МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

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

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інтелектуальна веб-орієнтована система планування і контролю робочого часу»
здобувача групи ІН - 92 Москвін Даніїл Олександрович

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

_____ Даніїл Москвін
(підпис)

Керівник,
кандидат технічних наук, старший
викладач Артем Коробов _____
(підпис)

Суми – 2023

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

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

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-92 Москвін Даніїл Олександрович

1. Тема роботи: «Інтелектуальна веб-орієнтована система планування і контролю робочого часу»
затверджую наказом по СумДУ від *«01» червня 2023 р. № 0475-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області.
2) Моделювання та проектування. 3) Розробка веб-застосунку 4) Розробка інтелектуальної системи планування і контролю робочого часу. 5) Аналіз результатів.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

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

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області*		
2	*Моделювання та проектування*		
3	*Розробка інтелектуальної системи планування і контролю робочого часу*		
4	*Аналіз результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____
Москвін Даніїл Олександрович (підпис) Коробов Артем (підпис)

АНОТАЦІЯ

Записка: 64 стр., 29 рис., 4 додатки, 26 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена контролю та плануванню робочого часу шляхом розробки відповідних методів, моделей та інформаційної технології.

Об’єкт дослідження — процес створення планів

Мета роботи — розробка інтелектуальної системи планування і контролю робочого часу

Методи дослідження — алгоритми створення завдань та керування ними

Результати — розроблено систему, за допомогою якої можна створювати плани, завдання, надавати доступ до перегляду, яка зберігає дані у базу даних. Проведено тестування розробки на створенні цих планів та завдань.

СИСТЕМА ПЛАНУВАННЯ І КОНТРОЛЮ РОБОЧОГО ЧАСУ, JAVA, SPRING FRAMEWORK, POSTGRESQL

ЗМІСТ

ВСТУП

Актуальність. Тема кваліфікаційної роботи є актуальною, оскільки присвячена контролю та плануванню робочого часу шляхом розробки відповідних методів, моделей та інформаційної технології.

Об’єкт дослідження. Процес створення планування.

Предмет дослідження. Методологія візуалізації даних та керування ними у веб-розробці.

Гіпотеза. Візуалізацію та керування даними можна досягнути шляхом застосування інформаційної технології, що реалізує сервіс для розміщення додатків які виконують ці функції.

Новизна. Описане у даній роботі програмне рішення дозволить досягти більшої ефективності в побудові планування компанії або підприємства, дозволяючи ретельно спланувати внутрішні процеси та завдання.

Апробація матеріалів роботи. Основні результати роботи оприлюднені та обговорені на міжнародній науково-технічній конференції студентів та молодих вчених «Інформатика, математика, автоматика» (ІМА – 2023).

Структура. Дана робота складається зі вступу, аналізу предметної області, постановки задачі, моделювання та проектування, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд останніх досліджень та публікацій

На даний момент доступно кілька типів технологій, які успішно використовуються для створення веб-застосунків. Усі подібні додатки мають загальну мету – реалізувати бізнес-логіку на стороні сервера та генерувати код для клієнта. Крім того, всі ці програми мають однакову архітектуру взаємодії сервер-клієнт і загальний протокол взаємодії - HTTP.

Робота серверних застосунків відбувається в три основних етапи:

1. Запит. Клієнт ініціює запит до сервера за допомогою веб-браузера.

2. Обробка запиту, підготовка відповіді. Після отримання запиту веб - сервер оброблює запитуваний ресурс. Якщо запитується статичний ресурс, такий як HTML сторінка, малюнок, документ, ця інформація форматується для протоколу HTTP і передається клієнтові в якості відповіді. Якщо ж запитується динамічний ресурс, запит передається на обробку відповідного контейнеру web - застосунків, де і відбувається подальша робота.

3. Після створення, дані надсилаються у відповідь клієнту за допомогою протоколу HTTP. Відповідь містить дані, а також будь-які додаткові параметри, передані в HTTP-заголовках відповіді.

Java Spring один з найпопулярніших та широко використовуваних фреймворків для розробки веб-застосунків мовою програмування Java. Він надає різні частини та інструменти для більш легкої розробки веб-застосунків, забезпечуючи рівень абстракції, безпеку, керування транзакціями та багатьма іншими функціями.

Дослідники та розробники постійно вивчають та досліджують можливості використання Java Spring у веб-застосунках. Вони вдосконалюють існуючі підходи, розробляють нові методики та створюють нові рішення, щоб полегшити розробку веб-застосунків на основі Java Spring.

За останні роки було багато публікацій та досліджень, що стосуються використання Java Spring. Деякі з них можуть включати теми, такі як:

- Використання Spring Boot для швидкої розробки веб-застосунків.
- Побудова мікросервісної архітектури з використанням Spring Cloud.
- Розробка RESTful API з використанням Spring MVC.
- Інтеграція баз даних з використанням Jdbc Template.
- Розгортання веб-застосунків з використанням інструментів, таких як Docker і Kubernetes, спільно з Java Spring.

1.2 Аналіз програмних продуктів-аналогів

На даний момент часу існують багато веб-застосунків аналогів зі схожими ідеями та підходами. Вони є передовими застосунками у своєму напрямку та ідеями.

Першим прикладом буде Jira[1]. Jira - це популярний інструмент керування проектами, що розробляється компанією Atlassian. Він використовується для відстеження задач, організації проектів та спільної роботи команд у режимі реального часу. Jira дозволяє керувати процесом розробки, відстежувати проблеми, планувати релізи та багато іншого.

Мова написання Jira - це Java. Jira побудована на базі Java-фреймворку під назвою Atlassian Plugin SDK, що дозволяє розробникам створювати розширення та плагіни для Jira. Використання Java дає можливість розробникам розширювати функціональність Jira, створювати нові модулі, забезпечувати інтеграцію з іншими системами та виконувати розширені налаштування.

Jira надає багато можливостей для керування проектами та розробки програмного забезпечення. Основні функції Jira включають[1]:

1. Ви можете створювати, описувати та присвоювати задачі членам команди. Кожна задача має свій статус, пріоритет, термін виконання та інші атрибути.

2. дозволяє створювати проекти, визначати їх структуру та ієрархію. Ви можете встановлювати залежності між задачами, встановлювати терміни виконання та стежити за прогресом проекту.
3. дозволяє створювати та відстежувати проблеми, пов'язані з проектом. Ви можете прикріплювати файли, коментувати проблеми, встановлювати пріоритети та вирішувати їх у співпраці з командою.
4. можна використовувати Jira для планування релізів проекту. Вона дозволяє визначати склад задач, які мають бути виконані для певного релізу, а також встановлювати терміни виконання цих задач. Jira надає можливість відстежувати прогрес виконання задач релізу, контролювати пункти замість, робити звіти та оцінювати успішність релізу.
5. надає можливості для спільної роботи команди. Ви можете додавати коментарі до задач, обговорювати проблеми, використовувати систему повідомлень та сповіщень для спілкування з колегами.
6. має гнучкість та розширюваність завдяки плагінам та розширенням. Ви можете налаштовувати Jira під свої потреби, використовуючи різноманітні плагіни, або розробляти власні розширення для додавання специфічної функціональності.

Загалом, Jira є потужним інструментом керування проектами, який допомагає організовувати роботу команди, відстежувати прогрес та керувати задачами. Вона широко використовується в галузі розробки програмного забезпечення, але також може бути корисною для керування будь-якими проектами, де потрібна систематизація та координація робочих завдань.

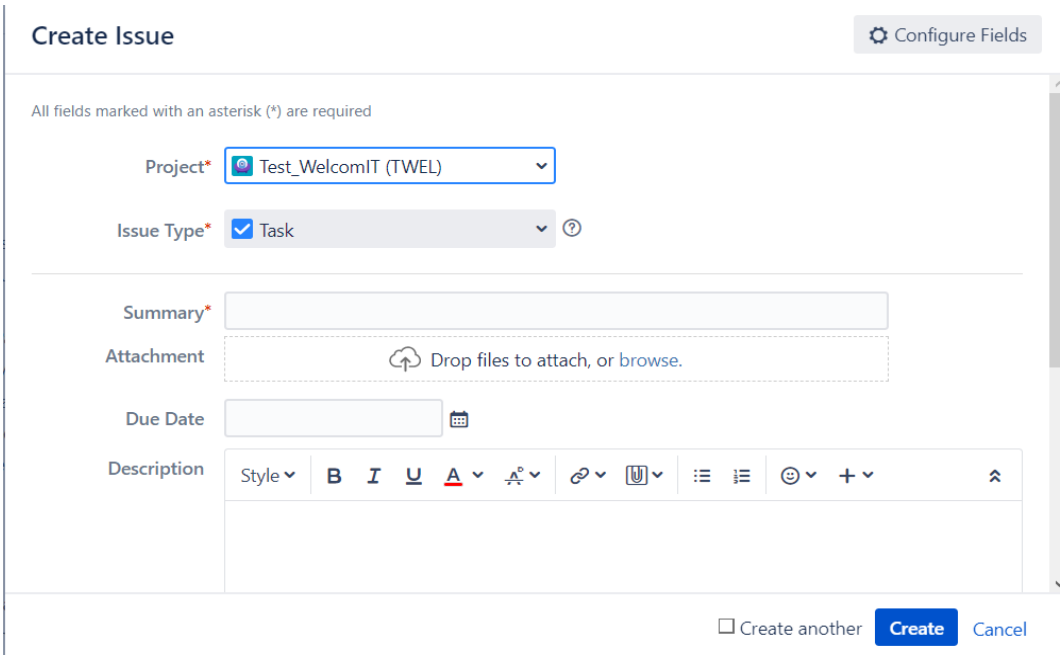

Рисунок 1.1 - створення плану в Jira

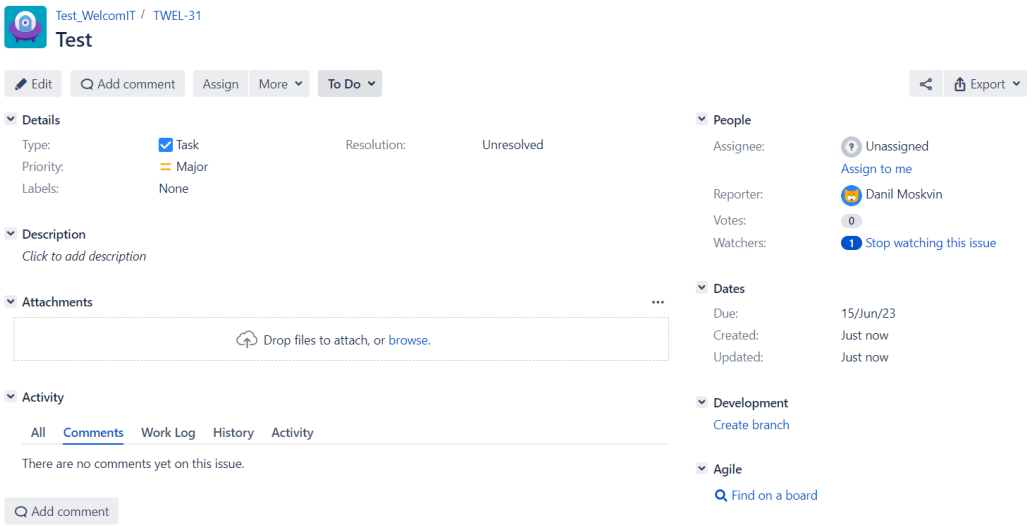

Рисунок 1.2 - створений план в Jira

Також, за приклад можна взяти Trello[2]. Trello - це інструмент керування проектами, який базується на парадигмі дошок і карток. Він дозволяє організовувати та відстежувати завдання, проекти та процеси спільної роботи команди. Trello є доволі простим у використанні та надає зручні можливості для спільної роботи.

Основні компоненти Trello:

1. Дошка є основною одиницею організації в Trello. Можна створювати різні дошки для різних проектів, команд або задач. На кожній дошці

можуть бути створені списки, які відповідають різним етапам процесу або категоріям задач.

2. Список у Trello являє собою групу пов'язаних карток на дошці. Ви можете створювати списки, такі як "В роботі", "Очікування", "Завершено" та інші, щоб організувати завдання або процеси.
3. Картки в Trello представляють окремі завдання або проекти. Можна створювати картки для кожного конкретного завдання або проекту, додавати опис, мітки, файли, дедлайни, коментарі та іншу важливу інформацію.
4. Мітки використовуються для категоризації та класифікації карток. Ви можете створювати власні мітки з різними кольорами та назвами, що допомагає швидко орієнтуватися серед великої кількості карток на дошці.
5. надає можливість додавати коментарі на картки, додавати відгуки, обговорювати деталі та спілкуватися з членами команди. Можна згадувати конкретних користувачів, використовуючи "@ім'я" для спрямування їх уваги.
6. дозволяє переміщувати картки між списками, щоб відстежувати їх прогрес. Ви можете перетягувати картки зі статусу "В роботі" до "Завершено", відображаючи виконані кроки проекту.

Trello підтримує інтеграцію з різними іншими інструментами, такими як Google Диск, GitHub, що робить його більш гнучким та розширюваним.

Загалом, Trello - це зручний та простий інструмент для організації та відстежування завдань, проектів та командної співпраці. Це популярний вибір для невеликих та середніх команд, що шукають простий та ефективний спосіб керування своїми проектами.

Дані застосунки є найефективнішими для контролювання та планування свого робочого часу, проекту та роботи в команді

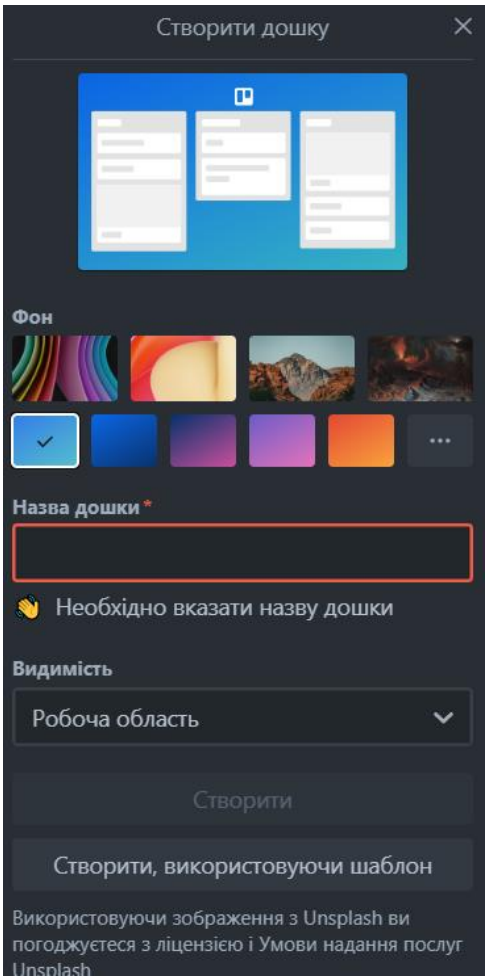

Рисунок 1.3 - створення дошки в Trello

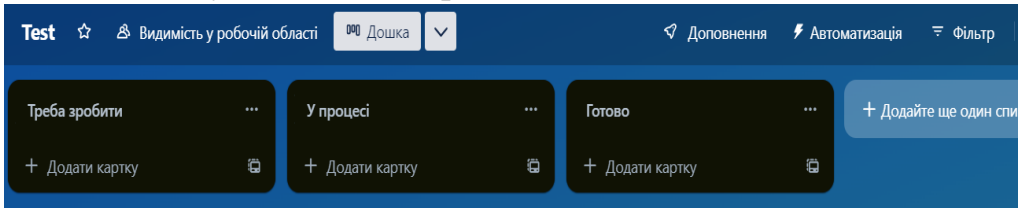

Рисунок 1.4 - створена дошка в Trello

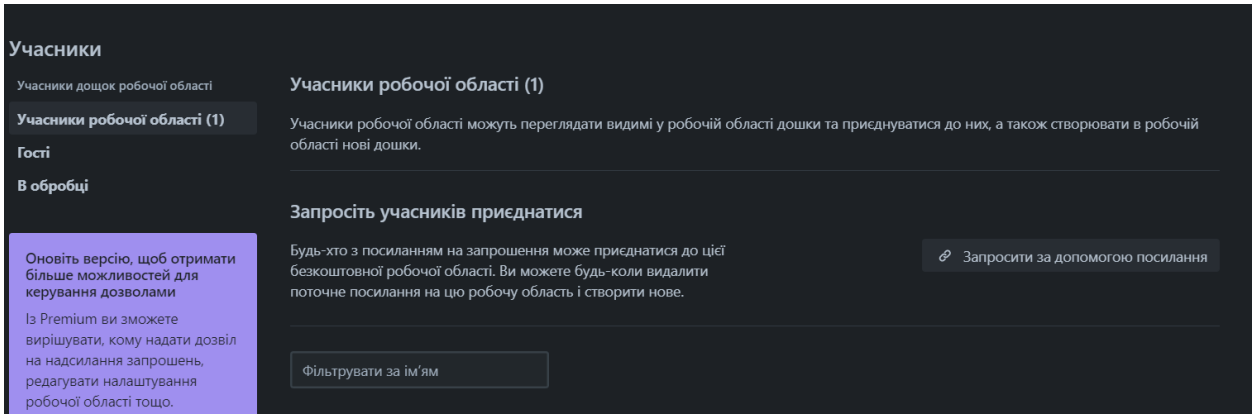

Рисунок 1.5 – додавання учасників до дошки в Trello

1.3 Постановка задачі

Мета роботи - розробити веб-застосунок для контролювання та планування свого робочого часу. Для досягнення мети потрібно вирішити такі задачі:

- Авторизація та реєстрація користувача (при реєстрації користувача потрібно ввести логін, пошту, ім’я, дату народження та пароль, при авторизації логін та пароль)
- Особиста сторінка користувача де користувач може переглянути свої дані та редагувати за необхідністю
- Створення користувачем плану роботи та додавання завдань
- Редагування завдань та видалення їх (у випадку якщо це зроблено)
- Надавання іншим користувачам ролі у своєму плані роботи (перегляд або перегляд та редагування)
- Оптимізація роботи між співробітниками

2 ВИБІР ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ

2.1 Вибір мови програмування

Основною мовою написання коду буде Java [3], за допомогою якої буде описана основна логіка програми передачі даних(Back-End)

Причини чому буде використана Java у проекті

- є однією з найпопулярніших мов програмування, використовуваних у розробці програмного забезпечення. Це означає, що у вас буде доступ до великої спільноти розробників, багато ресурсів, фреймворків та інструментів, що спрощують розробку.
- Одна з головних переваг Java полягає в тому, що вона працює на віртуальній машині Java (JVM), що дозволяє розробляти програми, які можуть виконуватись на будь-якій платформі, що підтримує JVM. Це спрощує розгортання та роботу вашого застосунку на різних операційних системах.
- має велику кількість стандартних бібліотек, фреймворків та інструментів, які допоможуть вам прискорити розробку дипломного проекту. Наприклад, фреймворки Spring і Hibernate забезпечують ефективну роботу з базами даних та веб-розробкою.
- є мовою програмування, яка добре підходить для розробки масштабованих застосунків. Вона має потужність та можливості для обробки великих обсягів даних, розподіленого обчислення та паралельного програмування.
- має вбудовану підтримку безпеки, включаючи механізми керування пам'яттю, перевірку типів та обмеження доступу до ресурсів. Це допомагає уникнути багатьох типових помилок програмування, таких як переповнення буфера, витоки пам'яті та інші вразливості.

Також для візуальної частини проекту будуть використані HTML, CSS та Thymeleaf, Thymeleaf - це шаблонізатор для розробки веб-застосунків у

Java. Він надає можливість вбудовувати HTML-код у Java-код та надавати динамічний контент на стороні сервера.

2.2 Вибір фреймворку

Так як основною мовою проекту буде Java, то слід використовувати фреймворк який найбільше підходить, а саме Spring[3].

Причини через які буде використаний саме цей фреймворк:

- використовує принцип інверсії керування та впровадження залежностей, що дозволяє ефективно керувати залежностями між компонентами застосунку.
- пропонує реалізацію шаблону проектування Model-View-Controller (MVC). Це дозволяє розбити програму на окремі компоненти, що відповідають за модель даних, представлення та логіку контролерів. Це полегшує керування станом застосунку та розділення логіки від представлення.
- надає великий набір інструментів та фреймворків для веб-розробки. Наприклад, Spring MVC дозволяє швидко розробляти веб-застосунки, роблячи задачі, такі як маршрутизація URL-адрес, обробка запитів та керування формами, простими і зручними.
- надає механізми для реалізації безпеки та автентифікації в вашому застосунку. Ви можете легко налаштувати права доступу, аутентифікацію користувачів та захистити ваші ресурси від несанкціонованого доступу.
- має просту доступність та зручність переглядання коду, є велика кількість книжок та інтернет ресурсів для отримання інформації

2.3 Вибір бази даних

Для проектування бази даних було обрано PostgreSQL[4]. PostgreSQL – це система керування базами даних, яка написана SQL мовою. Postgre пропонує надійність, швидкість та гнучкість для зберігання та керування великими обсягами структурованих та неструктурованих даних.

Чому саме було обрано PostgreSQL[4]**:**

- має великий вибір функцій та розширень які дозволяють робити складні операції з базою даних.
- може ефективно працювати з великими обсягами даних та високими навантаженнями. Postgre підтримує паралельні операції, реплікацію та кластеризацію, що дозволяє масштабувати базу даних для відповіді на зростаючі потреби вашого проекту.
- дотримується стандартів SQL, що робить його сумісним з іншими системами керування базами даних та дозволяє легко переміщувати дані між різними платформами.
- є відкритим програмним забезпеченням, що означає, що ви можете безкоштовно використовувати його, модифікувати та розповсюджувати.

2.4 Вибір інструментів для розробки

Для розробки проекту буде використано такі інструменти:

1. Intellij Idea Ultimate[5] – це вбудоване середовище розробки (IDE), яке було розроблене компанією JetBrains. Воно надає великі можливості для розробки програмного забезпечення різними мовами програмування, зокрема Java, Kotlin, і багатьох інших. IntelliJ IDEA Ultimate має досить сильні інструменти для рефакторингу, автоматичного завершення коду, аналізу помилок та інтеграції з різними засобами розробки. Воно також підтримує різні фреймворки і технології, що дозволяє розробникам доволі ефективно створювати високоякісне програмне забезпечення.

Найпопулярніші аналоги[6] IntelliJ IDEA:

- Eclipse: Безкоштовна платформа розробки Java з великою спільнотою розробників і широким набором плагінів. Eclipse також підтримує розробку для інших мов програмування.

- NetBeans: Також безкоштовна платформа розробки Java зі спеціалізованими інструментами для розробки Java-додатків. NetBeans також підтримує інші мови програмування.
- Visual Studio: Розроблене компанією Microsoft, це платформа розробки, яка підтримує різні мови програмування, включаючи Java, C#, JavaScript та інші.

Чому саме було обрано[7] Intellij Idea:

- Пропонує швидке та розумне редагування коду, автодоповнення, швидке виявлення помилок та пошук по проекту для ефективної роботи розробників.
- Має багато інструментів для поліпшення коду, тестування, аналізу та використанні з системами контролю версій. Підтримує різні фреймворки і технології для швидкого розроблення додатків.
- Має активну спільноту розробників, що надає допомогу, плагіни та розширення. Команда розробників JetBrains забезпечує регулярні оновлення та підтримку продукту.
- Існують інші спеціалізовані редактори для конкретних мов програмування, але Intellij IDEA спеціалізується на мові Java.

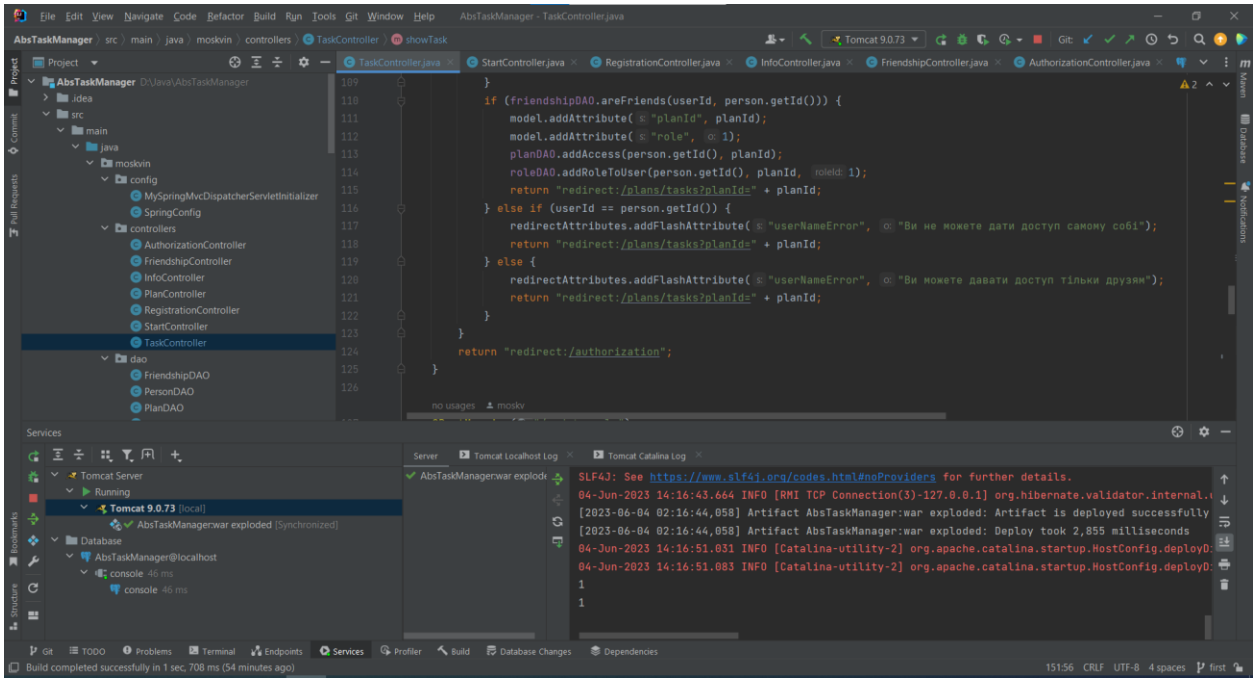

Рисунок 2.1 – Інтерфейс Intellij Idea Ultimate

2. PgAdmin 4[8] - це безкоштовна графічна програма для адміністрування та керування базами даних PostgreSQL. Вона надає зручне інтерфейсне середовище для створення, редагування та видалення баз даних, схем, таблиць та запитів. PgAdmin 4 [8] дозволяє виконувати різні завдання адміністрування, включаючи відслідковування запитів, керування ролями та правами доступу, імпорт та експорт даних.

Аналоги pgAdmin4[9]:

- DBeaver: Безкоштовна універсальна платформа для адміністрування баз даних, яка підтримує різні системи керування базами даних, включаючи PostgreSQL. DBeaver має широкий набір функцій, включаючи редагування даних, запити SQL, візуалізацію схеми та інші.
- Navicat: Комерційний інструмент для адміністрування баз даних з підтримкою PostgreSQL та інших популярних СУБД. Navicat надає зручне інтерфейсне середовище для розробки, редагування та керування базами даних, а також має розширені можливості для імпорту та експорту даних.

Чому саме було обрано pgAdmin 4[10]:

- є безкоштовним програмним забезпеченням з відкритим кодом, що дозволяє розробникам використовувати його безкоштовно та змінювати його відповідно до своїх потреб.
- розроблений спеціально для адміністрування баз даних PostgreSQL. Він надає широкий набір функцій та інструментів, що спрощують керування та моніторинг баз даних PostgreSQL.
- має інтуїтивно зрозумілий та зручний інтерфейс, що дозволяє легко навігувати та виконувати завдання адміністрування. Він надає доступ до різних функцій, таких як створення та керування об'єктами бази даних, виконання запитів SQL, перегляд та редагування даних.
- дозволяє розширювати його функціональність за допомогою розширень та плагінів. Це дає можливість налаштувати інструмент під потреби користувача та використовувати додаткові функції, які можуть забезпечити зручність та ефективність роботи з базами даних PostgreSQL.

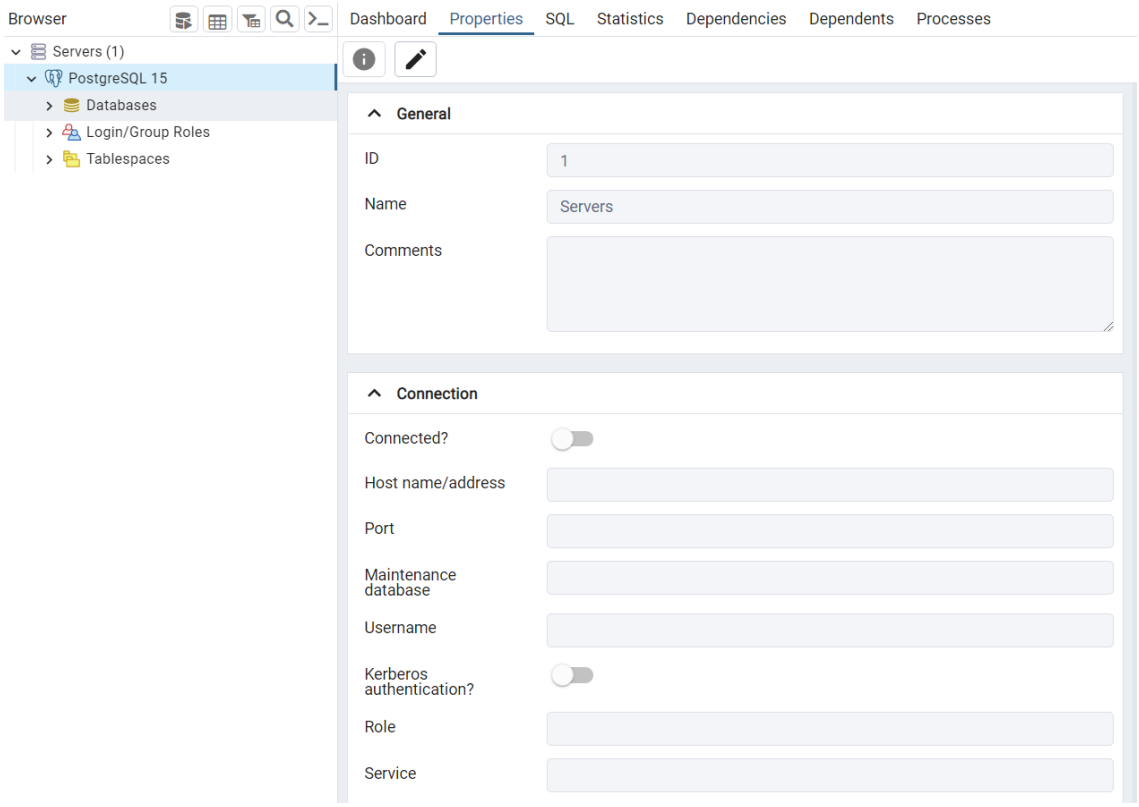

Рисунок 2.2 – Інтерфейс pgAdmin 4

3. Visual Studio Code [11] - це безкоштовне та легке інтегроване середовище розробки, розроблене компанією Microsoft. VS Code надає потужні можливості для редагування коду з підсвічуванням синтаксису, автодоповненням, вбудованими терміналами та інтеграцією зі засобами контролю версій. Це середовище підтримує багато мов програмування та платформ, а також має велику кількість розширень, що дозволяють налаштувати його під потреби розробника.

Аналоги [12] Microsoft Visual Studio Code:

- Atom: Безкоштовний текстовий редактор з відкритим вихідним кодом, відносно легкий у використанні, підтримує багато мов програмування та має багато функцій і розширень.
- Sublime Text: Комерційний текстовий редактор, відомий своєю швидкістю, елегантним інтерфейсом, підтримкою мов програмування та різними корисними функціями.

Чому саме було обрано [13] VS Code:

- є безкоштовним і має відкритий вихідний код, що дозволяє розробникам використовувати його безкоштовно, а також модифікувати та розширювати його за своїми потребами.
- надає широкий набір функцій та розширень, які полегшують процес розробки. Він підтримує багато мов програмування, має інтегровану підтримку систем керування версіями, дебагери, візуальні засоби розробки та багато іншого.
- має велику та активну спільноту розробників, яка надає підтримку, допомогу та надає власні розширення та плагіни.
- можна швидко створювати html-сторінки та у прямому часі перевіряти їх

3 РОЗРОБКА ВЕБ-ЗАСТОСУНКУ

3.1 Архітектура програмного застосунку

Розробка серверної частини:

- Реалізовано патерн MVC [14] (Model-View-Controller) де дані отримуються з моделі, яка виконує всю бізнес-логіку, передаються у контроллер та контроллер у свою чергу передає дані до відображення яке отримує клієнт
- Для взаємодії з базою даних використовується PostgreSQL [15] для передачі запитів та отримання інформацію з бази даних, для взаємодії використовуються класи DAO
- Реалізована оптимізація робочого процесу за допомогою алгоритму який перерозподіляє завдання між усіма співробітниками

Розробка клієнтської частини:

- У застосунку є сторінки які викликаються при певному запиті до серверу
- Для відображення даних використовується Thymeleaf [16] який отримує java класи або змінні які можна відобразити, редагувати або видалити
- Зроблена певна структуризація де є головні сторінки і від них є багато побічних
- Використані css-стилі для кращого відображення даних
- Зроблений доступ до сторінок, користувач повинен мати доступ до певних функцій або навіть сторінок щоб отримати інформацію з них

3.2 Програна реалізація

Під час запуску проекту відбувається перенаправлення на сторінку авторизації за допомогою контролеру. Використовується заданий порт 8080.

З’єднання з базою даних відбувається за допомогою конфігурації у файлі SpringConfig.

Всі запити оброблюються за допомогою контролерів, які викликають методи DAO класів (Data Access Object), які вже напряму взаємодіють з

базами даних. Також реалізовані сесії при авторизації, тобто, лише авторизований користувач має доступ до всіх сторінок які йому доступні, при виході з облікового запису сесія видаляється (Додаток Б2)

3.3 Використання програмного застосунку

При авторизації, або реєстрації (якщо користувач немає облікового запису) програма перенаправляє на сторінку профілю користувача який авторизувався. Після цього користувач може додати друзів, або створити план та додавати завдання у план. Також користувач може давати доступ до свого плану на редагування або перегляд іншим користувачам-друзям, які можуть редагувати та видаляти завдання (якщо роль редагування) або просто переглядати їх (якщо роль перегляд).

Редагування завдань має за мету відмітити чи виконане завдання чи ні. Також якщо у користувача є доступ до інших планів, він може їх побачити на своїй сторінці плани під надписом «Плани до яких я маю доступ».

3.4 Проектування бази даних

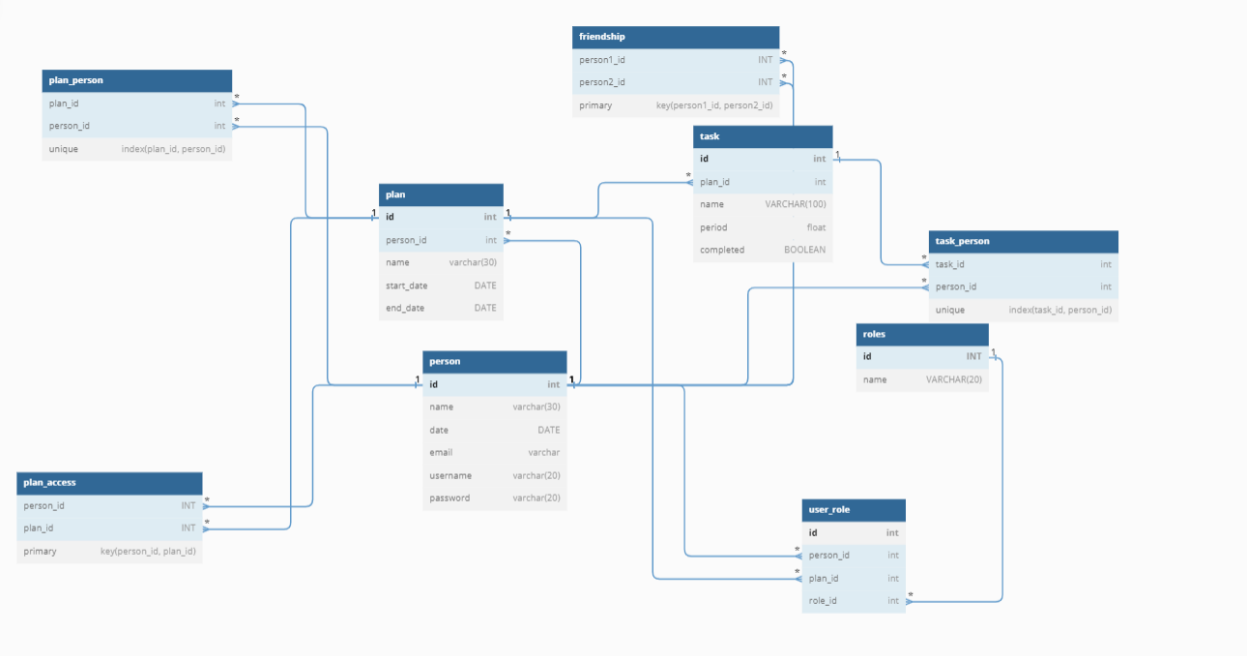

Рисунок 3.1 – діаграма бази даних

Для зв’язку користувача з базою даних використовуються класи з приміткою DAO. Під час проектування бази даних було створено 9 таблиць, які пов’язані між собою – рисунок 3.1

3.5 Основні класи

- Основні сутності – Додаток А

Person.java, Plan.Java, Task.java – ці класи це відповідні об’єкти з бази даних – додаток А1, А2, А3

- Класи контролери – Додаток Б

StartController – клас, який перенаправляє користувача на сторінку авторизації – Додаток Б1

AuthorizationController – клас, який створює сесію та керує авторизацією – Додаток Б2

RegistrationController – клас, який створює користувача у БД – Додаток Б3

InfoController – клас, який керує відображенням, редагуванням та видаленням користувача – Додаток Б4

PlanController – клас, який керує створенням, видаленням доступних планів – Додаток Б5

TaskController – клас, який керує завданнями у певному плані, а також видаленням завдань, редагуванням та видачі доступу – Додаток Б6

FriendshipController – клас, який керує дружбою між користувачами – Додаток Б7

- Класи DAO(Data Access Object) – Додаток В

PersonDAO – клас, для виконання запитів пов’язаних із класом Person – Додаток В1

PlanDAO – клас, для виконання запитів пов’язаних із класом Plan – Додаток В2

TaskDAO – клас, для виконання запитів пов’язаних із класом Task – Додаток В3

FriendshipDAO – клас, для виконання запитів пов’язаних із дружбою між користувачами – Додаток В4

RoleDAO – клас, для виконання запитів пов’язаних із видачею, зміною, видаленням ролей користувача – Додаток В5

OptimizationDAO – клас, для виконання запитів пов’язаних із оптимізацією роботи

- Клас Optimization – алгоритм розподілення робочого процесу - Додаток Г

3.6 Тестування результатів

При запуску проекту, розгортається сторінка авторизації – рисунок 3.2

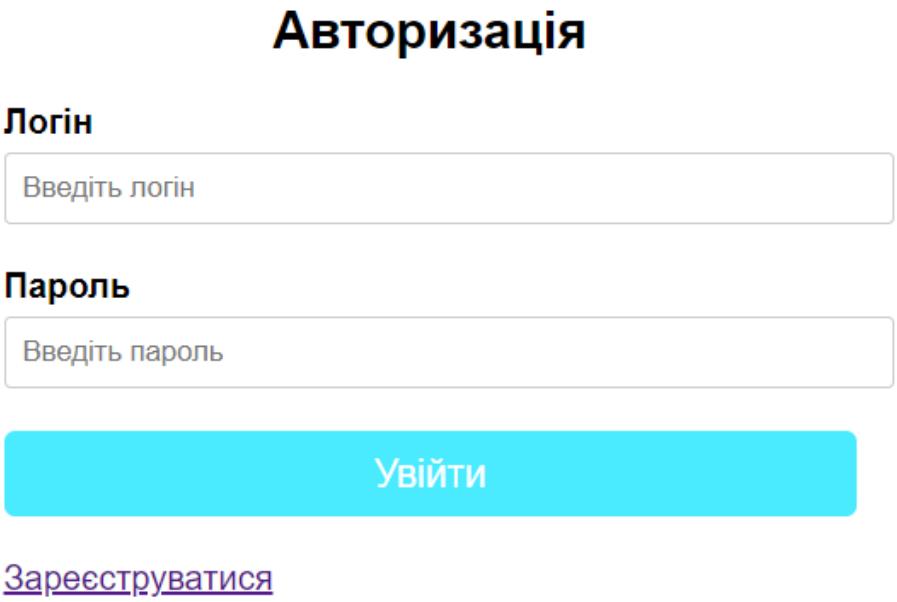

Рисунок 3.2 - авторизація у застосунку AbsManager

Так, як користувач ще не створений, натискаємо на кнопку реєстрації, та переходимо на відповідну сторінку реєстрації - рисунок

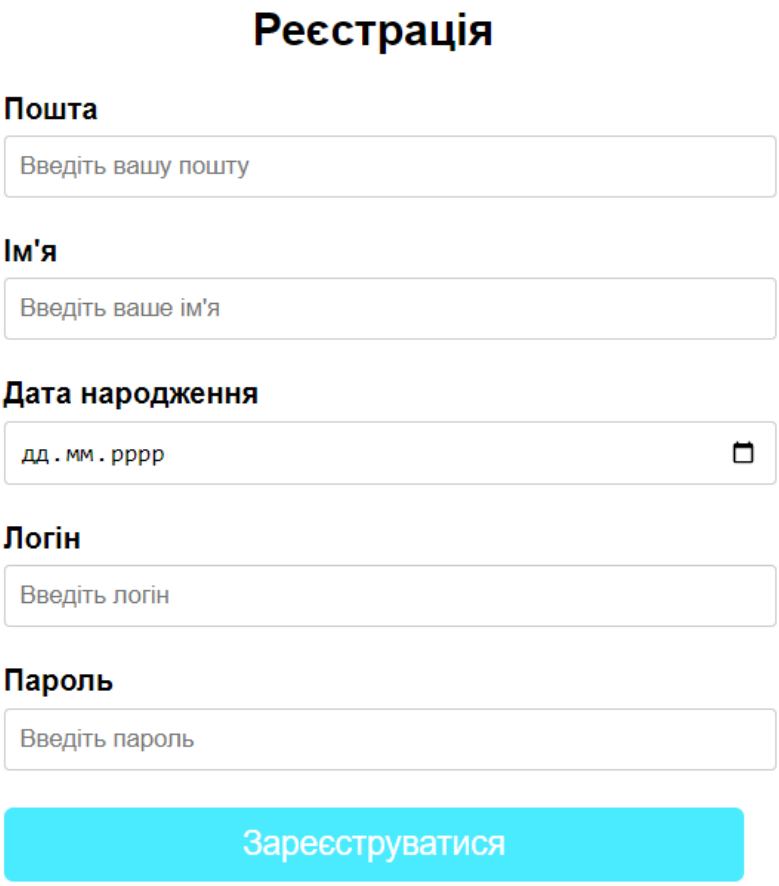

Рисунок 3.3 - реєстрація у застосунку AbsManager

Після вводу даних натискаємо зареєструватися і нас знову перекидає на сторінку авторизації – рисунок 3.2, вводимо дані(Логін та пароль) та натискаємо кнопку авторизуватись

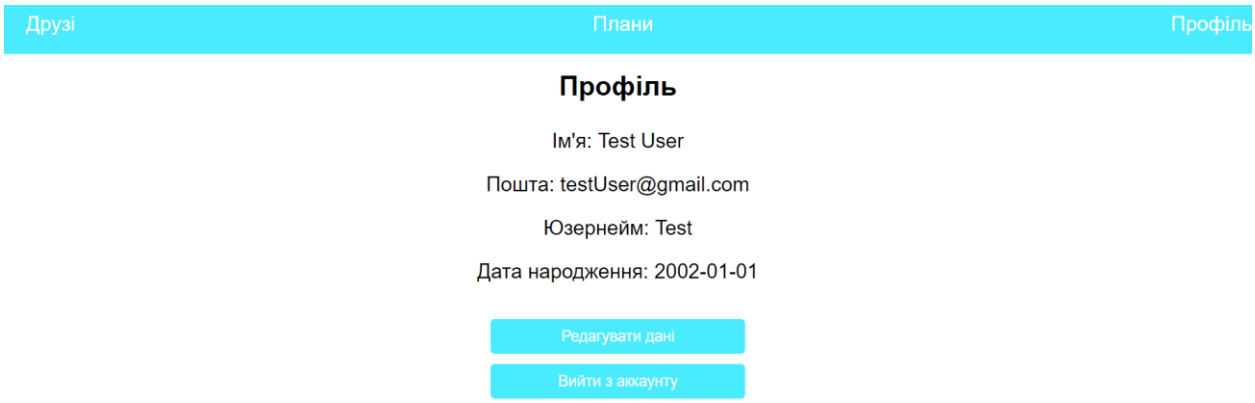

Рисунок 3.4 - сторінка користувача у застосунку AbsManager

На сторінці користувача – рисунок 3.4 будуть кнопки у верхній панелі та на самій сторінці, при натисканні на кнопку вийти з аккаунту, користувач буде перенаправлений на сторінку авторизації – рисунок 3.1 де потрібно буде або

знову ввести ті самі дані, або зареєструвати нового користувача. При натисканні на кнопку редагувати дані, буде перенаправлено на сторінку редагування даних користувача – рисунок 3.5

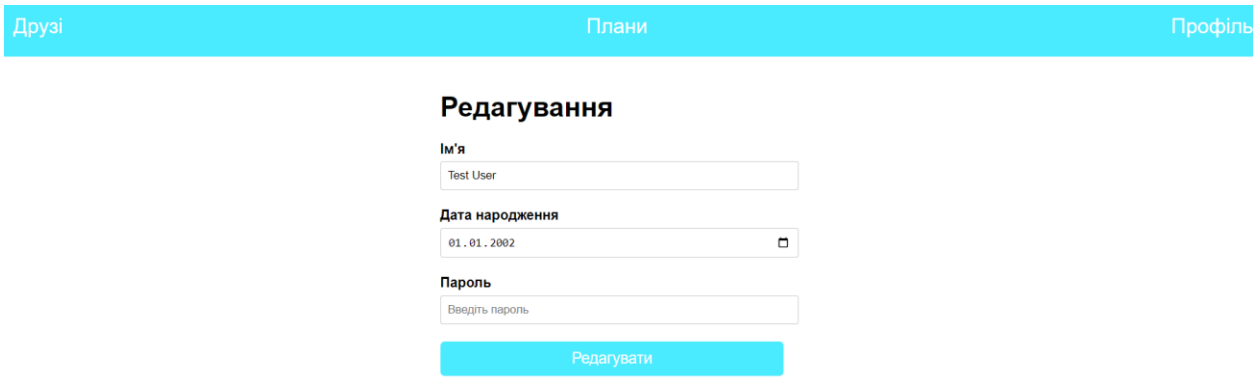

Рисунок 3.5 – сторінка редагування даних користувача

Де можна за необхідністю оновити необхідні дані, та натиснути кнопку редагувати, після цього користувач буде перенаправлений на свою сторінку користувача – рисунок 3.4, також зверху є панель з трьома кнопками, плани, друзі, профіль, профіль – це сама сторінка користувача, плани – це сторінка зі всіма планами, натиснувши на сторінку плани, користувач буде перенаправлаий на сторінку всіх планів які йому доступні – рисунок 3.6

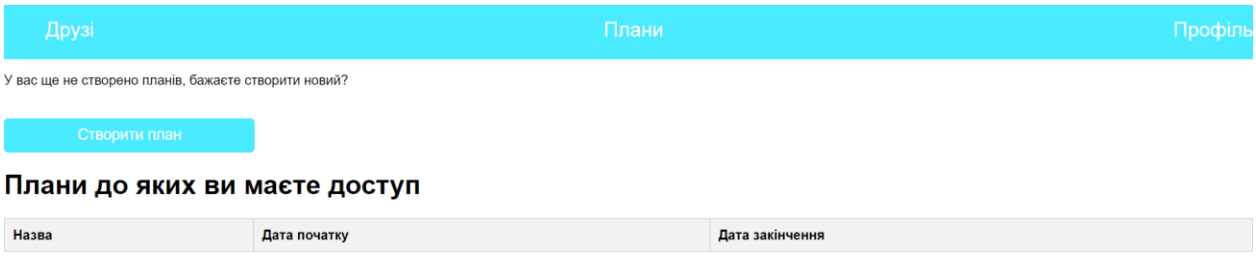

Рисунок 3.6 – сторінка планів

На цій сторінці є 2 основних пункти, плани які ви створили, та плани які дали доступ інші користувачі, так як у користувача ще не створено жодного плану, потрібно натиснути на сторінку створити план після цього буде переадресація на форму стоврення нового плану – рисунок 3.7

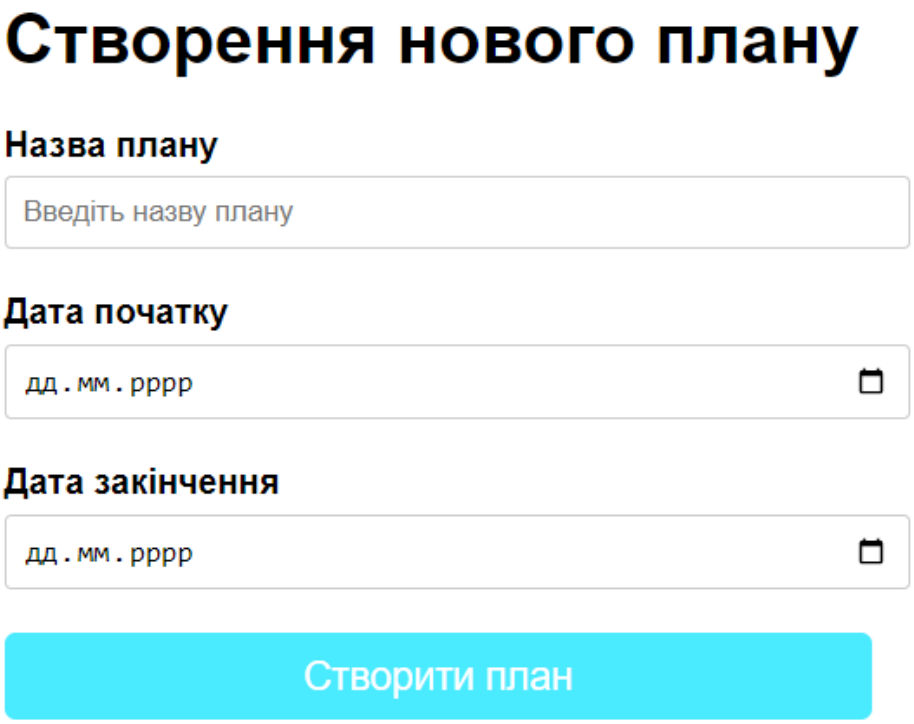

Рисунок 3.7 – форма створення нового плану

У цій формі користувачеві необхідно ввести назву плану, дату початку та дату закінчення, за умови що дата закінчення повинна бути пізнішою аніж дата початку та натиснути створити план, після цього буде перенаправлено на сторінку планів, але вже зі створеним планом – рисунок 3.8

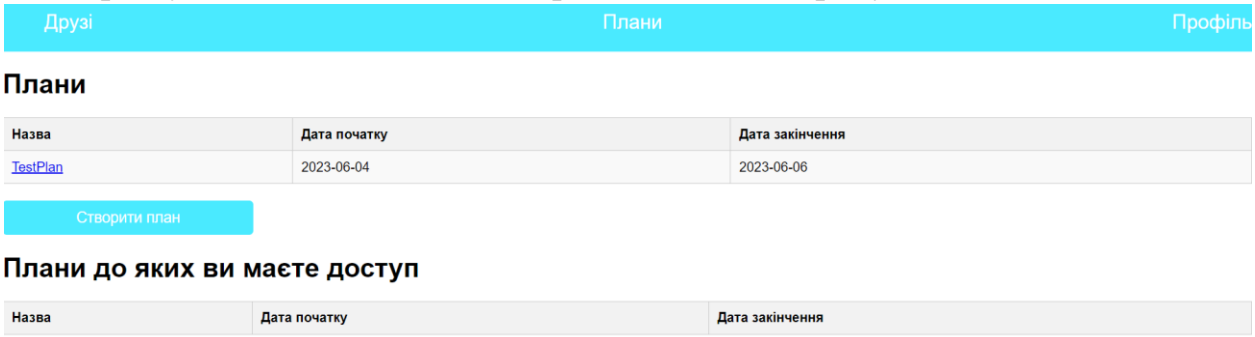

Рисунок 3.8 – сторінка планів зі створеним планом

Після цього користувач може натиснути на назву створеного плану, та перейти на його сторінку – рисунок 3.9

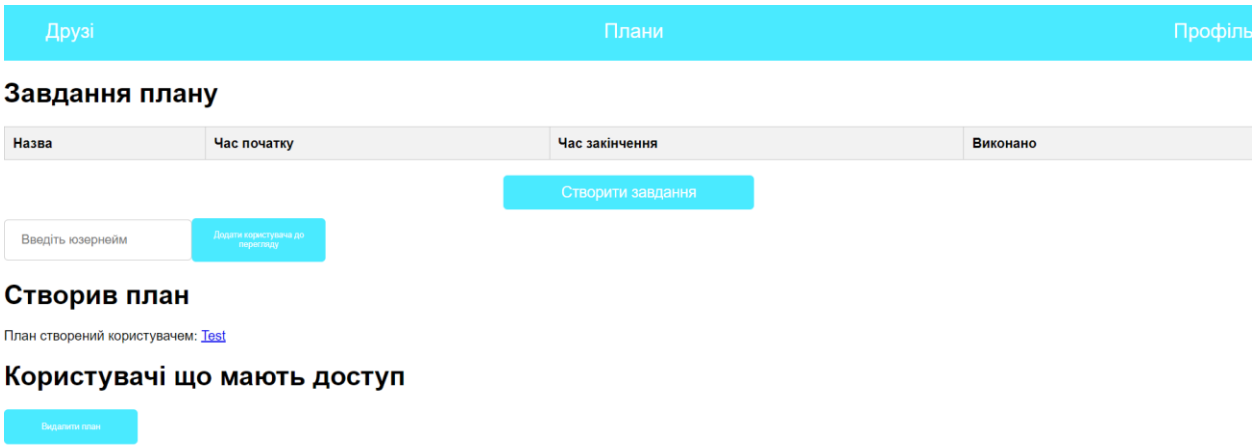

Рисунок 3.9 – сторінка плану

На сторінці плану є кнопка видалити план, натиснувши на яку, план буде видалений, та буде перенаправлення на сторінку планів – рисунок 3.6, також є кнопка створити завдання, натиснувши на яку буде переадресація на форму створення завдання – рисунок 3.10

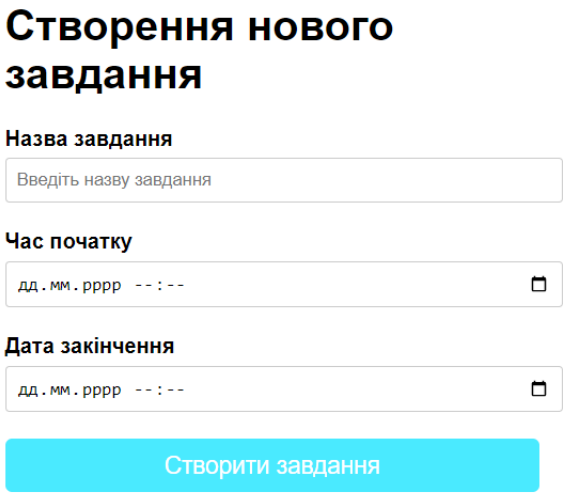

Рисунок 3.10 – форма створення завдання

У цій формі користувачу необхідно буде ввести назву, час початку та час закінчення цього завдання, час закінчення повинен бути пізніший аніж час початку та натиснути кнопку створити завдання, після цього буде виконана переадресація на сторінку планів, але вже з доданим завданням – рисунок 3.11

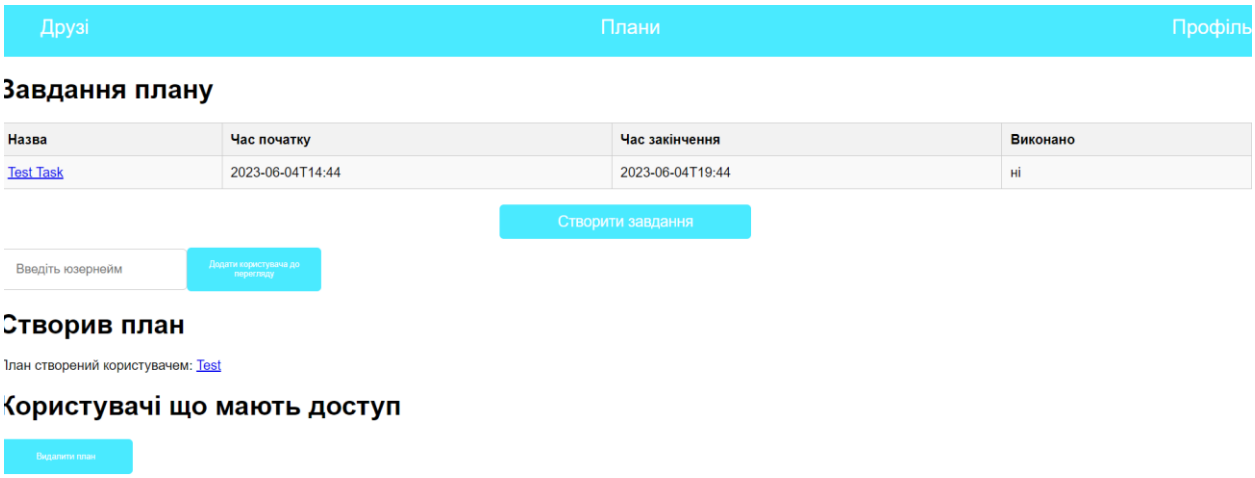

Рисунок 3.11 – сторінка плану з доданим завданням

Тепер користувач натиснувши на назву завдання, може перейти на сторінку завдання – рисунок 3.12 та помітити завданням виконаним, або видалити його

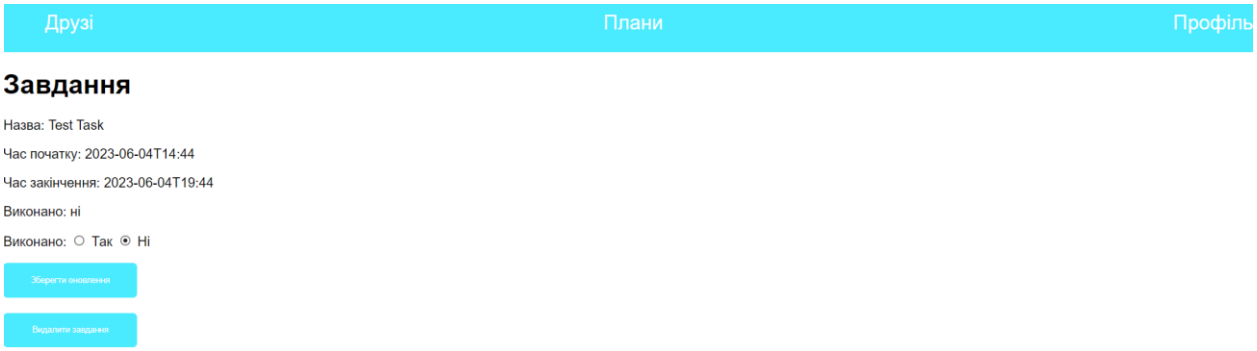

Рисунок 3.12 – сторінка завдання з можливістю видалення та редагування

Після цього користувач може повернутися на сторінку планів, та додавати або видаляти завдання.

Щоб надати доступ іншим користувачам до свого плану, ви повинні бути друзями, щоб подивитися список своїх друзів, ви можете натиснути у верхній панелі на сторінку друзі, та відкрити її – рисунок 3.13

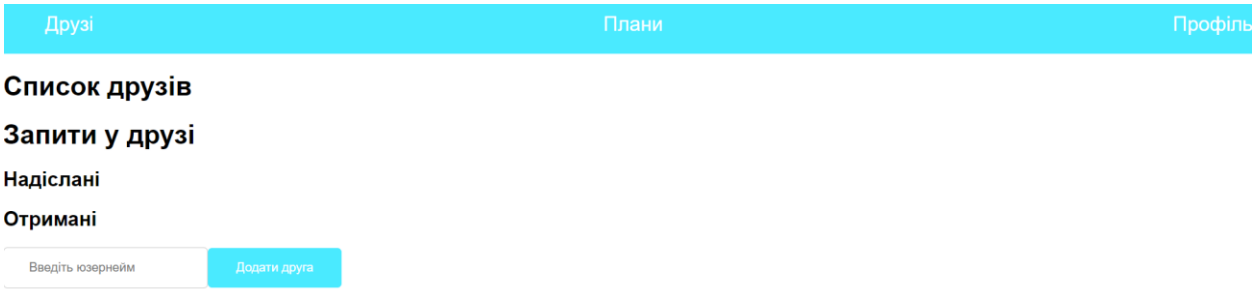

Рисунок 3.13 – сторінка друзів

Так як друзів ще немає, можна додати друга за його юзернеймом, потрібно ввести його юзернейм та натиснути додати друга

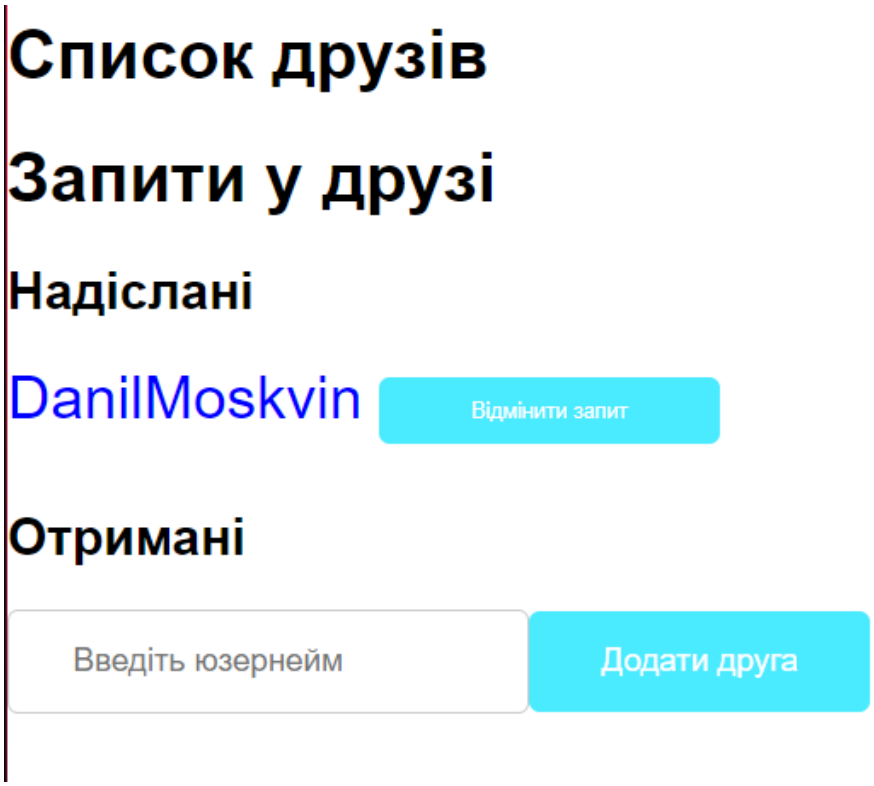

Рисунок 3.14 – сторінка друзів користувача test при надсиланні запиту

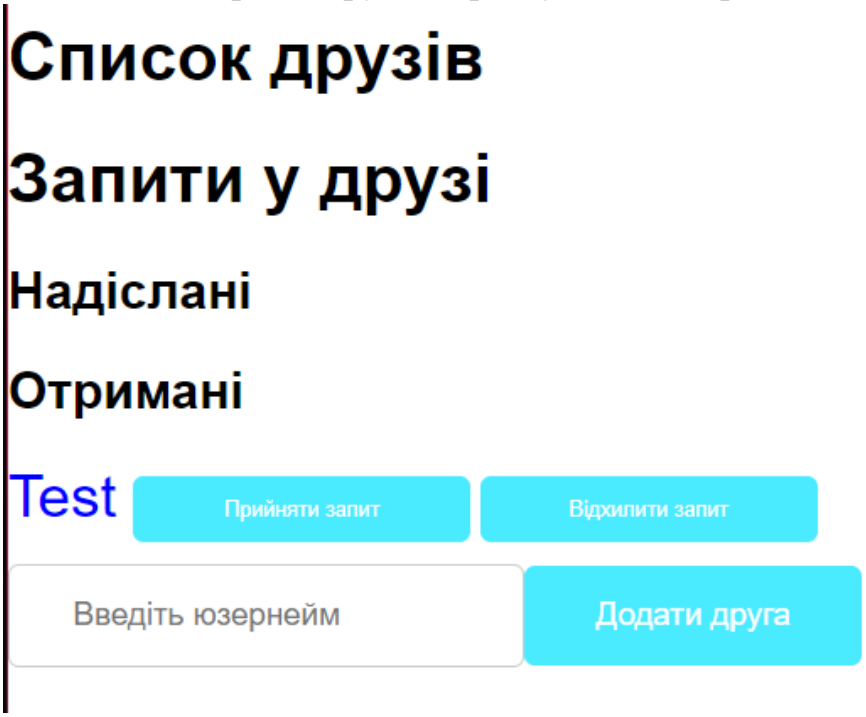

Рисунок 3.15 – сторінка друзів користувача DanilMoskvin

Користувач test може відмінити запит у друзі і тоді зникнуть ці запити у обох користувачів, користувач DanilMoskvin може прийняти або відхилити запит, якщо натиснути відхилити запит, то цей запит так само зникне, якщо натисне

прийняти, то ці користувачі стануть друзями і будуть відображатися друзями – рисунок 3.16

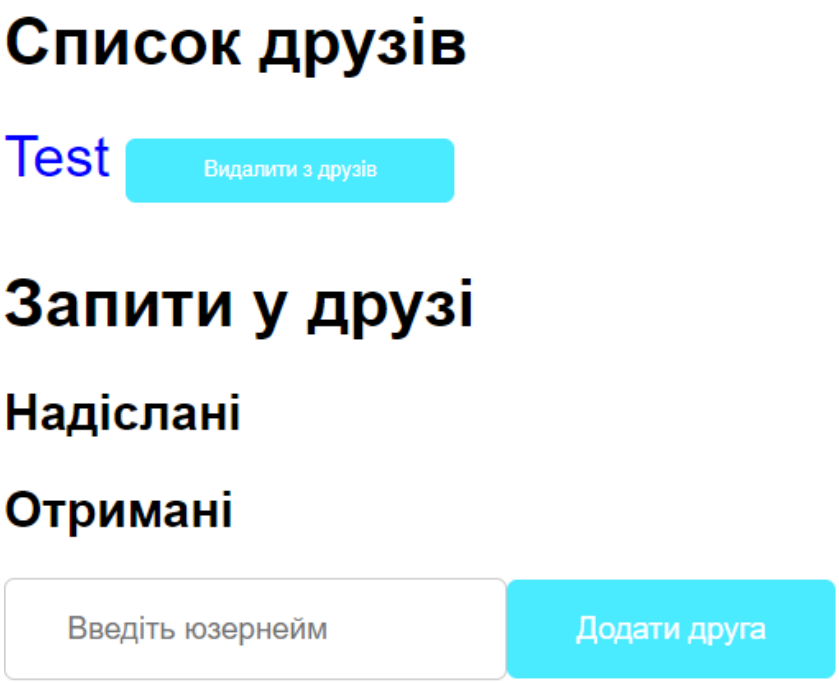

Рисунок 3.16 – сторінка друзів користувача DanilMoskvin

Тепер користувачі друзі і можуть за необхідністю видалити з друзів одне одного.

Тепер можна перейти до планів– рисунок 3.9 та додати дозвіл своєму другові до нього. Щоб додати доступ для друга, потрібно ввести його юзернейм та натиснути кнопку «Додати доступ до перегляду», після цього користувачеві буде наданий доступ до перегляду цього плану

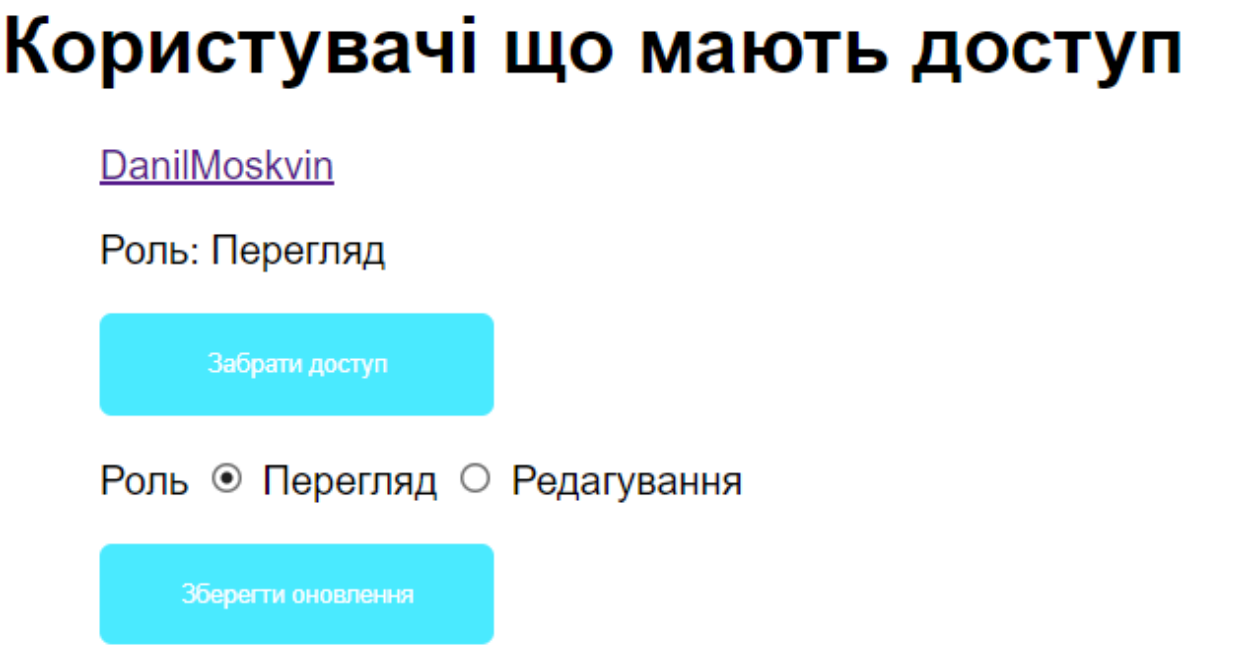

Рисунок 3.17 – вигляд доступу до плану

Тепер у плані з’явилась кнопка забрати доступ, натиснувши на яку, користувач втрачає доступ до цього плану, також з’явилась роль, по дефолту стоїть роль перегляд, можна змінити її на редагування та натиснути зберегти оновлення, якщо нам потрібно щоб користувач міг редагувати та додавати завдання до плану

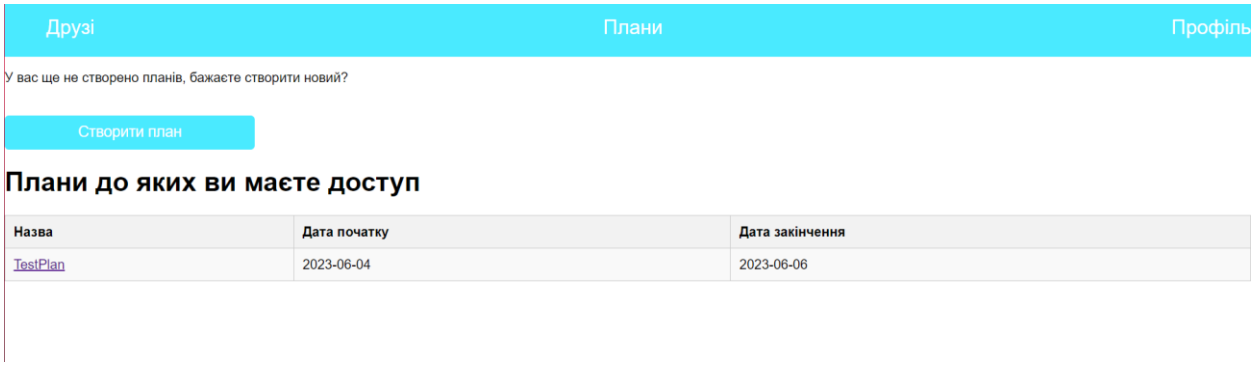

Рисунок 3.18 – вигляд планів користувача якому дали доступ

Тепер, користувач який має доступ до цього плану, може так само його відкрити та переглядати(якщо роль перегляд) або редагувати(якщо роль редагування). Користувач якому дали доступ у будь-якому випадку не може видаляти план та давати або змінювати ролі, це може робити тільки власник плану. Також у кожного користувача може бути безліч своїх планів та доступних планів, також у плані може бути доступ для багатьох користувачів яким потрібно надати доступ

Також власник плану може додати користувачів– рисунок 3.19 до роботи та система автоматично розподілить роботу між співробітниками – рисунок 3.20, 3.21

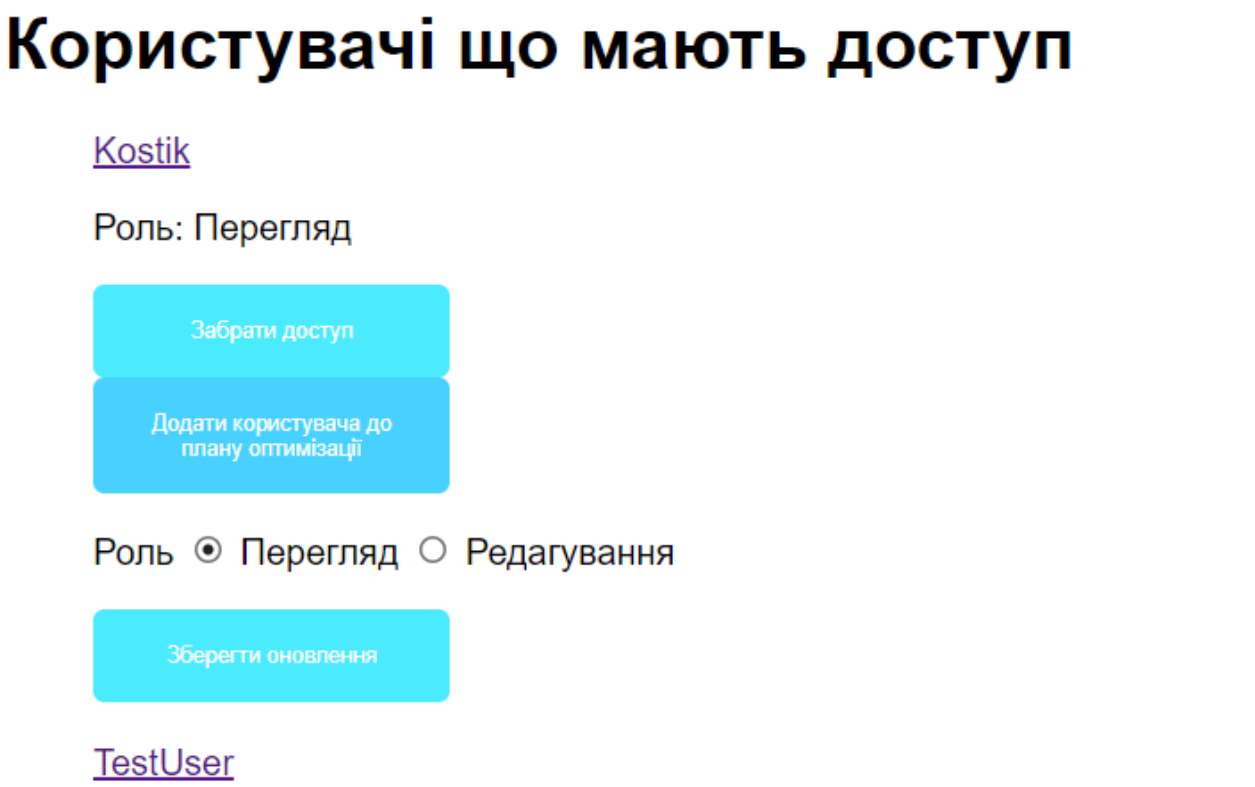

Рисунок 3.19 – Додавання користувача до плану оптимізації

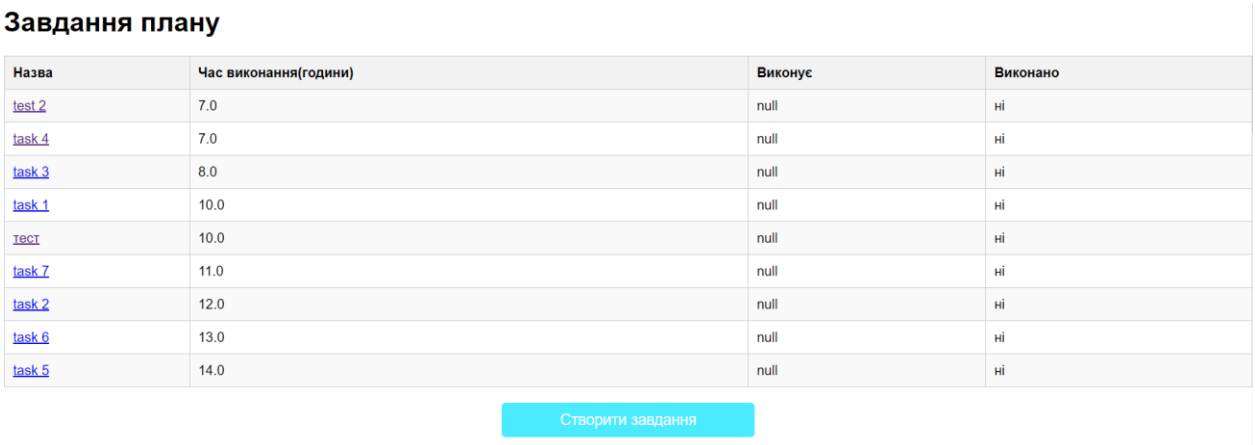

Рисунок 3.20 – вигляд таблиці завдань до оптимізації

Завдання плану

Назва	Час виконання(години)	Виконує	Виконано
test 2	7.0	Test	ні
task 4	7.0	Kostik	ні
task 3	8.0	TestUser	ні
task 1	10.0	TestUser	ні
тест	10.0	Kostik	ні
task 7	11.0	Test	ні
task 2	12.0	Test	ні
task 6	13.0	TestUser	ні
task 5	14.0	Kostik	ні

Рисунок 3.21 – вигляд таблиці завдань після оптимізації

Тепер власник плану може або видаляти завдання, або додавати нових користувачів до плану, або видаляти користувачів з плану оптимізації та система буде перерозподіляти це між користувачами. Також користувачі яким видано завдання можуть позначити завдання як виконане, або видалити його – рисунок 3.22.

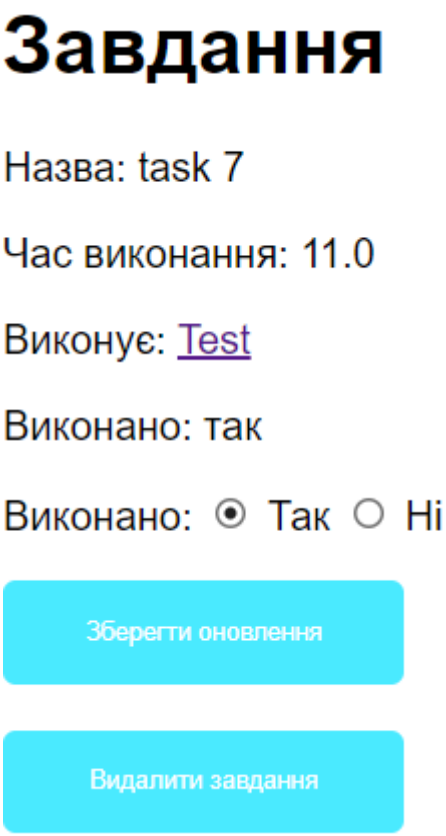

Рисунок 3.22 – сторінка завдання користувача Test

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було виконано основне завдання, а саме реалізована інтелектуальна система планування та контролю робочого часу.

Спочатку, був виконаний аналіз продуктів-аналогів, що був основою мого застосунку.

Для програмної реалізації використовувався фреймворк Spring, а саме патерн Spring MVC(Model-View-Controller).

Для візуалізації проекту були використані Html сторінки з css стилями.

У майбутньому, даний проект можна розширити, додати покращений функціонал та покращити візуальне відтворення для зручності у користуванні.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jira Software - Features | Atlassian. *Atlassian*. URL: https://www.atlassian.com/software/jira/features (дата звернення: 04.04.2023).
2. Trello Guides: Help Getting Started With Trello | Trello. *Manage Your Team's Projects From Anywhere | Trello*. URL: https://trello.com/guide (дата звернення: 04.04.2023).
3. JAVA І SPRING FRAMEWORK: РОЗРОБКА СЕРВІСІВ І ДОДАТКІВ. avada-media. URL: https://avada-media.ua/ua/services/java-and-spring-framework/ (дата звернення: 20.04.2023).
4. Douglas K., Douglas S. PostgreSQL. Sams, 2003. 816 с (дата звернення: 12.04.2023).
5. IntelliJ IDEA – the Leading Java and Kotlin IDE. JetBrains. URL: https://www.jetbrains.com/idea/ (дата звернення: 10.04.2023).
6. Top 10 IntelliJ IDEA Alternatives & Competitors. *G2*. (дата звернення: 13.04.2023).
7. The Total Economic Impact™ of IntelliJ IDEA. *JetBrains: Developer Tools for Professionals and Teams*. URL: https://www.jetbrains.com/lp/intellijidea-forrester-tei/ (дата звернення: 14.04.2023).
8. Download. *pgAdmin - PostgreSQL Tools*. URL: https://www.pgadmin.org/download/ (дата звернення: 18.04.2023).
9. Slant - 16 best alternatives to pgAdmin 4 as of 2023. *Slant*. URL: https://www.slant.co/options/208/alternatives/~pgadmin-4-alternatives (дата звернення: 18.04.2023).
10. What is pgAdmin? *Software Development & IT Service Provider â□□ Adservio*. URL: https://www.adservio.fr/post/what-is-pgadmin (дата звернення: 25.05.2023).

11. Microsoft. Visual Studio Code - Code Editing. Redefined. *Visual Studio Code - Code Editing. Redefined*. URL: https://code.visualstudio.com (дата звернення: 14.04.2023).
12. Top 10 Visual Studio Code Alternatives & Competitors. *g2*. URL: https://www.g2.com/products/visual-studio-code/competitors/alternatives (дата звернення: 14.04.2023).
13. What is Visual Studio Code? | Features And Advantages | Scope & Career. *EDUCBA*. URL: https://www.educba.com/what-is-visual-studio-code/ (дата звернення: 13.04.2023).
14. Розділяй та володарюй: що таке патерни MVC і MVP, та як їх використовувати. *Highload.today - медиа для разработчиков*. URL: https://highload.today/uk/blogs/shho-take-mvc-ta-mvp-patterni/ (дата звернення: 02.05.2023).
15. PostgreSQL: Downloads. PostgreSQL: The world's most advanced open source database. URL: https://www.postgresql.org/download/ (дата звернення: 20.04.2023).
16. Thymeleaf. *Thymeleaf*. URL: https://www.thymeleaf.org (дата звернення: 20.04.2023).
17. Що таке Патерн? Refactoring and Design Patterns. URL: https://refactoring.guru/uk/design-patterns/what-is-pattern (дата звернення: 22.04.2023).
18. IT-спеціаліст: професія майбутнього. URL: https://business.rayon.in.ua/news/420323-it-spetsialist-profesiya-maybutnogo (дата звернення: 05.04.2023).
19. Kumar A. Spring Cloud Fundamentals: With Java Codes. Independently Published, 2018. (дата звернення: 08.04.2023).
20. Pro Java™ EE Spring Patterns. Berkeley, CA: Apress, 2008. URL: https://doi.org/10.1007/978-1-4302-1010-8 (дата звернення: 04.05.2023).

21. Bretet A. Spring MVC cookbook: Over 40 recipes for creating cloud-ready Java web applications with Spring MVC. Birmingham: Packt Publishing, 2016(дата звернення: 03.05.2023).
22. Professional Java Development with the Spring Framework / R. Johnson та ін. Wiley & Sons, Incorporated, John, 2007(дата звернення: 29.04.2023).
23. Heidari A., Heidari B. Web Design Using Java. Bashir, 2004. 400 с(дата звернення: 26.04.2023)
24. Kayal D. Pro Java EE Spring Patterns: Best Practices and Design Strategies Implementing Java EE Patterns with the Spring Framework. Springer London, Limited, 2008(дата звернення: 22.04.2023).
25. Java Frameworks and Components. Cambridge University Press, 2003(дата звернення: 15.03.2023).
26. Späth P. Beginning Java MVC 1. 0: Model View Controller Development to Build Web, Cloud, and Microservices Applications. Apress L. P., 2020(дата звернення: 27.03.2023).

ДОДАТОК А

Класи сутності

А1 Person.java

```
public class Person {
   private int id;
   @Pattern(regexp = "[A-Z]\\w+ [A-Z]\\w+", message = "Ім'я повинно бути у форматі Name Surname")
   private String name;
   @DateTimeFormat(pattern = "yyyy-MM-dd")
   private Date date;
   @Email(message = "Пошта має бути існуючою")
   private String email;
   private String username;
   private String password;

   public Person() {
   }

   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public Date getDate() {
      return date;
   }

   public void setDate(Date date) {
      this.date = date;
   }

   public String getEmail() {
      return email;
```

```
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public int getUserIdByUsername(String username){
        return id;
    }
}
```

A2 Plan.Java

```
public class Plan {
    private int id;
    private int person_id;
    @NotBlank(message = "Назва не може бути порожньою")
    private String name;
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    @NotNull(message = "Дата початку не може бути порожньою")
    private Date start_date;
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    @NotNull(message = "Дата початку не може бути порожньою")
    private Date end_date;

    public Plan() {
    }

    public int getId() {
        return id;
    }
```

```
    public void setId(int id) {
        this.id = id;
    }

    public int getPerson_id() {
        return person_id;
    }

    public void setPerson_id(int person_id) {
        this.person_id = person_id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Date getStart_date() {
        return start_date;
    }

    public void setStart_date(Date start_date) {
        this.start_date = start_date;
    }

    public Date getEnd_date() {
        return end_date;
    }

    public void setEnd_date(Date end_date) {
        this.end_date = end_date;
    }
}
```

А3 Task.java

```
public class Task {
    private int id;
    private int plan_id;
    @NotBlank(message = "Назва не може бути порожньою")
    private String name;
    @Min(value = 1, message = "Час на виконання повинен бути мінімум година")
    private float period;
```

```
private boolean completed;

private Person person;

public Task() {
}

public int getId() {
   return id;
}

public void setId(int id) {
   this.id = id;
}

public int getPlan_id() {
   return plan_id;
}

public void setPlan_id(int plan_id) {
   this.plan_id = plan_id;
}

public String getName() {
   return name;
}

public void setName(String name) {
   this.name = name;
}

public boolean isCompleted() {
   return completed;
}

public void setCompleted(boolean completed) {
   this.completed = completed;
}

public float getPeriod() {
   return period;
}

public void setPeriod(float period) {
   this.period = period;
```

```
    }

    public Person getPerson() {
        return person;
    }

    public void setPerson(Person person) {
        this.person = person;
    }
}
```

ДОДАТОК Б

Класи контролери

Б1 StartController.java

```
@Controller
public class StartController {
   @RequestMapping("/")
   public String home() {
      return "redirect:/authorization";
   }
}
```

Б2 AuthorizationController.java

```
@Controller
@RequestMapping("/authorization")
public class AuthorizationController {
   private final PersonDAO personDAO;

   public AuthorizationController(PersonDAO personDAO) {
      this.personDAO = personDAO;
   }

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

   @PostMapping
   public String login(@RequestParam("username") String username,
                       @RequestParam("password") String password,
                       RedirectAttributes redirectAttributes, HttpSession session) {
      Person person = personDAO.findByUsername(username);
      if (person != null && person.getPassword().equals(password)) {
         session.setAttribute("userId", person.getId());
         return "redirect:/info";
      } else {
         redirectAttributes.addFlashAttribute("error", "Невірний логін або пароль");
         return "redirect:/authorization";
      }
   }

   @PostMapping("/logout")
   public String logout(HttpSession session) {
      session.invalidate();
      return "redirect:/authorization";
   }
```

```
}
```

Б3 RegistrationController.java

```
@Controller
@RequestMapping("/registration")
public class RegistrationController {
    private final PersonDAO personDAO;
    private final PersonValidator personValidator;

    public RegistrationController(PersonDAO personDAO, PersonValidator personValidator) {
        this.personDAO = personDAO;
        this.personValidator = personValidator;
    }

    @GetMapping
    public String showRegistrationForm(Model model) {
        model.addAttribute("person", new Person());
        return "registration/registration";
    }

    @PostMapping
    public String register(@ModelAttribute("person") @Valid Person person,
                           BindingResult bindingResult){
        personValidator.validate(person, bindingResult);
        if(bindingResult.hasErrors()){
            return "registration/registration";
        }
        personDAO.save(person);
        return "redirect:/authorization";
    }
}
```

Б4 InfoController.java

```
@Controller
@RequestMapping("/info")
public class InfoController {
    private final PersonDAO personDAO;
    private final FriendshipDAO friendshipDAO;

    public InfoController(PersonDAO personDAO, FriendshipDAO friendshipDAO) {
        this.personDAO = personDAO;
        this.friendshipDAO = friendshipDAO;
    }

    @GetMapping
    public String showInfo(HttpSession session, Model model) {
```

```
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            Person person = personDAO.findById(userId);
            if (person != null) {
                model.addAttribute("person", person);
                return "info/userProfile";
            }
        }
        return "redirect:/authorization";
    }

    @GetMapping("/editProfile")
    public String editProfile(Model model, HttpSession session) {
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            Person person = personDAO.show(userId);
            if (person != null) {
                model.addAttribute("person", person);
                return "info/editProfile";
            }
        }
        return "redirect:/authorization";
    }

    @GetMapping("/{id}")
    public String showOtherPerson(@PathVariable("id") int id, Model model, HttpSession
session){
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            Person person = personDAO.show(id);

            if (person.getId() == userId) {
                model.addAttribute("ownProfile", true);
                model.addAttribute("friendshipStatus", "Ви");
            } else {
                model.addAttribute("ownProfile", false);
                if (friendshipDAO.areFriends(userId, id)) {
                    model.addAttribute("friendshipStatus", "Друзі");
                } else if (friendshipDAO.hasPendingRequest(userId, id)) {
                    model.addAttribute("friendshipStatus", "Запит надіслано");
                } else {
                    model.addAttribute("friendshipStatus", "Не друзі");
                }
            }
            model.addAttribute("person", person);
```

```
            return "info/otherUserProfile";
        }
        return "redirect:/authorization";
    }

    @PatchMapping
    public String updateProfile(@ModelAttribute("person") @Valid Person person,
                                BindingResult bindingResult,
                                HttpSession session) {
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            if (bindingResult.hasErrors()) {
                return "info/editProfile";
            }
            personDAO.updateData(userId, person);
            return "redirect:/info";
        }
        return "redirect:/authorization";
    }
}
```

Б5 PlanController.java

```
@Controller
public class PlanController {
    private final PersonDAO personDAO;
    private final PlanDAO planDAO;
    private final RoleDAO roleDAO;
    private final PlanValidator planValidator;
    private final OptimizationDAO optimizationDAO;

    public PlanController(PersonDAO personDAO, PlanDAO planDAO, RoleDAO roleDAO,
PlanValidator planValidator, OptimizationDAO optimizationDAO) {
        this.personDAO = personDAO;
        this.planDAO = planDAO;
        this.roleDAO = roleDAO;
        this.planValidator = planValidator;
        this.optimizationDAO = optimizationDAO;
    }

    @GetMapping("/plans")
    public String viewAllPlans(Model model, HttpSession session){
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            model.addAttribute("plans", planDAO.getPlansByPersonId(userId));
            model.addAttribute("access_plans", planDAO.getPlansAccessByPersonId(userId));
```

```
            return "menu/plans";
        }
        return "redirect:/authorization";
    }

    @GetMapping("/plan/new")
    public String showCreatePlanForm(Model model, HttpSession session) {
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            model.addAttribute("plan", new Plan());
            return "menu/newPlan";
        }
        return "redirect:/authorization";
    }

    @PostMapping("/plan/new")
    public String createPlan(@ModelAttribute("plan") @Valid Plan plan, BindingResult
bindingResult, HttpSession session) {
        planValidator.validate(plan, bindingResult);
        if (bindingResult.hasErrors()) {
            return "menu/newPlan";
        }
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            Person person = personDAO.findById(userId);
            plan.setPerson_id(person.getId());
            planDAO.save(plan);
            return "redirect:/plans";
        }
        return "redirect:/authorization";
    }

    @DeleteMapping("/plan/delete")
    public String deletePlan(@RequestParam("planId") int planId, Model model, HttpSession
session){
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            planDAO.deletePlan(planId);
            planDAO.deleteAccess(planId);
            return "redirect:/plans";
        }
        return "redirect:/authorization";
    }

    @DeleteMapping("/take-away-access")
```

```
    public String takeawayAccess(@RequestParam("planId") int planId,
@RequestParam("personId") int personId, HttpSession session){
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            planDAO.takeawayAccess(planId, personId);
            roleDAO.deleteRole(personId, planId);
            optimizationDAO.deleteFromTaskPersonByPersonAndPlanId(personId,planId);
            return "redirect:/plans/tasks?planId="+planId;
        }
        return "redirect:/authorization";
    }
}
```

Б6 TaskController.java

```
@Controller
public class TaskController {
    private final TaskDAO taskDAO;
    private final TaskValidator taskValidator;
    private final PlanDAO planDAO;
    private final PersonDAO personDAO;
    private final FriendshipDAO friendshipDAO;
    private final RoleDAO roleDAO;
    private final OptimizationDAO optimizationDAO;

    public TaskController(TaskDAO taskDAO, TaskValidator taskValidator, PlanDAO planDAO,
PersonDAO personDAO, FriendshipDAO friendshipDAO, RoleDAO roleDAO,
OptimizationDAO optimizationDAO) {
        this.taskDAO = taskDAO;
        this.taskValidator = taskValidator;
        this.planDAO = planDAO;
        this.personDAO = personDAO;
        this.friendshipDAO = friendshipDAO;
        this.roleDAO = roleDAO;
        this.optimizationDAO = optimizationDAO;
    }

    @GetMapping("/plans/tasks")
    public String viewAllTasks(@RequestParam("planId") int planId, Model model, HttpSession
session) {
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            Plan plan = planDAO.findById(planId);
            if ((plan != null && plan.getPerson_id() == userId) || planDAO.isHaveAccess(userId,
planId)) {
                model.addAttribute("planId", planId);
                model.addAttribute("users", planDAO.getUsersByPlanAccess(planId));
```

```
            model.addAttribute("admin", planDAO.getPersonByPlanId(planId));
            model.addAttribute("userId", userId);
            model.addAttribute("optimizedPeople",
optimizationDAO.getPeopleForOptimization(planId));
            List<Task> tasks = taskDAO.getTaskByPlanId(planId);
            for(int i = 0;i<tasks.size();++i){
               tasks.get(i).setPerson(optimizationDAO.getPersonByTaskId(tasks.get(i).getId()));
            }
            model.addAttribute("tasks", tasks);
            //model.addAttribute("isPersonAlreadyInPlan", false);
            if (plan.getPerson_id() == userId) {
               model.addAttribute("isCreator", true);
            } else {
               model.addAttribute("isCreator", false);
            }
            List<Person> roles = planDAO.getUsersByPlanAccess(planId);
            List<Integer> numberRoles = new ArrayList<>();
            for (Person person : roles) {
               int roleId = roleDAO.getRoleByPersonIdAndPlanId(person.getId(), planId);
               numberRoles.add(roleId);
            }
            model.addAttribute("numberRoles", numberRoles);
            return "tasks/tasks";
         } else {
            return "error/access-denied";
         }
      }
      return "redirect:/authorization";
   }


   @GetMapping("/plan/tasks/new")
   public String showCreateTaskForm(@RequestParam("planId") int planId, Model model,
HttpSession session) {
      Integer userId = (Integer) session.getAttribute("userId");
      if (userId != null) {
         model.addAttribute("task", new Task());
         model.addAttribute("planId", planId); // Додати planId до моделі
         return "tasks/newTask";
      }
      return "redirect:/authorization";
   }

   @PostMapping("/plan/tasks/create")
   public String createTask(@ModelAttribute("task") @Valid Task task, BindingResult
```

```
bindingResult, HttpSession session, @RequestParam("planId") Integer planId) {
        taskValidator.validate(task, bindingResult);
        if (bindingResult.hasErrors()) {
            return "tasks/newTask";
        }
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            Plan plan = planDAO.findById(planId);
            if (plan != null) {

                task.setPlan_id(planId);
                taskDAO.save(task);
                return "redirect:/plans/tasks?planId=" + planId;
            }
        }
        return "redirect:/authorization";
    }

    @PostMapping("/plan/addPerson")
    public String addAccess(@RequestParam("username") String username,
@RequestParam("planId") int planId, HttpSession session, RedirectAttributes redirectAttributes,
Model model) {
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            Person person = personDAO.findByUsername(username);
            if (person == null) {
                redirectAttributes.addFlashAttribute("userNameError", "Користувача з таким
юзернеймом не знайдено");
                return "redirect:/plans/tasks?planId=" + planId;
            }
            if (friendshipDAO.areFriends(userId, person.getId())) {
                model.addAttribute("planId", planId);
                model.addAttribute("role", 1);
                planDAO.addAccess(person.getId(), planId);
                roleDAO.addRoleToUser(person.getId(), planId, 1);
                return "redirect:/plans/tasks?planId=" + planId;
            } else if (userId == person.getId()) {
                redirectAttributes.addFlashAttribute("userNameError", "Ви не можете дати доступ
самому собі");
                return "redirect:/plans/tasks?planId=" + planId;
            } else {
                redirectAttributes.addFlashAttribute("userNameError", "Ви можете давати доступ
тільки друзям");
                return "redirect:/plans/tasks?planId=" + planId;
            }
```

```
        }
        return "redirect:/authorization";
    }

    @PostMapping("/update-role")
    public String updateRole(@RequestParam("planId") int planId,
                             @RequestParam("personId") int personId,
                             @RequestParam("roleId") int roleId,
                             HttpSession session, Model model) {
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            roleDAO.updateRole(personId, planId, roleId);
            model.addAttribute("planId", planId);
            return "redirect:/plans/tasks?planId=" + planId;
        }
        return "redirect:/authorization";
    }

    @PostMapping("/add-to-plan")
    public String addToPlanPerson(@RequestParam("planId") int planId,
                                  @RequestParam("personId") int personId,
                                  HttpSession session, Model model) {
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            optimizationDAO.addPersonToPlanOptimization(planId, personId);
            model.addAttribute("planId", planId);
            //model.addAttribute("isPersonAlreadyInPlan",
optimizationDAO.isPersonAlreadyInPlan(planId, personId));
            return "redirect:/plans/tasks?planId=" + planId;
        }
        return "redirect:/authorization";
    }

    @PostMapping("/optimize-plan")
    public String optimizePlan(@RequestParam("planId") int planId,
                               HttpSession session, Model model) {
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            optimizationDAO.deleteOptimization(planId);
            List<Person> optimizedPeople = optimizationDAO.getPeopleForOptimization(planId);
            List<Task> tasks = taskDAO.getTaskByPlanId(planId);
            List<PersonTasks> personTasks = new ArrayList<>();
            for (int i = 0; i < optimizedPeople.size(); ++i) {
                personTasks.add(new PersonTasks(optimizedPeople.get(i).getName(),
optimizedPeople.get(i).getId()));
```

```
            }
            Optimization.optimizeTaskDistribution(personTasks, tasks);
            for (int i = 0; i < optimizedPeople.size(); ++i) {
                List<Task> pTasks = personTasks.get(i).getTasks();
                for (int j = 0; j < pTasks.size(); ++j) {
                    Task task = pTasks.get(j);
                    task.setPerson(optimizedPeople.get(i));
                    optimizationDAO.addPersonToTask(task.getId(), personTasks.get(i).getId());
                }
            }
            model.addAttribute("planId", planId);
            return "redirect:/plans/tasks?planId=" + planId;
        }
        return "redirect:/authorization";
    }

    @GetMapping("/plan/tasks/task")
    public String showTask(@RequestParam("taskId") int taskId, @RequestParam("planId") int
planId, Model model, HttpSession session) {
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            Plan plan = planDAO.findById(planId);
            if (plan.getPerson_id() == userId) {
                model.addAttribute("isCreator", true);
                model.addAttribute("roleId", 0);
            } else {
                model.addAttribute("roleId", roleDAO.getRoleByPersonIdAndPlanId(userId, planId));
                model.addAttribute("isCreator", false);
            }
            model.addAttribute("task", taskDAO.show(taskId));
            model.addAttribute("person", optimizationDAO.getPersonByTaskId(taskId));
            model.addAttribute("planId", planId);
            return "tasks/task";
        }
        return "redirect:/authorization";
    }

    @PostMapping("/task/update")
    public String updateCompleted(@RequestParam("taskId") int taskId, Model model,
@RequestParam("planId") int planId, HttpSession session, @RequestParam("completed")
boolean completed) {
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            Task task = taskDAO.show(taskId);
            task.setCompleted(completed);
```

```
            taskDAO.setCompleted(completed, taskId);
            model.addAttribute("planId", planId);
            return "redirect:/plan/tasks/task?taskId=" + taskId;
        }
        return "redirect:/authorization";
    }

    @DeleteMapping("/task/delete")
    public String deleteTask(@RequestParam("taskId") int taskId, @RequestParam("planId") int
planId, Model model, HttpSession session) {
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            taskDAO.deleteTask(taskId);
            model.addAttribute("planId", planId);
            return "redirect:/plans/tasks?planId=" + planId;
        }
        return "redirect:/authorization";
    }

    @DeleteMapping("/remove-person-optimization")
    public String removePersonOptimization(@RequestParam("planId") int planId,
                                           @RequestParam("personId") int personId,
                                           Model model, HttpSession session){
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            optimizationDAO.deleteFromTaskPersonByPersonAndPlanId(personId, planId);
            return "redirect:/plans/tasks?planId=" + planId;
        }
        return "redirect:/authorization";
    }
}
```

Б7 FriendshipController.java

```
@Controller
public class FriendshipController {
    private final FriendshipDAO friendshipDAO;
    private final PersonDAO personDAO;

    public FriendshipController(FriendshipDAO friendshipDAO, PersonDAO personDAO) {
        this.friendshipDAO = friendshipDAO;
        this.personDAO = personDAO;
    }

    @GetMapping("/friends")
    public String viewFriends(Model model, HttpSession session) {
        Integer userId = (Integer) session.getAttribute("userId");
```

```
        if (userId != null) {
            List<Person> friends = friendshipDAO.getFriendsByPersonId(userId);
            List<Person> sentFriendshipRequests =
friendshipDAO.getSentFriendshipRequestsByPersonId(userId);
            List<Person> receivedFriendshipRequests =
friendshipDAO.getReceivedFriendshipRequestsByPersonId(userId);
            model.addAttribute("friends", friends);
            model.addAttribute("sentFriendshipRequests", sentFriendshipRequests);
            model.addAttribute("receivedFriendshipRequests", receivedFriendshipRequests);
            return "friends/friends";
        }
        return "redirect:/authorization";
    }

    @PostMapping("/add-friend")
    public String addFriend(@RequestParam("friendId") int friendId, HttpSession session, Model
model) {
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            if(friendId==userId){
                model.addAttribute("ownProfile", true);
                model.addAttribute("friendshipStatus", "Ви");
            }
            else {
                model.addAttribute("ownProfile", false);
                Person person = personDAO.findById(friendId);
                model.addAttribute("person", person);
                if (friendshipDAO.areFriends(userId, friendId)) {
                    model.addAttribute("friendshipStatus", "Друзі");
                } else if (friendshipDAO.hasPendingRequest(userId, friendId)) {
                    model.addAttribute("friendshipStatus", "Запит надіслано");
                } else {
                    friendshipDAO.sendFriendshipRequest(userId, friendId);
                    model.addAttribute("friendshipStatus", "Запит відправлено");
                }
            }
        }
        return "redirect:/info/" + friendId;
    }

    @PostMapping("/accept-request")
    public String acceptRequest(@RequestParam("friendId") int friendId, HttpSession session){
        Integer userId = (Integer) session.getAttribute("userId");
        if(userId!=null){
            friendshipDAO.sendFriendshipRequest(userId, friendId);
        }
```

```
        return "redirect:/friends";
    }

    @PostMapping("/cancel-sent-request")
    public String cancelSentRequest(@RequestParam("friendId") int friendId, HttpSession
session) {
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            friendshipDAO.cancelFriendshipRequest(userId, friendId);
        }
        return "redirect:/friends";
    }

    @PostMapping("/cancel-received-request")
    public String cancelReceivedRequest(@RequestParam("friendId") int friendId, HttpSession
session) {
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId != null) {
            friendshipDAO.cancelFriendshipRequest(friendId, userId);
        }
        return "redirect:/friends";
    }

    @PostMapping("/search-friend")
    public String searchFriend(@RequestParam("username") String username, HttpSession
session, RedirectAttributes redirectAttributes) {
        Integer userId = (Integer) session.getAttribute("userId");
        if(userId!=null) {
            Person person = personDAO.findByUsername(username);
            if (person != null) {
                if(friendshipDAO.areFriends(userId, person.getId())) {
                    redirectAttributes.addFlashAttribute("userNameError", "Ви вже друзі");
                    return "redirect:/friends";
                }
                else if(friendshipDAO.hasPendingRequest(userId, person.getId())){
                    redirectAttributes.addFlashAttribute("userNameError", "Ви вже надіслали запит в
друзі цьому користувачу");
                    return "redirect:/friends";
                }
                else if(userId==person.getId()){
                    redirectAttributes.addFlashAttribute("userNameError", "Ви не можете надіслати
запит у друзі самому собі");
                    return "redirect:/friends";
                }
                else {
                    friendshipDAO.sendFriendshipRequest(userId, person.getId());
```

```
                return "redirect:/friends";
            }
        }
        else{
            redirectAttributes.addFlashAttribute("userNameError", "Користувача з таким
юзернеймом не знайдено");
            return "redirect:/friends";
        }
    }
    return "redirect:/friends";
  }

  @PostMapping("/delete-friend")
  public String deleteFriend(@RequestParam("friendId") int friendId, HttpSession session){
    Integer userId = (Integer) session.getAttribute("userId");
    if (userId != null) {
      friendshipDAO.deleteFriend(userId, friendId);
    }
    return "redirect:/friends";
  }
}
```

ДОДАТОК В

Класи DAO

В1 PersonDAO.java

```
@Component
public class PersonDAO {
   private final JdbcTemplate jdbcTemplate;

   @Autowired
   public PersonDAO(JdbcTemplate jdbcTemplate) {
      this.jdbcTemplate = jdbcTemplate;
   }

   public void save(Person person){
      jdbcTemplate.update("insert into person(name, date, email, username, password)
values(?,?,?,?,?)",
            person.getName(), person.getDate(), person.getEmail(), person.getUsername(),
person.getPassword());
   }

   public Person show(int id) {
      return jdbcTemplate.query("select * from person where id=?",
                  new Object[]{id}, new BeanPropertyRowMapper<>(Person.class))
            .stream().findAny().orElse(null);
   }

   public Person findById(int id){
      return jdbcTemplate.query("select * from person where id=?",
            new Object[]{id}, new
BeanPropertyRowMapper<>(Person.class)).stream().findAny().orElse(null);
   }

   public Person findByUsername(String username){
      return jdbcTemplate.query("select * from person where username=?",
            new Object[]{username}, new
BeanPropertyRowMapper<>(Person.class)).stream().findAny().orElse(null);
   }

   public Optional<Person> showByUsername(String username){
      return jdbcTemplate.query("select * from person where username=?",
                  new Object[]{username}, new BeanPropertyRowMapper<>(Person.class))
            .stream().findAny();
   }

   public Optional<Person> showByEmail(String email){
```

```
        return jdbcTemplate.query("select * from person where email=?",
                new Object[]{email}, new BeanPropertyRowMapper<>(Person.class))
            .stream().findAny();
    }

    public void updateData(int id, Person updatedPerson) {
        jdbcTemplate.update("update person set name=?, date=?, password=? where id=?",
            updatedPerson.getName(), updatedPerson.getDate(), updatedPerson.getPassword(),
id);
    }
}
```

B2 PlanDAO.java

```
@Component
public class PlanDAO {
    private final JdbcTemplate jdbcTemplate;

    public PlanDAO(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    public void save(Plan plan) {
        jdbcTemplate.update("insert into plan(person_id, name, start_date, end_date)
values(?,?,?,?)",
            plan.getPerson_id(), plan.getName(), plan.getStart_date(), plan.getEnd_date());
    }

    public List<Plan> getPlansByPersonId(int id) {
        return jdbcTemplate.query("select * from plan where person_id = ?",
            new Object[]{id}, new BeanPropertyRowMapper<>(Plan.class));
    }

    public Person getPersonByPlanId(int id){
        return jdbcTemplate.query("SELECT person.* FROM person JOIN plan ON person.id =
plan.person_id WHERE plan.id = ?",
            new Object[]{id}, new
BeanPropertyRowMapper<>(Person.class)).stream().findAny().orElse(null);
    }

    public Plan findById(int id) {
        return jdbcTemplate.query("select * from plan where id=?",
                new Object[]{id}, new BeanPropertyRowMapper<>(Plan.class))
            .stream().findAny().orElse(null);
    }

    public void addAccess(int personId, int planId){
```

```
        jdbcTemplate.update("insert into plan_access(person_id, plan_id) VALUES (?,?)",
personId, planId);
    }

    public List<Person> getUsersByPlanAccess(int planId){
        return jdbcTemplate.query("SELECT person.* FROM person JOIN plan_access ON
person.id = plan_access.person_id WHERE plan_access.plan_id = ?",
                new Object[]{planId}, new BeanPropertyRowMapper<>(Person.class));
    }

    public List<Plan> getPlansAccessByPersonId(Integer userId) {
        return jdbcTemplate.query("SELECT plan.* FROM plan JOIN plan_access ON plan.id =
plan_access.plan_id WHERE plan_access.person_id = ?",
                new Object[]{userId}, new BeanPropertyRowMapper<>(Plan.class));
    }

    public boolean isHaveAccess(int userId, int planId){
        String sql = "SELECT COUNT(*) FROM plan_access WHERE person_id = ? AND
plan_id = ?";
        int count = jdbcTemplate.queryForObject(sql, new Object[]{userId, planId}, Integer.class);
        return count > 0;
    }

    public void deletePlan(int planId) {
        jdbcTemplate.update("delete from plan where id=?", planId);
    }

    public void deleteAccess(int planId){
        jdbcTemplate.update("delete from plan_access where plan_id=?", planId);
    }

    public void takeawayAccess(int planId, int personId){
        jdbcTemplate.update("delete from plan_access where plan_id=? and person_id=?", planId,
personId);
    }
}
```

B3 TaskDAO.java

```
@Component
public class TaskDAO {
    private final JdbcTemplate jdbcTemplate;

    public TaskDAO(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }
```

```
    public void save(Task task) {
        jdbcTemplate.update("insert into task(plan_id, name, period, completed) values(?,?,?,?)",
                task.getPlan_id(), task.getName(), task.getPeriod(), false);
    }

    public List<Task> getTaskByPlanId(int id){
        return jdbcTemplate.query("select * from task where plan_id = ? order by period",
                new Object[]{id}, new BeanPropertyRowMapper<>(Task.class));
    }

    public Task show(int id) {
        return jdbcTemplate.query("select * from task where id=?",
                        new Object[]{id}, new BeanPropertyRowMapper<>(Task.class))
                .stream().findAny().orElse(null);
    }

    public void deleteTask(int id){
        jdbcTemplate.update("delete from task where id=?", id);
    }

    public void setCompleted(boolean completed, int taskId) {
        jdbcTemplate.update("UPDATE task SET completed = ? WHERE id = ?", completed,
taskId);
    }
}
```

B4 FriendshipDAO.java

```
@Component
public class FriendshipDAO {
    private final JdbcTemplate jdbcTemplate;

    public FriendshipDAO(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    public List<Person> getFriendsByPersonId(int id) {
        return jdbcTemplate.query("SELECT p.id, p.username, p.name\n" +
                        "FROM person p\n" +
                        "INNER JOIN friendship f1 ON p.id = f1.person2_id\n" +
                        "INNER JOIN friendship f2 ON p.id = f2.person1_id\n" +
                        "WHERE f1.person1_id = ? AND f2.person2_id = ?",
                new Object[]{id, id}, new BeanPropertyRowMapper<>(Person.class));
    }

    public void sendFriendshipRequest(int person1_id, int person2_id){
        jdbcTemplate.update("INSERT INTO friendship (person1_id, person2_id) VALUES (?,
```

```
?)", person1_id, person2_id);
    }

    public boolean areFriends(Integer userId, int friendId) {
        String query = "SELECT COUNT(*) FROM friendship WHERE (person1_id = ? AND
person2_id = ?) OR (person1_id = ? AND person2_id = ?)";
        int count = jdbcTemplate.queryForObject(query, new Object[]{userId, friendId, friendId,
userId}, Integer.class);
        return count > 1;
    }

    public boolean hasPendingRequest(Integer userId, int friendId) {
        String query = "SELECT COUNT(*) FROM friendship WHERE person1_id = ? AND
person2_id = ?";
        int count = jdbcTemplate.queryForObject(query, new Object[]{userId, friendId},
Integer.class);
        return count > 0;
    }


    public List<Person> getSentFriendshipRequestsByPersonId(Integer id) {
        return jdbcTemplate.query("SELECT p.id, p.username, p.name\n" +
                "FROM person p\n" +
                "        INNER JOIN friendship f ON p.id = f.person2_id\n" +
                "WHERE f.person1_id = ?\n" +
                "  AND f.person2_id NOT IN (SELECT person1_id FROM friendship WHERE
person2_id = ?)",
            new Object[]{id, id}, new BeanPropertyRowMapper<>(Person.class));
    }

    public List<Person> getReceivedFriendshipRequestsByPersonId(Integer id) {
        return jdbcTemplate.query("SELECT p.id, p.username, p.name\n" +
            "FROM person p\n" +
            "        INNER JOIN friendship f ON p.id = f.person1_id\n" +
            "WHERE f.person2_id = ?\n" +
            "  AND f.person1_id NOT IN (SELECT person2_id FROM friendship WHERE
person1_id = ?)",
            new Object[]{id, id}, new BeanPropertyRowMapper<>(Person.class));
    }

    public void cancelFriendshipRequest(Integer userId, int friendId) {
        jdbcTemplate.update("delete from friendship where person1_id = ? and person2_id=?",
userId, friendId);
    }
```

```
    public void deleteFriend(Integer userId, int friendId) {
        jdbcTemplate.update("delete from friendship where (person1_id=? and person2_id=?)
or(person2_id=? and person1_id=?)",
                userId, friendId, userId, friendId);
    }
}
```

B5 RoleDAO.java

```
@Component
public class RoleDAO {
    private final JdbcTemplate jdbcTemplate;

    public RoleDAO(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    public void addRoleToUser(int personId, int planId, int roleId){
        jdbcTemplate.update("insert into user_role(person_id, plan_id, role_id) VALUES (?,?,?)",
                personId, planId, roleId);
    }

    public void updateRole(int personId, int planId, int roleId){
        jdbcTemplate.update("update user_role set role_id=? where person_id=? and plan_id=?",
roleId, personId, planId);
    }

    public int getRoleByPersonIdAndPlanId(int personId, int planId){
        return jdbcTemplate.queryForObject("select role_id from user_role where person_id=? and
plan_id=?", Integer.class, personId, planId);
    }

    public void deleteRole(int personId, int planId){
        jdbcTemplate.update("delete from user_role where person_id=? and plan_id=?", personId,
planId);
    }
}
```

B6 OptimizationDAO.java

```
@Component
public class OptimizationDAO {
    private final JdbcTemplate jdbcTemplate;

    public OptimizationDAO(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    public void addPersonToPlanOptimization(int planId, int personId){
```

```
        jdbcTemplate.update("insert into plan_person(plan_id, person_id) values (?,?)", planId,
personId);
    }

    public void addPersonToTask(int taskId, int personId){
        jdbcTemplate.update("insert into task_person(task_id, person_id) values (?,?)", taskId,
personId);
    }

    public void deleteOptimization(int planId){
        jdbcTemplate.update("DELETE FROM task_person WHERE task_id IN (SELECT id
FROM Task WHERE plan_id = ?)", planId);
    }

    public Person getPersonByTaskId(int taskId){
        return jdbcTemplate.query("select p.* from task_person tp join person p on
tp.person_id=p.id where tp.task_id=?",
                        new Object[]{taskId}, new BeanPropertyRowMapper<>(Person.class))
                .stream().findAny().orElse(null);
    }

    public void deleteFromTaskPersonByPersonAndPlanId(int personId, int planId){
        jdbcTemplate.update("delete from task_person where task_id in(select id FROM task
WHERE plan_id = ?) AND person_id = ?",
                planId, personId);
        jdbcTemplate.update("delete from plan_person where plan_id = ? and person_id = ?",
planId, personId);
    }

    public List<Person> getPeopleForOptimization(int planId){
        return jdbcTemplate.query("SELECT p.* FROM plan_person pp JOIN person p ON
pp.person_id = p.id WHERE pp.plan_id = ?",
                new Object[]{planId}, new BeanPropertyRowMapper<>(Person.class));
    }

    public boolean isPersonAlreadyInPlan(int planId, int personId){
        String sql = "SELECT COUNT(*) FROM plan_person WHERE person_id = ? AND
plan_id = ?";
        int count = jdbcTemplate.queryForObject(sql, new Object[]{personId, planId},
Integer.class);
        return count > 0;
    }
}
```

ДОДАТОК Г

Клас оптимізації

Г1 Optimization.java

```
public class Optimization {
   public static void optimizeTaskDistribution(List<PersonTasks> people, List<Task> tasks) {
      Collections.sort(tasks, Comparator.comparingDouble(Task::getPeriod).reversed());
      int peopleCount = people.size();
      for (Task task : tasks) {
         PersonTasks minPerson = Collections.min(people,
Comparator.comparingDouble(PersonTasks::getTotalDuration));
         minPerson.addTask(task);
      }
      double averageDuration = tasks.stream().mapToDouble(Task::getPeriod).sum() /
peopleCount;
      for (PersonTasks personTasks : people) {
         personTasks.setAllocatedDuration(averageDuration);
      }
   }
}
```