ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-застосунку для управління персоналом IT-аутсорсингової компанії»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Олексій ЩЕРБАКОВ
(підпис)

Виконав: здобувач вищої освіти групи ПД-44

Олексій ЩЕРБАКОВ

Керівник: Богдан КАЛИНЮК
асистент кафедри ІПЗ

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Щербакову Олексію Олександровичу

1. Тема кваліфікаційної роботи: «Розробка web-застосунку для управління персоналом IT-аутсорсингової компанії»
керівник кваліфікаційної роботи асистент кафедри ІПЗ Богдан КАЛИНЮК,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи управління персоналом, опис підходів до проектування архітектури web-застосунків, Особливості управління персоналом в IT-футсорсингових компаніях , технічна документація з описом бібліотек для розробки web-застосунків.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Ознайомлення з предметною областю, огляд та аналіз існуючих застосунків для управління персоналом в IT-компаніях.
2. Проектування web-застосунку для управління персоналом в IT-компаніях.
3. Програмна реалізація та опис функціонування застосунку для управління персоналом в IT-компаніях.

4. Тестування web-застосунку.

5. Перелік графічного матеріалу: *презентація*
   1. Аналіз аналогів.
   2. Вимоги до програмного забезпечення.
   3. Програмні засоби реалізації.
   4. Діаграма варіантів використання.
   5. Структура бази данних
   6. Діаграма класів.
   7. Екранні форми.
   8. Апробація результатів дослідження
   9. Висновки

6. Дата видачі завдання«25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Визначення вимог до web-застосунку та огляд та аналіз засобів реалізації застосунку	14.03-26.03.2025	
4	Проектування архітектури застосунку для управління персоналом в IT-аутсорсингових компаніях	27.03-07.04.2025	
5	Програмна реалізація застосунку	08.04-20.04.2025	
6	Тестування web-застосунку	21.04-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Олексій ЩЕРБАКОВ
(підпис)

Керівник
кваліфікаційної роботи _____ Богдан КАЛИНЮК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 55 стор., 3 табл., 34 рис., 24 джерела.

Мета роботи – удосконалення процесів управління персоналом в IT-компаніях.

Об'єкт дослідження – процес управління персоналом в IT-аутсорсинговій компанії.

Предмет дослідження – web-застосунки, призначені для управління персоналом в аутсорсингових IT-компаніях.

Короткий зміст роботи: У роботі проаналізовано особливості процесів управління персоналом в IT-аутсорсингових компаніях та досліджено сучасні підходи до автоматизації цих процесів за допомогою web-застосунків. Розглянуто архітектуру та інструменти розробки web-застосунків на основі Java Spring Boot. Спроектовано та реалізовано web-застосунок для управління персоналом, який забезпечує реєстрацію та авторизацію користувачів, призначення ролей (Team Leader, Developer, QA, Admin), керування профілями співробітників, проектами та задачами. Реалізовано функціонал фільтрації задач за назвою, статусом та пріоритетом, а також алгоритм автоматичного підбору оптимальних співробітників на основі їх ролей і кваліфікацій. Здійснено функціональне та модульне тестування системи. Забезпечено відповідність нефункціональним вимогам: обмеження доступу згідно з роллю, хешування паролів, оптимізація продуктивності (відгук не перевищує 5 секунд). В роботі використано фреймворк Spring для створення конфігурації web-застосунку, Lombok для скорочення необхідного для написання коду, MySQL для збереження даних, HTML/CSS + JS для створення UI.

Сферою використання застосунку є управління персоналом а IT-футсорсингових компаніях.

КЛЮЧОВІ СЛОВА: WEB-ЗАСТОСУНОК, JAVA, SPRING BOOT, УПРАВЛІННЯ ПЕРСОНАЛОМ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ШІ - штучний інтелект.

ІТ - інформаційні технології.

HR - людські ресурси.

HRMS - Human Resource Management System.

МСБ - малий та середній бізнес.

OKR (Objectives and Key Results) - це методологія постановки та досягнення цілей, яка використовується для вимірювання прогресу та синхронізації роботи в компанії, команді або у окремої особи.

KPI (Key Performance Indicators) - це ключові показники ефективності, які використовуються для вимірювання успішності та продуктивності діяльності організації або конкретних процесів.

QA (Quality Assurance) – Забезпечення якості.

ОС - операційна система.

JVM - Java Virtual Machine(віртуальна машина, яка виконує байт-код Java).

MVC, Модель–представлення–контролер - архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

БД - база даних.

IDE - Інтегроване середовище розробки.

СУБД - система управління базою даних.

Моки (mock object) — це спеціальні тестові об'єкти, які імітують поведінку реальних об'єктів і дозволяють відстежувати виклики методів, передані аргументи та перевіряти очікувану поведінку під час тестування.

ВСТУП

Актуальність цієї роботи визначається стрімким розвитком технологій в сучасному світі. Це призводить до цифровізації великої кількості задач, яку необхідно виконувати. Для зменшення навантаження на IT-відділи багато компаній звертаються до сторонніх виконавців, що призводить до збільшення аутсорсингової індустрії. Такі компанії, що спеціалізуються на наданні IT-послуг, повинні постійно стикатися з нестабільністю сучасного ринку праці [1-2].

Економічна нестабільність і впровадження ШІ в IT-індустрії створюють ситуацію, в якій неможливо бути впевненим в стабільності свого робочого місця. Крім того, в IT-індустрії є досить розповсюджена практика зміни робочого місця після одного або декількох виконаних проектів, а молоді люди взагалі досить часто змінюють місця роботи, намагіаючись знайти найвигідніше робоче місце.

Вище зазначені причини разом з переходом багатьох IT-компаній на віддалений або гібридний формат роботи, призводить до зростання попиту на цифрові інструменти, які дозволяють не тільки проводити облік персоналу, а й відстежувати поточні проекти і задачі, призначені працівникам.

Web-застосунки дозволяють централізовано зберігати та обробляти дані працівників, забезпечуючи доступ як з боку HR-фахівців, так і самих співробітників, що сприяє прозорості процесів, оперативності прийнять рішень та підвищенню загальної продуктивності компанії. Таким чином розробка подібного застосунку є актуальним і практично значущим напрямом у сфері IT.

Об’єктом дослідження є процес управління персоналом в IT-аутсорсинговій компанії.

Предметом дослідження є web-застосунки, призначені для управління персоналом в аутсорсингових IT-компаніях.

Метою роботи є удосконалення процесів управління персоналом в IT-компаніях.

Методи дослідження: першим кроком були досліджені методи управління персоналом в IT-компаніях для розуміння з чим варто працювати. Далі були проаналізовані аналогічні програмні рішення для управління персоналом, що дозволило визначити їх переваги і недоліки. На основі отриманої інформації були визначені функціональні і нефункціональні вимоги до розроблюваного ПЗ. Маючи визначені вимоги до застосунку, був проведений огляд технологій для реалізації застосунку і була обрана мова програмування – Java. Для створення web-застосунку був обраний фреймворк - Spring Boot, який не тільки гарно підходить для створення застосунків, а й має велику кількість бібліотек для реалізації необхідного функціоналу в backend частині застосунку. Комбінація HTML + CSS + JS була обрана для створення фронтенд частини застосунку. MySQL була обрана як СУБД для зберігання даних. Дослідження реалізації застосунку проводилося під час безпосередньої розробки.

Наукова новизна роботи полягає в створенні інтегрованої системи управління персоналом з гнучкою рольовою моделлю, наявності декількох функціональних можливостей(сортування задач за їх статусом та пріорітетом, пошук проектів і задач за назвою, написання коментарів до задач) і реалізації алгоритму для підбору працівників базуючись на їх рівні кваліфікації та спеціалізації. Така комбінація дозволяє системі бути ефективною і масштабованою, управляти працівниками, проектами та завданнями з високим рівнем безпеки та гнучкості.

Практична значущість роботи полягає в створенні функціонального web-застосунку, який може бути використаний для управління персоналом в IT-аутсорсингових компаніях.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧ

1.1 Характеристика предметної галузі управління персоналом в IT-аутсорсингових компаніях

Управління персоналом, або ж управління людськими ресурсами – діяльність, пов’язана з підбором, наймом, оцінюванням, розвитком і координацією працівників у межах підприємства. Вона охоплює комплекс заходів, спрямованих на забезпечення компанії кваліфікованими фахівцями з необхідними знаннями, навичками та досвідом для ефективного виконання бізнес-цілей і завдань [3].

Головною метою управління персоналом є підвищення продуктивності та ефективності роботи працівників, одночасно забезпечуючи їхню задоволеність, залученість і мотивацію. Ця сфера охоплює широкий набір функцій, таких як аналіз і розвиток наявного кадрового потенціалу, планування персоналу, підбір і найм працівників, управління результативністю, організація навчання та професійного зростання, надання компенсацій і пільг, налагодження трудових відносин, а також дотримання стандартів охорони здоров’я і безпеки на робочому місці.

HR - надзвичайно важлива функція для будь-якої організації, незалежно від її розміру чи сфери діяльності. Вона виконує ключову роль у залученні й утриманні талановитих працівників, формуванні сприятливої корпоративної культури та забезпеченні відповідності діяльності компанії вимогам трудового законодавства. Ефективна система управління персоналом є основою для сталого розвитку та успішного функціонування підприємства в довгостроковій перспективі.

Система управління персоналом відповідає за:

1. Забезпечення відповідності кількісного та якісного складу персоналу актуальним і майбутнім потребам організації.
2. Залучення кандидатів із необхідними компетенціями та кваліфікацією для виконання визначених завдань в необхідних проектах.

3. Створення умов для професійного та особистісного зростання працівників, організація навчання і тренінгів з метою підвищення їхньої продуктивності.

4. Визначення критеріїв для оцінки результатів роботи та регулярне проведення оцінювання діяльності персоналу.

5. Встановлення справедливої системи оплати праці, запровадження бонусів і стимулюючих заходів для підвищення зацікавленості працівників.

6. Формування позитивного клімату в колективі, розвиток корпоративної культури та сприяння командній роботі.

7. Підтримку персоналу під час організаційних змін, таких як впровадження нових технологій, стратегічних перетворень або реструктуризації.

8. Ведення кадрової документації, оформлення трудових контрактів, облік заробітної плати та інші адміністративні процеси.

9. Налагодження ефективного зворотного зв’язку між керівництвом і персоналом, а також всередині команд.

1.2 Специфіка HR у IT-сфері

Сутність управління персоналом може змінюватися залежно від типу підприємства, оскільки кожна галузь має свої специфічні потреби та випробування. У зв’язку з цим доцільно зосередити увагу на особливостях функціонування системи управління персоналом в IT-сфері.

У контексті IT-компанії управління персоналом охоплює вирішення низки специфічних завдань, що обумовлені стрімкими технологічними змінами та динамікою ринку. Основною метою HR-процесів в IT є формування ефективної, гнучкої та висококваліфікованої команди, здатної реалізовувати стратегічні цілі компанії [4].

Для досягнення зазначеної мети, система управління персоналом в IT-компанії має враховувати наступні аспекти :

1. Розвиток і підтримка кадрів - передбачає пошук та залучення висококваліфікованих фахівців, організацію навчання й професійного розвитку відповідно до потреб компанії та працівників, а також впровадження системи оцінки ефективності роботи.

2. Мотивація персоналу - включає розробку систем оплати праці та бонусів, визнання досягнень співробітників, створення сприятливих умов праці та забезпечення балансу між професійним і особистим життям.

3. Менеджмент і корпоративна культура - охоплює розвиток відкритої та ефективної комунікації між працівниками та керівництвом, формування корпоративних цінностей, які сприяють зростанню компанії та особистому розвитку працівників.

4. Управління ризиками - включає впровадження політик інформаційної безпеки, дотримання вимог щодо конфіденційності, мінімізацію кадрових ризиків та оптимізацію витрат.

5. Управління змінами - передбачає планування й реалізацію змін у структурі організації, трансформацію корпоративної культури, вдосконалення внутрішніх процесів та адаптацію до нових технологій.

6. Управління талантами - спрямоване на створення ефективної стратегії залучення, розвитку та утримання ключових спеціалістів, підтримку програм професійного зростання, просування по кар’єрі й залучення до перспективних проектів.

7. Управління проектами та командами - включає розробку та впровадження ефективних методів керування командами та проектами, координацію завдань і забезпечення чіткої комунікації між учасниками. в умовах постійного розвитку та інновацій.

1.3 Цільова аудиторія

Застосунок для управління персоналом IT-аутсорсингової компанії, досить очевидно, має використовуватися в компаніях, які надають IT послуги на

замовлення. Варто також зауважити, що маються на увазі невеликі компанії на ринку. І в середині таких компаній є досить багато категорій користувачів, які варто було б виділити окремо:

- HR-менеджери, яким потрібно весь час вести кадровий облік, управляти кадрами і вести аналітику по персоналу;
- проджект-менеджери, яким потрібно переглядати склади команд, призначати працівників на проекти, відстежувати ефективність працівників, оцінювати навички та активність членів команди;
- працівники, яким потрібно переглядати особистий профіль, оновлювати власні дані, переглядати участі у проектах;
- керівництво компанії, яким потрібно отримувати звітності (аналітика персоналу, статистика по командам), приймати стратегічні рішення, контролювати продуктивність підрозділів;
- рекрутери, основними задачами яких є ведення бази кандидатів, проведення попереднього відбору, комунікація з потенційними працівниками;
- системні адміністратори, цілями роботи яких є підтримка працездатності системи, керування правами доступу, безпека даних.

Орієнтуючись на визначену аудиторію можна зрозуміти, які вимоги до системи ця аудиторія буде висувати:

- інтуїтивно зрозумілий інтерфейс, який легко освоїти без тривалого навчання.
- безпечна авторизація, з хешуванням паролів;
- гнучка система ролей та прав доступу;
- можливість фільтрації та сортування задач за різними параметрами;
- інструменти для прийняття рішень;
- швидкодія – очікується, що система працює без затримок;
- можливість настроювати систему під специфічні завдання.

Таким чином, можна сказати, що цільова аудиторія, незалежно від розміру компанії в якій вона працює, буде бажати кастомізувати систему під певні задачі, наявних працівників і інші особливості робочого процесу.

1.4 Дослідження існуючих аналогів

Дуже популярними аналогами в даній області є: BambooHR, Zoho People, Personio. Розглянемо їх детальніше:

1.4.1 BambooHR

BambooHR - це хмарна платформа для управління персоналом (HR), розроблена для малих і середніх компаній. Вона пропонує комплексний набір інструментів для автоматизації та оптимізації HR-процесів, починаючи від найму та адаптації нових співробітників і закінчуючи управлінням ефективністю та звільненнями. Сервіс позиціонується як зручне та інтуїтивно зрозуміле рішення, що допомагає HR-менеджерам зосередитися на стратегічних завданнях, а не на рутинних адміністративних справах (див. рисунок 1.1).

Основні особливості BambooHR:

1. Централізована база даних для зберігання всієї інформації про співробітників, включаючи особисті дані, контактну інформацію, історію роботи, документи тощо.
2. Інструменти для обліку робочого часу, запитів на відпустки, лікарняних та інших видів відсутності. Автоматизація процесів затвердження та сповіщень.
3. Модуль для управління процесом найму, включаючи публікацію вакансій, відстеження кандидатів, проведення співбесід та оформлення нових співробітників. Інструменти для створення та управління програмами адаптації.
4. Функціонал для постановки цілей, проведення оцінок ефективності, надання зворотного зв'язку та відстеження прогресу співробітників.
5. Управління навчанням: Можливості для організації та відстеження навчання співробітників, призначення курсів та моніторингу їхнього проходження.
6. Генерація різноманітних звітів з HR-даних для аналізу та прийняття обґрунтованих рішень.

7. Портал для співробітників, де вони можуть самостійно керувати своїми особистими даними, запитувати відпустки, переглядати інформацію про компанію тощо.

8. BambooHR інтегрується з багатьма іншими бізнес-додатками, такими як системи обліку заробітної плати, інструменти для спільної роботи та інші.

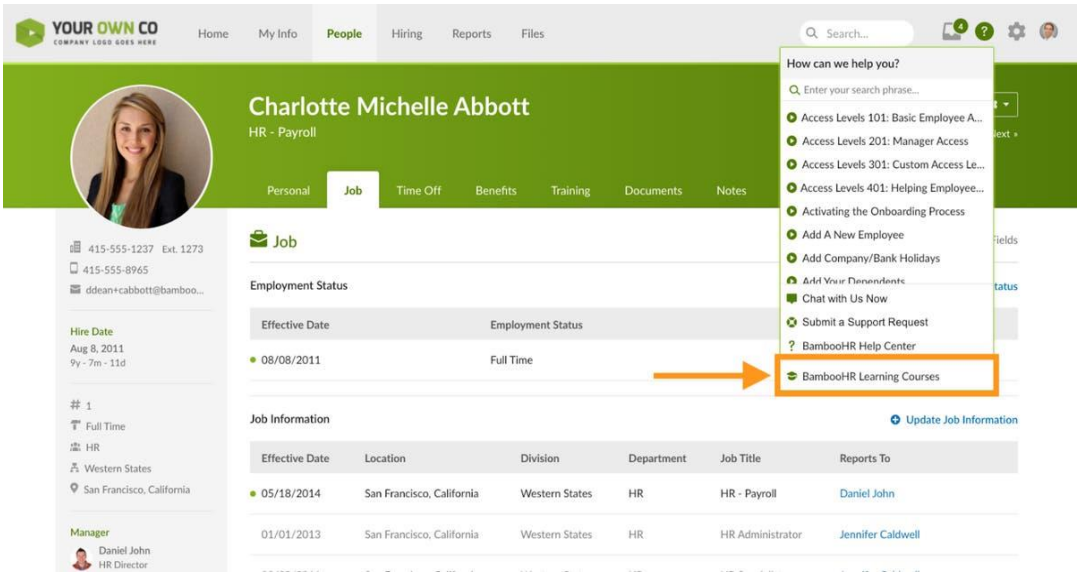


Рис. 1.1 Приклад екранної форми web-застосунку BambooHR

1.4.2 Zoho People

Zoho People - це комплексне хмарне HRMS, розроблене компанією Zoho Corporation. Воно охоплює широкий спектр HR-функцій і призначене для допомоги компаніям будь-якого розміру в управлінні своїм персоналом. Zoho People пропонує модульну структуру, що дозволяє компаніям вибирати та використовувати лише ті функції, які їм потрібні, забезпечуючи гнучкість та економічність (див. рисунок 1.2).

Основні особливості Zoho People:

1. Централізована база даних для зберігання та управління всією інформацією про співробітників, включаючи особисті дані, контактну інформацію, історію роботи, документи, фотографії тощо. Можливість створення кастомних полів та макетів.

2. Комплексна система обліку робочого часу, відстеження відпусток, лікарняних, відгулів та інших видів відсутності. Підтримка різних політик відсутності, автоматизація процесів затвердження, календарів відпусток та звітності.

3. Модуль для управління повним циклом найму, від публікації вакансій на різних платформах до відстеження кандидатів, проведення співбесід, оцінювання та прийняття на роботу. Інструменти для співпраці між членами команди найму.

4. Інструменти для постановки цілей (OKR та KPI), проведення оцінок ефективності (360-градусні оцінки, самооцінки, оцінки керівником), надання зворотного зв'язку, розробки планів розвитку та відстеження прогресу.

5. Можливості для створення, призначення та відстеження навчальних курсів, керування навчальними матеріалами, проведення вебінарів та оцінювання знань співробітників.

6. Інструменти для планування та управління графіками змінної роботи, врахування понаднормових годин та розрахунку відповідної оплати.

7. Портал для співробітників, де вони можуть оновлювати свої особисті дані, подавати заявки на відпустки, переглядати інформацію про компанію, політику, організаційну структуру, колег тощо.

8. Потужні інструменти для створення різноманітних звітів з HR-даних, візуалізації інформації за допомогою дашбордів, аналізу тенденцій та прийняття обґрунтованих рішень. Можливість створення кастомних звітів.

9. Інструмент Workflow для автоматизації рутинних HR-процесів, налаштування сповіщень, нагадувань та автоматичних дій на основі заданих правил.

10. Zoho People інтегрується з іншими продуктами Zoho (CRM, Projects, Finance тощо) та багатьма сторонніми додатками через API та готові інтеграції.

11. Доступні мобільні додатки для iOS та Android, що дозволяють співробітникам та менеджерам отримувати доступ до основних функцій системи з мобільних пристроїв.

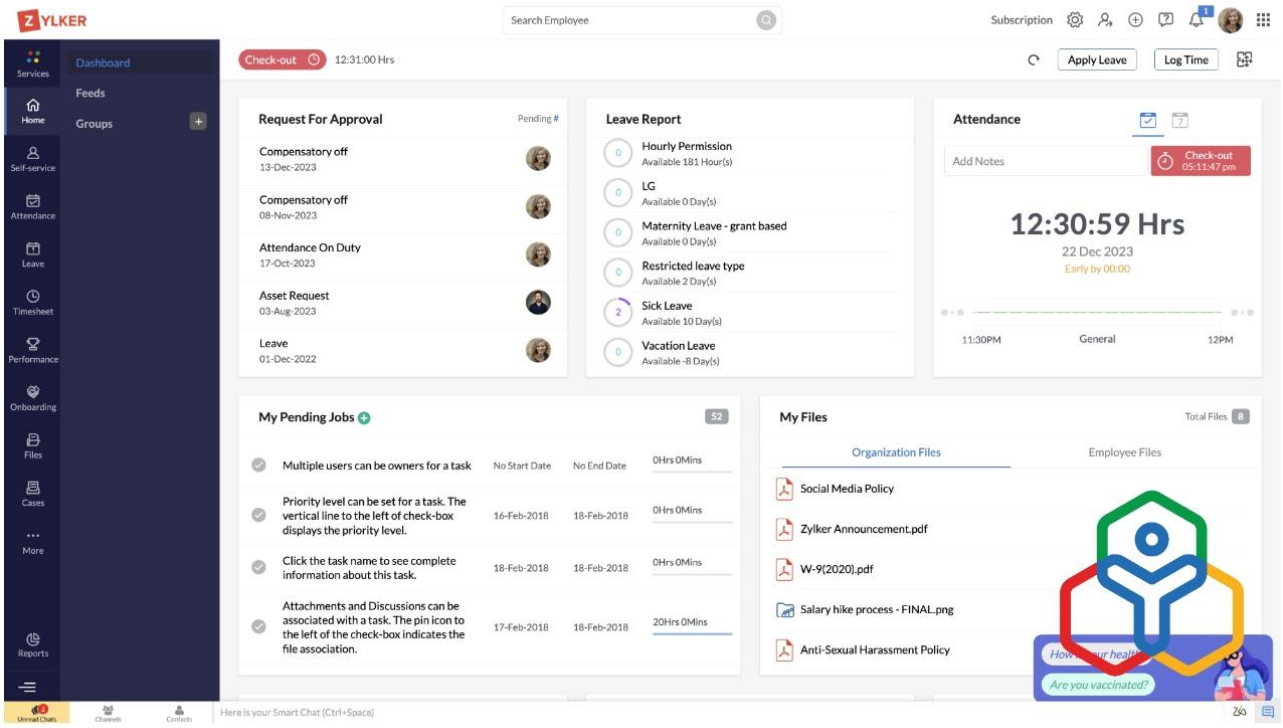


Рис. 1.2 Приклад екранної форми web-застосунку Zoho People

1.4.3 Personio

Personio - це комплексна хмарна HR-платформа, розроблена спеціально для малих і середніх підприємств (МСП). Сервіс позиціонується як універсальне рішення для управління всіма ключовими HR-процесами, починаючи від найму та онбордингу і закінчуючи управлінням ефективністю, відпустками та виплатами заробітної плати (в деяких регіонах).

Personio вирізняється фокусом на автоматизації, ефективності та створенні позитивного досвіду як для HR-менеджерів, так і для співробітників (див. рисунок 1.3).

Основні особливості Personio:

1. Централізована база даних для зберігання всієї важливої інформації про співробітників, включаючи особисті дані, контактну інформацію, історію роботи, документи, організаційну структуру тощо. Можливість налаштування полів та прав доступу.

2. Модуль для управління повним циклом найму, включаючи створення та публікацію вакансій, управління кандидатами, проведення співбесід, оцінювання, комунікацію з кандидатами та оформлення нових співробітників. Інтеграція з джерелами пошуку кандидатів.

3. Інструменти для створення структурованих програм адаптації нових співробітників, призначення завдань, обміну документами та забезпечення гладкого процесу інтеграції в компанію.

4. Функціонал для постановки цілей (OKR), проведення оцінок ефективності (цикли оцінювання, зворотний зв'язок), відстеження прогресу, документування результатів та розробки планів розвитку.

5. Можливості для планування та організації навчальних заходів, відстеження прогресу співробітників у навчанні, управління сертифікаціями та навичками.

6. Інтегрований модуль для управління виплатами заробітної плати, розрахунку податків та інших відрахувань (доступність залежить від країни).

7. Генерація різноманітних звітів з HR-даних, візуалізація інформації за допомогою дашбордів, експорт даних для подальшого аналізу. Можливість створення кастомних звітів.

8. Портал для співробітників, де вони можуть переглядати свою особисту інформацію, запитувати відпустки, переглядати новини компанії, завантажувати документи тощо.

9. Інструменти для обліку робочого часу співробітників, фіксації початку та закінчення робочого дня, відстеження понаднормових годин.

10. Personio інтегрується з багатьма іншими бізнес-додатками, такими як системи бухгалтерського обліку, інструменти для спільної роботи, комунікаційні платформи та інші.

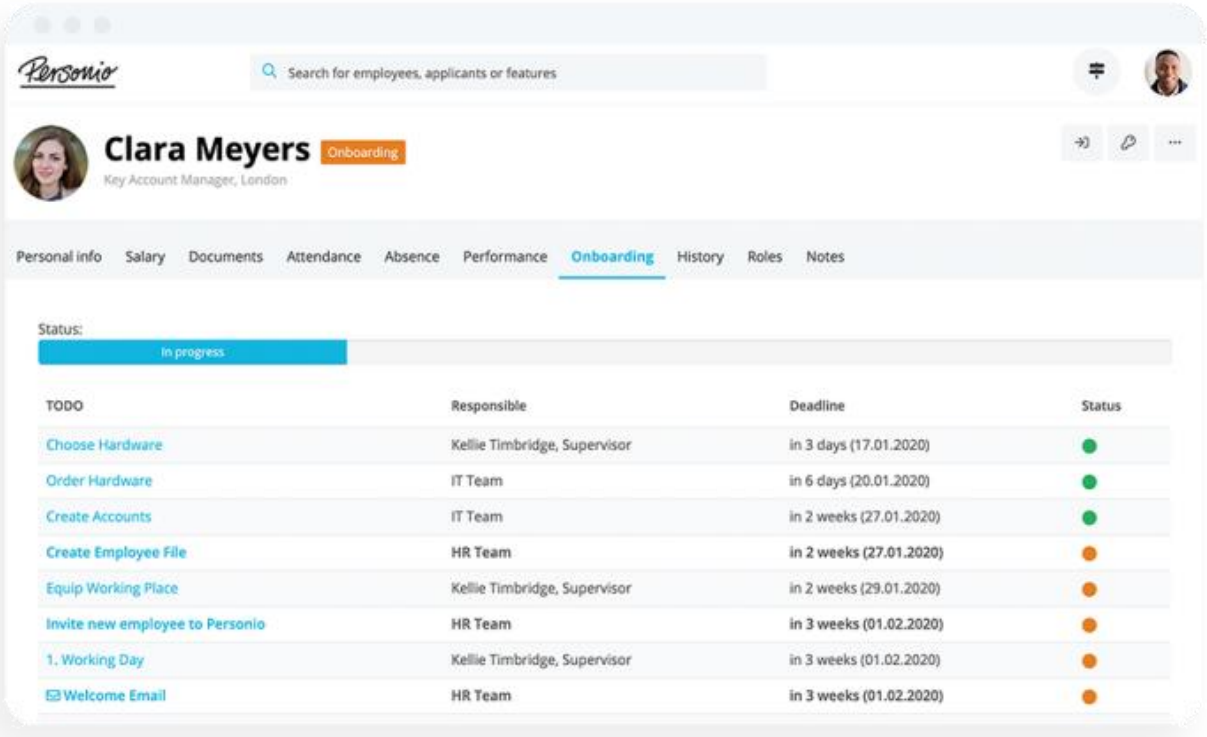


Рис. 1.3 Приклад екранної форми web-застосунку Personio

1.4.4 Порівняльна таблиця

Зведені результати аналізу характеристик розглянутих додатків наведено у таблиці 1.1:

Таблиця 1.1

Зведені результати аналізу характеристик додатків для управління персоналом IT-аутсорсингової компанії

Показник	BambooHR	Zoho People	Personio
Основний фокус	МСБ	Компанії будь-якого розміру, модульна структура	МСБ з акцентом на автоматизацію
Комплексність	Комплексний, але менш модульний, ніж Zoho	Дуже комплексний, модульна структура	Комплексний, спеціалізований для МСБ

Продовження таблиці 1.1

Зведені результати аналізу характеристик додатків для управління персоналом IT-аутсорсингової компанії

Показник	BambooHR	Zoho People	Personio
Інтерфейс	Зручний, інтуїтивно зрозумілий	Може бути складним для нових користувачів	Сучасний, інтуїтивно зрозумілий
Управління персоналом	Централізована база даних	Повна HRMS, кастомні поля та макети	Централізовані профілі, налаштування полів
Управління ефективністю	Цілі, оцінки ефективності, зворотний зв'язок	OKR/KPI, 360-градусні оцінки, плани розвитку	OKR, цикли оцінювання, зворотний зв'язок
Заробітна плата	Інтеграції з payroll-системами	Інтеграції з payroll-системами	Інтегрований модуль (доступність залежить від регіону)
Автоматизація	Автоматизація рутинних процесів	Потужний інструмент Workflow	Високий рівень автоматизації HR-процесів
Інтеграції	Багато інтеграцій з іншими сервісами	Широка екосистема Zoho та сторонні інтеграції	Багато інтеграцій з іншими сервісами
Мобільні додатки	Є	Є (iOS та Android)	Є (iOS та Android)
Спеціалізація	Загальне HR-рішення для МСБ	Гнучка система для компаній будь-якого розміру	HR-платформа для МСБ з акцентом на автоматизацію
Переваги	Зручність використання, фокус на МСБ	Комплексність, модульність, інтеграція з Zoho	Автоматизація, зручність, орієнтація на досвід співробітників
Недоліки	Обмеженість кастомізації для складних потреб	Складний інтерфейс для новачків, низька якість підтримки	Можлива обмеженість для дуже специфічних потреб, доступність Payroll залежить від регіону

На основі цієї таблиці можна сказати, що перевагами цих систем є:

- орієнтація на автоматизацію процесів HR;
- інтуїтивний та сучасний інтерфейс;
- наявність інтеграції з багатьма сервісами.

Недоліки:

- обмежена кастомізація для складних HR-потреб;
- може не відповідати специфічним або нестандартним потребам певних компаній;
- не адаптовані під специфіку невеликих аутсорсингових компаній;
- мають надлишковий функціонал або складні у впровадженні;
- коштують дорого для стартапів і малого бізнесу;
- не мають алгоритмів для автоматичного підбору персоналу під проекти.

1.5 Визначення вимог до застосунку

На основі всіх факторів, починаючи від цільової аудиторії і закінчуючи наявними конкурентами, були створені вимоги до застосунку. Ці вимоги:

Функціональні:

1. Реєстрація та авторизація в системі за допомогою логіну та паролю.
2. Призначення користувачу однієї з ролей(Team Leader, Developer, QA, Admin).
3. Можливість управління профілями користувачів (продивлятися акаунти, оновлювати дані, призначати ролі, видаляти акаунти).
4. Можливість управління проектами(створення, редагування, видалення, призначення співробітників).
5. Можливість управління задачами(створення, редагування, видалення, призначення виконавців, написання коментарів)
6. Відображення списоку проектів та задач користувача.
7. Фільтрація списку задач за назвою, статусом і пріорітетом.
8. Реалізація алгоритму, який автоматично підбирає оптимальних співробітників за параметрами кваліфікації та ролі.

Нефункціональні:

1. Доступ до даних і можливостей обмежено згідно ролі користувача.
2. Запити повинні виконуватися не більш ніж за 5 секунд.
3. Система повина виконувати хешування паролів.

2 ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Мова Програмування

Мова програмування - штучно створена система, призначена для формулювання інструкцій, які виконують машини, зокрема комп’ютери. Основне її призначення - створення програм, що керують діями пристроїв, а також опис алгоритмів розв’язання різноманітних задач.

З технічної точки зору, мова програмування є штучною формальною системою позначень, яка дозволяє описувати алгоритми та структури даних. Її функціонування базується на чітко визначених лексичних, синтаксичних і семантичних правилах, які визначають як зовнішній вигляд програмного коду, так і поведінку комп’ютера під час його виконання.

2.1.1 Java

Java - високорівнева мова програмування, орієнтована на об’єктно-орієнтований підхід і здатна підтримувати імперативну парадигму. Вона була розроблена Джеймсом Гослінгом у 1995 році в компанії Sun Microsystems, яка згодом стала частиною корпорації Oracle. Java здобула популярність завдяки своїй стабільності, безпеці та платформонезалежності [5-8].

Однією з визначальних рис Java є принцип "Write Once, Run Anywhere" (WORA) - це означає, що програму, написану на Java, можна запускати на будь-якому пристрої з JVM. Така властивість робить Java надзвичайно універсальною та придатною для кросплатформної розробки.

Java широко застосовується в різних сферах: від створення корпоративного програмного забезпечення, мобільних застосунків (особливо на платформі Android) до web-розробки, серверних рішень та вбудованих систем. Вона часто використовується в банківській справі, телекомунікаціях, фінансових технологіях та у великих розподілених системах.

Завдяки великій екосистемі бібліотек та фреймворків, таких як Spring, Hibernate та JavaFX, Java забезпечує потужну інфраструктуру для розробки складних програмних рішень. Вона також має добре розвинене середовище для тестування, налагодження і моніторингу додатків.

Переваги Java:

- платформонезалежність через JVM;
- висока продуктивність і стабільність при тривалому використанні;
- широка екосистема бібліотек, інструментів і фреймворків;
- безпека - ретельно контролюється виконання коду;
- масштабованість - підходить як для невеликих додатків, так і для великих систем.

Недоліки Java:

- вища складність синтаксису, порівняно з більш простими мовами (наприклад, Python);
- більше споживання пам’яті, особливо при запуску в JVM;
- повільніший старт додатків через необхідність запуску JVM;
- застарілість частини стандартної бібліотеки, порівняно з сучасними альтернативами в інших мовах.

Обмеження Java:

- не призначена для низькорівневого програмування (наприклад, розробки драйверів чи ОС);
- залежність від JVM, що може створювати несумісності між версіями;
- обмежена ефективність у сценаріях, де критична максимальна швидкість виконання (у порівнянні з C/C++).

2.1.2 JavaScript

JavaScript - це високорівнева, інтерпретована мова програмування, яка була створена у 1995 році Бренданом Айком для браузера Netscape. Спочатку вона призначалася для додавання інтерактивності до web-сторінок, однак з часом

еволюціонувала у потужний інструмент для розробки як клієнтських, так і серверних застосунків [9].

JavaScript підтримує імперативний, об’єктно-орієнтований та функціональний стилі програмування. Одна з її ключових переваг - можливість виконання коду безпосередньо у браузері, що дозволяє створювати динамічні інтерфейси без потреби у зворотньому зв’язку з сервером.

З появою Node.js JavaScript отримав потужність і на серверній стороні, ставши основою для повностекової розробки. Сьогодні JavaScript є однією з найпоширеніших мов у світі: він використовується у фронтенді (завдяки фреймворкам React, Vue, Angular), у бекенді (через Node.js, Express), а також для створення мобільних застосунків (React Native), настільних програм (Electron) і навіть для пристроїв IoT та автоматизації.

Мову стандартизує організація ECMA International, а її сучасна версія відома під назвою ECMAScript. JavaScript має динамічну типізацію, що робить її гнучкою й зручною для швидкої розробки, проте це також може ускладнювати масштабування та підтримку великих проектів. Щоб подолати ці обмеження, була створена надбудова TypeScript, яка додає підтримку статичної типізації.

Переваги JavaScript:

- безпосереднє виконання у браузері без додаткових інструментів;
- підходить для повного стеку (frontend і backend);
- швидкість розробки завдяки гнучкому синтаксису та величезній кількості готових бібліотек (npm);
- потужна спільнота та безліч фреймворків (React, Vue, Angular);
- адаптивність — використовується в мобільних, десктопних, серверних і web-додатках.

Недоліки JavaScript:

- динамічна типізація може призводити до важковловимих помилок;
- несумісність між браузерами (особливо у старих версіях);
- небезпека XSS-атак, якщо не дотримуватись безпечного програмування;

- складність підтримки великих протктів без додаткових інструментів.

Обмеження JavaScript:

- обмежений доступ до файлової системи та апаратного забезпечення у браузері;
- не оптимізований для складних обчислень (немає роботи з потоками на рівні ядра);
- потреба в середовищі виконання (браузер або Node.js), залежно від контексту.

2.2 Середовище розробки

Інтегроване середовище розробки, або IDE - комплексне програмне рішення для розробки програмного забезпечення. Зазвичай, складається з редактора початкового коду, інструментів для автоматизації складання та відлагодження програм. Більшість сучасних середовищ розробки мають можливість автодоповнення коду.

Деякі середовища розробки містять компілятор, інтерпретатор або ж обидва (наприклад NetBeans та Eclipse), інші не містять жодного з них (SharpDevelop та Lazarus). Деякі інтегровані середовища розробки містять систему керування версіями або інструменти для полегшення розробки графічного інтерфейсу користувача (XCode, Embarcadero Delphi). Багато сучасних ICP містять інспектор класів, інспектор об'єктів, схему ієрархії класів для полегшення об'єктно-орієнтованої розробки програмного забезпечення.

IntelliJ IDEA - це потужне IDE, розроблене компанією JetBrains, спеціально призначене для професійної розробки на мові Java, але також підтримує безліч інших мов та технологій. IntelliJ IDEA вважається одним із найкращих середовищ для Java-розробників завдяки своїй гнучкості, розумному автодоповненню, глибокому аналізу коду та широкій інтеграції з сучасними інструментами розробки [10].

IntelliJ IDEA повністю підтримує розробку на Java SE, Java EE, Spring, Kotlin, Groovy, а також забезпечує інтеграцію з популярними фреймворками, такими як

Spring Boot, Hibernate, JavaFX, Micronaut та багатьма іншими. Середовище здатне автоматично розпізнавати структуру проекту, налаштовувати SDK, інтерпретатори, збірки та тести, створюючи зручне й ефективне робоче середовище.

Крім Java, IntelliJ IDEA інтегрується з мовами HTML, CSS, JavaScript, а також підтримує TypeScript, SQL, Python (з відповідними плагінами). Це дозволяє розробникам працювати над повноцінними web- та корпоративними застосунками в одному середовищі, без потреби перемикатися між різними інструментами.

IntelliJ IDEA також пропонує потужні засоби для відлагодження, рефакторингу, аналізу продуктивності, версійного контролю (Git, SVN, Mercurial), а також повну інтеграцію з CI/CD-сервісами. Це дозволяє розробникам ефективно відслідковувати помилки, оптимізувати продуктивність застосунків і підтримувати високу якість коду.

Завдяки розумним підказкам, пошуку по проекту, глибокому контекстному аналізу, IntelliJ IDEA значно підвищує продуктивність розробника і підходить як для індивідуальної, так і для командної роботи на великих проектах.

2.3 Фреймворки

2.3.1 Spring Web

Spring Web - модуль фреймворку Spring, призначений для створення web-застосунків та RESTful web-сервісів на мові Java. Він є частиною екосистеми Spring Framework і забезпечує зручні засоби для розробки серверної логіки, обробки HTTP-запитів і організації архітектури web-застосунків на основі шаблону MVC [11-12].

Spring Web дозволяє будувати гнучкі та масштабовані web-застосунки завдяки анотаційному підходу, де основні компоненти, як-от контролери (@Controller, @RestController) та маршрути (@RequestMapping), легко налаштовуються. Розробники можуть ефективно обробляти GET, POST, PUT, DELETE запити, формувати відповіді у форматах JSON або XML.

2.3.2 Spring Security

Spring Security - це потужний модуль фреймворку Spring, призначений для забезпечення аутентифікації та авторизації в Java-додатках. Він надає гнучкий і розширюваний механізм захисту web-застосунків, REST API, мікросервісів та інтеграції з різними системамим безпеки [13-14].

Spring Security дозволяє реалізувати перевірку користувачів за допомогою форм, токенів (наприклад, JWT), базової автентифікації, OAuth2, LDAP та інших методів. Авторизація може бути реалізована на основі ролей, прав доступу або політик, з використанням анотацій (@PreAuthorize, @Secured) чи конфігураційного Java-коду.

2.3.3 Spring Data JPA

Spring Data JPA - це підпроект Spring Data, який спрощує реалізацію рівня доступу до даних (DAO), використовуючи Java Persistence API (JPA) [15]. Його мета - зменшити обсяг шаблонного коду, необхідного для реалізації репозиторіїв та роботи з базами даних.

Цей фреймворк автоматично реалізовує базові CRUD-операції на основі назв методів у інтерфейсах, використовуючи ORM-технології, зокрема Hibernate.

2.3.4 JUnit and Mockito

JUnit - це фреймворк для модульного тестування в Java. Він дозволяє розробникам писати й виконувати тести для перевірки правильності роботи окремих частин коду (зазвичай методів класів) [16-17]. JUnit підтримує анотації (наприклад, @Test, @BeforeEach, @AfterEach), що робить написання тестів зручним і структурованим.

Mockito - це бібліотека для створення mock-об’єктів (імітацій залежностей) у тестах. Вона дозволяє підмінювати реальні об’єкти фіктивними, щоб ізолювати поведінку тестованого коду та перевірити виклики методів, значення параметрів тощо [18]. Використовується разом із JUnit для написання ефективних unit-тестів.

Разом JUnit і Mockito дозволяють створювати надійні, ізольовані та керовані тести в Java-проектах.

2.4 Бібліотеки та інші інструменти

2.4.1 Spring Freemarker

Spring Freemarker - це інтеграція шаблонізатора FreeMarker у фреймворк Spring, яка дозволяє створювати динамічні HTML-сторінки, генеруючи їх на стороні сервера [19].

FreeMarker - це потужний шаблонізатор, який відокремлює логіку від представлення, дозволяючи передавати Java-об’єкти до шаблонів і будувати HTML динамічно.

2.4.2 MySQL Connector

MySQL Connector/J - офіційний JDBC-драйвер, розроблений для забезпечення взаємодії Java-застосунків із базою даних MySQL. Він підтримує повний набір SQL-операцій, API JDBC, роботу з транзакціями, а також інтегрується з ORM-фреймворками, такими як Hibernate.

Серед основних переваг цього драйвера можна відзначити: стабільність і надійність, адже він розроблений безпосередньо компанією Oracle; повну відповідність специфікації JDBC 4.2 і вище; хорошу інтеграцію з ORM-системами та бібліотеками на кшталт Spring Data; а також підтримку SSL-з'єднань, плагінів автентифікації та механізмів керування підключеннями (connection pooling).

Водночас є й певні обмеження: драйвер прив'язаний виключно до СУБД MySQL, тобто не є універсальним рішенням, а також його робота може залежати від конкретної версії бази даних і параметрів налаштування сервера.

2.4.3 Lombok

Project Lombok - бібліотека для Java, яка автоматизує створення стандартного коду, такого як геттери, сеттери, конструктори, методи toString, equals, hashCode та

інші, використовуючи спеціальні анотації [20]. Завдяки цьому значно скорочується кількість шаблонного коду, що покращує читабельність класів моделі, наприклад DTO або Entity.

Серед переваг бібліотеки: вона дозволяє мінімізувати рутину при написанні класів, підтримує анотації на кшталт @Getter, @Setter, @Data, @Builder, @AllArgsConstructor, @NoArgsConstructor, а також добре працює з такими фреймворками, як Spring і більшістю Java-проектів.

Втім, є і певні обмеження: робота бібліотеки часто вимагає встановлення додаткового плагіна в середовище розробки, наприклад, в IntelliJ IDEA; вона може ускладнити процес налагодження (debugging) і автоматичне створення документації, а також іноді приховує критичні деталі реалізації, що може ускладнити розуміння коду.

2.5 Система управління базою даних

MySQL - популярна система управління реляційними базами даних (СУРБД) з відкритим вихідним кодом, яка використовується для зберігання, обробки та управління даними в численних програмних застосунках. Вона була розроблена шведською компанією MySQL AB у 1995 році й згодом придбана корпорацією Oracle.

MySQL є однією з найпоширеніших баз даних у світі завдяки своїй високій продуктивності, масштабованості, простоті використання та сумісності з багатьма мовами програмування, зокрема з Java [21], Python, PHP, C/C++ та іншими. Найчастіше MySQL використовується у web-розробці, електронній комерції, аналітичних системах, а також у корпоративних та хмарних застосунках.

СУБД MySQL повністю підтримує мову SQL (Structured Query Language) для створення, модифікації та запиту структурованих даних. Вона дозволяє працювати з таблицями, представленнями, індексами, транзакціями, тригерами, збереженими процедурами, а також має вбудовані засоби для реплікації та резервного копіювання.

2.6 Інструменти проектування інтерфейсу користувача

2.6.1 HTML

HTML - мова розмітки гіпертексту, яка є основою структури web-сторінок в Інтернеті. Вона використовується для опису змісту web-сторінки, включаючи текст, заголовки, посилання, зображення, таблиці, форми та інші елементи. HTML був створений британським науковцем Тімом Бернерс-Лі у 1991 році як частина концепції Всесвітньої павутини[22].

HTML визначає структуру документа за допомогою тегів, які вказують браузеру, як відображати певний контент.

Поточна актуальна версія - HTML5, яка включає підтримку мультимедіа (відео, аудіо), графіки (Canvas, SVG), офлайн-сховища, а також семантичних тегів для покращення доступності та SEO.

2.6.2 CSS

CSS - каскадні таблиці стилів, які використовуються для оформлення зовнішнього вигляду HTML-документів. CSS визначає, як мають виглядати HTML-елементи на екрані, папері або інших носіях - включаючи кольори, шрифти, розміри, відступи, розміщення, анімації тощо.

Створений у 1996 році, CSS відокремлює структуру від дизайну, що дозволяє розробникам змінювати стиль без зміни HTML-коду. Остання версія CSS3 - містить модулі для анімації, гнучкої верстки (Flexbox, Grid), медіа-запитів (адаптивний дизайн) та інших потужних можливостей.

3 ПРОЕКТУВАННЯ ТА РОЗРОБКА WEB-ЗАСТОСУНКУ

3.1 Проектування архітектури застосунку

Базуючись на визначених вимогах до застосунку, була створена діаграма варіантів використання, показана на рисунку 3.1, в якій включено розділення на різні ролі в системі.

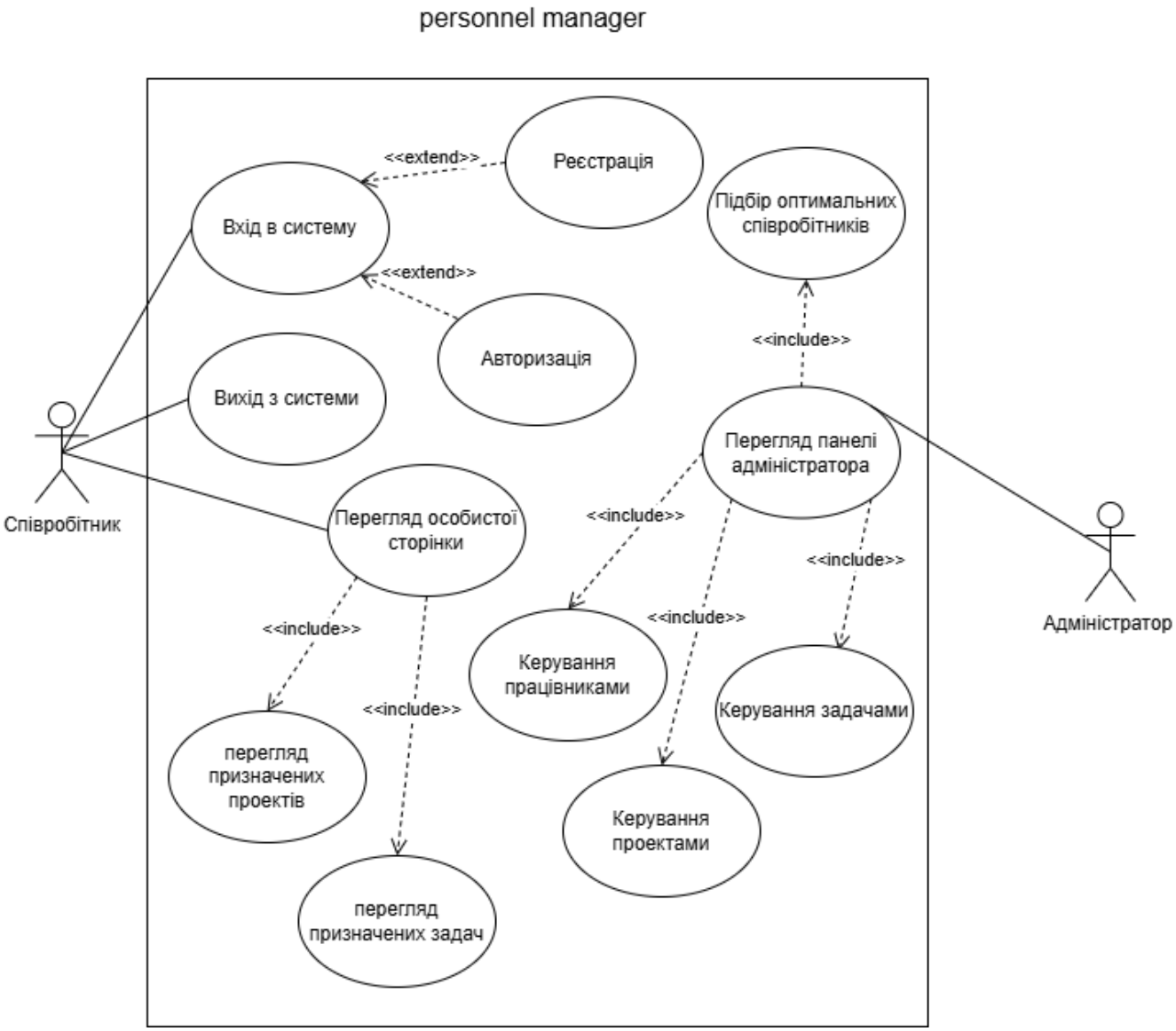


Рис. 3.1 Діаграма варіантів використання

Базуючись на цій діаграмі можна сказати, що основними сутностями, з якими буде працювати система – проекти та задачі. Ці проекти і задачі прив’язані до профілів співробітників, доступ до яких відкривається лише після авторизації в системі. На основі цієї інформації варто спроектувати базу даних для розроблюваної системи.

3.2 Архітектура бази даних

Базуючись на вже визначених сутностях і вимогах до застосунку можна створити основні таблиці і визначити назви та типи даних для колонок в цих таблицях. Не менш важливою частиною проектування є встановлення зв’язків між цими таблицями, для коректної роботи проекту. Логічно визначити, що в проекту може бути багато учасників, в той час як в одного співробітника може бути багато проектів, в яких він приймає участь. В одного проекту може бути багато призначених завдань, але одне завдання може мати лише один проект. В співробітника може бути багато призначених задач, в той час як в задачі може бути лише один виконавець. Такі зв’язки обумовлені специфікацію проекту і орієнтованістю на малі компанії та стартапи, в яких є лише невелика кількість співробітників і задачі доводиться часто виконувати поодинці.

Визначивши всі ці деталі створено даталогічну модель БД, представлену на рисунку 3.2.

Встановлені зв‘язки:

- проект - співробітник: багато до багатьох;
- проект - задача: один-до-багатьох;
- співробітник - задача: один-до-багатьох.

Тепер можна переходити до проектування серверної частини застосунку.

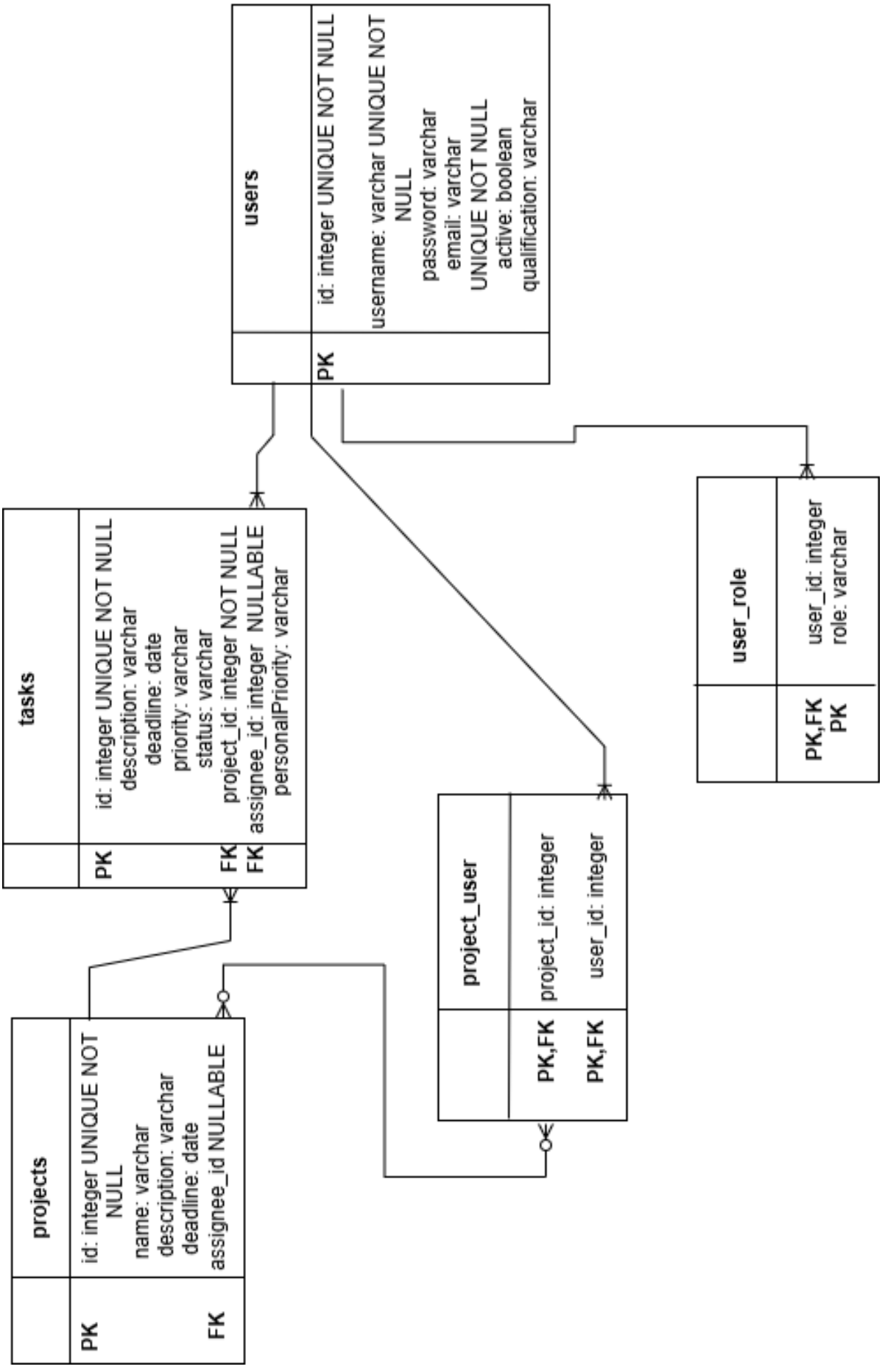


Рис. 3.2 Даталогічна модель БД

3.3Архітектура серверної частини

Для створення web-застосунку було прийнято рішення створити структуру серверної частини застосунку за класичною багатошаровою архітектурою Spring Boot[23-24]. Розглянемо кожен шар детально:

- модель даних (Model Layer): Цей шар містить сутності (entity-класи), що представляють структуру даних, які зберігаються в базі даних. Кожна сутність відображає таблицю в базі та анотована за допомогою JPA (наприклад, @Entity). Тут також можуть міститися зв’язки між сутностями (@OneToMany, @ManyToOne тощо);
- шар репозиторіїв (Repository Layer): Цей шар відповідає за доступ до бази даних. Він включає інтерфейси, що наслідують JpaRepository або CrudRepository, забезпечуючи базові CRUD-операції без необхідності писати SQL-код. За потреби, можна визначати користувацькі запити з використанням JPQL або нативного SQL;
- сервісний шар (Service Layer): Сервісний шар реалізує бізнес-логіку застосунку. Він взаємодіє з репозиторіями для отримання чи збереження даних, обробляє ці дані, виконує обчислення, перевірки тощо. Це дозволяє відокремити логіку від контролерів і зробити застосунок більш підтримуваним і масштабованим;
- контролери (Controller Layer): Цей шар відповідає за обробку HTTP-запитів від клієнта. Контролери приймають запити, викликають відповідні сервіси, формують відповіді та відправляють їх назад. Вони анотовані як @RestController або @Controller і працюють з маршрутами, заданими через @RequestMapping, @GetMapping, @PostMapping тощо;
- конфігурація (Configuration Layer): Цей шар включає конфігураційні класи та файли, що задають параметри роботи застосунку. Сюди входять конфігурації безпеки (Spring Security), взаємодії з базою даних, сторонні інтеграції, а також кастомні @Configuration класи для налаштування бінів, властивостей та поведінки застосунку.

Застосування класичної багатошарової архітектури у Spring Boot має низку суттєвих переваг, які роблять її ефективним, гнучким і масштабованим підходом до розробки застосунків.

Передусім, чітке розмежування функцій між окремими шарами дозволяє кожному з них виконувати свою визначену роль. Така структура сприяє логічній організації коду: модельний шар відповідає за опис даних, репозиторії — за їхнє зберігання та доступ, сервісний шар — за реалізацію бізнес-логіки, а контролери - за взаємодію з клієнтом. Це значно полегшує сприйняття коду як новим розробникам, так і всій команді.

Окрім цього, багатошарова структура сприяє повторному використанню компонентів. Наприклад, методи сервісного рівня можуть використовуватись у різних контролерах або інших сервісах без дублювання логіки, що зменшує ризик помилок і надмірного коду.

Ще одна суттєва перевага — спрощене обслуговування та підтримка. Завдяки чіткій структурі легше виявляти та виправляти помилки, вносити зміни до окремих частин системи чи оновлювати логіку без впливу на інші модулі.

Також архітектура підвищує тестованість застосунку. Завдяки ізоляції кожного шару з'являється можливість окремого тестування, зокрема unit- та інтеграційними тестами. Це дозволяє, наприклад, перевірити функціональність сервісів незалежно від контролерів чи бази даних із використанням моків або стабів.

Крім того, структура легко масштабується. Розширення функціональності не порушує роботу наявних компонентів, що спрощує інтеграцію з новими модулями чи зовнішніми сервісами.

У цілому, багатошарова архітектура в Spring Boot створює міцний фундамент для побудови сучасних web-застосунків, що є зрозумілими, масштабованими і підтримуваними.

Також для розбиття співробітників за ролями і фільтрації задач за пріорітетом та статусом потрібно буде створити спеціальні сутності для перерахування. В таблиці 3.1 будуть перераховані ці значення:

Таблиця 3.1

Таблиця значень, записаних в enums

Значення	Enum	Значення
LOW, MEDIUM, HIGH, CRITICAL	Priority	Визначає пріорітет певної задачі.
ROLE_ADMIN	Role	Визначає роль співробітника, яка надає розширені повноваження в системі.
ROLE_TEAM_LEADER, ROLE_DEVELOPER, ROLE_QA	Role	Визначає роль співробітника.
TODO, IN_PROGRESS, REVIEW, DONE	TaskStatus	Визначає статус певної задачі.

Базуючись на розробленій даталогічні моделі була створена діаграма класів для моделей в системі, показана на рисунку 3.3, де можна побачити, що багато enum-ів пов’язані з іншими класами зв’язком агрегації, оскільки ці енами є лише частинами відповідних класів і не є самостійними частинами програми.

Між користувачами і задачами, а також між користувачами і проектами є зв’язок асоціації, що дозволяє обмінюватися даними між класами.

Задача ж пов’язана з проектом композицією, оскільки в системі визначено, що задача не буде існувати без прив’язки до проекту і в випадку видалення проекту, всі його задачі також видаляються. Таким чином, можна сказати що задача дуже сильно пов’язана з проектом і відношення композиції є правильним вибором.

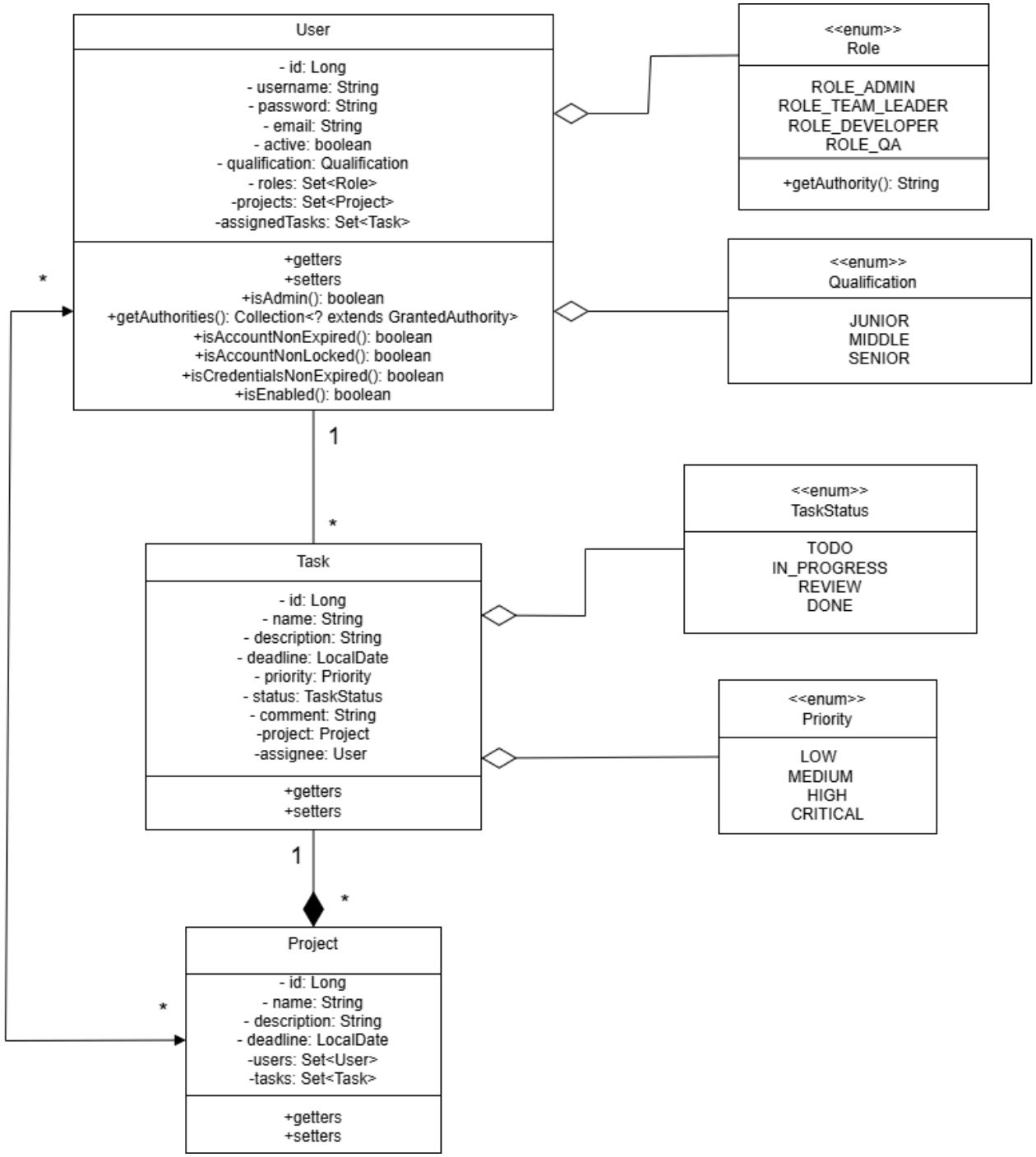


Рис. 3.3 Діаграма класів моделей

Тепер потрібно спроектувати шар репозиторіїв, шар сервісів, шар контролерів та зв’язки між ними. Для цього була створена наступна діаграма класів, показана на рисунку 3.4:

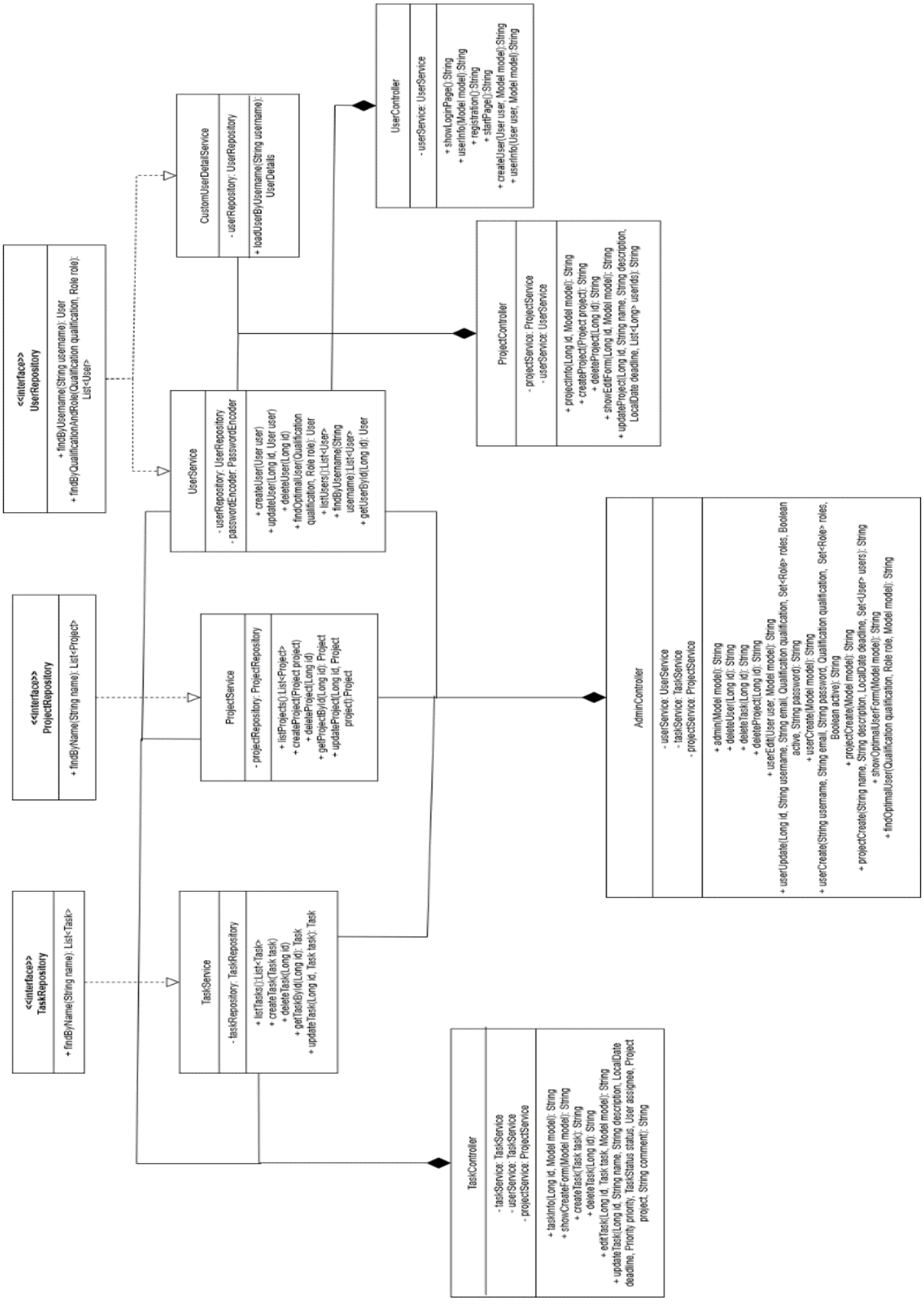


Рис. 3.4 Діаграма класів сервісів і контролерів

Репозиторії створені для зберігання даних та мають методи для знаходження відповідних сутностей за їх назвами, ці методи не реалізовані, як і потрібно робити в репозиторіях.

Всі сервіси наслідуються від відповідних репозиторіїв і повинні мати всі необхідні методи для управління своїх сутностей.

Контролери використовують функціонал сервісів і встановлює зв’язок з клієнтською частиною web-застосунку.

Також варто зауважити, що при проектуванні було прийнято рішення створити AdminController, який буде відповідати за управління проектами, задачами і профілями співробітниками.

CustomUserDetailService буде відігравати ключову роль в системі автентифікації Spring Security. Він відповідає за завантаження користувацьких даних під час автентифікації і є мостом між базою даних та системою безпеки Spring.

Архітектуру розроблюваного web-застосунку можна побачити на рисунку 3.5.

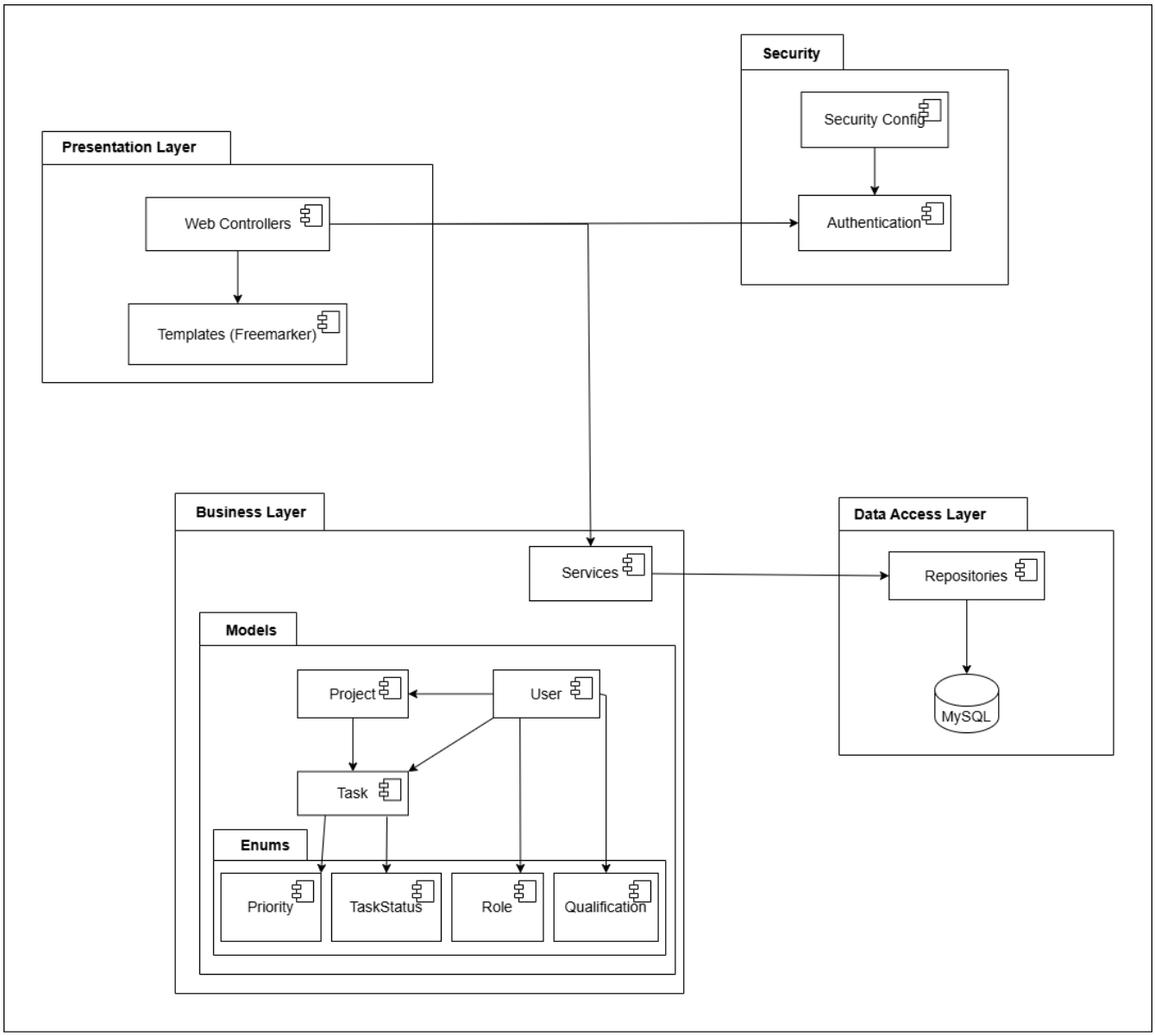


Рис. 3.5 Архітектура застосунку

3.4 Архітектура клієнтської частини

Проектування клієнтської частини буде визначатися розробленим функціоналом серверної частини.

Користувачі починають роботу з додатком на Start page, звідки можуть перейти до Login page (для входу) або Registration page (для створення облікового запису). Після успішної авторизації, користувачі з певними правами доступу повинні бути перенаправлені на User info page.

Admin page є центральним вузлом для адміністраторів, надаючи їм можливість управляти проектами, задачами та профілями користувачів. Звідси можна створювати (Project create page, Task create page, User create page), редагувати (Project edit page, Task edit page, User edit page) та переглядати інформацію (Project info page, Task info page, User info page) про відповідні об'єкти.

Також існує User optimal page для підбору працівників за ролями та кваліфікацією. Project info page та Task info page дозволяють переглядати деталі та інтегровані зі сторінками редагування та профілями користувачів.

Згідно створеної архітектури був створений перелік всіх сторінок у web-застосунку, показаний у таблиці 3.2.

Таблиця 3.2

Таблиця переліку всіх наявних сторінок

Назва сторінки	Функціонал	Пов’язана з
Start page	Можливість перейти до реєстрації або авторизації	Login page і Registration page
Login page	Можливість авторизуватися в системі	Start page і User info page
Registration page	Можливість зареєструватися в системі	Start page і Login page
Admin page	Можливість управляти проектами, задачами та профілями користувачів.	Task create page , Task info page, Task edit page, Project create page, Project info page, Project edit page, User create page, User edit page, User info page, User optimal page.

Продовження таблиці 3.2

Таблиця переліку всіх наявних сторінок

Project create page	Можливість створення проекту	Admin page
Project edit page	Редагування деталей проекту	Admin page
Project info page	Перегляд деталей проекту	Project edit page, User info page, Admin page
Task create page	Можливість створення задачі	Admin page
Task edit page	Редагування деталей задачі	Admin page
Task info page	Перегляд деталей задачі	Task edit page, User info page, Admin page
User create page	Можливість створення профілю співробітника	Admin page
User edit page	Редагування деталей профілю \співробітника	Admin page
User info page	Перегляд проектів, задач та профілю користувача.	Admin page, Login page, Task info page, Task edit page, Project info page, Project edit page.
User optimal page	Підбір працівників базуючись на їх ролі та кваліфікації	Admin page

3.5 Розробка серверної частини застосунку

Спочатку, були створені Enum-и для переліку всіх необхідних значень (див. рисунок 3.6):

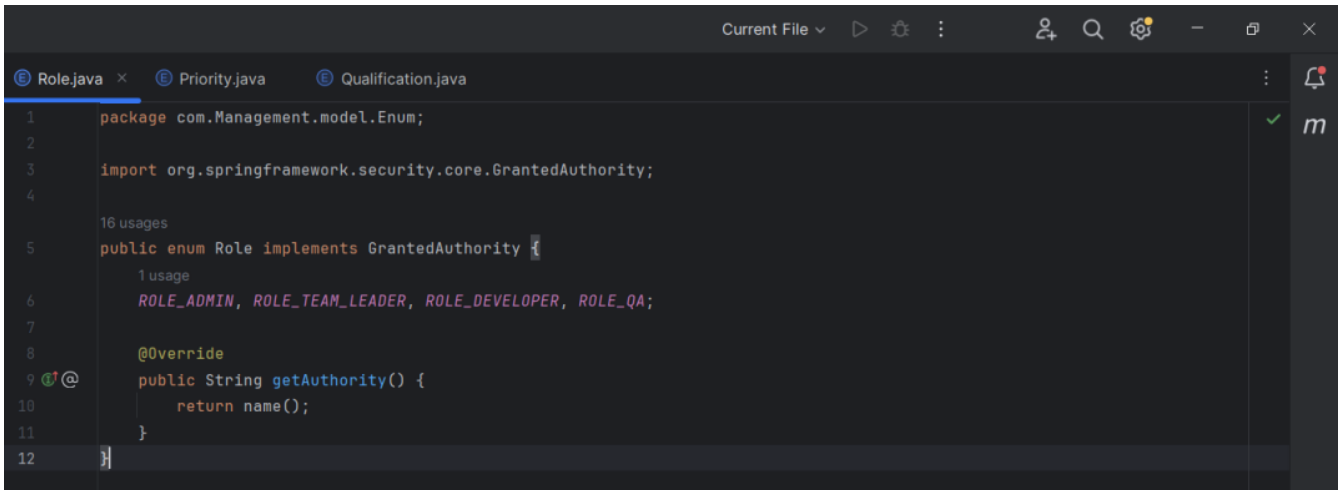


Рис. 3.6 Створення enum-ів

В наведеному прикладі – Enum Role також реалізує інтерфейс GrantedAuthority, для подальшої реалізації авторизації і налаштування безпеки в застосунку.

Після чого створюються моделі, відразу з анотаціями Data Jpa, для створення таблиць в MySQL. Також використовуються анотації Lombok для зменшення необхідного для написання коду (див. рисунок 3.7).

Рис. 3.7 Створення моделей

В моделі користувача є реалізація UserDetails, для подальшої авторизації в системі і налаштування безпеки (див. рисунок 3.8). Також був перевизначений метод toString(), для кращого виведення інформації про користувача.

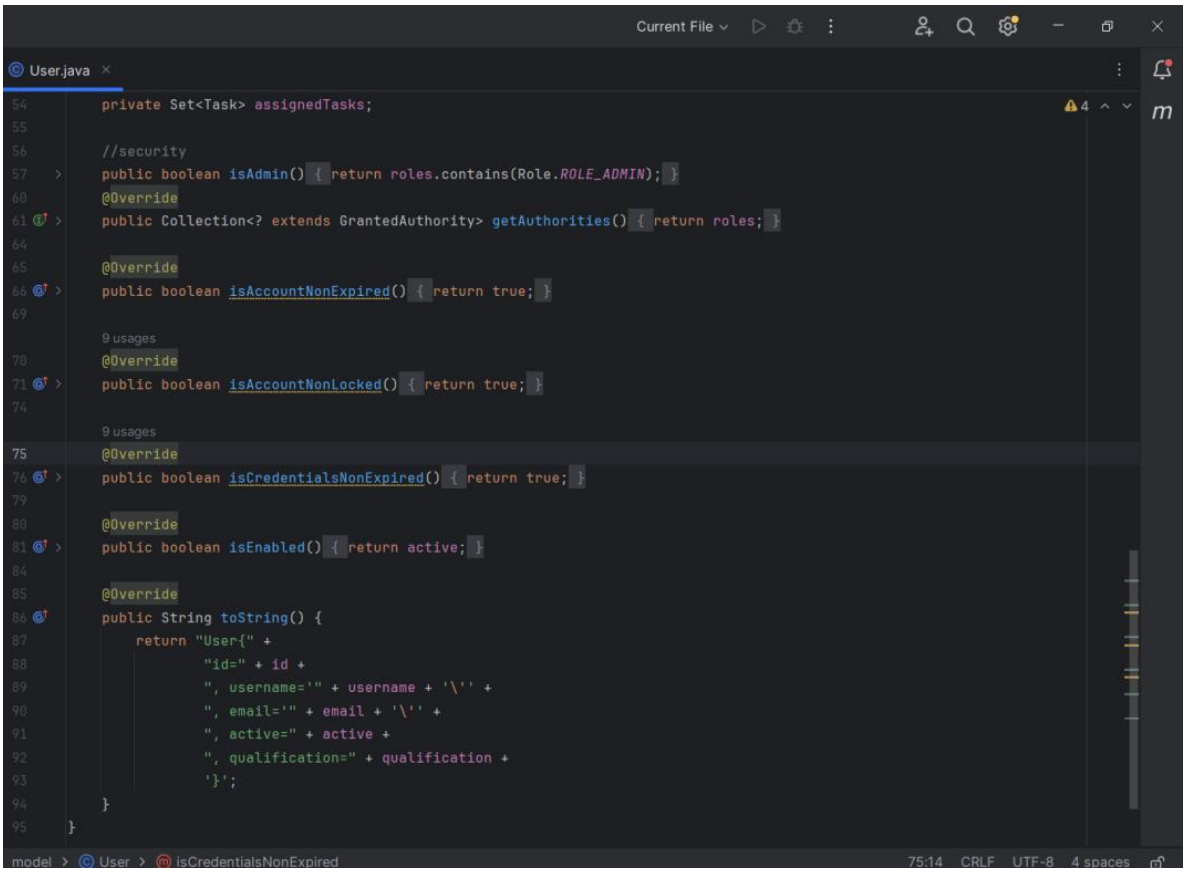


Рис. 3.8 Реалізація методів UserDetails в User

Далі були створені репозиторії для зберігання інформації. В них ініціалізовані методи findByName(String name); Для UserRepository також був створений метод findByQualificationAndRole, для подільшої реалізації алгоритму для підбору співробітників за їх кваліфікацією та роллю (див. рисунок 3.9).

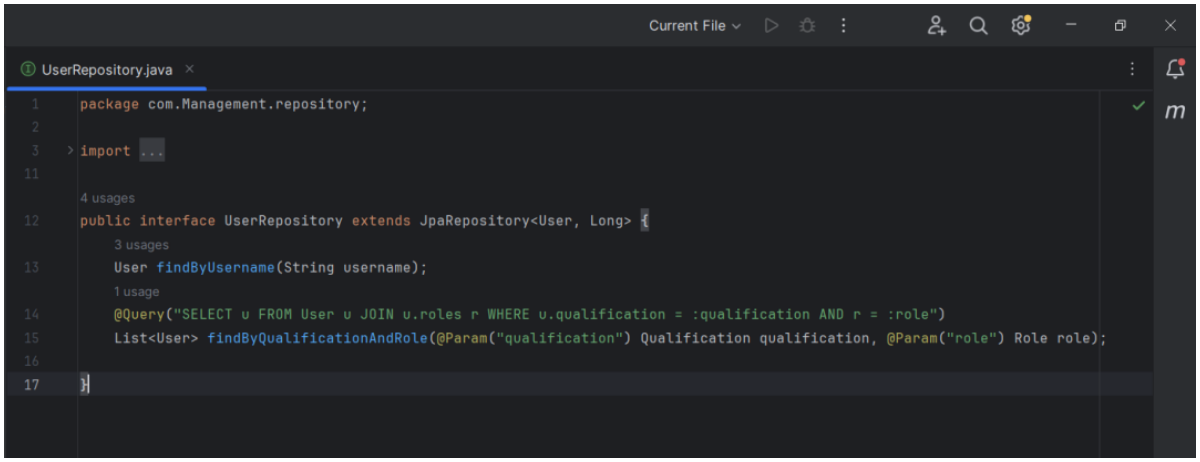


Рис. 3.9 Створення репозиторіїв

Далі потрібно підключити обрану СУБД до застосунку. Для цього в файлі application.properties створюємо спеціальні налаштування для цього (див. рисунок 3.10):

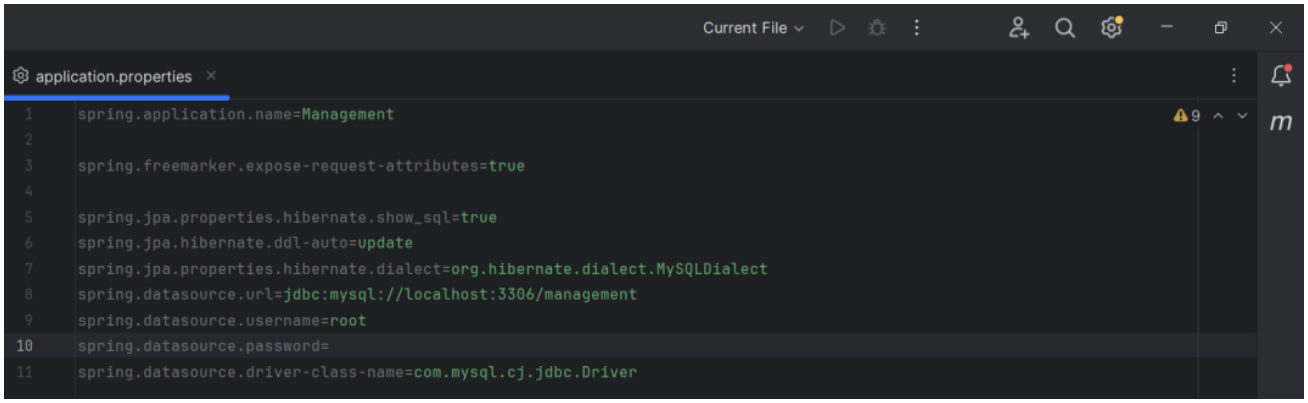


Рис. 3.10 Налаштування зв’язку з MySQL

Далі для налоштування безпеки, прав доступу та авторизації був створений клас – CustomUserDetailService з реалізацією UserDetailsService (див. рисунок 3.11). Цей клас спеціально налаштований для розроблюваного застосунку.

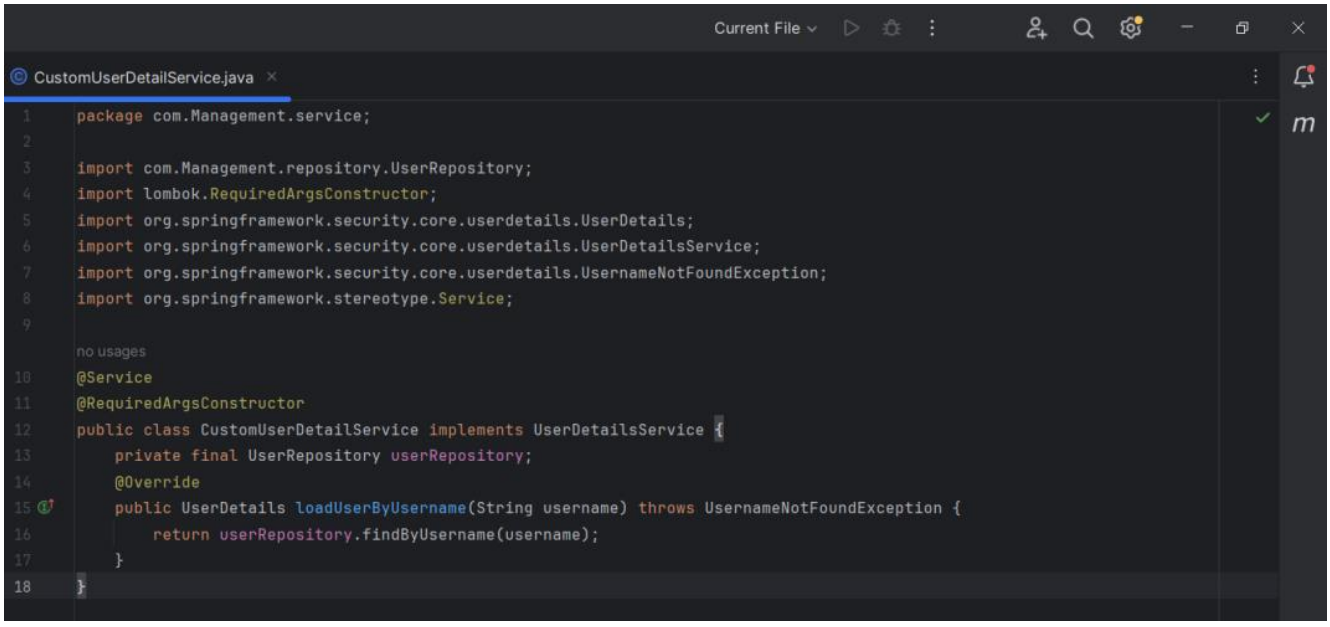


Рис. 3.11 Створення кастомного UserDetailService

Далі настав час створити сервісний шар застосунку, де будуть реалізовані базові CRUD операції і додатковий функціонал. Варто зауважити, що методи для

оновлення даних мають додаткові перевірки для забезпечення оновлення даних в тому випадку, коли користувач бажає оновити лише одне поле і залиши всі інші незмінними (див. рисунок 3.12).

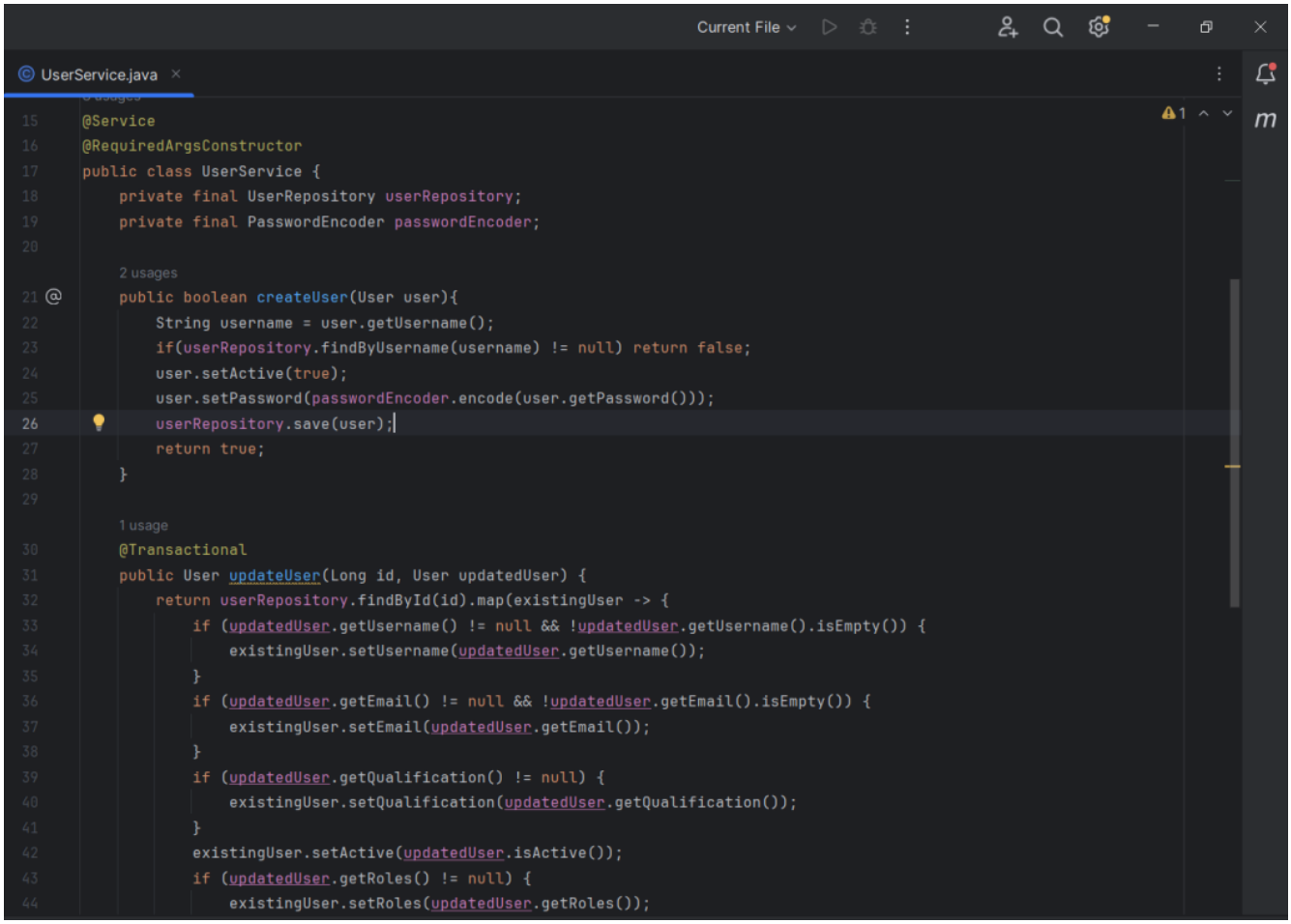


Рис. 3.12 Створення UserService

В UserService також реалізовано метод для підбору співробітників за їх кваліфікацією та роллю (див. рисунок 3.13). Алгоритм отримує налаштування для пошуку і шукає серед всіх співробітників тих, хто відповідає визначеним вимогам. Далі серед всіх отриманих співробітників обирається той, хто має найменшу кількість призначених задач і проектів.

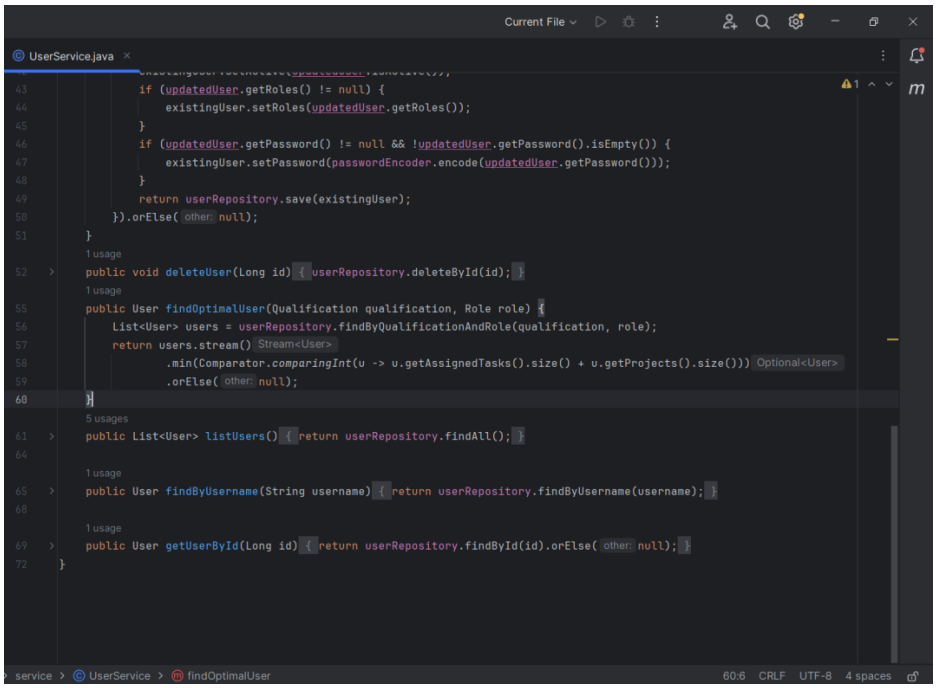


Рис. 3.13 Реалізація алгоритму для пошуку співробітників

3.6 Налаштування безпеки

На решті, для налаштування прав доступу в додатку створюється SecurityConfig. Також був створений passwordEncoder(), який відповідає за хешування паролю користувачів (див. рисунок 3.14).

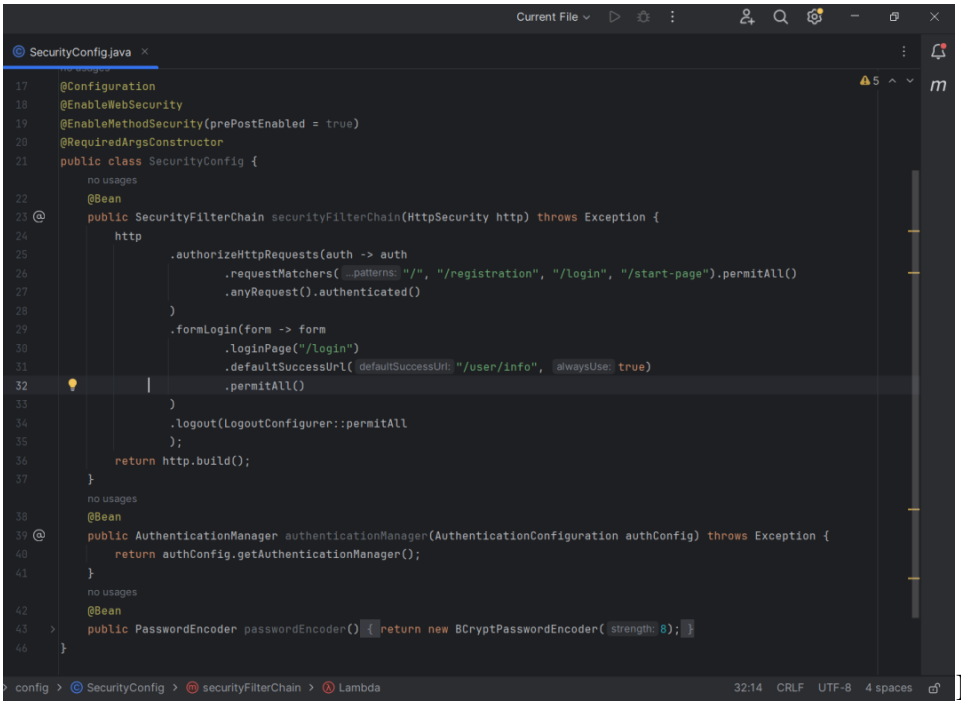


Рис. 3.14 Налаштування безпеки

3.7 Створення клієнтської частини застосунку

За допомогою Spring Freemarker в розроблюваному застосунку буде створено клієнтську частину. Спочатку були написані контроллери, які будуть слугувати мостом між браузерною сторінкою та серверною частиною. Отримуючи всі необхідні дані з сторінок, вони використовують логіку в серверній частині для оперування даними (див. рисунок 3.15).

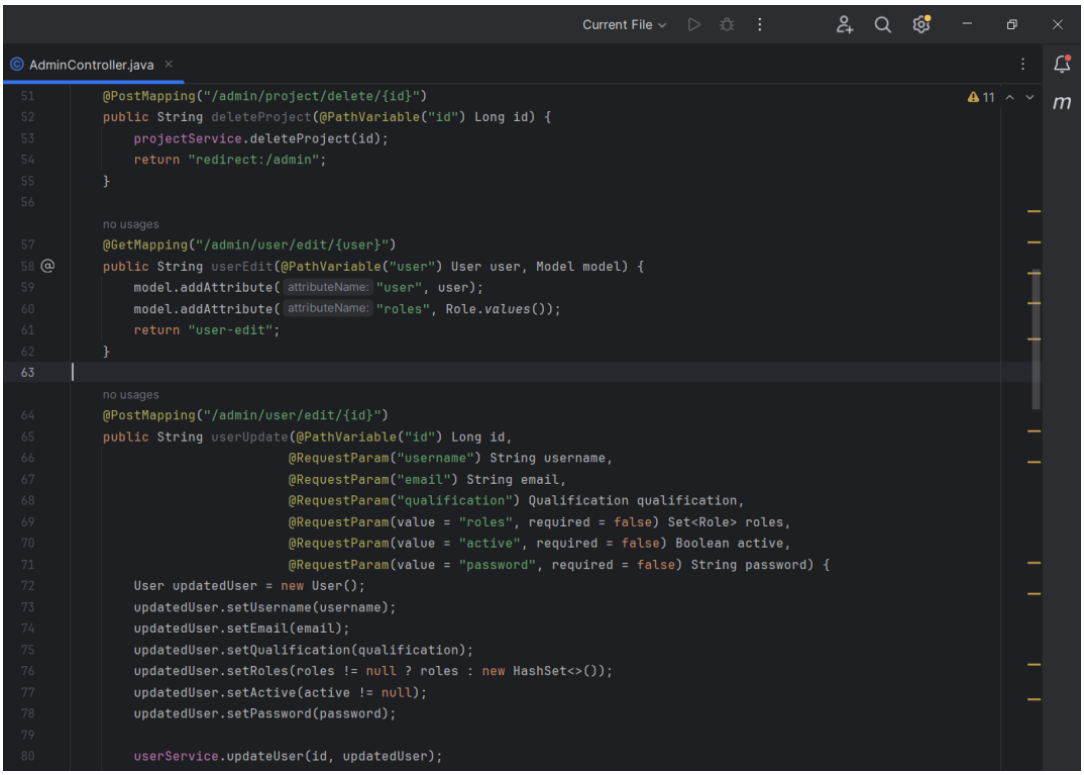


Рис. 3.15 Створення контролерів

Далі створюються відповідні .ftlh файли, як це показано на рисунку 3.16, в яких за допомогою html та javascript створюються всі необхідні елементи та функціонал сторінок,як це показано на рисунку 3.17:

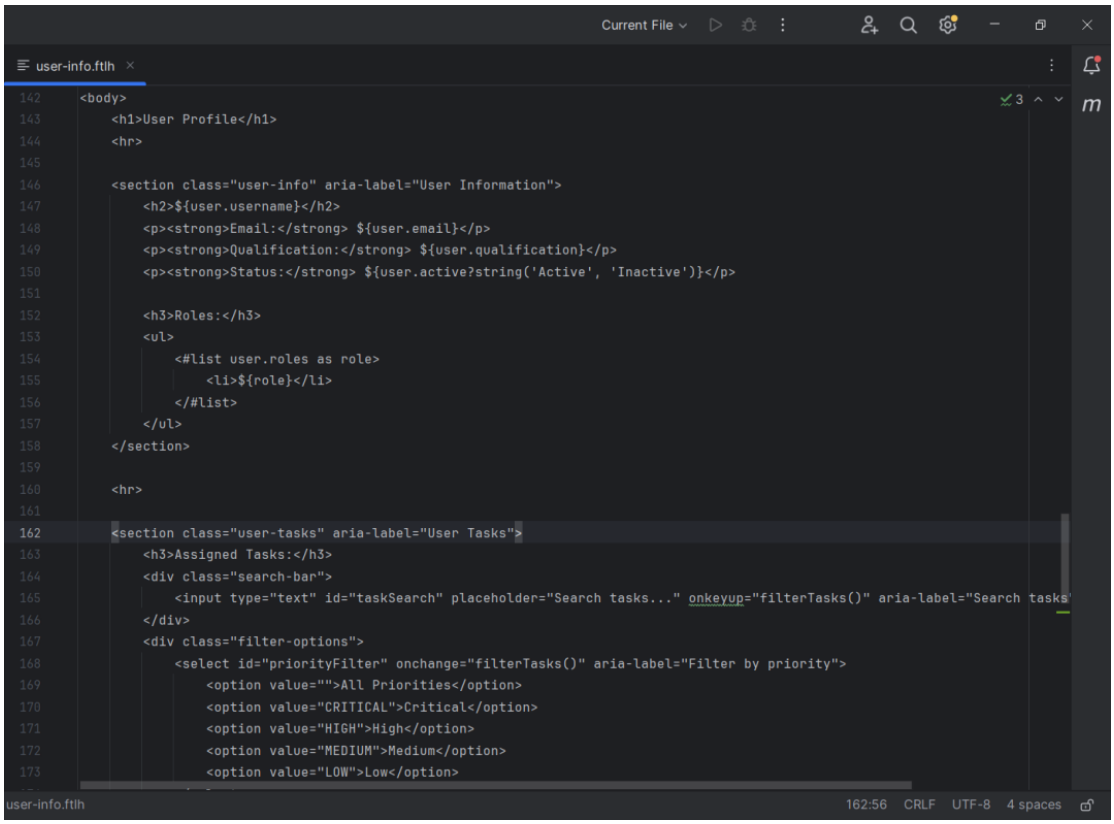


Рис. 3.16 Створення сторінок

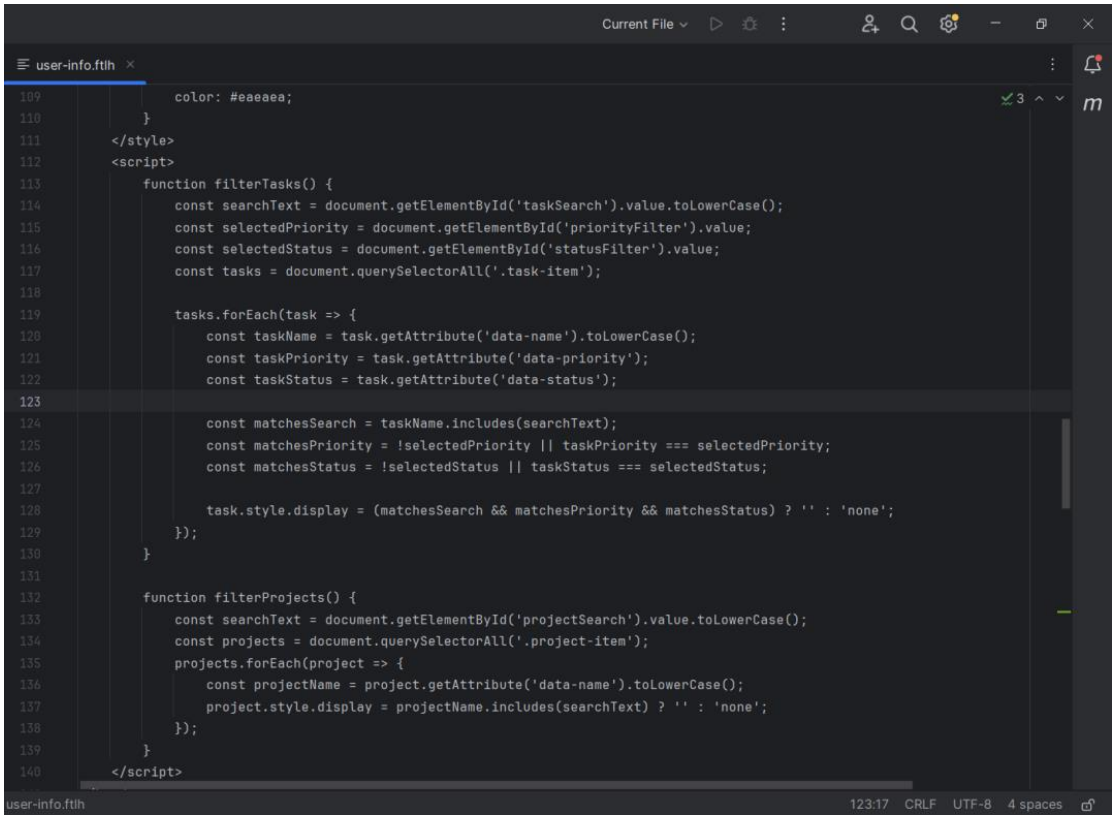


Рис. 3.17 Використання JS

На решті, для оформлення цих сторінок був застосований css та стилізований за допомогою колірна гамма - Unexpected Contrast, як це показано на рисунку 3.18. Ця схема колірного контрасту передбачає поєднання кольорів, які зазвичай не поєднуються, що часто призводить до візуально вражаючої або унікальної естетики.

Це може включати використання кольорів з протилежних сторін колірного кола, як-от помаранчевий і синій, або поєднання світлого кольору з дуже темним, як-от жовтий і чорний. Неочікуваний контраст також може включати використання приглушених або пастельних кольорів з яскравими або насиченими кольорами.

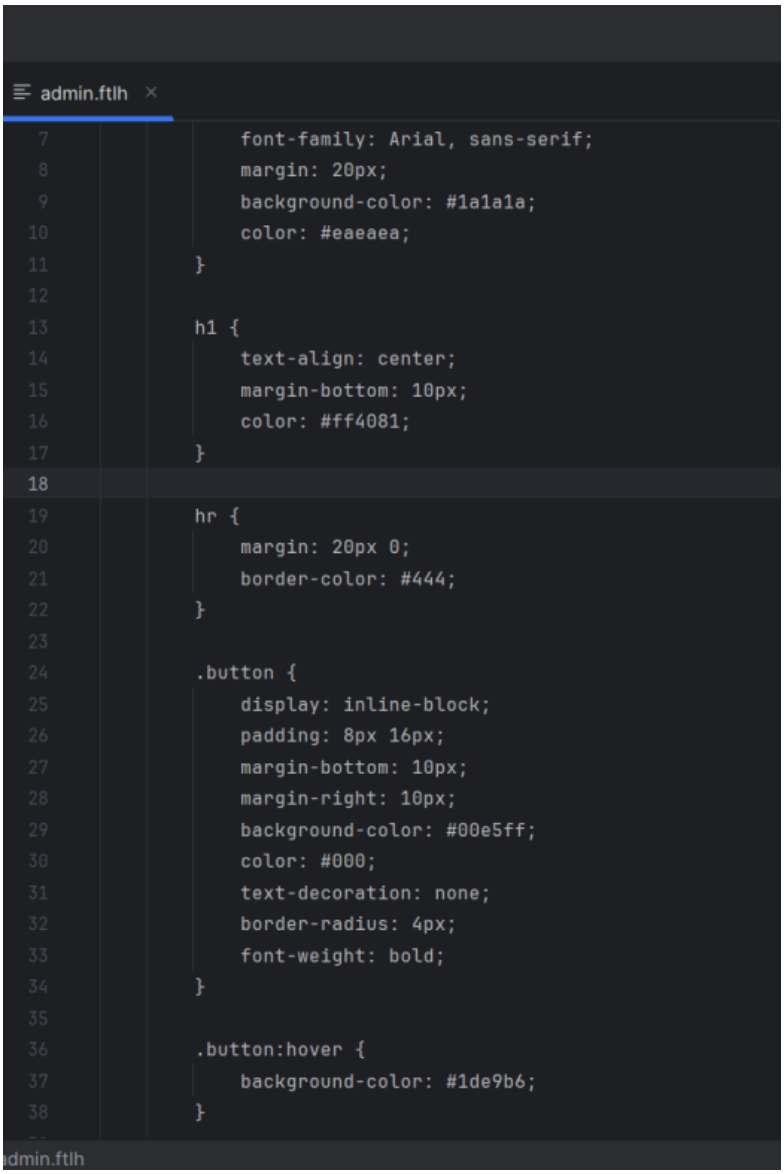


Рис. 3.18 Використання CSS

4 ТЕСТУВАННЯ WEB-ЗАСТОСУНКУ

4.1 Підхід до тестування

Для найкращого тестування застосунку з різними підходами були обрані наступні типи тестування:

- мануальне тестування - процес перевірки функціональності застосунку вручну без використання автоматизованих засобів. Тестувальник взаємодіє з інтерфейсом користувача, виконує сценарії, виявляє помилки, перевіряє відповідність очікуваному результату;
- модульне тестування - це тип тестування, що зосереджується на перевірці окремих модулів або компонентів програмного забезпечення в ізоляції. Метою є виявлення помилок у логіці функцій, методів або класів ще на ранніх етапах розробки. Модульні тести зазвичай пишуть розробники із застосуванням фреймворків, таких як Jest, Mocha, JUnit або інші, залежно від мови програмування. Це дозволяє швидко перевіряти коректність змін у коді та забезпечує регресійний захист.

4.2 Мануальне тестування

Ручне тестування web-сайтів є невід’ємною частиною процесу розробки, адже воно дає змогу виявити помилки й недоліки, що можуть негативно вплинути на зручність користування. Під час такого тестування фахівці уважно перевіряють усі функціональні можливості та елементи інтерфейсу сайту.

Перед початком робіт тестувальники мають чітко ознайомитися з вимогами до продукту та очікуваними результатами його роботи. Далі складається тестовий план, в якому визначаються обсяги перевірки, строки виконання та інші ключові аспекти.

Наступним кроком є створення детальних тест-кейсів, які описують послідовність дій, що будуть виконуватися на сайті. У ході тестування перевіряються такі аспекти, як:

- правильність роботи функціоналу;
- коректне відображення інтерфейсу;
- зручність взаємодії з користувачем тощо.

Усі виявлені помилки фіксуються для подальшого усунення. Після внесення виправлень проводиться повторна перевірка, щоб упевнитися в усуненні проблем та переконатися, що не з’явились нові баги. Додатково тестується сумісність сайту з різними пристроями, браузерами та операційними системами. Оцінюється також зручність користування та рівень безпеки.

На фінальному етапі складається звіт, що містить перелік усіх знайдених дефектів, проблем і рекомендацій щодо поліпшення якості web-сайту.

Для виконання цього тестування були створені наступні тест кейси:

Тест кейс #1: Реєстрація нового користувача

ID тесту : TC001

Опис тесту : Перевірка можливості реєстрації нового користувача в системі через форму реєстрації, як це показано на рисунку 4.1

Очікуваний результат : Система успішно створює нового користувача з введеними даними та перенаправляє на сторінку входу

Фактичний результат : Користувач успішно створюється в базі даних, пароль хешується, встановлюється статус active=true (див. рисунок 4.2)

Статус : Пройдено

Примітки : Перевірити валідацію унікальності email та username

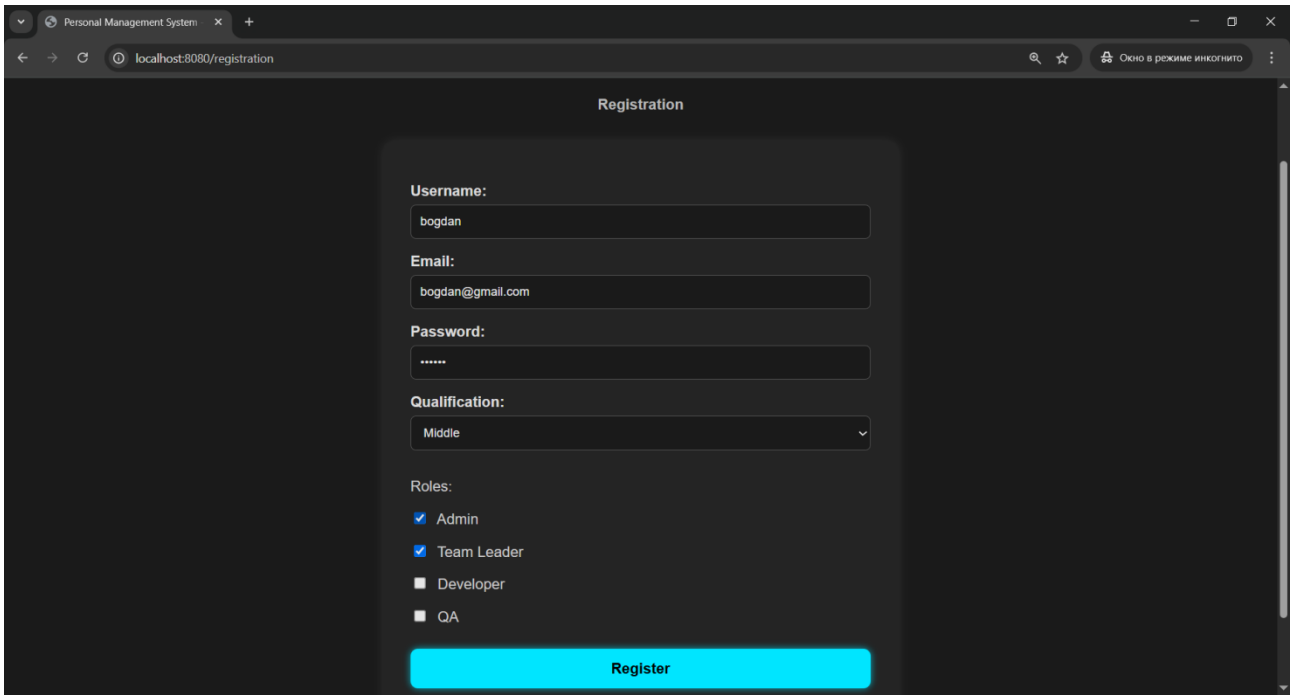


Рис. 4.1 Введення валідних даних при реєстрації

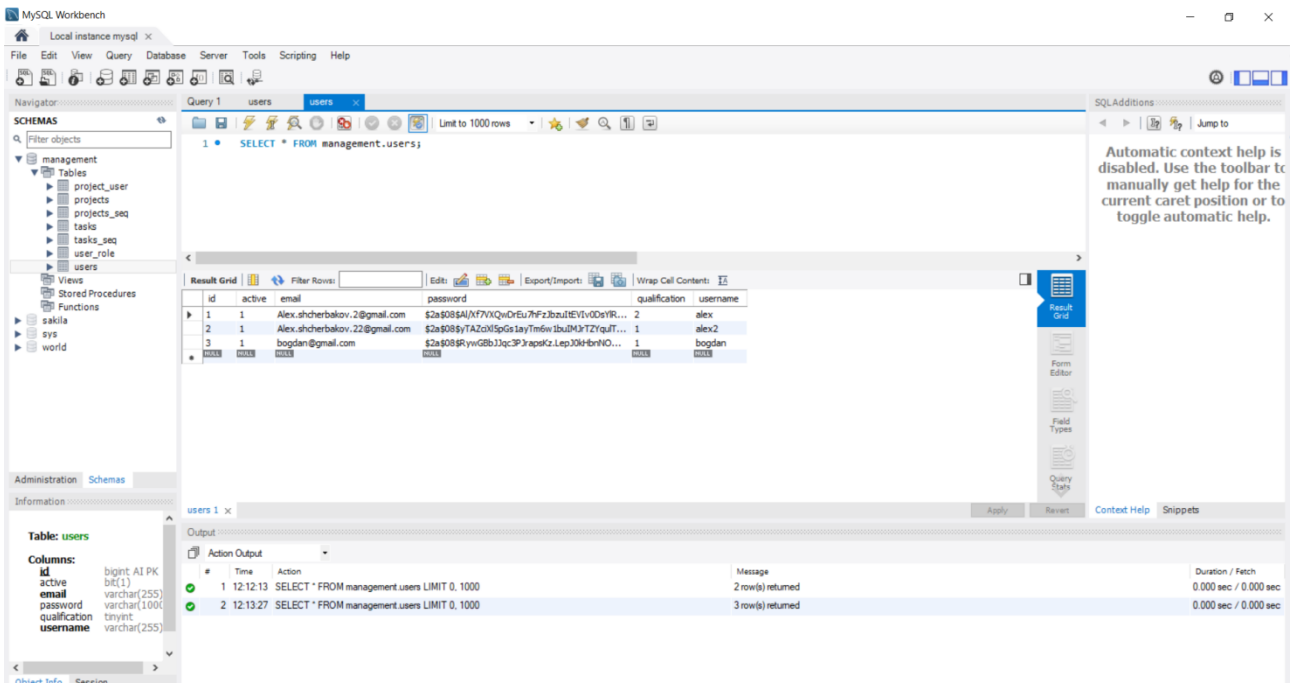


Рис. 4.2 Зміни в БД

Тест кейс #2: Створення нового проекту адміністратором

ID тесту : TC002

Опис тесту : Перевірка функціоналу створення нового проекту з призначенням користувачів (див. рисунок 4.3)

Очікуваний результат : Проект створюється з вказаними параметрами (назва, опис, дедлайн) та призначеними користувачами

Фактичний результат : Проект зберігається в базі даних з усіма вказаними параметрами (див. рисунок 4.4)

Статус : Пройдено

Примітки : Перевірити коректність відображення призначених користувачів у проекті

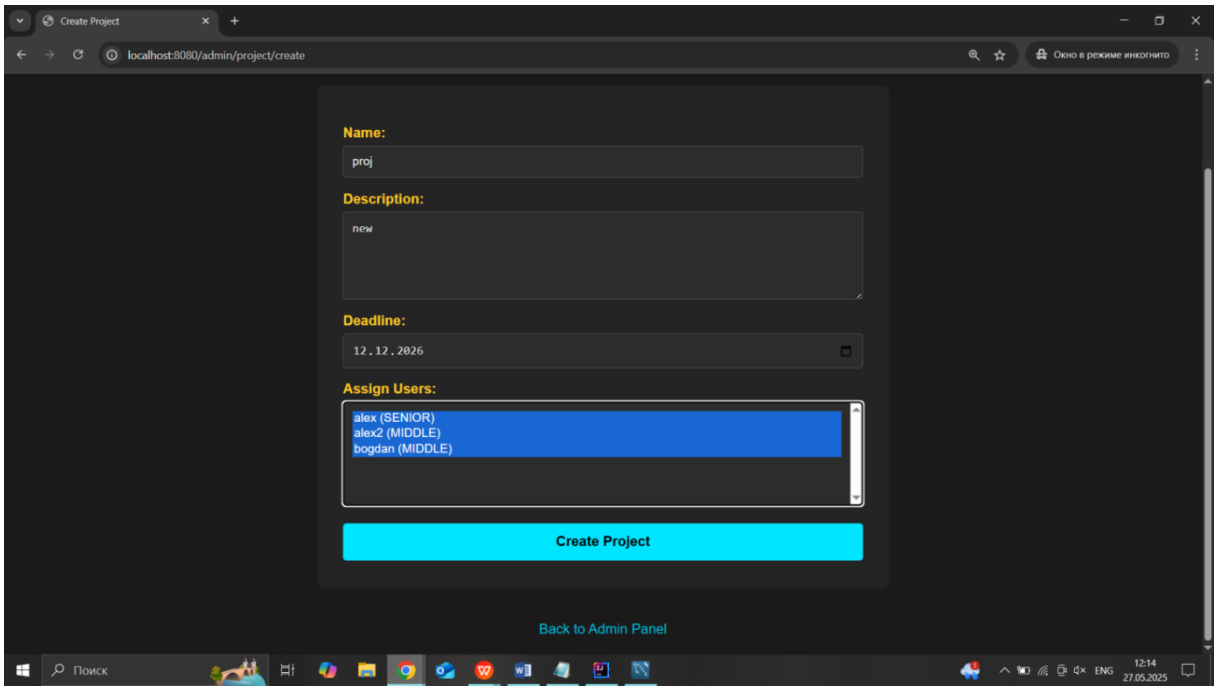


Рис. 4.3 Створення нового проекту

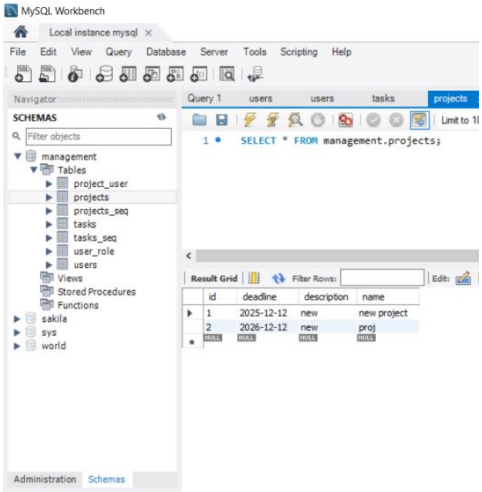


Рис. 4.4 Зміни в БД

Тест кейс #3: Пошук оптимального користувача для завдання

ID тесту : TC003

Опис тесту : Тестування функції findOptimalUser для пошуку користувача з певною кваліфікацією та роллю

Очікуваний результат : Система знаходить користувача з найменшою кількістю поточних завдань та проектів

Фактичний результат : Повертається користувач, що відповідає заданим критеріям та має найменше навантаження (див. рисунок 4.5)

Статус : Пройдено

Примітки : Перевірити коректність підрахунку загального навантаження (проекти + завдання)

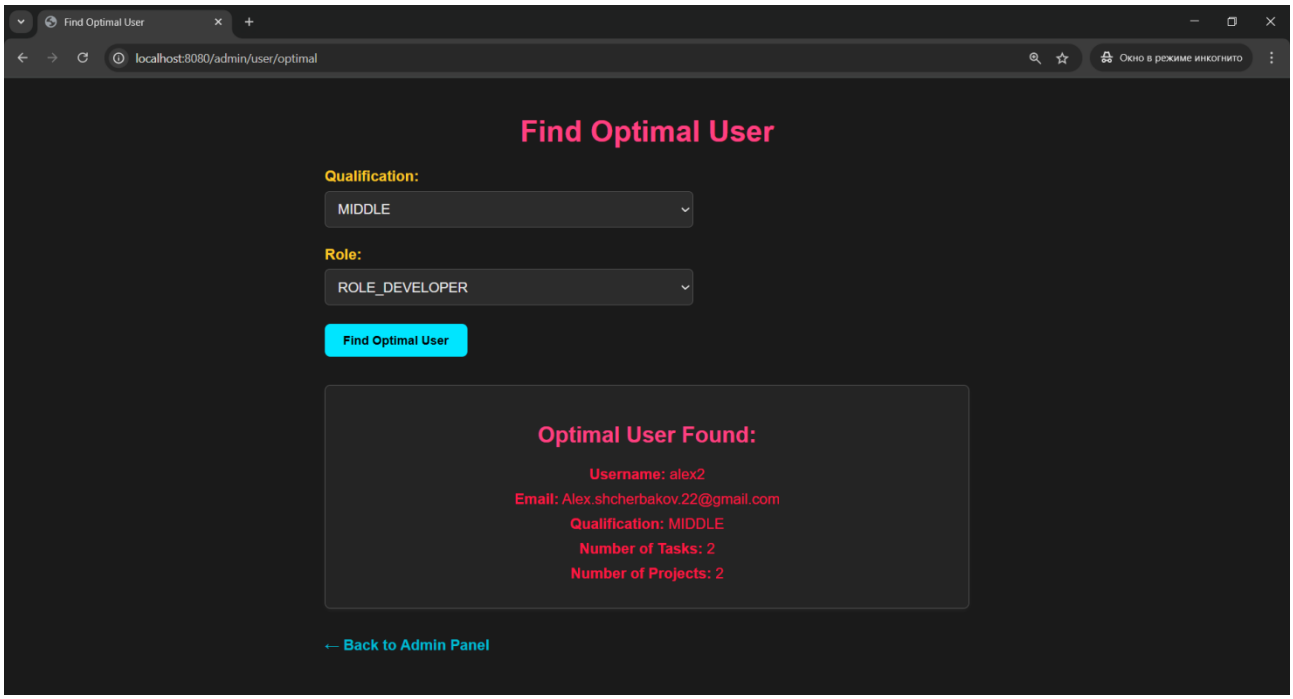


Рис. 4.5 Пошук оптимального співробітника

Тест кейс #4: Оновлення статусу завдання

ID тесту : TC004

Опис тесту : Перевірка можливості зміни статусу існуючого завдання

Очікуваний результат : Статус завдання успішно оновлюється, зміни зберігаються в базі даних

Фактичний результат : Система зберігає новий статус завдання та відображає оновлену інформацію (див. рисунок 4.6 та рисунок 4.7)

Статус : Пройдено

Примітки : Перевірити всі можливі статуси завдань та права доступу для їх зміни

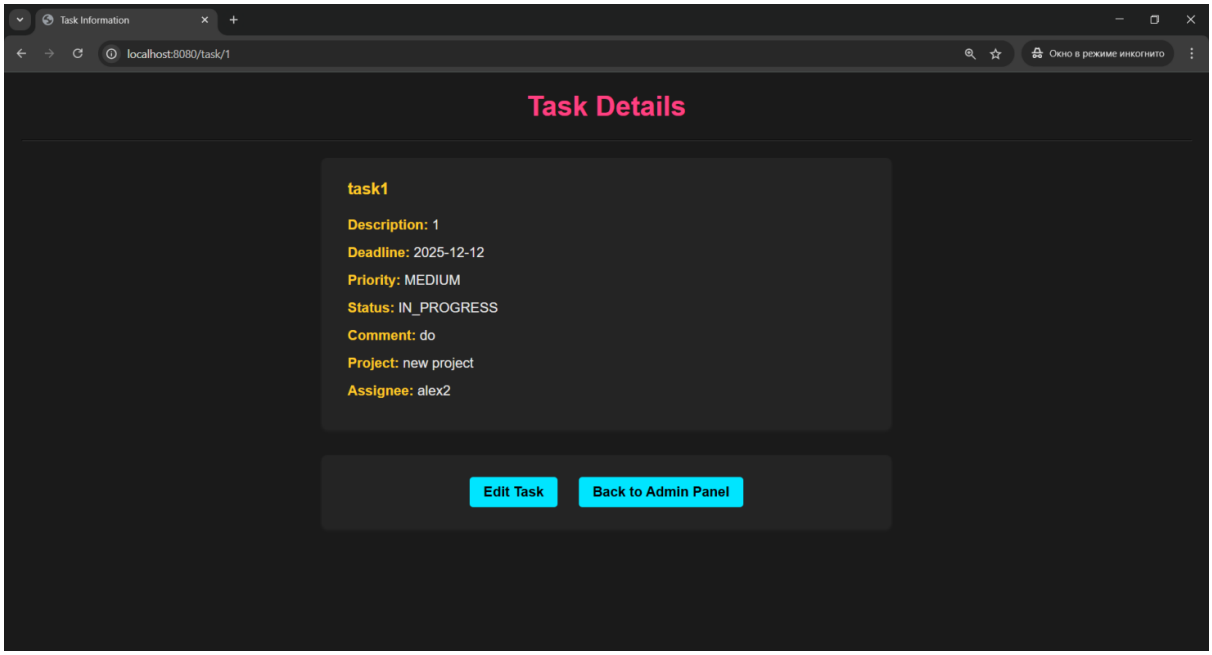


Рис. 4.6 Початковий стан задачі

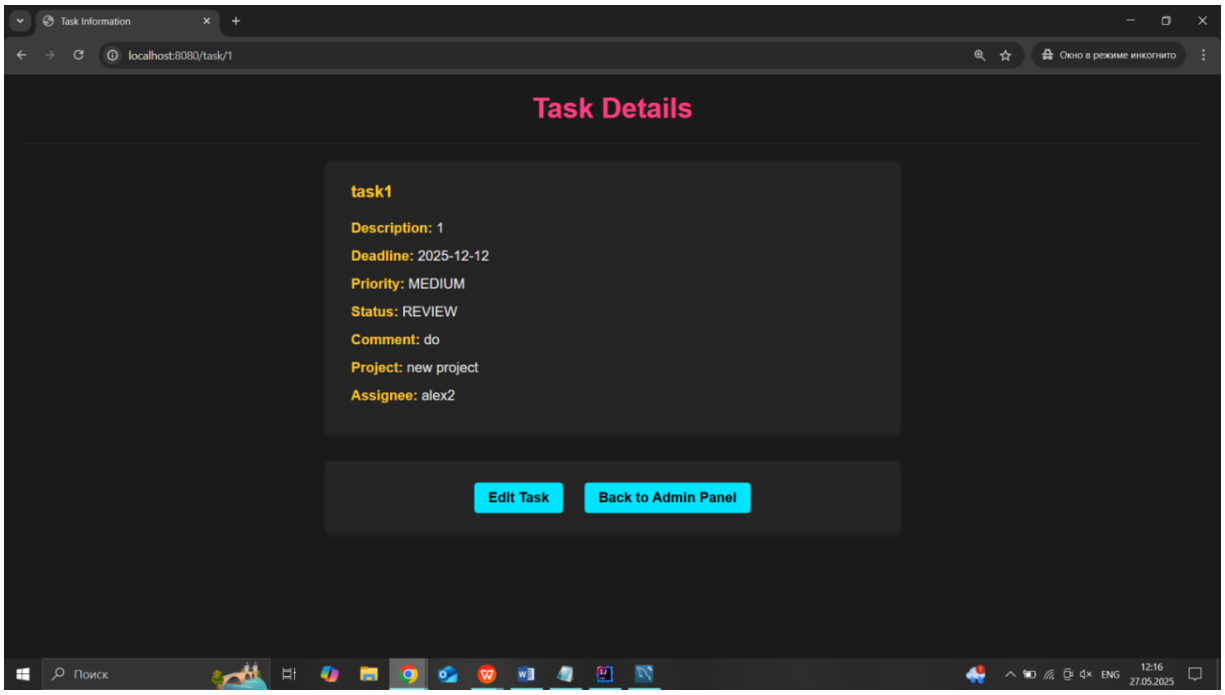


Рис. 4.7 Змінений стан задачі

4.3 Модульне тестування

Модульне тестування є ключовим етапом процесу розробки програмного забезпечення, оскільки дозволяє виявити помилки на рівні окремих компонентів (модулів) програми. Під час цього етапу розробники або тестувальники перевіряють кожен модуль окремо, ізолюючи його від решти системи для точної діагностики.

Перед початком тестування необхідно чітко визначити функціональність кожного модуля та очікувані результати його роботи. На основі цього складається план модульного тестування, де зазначаються обсяг перевірки, критерії успішності, строки та інші організаційні моменти.

Після цього створюються тестові сценарії або юніт-тести, що описують конкретні вхідні дані, очікувану поведінку модуля та кінцевий результат. Під час виконання тестів спеціалісти оцінюють:

- правильність обробки вхідних даних;
- точність логіки виконання;
- відповідність результатів очікуванням;
- поведінку при нештатних умовах.

У разі виявлення помилок вони фіксуються для виправлення. Після внесення змін до коду модуль тестується повторно, щоб переконатися у відсутності проблем і збереженні працездатності.

Модульне тестування часто автоматизується, що дозволяє оперативно перевіряти компоненти при кожній зміні коду. Завершується процес складанням звіту з результатами тестування, де зазначаються всі виявлені помилки, дані про покриття тестами та рекомендації щодо вдосконалення коду.

Для функціоналу web-застосунку були написані тест кейси, показані на рисунках 4.8, 4.10, 4.12, які мають результати проходження, показані на рисунках 4.9, 4.11, 4.13:

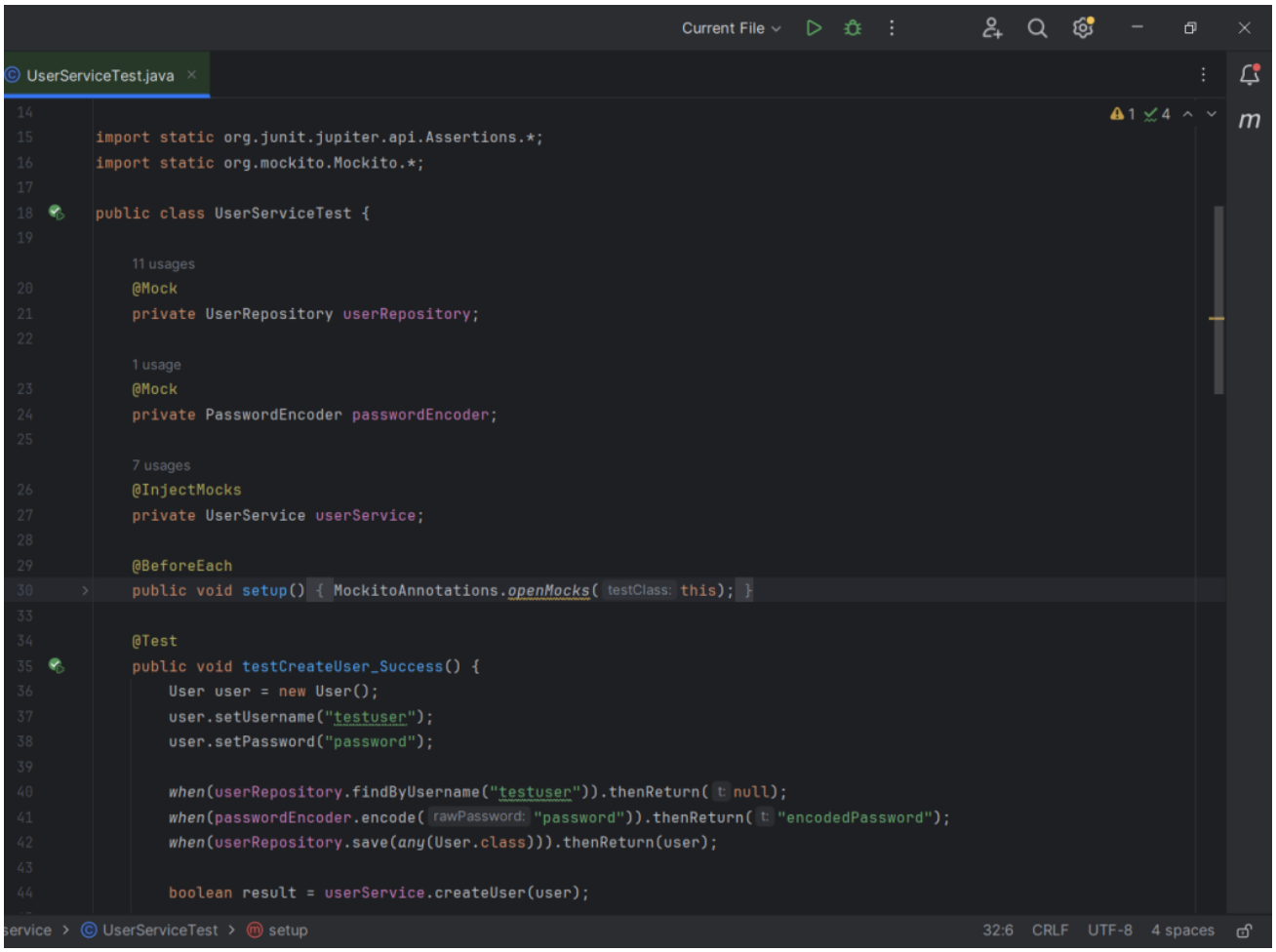


Рис. 4.8 Тести для UserService

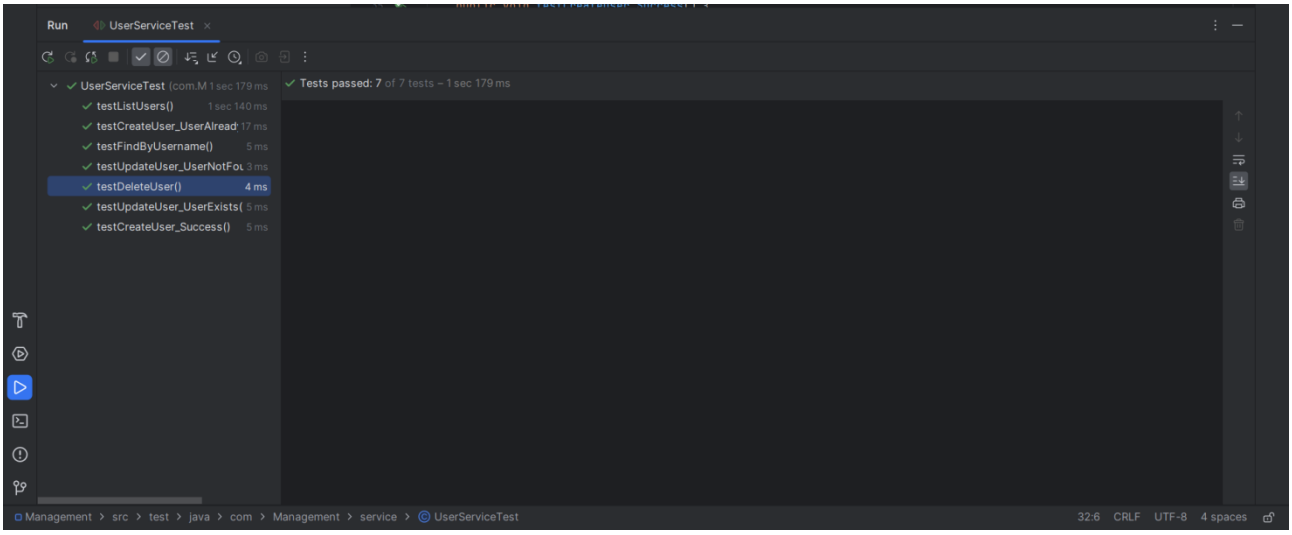


Рис. 4.9 Успішне проходження тестів

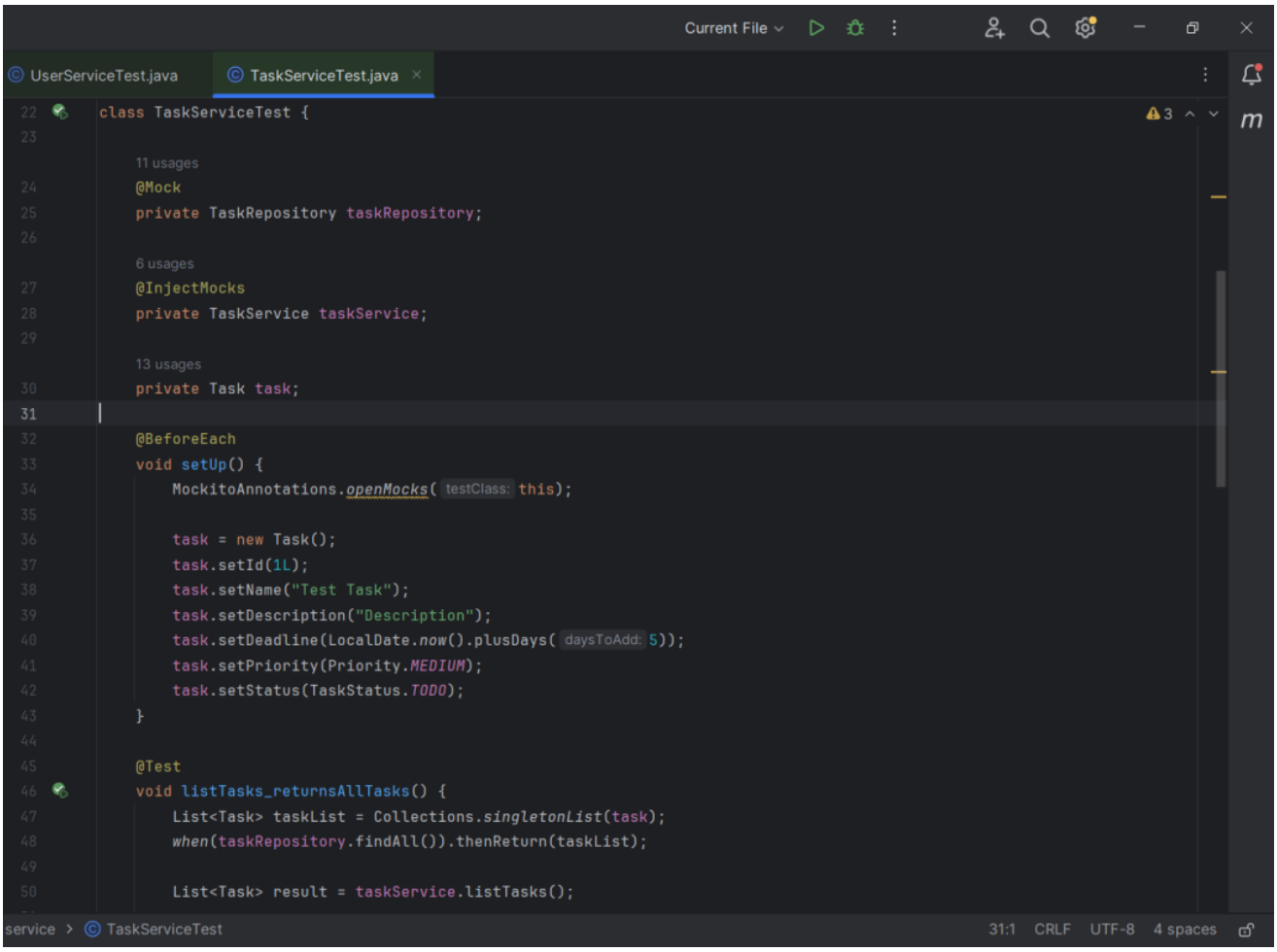


Рис. 4.10 Тести для TaskService

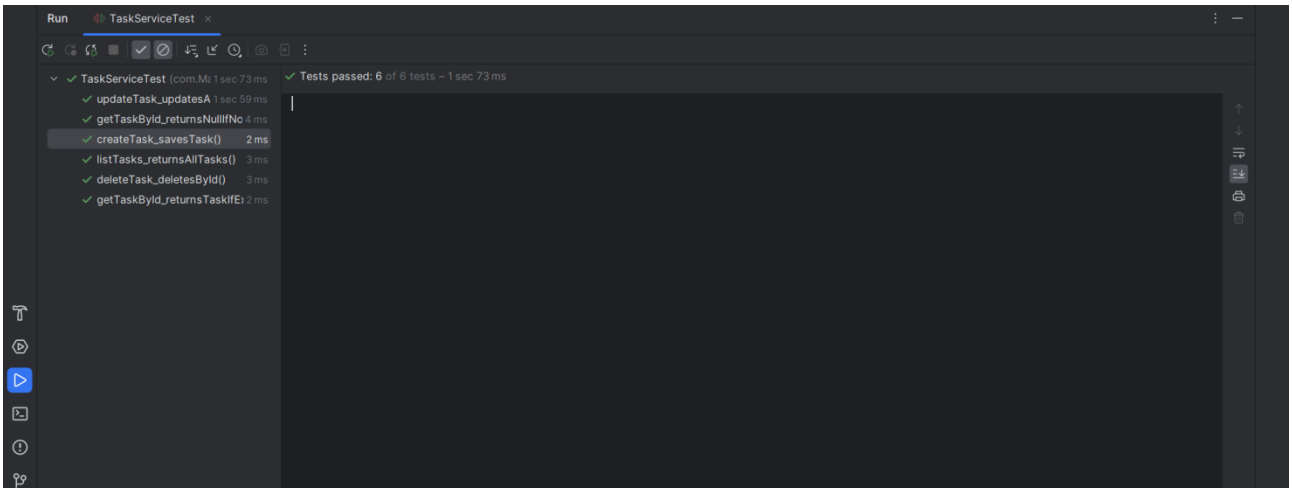


Рис.4. 11 Успішне проходження тестів

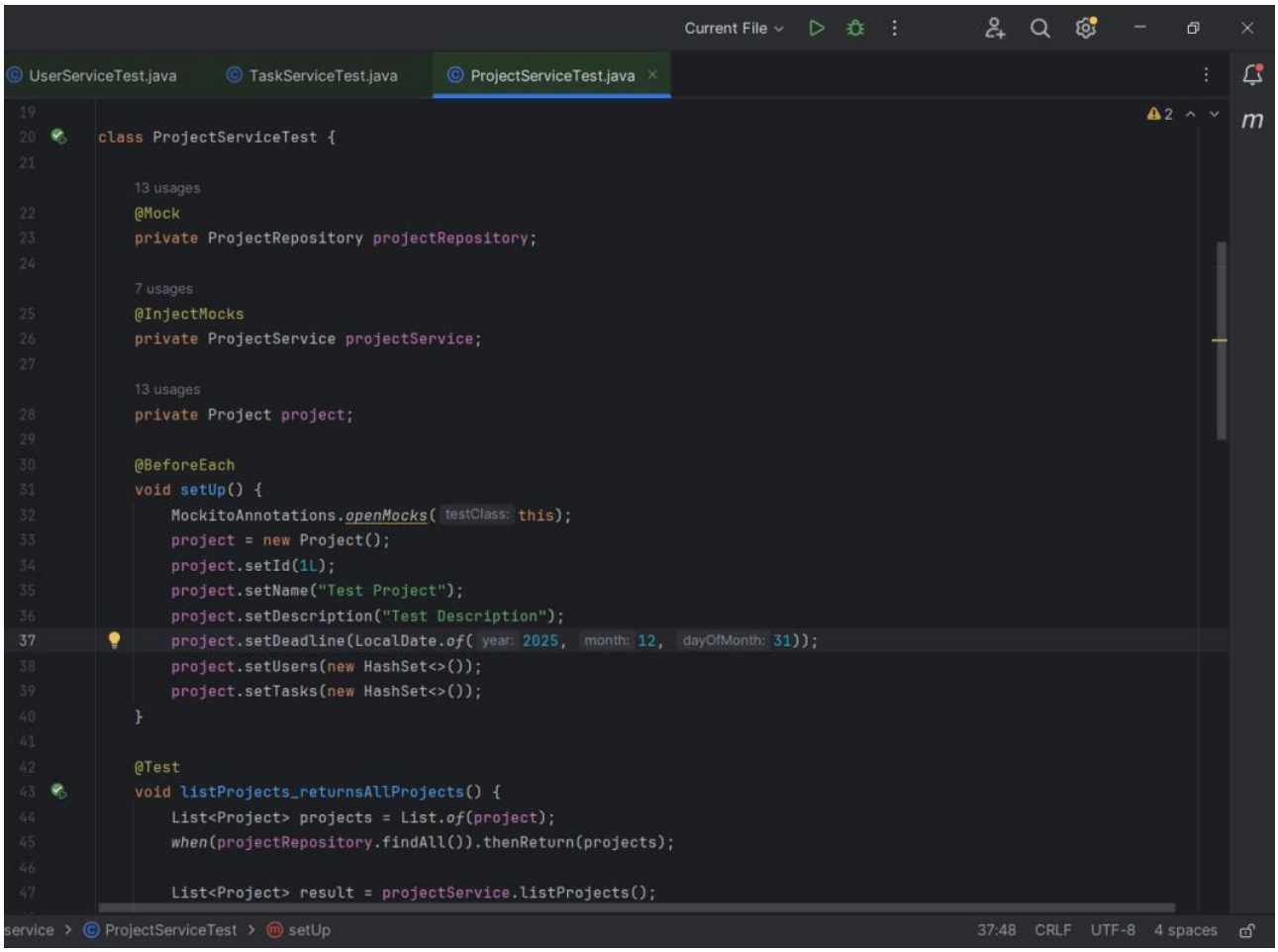


Рис. 4.12 Тести для ProjectService

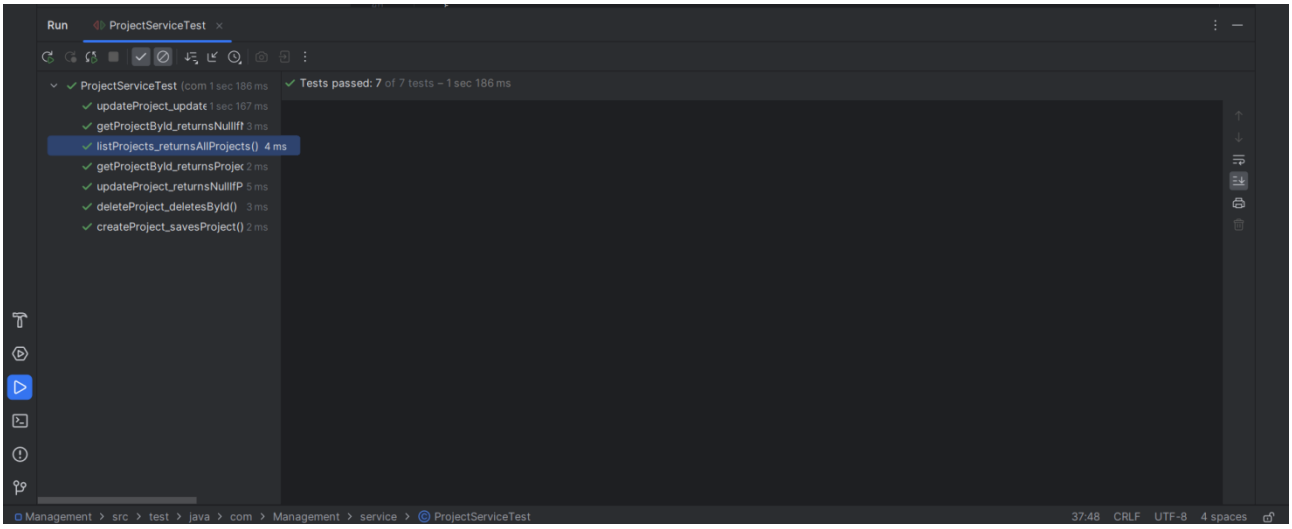


Рис.4. 13 Успішне проходження тестів

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було досягнуто мети — розроблено функціональний web-застосунок, що дозволяє автоматизувати процеси управління персоналом в IT-аутсорсинговій компанії. У роботі здійснено комплексний аналіз предметної області, вивчено існуючі програмні рішення (BambooHR, Zoho People, Personio), на основі якого сформульовано вимоги до розроблюваної системи.

Проект було реалізовано за допомогою сучасного технологічного стеку: Java (Spring Boot, Spring Security, Spring Data JPA) для серверної частини, HTML/CSS/JavaScript для клієнтської частини та MySQL для зберігання даних. Було застосовано багатошарову архітектуру, що забезпечує гнучкість, масштабованість і зручність підтримки застосунку.

Web-застосунок реалізує важливі функції: управління користувачами, проектами та задачами, сортування та фільтрацію задач, гнучку рольову модель доступу, а також алгоритм автоматичного підбору працівників за кваліфікацією та роллю. Проведено тестування — модульне та мануальне — що підтвердило працездатність та відповідність програмного продукту заявленим вимогам.

Розроблене рішення є актуальним, оскільки враховує специфіку IT-аутсорсингової галузі та може бути використане в малих компаніях та стартапах для покращення організації роботи персоналу.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Щербаков О.О. Технології, методи та засоби захисту інформації в інформаційно-комунікаційних системах. Матеріали XII Всеукраїнської науково-практичної конференції молодих учених "IT-2025". Збірник тез 15 травня 2025 року, КСУ ім.Грінченка, м.Київ - С.345-347.

2. Щербаков О.О., Калинюк Б.С. Актуальність використання web-застосунків для управління персоналом в аутсорсингових компаніях. Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24 квітня 2025 року, ДУІКТ, м.Київ - С.72-73.

ПЕРЕЛІК ПОСИЛАНЬ

1. n-ix.com. How to Solve 13 Harsh Problems of Outsourcing. Режим доступу: https://www.n-ix.com/how-to-solve-problems-of-it-outsourcing/ (дата звернення: 21.04.2025).
2. kodytechnolab.com. Top 10 Outsourcing Problems: Effective Solutions. Режим доступу: https://kodytechnolab.com/blog/top-10-outsourcing-problems-effective-solutions/ (дата звернення: 21.04.2025).
3. easy-software.com. Personnel management: basics, goals and central tasks. Режим доступу: https://easy-software.com/en/newsroom/personnel-management-basics-goals-and-central-tasks/ (дата звернення: 22.04.2025).
4. researchgate.net. Specificities Human Resource Management In It Companies. Режим доступу: https://www.researchgate.net/publication/374053033_Specificities_Human_Resource_Management_In_It_Companies (дата звернення: 22.04.2025).
5. Evans B., Clarke J., Verburg M. The Well-Grounded Java Developer. 2nd Edition. Manning Publications, 2022. 704p.
6. Sierra K., Bates B., Gee T. Head First Java. A Brain-Friendly Guidey. 3rd Edition. O'Reilly Media, 2022. 752 p.
7. Burd B. Java for dummies. 9th Edition. For Dummies, 2025. 496 p.
8. Oracle. Official Documentation. Режим доступу: https://docs.oracle.com/en/java/javase/24/ (дата звернення: 27.04.2025).
9. Flanagan D. JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language. 7rd Edition. O'Reilly Media, 2020. 706p.
10. IntelliJ IDEA. Official Overview. Режим доступу: https://www.jetbrains.com/help/idea/discover-intellij-idea.html (дата звернення: 30.04.2025).
11. Spring Boot. Official Overview. Режим доступу: https://spring.io/projects/spring-boot (дата звернення: 29.04.2025).

12. Walls C. Spring in Action. 6th Edition. Manning Publications, 2022. 520 p.

13. Oliveira C., Turnquist G., Antonov A. Developing Java Applications with Spring and Spring Boot. Packt Publishing, 2018. 982 p.

14. Spilca L. Spring Security in Action. Manning Publications, 2021. 480 p.

15. Spring Security. Spring Security Official Overview. Режим доступу: https://spring.io/projects/spring-security (дата звернення: 01.05.2025).

16. Spring Data JPA. Official Overview. Режим доступу: https://spring.io/projects/spring-data-jpa (дата звернення: 29.04.2025).

17. Tudose C. JUnit in Action. 3rd Edition. Manning Publications, 2020. 560p.

18. junit.org. Official Documentation. Режим доступу: https://junit.org/junit5/docs/current/user-guide/ (дата звернення: 03.05.2025).

19. baeldung.com. Getting Started with Mockito. Режим доступу: https://www.baeldung.com/mockito-annotations (дата звернення: 03.05.2025).

20. Spring Freemarker. Official Overview. Режим доступу: https://docs.spring.io/spring-framework/reference/web/webmvc-view/mvc-freemarker.html (дата звернення: 02.05.2025).

21. Project Lombok. Official documentation. Режим доступу: https://projectlombok.org/contributing/ (дата звернення: 29.04.2025).

22. Spring MySQL. Official Overview. Режим доступу: https://spring.io/guides/gs/accessing-data-mysql (дата звернення: 02.05.2025).

23. DuRocher D. HTML and CSS QuickStart Guide: The Simplified Beginners Guide to Developing a Strong Coding Foundation, Building Responsive Websites, and Mastering the Fundamentals of Modern Web Design. ClydeBank Media LLC, 2021. 352p.

24. medium.com. The Anatomy of a Spring Boot Application: Structure, Best Practices, and Why It Matters. Режим доступу: https://medium.com/@TechiesSpot/the-anatomy-of-a-spring-boot-application-structure-best-practices-and-why-it-matters-1d404700a433 (дата звернення: 23.04.2025).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-застосунку для управління персоналом IT-аутсорсингової компанії

Виконав студент 4 курсу
групи ПД-44
Щербаков Олексій Олександрович
Керівник роботи
асистент кафедри ІПЗ Калинюк Богдан Сергійович

Київ – 2024

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи:** удосконалення процесів управління персоналом в IT-компаніях.
- **Об'єкт дослідження:** процес управління персоналом в IT-аутсорсинговій компанії.
- **Предмет дослідження:** вебзастосунки, призначені для управління персоналом в аутсорсингових IT-компаніях.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати процеси управління персоналом в IT-компаніях.
2. Проаналізувати існуючі програмні рішення для управління персоналом в IT-компаніях і визначити основні потреби користувачів.
3. Сформувати функціональні і нефункціональні вимоги до вебзастосунку , базуючись на вимогах користувачів.
4. Провести огляд засобів для розробки вебзастосунку для управління персоналом в IT-аутсорсингових компаніях.
5. Спроектувати архітектуру вебзастосунку.
6. Реалізувати вебзастосунок згідно визначених вимог.
7. Провести тестування функціоналу розробленого вебзастосунку.

3

АНАЛІЗ АНАЛОГІВ

Показник	BambooHR	Zoho People	Personio	Personnel Management(Мій застосунок)
Основна спеціалізація	Малий та середній бізнес	Компанії будь-якого розміру	Середній та великий бізнес	Малі компанії та стартапи
Інтерфейс	Зручний, інтуїтивний	Гнучкий, може потребувати навчання	Сучасний, орієнтований на користувача	Мінімалістичний дизайн з використанням HTML та CSS
Платформа	Веб, мобільний додаток	Веб, мобільний додаток	Веб, мобільний додаток	Веб
Особливості	Вбудований AI для допомоги та аналітики	Гнучка настройка під потреби бізнесу	Особливо орієнтований на GDPR та європейські стандарти безпеки даних	Наявність розподілу по ролях, можливість відстеження статусу завдань
Переваги	Легкий у використанні, гарна підтримка	Дешевший варіант, багато можливостей кастомізації	Комплексне рішення для Європи, підтримка GDPR	Комплексне управління проектами та завданнями, детальна інформація про проекти та завдання
Недоліки	Не дуже гнучкий для великих компаній	Складний інтерфейс для новачків, низька якість підтримки	Можлива обмеженість для дуже специфічних потреб, доступність Payroll залежить від регіону	Обмежена функціональність для спільної роботи

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні:

1. Реєстрація та авторизація в системі за допомогою логіну та паролю.
2. Призначення користувачу однієї з ролей(Team Leader, Developer, QA, Admin).
3.Можливість управління профілями користувачів (продивлятися акаунти, оновлювати дані, призначати ролі, видаляти акаунти).
4. Можливість управління проектами(створення, редагування, видалення, призначення співробітників).
5. Можливість управління задачами(створення, редагування, видалення, призначення виконавців, написання коментарів)
6. Відображення списоку проєктів та задач користувача.
7. Фільтрація списку задач за назвою, статусом і пріорітетом.
8. Реалізація алгоритму, який автоматично підбирає оптимальних співробітників за параметрами кваліфікації та ролі.

Нефункціональні:

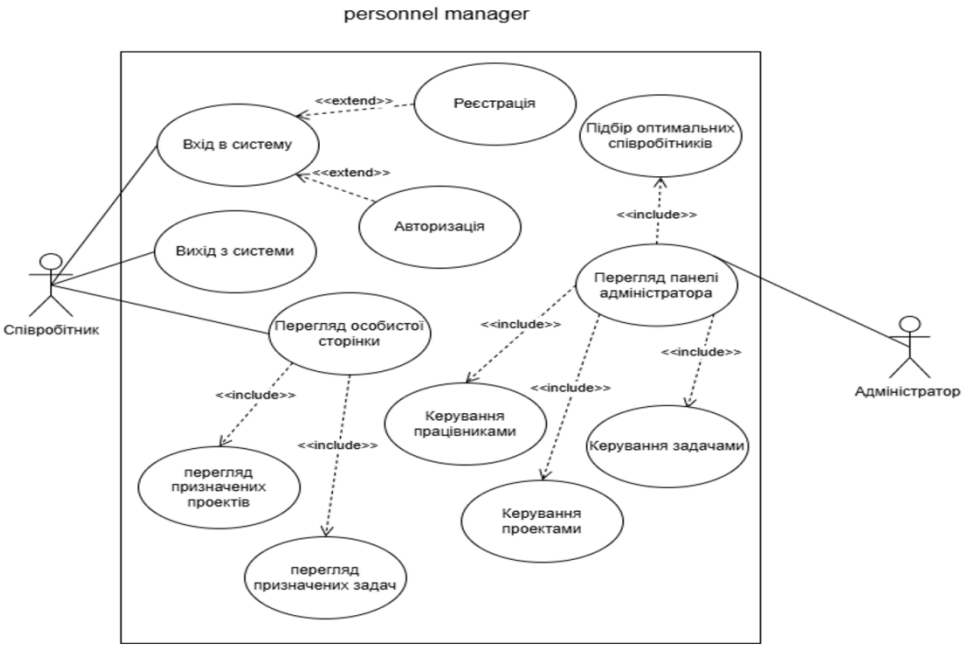
1. Доступ до даних і можливостей обмежено згідно ролі користувача.
2. Запити повинні виконуватися не більш ніж за 5 секунд.
3.Система повина виконувати хешування паролів.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

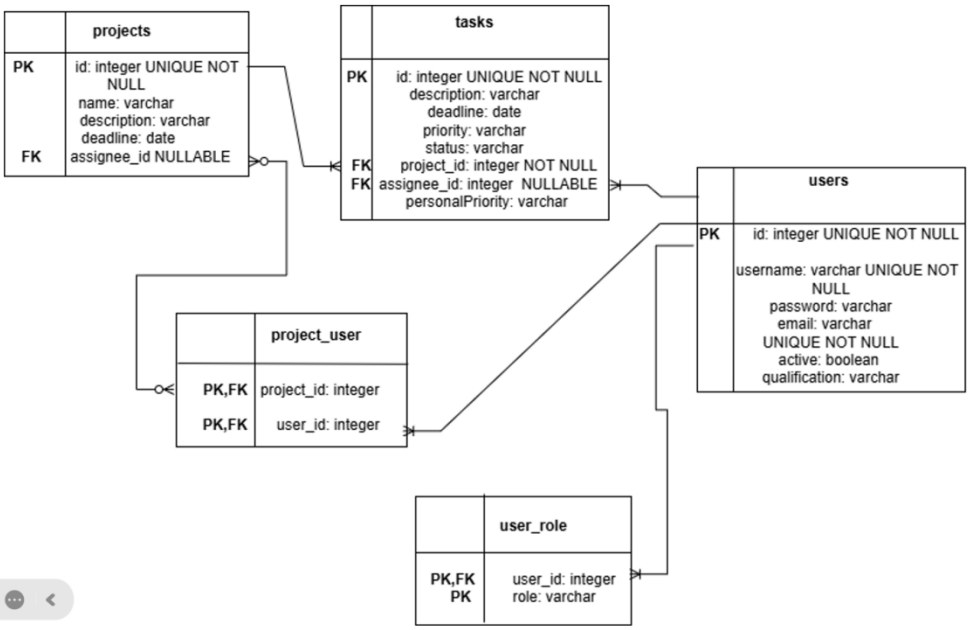
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



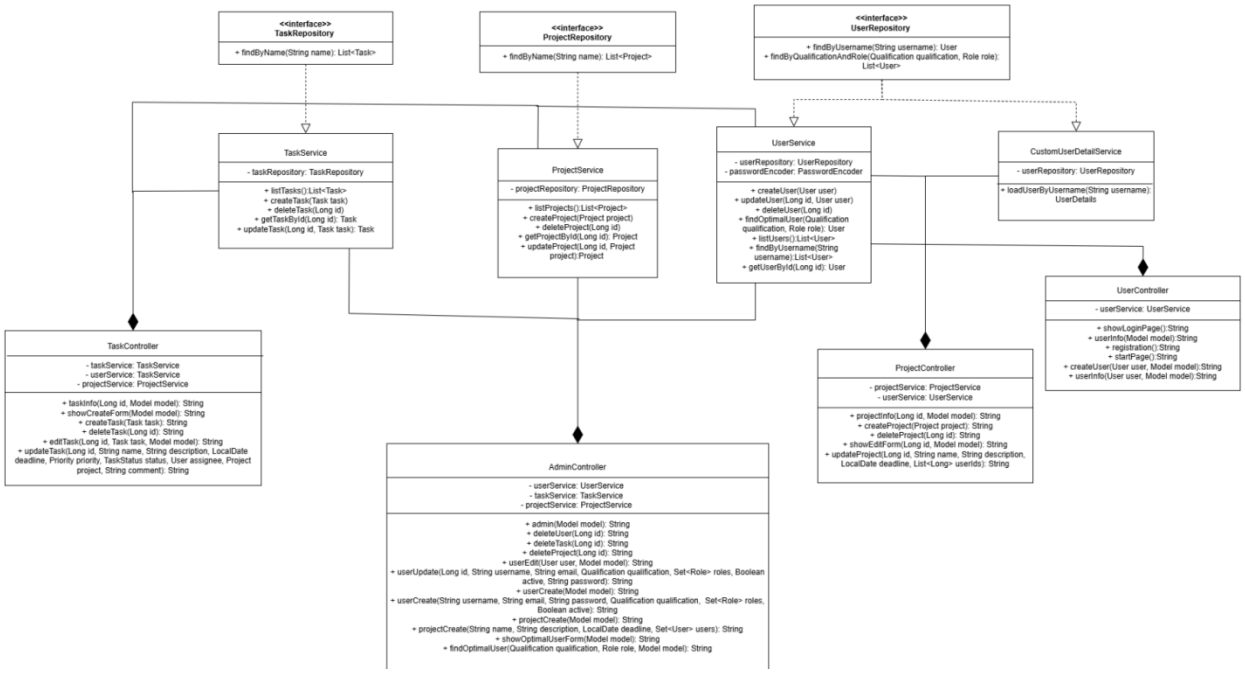
7

СТРУКТУРА БАЗИ ДАННИХ

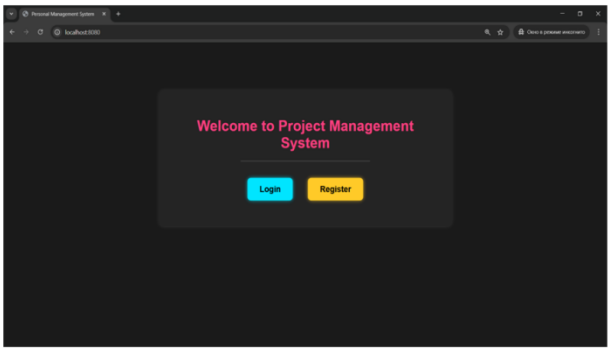


8

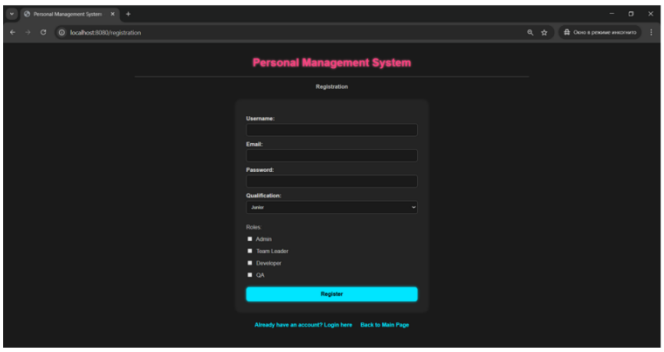
ДІАГРАМА КЛАСІВ СЕРВІСІВ І КОНТРОЛЕРІВ



ЕКРАННІ ФОРМИ

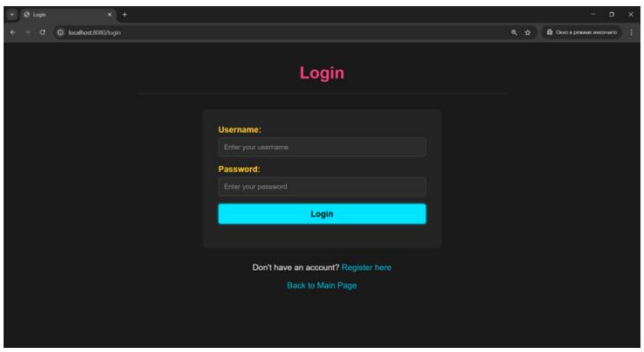


Початкова сторінка

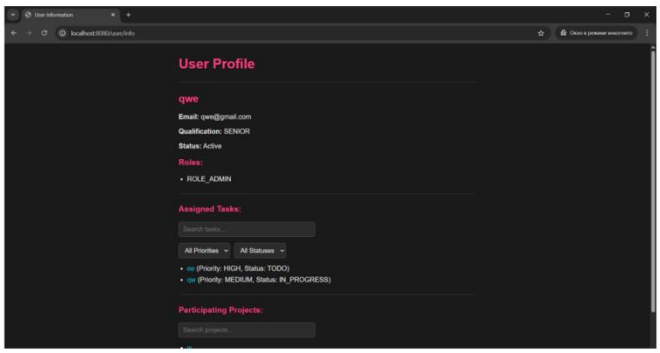


Сторінка реєстрації користувача

ЕКРАННІ ФОРМИ

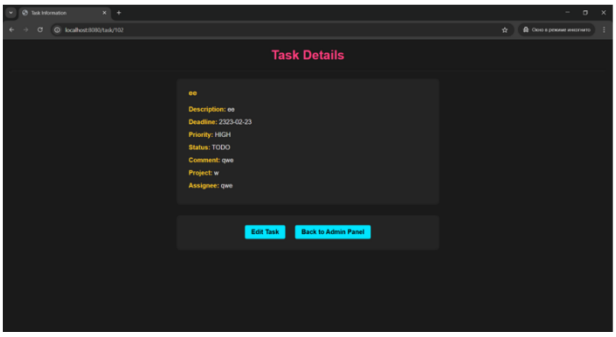


Сторінка логіну

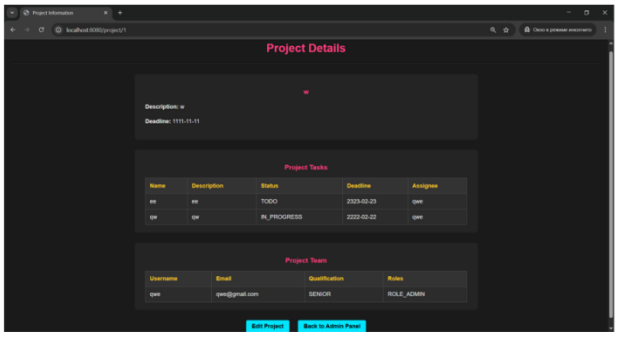


Профіль користувача

ЕКРАННІ ФОРМИ

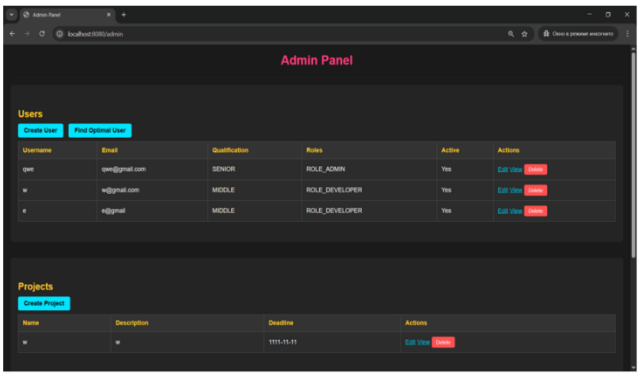


Перегляд деталей завдання

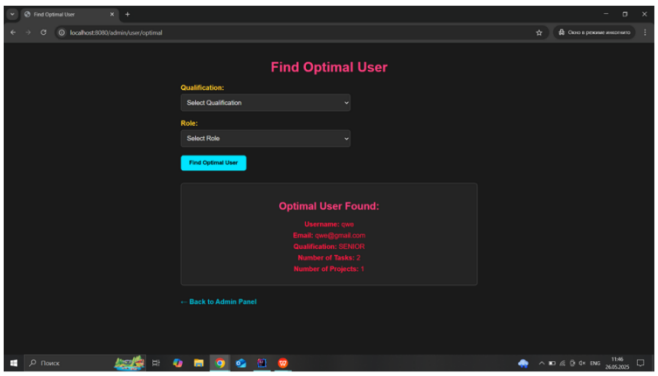


Перегляд деталей проекту

ЕКРАННІ ФОРМИ

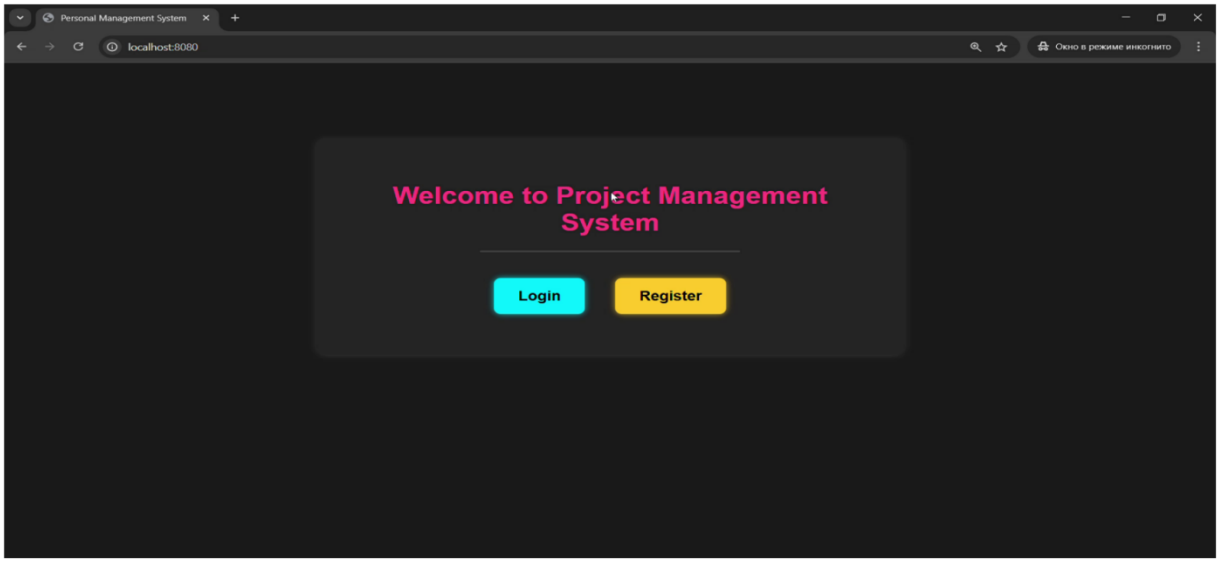


Панель адміністратора



Алгоритм підбору працівників

ДЕМОНСТРАЦІЯ РОБОТИ ПРОГРАМИ



АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

1. Щербаков О.О. Технології, методи та засоби захисту інформації в інформаційно-комунікаційних системах. Матеріали XII Всеукраїнської науково-практичної конференції молодих учених "ІТ-2025". Збірник тез 15 травня 2025 року, КСУ ім.Грінченка, м.Київ - С.345-347.

2. Щербаков О.О., Калинюк Б.С. Актуальність використання веб-застосунків для управління персоналом в аутсорсингових компаніях. Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24 квітня 2025 року, ДУІКТ, м.Київ - С.72-73.

ВИСНОВКИ

1. Проаналізовані процеси управління персоналом в ІТ-компаніях та визначено ефективні підходи до управління персоналом.
2. Проаналізовано існуючі вебзастосунків для управління персоналом і визначені вимоги користувачів до застосунків цього типу, їх переваги та недоліки.
3. Сформовані вимоги до вебзастосунку на основі основних потреб користувачів.
4. Проведений огляд інструментів для розробки вебзастосунку та обгрунтовано їх вибір.
5. Спроектована архітектура для вебзастосунку, на основі базової архітектури(model,service,controller).
6. Реалізовано вебзастосунок, що дозволяє управляти записами користувачів, проектів і задач.
7. Проведено тестування вебзастосунку на відповідність вимогам.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
public enum Priority {
   LOW, MEDIUM, HIGH, CRITICAL
}
public enum Qualification {
   JUNIOR, MIDDLE, SENIOR
}
public enum Role implements GrantedAuthority {
   ROLE_ADMIN, ROLE_TEAM_LEADER,
ROLE_DEVELOPER, ROLE_QA;

   @Override
   public String getAuthority() {
      return name();
   }
}
public enum TaskStatus {
   TODO, IN_PROGRESS, REVIEW, DONE
}
@Entity
@Table(name="projects")
@Data
@AllArgsConstructor
@NoArgsConstructor
@EqualsAndHashCode(exclude = {"users",
"tasks"})
public class Project {
   @Id
   @GeneratedValue(strategy =
GenerationType.AUTO)
   @Column(name = "id")
   private Long id;

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

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

   @Column(name = "deadline")
   private LocalDate deadline;

   @ManyToMany
   @JoinTable(
         name = "project_user",
         joinColumns = @JoinColumn(name =
"project_id"),
         inverseJoinColumns = @JoinColumn(name
= "user_id")
   )
   private Set<User> users;

   @OneToMany(mappedBy = "project", cascade =
CascadeType.ALL, orphanRemoval = true)
   private Set<Task> tasks;

   @Override
   public String toString() {
      return "Project{" +
            "id=" + id +
            ", name='" + name + '\'' +
            ", description='" + description + '\'' +
            ", deadline=" + deadline +
            '}';
   }
}
@Entity
@Table(name="tasks")
@Data
@AllArgsConstructor
@NoArgsConstructor
@EqualsAndHashCode(exclude = {"project",
"assignee"})
public class Task {
   @Id
   @GeneratedValue(strategy =
GenerationType.AUTO)
   @Column(name = "id")
   private Long id;

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

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

   @Column(name = "deadline")
   private LocalDate deadline;

   @Enumerated(EnumType.STRING)
   @Column(name = "priority")
   private Priority priority;

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

   @ManyToOne(cascade =
CascadeType.REFRESH , fetch =
FetchType.LAZY)
   @JoinColumn(name = "project_id", nullable =
false)
   private Project project;

   @ManyToOne(cascade =
CascadeType.REFRESH, fetch = FetchType.LAZY)
   @JoinColumn(name = "assignee_id")
```

```
    private User assignee;

    @Column(name = "comment")
    private String comment;
}
@Entity
@Table(name = "users")
@Data
@AllArgsConstructor
@NoArgsConstructor
@EqualsAndHashCode(exclude = {"projects", "assignedTasks"})
public class User implements UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id")
    private Long id;

    @Column(name="username", unique = true)
    private String username;

    @Column(name="password", length = 1000)
    private String password;

    @Column(name="email", unique = true)
    private String email;

    @Column(name="active")
    private boolean active;

    @ElementCollection(targetClass = Role.class, fetch = FetchType.EAGER)
    @CollectionTable(name="user_role",
            joinColumns = @JoinColumn(name = "user_id"))
    @Enumerated(EnumType.STRING)
    private Set<Role> roles = new HashSet<>();

    @Column(name = "qualification")
    private Qualification qualification;

    @ManyToMany(mappedBy = "users")
    private Set<Project> projects;

    @OneToMany(mappedBy = "assignee", orphanRemoval = false)
    private Set<Task> assignedTasks;

    //security
    public boolean isAdmin() {
        return roles.contains(Role.ROLE_ADMIN);
    }
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return roles;
    }
    @Override
    public boolean isAccountNonExpired() {
        return true;
    }
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }
    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }
    @Override
    public boolean isEnabled() {
        return active;
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", username='" + username + '\'' +
                ", email='" + email + '\'' +
                ", active=" + active +
                ", qualification=" + qualification +
                '}';
    }
}
public interface ProjectRepository extends JpaRepository<Project, Long> {
    List<Project> findByName(String name);
}
public interface TaskRepository extends JpaRepository<Task, Long> {
    List<Task> findByName(String name);
}
public interface UserRepository extends JpaRepository<User, Long> {
    User findByUsername(String username);
    @Query("SELECT u FROM User u JOIN u.roles r WHERE u.qualification = :qualification AND r = :role")
    List<User> findByQualificationAndRole(@Param("qualification") Qualification qualification, @Param("role") Role role);
}
@Service
@RequiredArgsConstructor
public class CustomUserDetailService implements UserDetailsService {
    private final UserRepository userRepository;
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
```

```
        return
userRepository.findByUsername(username);
    }
}
@Service
@RequiredArgsConstructor
public class ProjectService {
    private final ProjectRepository projectRepository;

    public List<Project> listProjects(){
        return projectRepository.findAll();
    }
    public void createProject(Project project){
        projectRepository.save(project);
    }
    public void deleteProject(Long id){
        projectRepository.deleteById(id);
    }
    public Project getProjectById(Long id) {
        return
projectRepository.findById(id).orElse(null);
    }
    @Transactional
    public Project updateProject(Long id, Project
updatedProject) {
        return
projectRepository.findById(id).map(existingProject
-> {
            if (updatedProject.getName() != null &&
!updatedProject.getName().isEmpty()) {

existingProject.setName(updatedProject.getName()
);
            }
            if (updatedProject.getDescription() != null
&& !updatedProject.getDescription().isEmpty()) {

existingProject.setDescription(updatedProject.getD
escription());
            }
            if (updatedProject.getDeadline() != null) {

existingProject.setDeadline(updatedProject.getDead
line());
            }
            if (updatedProject.getUsers() != null &&
!updatedProject.getUsers().isEmpty()) {

existingProject.setUsers(updatedProject.getUsers())
;
            }
            if (updatedProject.getTasks() != null &&
!updatedProject.getTasks().isEmpty()) {

existingProject.setTasks(updatedProject.getTasks())
;
            }
            return
projectRepository.save(existingProject);
        }).orElse(null);
    }
}
@Service
@RequiredArgsConstructor
public class TaskService {
    private final TaskRepository taskRepository;

    public List<Task> listTasks(){
        return taskRepository.findAll();
    }
    public void createTask(Task task){
        taskRepository.save(task);
    }
    public void deleteTask(Long id){
        taskRepository.deleteById(id);
    }
    @Transactional
    public Task updateTask(Long id, Task
updatedTask) {
        return
taskRepository.findById(id).map(existingTask -> {
            if (updatedTask.getName() != null &&
!updatedTask.getName().isEmpty()) {

existingTask.setName(updatedTask.getName());
            }
            if (updatedTask.getDescription() != null &&
!updatedTask.getDescription().isEmpty()) {

existingTask.setDescription(updatedTask.getDescri
ption());
            }
            if (updatedTask.getDeadline() != null) {

existingTask.setDeadline(updatedTask.getDeadline
());
            }
            if (updatedTask.getPriority() != null) {

existingTask.setPriority(updatedTask.getPriority());
            }
            if (updatedTask.getStatus() != null) {

existingTask.setStatus(updatedTask.getStatus());
            }
            if (updatedTask.getAssignee() != null) {

existingTask.setAssignee(updatedTask.getAssignee
());
            }
            if (updatedTask.getComment() != null) {

existingTask.setComment(updatedTask.getComme
nt());
```

```
        }
        return taskRepository.save(existingTask);
      }).orElse(null);
  }

  public Task getTaskById(Long id) {
    return
taskRepository.findById(id).orElse(null);
  }
}
@Service
@RequiredArgsConstructor
public class UserService {
  private final UserRepository userRepository;
  private final PasswordEncoder passwordEncoder;

  public boolean createUser(User user){
    String username = user.getUsername();
    if(userRepository.findByUsername(username)
!= null) return false;
    user.setActive(true);

user.setPassword(passwordEncoder.encode(user.get
Password()));
    userRepository.save(user);
    return true;
  }

  @Transactional
  public User updateUser(Long id, User
updatedUser) {
    return
userRepository.findById(id).map(existingUser -> {
      if (updatedUser.getUsername() != null &&
!updatedUser.getUsername().isEmpty()) {

existingUser.setUsername(updatedUser.getUserna
me());
      }
      if (updatedUser.getEmail() != null &&
!updatedUser.getEmail().isEmpty()) {

existingUser.setEmail(updatedUser.getEmail());
      }
      if (updatedUser.getQualification() != null) {

existingUser.setQualification(updatedUser.getQuali
fication());
      }

existingUser.setActive(updatedUser.isActive());
      if (updatedUser.getRoles() != null) {

existingUser.setRoles(updatedUser.getRoles());
      }
      if (updatedUser.getPassword() != null &&
!updatedUser.getPassword().isEmpty()) {

existingUser.setPassword(passwordEncoder.encode
(updatedUser.getPassword()));
      }
      return userRepository.save(existingUser);
    }).orElse(null);
  }
  public void deleteUser(Long id) {
    userRepository.deleteById(id);
  }
  public User findOptimalUser(Qualification
qualification, Role role) {
    List<User> users =
userRepository.findByQualificationAndRole(qualifi
cation, role);
    return users.stream()
        .min(Comparator.comparingInt(u ->
u.getAssignedTasks().size() +
u.getProjects().size()))
        .orElse(null);
  }
  public List<User> listUsers() {
    return userRepository.findAll();
  }

  public User findByUsername(String username) {
    return
userRepository.findByUsername(username);
  }

  public User getUserById(Long id) {
    return
userRepository.findById(id).orElse(null);
  }
}
@Controller
@RequiredArgsConstructor
@PreAuthorize("hasAuthority('ROLE_ADMIN')")
public class AdminController {
  private final UserService userService;
  private final TaskService taskService;
  private final ProjectService projectService;

  @GetMapping("/admin")
  public String admin(Model model) {
    model.addAttribute("users",
userService.listUsers());
    model.addAttribute("tasks",
taskService.listTasks());
    model.addAttribute("projects",
projectService.listProjects());
    return "admin";
  }

  @PostMapping("/admin/user/delete/{id}")
  public String deleteUser(@PathVariable("id")
Long id) {
```

```
        userService.deleteUser(id);
        return "redirect:/admin";
    }

    @PostMapping("/admin/task/delete/{id}")
    public String deleteTask(@PathVariable("id")
Long id) {
        taskService.deleteTask(id);
        return "redirect:/admin";
    }

    @PostMapping("/admin/project/delete/{id}")
    public String deleteProject(@PathVariable("id")
Long id) {
        projectService.deleteProject(id);
        return "redirect:/admin";
    }

    @GetMapping("/admin/user/edit/{user}")
    public String userEdit(@PathVariable("user")
User user, Model model) {
        model.addAttribute("user", user);
        model.addAttribute("roles", Role.values());
        return "user-edit";
    }

    @PostMapping("/admin/user/edit/{id}")
    public String userUpdate(@PathVariable("id")
Long id,
                             @RequestParam("username")
String username,
                             @RequestParam("email") String
email,
                             @RequestParam("qualification")
Qualification qualification,
                             @RequestParam(value = "roles",
required = false) Set<Role> roles,
                             @RequestParam(value = "active",
required = false) Boolean active,
                             @RequestParam(value =
"password", required = false) String password) {
        User updatedUser = new User();
        updatedUser.setUsername(username);
        updatedUser.setEmail(email);
        updatedUser.setQualification(qualification);
        updatedUser.setRoles(roles != null ? roles : new
HashSet<>());
        updatedUser.setActive(active != null);
        updatedUser.setPassword(password);

        userService.updateUser(id, updatedUser);
        return "redirect:/admin";
    }

    @GetMapping("/admin/user/create")
    public String userCreate(Model model) {
        model.addAttribute("roles", Role.values());
        return "user-create";
    }

    @PostMapping("/admin/user/create")
    public String
userCreate(@RequestParam("username") String
username,
                             @RequestParam("email") String
email,
                             @RequestParam("password")
String password,
                             @RequestParam("qualification")
Qualification qualification,
                             @RequestParam(value = "roles",
required = false) Set<Role> roles,
                             @RequestParam(value = "active",
required = false) Boolean active) {
        User newUser = new User();
        newUser.setUsername(username);
        newUser.setEmail(email);
        newUser.setPassword(password);
        newUser.setQualification(qualification);
        newUser.setRoles(roles != null ? roles : new
HashSet<>());
        newUser.setActive(active != null);

        if (userService.createUser(newUser)) {
            return "redirect:/admin";
        } else {
            return
"redirect:/admin/user/create?error=true";
        }
    }

    @GetMapping("/admin/project/create")
    public String projectCreate(Model model) {
        model.addAttribute("users",
userService.listUsers());
        return "project-create";
    }

    @PostMapping("/admin/project/create")
    public String
projectCreate(@RequestParam("name") String
name,
                              @RequestParam("description")
String description,
                              @RequestParam("deadline")
LocalDate deadline,
                              @RequestParam(value = "users",
required = false) Set<User> users) {
        Project newProject = new Project();
        newProject.setName(name);
        newProject.setDescription(description);
        newProject.setDeadline(deadline);
        newProject.setUsers(users != null ? users : new
HashSet<>());
```

```
        projectService.createProject(newProject);
        return "redirect:/admin";
    }

    @GetMapping("/admin/user/optimal")
    public String showOptimalUserForm(Model
model) {
        model.addAttribute("qualifications",
Qualification.values());
        model.addAttribute("roles", Role.values());
        return "user-optimal";
    }

    @PostMapping("/admin/user/optimal")
    public                                String
findOptimalUser(@RequestParam("qualification")
Qualification qualification,
                    @RequestParam("role")    Role
role,
                    Model model) {
        User                optimalUser          =
userService.findOptimalUser(qualification, role);
        model.addAttribute("qualifications",
Qualification.values());
        model.addAttribute("roles", Role.values());
        model.addAttribute("optimalUser",
optimalUser);
        model.addAttribute("searchPerformed", true);
        return "user-optimal";
    }
}
@Controller
@RequiredArgsConstructor
public class ProjectController {
    private final ProjectService projectService;
    private final UserService userService;

    @GetMapping("/project/{id}")
    public String projectInfo(@PathVariable Long id,
Model model){

model.addAttribute("project",projectService.getProj
ectById(id));
        return "project-info";
    }
    @PostMapping("/project/create")
    public String createProject(Project project){
        projectService.createProject(project);
        return "redirect:/";
    }
    @PostMapping("/project/delete/{id}")
    public String deleteProject(@PathVariable Long
id){
        projectService.deleteProject(id);
        return "redirect:/";
    }
    @GetMapping("/project/edit/{id}")
    public String showEditForm(@PathVariable
Long id, Model model) {
        Project                project          =
projectService.getProjectById(id);
        model.addAttribute("project", project);
        model.addAttribute("users",
userService.listUsers());
        return "project-edit";
    }

    @PostMapping("/project/edit/{id}")
    public String updateProject(@PathVariable Long
id,
                    @RequestParam("name")  String
name,
                    @RequestParam("description")
String description,
                    @RequestParam("deadline")
LocalDate deadline,
                    @RequestParam(value          =
"userIds", required = false) List<Long> userIds) {
        Project updatedProject = new Project();
        updatedProject.setName(name);
        updatedProject.setDescription(description);
        updatedProject.setDeadline(deadline);

        if (userIds != null) {
            Set<User> selectedUsers = userIds.stream()
                .map(userService::getUserById)
                .filter(Objects::nonNull)
                .collect(Collectors.toSet());
            updatedProject.setUsers(selectedUsers);
        }

        projectService.updateProject(id,
updatedProject);
        return "redirect:/project/" + id;
    }
}
@Controller
@RequiredArgsConstructor
public class TaskController {
    private final TaskService taskService;
    private final UserService userService;
    private final ProjectService projectService;

    @GetMapping("/task/{id}")
    public String taskInfo(@PathVariable Long id,
Model model){

model.addAttribute("task",taskService.getTaskById
(id));
        return "task-info";
    }
    @GetMapping("/task/create")
    public String showCreateForm(Model model) {
```

```
        model.addAttribute("users",
userService.listUsers());
        model.addAttribute("projects",
projectService.listProjects());
        return "task-create";
    }
    @PostMapping("/task/create")
    public String createTask(Task task) {
        taskService.createTask(task);
        return "redirect:/admin";
    }
    @PostMapping("/task/delete/{id}")
    public String deleteTask(@PathVariable Long
id){
        taskService.deleteTask(id);
        return "redirect:/";
    }
    @GetMapping("/task/edit/{id}")
    public String editTask(@PathVariable Long id,
Task task, Model model) {
        model.addAttribute("task",
taskService.getTaskById(id));
        model.addAttribute("users",
userService.listUsers());
        model.addAttribute("projects",
projectService.listProjects());
        return "task-edit";
    }
    @PostMapping("/task/edit/{id}")
    public String updateTask(@PathVariable Long id,
                             @RequestParam("name")     String
name,
                             @RequestParam("description")
String description,
                             @RequestParam("deadline")
LocalDate deadline,
                             @RequestParam("priority")
Priority priority,
                             @RequestParam("status")
TaskStatus status,
                             @RequestParam(value
= "assignee", required = false) User assignee,
                             @RequestParam(value = "project",
required = false) Project project,
                             @RequestParam(value
= "comment", required = false) String comment) {
        Task updatedTask = new Task();
        updatedTask.setName(name);
        updatedTask.setDescription(description);
        updatedTask.setDeadline(deadline);
        updatedTask.setPriority(priority);
        updatedTask.setStatus(status);
        updatedTask.setAssignee(assignee);
        updatedTask.setProject(project);
        updatedTask.setComment(comment);

        taskService.updateTask(id, updatedTask);
        return "redirect:/task/" + id;
    }
}
@Controller
@RequiredArgsConstructor
public class UserController {
    private final UserService userService;

    @GetMapping("/login")
    public String showLoginPage() {
        return "login";
    }

    @GetMapping("/user/info")
    public String userInfo(Model model) {
        Authentication authentication =
SecurityContextHolder.getContext().getAuthenticat
ion();
        if (authentication == null ||
!(authentication.getPrincipal() instanceof
UserDetails)) {
            return "redirect:/login";
        }

        User user = (User)
authentication.getPrincipal();
        User fullUser =
userService.findByUsername(user.getUsername());

        if (fullUser != null) {
            model.addAttribute("user", fullUser);
            model.addAttribute("tasks",
fullUser.getAssignedTasks());
            model.addAttribute("projects",
fullUser.getProjects());
            return "user-info";
        }

        return "redirect:/login";
    }

    @GetMapping("/registration")
    public String registration() {
        return "registration";
    }

    @GetMapping("/")
    public String startPage() {
        return "start-page";
    }

    @PostMapping("/registration")
    public String createUser(User user, Model model)
{
        if (!userService.createUser(user)) {
            model.addAttribute("errorMessage",     "user
with name: " + user.getEmail() + " already exists");
```

```
            return "registration";
        }
        return "redirect:/login";
    }

    @GetMapping("/user/{user}")
    public String userInfo(@PathVariable("user")
User user, Model model) {
        model.addAttribute("user", user);
        model.addAttribute("tasks",
user.getAssignedTasks());
        model.addAttribute("projects",
user.getProjects());
        return "user-info";
    }
}
@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true)
@RequiredArgsConstructor
public class SecurityConfig {
    @Bean
    public SecurityFilterChain
securityFilterChain(HttpSecurity http) throws
Exception {
        http
                .authorizeHttpRequests(auth -> auth
                        .requestMatchers("/",  "/registration",
"/login", "/start-page").permitAll()
                        .anyRequest().authenticated()
                )
                .formLogin(form -> form
                        .loginPage("/login")
                        .defaultSuccessUrl("/user/info", true)
                        .permitAll()
                )
                .logout(LogoutConfigurer::permitAll
                );
        return http.build();
    }
    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguratio
n authConfig) throws Exception {
        return
authConfig.getAuthenticationManager();
    }
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder(8);
    }
}
<!DOCTYPE html>
<html>
<head>
    <title>Admin Panel</title>
    <style>
body {
    font-family: Arial, sans-serif;
    margin: 20px;
    background-color: #1a1a1a;
    color: #eaeaea;
}
h1 {
    text-align: center;
    margin-bottom: 10px;
    color: #ff4081;
}
hr {
    margin: 20px 0;
    border-color: #444;
}
.button {
    display: inline-block;
    padding: 8px 16px;
    margin-bottom: 10px;
    margin-right: 10px;
    background-color: #00e5ff;
    color: #000;
    text-decoration: none;
    border-radius: 4px;
    font-weight: bold;
}
.button:hover {
    background-color: #1de9b6;
}
table {
    width: 100%;
    border-collapse: collapse;
    margin-bottom: 30px;
    background-color: #2c2c2c;
}
th, td {
    padding: 12px;
    text-align: left;
    border: 1px solid #555;
}
th {
    background-color: #333;
    color: #ffca28;
}
form {
    display: inline;
}
h2 {
    margin-top: 40px;
    margin-bottom: 10px;
    color: #ffca28;
}
.section {
    background-color: #242424;
    padding: 20px;
    border-radius: 6px;
```

```
        box-shadow: 0 2px 4px rgba(255,255,255,0.05);
        margin-bottom: 40px;
      }
      a {
        color: #00bcd4;
      }
      a:hover {
        text-decoration: underline;
      }
      input[type="submit"] {
        background-color: #ff5252;
        color: white;
        border: none;
        padding: 6px 12px;
        border-radius: 4px;
        cursor: pointer;
      }
      input[type="submit"]:hover {
        background-color: #ff1744;
      }
      h2[style] {
        margin-top: 20px;
        font-weight: bold;
      }
    </style>
</head>
<body>
    <h1>Admin Panel</h1>
    <hr>
    <div class="section">
      <h2>Users</h2>
      <a href="/admin/user/create" class="button">Create User</a>
      <a href="/admin/user/optimal" class="button">Find Optimal User</a>
      <table>
        <thead>
          <tr>
            <th>Username</th>
            <th>Email</th>
            <th>Qualification</th>
            <th>Roles</th>
            <th>Active</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          <#list users as user>
            <tr>
              <td>${user.username}</td>
              <td>${user.email}</td>
              <td>${user.qualification}</td>
              <td>
                <#list user.roles as role>
                  ${role}<#sep>,
                </#list>
              </td>
              <td>${user.active?string('Yes', 'No')}</td>
              <td>
                <a href="/admin/user/edit/${user.id}">Edit</a>
                <a href="/user/${user.id}">View</a>
                <form action="/admin/user/delete/${user.id}" method="post">
                  <input type="hidden" name="_csrf" value="${_csrf.token}">
                  <input type="submit" value="Delete">
                </form>
              </td>
            </tr>
          </#list>
        </tbody>
      </table>
    </div>
    <div class="section">
      <h2>Projects</h2>
      <a href="/admin/project/create" class="button">Create Project</a>
      <table>
        <thead>
          <tr>
            <th>Name</th>
            <th>Description</th>
            <th>Deadline</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          <#list projects as project>
            <tr>
              <td>${project.name}</td>
              <td>${project.description}</td>
              <td>${project.deadline}</td>
              <td>
                <a href="/project/edit/${project.id}">Edit</a>
                <a href="/project/${project.id}">View</a>
                <form action="/admin/project/delete/${project.id}" method="post">
                  <input type="hidden" name="_csrf" value="${_csrf.token}">
                  <input type="submit" value="Delete">
                </form>
              </td>
            </tr>
          </#list>
```

```
        </tbody>
      </table>
    </div>
    <div class="section">
      <h2>Tasks</h2>
      <a  href="/task/create"  class="button">Create
Task</a>
      <table>
        <thead>
          <tr>
            <th>Name</th>
            <th>Description</th>
            <th>Deadline</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          <#list tasks as task>
            <tr>
              <td>${task.name}</td>
              <td>${task.description}</td>
              <td>${task.deadline}</td>
              <td>
                <a
href="/task/edit/${task.id}">Edit</a>
                <a
href="/task/${task.id}">View</a>
                <form
action="/admin/task/delete/${task.id}"
method="post">
                  <input  type="hidden"
name="_csrf" value="${_csrf.token}">
                  <input  type="submit"
value="Delete">
                </form>
              </td>
            </tr>
          </#list>
        </tbody>
      </table>
    </div>
    <#if errorMessage??>
      <h2  style="color:
#ff1744">${errorMessage}</h2>
    </#if>
</body>
</html>

<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
            background-color: #1a1a1a;
            color: #eaeaea;
        }
        h1 {
            text-align: center;
            margin-top: 40px;
            color: #ff4081;
        }
        hr {
            width: 60%;
            margin: 20px auto;
            border: none;
            border-top: 1px solid #444;
        }
        form {
            background-color: #242424;
            max-width: 400px;
            margin: 30px auto;
            padding: 30px;
            border-radius: 6px;
            box-shadow: 0 2px 4px rgba(255, 255, 255, 0.05);
        }
        form div {
            margin-bottom: 15px;
        }
        label {
            display: block;
            margin-bottom: 6px;
            font-weight: bold;
            color: #ffca28;
        }
        input[type="text"],
        input[type="password"] {
            width: 100%;
            padding: 10px;
            box-sizing: border-box;
            border: 1px solid #555;
            border-radius: 4px;
            background-color: #2c2c2c;
            color: #eaeaea;
        }
        input[type="text"]::placeholder,
        input[type="password"]::placeholder {
            color: #888;
        }
        button {
            width: 100%;
            padding: 10px;
            background-color: #00e5ff;
            border: none;
            border-radius: 4px;
            color: #000;
            font-size: 16px;
            font-weight: bold;
            cursor: pointer;
            box-shadow: 0 0 5px rgba(0, 229, 255, 0.7);
            transition: background-color 0.3s ease;
        }

        button:hover {
```

```
            background-color: #1de9b6;
            box-shadow: 0 0 10px rgba(29, 233, 182, 0.8);
        }
        .message {
            max-width: 400px;
            margin: 10px auto;
            color: #ff1744;
            text-align: center;
            font-weight: bold;
        }
        .links {
            max-width: 400px;
            margin: 20px auto;
            text-align: center;
        }
        .links a {
            color: #00bcd4;
            text-decoration: none;
        }
        .links a:hover {
            text-decoration: underline;
        }
    </style>
</head>
<body>
    <h1>Login</h1>
    <hr>
    <#if errorMessage??>
        <div class="message">
            ${errorMessage}
        </div>
    </#if>
    <#if RequestParameters.error??>
        <div class="message">
            Invalid username or password.
        </div>
    </#if>
    <form action="/login" method="post">
        <div>
            <label for="username">Username:</label>
            <input type="text" id="username" name="username" required placeholder="Enter your username">
        </div>
        <div>
            <label for="password">Password:</label>
            <input type="password" id="password" name="password" required placeholder="Enter your password">
        </div>
        <div>
            <button type="submit">Login</button>
        </div>
        <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>
    </form>
    <div class="links">
        <p>Don't have an account? <a href="/registration">Register here</a></p>
        <a href="/">Back to Main Page</a>
    </div>
</body>
</html>
<!DOCTYPE html>
<html>
<head>
    <title>Personal Management System - Registration</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #1a1a1a;
            margin: 0;
            padding: 0;
            color: #eaeaea;
        }
        h1 {
            text-align: center;
            margin-top: 40px;
            color: #ff4081;
            text-shadow: 0 0 6px #ff80ab;
        }
        hr {
            width: 60%;
            margin: 20px auto;
            border: none;
            height: 2px;
            background-color: #444;
        }
        h4 {
            text-align: center;
            color: #bbb;
            margin-top: 10px;
        }
        form {
            background-color: #242424;
            max-width: 500px;
            margin: 30px auto;
            padding: 30px;
            border-radius: 12px;
            box-shadow: 0 0 10px rgba(255, 255, 255, 0.1);
        }
        form div {
            display: flex;
            flex-direction: column;
        }
        label {
            margin-top: 15px;
            font-weight: bold;
            color: #ddd;
        }
        input[type="text"],
        input[type="email"],
        input[type="password"],
        select {
            padding: 10px;
```

```
            margin-top: 5px;
            border: 1px solid #555;
            border-radius: 6px;
            background-color: #1a1a1a;
            color: #eaeaea;
            transition: border-color 0.3s ease;
        }
        input[type="text"]:focus,
        input[type="email"]:focus,
        input[type="password"]:focus,
        select:focus {
            border-color: #ff4081;
            outline: none;
            box-shadow: 0 0 8px #ff4081;
            background-color: #222;
        }
        input[type="checkbox"] {
            margin-right: 8px;
            cursor: pointer;
            filter: brightness(0.9);
        }
        .checkbox-group {
            margin-top: 15px;
            color: #ddd;
        }
        .checkbox-group label {
            font-weight: normal;
            color: #ccc;
            margin-right: 20px;
            cursor: pointer;
        }
        input[type="submit"] {
            margin-top: 25px;
            padding: 12px;
            background-color: #00e5ff;
            color: #000;
            border: none;
            font-size: 16px;
            border-radius: 8px;
            cursor: pointer;
            box-shadow: 0 0 8px rgba(0, 229, 255, 0.7);
            font-weight: 600;
            transition: background-color 0.3s ease, box-
shadow 0.3s ease;
        }
        input[type="submit"]:hover {
            background-color: #1de9b6;
            box-shadow: 0 0 16px rgba(29, 233, 182, 0.9);
        }
        .error-message {
            color: #ff4081;
            text-align: center;
            font-weight: bold;
            margin-top: 20px;
            text-shadow: 0 0 4px #ff80ab;
        }
        .links {
            max-width: 500px;
            margin: 20px auto;
            text-align: center;
        }
        .links a {
            margin: 0 10px;
            color: #00e5ff;
            text-decoration: none;
            font-weight: 600;
            transition: color 0.3s ease;
        }
        .links a:hover {
            color: #1de9b6;
            text-decoration: underline;
        }
        @media (max-width: 480px) {
            form {
                padding: 20px 15px;
            }
            input[type="submit"] {
                width: 100%;
            }
            .checkbox-group label {
                display: block;
                margin: 10px 0;
            }
        }
    </style>
</head>
<body>
    <h1>Personal Management System</h1>
    <hr>
    <h4>Registration</h4>
    <form action="/registration" method="post">
        <div>
            <label for="username">Username:</label>
            <input type="text" name="username" required>

            <label for="email">Email:</label>
            <input type="email" name="email" required>

            <label for="password">Password:</label>
            <input type="password" name="password"
required>

            <label for="qualification">Qualification:</label>
            <select name="qualification" required>
                <option value="JUNIOR">Junior</option>
                <option value="MIDDLE">Middle</option>
                <option value="SENIOR">Senior</option>
            </select>
            <div class="checkbox-group">
                <label>Roles:</label>
                <label><input type="checkbox" name="roles"
value="ROLE_ADMIN"> Admin</label>
                <label><input type="checkbox" name="roles"
value="ROLE_TEAM_LEADER"> Team
Leader</label>
                <label><input type="checkbox" name="roles"
value="ROLE_DEVELOPER"> Developer</label>
```

```
            <label><input type="checkbox" name="roles"
value="ROLE_QA"> QA</label>
        </div>
        <input type="hidden" name="_csrf"
value="${_csrf.token}">
        <input type="submit" value="Register"/>
    </div>
  </form>
  <#if errorMessage??>
    <div class="error-
message">${errorMessage}</div>
  </#if>
  <div class="links">
    <a href="/login">Already have an account? Login
here</a>
    <a href="/">Back to Main Page</a>
  </div>
</body>
</html>
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>User Information</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      max-width: 700px;
      margin: 30px auto;
      padding: 0 20px;
      color: #eaeaea;
      background-color: #1a1a1a;
    }
    h1, h2, h3 {
      color: #ff4081;
    }
    hr {
      border: none;
      border-top: 1px solid #444;
      margin: 25px 0;
    }
    .search-bar {
      margin: 10px 0 15px 0;
    }
    .search-bar input {
      padding: 8px 10px;
      width: 100%;
      max-width: 300px;
      border: 1px solid #555;
      border-radius: 5px;
      font-size: 15px;
      background-color: #2c2c2c;
      color: #eaeaea;
    }
    ul {
      padding-left: 20px;
      margin-top: 0;
    }
    li {
      margin-bottom: 8px;
    }
    a {
      color: #00bcd4;
      text-decoration: none;
    }
    a:hover,
    a:focus {
      text-decoration: underline;
    }
    .actions {
      margin-top: 25px;
      display: flex;
      gap: 15px;
      align-items: center;
    }
    .button {
      display: inline-block;
      padding: 8px 14px;
      background-color: #00e5ff;
      color: black;
      border-radius: 6px;
      text-decoration: none;
      font-weight: 600;
      transition: background-color 0.3s ease;
    }
    .button:hover {
      background-color: #1de9b6;
    }
    button[type="submit"] {
      padding: 8px 14px;
      background-color: #ff5252;
      color: white;
      border: none;
      border-radius: 6px;
      font-weight: 600;
      cursor: pointer;
      transition: background-color 0.3s ease;
    }
    button[type="submit"]:hover {
      background-color: #ff1744;
    }
    .filter-options {
      margin: 15px 0;
      display: flex;
      gap: 10px;
    }
    .filter-options select {
      padding: 8px 10px;
      border: 1px solid #555;
      border-radius: 5px;
      font-size: 15px;
      background-color: #2c2c2c;
      color: #eaeaea;
    }
  </style>
  <script>
    function filterTasks() {
```

```
        const searchText = document.getElementById('taskSearch').value.toLowerCase();
        const selectedPriority = document.getElementById('priorityFilter').value;
        const selectedStatus = document.getElementById('statusFilter').value;
        const tasks = document.querySelectorAll('.task-item');

        tasks.forEach(task => {
          const taskName = task.getAttribute('data-name').toLowerCase();
          const taskPriority = task.getAttribute('data-priority');
          const taskStatus = task.getAttribute('data-status');

          const matchesSearch = taskName.includes(searchText);
          const matchesPriority = !selectedPriority || taskPriority === selectedPriority;
          const matchesStatus = !selectedStatus || taskStatus === selectedStatus;

          task.style.display = (matchesSearch && matchesPriority && matchesStatus) ? '' : 'none';
        });
      }

      function filterProjects() {
        const searchText = document.getElementById('projectSearch').value.toLowerCase();
        const projects = document.querySelectorAll('.project-item');
        projects.forEach(project => {
          const projectName = project.getAttribute('data-name').toLowerCase();
          project.style.display = projectName.includes(searchText) ? '' : 'none';
        });
      }
    </script>
</head>
<body>
    <h1>User Profile</h1>
    <hr>
    <section class="user-info" aria-label="User Information">
      <h2>${user.username}</h2>
      <p><strong>Email:</strong> ${user.email}</p>
      <p><strong>Qualification:</strong> ${user.qualification}</p>
      <p><strong>Status:</strong> ${user.active?string('Active', 'Inactive')}</p>

      <h3>Roles:</h3>
      <ul>
        <#list user.roles as role>
          <li>${role}</li>
        </#list>
      </ul>
    </section>
    <hr>
    <section class="user-tasks" aria-label="User Tasks">
      <h3>Assigned Tasks:</h3>
      <div class="search-bar">
        <input type="text" id="taskSearch" placeholder="Search tasks..." onkeyup="filterTasks()" aria-label="Search tasks" />
      </div>
      <div class="filter-options">
        <select id="priorityFilter" onchange="filterTasks()" aria-label="Filter by priority">
          <option value="">All Priorities</option>
          <option value="CRITICAL">Critical</option>
          <option value="HIGH">High</option>
          <option value="MEDIUM">Medium</option>
          <option value="LOW">Low</option>
        </select>
        <select id="statusFilter" onchange="filterTasks()" aria-label="Filter by status">
          <option value="">All Statuses</option>
          <option value="TODO">To Do</option>
          <option value="IN_PROGRESS">In Progress</option>
          <option value="REVIEW">Review</option>
          <option value="DONE">Done</option>
        </select>
      </div>
      <#if tasks?has_content>
        <ul>
          <#list tasks as task>
            <li class="task-item"
              data-name="${task.name}"
              data-priority="${task.priority}"
              data-status="${task.status}">
              <a href="/task/${task.id}">${task.name}</a> (Priority: ${task.priority}, Status: ${task.status})
            </li>
          </#list>
        </ul>
      <#else>
        <p>No tasks assigned.</p>
      </#if>
    </section>
    <hr>
    <section class="user-projects" aria-label="User Projects">
      <h3>Participating Projects:</h3>
      <div class="search-bar">
        <input type="text" id="projectSearch" placeholder="Search projects..." onkeyup="filterProjects()" aria-label="Search projects" />
      </div>
```

```
    <#if projects?has_content>
      <ul>
        <#list projects as project>
          <li class="project-item" data-name="${project.name}">
            <a href="/project/${project.id}">${project.name}</a>
          </li>
        </#list>
      </ul>
    <#else>
      <p>No projects.</p>
    </#if>
  </section>
  <hr>
  <div class="actions">
    <#if user.roles?seq_contains('ROLE_ADMIN')>
      <a href="/admin" class="button">Go to Admin Panel</a>
    </#if>

    <form action="/logout" method="post" style="margin:0;">
      <input type="hidden" name="_csrf" value="${_csrf.token}">
      <button type="submit" aria-label="Logout">Logout</button>
    </form>
  </div>
</body>
</html>
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Find Optimal User</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      max-width: 700px;
      margin: 40px auto;
      padding: 0 20px;
      background-color: #1a1a1a;
      color: #eaeaea;
    }
    h1, h2 {
      color: #ff4081;
      text-align: center;
    }
    form > div {
      margin-bottom: 20px;
    }
    label {
      display: block;
      font-weight: bold;
      margin-bottom: 6px;
      color: #ffca28;
    }
    select {
      width: 100%;
      max-width: 400px;
      padding: 10px;
      font-size: 15px;
      border-radius: 5px;
      border: 1px solid #555;
      background-color: #2c2c2c;
      color: #eaeaea;
    }
    button[type="submit"] {
      padding: 10px 20px;
      background-color: #00e5ff;
      color: #000;
      border: none;
      border-radius: 6px;
      font-weight: bold;
      cursor: pointer;
      transition: background-color 0.3s ease;
    }
    button[type="submit"]:hover {
      background-color: #1de9b6;
    }
    div.result {
      background-color: #242424;
      border: 1px solid #555;
      border-radius: 6px;
      padding: 20px;
      margin-top: 30px;
      box-shadow: 0 2px 4px rgba(255,255,255,0.05);
    }
    div.result p {
      margin: 8px 0;
    }
    a {
      display: inline-block;
      margin-top: 30px;
      color: #00bcd4;
      text-decoration: none;
      font-weight: bold;
    }
    a:hover {
      text-decoration: underline;
    }
    p {
      text-align: center;
      margin-top: 30px;
      color: #ff1744;
    }
  </style>
</head>
<body>
  <h1>Find Optimal User</h1>

  <form action="/admin/user/optimal" method="post" aria-label="Find optimal user form">
    <input type="hidden" name="_csrf" value="${_csrf.token}">

    <div>
```

```
        <label for="qualification">Qualification:</label>
        <select name="qualification" id="qualification" required>
          <option value="" disabled selected>Select Qualification</option>
          <#list qualifications as qualification>
            <option value="${qualification}">${qualification}</option>
          </#list>
        </select>
      </div>

      <div>
        <label for="role">Role:</label>
        <select name="role" id="role" required>
          <option value="" disabled selected>Select Role</option>
          <#list roles as role>
            <option value="${role}">${role}</option>
          </#list>
        </select>
      </div>

      <button type="submit">Find Optimal User</button>
    </form>

    <#if optimalUser??>
      <div class="result" aria-live="polite">
        <h2>Optimal User Found:</h2>
        <p><strong>Username:</strong> ${optimalUser.username}</p>
        <p><strong>Email:</strong> ${optimalUser.email}</p>
        <p><strong>Qualification:</strong> ${optimalUser.qualification}</p>
        <p><strong>Number of Tasks:</strong> ${optimalUser.assignedTasks?size}</p>
        <p><strong>Number of Projects:</strong> ${optimalUser.projects?size}</p>
      </div>
    <#elseif searchPerformed??>
      <p>No optimal user found for the selected criteria.</p>
    </#if>

    <a href="/admin" aria-label="Back to Admin Panel">← Back to Admin Panel</a>
  </body>
</html>
```