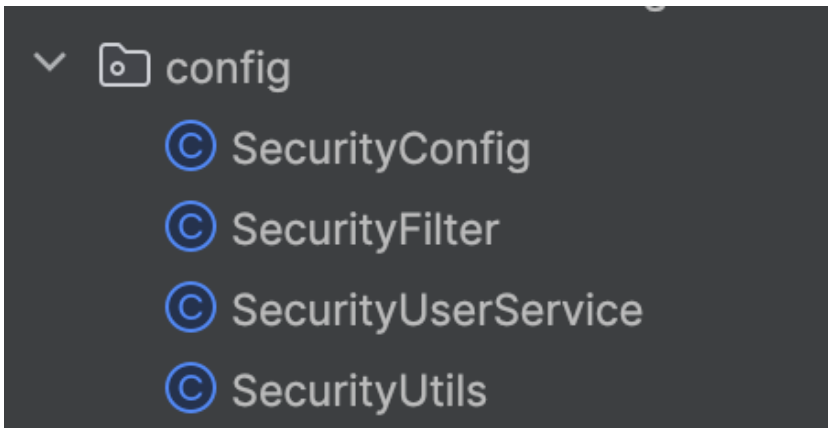

Рисунок 3.4 – Файли налаштування проєкту

SecurityUserService – це сервіс, який використовується для завантаження даних користувача за його ім’ям користувача у Spring Security. Він реалізує інтерфейс UserDetailsService та використовує інші сервіси, наприклад, userService, для отримання даних про користувача з бази даних та перетворення їх у об’єкт UserDetails, який використовується для аутентифікації та авторизації користувача.

SecurityUtils – це утилітарний клас, який забезпечує роботу з токенами авторизації. Він містить методи для генерації, перевірки та отримання даних з JWT-токенів. Цей клас використовується у фільтрі SecurityFilter для обробки та перевірки токенів у кожному HTTP-запиті.

3.5 Створення сервісів для імплементації бізнес логіки

У папці сервісів реалізовано логіку бізнес-процесів додатку. Ось деякі ключові аспекти цих сервісів:

1. JobApplicationService: цей сервіс відповідає за логіку пов’язану з заявками на роботу. Він має методи для подання заявки на роботу, створення нової заявки, отримання всіх заявок, прийняття або відхилення заявки, отримання проєктів користувача та перевірки, чи була заявка на проєкт прийнята.

2. JobService: цей сервіс відповідає за логіку пов’язану з управлінням проєктами. Він має методи для отримання всіх проєктів, отримання проєкту за його ідентифікатором, зміни статусу проєкту, створення нового проєкту та видалення проєкту.

3. RefreshService: цей сервіс відповідає за управління оновленням токенів. Він має методи для створення нового оновлення токену, пошуку оновлення токену за його значенням, перевірки терміну дії оновлення токену та видалення оновлення токену за ідентифікатором користувача.

4. UserService: цей сервіс відповідає за логіку пов’язану з користувачами. Він має методи для пошуку користувача за ім’ям користувача, реєстрації нового користувача, отримання користувача за його ідентифікатором та отримання інформації про користувача в форматі DTO.

Ці сервіси містять логіку, що забезпечує функціональність додатку та взаємодію з базою даних. Вони допомагають у виконанні бізнес-логіки та забезпеченні відповідного функціоналу користувачам.

3.5 Створення структури папок майбутньої клієнтської частини додатку

Структура папок доволі невелика, маємо папку components яка включає основні компоненти проєкту, так як було використано React фреймворк, у якому все будується на компонентах, які перевикористовуються по всьому проєкту, папка pages також має компоненти але які відповідають за повноцінні сторінки вебсайту.

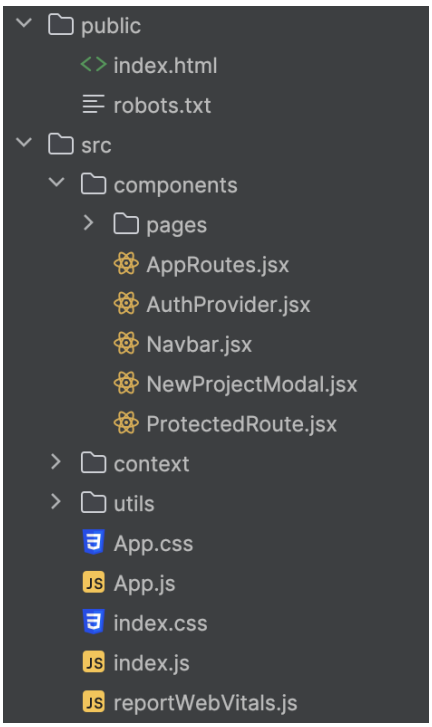

Рисунок 3.5 – Структура папок front-end проєкту

Папка context матиме файли з налаштуваннями контекстів, у нашому випадку пов’язаного з безпекою. Папка utils матиме допоміжні файли та функції для роботи проєкту.

3.7 Створення допоміжних файлів для роботи з проєктом

У папці utils створені два файли, що містять допоміжні функції для взаємодії з сервером та роботи з автентифікацією в React додатку.

1. api.jsx: Цей файл містить змінну API_HOST, яка визначає базову URL-адресу сервера. Також містить об’єкт endpoints, у якому зберігаються шляхи до різних ендпоінтів на сервері. Файл також містить функцію fetchWrapper, яка виконує запити до сервера з врахуванням CORS-політики та форматує запити у форматі JSON.

2. auth.jsx: Цей файл містить набір функцій для роботи з автентифікацією. Він містить функції для отримання, збереження та очищення даних користувача в локальному сховищі браузера, а також функції для отримання, оновлення та

перевірки токенів доступу та оновлення. Також в ньому міститься функція для отримання об’єкта користувача з токену.

Ці файли допомагають забезпечити зручний та ефективний спосіб взаємодії з сервером та роботи з автентифікацією в React додатку.

У папці context створений файл, який містить визначення контексту для автентифікації в React додатку. Файл AuthContext.jsx використовує функцію createContext з бібліотеки React для створення контексту під назвою AuthContext. Цей контекст забезпечує доступ до даних та методів, пов’язаних з автентифікацією користувачів.

Структура контексту включає:

- token: змінна для зберігання токену автентифікації користувача.

- onLogin: функція-заглушка для входу користувача. Її можна буде замінити на реальну функцію під час налаштування провайдера контексту.

- onLogout: функція-заглушка для виходу користувача.

Цей контекст використовується для забезпечення глобального доступу до даних та методів автентифікації в різних компонентах додатку, що робить управління станом автентифікації більш централізованим та зручним.

3.8 Створення основних файлів сторінок вебсайту

3.8.1 ClientApplications

Цей компонент призначений для відображення списку заявок, які отримав клієнт на свої проєкти.

1. Ініціалізація станів і хуків

- Використовується useState для керування станом заявок, доступного токена та помилки.

- useNavigate з react-router-dom для навігації між сторінками.

- useEffect для завантаження заявок при монтуванні компонента.

2. Функції

- fetchJobAccepted: асинхронна функція, яка перевіряє, чи було прийнято заявку для певного проєкту.

- fetchApplications: асинхронна функція, яка отримує всі заявки клієнта.

- handleViewProfile: функція для навігації на сторінку профілю заявника.

- handleAccept: функція для прийняття заявки на проєкт.

- handleReject: функція для відхилення заявки на проєкт.

3. Рендеринг

- Використовує MUI компоненти для побудови інтерфейсу: Container, Grid, Typography, Card, CardContent, Button, Alert.

- Відображає кожну заявку у вигляді картки з кнопками для перегляду профілю, прийняття або відхилення заявки.

- Виводить повідомлення про помилку, якщо проєкт вже має прийняту заявку.

3.8.2 FreelancerProfile

Цей компонент відповідає за відображення профілю фрілансера, включаючи його проєкти.

1. Ініціалізація станів і хуків

- Використовується useState для керування станом проєктів, користувача та станом завантаження.

- useParams для отримання ID фрілансера з URL.

- useNavigate для навігації між сторінками.

2. Функції

- fetchProfile: асинхронна функція для отримання проєктів фрілансера.

- fetchUser: асинхронна функція для отримання даних фрілансера.

- getStatusClass: функція для визначення класу CSS для статусу проєкту.

3. Рендеринг

- Використовує MUI компоненти для побудови інтерфейсу: Container, Grid, Typography, Card, CardContent, CardActions, Button, CircularProgress, Avatar.

- Відображає інформацію про фрілансера, включаючи ім’я, прізвище та користувацьке ім’я.

- Відображає список проєктів фрілансера з відповідним статусом.

- Кнопка для повернення до списку заявок.

Обидва компоненти активно використовують API для отримання даних та оновлення інтерфейсу в режимі реального часу. Вони інтегруються з навігацією для зручного переходу між сторінками та динамічного відображення інформації.

3.8.3 Home

Цей компонент відображає домашню сторінку, яка привітає користувачів і представляє основні функції платформи.

1. Ініціалізація станів і хуків

- Використовується useStyles для застосування стилів з makeStyles.

- Використовується getAccessToken з "../../utils/auth" для отримання токена доступу.

2. Рендеринг

- Відображає привітальне повідомлення та заклик до дії для нових користувачів (кнопка "Get Started", яка веде на сторінку реєстрації).

- Відображає основні функції платформи у вигляді карток: "Find Projects", "Manage Profile", "Showcase Portfolio", "Get Reviews".

- Відображає заклик до приєднання до спільноти фрілансерів для нових користувачів.

3. Стилі

- Стилі визначені для різних частин сторінки, таких як заголовок, кнопки, функціональні елементи, заклики до дії та інше.

3.8.4 Login

Цей компонент відображає сторінку входу для користувачів.

1. Ініціалізація станів і хуків

- Використовуються useState для керування станом введених даних (ім’я користувача, пароль) та станом помилки входу.

- Використовується useAuth з "../../utils/auth" для аутентифікації користувача.

- Використовується useStyles для застосування стилів з makeStyles.

2. Функції

- handleAlertClose: закриває повідомлення про помилку.

- handleLogin: обробляє спробу входу, відправляючи запит на сервер і оновлюючи стан залежно від відповіді.

3. Рендеринг

- Відображає форму для введення імені користувача та пароля.

- Відображає кнопку для входу.

- Відображає посилання для переходу на сторінку реєстрації.

- Відображає повідомлення про помилку, якщо дані для входу некоректні.

4. Стилі

- Стилі визначені для компонента, включаючи форму, кнопку входу, повідомлення про помилку та інше.

3.8.5 ProjectDetails

Цей компонент відображає деталі конкретного проєкту.

1. Ініціалізація станів і хуків

- Використовуються useParams для отримання ID проєкту з URL.

- Використовуються useState для керування станом проєкту, користувача, завантаження, типу користувача, стану поданої заявки та помилки.

- Використовується useNavigate для навігації між сторінками.

- Використовується getAccessToken з "../../utils/auth" для отримання токена доступу.

- Використовується useStyles для застосування стилів з makeStyles.

2. Функції

- fetchType: отримує тип користувача (фрілансер або клієнт).

- fetchProject: отримує дані проєкту з сервера.

- handleApply: обробляє подання заявки на проєкт.

- handleCompleteProject: обробляє позначення проєкту як завершеного.

- handleDelete: обробляє видалення проєкту.

3. Рендеринг

- Відображає деталі проєкту, включаючи назву, клієнта, статус, опис та дату створення.

- Відображає кнопки для подання заявки на проєкт (для фрілансерів), позначення проєкту як завершеного та видалення проєкту (для клієнтів).

- Відображає повідомлення про помилку, якщо сталася помилка під час обробки запиту.

4. Стилі

- Стилі визначені для різних частин компонента, таких як кореневий елемент, контейнер для деталей проєкту, завантажувальний індикатор, кнопки та інше.

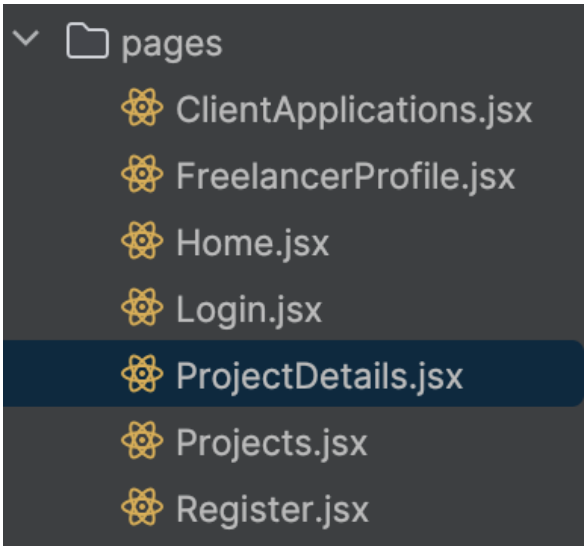

Рисунок 3.6 – Компоненти сторінок сайту

Усі компоненти використовують бібліотеку Material-UI для створення сучасного та зручного інтерфейсу. Компоненти також активно використовують асинхронні запити для взаємодії з сервером, що дозволяє отримувати та

відправляти дані в реальному часі. Структура компонентів добре організована, що робить їх легкими для розуміння та розширення.

3.9 Розробка основних компонентів

Огляд компонентів, які використовуються по всьому проєкту

3.9.1 Файл AppRoutes.jsx

Цей файл визначає маршрутизацію для додатка, використовуючи react-router-dom. Він включає в себе компоненти Navbar, AuthProvider, та ProtectedRoute для управління доступом до захищених маршрутів. Також визначені основні маршрути, такі як Home, Login, Register, Projects, ProjectDetails, ClientApplications, FreelancerProfile та маршрут за замовчуванням для обробки 404 помилок.

```jsx
function AppRoutes() {
  return (
    <Router>
      <AuthProvider>
        <Navbar/>
        <Routes>
          <Route path="/" element={<Home/>}/>
          <Route path="/home" element={<Home/>}/>
          <Route
            path="/projects"
            element={(
              <ProtectedRoute>
                <Projects/>
              </ProtectedRoute>
            )}
          />
          <Route path="/project/:projectId" element={(
            <ProtectedRoute>
              <ProjectDetails/>
            </ProtectedRoute>
          )}/>
          <Route path="/applications" element={(
            <ProtectedRoute>
              <ClientApplications/>
            </ProtectedRoute>
          )}/>
          <Route path="/profile/:id" element={(
            <ProtectedRoute>
              <FreelancerProfile/>
            </ProtectedRoute>
          )}/>
          <Route path="/login" element={<Login/>}/>
          <Route path="/register" element={<Register/>}/>
          <Route path="*" element={<p>There&apos;s nothing here: 404!</p>}/>
        </Routes>
      </AuthProvider>
    </Router>
  );
}

export default AppRoutes;
```

Рисунок 3.7 – Файл AppRoutes.jsx

3.9.2 Файл AuthProvider.jsx

Цей компонент забезпечує контекст аутентифікації для всього додатка. Він відповідає за отримання та оновлення токенів доступу, а також за зберігання та очищення даних користувача.

```
function AuthProvider({children}) {
  const localAccessToken = getAccessToken() || null;
  const refreshToken = getRefreshToken() || null;
  const navigate : NavigateFunction  = useNavigate();
  const location : Location  = useLocation();
  const [isFirstMounted : boolean , setIsFirstMounted] = useState( initialState: tr

  const handleLogin = (userData) : void  => {
    setUserData(userData);
    const origin = (location.state)?.from?.pathname || '/home';
    navigate(origin);
  };

  const handleLogout = async () : Promise<void>  => {
    clearUserData();
    await fetchWrapper(endpoints.logout,  opts: {
      method: 'DELETE',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${localAccessToken}`,
      },
      body: {refreshToken},
    });
    navigate('/login');
  };

  async function updateRefreshtoken() : Promise<void>  {
    const response : Response  = await fetchWrapper(endpoints.token,  opts: {
      method: 'POST',
      headers: {'Content-Type': 'application/json'},
      body: {refreshToken},
    });

    if (response.ok) {
      const {accessToken} = await response.json();
      updateAccessToken(accessToken);
    } else {
      clearUserData();
      navigate('/login');
      window.location.reload();
    }
```

Рисунок 3.8 – Файл AuthProvider.jsx 1 частина

```
    if (isFirstMounted) {
      setIsFirstMounted( value: false);
    }
  }

  useEffect( effect: () : function(): void | any  => {
    if (refreshToken) {
      if (isFirstMounted) {
        updateRefreshtoken();
      }

      const intervalId : number  = setInterval( handler: () : void  => {
        updateRefreshtoken();
      }, ACCESS_TOKEN_EXPIRES_TIME);
      return () : void  => clearInterval(intervalId);
    }
    return undefined;
  }, deps: [localAccessToken]);

  const value : {...}  = useMemo( factory: () : {...}  => ({
    token: localAccessToken,
    onLogin: handleLogin,
    onLogout: handleLogout,
  }), deps: [localAccessToken]);

  return (
    <AuthContext.Provider value={value}>
      {children}
    </AuthContext.Provider>
  );
}

export default AuthProvider;
```

Рисунок 3.9 – Файл AuthProvider.jsx 2 частина

3.9.3 Файл Navbar.jsx

Цей компонент відповідає за верхню панель навігації, яка включає кнопки для переходу на домашню сторінку, до списку проєктів, а також кнопки для входу, реєстрації або виходу, залежно від стану аутентифікації користувача.

```
return (
  <Box sx={{flexGrow: 1}}>
    <AppBar position="static">
      <Toolbar>
        <IconButton
          size="large"
          edge="start"
          color="inherit"
          aria-label="menu"
          sx={{mr: 2}}
        >
          <WorkIcon/>
        </IconButton>
        <Grid container justifyContent="space-between" alignItems="center">
          <Grid>
            <Button color="inherit" href="/home">Home</Button>
            <Button color="inherit" href="/projects">Projects</Button>
          </Grid>
          {!token ?
            <Grid item>
              <Button color="inherit" href="/login">Login</Button>
              <Button color="inherit" href="/register">Register</Button>
            </Grid>
            :
            ''
          }
          {token ?
            <Button color="inherit" onClick={handleLogout}>Logout</Button>
            :
            ''
          }
        </Grid>
      </Toolbar>
    </AppBar>
  </Box>
);
}
```

Рисунок 3.10 – файл Navbar.jsx

3.9.4 Файл NewProjectModal.jsx

Цей компонент відповідає за модальне вікно створення нового проєкту. Він включає форми для введення назви та опису проєкту, а також кнопку для підтвердження створення проєкту.

```jsx
const NewProjectModal = ({ open, onClose, onCreate }) => {
  const handleCreate = () : void  => {
    onCreate({ jobTitle: projectTitle, description });
    onClose();
    setProjectTitle( value: '');
    setDescription( value: '');
  };

  return (
    <Modal
      className={classes.modal}
      open={open}
      onClose={onClose}
      closeAfterTransition
    >
      <Fade in={open}>
        <div className={classes.paper}>
          <Typography variant="h5" gutterBottom>Create New Project</Typography>
          <TextField
            className={classes.textField}
            label="Project Title"
            variant="outlined"
            fullWidth
            value={projectTitle}
            onChange={(e : ChangeEvent<...> ) : void  => setProjectTitle(e.target.value)}
          />
          <TextField
            className={classes.textField}
            label="Description"
            variant="outlined"
            fullWidth
            multiline
            rows={4}
            value={description}
            onChange={(e : ChangeEvent<...> ) : void  => setDescription(e.target.value)}
          />
          <Button variant="contained" color="primary" onClick={handleCreate}>Create Project</Button>
        </div>
      </Fade>
    </Modal>
  );
};
```

Рисунок 3.11 – Файл NewProjectModal.jsx

3.9.5 Файл ProtectedRoute.jsx

Цей компонент використовується для захисту маршрутів, що потребують аутентифікації. Якщо користувач не автентифікований, він буде перенаправлений на сторінку входу.

```jsx
import {Navigate, Outlet, useLocation} from "react-router-dom";
import {useAuth} from "../utils/auth";

function ProtectedRoute({
                          redirectPath : string  = '/login',
                          children,
                        }) {
  const { token : string  } = useAuth();
  const location : Location  = useLocation();

  if (!token) {
    return <Navigate to={redirectPath} replace state={{ from: location }} />;
  }

  return children || <Outlet />;
}

export default ProtectedRoute;
```

Рисунок 3.12 – файл ProtectedRoute.jsx

3.10 Аналіз результатів

Запускаємо проєкт, переходимо на головну сторінку.

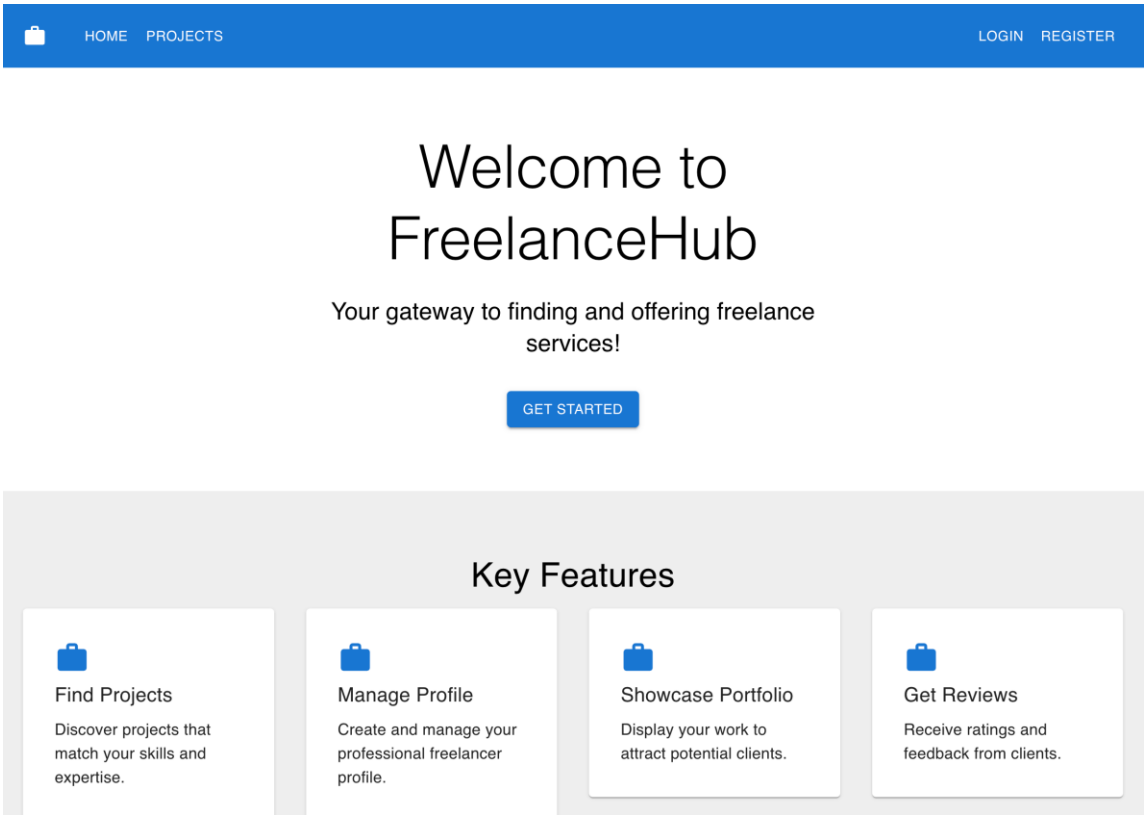

Рисунок 3.13 – Головна сторінка

Тиснемо REGISTER для реєстрації

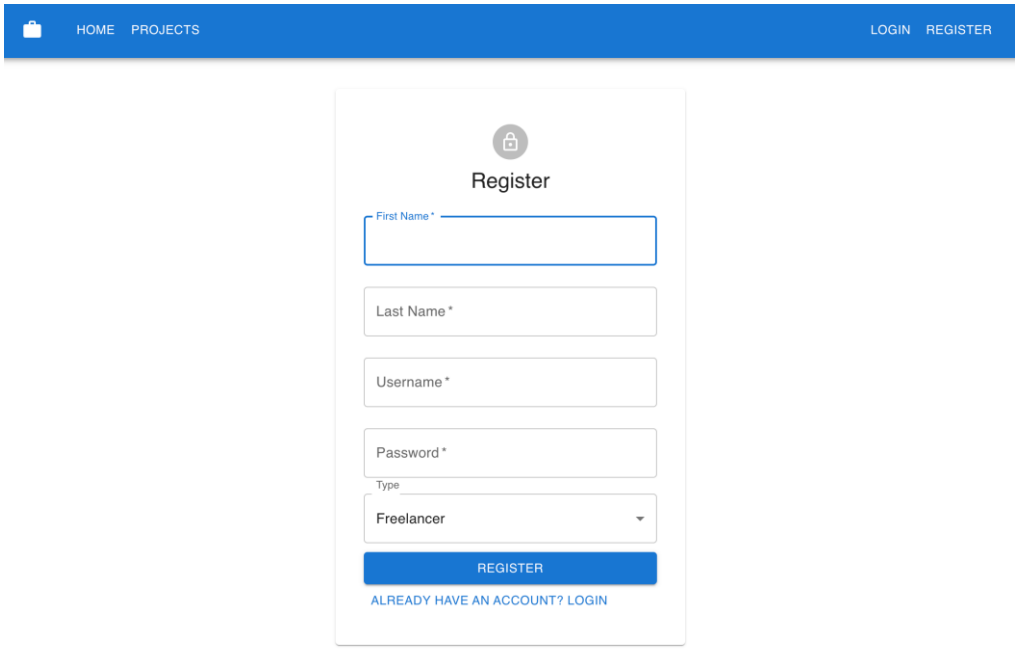

Рисунок 3.14 – Сторінка реєстрації

Вводимо вказані дані, та реєструємося, одразу автоматично перенаправляємося на сторінку логіну.

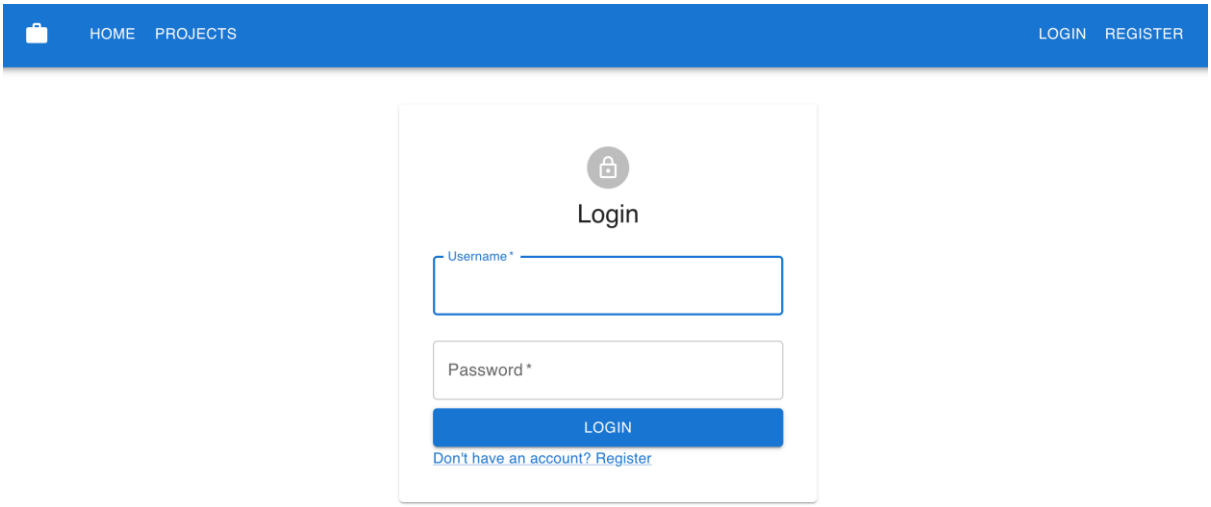

Рисунок 3.15 – Сторінка авторизації

Вводимо дані які вказали, та опиняємося знову на головній сторінці. На панелі вгорі бачимо кнопку PROJECTS, переходимо туди.

Available Projects

Рисунок 3.16 – Сторінка доступних проєктів

Бачимо що доступних проєктів немає, це тому що ми зареєструвалися як freelancer, але при цьому жоден клієн не створив проєкти на які б ми могли подати заявку. При цьому якщо зайти на цю сторінку з аккаунту клієнта бачимо наступну ситуацію.

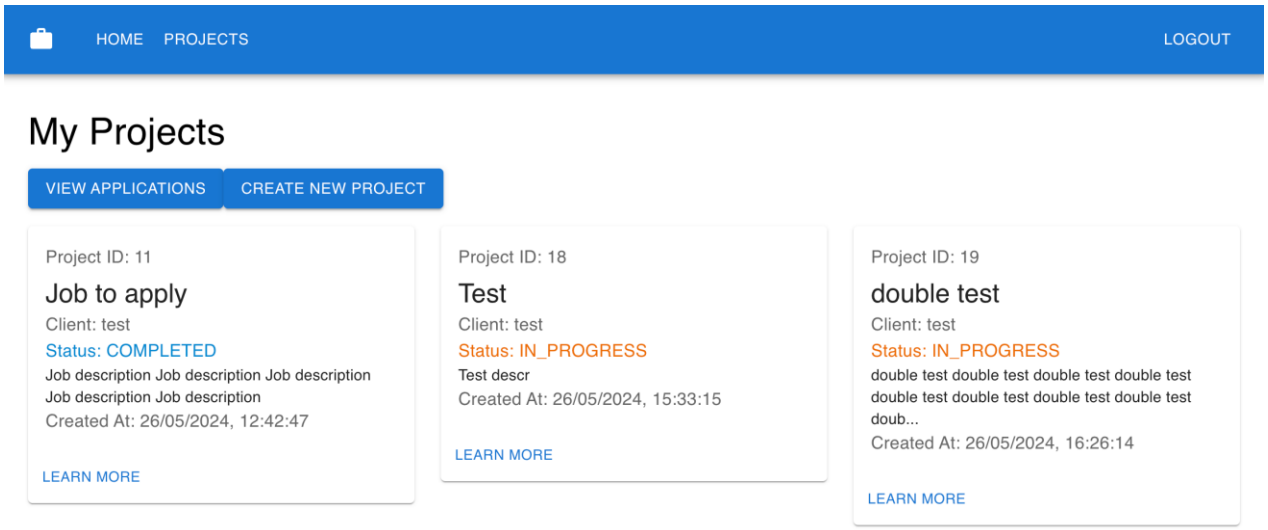

Рисунок 3.17 – Сторінка проєктів зі сторони замовника

Кнопки для створення нового проєкту, для перегляду заявок від фрілансерів на свої проєкти та самі проєкти.

Проєкти мають різні статуси, один статус completed, що означає що він був виконаний, інші статуси IN PROGRESS, що означає що клієнт передивився список заявок на проєкт, вибрав одного фрілансера і доручив йому завдання. Натиснувши LEARN MORE бачимо повний опис завдання. Натиснувши CREATE NEW PROJECT як клієнт маємо змогу створити новий проєкт.

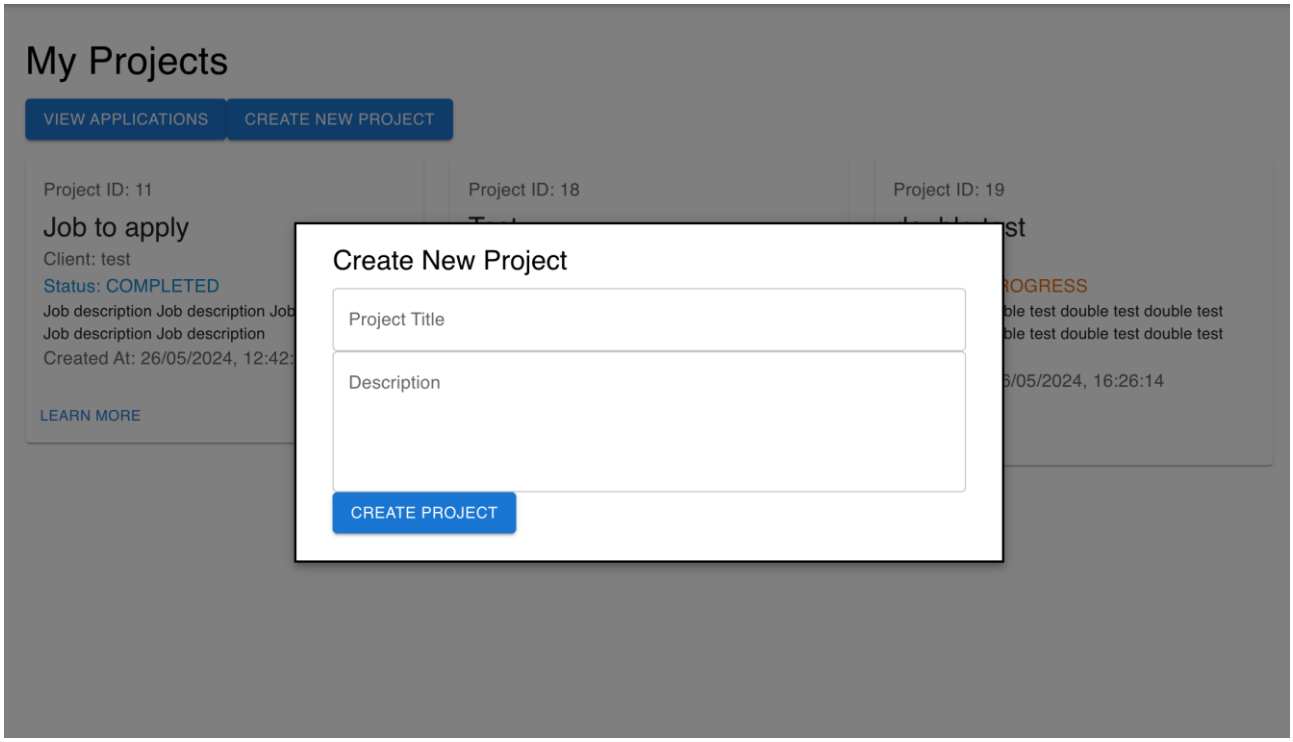

Рисунок 3.18 – Вікно створення нового проєкту

Після створення нового проєкту він з’явиться у фрілансерів у пошуку, зайшовши на сторінку проєкту фрілансер зможе подати заявку.

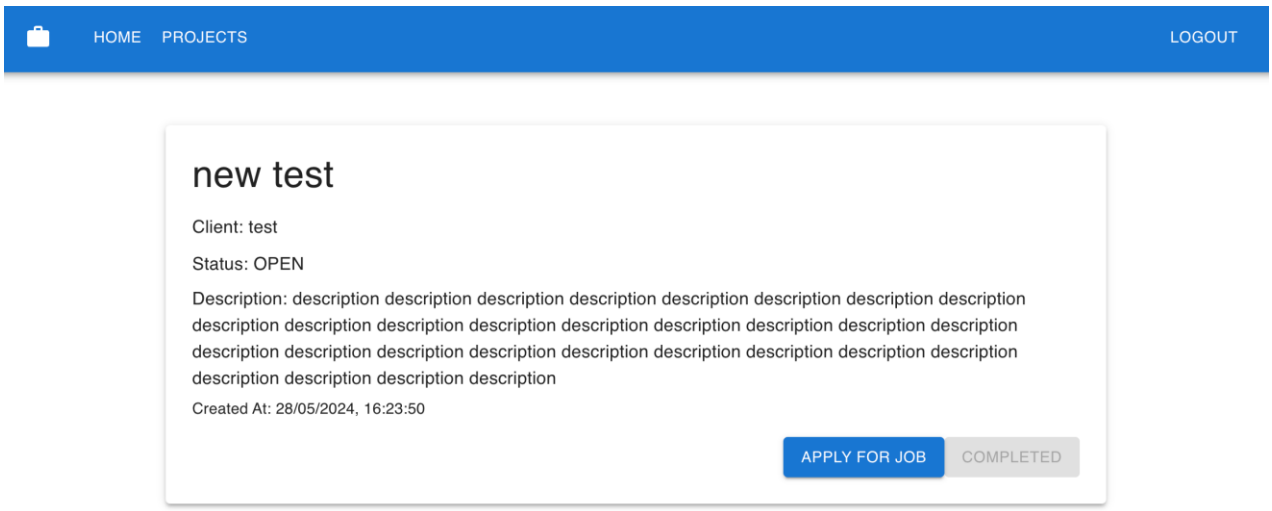

Рисунок 3.19 – Вікно проєкту для подачі заявки

Після подання заявки клієнт зможе перевірити заявку, подивитись на профіль людини, глянути над якими проєктами він вже працював до цього.

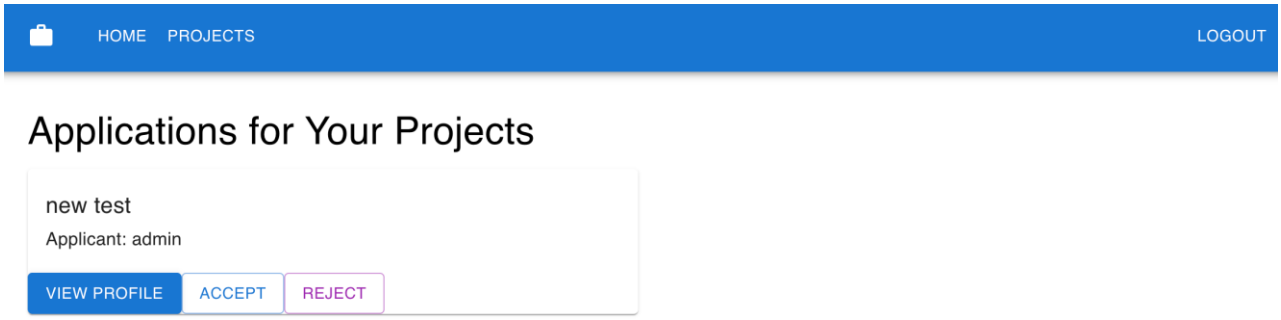

Рисунок 3.20 – Вікно заявок на проєкт зі сторони клієнта

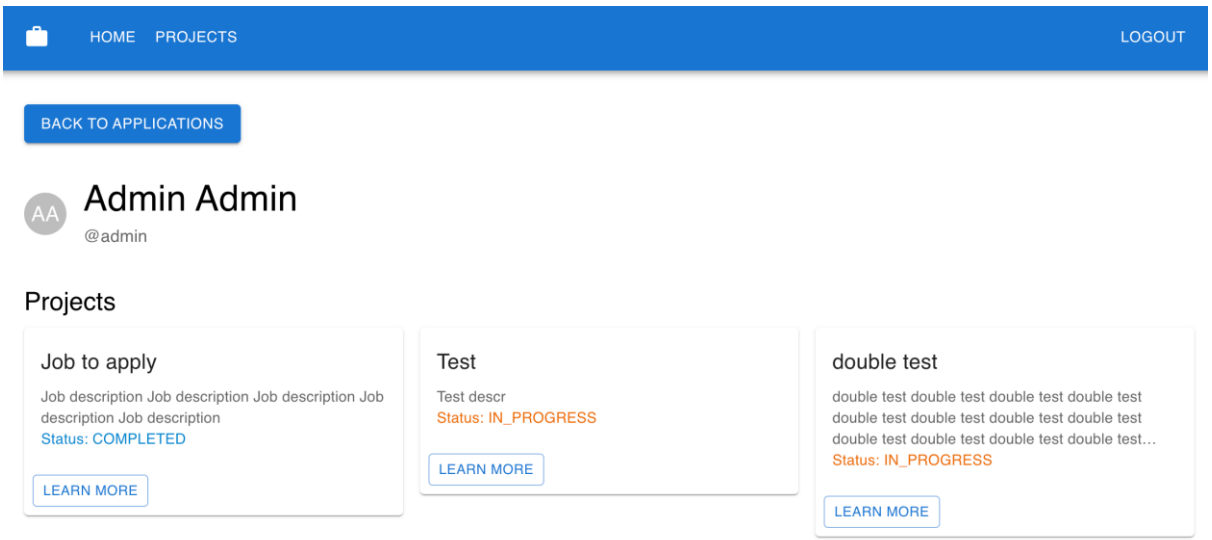

Рисунок 3.21 – сторінка профілю фрілансера для клієнтів.

Після прийняття заявки клієнтом фрілансер починає роботу, по закінченню фрілансер натискає Submit для підтвердження роботи клієнта.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розглянуто та досліджено розробку вебдодатку для спільної роботи між клієнтами та фрілансерами. Додаток був розроблений з використанням сучасних технологій, включаючи React.js для фронтенду, Spring boot для бекенду та базу даних MySQL.

У процесі розробки додатку були враховані сучасні підходи до проєктування, включаючи застосування патернів проєктування та чистої архітектури. Також були використані стандарти безпеки для захисту від атак та забезпечення конфіденційності даних користувачів.

У результаті роботи був розроблений функціональний вебдодаток, який може бути використаний для управління проєктами та спільної роботи між клієнтами та фрілансерами. Його можна розгорнути на сервері та використовувати у комерційних цілях або для особистого використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Повний гід по бекенд розробці: теорія та практика. URL: https://galaktica.io/blog/backend/#:~:text=Бекенд%20розробка%20–%20це%20інженерія%20серверної,з%20іншими%20програмами%20та%20сервісами (дата звернення 03.05.2024).

2. Що таке фріланс і як на ньому заробляти: види діяльності та як відкрити ФОП. URL: https://fakty.com.ua/ua/ukraine/ekonomika/20230314-shho-take-frilans-i-yak-na-nomu-zaroblyaty-vydy-diyalnosti-ta-yak-vidkryty-fop/ (дата звернення 03.05.2024).

3. UPWORK ДЛЯ УКРАЇНЦІВ: ЯК ПОЧАТИ ПРАЦЮВАТИ БЕЗ ДОСВІДУ ЧИ ЗНАННЯ АНГЛІЙСЬКОЇ. URL: https://it-kharkiv.com/upwork-dlya-ukrayintsiv/ (дата звернення 03.05.2024).

4. Freelancer.com Огляд 2024 – Відмінна репутація, АЛЕ… URL: https://www.websiteplanet.com/uk/freelance-websites/freelancer/#overview (дата звернення 03.05.2024).

5. Як фрилансити на Fiverr: від старту з нуля до черги з клієнтів. URL: https://happymonday.ua/yak-frylansyty-na-fiverr (дата звернення 03.05.2024).

6. Що таке Java і де вона використовується. URL: https://goit.global/ua/articles/shcho-take-java-i-de-vona-vykorystovuietsia/ (дата звернення 04.05.2024).

7. Фреймворк Spring та його особливості. URL: https://highload.today/uk/frejmvork-spring-ta-jogo-osoblivosti/ (дата звернення 04.05.2024).

8. Що таке JavaScript. URL: https://cases.media/en/article/sho-take-javascript (дата звернення 04.05.2024).

9. Що таке React JS? Як почати вивчати Реакт? Навички для react developer. URL: https://cases.media/en/article/sho-take-react-js-yak-pochati-vivchati-reakt-navichki-dlya-react-developer (дата звернення 04.05.2024).

10. SQL чи NoSQL – ось в чому питання. URL: https://alternativescience.net/programming/242-sql-chy-nosql-os-v-chomu-pytannya/ (дата звернення 04.05.2024).

11. Що таке GitHub і навіщо він потрібен розробнику. URL: https://foxminded.ua/shho-take-github-i-navishho-vin-potriben-rozrobniku/ (дата звернення 04.05.2024).

ДОДАТКИ

Додаток А

```
package sumdu.edu.ua.freelanceforge.config;

import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.configuration.Authenticatio
nConfiguration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@RequiredArgsConstructor
@EnableWebSecurity
@Configuration
public class SecurityConfig {
    private final SecurityFilter securityFilter;

    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration
authenticationConfiguration) throws Exception {
        return authenticationConfiguration.getAuthenticationManager();
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.authorizeHttpRequests(authz -> authz
                .requestMatchers("/auth/**").permitAll()
                .requestMatchers("/admin/**").hasRole("ADMIN")
                .anyRequest().authenticated()
        );
        return http
                .sessionManagement(session ->
session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .csrf(AbstractHttpConfigurer::disable)
                .addFilterBefore(securityFilter,
UsernamePasswordAuthenticationFilter.class)
                .build();
    }
}
```

Додаток Б

```
package sumdu.edu.ua.freelanceforge.config;

import io.jsonwebtoken.ExpiredJwtException;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Component
@RequiredArgsConstructor
public class SecurityFilter extends OncePerRequestFilter {
    private final SecurityUtils securityUtils;
    private final SecurityUserService userService;

    @Override
    protected void doFilterInternal(HttpServletRequest httpServletRequest,
HttpServletResponse httpServletResponse,
                                    FilterChain filterChain) throws ServletException,
IOException {
        String authorization = httpServletRequest.getHeader("Authorization");
        String token = null;
        String userName = null;

        try {
            if (null != authorization && authorization.startsWith("Bearer ")) {
                token = authorization.substring(7);
                userName = securityUtils.getUsernameFromToken(token);
            }

            if (null != userName &&
SecurityContextHolder.getContext().getAuthentication() == null) {
                UserDetails userDetails
                        = userService.loadUserByUsername(userName);

                if (Boolean.TRUE.equals(securityUtils.validateToken(token, userDetails)))
{
                    UsernamePasswordAuthenticationToken
usernamePasswordAuthenticationToken
                            = new UsernamePasswordAuthenticationToken(userDetails,
                            null, userDetails.getAuthorities());

                    usernamePasswordAuthenticationToken.setDetails(
                            new
WebAuthenticationDetailsSource().buildDetails(httpServletRequest)
                    );


SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken)
;
                }

            }
            filterChain.doFilter(httpServletRequest, httpServletResponse);
        } catch (ExpiredJwtException e) {
            httpServletResponse.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
            httpServletResponse.setContentType("application/json");
            httpServletResponse.setCharacterEncoding("UTF-8");
            httpServletResponse.getWriter().write("{\"message\":\"" + e.getMessage() +
"\"}");
        }
```

```
    }
}
```

Додаток В.

```
package sumdu.edu.ua.freelanceforge.config;

import lombok.RequiredArgsConstructor;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import sumdu.edu.ua.freelanceforge.dao.Role;
import sumdu.edu.ua.freelanceforge.service.UserService;

import java.util.Collection;

@Service
@RequiredArgsConstructor
public class SecurityUserService implements UserDetailsService {
    private final UserService userService;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        var user = userService.findByUsername(username);

        if (user == null) {
            throw new UsernameNotFoundException("IN loadUserByUsername user not found by
username: " + username);
        }

        Collection<SimpleGrantedAuthority> authorities = user.getRoles()
                .stream().map(Role::getName)
                .map(SimpleGrantedAuthority::new)
                .toList();

        return new User(user.getUsername(), user.getPassword(), authorities);
    }
}
```

Додаток Г.

```
package sumdu.edu.ua.freelanceforge.config;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import java.io.Serial;
import java.io.Serializable;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

@Component
public class SecurityUtils implements Serializable {
    @Serial
    private static final long serialVersionUID = 234234523523L;
    @Value("${jwt.expiration}")
    private long jwtTokenValidity;
    @Value("${jwt.secret}")
    private String secretKey;

    public String getUsernameFromToken(String token) {
        return getClaimFromToken(token, Claims::getSubject);
    }

    public Date getExpirationDateFromToken(String token) {
        return getClaimFromToken(token, Claims::getExpiration);
    }

    public <T> T getClaimFromToken(String token, Function<Claims, T> claimsResolver) {
        final Claims claims = getAllClaimsFromToken(token);
        return claimsResolver.apply(claims);
    }

    private Claims getAllClaimsFromToken(String token) {
        return Jwts.parser().setSigningKey(secretKey).parseClaimsJws(token).getBody();
    }

    private Boolean isTokenExpired(String token) {
        final Date expiration = getExpirationDateFromToken(token);
        return expiration.before(new Date());
    }

    public String generateToken(UserDetails userDetails) {
        Map<String, Object> claims = new HashMap<>();
        return doGenerateToken(claims, userDetails.getUsername());
    }

    private String doGenerateToken(Map<String, Object> claims, String subject) {
        return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new
Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + jwtTokenValidity))
                .signWith(SignatureAlgorithm.HS512, secretKey).compact();
    }

    public Boolean validateToken(String token, UserDetails userDetails) {
        final String username = getUsernameFromToken(token);
        return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
    }

}
```

Додаток Д.

```
package sumdu.edu.ua.freelanceforge.dao.dto;

import lombok.Value;
import sumdu.edu.ua.freelanceforge.dao.JobApplicationStatus;

import java.time.LocalDateTime;

@Value
public class JobApplicationDto {
    Long id;
    JobDto job;
    UserDto freelancer;
    JobApplicationStatus status;
    LocalDateTime createdAt;
}
```

Додаток Е.

```
package sumdu.edu.ua.freelanceforge.dao.dto;

import lombok.Value;
import sumdu.edu.ua.freelanceforge.dao.JobStatus;

import java.time.LocalDateTime;

@Value
public class JobDto {
    Long id;
    UserDto client;
    String title;
    String description;
    JobStatus status;
    LocalDateTime createdAt;
}
```

Додаток Є.

```
package sumdu.edu.ua.freelanceforge.dao.dto;

import lombok.Value;

@Value
public class LoginDataDto {
    String username;
    String password;
}
```

Додаток Ж.

```
package sumdu.edu.ua.freelanceforge.dao.dto;

import lombok.Value;
import sumdu.edu.ua.freelanceforge.dao.UserType;

import java.util.List;

@Value
public class LoginResponseDataDto {
    String accessToken;
    String refreshToken;
    UserType type;
```

```
    List<RoleDto> roles;
}
```

Додаток З.

```
package sumdu.edu.ua.freelanceforge.dao.dto;

import lombok.Value;

public record RefreshDataDto(String refreshToken) {
}
```

Додаток И.

```
package sumdu.edu.ua.freelanceforge.dao.dto;

import lombok.Value;

@Value
public class RefreshResponseDataDto {
    String accessToken;
    String refreshToken;
}
```

Додаток І.

```
package sumdu.edu.ua.freelanceforge.dao.dto;

import lombok.Value;
import sumdu.edu.ua.freelanceforge.dao.UserType;

@Value
public class RegisterDataDto {
    String username;
    String password;
    String firstName;
    String lastName;
    UserType userType;
}
```

Додаток Ї.

```
package sumdu.edu.ua.freelanceforge.dao.dto;

import lombok.Value;

@Value
public class RoleDto {
    String name;
}
```

Додаток Й.

```
package sumdu.edu.ua.freelanceforge.dao.dto;

import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.Value;

import java.util.List;

@Value
```

```
@JsonInclude(JsonInclude.Include.NON_NULL)
public class UserDto {
    Long id;
    String username;
    String firstName;
    String lastName;
    String userStatus;
    List<RoleDto> roles;
}
```

Додаток К.

```
package sumdu.edu.ua.freelanceforge.dao.dto;

import lombok.Value;
import sumdu.edu.ua.freelanceforge.dao.UserType;

@Value
public class UserTypeDto {
    UserType userType;
}
```

Додаток Л.

```
package sumdu.edu.ua.freelanceforge.dao.mapper;

import org.mapstruct.Mapper;
import sumdu.edu.ua.freelanceforge.dao.Job;
import sumdu.edu.ua.freelanceforge.dao.JobApplication;
import sumdu.edu.ua.freelanceforge.dao.Role;
import sumdu.edu.ua.freelanceforge.dao.User;
import sumdu.edu.ua.freelanceforge.dao.dto.*;

@Mapper(componentModel = "spring")
public interface MyMapper {
    User registerDataDtoToUser(RegisterDataDto registerDataDto);

    UserDto userToUserDto(User user);

    RoleDto roleToRoleDto(Role role);

    JobDto jobToJobDto(Job job);

    Job jobDtoToJob(JobDto jobDto);

    JobApplicationDto jobApplicationToJobApplicationDto(JobApplication jobApplication);
}
```

Додаток М.

```
package sumdu.edu.ua.freelanceforge.dao;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.time.LocalDateTime;

@Entity
@Table(name = "jobs")
@AllArgsConstructor
@NoArgsConstructor
```

```
@Data
@Builder
public class Job {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @ManyToOne
    @JoinColumn(name = "client_id")
    private User client;

    @Column(name = "title")
    private String title;

    @Column(name = "description")
    private String description;

    @Column(name = "status")
    @Enumerated(EnumType.STRING)
    private JobStatus status;

    @Column(name = "created_at")
    private LocalDateTime createdAt;

    @Column(name = "updated_at")
    private LocalDateTime updatedAt;
}
```

Додаток Н.

```
package sumdu.edu.ua.freelanceforge.dao;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.time.LocalDateTime;

@Entity
@Table(name = "job_applications")
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class JobApplication {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @ManyToOne
    @JoinColumn(name = "job_id", nullable = false)
    private Job job;

    @ManyToOne
    @JoinColumn(name = "freelancer_id", nullable = false)
    private User freelancer;

    @Column(name = "status")
    @Enumerated(EnumType.STRING)
    private JobApplicationStatus status;

    @Column(name = "created_at")
    private LocalDateTime createdAt;

    @Column(name = "updated_at")
```

```
    private LocalDateTime updatedAt;
}
```

Додаток О.

```
package sumdu.edu.ua.freelanceforge.dao;

public enum JobApplicationStatus {
    PENDING, ACCEPTED, REJECTED
}
```

Додаток П.

```
package sumdu.edu.ua.freelanceforge.dao;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.time.LocalDateTime;

@Entity
@Table(name = "job_history")
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class JobHistory {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @ManyToOne
    @JoinColumn(name = "job_id", nullable = false)
    private Job job;

    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @Column(name = "status")
    private String status;

    @Column(name = "created_at")
    private LocalDateTime createdAt;
}
```

Додаток Р

```
package sumdu.edu.ua.freelanceforge.dao;

public enum JobStatus {
    OPEN, IN_PROGRESS, COMPLETED, CANCELED
}
```

Додаток С.

```
package sumdu.edu.ua.freelanceforge.dao;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

import java.time.Instant;

@Entity
@Table(name = "refresh_tokens")
@Data
@AllArgsConstructor
@ToString(exclude = "user")
@NoArgsConstructor
public class RefreshToken {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id")
    private User user;

    @Column(name = "token")
    private String token;

    @Column(name = "expiration")
    private Instant expiryDate;
}
```

Додаток Т.

```java
package sumdu.edu.ua.freelanceforge.dao;

import jakarta.persistence.*;
import lombok.Data;
import lombok.RequiredArgsConstructor;
import lombok.ToString;

import java.util.List;

@RequiredArgsConstructor
@Entity
@Table(name = "roles")
@Data
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "name")
    private String name;

    @ManyToMany(mappedBy = "roles", fetch = FetchType.LAZY)
    @ToString.Exclude
    private List<User> users;
}
```

Додаток У.

```
package sumdu.edu.ua.freelanceforge.dao;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.time.LocalDateTime;
import java.util.List;

@Entity
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "users")
@Data
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "username")
    private String username;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "password")
    private String password;

    @Column(name = "type")
    @Enumerated(EnumType.STRING)
    private UserType userType;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(
            name = "user_roles",
            joinColumns = {@JoinColumn(name = "user_id", referencedColumnName = "id")},
            inverseJoinColumns = {@JoinColumn(name = "role_id", referencedColumnName =
"id")}
    )
    private List<Role> roles;

    @Column(name = "created_at")
    private LocalDateTime created;

    @Column(name = "updated_at")
    private LocalDateTime updated;
}
```

Додаток Ф.

```
package sumdu.edu.ua.freelanceforge.dao;

public enum UserType {
    CUSTOMER,
    FREELANCER
}
```

Додаток Х.

```
package sumdu.edu.ua.freelanceforge.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import sumdu.edu.ua.freelanceforge.dao.JobApplication;

public interface JobApplicationRepository extends JpaRepository<JobApplication, Long> {
}
```

Додаток Ц.

```
package sumdu.edu.ua.freelanceforge.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import sumdu.edu.ua.freelanceforge.dao.Job;

public interface JobRepository extends JpaRepository<Job, Long> {
}
```

Додаток Ч.

```
package sumdu.edu.ua.freelanceforge.repository;

import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.repository.CrudRepository;
import sumdu.edu.ua.freelanceforge.dao.RefreshToken;
import sumdu.edu.ua.freelanceforge.dao.User;

import java.util.Optional;

public interface RefreshTokenRepository extends CrudRepository<RefreshToken, Long> {
    Optional<RefreshToken> findByToken(String token);
    @Modifying
    int deleteByUser(User user);
}
```

Додаток Ш.

```
package sumdu.edu.ua.freelanceforge.repository;

import org.springframework.data.repository.CrudRepository;
import sumdu.edu.ua.freelanceforge.dao.Role;

public interface RoleRepository extends CrudRepository<Role, Long> {
    Role findByName(String name);
}
```

Додаток Щ.

```
package sumdu.edu.ua.freelanceforge.repository;

import org.springframework.data.repository.CrudRepository;
import sumdu.edu.ua.freelanceforge.dao.User;

import java.util.Optional;

public interface UserRepository extends CrudRepository<User, Long> {
    Optional<User> findByUsername(String username);
}
```

Додаток Ь.

```
package sumdu.edu.ua.freelanceforge.rest.exception;

public class NoAuthenticationException extends RuntimeException {
    public NoAuthenticationException(String message) {
        super(message);
    }

    public NoAuthenticationException() {
        super();
    }
}
```

Додаток Ю.

```
package sumdu.edu.ua.freelanceforge.rest.exception;

public class RefreshTokenExpiredException extends RuntimeException {
    public RefreshTokenExpiredException() {
        super();
    }

    public RefreshTokenExpiredException(String message) {
        super(message);
    }
}
```

Додаток Я.

```
package sumdu.edu.ua.freelanceforge.service;

import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import sumdu.edu.ua.freelanceforge.dao.RefreshToken;
import sumdu.edu.ua.freelanceforge.repository.RefreshTokenRepository;
import sumdu.edu.ua.freelanceforge.repository.UserRepository;
import sumdu.edu.ua.freelanceforge.rest.exception.RefreshTokenExpiredException;

import java.time.Instant;
import java.util.Optional;
import java.util.UUID;

@Service
@RequiredArgsConstructor
public class RefreshService {
    private final UserRepository userRepository;
    private final RefreshTokenRepository refreshTokenRepository;
```

```
    @Value("${jwt.refreshExpiration}")
    private long refreshExpiration;


    public RefreshToken createRefreshToken(Long userId) {
        RefreshToken refreshToken = new RefreshToken();

        refreshToken.setUser(userRepository.findById(userId).orElseThrow(() -> new
RuntimeException("user not found")));
        refreshToken.setExpiryDate(Instant.now().plusMillis(refreshExpiration));
        refreshToken.setToken(UUID.randomUUID().toString());

        refreshToken = refreshTokenRepository.save(refreshToken);
        return refreshToken;
    }

    public Optional<RefreshToken> findByToken(String token) {
        return refreshTokenRepository.findByToken(token);
    }

    public RefreshToken verifyExpiration(RefreshToken token) {
        if (token.getExpiryDate().compareTo(Instant.now()) < 0) {
            refreshTokenRepository.delete(token);
            throw new RefreshTokenExpiredException("refresh token expired");
        }
        return token;
    }

    @Transactional
    public void deleteByUserId(Long userId) {

refreshTokenRepository.deleteByUser(userRepository.findById(userId).orElseThrow(() -> new
RuntimeException("user not found")));
    }
}
```

Додаток А1.

```
package sumdu.edu.ua.freelanceforge.rest;

import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.*;
import sumdu.edu.ua.freelanceforge.dao.*;
import sumdu.edu.ua.freelanceforge.dao.dto.JobApplicationDto;
import sumdu.edu.ua.freelanceforge.dao.dto.JobDto;
import sumdu.edu.ua.freelanceforge.dao.mapper.MyMapper;
import sumdu.edu.ua.freelanceforge.service.JobApplicationService;
import sumdu.edu.ua.freelanceforge.service.JobService;
import sumdu.edu.ua.freelanceforge.service.UserService;

import java.util.List;

@RestController
@RequiredArgsConstructor
@RequestMapping(value = "/jobs")
public class JobsController {
    private final JobService jobService;
    private final MyMapper mapper;
    private final UserService userService;
    private final JobApplicationService jobApplicationService;

    @GetMapping
    public ResponseEntity<List<JobDto>> getAllJobs(Authentication authentication) {
        String username = authentication.getName();
        User user = userService.findByUsername(username);
```

```
        List<JobDto> jobDtos;
        if (user.getUserType() == UserType.FREELANCER) {
            List<Job> jobs = jobService.getAllJobs();
            List<JobDto> allAvailableJobs = new java.util.ArrayList<>(jobs.stream()
                    .filter(job -> job.getStatus() == JobStatus.OPEN)
                    .map(mapper::jobToJobDto)
                    .toList());
            List<JobDto> allMyJobs =
jobApplicationService.getUserProjects(user.getId()).stream()
                    .filter(job -> job.getStatus() == JobStatus.OPEN).toList();
            allAvailableJobs.addAll(allMyJobs);
            jobDtos = allAvailableJobs;
        } else if (user.getUserType() == UserType.CUSTOMER) {
            jobDtos = jobService.getAllJobs().stream()
                    .filter(job -> job.getClient().getId().equals(user.getId()))
                    .map(mapper::jobToJobDto)
                    .toList();
        } else {
            return new ResponseEntity<>(HttpStatus.UNAUTHORIZED);
        }
        return new ResponseEntity<>(jobDtos, HttpStatus.OK);
    }


    @GetMapping("/{id}")
    public ResponseEntity<JobDto> getJobById(@PathVariable Long id) {
        Job job = jobService.getJobById(id);
        JobDto jobDto = mapper.jobToJobDto(job);
        return new ResponseEntity<>(jobDto, HttpStatus.OK);
    }

    @GetMapping("/apply/{id}")
    public ResponseEntity<HttpStatus> applyForJob(@PathVariable Long id, Authentication
authentication) {
        String username = authentication.getName();
        jobApplicationService.applyForJob(username, id);
        return new ResponseEntity<>(HttpStatus.OK);
    }

    @GetMapping("/applied/{id}")
    public ResponseEntity<HttpStatus> hasAlreadyApplied(@PathVariable Long id,
Authentication authentication) {
        String username = authentication.getName();
        User user = userService.findByUsername(username);

        boolean alreadyApplied = jobApplicationService.getAllJobApplications().stream()
                .anyMatch(jobApplication ->
jobApplication.getFreelancer().getId().equals(user.getId())
                        && jobApplication.getJob().getId().equals(id));

        if (alreadyApplied) {
            return new ResponseEntity<>(HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

    @GetMapping("/applications")
    public ResponseEntity<List<JobApplicationDto>> getApplications(Authentication
authentication) {
        String username = authentication.getName();
        User user = userService.findByUsername(username);
        List<JobApplication> allJobApplications =
jobApplicationService.getAllJobApplications().stream()
                .filter(jobApplication ->
jobApplication.getJob().getClient().getId().equals(user.getId()))
                .filter(jobApplication -> jobApplication.getStatus() ==
JobApplicationStatus.PENDING)
                .toList();
        List<JobApplicationDto> jobDtos =
allJobApplications.stream().map(mapper::jobApplicationToJobApplicationDto).toList();
```

```
        return new ResponseEntity<>(jobDtos, HttpStatus.OK);
    }

    @GetMapping("/applications/accept/{id}")
    public ResponseEntity<HttpStatus> acceptApplication(@PathVariable Long id) {
        jobApplicationService.acceptApplication(id);
        return new ResponseEntity<>(HttpStatus.OK);
    }

    @GetMapping("/applications/isAccepted/{id}")
    public ResponseEntity<HttpStatus> isAccepted(@PathVariable Long id) {
        jobApplicationService.isJobAccepted(id);
        return new ResponseEntity<>(HttpStatus.OK);
    }

    @GetMapping("/applications/reject/{id}")
    public ResponseEntity<HttpStatus> rejectApplication(@PathVariable Long id) {
        jobApplicationService.rejectApplication(id);
        return new ResponseEntity<>(HttpStatus.OK);
    }

    @GetMapping("/complete/{id}")
    public ResponseEntity<HttpStatus> completeJob(@PathVariable Long id) {
        jobService.setJobStatus(id, JobStatus.COMPLETED);
        return new ResponseEntity<>(HttpStatus.OK);
    }

    @PostMapping
    public ResponseEntity<JobDto> createJob(@RequestBody JobDto jobDto, Authentication
authentication) {
        String username = authentication.getName();
        User user = userService.findByUsername(username);
        Job job = mapper.jobDtoToJob(jobDto);
        Job savedJob = jobService.createJob(user, job);
        JobDto savedJobDto = mapper.jobToJobDto(savedJob);
        return new ResponseEntity<>(savedJobDto, HttpStatus.CREATED);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<HttpStatus> deleteJob(@PathVariable Long id) {
        jobService.deleteJob(id);
        return new ResponseEntity<>(HttpStatus.OK);
    }
}
```

Додаток Б1.

```
package sumdu.edu.ua.freelanceforge.rest;

import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import sumdu.edu.ua.freelanceforge.dao.Job;
import sumdu.edu.ua.freelanceforge.dao.UserType;
import sumdu.edu.ua.freelanceforge.dao.dto.JobDto;
import sumdu.edu.ua.freelanceforge.dao.dto.UserDto;
import sumdu.edu.ua.freelanceforge.dao.dto.UserTypeDto;
import sumdu.edu.ua.freelanceforge.dao.mapper.MyMapper;
import sumdu.edu.ua.freelanceforge.service.JobApplicationService;
import sumdu.edu.ua.freelanceforge.service.JobService;
import sumdu.edu.ua.freelanceforge.service.UserService;

import java.util.List;
```

```
@RestController
@RequestMapping(value = "/user")
@RequiredArgsConstructor
public class UserController {
    private final UserService userService;
    private final JobService jobService;
    private final MyMapper mapper;
    private final JobApplicationService jobApplicationService;

    @GetMapping("/type")
    public ResponseEntity<UserTypeDto> getCurrentUserType(Authentication authentication)
{
        String username = authentication.getName();
        UserType userType = userService.findByUsername(username).getUserType();
        return new ResponseEntity<>(new UserTypeDto(userType), HttpStatus.OK);
    }

    @GetMapping("/{id}")
    public ResponseEntity<UserDto> getUserById(@PathVariable Long id) {
        UserDto userDto = userService.getUserById(id);
        return new ResponseEntity<>(userDto, HttpStatus.OK);
    }

    @GetMapping("/{id}/projects")
    public ResponseEntity<List<JobDto>> getUserProjects(@PathVariable Long id) {
        List<JobDto> userProjects = jobApplicationService.getUserProjects(id);
        return new ResponseEntity<>(userProjects, HttpStatus.OK);
    }
}
```

Додаток В1.

```
package sumdu.edu.ua.freelanceforge.service;

import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import sumdu.edu.ua.freelanceforge.dao.*;
import sumdu.edu.ua.freelanceforge.dao.dto.JobApplicationDto;
import sumdu.edu.ua.freelanceforge.dao.dto.JobDto;
import sumdu.edu.ua.freelanceforge.dao.mapper.MyMapper;
import sumdu.edu.ua.freelanceforge.repository.JobApplicationRepository;

import java.time.LocalDateTime;
import java.util.List;

@Service
@RequiredArgsConstructor
public class JobApplicationService {
    private final JobApplicationRepository jobApplicationRepository;
    private final UserService userService;
    private final JobService jobService;
    private final MyMapper mapper;

    public void applyForJob(String username, Long id) {
        User freelancer = userService.findByUsername(username);
        Job job = jobService.getJobById(id);
        JobApplication jobApplication = JobApplication.builder()
                .job(job)
                .freelancer(freelancer)
                .status(JobApplicationStatus.PENDING)
                .createdAt(LocalDateTime.now())
                .updatedAt(LocalDateTime.now())
                .build();
        createJobApplication(jobApplication);
    }
```

```
    public JobApplicationDto createJobApplication(JobApplication jobApplication) {
        return
mapper.jobApplicationToJobApplicationDto(jobApplicationRepository.save(jobApplication));
    }

    public List<JobApplication> getAllJobApplications() {
        return jobApplicationRepository.findAll();
    }

    @Transactional
    public void acceptApplication(Long id) {
        JobApplication jobApplication =
jobApplicationRepository.findById(id).orElseThrow();
        jobApplication.setStatus(JobApplicationStatus.ACCEPTED);
        jobService.setJobStatus(jobApplication.getJob().getId(), JobStatus.IN_PROGRESS);
        jobApplication.setUpdatedAt(LocalDateTime.now());
    }

    @Transactional
    public void rejectApplication(Long id) {
        JobApplication jobApplication =
jobApplicationRepository.findById(id).orElseThrow();
        jobApplication.setStatus(JobApplicationStatus.REJECTED);
        jobApplication.setUpdatedAt(LocalDateTime.now());
    }

    public List<JobDto> getUserProjects(Long id) {
        List<JobApplication> allJobApplications = getAllJobApplications();
        return allJobApplications.stream()
                .filter(application -> application.getFreelancer().getId().equals(id))
                .filter(application -> application.getStatus() !=
JobApplicationStatus.REJECTED)
                .map(JobApplication::getJob)
                .filter(job -> job.getStatus() != JobStatus.OPEN)
                .map(mapper::jobToJobDto)
                .toList();
    }

    public boolean isJobAccepted(Long id) {
        return getAllJobApplications().stream()
                .anyMatch(application -> application.getJob().getId().equals(id)
                        && application.getStatus() == JobApplicationStatus.ACCEPTED);
    }
}
```

Додаток Г1.

```
package sumdu.edu.ua.freelanceforge.service;

import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import sumdu.edu.ua.freelanceforge.dao.Job;
import sumdu.edu.ua.freelanceforge.dao.JobStatus;
import sumdu.edu.ua.freelanceforge.dao.User;
import sumdu.edu.ua.freelanceforge.repository.JobRepository;

import java.time.LocalDateTime;
import java.util.List;

@Service
@RequiredArgsConstructor
public class JobService {
    private final JobRepository jobRepository;

    public List<Job> getAllJobs() {
        return jobRepository.findAll();
    }
```

```
    public Job getJobById(Long id) {
        return jobRepository.findById(id).orElseThrow(() -> new RuntimeException("job not
found"));
    }

    public Job setJobStatus(Long id, JobStatus status) {
        Job job = getJobById(id);
        job.setStatus(status);
        return jobRepository.save(job);
    }

    public Job createJob(User user, Job job) {
        job.setClient(user);
        job.setStatus(JobStatus.OPEN);
        job.setCreatedAt(LocalDateTime.now());
        job.setUpdatedAt(LocalDateTime.now());
        return jobRepository.save(job);
    }

    public void deleteJob(Long id) {
        jobRepository.deleteById(id);
    }
}
```

Додаток Д1.

```
package sumdu.edu.ua.freelanceforge.service;

import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import sumdu.edu.ua.freelanceforge.dao.RefreshToken;
import sumdu.edu.ua.freelanceforge.repository.RefreshTokenRepository;
import sumdu.edu.ua.freelanceforge.repository.UserRepository;
import sumdu.edu.ua.freelanceforge.rest.exception.RefreshTokenExpiredException;

import java.time.Instant;
import java.util.Optional;
import java.util.UUID;

@Service
@RequiredArgsConstructor
public class RefreshService {
    private final UserRepository userRepository;
    private final RefreshTokenRepository refreshTokenRepository;

    @Value("${jwt.refreshExpiration}")
    private long refreshExpiration;


    public RefreshToken createRefreshToken(Long userId) {
        RefreshToken refreshToken = new RefreshToken();

        refreshToken.setUser(userRepository.findById(userId).orElseThrow(() -> new
RuntimeException("user not found")));
        refreshToken.setExpiryDate(Instant.now().plusMillis(refreshExpiration));
        refreshToken.setToken(UUID.randomUUID().toString());

        refreshToken = refreshTokenRepository.save(refreshToken);
        return refreshToken;
    }

    public Optional<RefreshToken> findByToken(String token) {
        return refreshTokenRepository.findByToken(token);
    }

    public RefreshToken verifyExpiration(RefreshToken token) {
        if (token.getExpiryDate().compareTo(Instant.now()) < 0) {
```

```
            refreshTokenRepository.delete(token);
            throw new RefreshTokenExpiredException("refresh token expired");
        }
        return token;
    }

    @Transactional
    public void deleteByUserId(Long userId) {

refreshTokenRepository.deleteByUser(userRepository.findById(userId).orElseThrow(() -> new
RuntimeException("user not found")));
    }
}
```

Додаток Е1.

```
package sumdu.edu.ua.freelanceforge.service;

import lombok.RequiredArgsConstructor;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import sumdu.edu.ua.freelanceforge.dao.Job;
import sumdu.edu.ua.freelanceforge.dao.JobApplication;
import sumdu.edu.ua.freelanceforge.dao.Role;
import sumdu.edu.ua.freelanceforge.dao.User;
import sumdu.edu.ua.freelanceforge.dao.dto.JobDto;
import sumdu.edu.ua.freelanceforge.dao.dto.UserDto;
import sumdu.edu.ua.freelanceforge.dao.mapper.MyMapper;
import sumdu.edu.ua.freelanceforge.repository.RoleRepository;
import sumdu.edu.ua.freelanceforge.repository.UserRepository;

import java.time.LocalDateTime;
import java.util.List;

@Service
@RequiredArgsConstructor
public class UserService {
    private final UserRepository repository;
    private final RoleRepository roleRepository;
    private final PasswordEncoder passwordEncoder;
    private final UserRepository userRepository;
    private final MyMapper mapper;

    public User findByUsername(String username) {
        return repository.findByUsername(username).orElseThrow(() -> new
RuntimeException("username not found"));
    }

    public User register(User user) {
        Role roleUser = roleRepository.findByName("ROLE_USER");
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        user.setRoles(List.of(roleUser));
        user.setCreated(LocalDateTime.now());
        user.setUpdated(LocalDateTime.now());
        return repository.save(user);
    }

    public UserDto getUserById(Long id) {
        User user = userRepository.findById(id).orElseThrow(() -> new
RuntimeException("user not found"));
        return mapper.userToUserDto(user);
    }
}
```

Додаток Є1.

```
package sumdu.edu.ua.freelanceforge;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@SpringBootApplication
public class FreelanceForgeApplication {

    public static void main(String[] args) {
        SpringApplication.run(FreelanceForgeApplication.class, args);
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder(8);
    }
}
```

Додаток Ж1.

```
create table users
(
    id         bigint auto_increment primary key,
    username   varchar(50)  not null unique,
    first_name varchar(50)  not null,
    last_name  varchar(50)  not null,
    password   varchar(150) not null,
    type       varchar(50)  not null default 'FREELANCER',
    created_at timestamp    not null default current_timestamp,
    updated_at timestamp    not null default current_timestamp on update
current_timestamp
);

create table roles
(
    id   bigint auto_increment primary key,
    name varchar(50) not null unique
);

create table user_roles
(
    user_id bigint not null,
    role_id bigint not null,
    foreign key (role_id) references roles (id) on delete cascade on update restrict,
    foreign key (user_id) references users (id) on delete cascade on update restrict,
    primary key (user_id, role_id)
);

insert into users (username, first_name, last_name, password)
values ('admin', 'Admin', 'Admin',
'$2a$08$oxNpcH6l3a6YX.4NiUFIB.ZP3O1BIqzzE9b/Um5W4kTrv5GDjQKuS');

insert into roles (name)
values ('role_user'),
       ('role_admin');

insert into user_roles (user_id, role_id)
values (1, 2),
       (1, 1);

create table if not exists refresh_tokens
```

```
(
    id
    bigint
    auto_increment
    primary
    key,
    user_id
    bigint
    not
    null,
    token
    varchar
(
    255
) not null unique,
    expiration timestamp not null,
    foreign key
(
    user_id) references users
(
    id
)
    );
CREATE TABLE jobs
(
    id          BIGINT AUTO_INCREMENT PRIMARY KEY,
    client_id   BIGINT       NOT NULL,
    title       VARCHAR(255) NOT NULL,
    description TEXT         NOT NULL,
    status      VARCHAR(50)  NOT NULL DEFAULT 'OPEN', -- Status of the job (OPEN,
IN_PROGRESS, COMPLETED, CANCELED)
    created_at  TIMESTAMP    NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at  TIMESTAMP    NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    FOREIGN KEY (client_id) REFERENCES users (id)
);

CREATE TABLE job_applications
(
    id            BIGINT AUTO_INCREMENT PRIMARY KEY,
    job_id        BIGINT      NOT NULL,
    freelancer_id BIGINT      NOT NULL,
    status        VARCHAR(50) NOT NULL DEFAULT 'PENDING', -- Status of the application
(PENDING, ACCEPTED, REJECTED)
    created_at    TIMESTAMP   NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at    TIMESTAMP   NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    FOREIGN KEY (job_id) REFERENCES jobs (id),
    FOREIGN KEY (freelancer_id) REFERENCES users (id)
);

CREATE TABLE job_history
(
    id         BIGINT AUTO_INCREMENT PRIMARY KEY,
    job_id     BIGINT      NOT NULL,
    user_id    BIGINT      NOT NULL,
    status     VARCHAR(50) NOT NULL, -- Status of the job (DONE, PAID, APPROVED)
    created_at TIMESTAMP   NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (job_id) REFERENCES jobs (id),
    FOREIGN KEY (user_id) REFERENCES users (id)
);
```

Додаток З1.

```
spring.application.name=freelanceForge

spring.datasource.url=jdbc:mysql://localhost:3306/db
spring.datasource.username=root
spring.datasource.password=root

jwt.secret=secret
jwt.expiration=900000
jwt.refreshExpiration=1209600000
```

Додаток И1.

```
import makeStyles from "@mui/styles/makeStyles";
import {useEffect, useState} from "react";
import {endpoints, fetchWrapper} from "../../utils/api";
import {Button, Card, CardContent, Container, Grid, Typography} from "@mui/material";
import {getAccessToken} from "../../utils/auth";
import {useNavigate} from "react-router-dom";
import {Alert} from "@mui/material";

const useStyles = makeStyles((theme) => ({
  root: {
    paddingTop: theme.spacing(4),
    paddingBottom: theme.spacing(4),
  },
  card: {
    marginBottom: theme.spacing(2),
  },
  title: {
    fontSize: 20,
    marginBottom: theme.spacing(2),
  },
  cardContent: {
    paddingBottom: theme.spacing(2),
  },
  button: {
    marginRight: theme.spacing(2),
    marginTop: theme.spacing(1),
    marginLeft: theme.spacing(1),
  },
}));

const ClientApplications = () => {
  const classes = useStyles();
  const [applications, setApplications] = useState([]);
  const accessToken = getAccessToken() || null;
  const navigate = useNavigate();
  const [error, setError] = useState(false);

  const fetchJobAccepted = async (jobId) => {
    const response = await fetchWrapper(endpoints.jobApplications + /isAccepted/${jobId},
{
      method: 'GET',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': Bearer ${accessToken},
      }
    });
    return response.ok;
  };

  useEffect(() => {
    const fetchApplications = async () => {
      const response = await fetchWrapper(endpoints.jobApplications, {
```

```
        method: 'GET',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': Bearer ${accessToken},
        }
      });
      const data = await response.clone().json();
      setApplications(data);
    };

    fetchApplications();
  }, []);

  const handleViewProfile = (applicantId) => {
    navigate('/profile/' + applicantId);
  }

  const handleAccept = async (jobId) => {
    const isJobAccepted = await fetchJobAccepted(jobId);
    if (isJobAccepted) {
      setError(true);
      setTimeout(() => {
        setError(false);
      }, 5000);
      return;
    }

    await fetchWrapper(endpoints.jobApplications + /accept/${jobId}, {
      method: 'GET',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': Bearer ${accessToken},
      }
    });
    navigate('/projects');
  }

  const handleReject = (applicationId) => {
    const rejectApplication = async () => {
      const response = await fetchWrapper(endpoints.jobApplications +
/reject/${applicationId}, {
        method: 'GET',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': Bearer ${accessToken},
        }
      });
    }
    rejectApplication();
    navigate('/projects');
  }

  return (
    <Container className={classes.root}>
      <Typography variant="h4" className={classes.title} gutterBottom>
        Applications for Your Projects
      </Typography>
      <Grid container spacing={3}>
        {applications.map((application) => (
          <Grid item xs={12} md={6} key={application.id}>
            <Card className={classes.card}>
              <CardContent className={classes.cardContent}>
                <Typography variant="h6">{application.job.title}</Typography>
                <Typography variant="subtitle1">Applicant:
{application.freelancer.username}</Typography>
              </CardContent>
              <Button
                variant="contained"
                color="primary"
                className={classes.button}
                onClick={() => handleViewProfile(application.freelancer.id)}
```

```
              >
                View Profile
              </Button>
              <Button variant="outlined" color="primary" className={classes.button}
onClick={() => handleAccept(application.id)} disabled={application.status !== 'PENDING'}>
                Accept
              </Button>
              <Button variant="outlined" color="secondary" className={classes.button}
onClick={() => handleReject(application.id)}>
                Reject
              </Button>
            </Card>
          </Grid>
        ))}
      </Grid>
      {error ?
        <Alert sx={{
          mt: 4,
          width: '40%',
        }}
               variant="filled" severity="error" >
          Job already has an accepted application
        </Alert>
        :
        ""
      }
    </Container>
  );
};

export default ClientApplications;
```

Додаток І1.

```
import makeStyles from "@mui/styles/makeStyles";
import { Button, Card, CardActions, CardContent, CircularProgress, Container, Grid,
Typography, Avatar } from "@mui/material";
import { useEffect, useState } from "react";
import { endpoints, fetchWrapper } from "../../utils/api";
import { getAccessToken } from "../../utils/auth";
import { useParams, useNavigate } from "react-router-dom";

const useStyles = makeStyles((theme) => ({
  root: {
    paddingTop: theme.spacing(4),
    paddingBottom: theme.spacing(4),
  },
  card: {
    marginBottom: theme.spacing(2),
  },
  profileHeader: {
    display: 'flex',
    alignItems: 'center',
    marginBottom: theme.spacing(4),
  },
  avatar: {
    width: theme.spacing(10),
    height: theme.spacing(10),
    marginRight: theme.spacing(2),
  },
  projectsTitle: {
    marginTop: theme.spacing(4),
  },
  loader: {
    display: 'flex',
    justifyContent: 'center',
    alignItems: 'center',
```

```
    height: '100vh',
  },
  projectDescription: {
    overflow: 'hidden',
    textOverflow: 'ellipsis',
    display: '-webkit-box',
    WebkitLineClamp: 3,
    WebkitBoxOrient: 'vertical',
  },
  statusOpen: {
    color: theme.palette.success.main,
  },
  statusInProgress: {
    color: theme.palette.warning.main,
  },
  statusCompleted: {
    color: theme.palette.info.main,
  },
  statusCanceled: {
    color: theme.palette.error.main,
  },
  backButton: {
    marginBottom: theme.spacing(2),
  },
}));

const FreelancerProfile = () => {
  const classes = useStyles();
  const params = useParams();
  const navigate = useNavigate();
  const accessToken = getAccessToken() || null;
  const [projects, setProjects] = useState([]);
  const [user, setUser] = useState({});
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchProfile = async () => {
      const response = await fetchWrapper(endpoints.user + /${params.id}/projects, {
        method: 'GET',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': Bearer ${accessToken},
        }
      });
      const projects = await response.json();
      setProjects(projects);
    };
    const fetchUser = async () => {
      const response = await fetchWrapper(endpoints.user + /${params.id}, {
        method: 'GET',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': Bearer ${accessToken},
        }
      });
      const user = await response.json();
      setUser(user);
    };
    if (accessToken) {
      Promise.all([fetchProfile(), fetchUser()]).then(() => {
        setLoading(false);
      });
    }
  }, [accessToken, params.id]);

  const getStatusClass = (status) => {
    switch (status) {
      case 'OPEN':
        return classes.statusOpen;
      case 'IN_PROGRESS':
        return classes.statusInProgress;
```

```
      case 'COMPLETED':
        return classes.statusCompleted;
      case 'CANCELED':
        return classes.statusCanceled;
      default:
        return '';
    }
  };

  return (
    <Container className={classes.root}>
      {loading ? (
        <div className={classes.loader}>
          <CircularProgress />
        </div>
      ) : (
        <Grid container spacing={4}>
          <Grid item xs={12}>
            <Button
              variant="contained"
              color="primary"
              onClick={() => navigate('/applications')}
              className={classes.backButton}
            >
              Back to Applications
            </Button>
          </Grid>
          <Grid item xs={12} className={classes.profileHeader}>
            <Avatar
className={classes.avatar}>{user.firstName?.charAt(0)}{user.lastName?.charAt(0)}</Avatar>
            <div>
              <Typography variant="h4">
                {user.firstName} {user.lastName}
              </Typography>
              <Typography variant="subtitle1" color="textSecondary">
                @{user.username}
              </Typography>
            </div>
          </Grid>
          <Grid item xs={12}>
            <Typography variant="h5" className={classes.projectsTitle} gutterBottom>
              Projects
            </Typography>
            <Grid container spacing={2}>
              {projects.map((project) => (
                <Grid item xs={12} sm={6} md={4} key={project.id}>
                  <Card className={classes.card}>
                    <CardContent>
                      <Typography variant="h6" gutterBottom>
                        {project.title}
                      </Typography>
                      <Typography variant="body2" color="textSecondary"
className={classes.projectDescription}>
                        {project.description}
                      </Typography>
                      <Typography variant="body2"
className={getStatusClass(project.status)}>
                        Status: {project.status}
                      </Typography>
                    </CardContent>
                    <CardActions>
                      <Button size="small" variant="outlined" color="primary"
href={/project/${project.id}}>
                        Learn More
                      </Button>
                    </CardActions>
                  </Card>
                </Grid>
              ))}
            </Grid>
          </Grid>
```

```
        </Grid>
      ))}
    </Container>
  );
};

export default FreelancerProfile;
```

Додаток Ї1.

```
import {Button, Container, Grid, Paper, Typography} from "@mui/material";
import makeStyles from '@mui/styles/makeStyles';
import WorkIcon from "@mui/icons-material/Work";
import {getAccessToken} from "../../utils/auth";

const useStyles = makeStyles((theme) => ({
  root: {
    flexGrow: 1,
  },
  appBar: {
    marginBottom: theme.spacing(4),
  },
  hero: {
    padding: theme.spacing(8, 0),
    backgroundColor: theme.palette.background.paper,
  },
  heroContent: {
    maxWidth: 600,
    margin: '0 auto',
  },
  heroButtons: {
    marginTop: theme.spacing(4),
  },
  features: {
    padding: theme.spacing(8, 0),
    backgroundColor: theme.palette.grey[200],
  },
  featureItem: {
    padding: theme.spacing(4),
  },
  testimonials: {
    padding: theme.spacing(8, 0),
  },
  testimonialItem: {
    padding: theme.spacing(4),
  },
  cta: {
    padding: theme.spacing(8, 0),
    backgroundColor: theme.palette.primary.main,
    color: theme.palette.common.white,
  },
  avatar: {
    backgroundColor: theme.palette.primary.main,
    margin: '0 auto',
  },
}));

const Home = () => {
  const classes = useStyles();
  const token = getAccessToken() || null;

  return (
    <div className={classes.root}>
      <div className={classes.hero}>
        <Container className={classes.heroContent}>
          <Typography variant="h2" align="center" gutterBottom>
            Welcome to FreelanceHub
```

```
          </Typography>
          <Typography variant="h5" align="center" paragraph>
            Your gateway to finding and offering freelance services!
          </Typography>
          <div className={classes.heroButtons}>
            <Grid container spacing={2} justifyContent="center">
              {!token ?
                <Grid item>
                  <Button variant="contained" color="primary" href="/signup">
                    Get Started
                  </Button>
                </Grid>
                :
                ``
              }
            </Grid>
          </div>
        </Container>
      </div>

      <div className={classes.features}>
        <Container>
          <Typography variant="h4" align="center" gutterBottom>
            Key Features
          </Typography>
          <Grid container spacing={4}>
            <Grid item xs={12} sm={6} md={3}>
              <Paper className={classes.featureItem}>
                <WorkIcon fontSize="large" color="primary"/>
                <Typography variant="h6" gutterBottom>
                  Find Projects
                </Typography>
                <Typography>
                  Discover projects that match your skills and expertise.
                </Typography>
              </Paper>
            </Grid>
            <Grid item xs={12} sm={6} md={3}>
              <Paper className={classes.featureItem}>
                <WorkIcon fontSize="large" color="primary"/>
                <Typography variant="h6" gutterBottom>
                  Manage Profile
                </Typography>
                <Typography>
                  Create and manage your professional freelancer profile.
                </Typography>
              </Paper>
            </Grid>
            <Grid item xs={12} sm={6} md={3}>
              <Paper className={classes.featureItem}>
                <WorkIcon fontSize="large" color="primary"/>
                <Typography variant="h6" gutterBottom>
                  Showcase Portfolio
                </Typography>
                <Typography>
                  Display your work to attract potential clients.
                </Typography>
              </Paper>
            </Grid>
            <Grid item xs={12} sm={6} md={3}>
              <Paper className={classes.featureItem}>
                <WorkIcon fontSize="large" color="primary"/>
                <Typography variant="h6" gutterBottom>
                  Get Reviews
                </Typography>
                <Typography>
                  Receive ratings and feedback from clients.
                </Typography>
              </Paper>
            </Grid>
          </Grid>
```

```
        </Container>
      </div>

      {!token ?
        <div className={classes.cta}>
          <Container>
            <Typography variant="h4" align="center" gutterBottom>
              Join Our Freelance Community!
            </Typography>
            <Grid container spacing={2} justifyContent="center">
              <Grid item>
                <Button variant="contained" color="secondary" href="/signup">
                  Sign Up Now
                </Button>
              </Grid>
            </Grid>
          </Container>
        </div>
        :
        ``
      }
    </div>
  );
};

export default Home;
```

Додаток Й1.

```
import makeStyles from "@mui/styles/makeStyles";
import {useState} from "react";
import {useAuth} from "../../utils/auth";
import {endpoints, fetchWrapper} from "../../utils/api";
import {Avatar, Box, Button, Container, Grid, Link, Paper, TextField, Typography} from
"@mui/material";
import LockOutlinedIcon from '@material-ui/icons/LockOutlined';
import {Alert} from "@mui/lab";

const useStyles = makeStyles((theme) => ({
  root: {
    height: '100vh',
    display: 'flex',
    alignItems: 'center',
    justifyContent: 'center',
    paddingTop: '35px'
  },
  paper: {
    padding: theme.spacing(4),
    display: 'flex',
    flexDirection: 'column',
    alignItems: 'center',
  },
  avatar: {
    margin: theme.spacing(1),
    backgroundColor: theme.palette.primary.main,
  },
  form: {
    width: '100%',
    marginTop: theme.spacing(1),
  },
  submit: {
    margin: theme.spacing(3, 0, 2),
  },
}));

function Login() {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [loginError, setLoginError] = useState(false);
```

```
const {onLogin} = useAuth();

const handleAlertClose = () => {
  setLoginError(false);
}

const handleLogin = async (e) => {
  e.preventDefault();
  const response = await fetchWrapper(endpoints.login, {
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: {
      username,
      password,
    },
  });
  if (response.ok) {
    const userData = await response.json();
    onLogin(userData);
  } else {
    setLoginError(true)
    setTimeout(() => {
      setLoginError(false)
    }, 5000)
  }
};

const classes = useStyles();

return (
  <Container component="main" maxWidth="xs" className={classes.root}>
    <Paper className={classes.paper}>
      <Avatar className={classes.avatar}>
        <LockOutlinedIcon />
      </Avatar>
      <Typography component="h1" variant="h5">
        Login
      </Typography>
      <form className={classes.form} onSubmit={handleLogin} noValidate>
        <TextField
          variant="outlined"
          margin="normal"
          required
          fullWidth
          id="username"
          label="Username"
          name="username"
          autoComplete="username"
          autoFocus
          onChange={(e) => setUsername(e.target.value)}
        />
        <TextField
          variant="outlined"
          margin="normal"
          required
          fullWidth
          name="password"
          label="Password"
          type="password"
          id="password"
          autoComplete="current-password"
          onChange={(e) => setPassword(e.target.value)}
        />
        <Button
          type="submit"
          fullWidth
          variant="contained"
          color="primary"
          className={classes.submit}
        >
```

```
            Login
          </Button>
          <Grid container>
            <Grid item>
              <Link href="/register" variant="body2">
                {"Don't have an account? Register"}
              </Link>
            </Grid>
          </Grid>
        </form>
      </Paper>
      <Box style={{
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center',
      }}>
        {loginError ?
          <Alert sx={{
            mt: 4,
            width: '40%',
          }}
                 variant="filled" severity="error" onClick={handleAlertClose}>
            Wrong username or password
          </Alert>
          :
          ""
        }
      </Box>
    </Container>
  );
}

  export default Login;
```

Додаток К1.

```
import {useNavigate, useParams} from "react-router-dom";
import {useEffect, useState} from "react";
import makeStyles from "@mui/styles/makeStyles";
import {Button, CircularProgress, Container, Paper, Typography} from "@mui/material";
import {endpoints, fetchWrapper} from "../../utils/api";
import {getAccessToken} from "../../utils/auth";
import {Alert} from "@mui/lab";

const useStyles = makeStyles((theme) => ({
  root: {
    marginTop: theme.spacing(4),
    padding: theme.spacing(2),
  },
  paper: {
    padding: theme.spacing(3),
  },
  loader: {
    display: 'flex',
    justifyContent: 'center',
    alignItems: 'center',
    height: '200px',
  },
  buttonContainer: {
    marginTop: theme.spacing(2),
    display: 'flex',
    justifyContent: 'flex-end',
  },
}));

const ProjectDetails = () => {
  const classes = useStyles();
```

```
  const {projectId} = useParams();
  const [project, setProject] = useState(null);
  const accessToken = getAccessToken() || null;
  const [loading, setLoading] = useState(true);
  const [userType, setUserType] = useState('');
  const [alreadyApplied, setAlreadyApplied] = useState(false);
  const navigate = useNavigate();
  const [error, setError] = useState(false);

  useEffect(() => {
    const fetchType = async () => {
      const response = await fetchWrapper(endpoints.userType, {
        method: 'GET',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': Bearer ${accessToken},
        }
      });
      const type = await response.clone().json();
      setUserType(type.userType);
    }
    if (accessToken) {
      fetchType();
    }
  }, []);

  useEffect(() => {
    const fetchProject = async () => {
      const response = await fetchWrapper(endpoints.jobs + /${projectId}, {
        method: 'GET',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': Bearer ${accessToken},
        }
      });
      const project = await response.clone().json();
      setProject(project);
      setLoading(false);
    }
    fetchProject()
  }, [projectId]);

  useEffect(() => {
    if (userType !== 'FREELANCER') return;
    const fetchProject = async () => {
      const response = await fetchWrapper(endpoints.jobs + /applied/${projectId}, {
        method: 'GET',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': Bearer ${accessToken},
        }
      });
      if (response.ok) {
        setAlreadyApplied(true)
      } else {
        setAlreadyApplied(false)
      }
    }
    fetchProject()
  }, [projectId]);

  const handleApply = () => {
    const applyForJob = async () => {
      const response = await fetchWrapper(endpoints.jobs + /apply/${projectId}, {
        method: 'GET',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': Bearer ${accessToken},
        }
      });
      if (response.ok) {
```

```
        navigate('/projects')
      } else {
        setError(true);
        setTimeout(() => {
          setError(false);
        }, 5000)
      }
    }
    applyForJob()
  };

  const handleCompleteProject = () => {
    const completeJob = async () => {
      const response = await fetchWrapper(endpoints.jobs + /complete/${projectId}, {
        method: 'GET',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': Bearer ${accessToken},
        }
      });
      if (response.ok) {
        navigate('/projects')
      } else {
        setError(true);
        setTimeout(() => {
          setError(false);
        }, 5000)
      }
    }
    completeJob();
  }

  const handleDelete = () => {
    const deleteJob = async () => {
      const response = await fetchWrapper(endpoints.jobs + /${projectId}, {
        method: 'DELETE',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': Bearer ${accessToken},
        }
      });
      if (response.ok) {
        // todo
        navigate('/projects')
        console.log('Job deleted');
      } else {
        console.log('Failed to delete job')
      }
    }
    deleteJob();
  }

  return (
    <Container maxWidth="md" className={classes.root}>
      {loading ? (
        <div className={classes.loader}>
          <CircularProgress/>
        </div>
      ) : (
        <Paper elevation={3} className={classes.paper}>
          <Typography variant="h4" gutterBottom>
            {project.title}
          </Typography>
          <Typography variant="subtitle1" gutterBottom>
            Client: {project.client.username}
          </Typography>
          <Typography variant="subtitle1" gutterBottom>
            Status: {project.status}
          </Typography>
          <Typography variant="body1" gutterBottom>
            Description: {project.description}
```

```
          </Typography>
          <Typography variant="body2" gutterBottom>
            Created At: {new Date(project.createdAt).toLocaleString()}
          </Typography>
          {userType === 'FREELANCER' && (
            <div className={classes.buttonContainer}>
              <Button variant="contained" color="primary" onClick={handleApply}
disabled={alreadyApplied || project.status !== 'OPEN'}>
                Apply for Job
              </Button>
              <Button variant="contained" color="primary" onClick={handleCompleteProject}
disabled={project.status !== 'IN_PROGRESS'}>
                Completed
              </Button>
            </div>
          )}
          {userType === 'CUSTOMER' && (
            <div className={classes.buttonContainer}>
              <Button variant="contained" color="primary" onClick={handleDelete}
disabled={project.status !== 'OPEN'}>
                Delete job
              </Button>
            </div>
          )}
        </Paper>
      )}
      {error ?
        <Alert sx={{
          mt: 4,
          width: '40%',
        }}
               variant="filled" severity="error">
          Something went wrong
        </Alert>
        :
        ""
      }
    </Container>
  );
};

export default ProjectDetails;
```

Додаток Л1.

```
import makeStyles from "@mui/styles/makeStyles";
import {useEffect, useState} from "react";
import {Button, Card, CardActions, CardContent, CircularProgress, Container, Grid,
Typography} from "@mui/material";
import {endpoints, fetchWrapper} from "../../utils/api";
import {getAccessToken} from "../../utils/auth";
import {useNavigate} from "react-router-dom";
import NewProjectModal from "../NewProjectModal";

const useStyles = makeStyles((theme) => ({
  root: {
    paddingTop: theme.spacing(4),
    paddingBottom: theme.spacing(4),
  },
  card: {
    marginBottom: theme.spacing(2),
    marginTop: theme.spacing(2),
  },
  title: {
    fontSize: 14,
  },
  pos: {
```

```
    marginBottom: 12,
  },
  statusOpen: {
    color: theme.palette.success.main,
  },
  statusInProgress: {
    color: theme.palette.warning.main,
  },
  statusCompleted: {
    color: theme.palette.info.main,
  },
  statusCanceled: {
    color: theme.palette.error.main,
  },
}));

const Projects = () => {
  const classes = useStyles();
  const [projects, setProjects] = useState([]);
  const [userType, setUserType] = useState(``);
  const [loading, setLoading] = useState(true);
  const [open, setOpen] = useState(false);
  const accessToken = getAccessToken() || null;
  const navigate = useNavigate();

  const handleApplicationsPage = () => {
    navigate('/applications');
  };

  const handleModalOpen = () => {
    setOpen(true);
  }

  const handleModalClose = () => {
    setOpen(false);
  }

  const handleCreateJob = async (data) => {
    const createJob = async () => {
      const response = await fetchWrapper(endpoints.jobs, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': Bearer ${accessToken},
        },
        body: {
          title: data.jobTitle,
          description: data.description,
        }
      });
      const newProject = await response.clone().json();
      setProjects([...projects, newProject]);
    }
    createJob()
  }

  useEffect(() => {
    const fetchType = async () => {
      const response = await fetchWrapper(endpoints.userType, {
        method: 'GET',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': Bearer ${accessToken},
        }
      });
      const type = await response.clone().json();
      setUserType(type.userType);
    }
    if (accessToken) {
      fetchType();
    }
```

```
  }, []);

  useEffect(() => {
    const fetchProjects = async () => {
      const response = await fetchWrapper(endpoints.jobs, {
        method: 'GET',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': Bearer ${accessToken},
        }
      });
      const projects = await response.clone().json();
      setProjects(projects);
      setLoading(false);
    }
    fetchProjects()
  }, [userType]);

  const getStatusClass = (status) => {
    switch (status) {
      case 'OPEN':
        return classes.statusOpen;
      case 'IN_PROGRESS':
        return classes.statusInProgress;
      case 'COMPLETED':
        return classes.statusCompleted;
      case 'CANCELED':
        return classes.statusCanceled;
      default:
        return '';
    }
  };

  const truncateDescription = (description, maxLength) => {
    if (description.length <= maxLength) {
      return description;
    }
    return description.substring(0, maxLength) + '...';
  };

  return (
    <Container className={classes.root}>
      {loading ? (
        <div className={classes.loader}>
          <CircularProgress/>
        </div>
      ) : (
        <Grid>
          <Typography variant="h4" gutterBottom>
            {
              userType === 'FREELANCER' ? 'Available Projects' : 'My Projects'
            }
          </Typography>
          {userType !== 'FREELANCER' && (
            <Grid>
              <Button variant="contained" color="primary"
onClick={handleApplicationsPage}>
                View Applications
              </Button>
              <Button variant="contained" color="primary" onClick={handleModalOpen}>
                Create new project
              </Button>
            </Grid>
          )}
          <Grid container spacing={3}>
            {projects.map((project) => (
              <Grid item xs={12} sm={6} md={4} key={project.id}>
                <Card className={classes.card}>
                  <CardContent>
                    <Typography className={classes.title} color="textSecondary"
gutterBottom>
```

```
                    Project ID: {project.id}
                  </Typography>
                  <Typography variant="h5" component="h2">
                    {project.title}
                  </Typography>
                  <Typography className={classes.pos} color="textSecondary">
                    Client: {project.client.username}
                  </Typography>
                  <Typography className={getStatusClass(project.status)}>
                    Status: {project.status}
                  </Typography>
                  <Typography variant="body2" component="p">
                    {truncateDescription(project.description, 100)}
                  </Typography>
                  <Typography className={classes.pos} color="textSecondary">
                    Created At: {new Date(project.createdAt).toLocaleString()}
                  </Typography>
                </CardContent>
                <CardActions>
                  <Button size="small" href={/project/${project.id}}>Learn
More</Button>
                </CardActions>
              </Card>
            </Grid>
          ))}
        </Grid>
      </Grid>
    )}
    <NewProjectModal
      open={open}
      onClose={handleModalClose}
      onCreate={handleCreateJob}
    />
  </Container>
 );
};

export default Projects;
import {
  Avatar,
  Box,
  Button,
  Container,
  FormControl,
  Grid,
  InputLabel, MenuItem,
  Paper, Select,
  TextField,
  Typography
} from "@mui/material";
import LockOutlinedIcon from "@material-ui/icons/LockOutlined";
import makeStyles from "@mui/styles/makeStyles";
import {useState} from "react";
import {useNavigate} from "react-router-dom";
import {endpoints, fetchWrapper} from "../../utils/api";
import {clearUserData} from "../../utils/auth";
import {Alert} from "@mui/lab";

const useStyles = makeStyles((theme) => ({
  root: {
    height: '100vh',
    display: 'flex',
    alignItems: 'center',
    justifyContent: 'center',
    paddingTop: '35px'
  },
  paper: {
    padding: theme.spacing(4),
    display: 'flex',
    flexDirection: 'column',
    alignItems: 'center',
```

```
  },
  avatar: {
    margin: theme.spacing(1),
    backgroundColor: theme.palette.primary.main,
  },
  form: {
    width: '100%',
    marginTop: theme.spacing(1),
  },
  submit: {
    margin: theme.spacing(3, 0, 2),
  },
}));

const Register = () => {
  const classes = useStyles();

  const [firstName, setFirstName] = useState('');
  const [lastName, setLastName] = useState('');
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [userType, setUserType] = useState('Freelancer');
  const [loginError, setLoginError] = useState(false);
  const navigate = useNavigate();

  const handleAlertClose = () => {
    setLoginError(false);
  }

  const handleChangeType = (event) => {
    const value = event.target ? event.target.value : event;
    setUserType(value);
  };

  const handleLogin = async (e) => {
    e.preventDefault();
    const response = await fetchWrapper(endpoints.signup, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: {
        username,
        password,
        firstName,
        lastName,
        userType: userType.toUpperCase()
      },
    });
    if (response.ok) {
      clearUserData();
      navigate('/login');
    } else {
      setLoginError(true);
      setTimeout(() => {
        setLoginError(false);
      }, 5000)
    }
  };

  return (
    <Container component="main" maxWidth="xs" className={classes.root}>
      <Paper className={classes.paper}>
        <Avatar className={classes.avatar}>
          <LockOutlinedIcon />
        </Avatar>
        <Typography component="h1" variant="h5">
          Register
        </Typography>
        <form className={classes.form} onSubmit={handleLogin} noValidate>
          <TextField
            variant="outlined"
            margin="normal"
```

```
            required
            fullWidth
            id="firstName"
            label="First Name"
            name="firstName"
            autoComplete="fname"
            autoFocus
            onChange={(e) => setFirstName(e.target.value)}
          />
          <TextField
            variant="outlined"
            margin="normal"
            required
            fullWidth
            id="lastName"
            label="Last Name"
            name="lastName"
            autoComplete="lname"
            onChange={(e) => setLastName(e.target.value)}
          />
          <TextField
            variant="outlined"
            margin="normal"
            required
            fullWidth
            id="username"
            label="Username"
            name="username"
            autoComplete="username"
            onChange={(e) => setUsername(e.target.value)}
          />
          <TextField
            variant="outlined"
            margin="normal"
            required
            fullWidth
            name="password"
            label="Password"
            type="password"
            id="password"
            autoComplete="current-password"
            onChange={(e) => setPassword(e.target.value)}
          />
          <FormControl sx={{paddingTop: '10px', paddingBottom: '10px'}} fullWidth>
            <InputLabel id="demo-simple-select-label">Type</InputLabel>
            <Select
              labelId="demo-simple-select-label"
              id="demo-simple-select"
              label="Age"
              value={userType}
              onChange={handleChangeType}
            >
              <MenuItem value={'Freelancer'}>Freelancer</MenuItem>
              <MenuItem value={'Customer'}>Customer</MenuItem>
            </Select>
          </FormControl>
          <Button
            type="submit"
            fullWidth
            variant="contained"
            color="primary"
            className={classes.submit}
          >
            Register
          </Button>
          <Grid container>
            <Grid item xs>
              <Button color="primary" href="/login">
                Already have an account? Login
              </Button>
            </Grid>
```

```
          </Grid>
        </form>
      </Paper>
      <Box style={{
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center',
      }}>
        {loginError ?
          <Alert sx={{
            mt: 4,
            width: '40%',
          }}
                 variant="filled" severity="error" onClick={handleAlertClose}>
            Error signing up
          </Alert>
          :
          ""
        }
      </Box>
    </Container>
  );
};

export default Register;
```

Додаток М1.

```
import Navbar from "../components/Navbar";
import {BrowserRouter as Router, Route, Routes} from "react-router-dom";
import AuthProvider from "./AuthProvider";
import ProtectedRoute from "./ProtectedRoute";
import Home from "./pages/Home";
import Login from "./pages/Login";
import Register from "./pages/Register";
import Projects from "./pages/Projects";
import ProjectDetails from "./pages/ProjectDetails";
import ClientApplications from "./pages/ClientApplications";
import FreelancerProfile from "./pages/FreelancerProfile";

function AppRoutes() {
  return (
    <Router>
      <AuthProvider>
        <Navbar/>
        <Routes>
          <Route path="/" element={<Home/>}/>
          <Route path="/home" element={<Home/>}/>
          <Route
            path="/projects"
            element={(
              <ProtectedRoute>
                <Projects/>
              </ProtectedRoute>
            )}
          />
          <Route path="/project/:projectId" element={(
            <ProtectedRoute>
              <ProjectDetails/>
            </ProtectedRoute>
          )}/>
          <Route path="/applications" element={(
            <ProtectedRoute>
              <ClientApplications/>
            </ProtectedRoute>
          )}/>
          <Route path="/profile/:id" element={(
```

```
            <ProtectedRoute>
              <FreelancerProfile/>
            </ProtectedRoute>
          )}/>
          <Route path="/login" element={<Login/>}/>
          <Route path="/register" element={<Register/>}/>
          <Route path="*" element={<p>There&apos;s nothing here: 404!</p>}/>
        </Routes>
      </AuthProvider>
    </Router>
  );
}

export default AppRoutes;
```

Додаток Н1.

```
import {useLocation, useNavigate} from "react-router-dom";
import {useEffect, useMemo, useState} from "react";
import {clearUserData, getAccessToken, getRefreshToken, setUserData, updateAccessToken}
from "../utils/auth";
import {AuthContext} from '../context/AuthContext.jsx';
import {endpoints, fetchWrapper} from "../utils/api";

const ACCESS_TOKEN_EXPIRES_TIME = 10000 * 60 * 9; // 9 min

function AuthProvider({children}) {
  const localAccessToken = getAccessToken() || null;
  const refreshToken = getRefreshToken() || null;
  const navigate = useNavigate();
  const location = useLocation();
  const [isFirstMounted, setIsFirstMounted] = useState(true);

  const handleLogin = (userData) => {
    setUserData(userData);
    const origin = (location.state)?.from?.pathname || '/home';
    navigate(origin);
  };

  const handleLogout = async () => {
    clearUserData();
    await fetchWrapper(endpoints.logout, {
      method: 'DELETE',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': Bearer ${localAccessToken},
      },
      body: {refreshToken},
    });
    navigate('/login');
  };

  async function updateRefreshtoken() {
    const response = await fetchWrapper(endpoints.token, {
      method: 'POST',
      headers: {'Content-Type': 'application/json'},
      body: {refreshToken},
    });

    if (response.ok) {
      const {accessToken} = await response.json();
      updateAccessToken(accessToken);
    } else {
      clearUserData();
      navigate('/login');
      window.location.reload();
    }
```

```
    if (isFirstMounted) {
      setIsFirstMounted(false);
    }
  }

  useEffect(() => {
    if (refreshToken) {
      if (isFirstMounted) {
        updateRefreshtoken();
      }

      const intervalId = setInterval(() => {
        updateRefreshtoken();
      }, ACCESS_TOKEN_EXPIRES_TIME);
      return () => clearInterval(intervalId);
    }
    return undefined;
  }, [localAccessToken]);

  const value = useMemo(() => ({
    token: localAccessToken,
    onLogin: handleLogin,
    onLogout: handleLogout,
  }), [localAccessToken]);

  return (
    <AuthContext.Provider value={value}>
      {children}
    </AuthContext.Provider>
  );
}

export default AuthProvider;
```

Додаток О1.

```
import {AppBar, Box, Button, Grid, IconButton, Toolbar} from "@mui/material";
import WorkIcon from '@mui/icons-material/Work';
import {getAccessToken, useAuth} from "../utils/auth";

function Navbar() {
  const token = getAccessToken() || null;
  const {onLogout} = useAuth();

  const handleLogout = () => {
    onLogout();
  }

  return (
    <Box sx={{flexGrow: 1}}>
      <AppBar position="static">
        <Toolbar>
          <IconButton
            size="large"
            edge="start"
            color="inherit"
            aria-label="menu"
            sx={{mr: 2}}
          >
            <WorkIcon/>
          </IconButton>
          <Grid container justifyContent="space-between" alignItems="center">
            <Grid>
              <Button color="inherit" href="/home">Home</Button>
              <Button color="inherit" href="/projects">Projects</Button>
            </Grid>
            {!token ?
              <Grid item>
                <Button color="inherit" href="/login">Login</Button>
```

```
                <Button color="inherit" href="/register">Register</Button>
              </Grid>
              :
              ``
            }
            {token ?
              <Button color="inherit" onClick={handleLogout}>Logout</Button>
              :
              ``
            }
          </Grid>
        </Toolbar>
      </AppBar>
    </Box>
  );
}

export default Navbar;
```

Додаток П1.

```
import makeStyles from "@mui/styles/makeStyles";
import {useState} from "react";
import {Button, Fade, Modal, TextField, Typography} from "@mui/material";

const useStyles = makeStyles((theme) => ({
  modal: {
    display: 'flex',
    alignItems: 'center',
    justifyContent: 'center',
  },
  paper: {
    backgroundColor: theme.palette.background.paper,
    border: '2px solid #000',
    boxShadow: theme.shadows[5],
    padding: theme.spacing(2, 4, 3),
    minWidth: 300,
  },
  textField: {
    marginBottom: theme.spacing(2),
  },
}));

const NewProjectModal = ({ open, onClose, onCreate }) => {
  const classes = useStyles();
  const [projectTitle, setProjectTitle] = useState('');
  const [description, setDescription] = useState('');

  const handleCreate = () => {
    onCreate({ jobTitle: projectTitle, description });
    onClose();
    setProjectTitle('');
    setDescription('');
  };

  return (
    <Modal
      className={classes.modal}
      open={open}
      onClose={onClose}
      closeAfterTransition
    >
      <Fade in={open}>
        <div className={classes.paper}>
          <Typography variant="h5" gutterBottom>Create New Project</Typography>
          <TextField
            className={classes.textField}
            label="Project Title"
            variant="outlined"
```

```
            fullWidth
            value={projectTitle}
            onChange={(e) => setProjectTitle(e.target.value)}
          />
          <TextField
            className={classes.textField}
            label="Description"
            variant="outlined"
            fullWidth
            multiline
            rows={4}
            value={description}
            onChange={(e) => setDescription(e.target.value)}
          />
          <Button variant="contained" color="primary" onClick={handleCreate}>Create
Project</Button>
        </div>
      </Fade>
    </Modal>
  );
};

export default NewProjectModal;
```

Додаток Р1.

```
import {Navigate, Outlet, useLocation} from "react-router-dom";
import {useAuth} from "../utils/auth";

function ProtectedRoute({
                          redirectPath = '/login',
                          children,
                        }) {
  const { token } = useAuth();
  const location = useLocation();

  if (!token) {
    return <Navigate to={redirectPath} replace state={{ from: location }} />;
  }

  return children || <Outlet />;
}

export default ProtectedRoute;
```

Додаток С1.

```
import {createContext} from 'react';

export const AuthContext = createContext({
  token: '',
  onLogin: () => {
  },
  onLogout: () => {
  },
});
```

Додаток Т1.

```
import {useContext} from 'react';
import {AuthContext} from '../context/AuthContext';

export const useAuth = () => useContext(AuthContext);

export const getUserData = () => {
  if (typeof Storage === 'undefined') return {};
  return JSON.parse(localStorage.getItem('user') || '{}');
};

export const setUserData = (user) => {
  if (user?.constructor.name !== 'Object') {
    throw new Error('No valid data found');
  }
  if (Object.keys(user).length === 0) {
    throw new Error('No data found');
  }
  if (typeof Storage === 'undefined') {
    throw new Error('No valid storage type found');
  }
  localStorage.setItem('user', JSON.stringify(user));
};

export function clearUserData() {
  if (typeof Storage === 'undefined') return;
  localStorage.removeItem('user');
}

export const getRefreshToken = () => {
  if (typeof Storage === 'undefined') return false;
  return JSON.parse(localStorage.getItem('user') || '{}')?.refreshToken;
};

export const getAccessToken = () => {
  if (typeof Storage === 'undefined') {
    return new Error('Storage type not valid');
  }
  return JSON.parse(localStorage.getItem('user') || '{}')?.accessToken;
};

export const updateAccessToken = (token) => {
  if (typeof Storage === 'undefined') return;
  const user = JSON.parse(localStorage.getItem('user') || '{}');
  user.accessToken = token;
  localStorage.setItem('user', JSON.stringify(user));
};

export const isAuthenticated = () => {
  const accessToken = getAccessToken();
  return !(!accessToken);
};

export function getPayloadFromToken(token) {
  if (!token) {
    return {};
  }
  const base64Url = token.split('.')[1];
  const base64 = base64Url.replace('-', '+').replace('_', '/');
  return JSON.parse(window.atob(base64));
}
```

Додаток У1.

```
export const API_HOST = 'http://localhost:8080';

export const endpoints = {
  login: '/auth/login',
  logout: '/auth/logout',
  signup: '/auth/register',
  token: '/auth/refresh',
  userType: '/user/type',
  user: '/user',
  jobs: '/jobs',
  jobApplications: '/jobs/applications',
};

export async function fetchWrapper(endpoint, opts) {
  opts.headers = {
    'Access-Control-Allow-Origin': '*',
    'Content-Type': 'application/json',
    ...opts.headers,
  };
  opts.mode = 'cors';
  if (opts.body) {
    opts.body = JSON.stringify(opts.body);
  }
  return fetch(${API_HOST}${endpoint}, opts);
}
```

Додаток Ф1.

```
import './App.css';
import {createTheme, ThemeProvider} from "@mui/material";
import {LocalizationProvider} from "@mui/x-date-pickers";
import {AdapterLuxon} from "@mui/x-date-pickers/AdapterLuxon";
import AppRoutes from "./components/AppRoutes";

const theme = createTheme({
  components: {
    MuiButtonBase: {
      defaultProps: {
        disableRipple: true,
      },
    },
  },
});

function App() {
  return (
    <LocalizationProvider dateAdapter={AdapterLuxon}>
      <ThemeProvider theme={theme}>
        <AppRoutes/>
      </ThemeProvider>
    </LocalizationProvider>
  );
}

export default App;
```

Додаток Х1.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App/>
  </React.StrictMode>
);

reportWebVitals();
```