МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

_____

(підпис)

01 червня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 – Комп'ютерні науки,

освітньо-професійної програми «Інформатика»

на тему: «Система планування та спільної роботи над проектами»

здобувача групи ІН-02 Пушкаря Дмитра Євгеновича

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

Дмитро ПУШКАРЬ

_____

(підпис)

Керівник

асистент комп'ютерних наук,

кандидат фізико-математичних наук Олександр ВЛАСЕНКО_____

(підпис)

Суми – 2024

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

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

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
на здобуття освітнього ступеня бакалавра
зі спеціальності 122 – Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-02 Пушкаря Дмитра Євгеновича

1. Тема роботи: «Система планування та спільної роботи над проектами.»
затверджую наказом по СумДУ від *«22» квітня 2024 р. № 0414-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 01 червня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити): *Інформаційний огляд, існуючі рішення, постановка задачі. Вибір методів рішення задач. Програмна реалізація. Взаємодія користувача з системою.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

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

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

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Затвердження теми роботи*	22.04.2024	
2	*Вивчення та аналіз задачі*	6.05.2024	
3	*Поглиблене дослідження бібліотек, інструментів, технологій, що будуть використовуватися*	11.05.2024	
4	*Програмна реалізація системи*	19.05.2024	
5	*Аналіз отриманих результатів*	21.05.2024	
6	*Оформлення пояснювальної записки до кваліфікаційної роботи*	25.05.2024	

Здобувач вищої освіти _____ (підпис) Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 130 стр., 23 рис., 7 таблиць, 61 додаток, 12 використаних джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною тому що в сучасному світі ефективне управління проєктами та спільна робота над ними є ключовими факторами успішної діяльності організацій. Зростаюча складність проєктів, швидкість змін у бізнес-середовищі та потреба в ефективному використанні ресурсів створюють попит на системи, які допомагають забезпечити структуроване планування, координацію робіт та спільну роботу у всіх етапах проєктного циклу.

Об’єкт дослідження — функціональні аспекти створення вебсайту, включаючи його серверну та клієнтську частини.

Мета роботи — розробка веб-сайту, який буде використовуватися у сфері управління проєктами.

Методи дослідження — системний аналіз, порівняльний аналіз, моделювання, емпіричний метод, експериментальний метод.

Результати — було розроблено повнофункціональну інформаційну систему, що відповідає вимогам та потребам користувачів у сфері планування та управління проєктами.

ІНФОРМАЦІЙНА СИСТЕМА, ПЛАНУВАННЯ БАЗИ ДАНИХ, FRONT-END, BACK-END, REACT, JAVA, REDUX, LOMBOK, MAPSTRUCT, SPRING BOOT, MATERIAL UI

ЗМІСТ

ВСТУП

Використання сучасних інформаційних технологій у сфері планування та спільної роботи над проектами набуває все більшого значення в умовах стрімкого розвитку бізнесу. Завдяки цим технологіям компанії можуть оптимізувати процеси управління та забезпечувати високий рівень контролю за виконанням завдань. Системи управління проектами допомагають не тільки у розподілі задач, але й у моніторингу їхнього виконання, що є критично важливим для досягнення успіху в конкурентному середовищі.

Однією з ключових функцій таких систем є забезпечення зручного доступу до інформації та інструментів з будь-якого пристрою. Веб-орієнтовані платформи дозволяють учасникам проекту взаємодіяти в режимі реального часу, що значно спрощує процес комунікації та прийняття рішень. Ці інструменти також сприяють організації важливої документації, полегшують управління строками виконання завдань та надають можливість ефективного контролю за прогресом роботи.

Актуальність дослідження зумовлена зростаючою потребою у високоефективних інструментах для управління проектами в сучасному бізнес-середовищі. Щодня збільшується кількість команд, які працюють над різними проектами, що потребує надійних та зручних рішень для планування, координації та контролю задач. Розробка унікальних продуктів, що відповідають специфічним вимогам кожної компанії, дозволяє підвищити продуктивність та гнучкість управління.

Об’єктом дослідження є процес створення веб-орієнтованої системи для планування та спільної роботи над проектами. Метою роботи є розробка інформаційної системи, яка сприятиме ефективному управлінню проектами та задачами, забезпечуватиме інтуїтивно зрозумілий інтерфейс і включатиме основні функції для створення, видалення та моніторингу задач.

Гіпотеза дослідження полягає в тому, що зростаюча кількість команд вимагає нових ефективних інструментів для управління внутрішніми процесами розробки проектів. Новизна розробки полягає у створенні клієнто-

орієнтованої веб-системи, яка дозволить покращити управління процесами розробки програмного забезпечення. Веб-додаток буде корисним як для особистого контролю робочих процесів, так і для великих команд розробників.

Структура роботи складається зі вступу, огляду сучасних технологій управління проектами та задачами, аналізу існуючих аналогічних програмних продуктів, постановки задачі, вибору мов програмування, фреймворків, бібліотек та СУБД для реалізації веб-додатку, практичної реалізації та огляду використання програмного продукту, висновків, списку використаних джерел та додатків.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Системи управління проєктами та спільною роботою

Сучасні системи управління проєктами – це інструменти, які допомагають організувати, планувати та виконувати різноманітні завдання, пов’язані з розробкою та реалізацією проєктів. Вони надають можливість ефективно керувати ресурсами, розподіляти завдання між учасниками команди, встановлювати терміни та відстежувати прогрес виконання. Крім того, ці системи сприяють спільній роботі над проєктами, забезпечуючи зручний доступ до необхідної інформації та можливість спілкування між учасниками команди. Таким чином, вони стають невід’ємною частиною процесу управління проєктами та сприяють досягненню кращих результатів.

1.2 Існуючі рішення

Існують різноманітні інструменти для управління проєктами та спільної роботи, серед яких популярними є Jira [2] та Trello [3].

Jira – це потужний інструмент управління проєктами, який надає широкий набір функцій для планування, виконання та відстеження проєктів у реальному часі. У Jira можна створювати завдання, встановлювати терміни виконання, розподіляти ресурси та відстежувати прогрес виконання завдань. Крім того, Jira дозволяє використовувати різні методології управління проєктами, включаючи Agile, Scrum та Kanban.

Недоліки Jira включають складність в освоєнні та конфігурації [4], що може вимагати значного часу та зусиль від користувачів. Крім того, великий функціонал Jira може бути надто важким для менших команд або проєктів, що потребують простішого підходу до управління завданнями. Ще одним недоліком може бути висока вартість платних планів та додаткових розширень, що може зробити Jira не доступною для деяких організацій або команд.

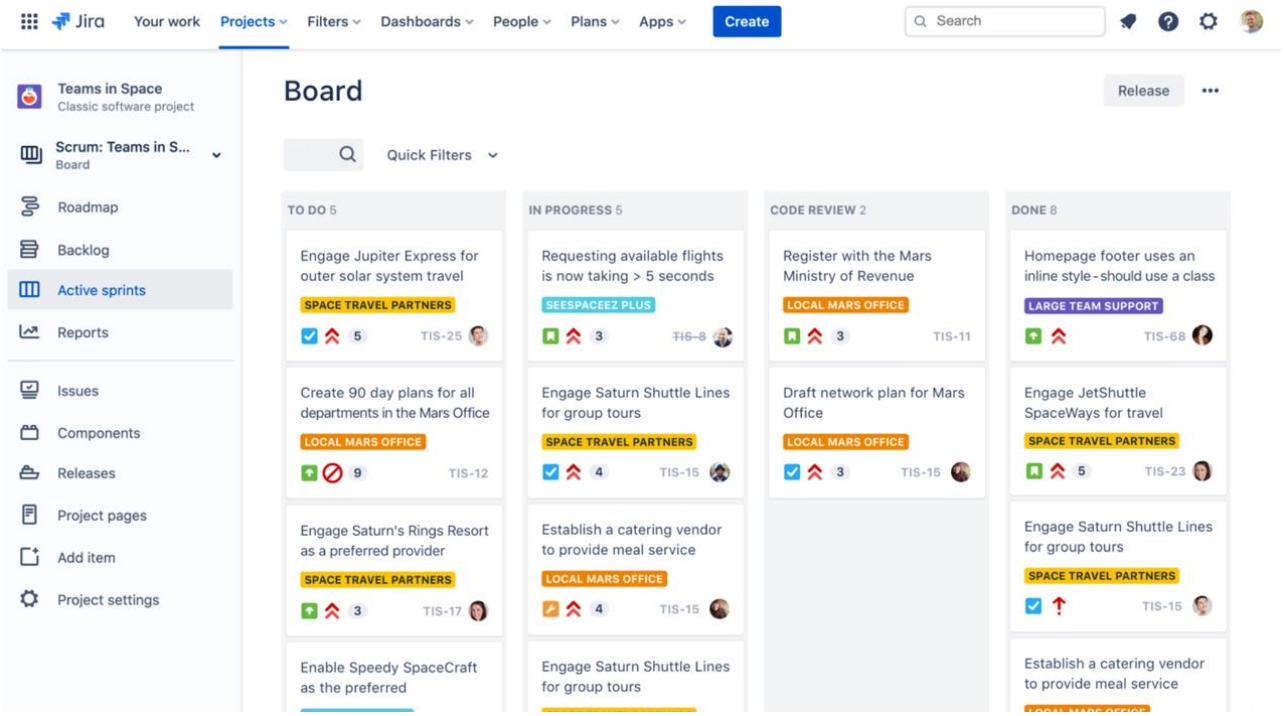


Рисунок 1.1 – Приклад дошки Jira

Trello – це інтерактивна канбан-дошка, яка дозволяє створювати та організовувати завдання у вигляді карток, які можна пересувати між колонками. Кожна картка може містити різну інформацію, включаючи опис завдання, додаткові файли, коментарі та терміни виконання. Trello використовується для спільної роботи над проєктами, організації задач та відстеження їх прогресу.

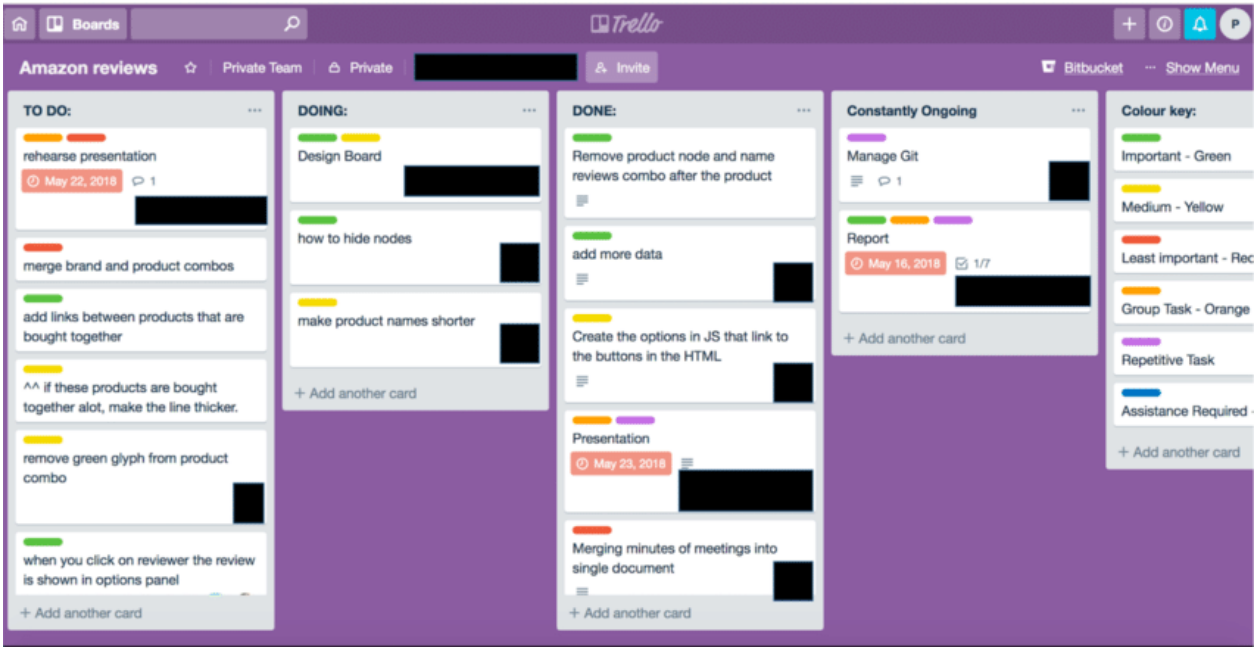


Рисунок 1.2 – Приклад дошки Trello

У випадку з Trello, недоліком є обмежена функціональність порівняно з Jira. Також, для деяких користувачів може бути недостатньою можливість розширення функціоналу за допомогою додаткових плагінів або інтеграцій.

Крім того, Trello може виявитися менш ефективним для складних проєктів з великою кількістю завдань і ресурсів, оскільки його інтерфейс іноді може стати занадто загромадженим і не забезпечувати достатньої структуризації для таких ситуацій.

1.3 Постановка задачі

Для виконання поставлених завдань і досягнення мети проєкту необхідно мати широкий функціонал системи, який забезпечить високу продуктивність та зручність в користуванні. Основні складові цього функціоналу включають:

1. Реалізація інструментів для планування, організації та виконання завдань в межах проєктів. Забезпечення можливості додавання нових проєктів, призначення завдань та встановлення термінів їх виконання.

2. Створення інтерфейсу користувача: Розробка зручного та інтуїтивно зрозумілого інтерфейсу для користувачів. Можливість швидкого доступу до необхідної інформації та виконання дій без зайвих складнощів.

3. Розширені можливості звітності: Впровадження функціоналу, який дозволить генерувати детальні звіти та аналітику про хід виконання проєктів. Створення зручних інструментів для аналізу даних та оцінки продуктивності проєктів, що сприятиме прийняттю обґрунтованих рішень та оптимізації робочих процесів.

4. Тестування та валідація: Проведення тестування системи з метою виявлення та виправлення можливих помилок та недоліків. Перевірка на відповідність функціональним вимогам та ефективність роботи в реальних умовах використання.

5. Можливість комунікації та співпраці: Створення зручних інструментів для спілкування між учасниками проєктів, обміну інформацією та координації робочих процесів.

6. Система сповіщень та нагадувань: Впровадження механізму автоматичних сповіщень та нагадувань про наближення термінів виконання завдань допоможе уникнути протермінування та забезпечить своєчасність виконання робіт.

2. ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧ

2.1 Вибір фреймворку серверної частини

Під час розробки програмного забезпечення, вибір правильного фреймворку є критичним етапом. Фреймворк – це [5] набір готових компонентів, бібліотек та інструментів, які допомагають розробникам створювати програми швидше та ефективніше. Він надає структуру для розробки та виконання програм, визначає правила та шаблони для організації коду та спрощує процес розробки.

Вибір фреймворку – важливий етап, оскільки від цього залежить якість та продуктивність програмного продукту. Правильно обране рішення може значно полегшити розробку та підтримку проєкту, у той час як неправильний вибір може призвести до зайвих труднощів та обмежень у майбутньому.

Spring Framework є одним із найпопулярніших та широко використовуваних фреймворків для розробки програм на мові Java. [6] Він відомий своєю гнучкістю, великим спектром функціоналу та високою продуктивністю.

Плюсами Spring є його широкий функціонал, велика активна спільнота розробників, висока продуктивність та гнучкість. Він також має велику кількість документації та сторонніх бібліотек, що спрощує процес розробки та підтримки проєктів.

Проте, серед мінусів Spring можна відзначити деяку складність в освоєнні, особливо для початківців. Також він може вимагати більше ресурсів для запуску та підтримки порівняно з іншими фреймворками.

Між конкурентами Spring Framework, Micronaut і Ktor знаходяться ряд суттєвих відмінностей та переваг, які варто враховувати при виборі фреймворку для розробки програм на мові Java або Kotlin.

Micronaut [7] – це революційний фреймворк, побудований з урахуванням принципів обліково-орієнтованого проєктування та мінімалізму. Він

відрізняється високою продуктивністю та швидкодією завдяки своєму низькорівневому підходу до управління залежностями та виконання коду. Micronaut став дуже популярним серед розробників мікросервісних архітектур та проєктів, які вимагають великої масштабованості та ефективності.

Ktor – це модерний фреймворк для розробки веб-застосунків та мікросервісів на мові Kotlin. Він відрізняється простотою використання та гнучкістю, а також має високу швидкодію. Ktor зазвичай використовується для швидкого розгортання вебзастосунків, а основною його перевагою є зручність роботи з асинхронним кодом.

При порівнянні зі Spring Framework, Micronaut та Ktor мають меншу кількість вбудованих функцій та меншу спільноту розробників. [8]

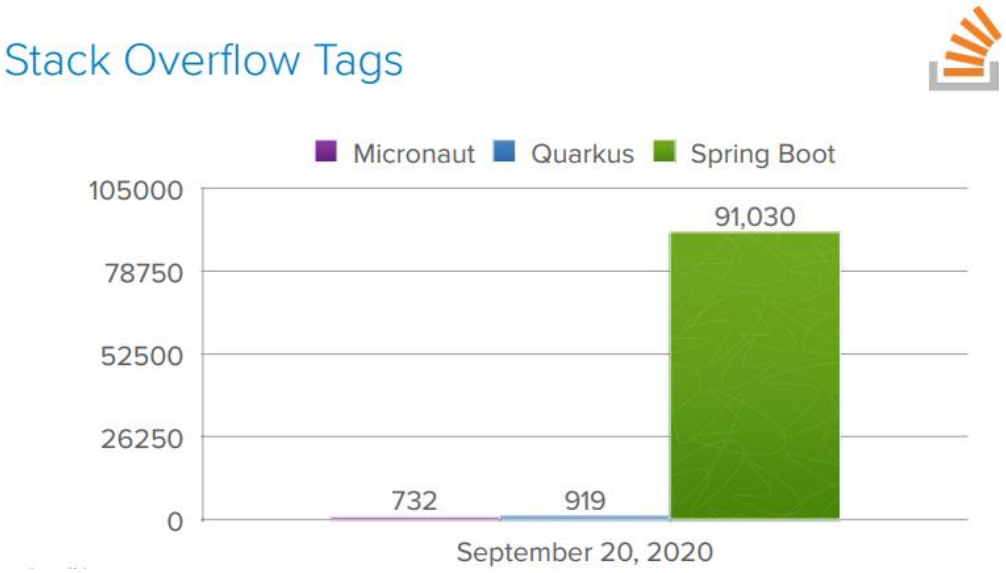


Рисунок 2.1 – Порівняння тегів на StackOverflow

2.3 Вибір фреймворку фронтенд частини

Розробка фронтенду без фреймворку вимагає від розробників написання всього коду вручну, без використання готових інструментів та бібліотек. Це може бути варіантом для простих проєктів або для розробників, які віддають перевагу повній контролю над кодом. Однак, такий підхід може призвести до

більшої складності та витрат часу на розробку, особливо для складних веб-додатків.

React – це один з найпопулярніших фреймворків для розробки фронтенду. Він базується на компонентній архітектурі, що дозволяє розбивати користувацький інтерфейс на невеликі незалежні компоненти, що спрощує розробку та підтримку коду. React активно використовується в багатьох вебдодатках та сайтах, від невеликих проєктів до великих корпоративних систем.

Angular – це фреймворк для розробки фронтенду, розроблений компанією Google. Він надає велику кількість готових інструментів та бібліотек для розробки веб-додатків, включаючи систему модулів, сервіси, директиви та інші. Angular відзначається своєю потужністю та можливістю розробки складних односторінкових додатків.

Vue – це фреймворк для розробки фронтенду, який поєднує в собі простоту використання з великим функціоналом. Він має легку та інтуїтивно зрозумілу синтаксис, що дозволяє швидко вивчити його та почати розробку. Vue набирає все більшу популярність серед розробників та використовується в різних вебпроєктах.

Переваги використання фреймворків включають велику кількість готових інструментів та бібліотек, спрощення розробки та підтримки коду, швидкість розробки та високу продуктивність. Мінуси можуть включати складність вивчення та використання, а також обмеження у виборі технологій.

У випадку вибору між React, Angular та Vue, [9] кожен з цих фреймворків має свої сильні та слабкі сторони, які варто враховувати в залежності від конкретних потреб та особливостей проєкту.

Обираючи фреймворк для фронтенду, було вирішено вибрати React. Це рішення базується на декількох ключових чинниках.

● По-перше, React має велику та активну спільноту розробників, що забезпечує широкий спектр документації, підтримки та ресурсів для вирішення проблем та підтримки проєктів. Це робить розробку на React більш простою та ефективною.

● По-друге, React пропонує компонентну архітектуру, яка спрощує розробку та підтримку великих та складних користувацьких інтерфейсів. Розбивка інтерфейсу на незалежні компоненти полегшує управління кодом та дозволяє швидше вносити зміни.

Крім того, React відомий своєю швидкодією та продуктивністю, що дозволяє створювати швидкі та реактивні вебдодатки. Його висока ефективність робить його ідеальним вибором для будь-яких проєктів, від невеликих сайтів до великих корпоративних застосунків.

Таким чином, обираючи React, можна бути впевненим, що цей фреймворк надасть потрібний набір інструментів, продуктивність та підтримку, щоб успішно реалізувати проєкт та досягти поставлених цілей.

2.4 Вибір бази даних

Бази даних є важливою складовою сучасних інформаційних технологій і використовуються в різних сферах, починаючи від побутових додатків до великих корпоративних систем. База даних - це організована колекція даних, яка зберігається та обробляється з допомогою комп’ютерної системи. Вони забезпечують можливість ефективного зберігання, організації та доступу до великих обсягів даних.

Існують різні типи баз даних, але їх можна узагальнити до двох основних категорій: реляційні (SQL) та нереляційні (NoSQL).

Реляційні бази даних (SQL) [10] організовані у вигляді таблиць, які складаються з рядків та стовпців. Вони використовують мову запитів SQL для маніпулювання даними та забезпечують стандартні властивості ACID (Atomicity, Consistency, Isolation, Durability), що гарантують надійність та цілісність даних.

Нереляційні бази даних (NoSQL) не використовують традиційні таблиці, а замість цього зберігають дані у форматі ключ-значення, документів, стовпців або графів. Ці бази даних надають більшу гнучкість та масштабованість для зберігання невеликих або незначних даних.

Бази даних потрібні для забезпечення доступу до інформації, забезпечення її зберігання та організації, а також для забезпечення цілісності та безпеки даних. Вони використовуються в різних галузях, включаючи бізнес, медицину, освіту, науку та інші. Бази даних дозволяють організаціям ефективно управляти своєю інформацією та забезпечують основу для розробки різних програмних продуктів та додатків.

У проєкті було обрано SQL базу даних, зокрема PostgreSQL. PostgreSQL – це потужна, відкрита та безкоштовна реляційна система керування базами даних з великим функціоналом та розвинутою підтримкою стандартів SQL.

Переваги PostgreSQL включають:

● Розширені можливості: підтримка відомих функцій, таких як JSON, XML, та географічні об’єкти.

● Гнучкість: можливість розширення та налаштування бази даних відповідно до потреб проєкту.

● Велика активна спільнота: доступність ресурсів та підтримки від широкого кола розробників.

Недоліки PostgreSQL включають:

● Складність конфігурації: для максимального використання потенціалу PostgreSQL може знадобитися досить глибока експертиза в області адміністрування баз даних.

● Використання PostgreSQL рекомендується для проєктів, які потребують надійності, масштабованості та розширюваності бази даних, таких як веб-додатки, великі портали та системи управління даними.

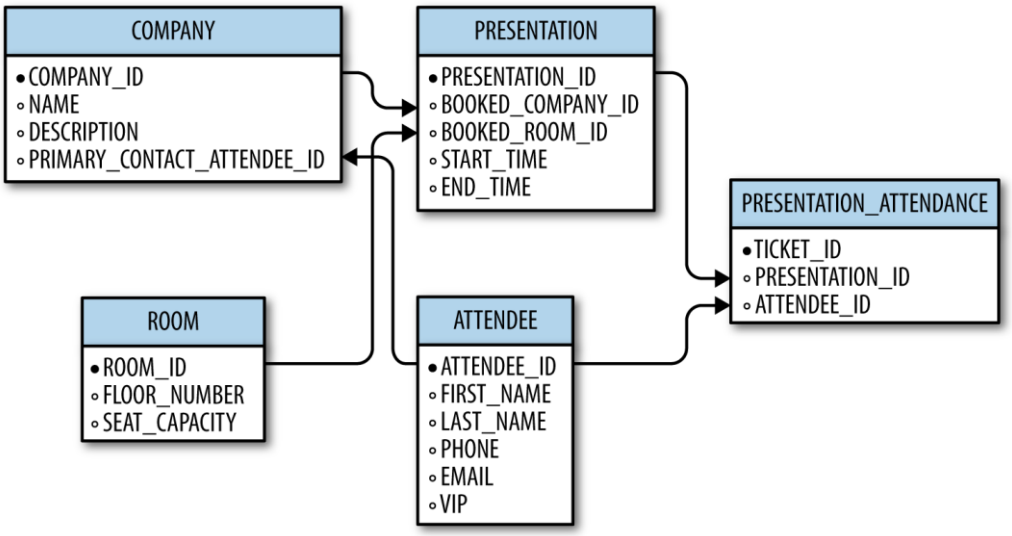


Рисунок 2.2 – Приклад планування реляційної бази даних

```
{
  "_id": "5cf0029caff5056591b0ce7d",
  "firstname": "Jane",
  "lastname": "Wu",
  "address": {
    "street": "1 Circle Rd",
    "city": "Los Angeles",
    "state": "CA",
    "zip": "90404"
  }
  "hobbies": ["surfing", "coding"]
}
```

Рисунок 2.3 – Приклад нереляційної бази даних

2.5 Вибір місця зберігання коду

Репозиторій для коду – це централізоване сховище, де зберігається весь вихідний код проєкту, а також інші файлові ресурси, такі як зображення, документація та конфігураційні файли. Це дозволяє розробникам працювати над проєктом колективно, вносячи зміни, виправляючи помилки та роблячи нові функції, та відстежувати всі ці зміни в хронологічному порядку.

Переваги використання репозиторію для коду включають:

● Версіонування: можливість відстежувати всі зміни в коді та відновлювати попередні версії, що дозволяє уникнути втрати даних та виправити помилки.

● Колективна робота: розробники можуть працювати над проєктом одночасно, вносячи зміни та обговорюючи їх у спеціальних форумах або коментарях.

● Розподілений доступ: репозиторії для коду зазвичай зберігаються в хмарних сервісах, що дозволяє отримати доступ до них з будь-якого пристрою та місця.

У проєкті було обрано GitHub як репозиторій для коду. GitHub є одним з найпопулярніших та найбільш широко використовуваних хмарних сервісів для зберігання та управління вихідним кодом. Він пропонує широкий спектр інструментів для спільної роботи, включаючи можливість створювати гілки для різних функцій, вносити зміни та обговорювати їх у спеціальних форумах. Крім того, GitHub надає безкоштовний доступ для відкритих проєктів, що робить його ідеальним вибором для відкритих проєктів та командної розробки.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Проєктування бази даних

Далі розглянемо процес розробки бази даних. Спочатку описується функціональні та нефункціональні вимоги до системи, щоб чітко зрозуміти, які дані потрібно зберігати та яким чином вони будуть використовуватися. Після цього переходимо до проєктування схеми бази даних, включаючи таблиці, взаємозв’язки та індекси, щоб забезпечити ефективне зберігання та швидкий доступ до даних. Також розглядаються варіанти забезпечення безпеки, резервного копіювання та масштабування бази даних для забезпечення її надійності та продуктивності.

Функціональні вимоги для "Системи планування та спільної роботи над проєктами":

● Створення проєктів: Користувачі можуть створювати нові проєкти та надавати їм назви та опис для управління роботою над ними.

● Створення завдань: Можливість створення нових завдань в межах кожного проєкту з назвою, описом, термінами виконання та призначенням для відповідальних користувачів.

● Управління завданнями: Користувачі можуть переміщувати завдання між різними статусами (наприклад, "В очікуванні", "У роботі", "Завершено") для відслідковування прогресу.

● Призначення користувачів: Можливість призначати відповідальних користувачів для кожного завдання та відстежувати, хто відповідає за його виконання.

● Коментування завдань: Можливість додавати коментарі до завдань для обговорення деталей та спілкування між користувачами.

● Пошук та фільтрація: Можливість швидко знаходити та фільтрувати завдання за різними параметрами, такими як статус, відповідальні користувачі тощо.

● Керування дозволами: Можливість налаштовувати рівні доступу для користувачів до проєктів та завдань залежно від їхніх ролей та обов’язків.

Ці функціональні вимоги допоможуть забезпечити повний функціонал системи планування та спільної роботи над проєктами, аналогічно до Trello, та забезпечать ефективне керування завданнями та співпрацю між користувачами.

Нефункціональні вимоги для "Системи планування та спільної роботи над проєктами":

● Продуктивність: Система повинна забезпечувати швидкий доступ до даних та оперативну відповідь на запити користувачів навіть при великому обсязі даних та одночасному використанні багатьма користувачами.

● Надійність: Система повинна бути стійкою до відмов та забезпечувати безперебійну роботу навіть в умовах незапланованих відключень або помилок.

● Безпека: Забезпечення конфіденційності, цілісності та доступності даних шляхом застосування відповідних методів аутентифікації, авторизації та шифрування.

● Масштабованість: Система повинна бути готовою до масштабування, щоб забезпечити нормальну роботу навіть при збільшенні обсягу даних та кількості користувачів.

● Інтерфейс користувача: Забезпечення зручного та інтуїтивно зрозумілого інтерфейсу користувача для зменшення часу на навчання та підвищення продуктивності роботи.

● Сумісність: Система повинна бути сумісною з різними операційними системами, веб-браузерами та пристроями, щоб забезпечити доступність для широкого кола користувачів.

Таблиця 3.1 – Схема користувачів

Поле	Тип даних
id	bigserial
username	Varchar(50)
first_name	Varchar(50)

Last_name	Varchar(50)
Password	Varchar(150)
User_status	Varchar(50)
Created_at	TIMESTAMP
Updated_at	TIMESTAMP

Таблиця 3.2 – Ролі користувачів

Поле	Тип даних
id	BIGSERIAL
name	varchar(50)

Таблиця 3.3 – Поєднання користувача і ролей

Поле	Тип даних
User_id	Bigint
Role_id	Bigint

Таблиця 3.4 – Рефреш токени

Поле	Тип даних
Id	Bigserial
User_id	Bigint
Token	Varchar(255)
expiration	timestamp

Таблиця 3.5 – Дошки проєктів

Поле	Тип даних
Id	Bigserial
Name	Varchar(50)
User_id	Bigint
Created_at	Timestamp

Таблиця 3.6 – Елементи у дошках

Поле	Тип даних
Id	Bigserial
Title	Varchar(50)
Text	Varchar(1500)
Board_id	Bigint
Created_at	Timestamp
Status	Varchar(50)
index	int

Таблиця 3.7 – Користувачі дошок

Поле	Тип даних
User_id	Bigint
Board_id	Bigint

3.2 Розробка структури проєкту та підготовка бази даних

При запуску команди насправді запускається PostgreSQL у віртуальному середовищі. Ось кілька причин, чому це зручно робити:

- Запуск PostgreSQL у Docker дозволяє швидко створити та налаштувати базу даних без необхідності встановлення його вручну на вашому комп’ютері.
- Кожен контейнер Docker має своє власне ізольоване середовище, що дозволяє уникнути конфліктів між різними додатками та сервісами, які використовують одні й ті самі ресурси.
- Docker дозволяє швидко створювати та запускати додатки у віртуальних контейнерах, що спрощує процес масштабування та розгортання.

```
→  ~
docker run --name trellolo -e POSTGRES_PASSWORD=postgres -p 5432:5432 -d postgres
```

Рисунок 3.1 – Команда docker для запуску бази даних

Тепер розберемо кожен пункт з команди:

- docker run: Ця команда вказує Docker запустити новий контейнер.
- --name trellolo: Параметр --name встановлює ім’я контейнера як trellolo.
- -e POSTGRES_PASSWORD=postgres: Параметр -e встановлює змінну середовища для контейнера, в цьому випадку встановлюється пароль для користувача PostgreSQL як postgres.
- -p 5432:5432: Параметр -p мапує порти між контейнером та хост-системою. Мапується порт 5432 контейнера PostgreSQL на порт 5432 хост-системи.
- -d: Цей параметр запускає контейнер у фоновому режимі.
- postgres: Завершальна частина команди вказує образ Docker, який використовується для створення контейнера. У цьому випадку, це офіційний образ PostgreSQL.

Далі створюємо структуру папок бекенд частини проєкту:

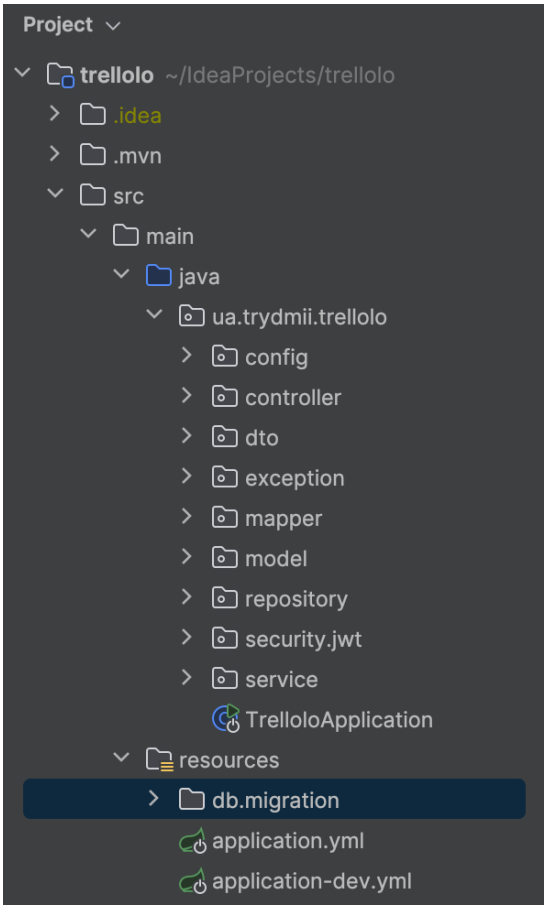


Рисунок 3.2 – Структура проєкту

- Папка config має загальні конфігурації бібліотек та проєкту
- Папка controller має контролери проєкту
- В папці dto розташовані об’єкти передачі даних, які використовуються для передачі даних між різними частинами програмного коду, такими як контролери та сервіси. Ці об’єкти зазвичай відображають структуру даних, яка передається між клієнтом та сервером, або в межах серверної частини застосунку.
- папка exception має створені кастомні виключення
- папка mapper має один інтерфейс маперу з використанням бібліотеки MapStruct, який динамічно генерує мапери для сутностей та DTO
- Папка model, має сутності які відображають таблички з бази даних через механізм JPA (стандартна специфікація Java EE, яка надає можливості для управління реляційними даними в програмах Java, спрощуючи взаємодію з базами даних. Використання JPA дозволяє розробникам створювати об’єктно-орієнтовані моделі даних та працювати з ними в мові програмування Java без прямого використання SQL-запитів).
- Папка repository має інтерфейси для роботи з класами з папки model (заміняє механізм sql запитів на виклик методів)
- Папка security має класи та налаштування пов’язані з авторизацією/аутентифікацією
- Папка service має сервіси які виконують усі бізнес логіку застосунку.
- Папка resources має усі ресурси які не являються java кодом.

3.3 Створення сутностей системи для мапінгу з базою даних

У цьому розділі ми розглянемо створення сутностей для мапінгу з базою даних у системі. Основна мета полягає в забезпеченні зручного та ефективного збереження і управління даними за допомогою об’єктно-реляційного мапінгу (ORM). Ми використовуємо JPA (Java Persistence API) разом із Lombok для зручності розробки.

BaseEntity Першою сутністю є базовий клас BaseEntity, який використовується для наслідування іншими сутностями. Він містить загальні атрибути, які потрібні всім сутностям, такі як ідентифікатор.

```java
@MappedSuperclass
@Getter
@Setter
@ToString
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
public class BaseEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
}
```

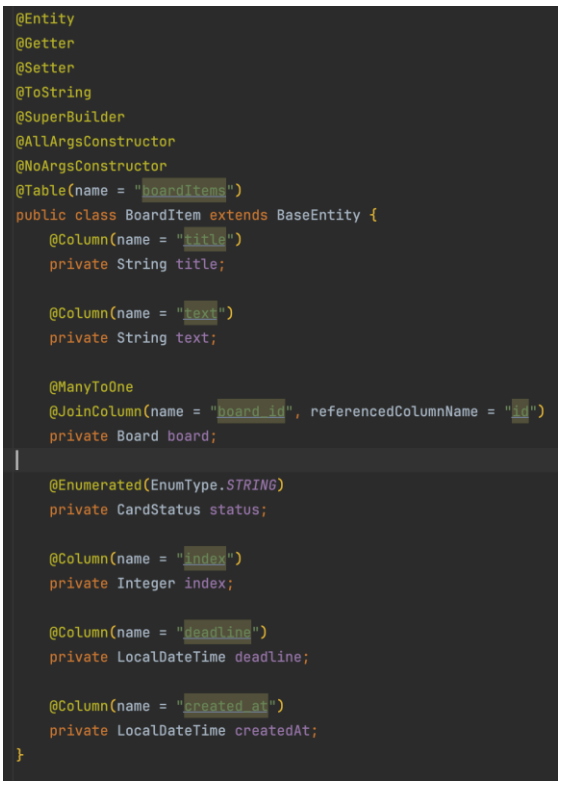
Рисунок 3.3 – сутність BaseEntity

Сутність Board представляє дошку у системі. Вона містить інформацію про назву дошки, власника, дату створення та список користувачів, які мають доступ до цієї дошки.

```java
@Entity
@Getter
@Setter
@ToString
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "boards")
public class Board extends BaseEntity {
    @Column(name = "name")
    private String name;

    @ManyToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id")
    private User owner;

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

    @ManyToMany(mappedBy = "boards", fetch = FetchType.EAGER)
    private List<User> users;
}
```

Рисунок 3.4 – сутність Board

Сутність BoardItem представляє окремий елемент дошки. Вона включає заголовок, текст, статус, індекс, дедлайн і дату створення. Крім того, вона зв’язана з певною дошкою.



```
@Entity
@Getter
@Setter
@ToString
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "boardItems")
public class BoardItem extends BaseEntity {
    @Column(name = "title")
    private String title;

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

    @ManyToOne
    @JoinColumn(name = "board_id", referencedColumnName = "id")
    private Board board;

    @Enumerated(EnumType.STRING)
    private CardStatus status;

    @Column(name = "index")
    private Integer index;

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

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

Рисунок 3.5 – сутність BoardItem

Сутність RefreshToken використовується для управління токенами оновлення, що забезпечують безпеку авторизації. Вона містить інформацію про токен, дату закінчення терміну дії та зв’язок з користувачем.

```
@EqualsAndHashCode(callSuper = true)
@Entity
@Table(name = "refresh_tokens")
@Data
@AllArgsConstructor
@ToString(exclude = "user")
@SuperBuilder
@NoArgsConstructor
public class RefreshToken extends BaseEntity {
    @OneToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id")
    private User user;

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

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

Рисунок 3.6 – сутність RefreshToken

Сутність Role представляє роль користувача у системі. Вона включає назву ролі та список користувачів, які мають цю роль.

```java
@Getter
@Setter
@ToString
@RequiredArgsConstructor
@Entity
@SuperBuilder
@Table(name = "roles")
public class Role extends BaseEntity {
    @Column(name = "name")
    private String name;

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

Рисунок 3.7 – сутність Role

Сутність User представляє користувача системи. Вона включає ім’я користувача, паролі, інформацію про ролі, статус користувача, дати створення та оновлення, а також список дошок, до яких користувач має доступ.

```
public class User extends BaseEntity {
    @Column(name = "username")
    private String username;

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

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

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

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

    @Column(name = "user_status")
    @Enumerated(EnumType.STRING)
    private UserStatus userStatus;

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

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

    @ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @JoinTable(
            name = "board_users",
            joinColumns = { @JoinColumn(name = "user_id", referencedColumnName = "id") },
            inverseJoinColumns = { @JoinColumn(name = "board_id", referencedColumnName = "id") }
    )
    private List<Board> boards;
}
```

Рисунок 3.8 – сутність User

Перерахування UserStatus визначає можливі статуси користувача у системі.

```
public enum UserStatus {
    ACTIVE, BANNED
}
```

Рисунок 3.9 – enum UserStatus

Ці сутності забезпечують основну структуру даних для нашої системи, дозволяючи ефективно зберігати та обробляти інформацію, пов’язану з користувачами, дошками, елементами дошок та коментарями до них.

Анотації, які використовуються у сутностях для мапінгу з базою даних є частиною JPA (Java Persistence API) і Lombok, які полегшують роботу з базами даних та знижують кількість шаблонного коду.

3.3.1 Анотації JPA:

1. @Entity використовується для позначення класу як сутності JPA. Сутність представляє таблицю у базі даних, а її поля відповідають стовпцям цієї таблиці.

2. @Table дозволяє визначити додаткові властивості таблиці, такі як її назва. Якщо ця анотація не вказана, то ім’я таблиці буде відповідати імені класу.

3. @Id позначає поле як первинний ключ таблиці.

4. @GeneratedValue вказує, що значення цього поля буде автоматично згенеровано. Стратегія генерації визначається параметром strategy.

5. @MappedSuperclass позначає клас як базовий для інших сутностей. Поля цього класу успадковуються підкласами, але сам клас не відображається у базі даних.

6. @Column використовується для налаштування властивостей стовпця бази даних, таких як ім’я, унікальність, довжина тощо.

7. @ManyToOne позначає зв’язок "багато до одного" між сутностями. Наприклад, багато елементів дошки можуть належати одній дошці.

8. @JoinColumn використовується для налаштування властивостей зовнішнього ключа у зв’язку між таблицями.

9. @ManyToMany позначає зв’язок "багато до багатьох" між сутностями. Наприклад, багато користувачів можуть мати доступ до багатьох дошок.

10. @JoinTable використовується для налаштування проміжної таблиці у зв’язку "багато до багатьох".

11. @Enumerated використовується для збереження значень перерахування (enum) у базі даних. Вона дозволяє визначити, як саме значення перерахування будуть збережені (як строки або як порядкові значення).

3.3.2 Анотації Lombok:

1. @Getter і @Setter автоматично генерують методи доступу (гетери і сетери) для всіх полів класу.

2. @ToString генерує метод toString() для класу, включаючи всі поля.

3. @SuperBuilder генерує шаблон Builder для класу, дозволяючи зручно створювати його екземпляри з використанням шаблону будівельника. Працює з успадкуванням.

4. @AllArgsConstructor генерує конструктор, який приймає всі поля класу як параметри.

5. @NoArgsConstructor генерує конструктор без параметрів.

6. @Data є комплексною анотацією, яка включає в себе @Getter, @Setter, @ToString, @EqualsAndHashCode, а також @RequiredArgsConstructor.

7. @EqualsAndHashCode генерує методи equals і hashCode для класу, включаючи всі поля або зазначені поля.

Використання цих анотацій значно спрощує процес розробки та дозволяє зосередитися на бізнес-логіці, а не на шаблонному коді.

3.4 Створення власного мапперу.

Далі розглянемо створення власного мапера за допомогою бібліотеки MapStruct. Мапер допомагає перетворювати одні об’єкти в інші, зокрема, сутності бази даних у Data Transfer Objects (DTO) і навпаки.

Для реалізації власного мапера ми створили інтерфейс CustomMapper, який визначає методи мапінгу між різними об’єктами.

Рисунок 3.10 – власний маппер

Анотація @Mapper позначає інтерфейс як маппер. Параметр componentModel = "spring" вказує, що MapStruct повинен створити реалізацію цього інтерфейсу як Spring Bean, що дозволяє легко використовувати його в додатку, який базується на Spring Framework.

Мапінг методів:

1. userToUserDto(User user) метод перетворює сутність User в UserDto. Анотація @Mapping з параметром target = "userStatus", ignore = true вказує, що поле userStatus не потрібно включати у результат мапінгу.

2. requestToUser(RegisterRequestDto requestDto) метод перетворює об’єкт RegisterRequestDto в сутність User. Поля userStatus, updated, created і roles ігноруються.

3. roleToRoleDto(Role role) метод перетворює сутність Role в RoleDto.

4. boardToBoardDto(Board board) метод перетворює сутність Board в BoardDto. Поле owner мапиться з використанням іменованого методу mapUserToUsername.

5. mapUserToUsername(User user) іменований метод (@Named("mapUserToUsername")) мапує об’єкт User на ім’я користувача (username). Використовується в методі boardToBoardDto.

6. boardDtoToBoard(BoardDto boardDto, User owner) метод перетворює BoardDto в сутність Board. Поле id мапується з boardDto.id, а поле owner мапується з параметра owner.

7. boardItemToBoardItemDto(BoardItem boardItem) метод перетворює сутність BoardItem в BoardItemDto.

8. boardItemDtoToBoardItem(BoardItemCreateRequestDto boardItemDto) метод перетворює BoardItemCreateRequestDto в сутність BoardItem. Поля board, createdAt і id ігноруються.

9. userToBoardUsersDto(User user) метод перетворює сутність User в BoardUsersDto.

Використання MapStruct значно спрощує процес мапінгу між сутностями і DTO, автоматично генеруючи необхідний код. Це дозволяє уникнути ручного кодування та зменшує кількість помилок, покращуючи продуктивність розробки та підтримку коду. Маппер інтегрується зі Spring, що робить його використання зручним у будь-якому проєкті, побудованому на цьому фреймворку.

3.5 Створення репозиторіїв для сутностей

Розглянемо створення репозиторіїв для всіх сутностей нашої системи. Репозиторії використовуються для забезпечення доступу до даних і виконання CRUD операцій (Create, Read, Update, Delete). Для створення репозиторіїв ми використовуємо Spring Data JPA, що дозволяє швидко і легко реалізувати доступ до бази даних. Загальний підхід до створення репозиторіїв Кожен репозиторій буде інтерфейсом, який розширює JpaRepository. Це надає доступ до

стандартних методів CRUD, таких як save(), findById(), findAll(), deleteById(), та інші. Крім того, ми додамо декілька кастомних методів для специфічних запитів.

```
public interface BoardItemRepository extends JpaRepository<BoardItem, Long> {
    List<BoardItem> findByBoardId(Long boardId);
}
```

Рисунок 3.11 – приклад одного з репозиторіїв.

3.6 Створення контролерів

3.6.1 Контролер для авторизації та аутентифікації

Контролер AuthControllerV1 відповідає за процеси реєстрації, входу в систему, оновлення токенів та виходу користувача з системи. Основні методи включають:

- POST /sign-up: реєстрація нового користувача. Приймає дані реєстрації, перевіряє наявність користувача з таким ім’ям і, якщо такого немає, реєструє нового користувача.

- POST /sign-in: вхід користувача. Перевіряє наявність користувача, автентифікує його та генерує JWT токен і токен оновлення.

- POST /refresh: оновлення JWT токена. Приймає токен оновлення, перевіряє його дійсність і генерує новий JWT токен.

- DELETE /logout: вихід користувача. Видаляє токени оновлення користувача, що фактично здійснює його вихід із системи.

3.6.2 Контролер для роботи з дошками

Контролер BoardControllerV1 забезпечує роботу з дошками. Основні методи включають:

- GET /: отримання списку дошок користувача, який виконав запит.

- GET /{id}: отримання конкретної дошки за її ID.

- POST /: створення нової дошки. Приймає дані нової дошки та зберігає її.

- DELETE /{id}: видалення дошки за її ID.

- GET /users/{id}: отримання списку користувачів, які мають доступ до дошки.

- POST /users/{id}: додавання користувачів до дошки.

- GET /owner/{id}: перевірка, чи є поточний користувач власником дошки.

3.6.3 Контролер для роботи з елементами дошки

Контролер BoardItemControllerV1 відповідає за управління елементами дошки. Основні методи включають:

- GET /{id}: отримання списку елементів дошки за її ID.

- POST /{id}: створення нового елемента дошки. Приймає дані нового елемента та зберігає його.

- DELETE /{id}: видалення елемента дошки за його ID.

- PATCH /: оновлення статусів елементів дошки. Приймає список змін статусів і застосовує їх.

- PUT /: оновлення елемента дошки. Приймає оновлені дані елемента та зберігає їх.

Ці контролери забезпечують повний цикл CRUD операцій для користувачів, дошок та елементів дошки, дозволяючи керувати основними сутностями системи.

3.7 Створення сервісів з основною бізнес логікою серверної частини

Було створено наступні сервіси

1. UserServiceImpl: цей сервіс відповідає за операції, пов’язані з користувачами. Він реалізує методи для пошуку користувачів за їхнім ім’ям користувача та реєстрації нових користувачів. Під час реєстрації, сервіс генерує хеш паролю та встановлює додаткові атрибути, такі як ролі та статус користувача.

2. RefreshTokenServiceImpl: даний сервіс відповідає за роботу з оновлюваними токенами. Він має методи для створення нових токенів, перевірки та оновлення токенів, а також видалення токенів користувачів.

3. BoardServiceImpl: сервіс, який відповідає за операції, пов'язані з дошками. Він реалізує методи для пошуку дошок за власником, знаходження дошки за ідентифікатором, створення нової дошки, видалення дошки, отримання користувачів дошки та збереження користувачів до дошки.

4. BoardItemServiceImpl: цей сервіс відповідає за операції, пов'язані з елементами дошки. Він реалізує методи для пошуку елементів дошки за ідентифікатором дошки, створення нового елемента дошки, видалення елемента дошки та оновлення статусу елементів дошки.

3.8 Створення структури папок клієнтської частини застосунку

Для початку створюємо структуру папок (рис.3.12). Створюємо два файли api.js та auth.js, які допоможуть керувати токенами для авторизації та аутентифікації.

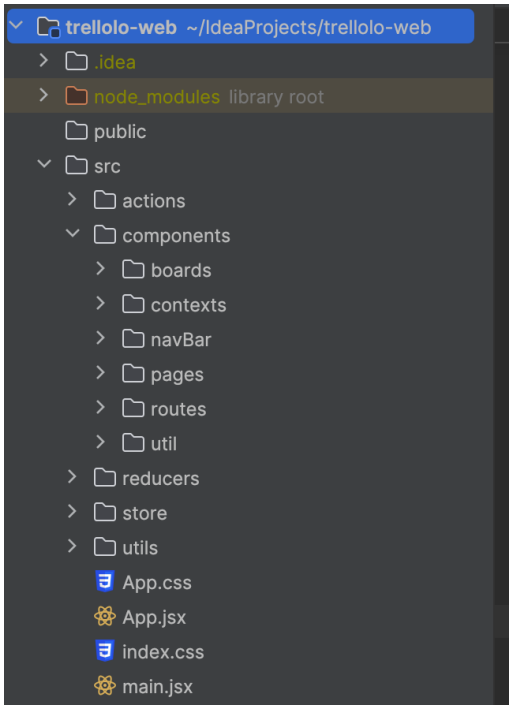


Рисунок 3.12 – Структура папок веб частини проєкту

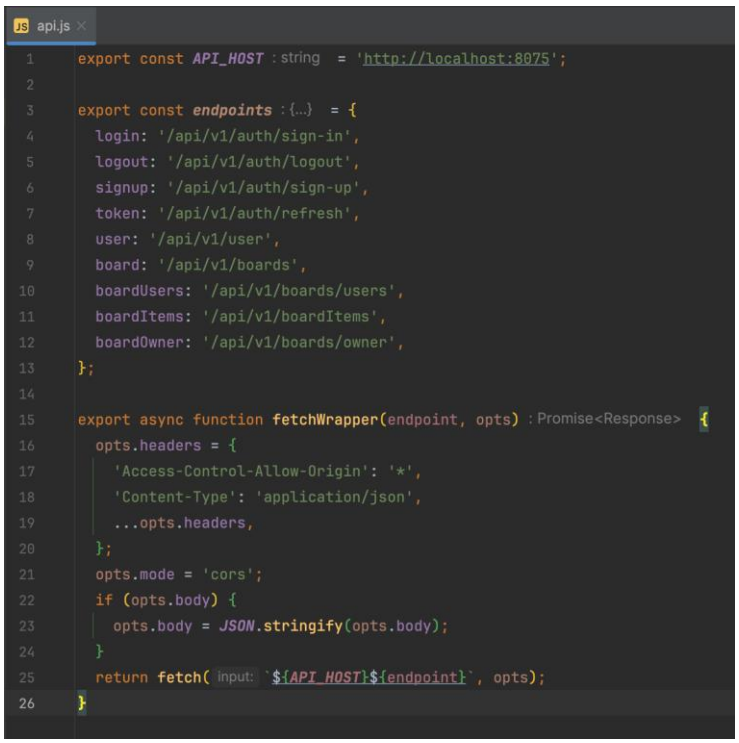


Рисунок 3.13 – файл api.js

Рисунок 3.14 – файл auth.js

Перший файл містить константи, що визначають кінцеві точки API, а також функцію fetchWrapper, яка використовується для виконання запитів до API з переданими параметрами. Функція fetchWrapper також додає необхідні заголовки до запитів та встановлює режим CORS для передачі даних між різними доменами.

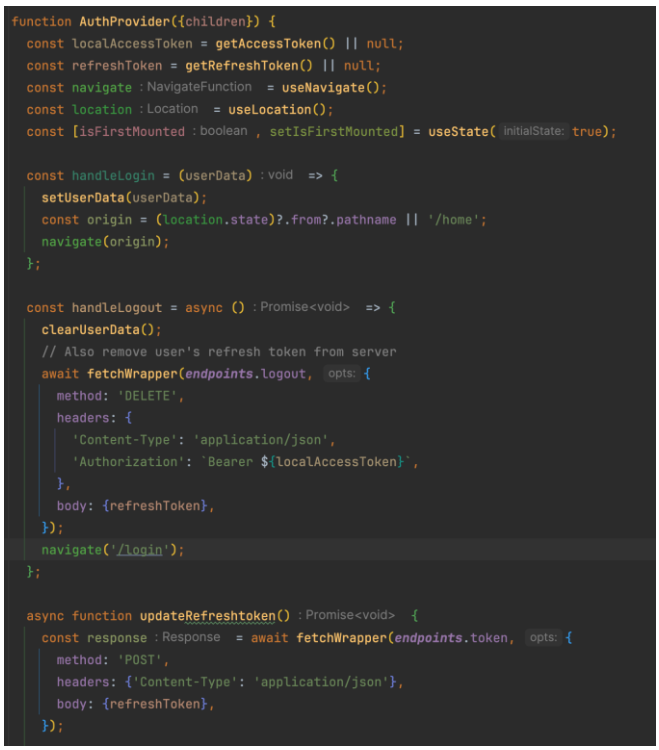
Другий файл містить набір функцій для роботи з автентифікацією користувача та локальним сховищем даних. Функції включають в себе отримання, збереження та очищення даних користувача в локальному сховищі, а також отримання та оновлення токенів доступу, перевірку статусу автентифікації та отримання даних користувача з токену.

Далі маємо файл AuthProvider.jsx, він містить компонент AuthProvider, який використовується для надання контексту автентифікації в додатку React. В компоненті здійснюється логіка аутентифікації та управління токенами доступу.

Функціональність AuthProvider включає в себе налаштування та оновлення токенів доступу, перевірку їх наявності та строку дії, виклик методів автентифікації та виходу з системи, а також передачу відповідних значень через контекст.

Компонент використовує ефекти React для взаємодії з токенами та виклику методів автентифікації відповідно до потреб додатка. Він також використовується для перевірки та оновлення токенів доступу з певним інтервалом часу, щоб забезпечити безперервну роботу автентифікації в додатку.

Крім того, компонент використовує контекст та хуки роутингу з бібліотеки react-router-dom для навігації між сторінками додатку в залежності від статусу автентифікації користувача.



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

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

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

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

Рисунок 3.15 – файл AuthProvider.jsx 1 частина

```
43      if (response.ok) {
44        const {accessToken} = await response.json();
45        updateAccessToken(accessToken);
46      } else {
47        clearUserData();
48        navigate('/login');
49        window.location.reload();
50      }
51      if (isFirstMounted) {
52        setIsFirstMounted( value: false);
53      }
54    }
55
56    useEffect( effect: () : function(): void | any  => {
57      if (refreshToken) {
58        // Check on the first render
59        if (isFirstMounted) {
60          updateRefreshtoken();
61        }
62
63        // Keep checking after a certain time
64        const intervalId : number  = setInterval( handler: () : void  => {
65          updateRefreshtoken();
66        }, ACCESS_TOKEN_EXPIRES_TIME);
67        return () : void  => clearInterval(intervalId);
68      }
69      return undefined;
70    },  deps: [localAccessToken]);
71
72    const value : {...}  = useMemo( factory: () : {...}  => ({
73      token: localAccessToken,
74      onLogin: handleLogin,
75      onLogout: handleLogout,
76    }),  deps: [localAccessToken]);
77
78    return (
79      <AuthContext.Provider value={value}>
80        {children}
81      </AuthContext.Provider>
82    );
```

Рисунок 3.16 – файл AuthProvider.jsx 2 частина

Далі маємо файл listReducer.js, він містить редуктор listReducer, який відповідає за управління станом списків і карт у додатку. Він обробляє різні дії (actions), такі як додавання карти, видалення карти, оновлення карти, перетягування карти між списками та отримання списків карт з сервера.

Файл також містить декілька функцій екшенів, які виконують асинхронні операції над картами, такі як отримання, додавання та видалення карт. Кожен з цих екшенів взаємодіє з сервером за допомогою власної функції fetchWrapper, яка використовується для виконання запитів HTTP.

Крім того, у файлі також описані декілька допоміжних функцій, які використовуються для перетягування карт між списками та оновлення карт у стані за допомогою даних з сервера.

Редуктор listReducer приймає дії та змінює стан додатку відповідно до цих дій. Він має початковий стан initialState, який містить початкові списки з пустими картами та порожній масив помилок.

Створення основних компонентів застосунку:

1. BoardCard.js: компонент React для відображення картки, що представляє дошку. Включає функціонал для відкриття дошки та видалення її.

2. BoardModal.js: модальний компонент для створення нової дошки. Дозволяє користувачам вводити назву дошки та створювати її.

3. TrelloActionButton.js: компонент для відображення кнопки для додавання нової картки до списку у додатку на зразок Trello.

4. TrelloBoard.js: Компонент для відображення головного виду дошки у додатку на зразок Trello. Показує списки карток та дозволяє користувачам перетягувати картки між списками.

5. TrelloCard.js: компонент для відображення окремих карток у додатку на зразок Trello. Включає опції для редагування або видалення картки.

6. TrelloList.js: компонент для відображення списків карток у додатку на зразок Trello. Надає контейнер для кількох карток та включає кнопку дії для додавання нових карток.

7. TrelloModal.js: модальний компонент для створення або оновлення картки у додатку на зразок Trello. Дозволяє користувачам вводити назву, текст та крайній термін для картки.

8. UsersModal.js: модальний компонент для управління користувачами, пов’язаними з дошкою у додатку на зразок Trello. Дозволяє користувачам переглядати, додавати та видаляти користувачів з дошки.

3.9 Створення основних сторінок

1. Boards (Дошки): ця сторінка дозволяє користувачам переглядати свої дошки. Вона відображає список дошок, які належать користувачеві, і дозволяє йому створювати нові дошки, редагувати існуючі та видаляти їх. Кожна дошка відображається у вигляді картки з назвою і можливістю видалення.

2. Home (Головна): ця сторінка є стартовою точкою для користувачів, які щойно зайшли в додаток. Вона пропонує короткий огляд можливостей додатку, ключові функції, а також можливість реєстрації або входу в систему. Для зареєстрованих користувачів вона також містить посилання на дошки користувача.

3. Signup (Реєстрація): ця сторінка надає форму для реєстрації нового облікового запису в додатку. Вона запитує від користувача персональні дані, такі як ім’я, прізвище, ім’я користувача та пароль. Після успішної реєстрації користувач буде перенаправлений на сторінку входу. Якщо сталася помилка при реєстрації, вона повідомить користувача про це.

4. Login (Вхід): цей компонент відповідає за сторінку входу в додаток. Він містить форму для введення ім’я користувача та пароля. Після введення користувачем необхідних даних і натискання кнопки "Увійти", дані надсилаються на сервер для перевірки. Якщо вони вірні, користувач авторизується і перенаправляється до додатку, в іншому випадку він отримає повідомлення про невірне ім’я користувача або пароль. Крім того, на цій сторінці є посилання на форму реєстрації та можливість відновлення паролю.

Кожна з цих сторінок відповідає за певні функціональні можливості додатку і дозволяє користувачам взаємодіяти з ним у відповідний спосіб.

3.10 Взаємодія користувача з програмною системою

Запускаємо застосунок і бачимо головну сторінку

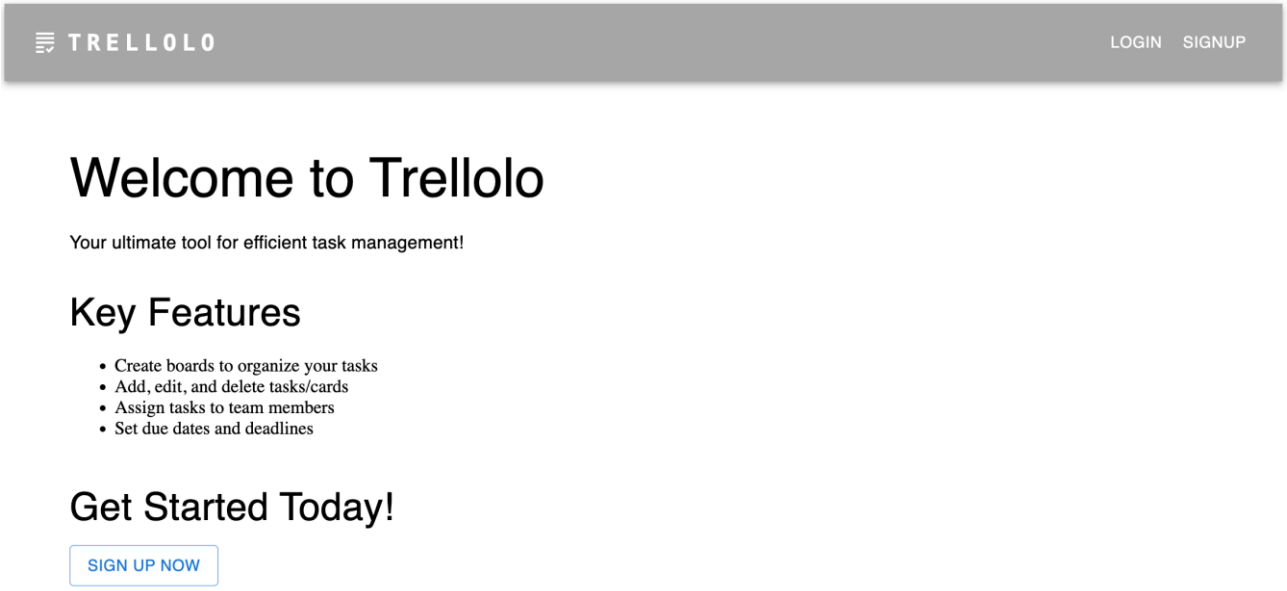


Рисунок 3.17 – Головна сторінка

Бачимо можливість зареєструватися або створити нового користувача, переходимо до реєстрації

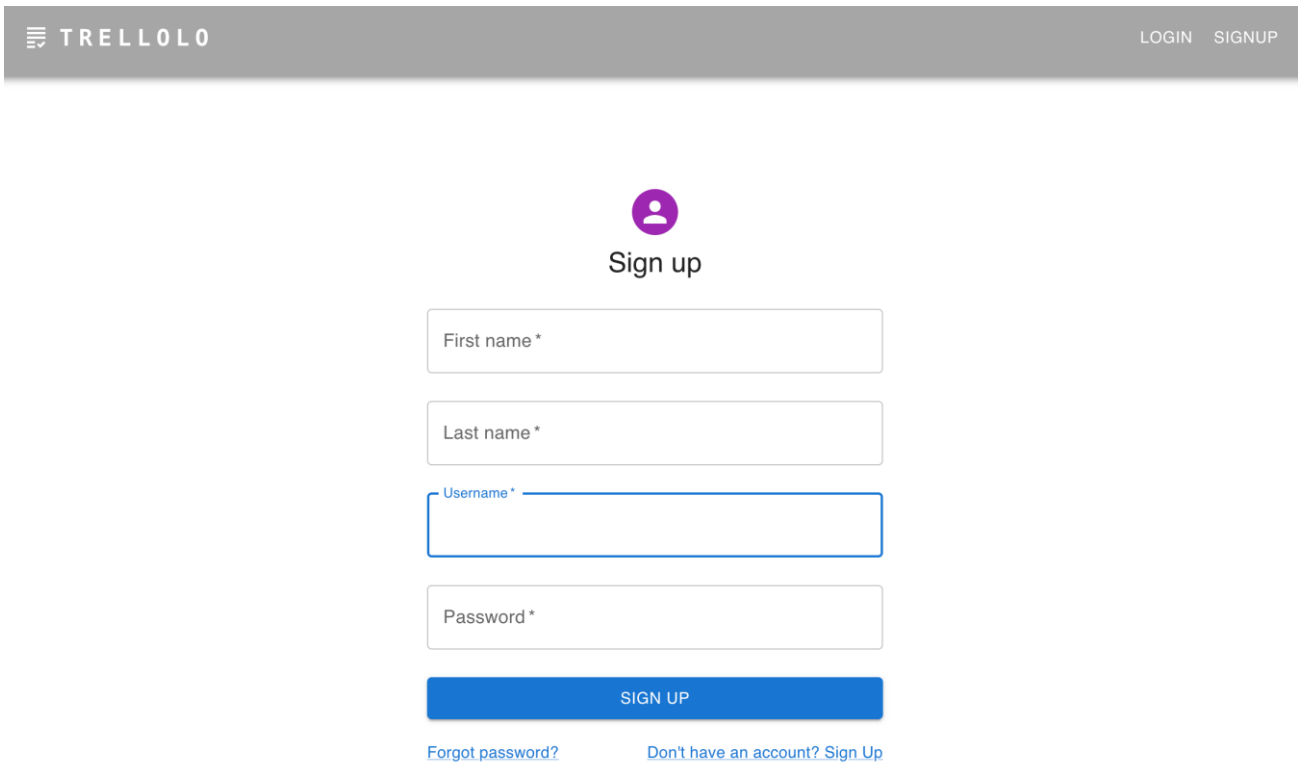


Рисунок 3.18 – Сторінка реєстрації

Перенаправлення на сторінку входу де користувач, вводить дані, для логування.

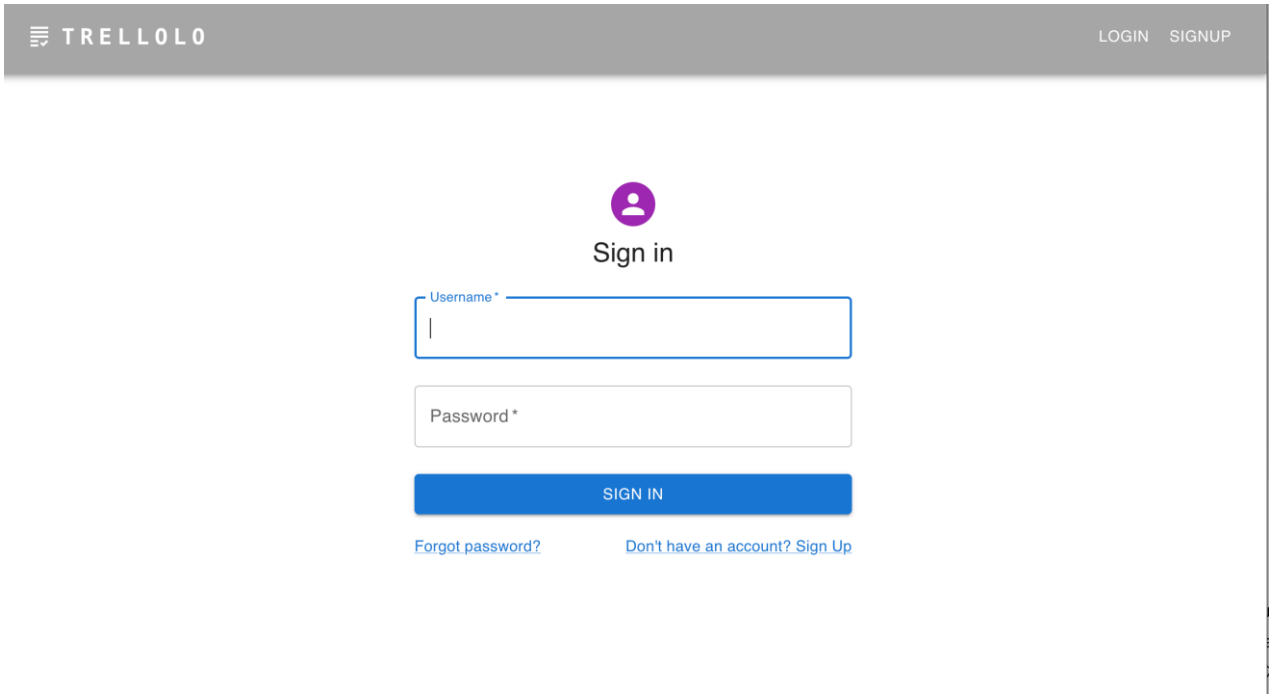


Рисунок 3.19 – Сторінка логіну

Після того як залогувалися в системі з’являється кнопка BOARDS на панелі вгорі, натискаємо і переходимо на сторінку зі своїми дошками. Маємо кнопки для створення дошки, відкриття вже існуючих та видалення.

Рисунок 3.20 – Сторінка Boards

Заходимо на свою дошку.

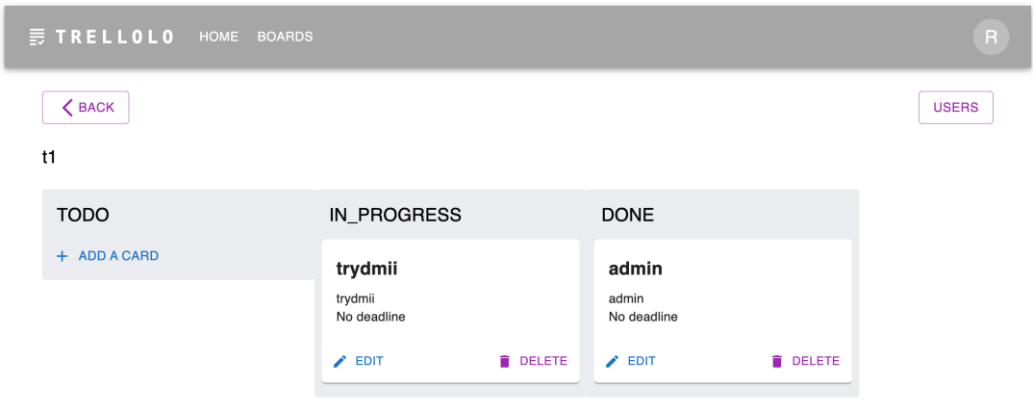


Рисунок 3.21 – Сторінка дошки

На сторінці дошки бачимо саму інтерактивну дошку, у якій маємо функціонал створення задач, зміни їх статусу, редагування, видалення.

Також маємо кнопку щоб повернутися назад і кнопку перегляду користувачів цієї дошки, натискаємо на останню.

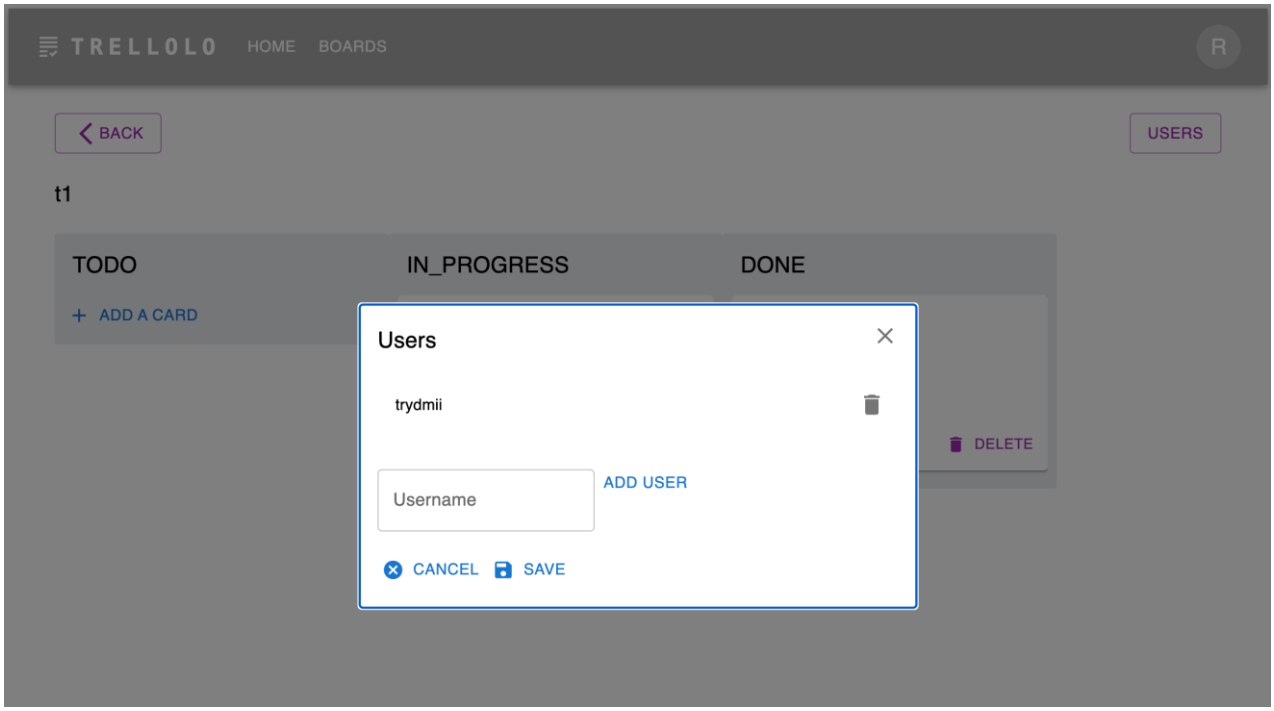


Рисунок 3.22 – Модальне вікно з користувачами дошки

Бачимо модальне вікно яке зображує користувачів, які були додані до дошки. Маємо можливість як власник дошки додати інших користувачів по імені користувача або видалити існуючих.

Повертаємость до дошки, пробуємо створити новий елемент, бачимо модальне вікно для створення куди треба ввести відповідні дані.

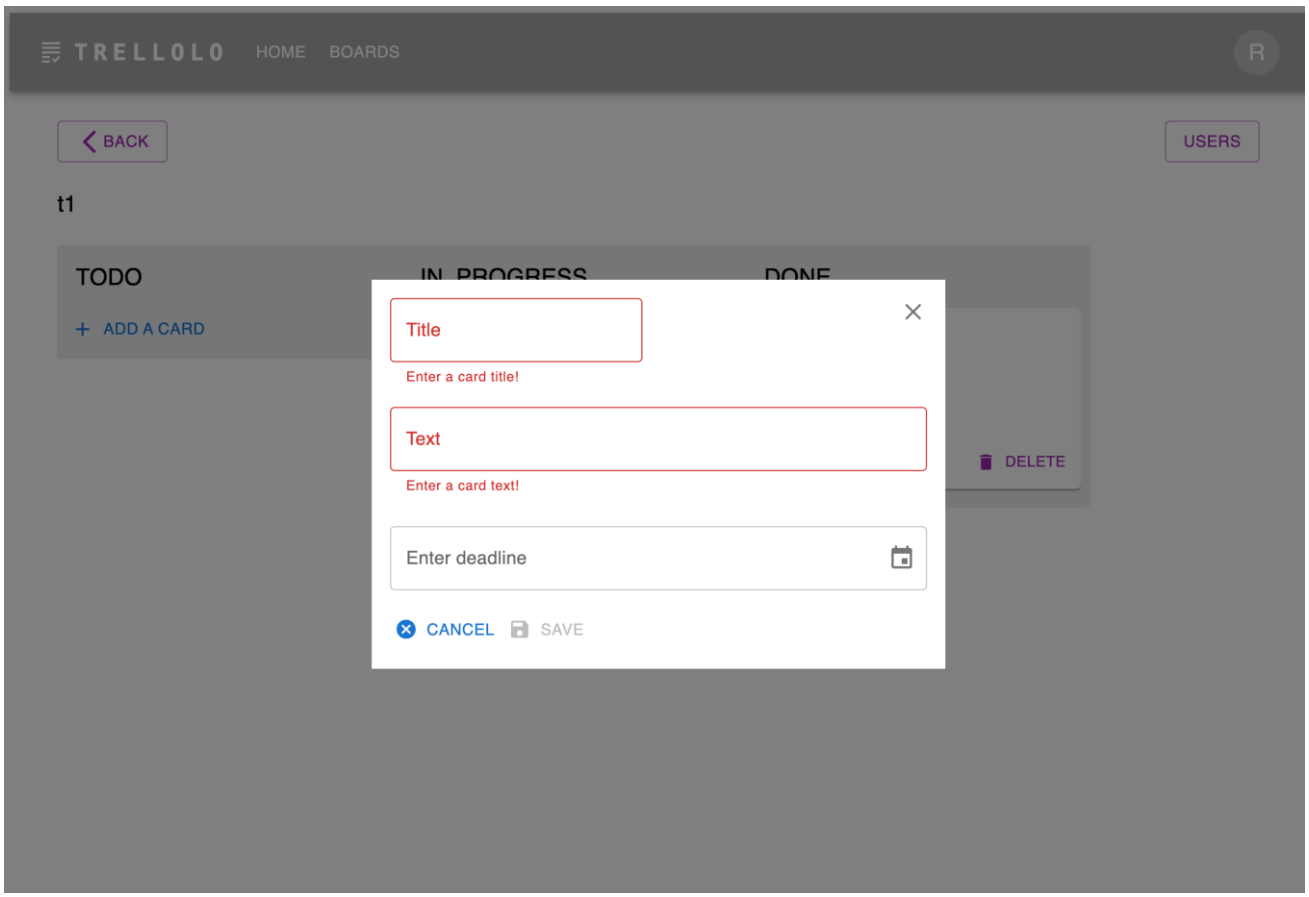


Рисунок 3.23– модальне вікно створення нового елементу дошки

На цьому процес розробки є успішно завершений.

ВИСНОВКИ

Під час розробки кваліфікаційної роботи було проведено значний обсяг робіт з аналізу, проєктування та реалізації інформаційної системи. Основною метою роботи було створення інструменту, що забезпечує ефективне керування проєктами та спільну роботу команди над ними. В результаті досліджень та аналізу сучасних тенденцій були обрані оптимальні технології та інструменти для реалізації функціоналу системи.

Під час проєктування була розроблена модель інформаційної системи, яка включала в себе визначення структури сторінок та компонентів. Реалізація системи передбачала можливість планування, організації та відстеження проєктів, а також спрощення взаємодії між учасниками команди.

Окрім цього, велика увага була приділена адаптивності і зручності використання системи на різних пристроях та роздільних здатностях екрану. Для досягнення цієї мети були використані передові методи верстки та технології.

Результатом роботи є повнофункціональна інформаційна система, що відповідає вимогам та потребам користувачів у сфері планування та управління проєктами. Отриманий продукт є основою для подальшого розширення та вдосконалення функціоналу з урахуванням потреб користувачів та нових вимог ринку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Frontend, Backend і Fullstack: що це таке та в чому відмінності. URL: https://goit.global/ua/articles/frontend-backend-i-fullstack-shcho-tse-take-ta-v-chomu-vidminnosti/ (дата звернення 04.05.2024).
2. Great outcomes start with Jira. URL: https://www.atlassian.com/software/jira. (дата звернення 04.05.2024).
3. Trello brings all your tasks, teammates, and tools together. URL: https://trello.com (дата звернення 04.05.2024).
4. Jira — орієнтація на корпоративні рішення. URL: https://iampm.club/ua/blog/jira-vs-trello-porivnyannya-funkczionalu-zruchnosti-vartosti/#:~:text=Недоліки%20Jira%20часто%20пов'язані,оптимізувати%20робочі%20процеси%20за%20потреби. (дата звернення 04.05.2024).
5. Що таке фреймворк?. URL: https://asabix.com.ua/what-is-a-framework (дата звернення 05.05.2024).
6. Переваги та недоліки використання Spring Boot. URL: https://javarush.com/ua/groups/posts/uk.3380.kava-breyk-75-perevagi-ta-nedolki-vikoristannja-spring-boot-funkc-dlja-rjadkv-u-java (дата звернення 04.05.2024).
7. Serverless, Micronaut Framework та розподілені системи: тренди у Java, на які варто звернути увагу. URL: https://dou.ua/lenta/articles/java-trends/ (дата звернення 04.05.2024).
8. Чи може Micronaut замінити Spring Boot? Розберемося на прикладі. URL: https://dou.ua/forums/topic/35583 (дата звернення 05.05.2024).
9. Порівнюємо React, Angular і Vue — найпопулярніші бібліотеки й фреймворки у 2022 році. URL: https://dou.ua/forums/topic/39933 (дата звернення 05.05.2024).
10. Типи баз даних: особливості, відмінності та приклади. URL: https://dou.ua/lenta/articles/types-of-databases (дата звернення 05.05.2024).

11. Getting started with PostgreSQL on Docker. URL: https://www.sqlshack.com/getting-started-with-postgresql-on-docker (дата звернення 05.05.2024).

12. Trellolo. URL: https://github.com/trydmii/trellolo/tree/master (дата звернення 06.05.2024).

ДОДАТКИ

Додаток А. SecurityConfig.java

```
package ua.trydmii.trellolo.config;

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

@Configuration
@RequiredArgsConstructor
@EnableWebSecurity
public class SecurityConfig {
    private final JwtFilter jwtFilter;

    private static final String ADMIN_ENDPOINT = "/api/v1/admin/**";
    private static final String AUTH_ENDPOINT = "/api/v1/auth/**";
    private static final String ADMIN_ROLE = "ADMIN";

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

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.authorizeHttpRequests(authz -> authz
                .requestMatchers(AUTH_ENDPOINT).permitAll()
                .requestMatchers(ADMIN_ENDPOINT).hasRole(ADMIN_ROLE)
                .anyRequest().authenticated()
        );
        return http
                .sessionManagement(session ->
session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .csrf(AbstractHttpConfigurer::disable)
                .addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class)
                .build();
    }
}
```

Додаток Б. ApiExceptionHandler.java

```
package ua.trydmii.trellolo.controller.exception;

import jakarta.persistence.EntityNotFoundException;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestControllerAdvice;
import ua.trydmii.trellolo.exception.*;

import java.nio.file.FileAlreadyExistsException;
import java.util.HashMap;
import java.util.Map;

@RestControllerAdvice
public class ApiExceptionHandler {
    @ExceptionHandler(value = {UsernameAlreadyExistsException.class})
    @ResponseStatus(HttpStatus.UNAUTHORIZED)
    public ResponseEntity<ErrorMessage>
handleUsernameAlreadyExistsException(UsernameAlreadyExistsException exception) {
        ErrorMessage customErrorMessage = new ErrorMessage(
                HttpStatus.UNAUTHORIZED.value(),
                exception.getMessage()
        );

        return new ResponseEntity<>(customErrorMessage, HttpStatus.UNAUTHORIZED);
    }

    @ExceptionHandler(value = {NotAuthenticatedException.class})
    @ResponseStatus(HttpStatus.UNAUTHORIZED)
    public ResponseEntity<ErrorMessage>
handleInvalidPasswordException(NotAuthenticatedException exception) {
        ErrorMessage customErrorMessage = new ErrorMessage(
                HttpStatus.UNAUTHORIZED.value(),
                exception.getMessage()
        );

        return new ResponseEntity<>(customErrorMessage, HttpStatus.UNAUTHORIZED);
    }

    @ExceptionHandler(MethodArgumentNotValidException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public Map<String, String> handleInvalidArgument(MethodArgumentNotValidException ex)
{
        Map<String, String> errorMap = new HashMap<>();
        ex.getBindingResult().getFieldErrors().forEach(error ->
errorMap.put(error.getField(), error.getDefaultMessage()));
        return errorMap;
    }

    @ExceptionHandler(value = {RefreshTokenExpiredException.class})
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public ResponseEntity<ErrorMessage>
handleRefreshTokenExpiredException(RefreshTokenExpiredException exception) {
        ErrorMessage customErrorMessage = new ErrorMessage(
                HttpStatus.BAD_REQUEST.value(),
                exception.getMessage()
        );

        return new ResponseEntity<>(customErrorMessage, HttpStatus.BAD_REQUEST);
    }

    @ExceptionHandler(value = {TokenExpiredException.class})
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public ResponseEntity<ErrorMessage> handleTokenExpiredException(TokenExpiredException
exception) {
        ErrorMessage customErrorMessage = new ErrorMessage(
```

```
                HttpStatus.BAD_REQUEST.value(),
                exception.getMessage()
        );

        return new ResponseEntity<>(customErrorMessage, HttpStatus.UNAUTHORIZED);
    }

    @ExceptionHandler(value = {EntityNotFoundException.class})
    public ResponseEntity<ErrorMessage> handleEntityNotFoundException() {
        ErrorMessage customErrorMessage = new ErrorMessage(
                HttpStatus.NOT_FOUND.value(),
                "Entity not found"
        );

        return new ResponseEntity<>(customErrorMessage, HttpStatus.NOT_FOUND);
    }

    @ExceptionHandler(value = {FileAlreadyExistsException.class})
    public ResponseEntity<ErrorMessage>
handleFileAlreadyExistsException(FileAlreadyExistsException exception) {
        ErrorMessage customErrorMessage = new ErrorMessage(
                HttpStatus.NOT_FOUND.value(),
                exception.getMessage()
        );

        return new ResponseEntity<>(customErrorMessage, HttpStatus.BAD_REQUEST);
    }

    @ExceptionHandler(value = {UserNotFoundException.class})
    public ResponseEntity<ErrorMessage> handleUserNotFoundException(UserNotFoundException
exception) {
        ErrorMessage customErrorMessage = new ErrorMessage(
                HttpStatus.NOT_FOUND.value(),
                exception.getMessage()
        );

        return new ResponseEntity<>(customErrorMessage, HttpStatus.BAD_REQUEST);
    }
}
```

Додаток В. ErrorMessage.java

```
package ua.trydmii.trellolo.controller.exception;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.Setter;

@AllArgsConstructor
@Getter
@Setter
public class ErrorMessage {
    private int httpStatusCode;
    private String messages;
}
```

Додаток Г. AuthControllerV1.java

```java
package ua.trydmii.trellolo.controller;

import io.jsonwebtoken.ExpiredJwtException;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.web.bind.annotation.*;
import ua.trydmii.trellolo.dto.*;
import ua.trydmii.trellolo.exception.NotAuthenticatedException;
import ua.trydmii.trellolo.exception.RefreshTokenExpiredException;
import ua.trydmii.trellolo.exception.UserNotFoundException;
import ua.trydmii.trellolo.exception.UsernameAlreadyExistsException;
import ua.trydmii.trellolo.mapper.CustomMapper;
import ua.trydmii.trellolo.model.RefreshToken;
import ua.trydmii.trellolo.model.User;
import ua.trydmii.trellolo.model.UserStatus;
import ua.trydmii.trellolo.security.jwt.JwtUserService;
import ua.trydmii.trellolo.security.jwt.JwtUtility;
import ua.trydmii.trellolo.service.RefreshTokenService;
import ua.trydmii.trellolo.service.UserService;

import java.util.List;

@RestController
@RequestMapping(value = "/api/v1/auth")
@RequiredArgsConstructor
public class AuthControllerV1 {
    private final UserService userService;
    private final CustomMapper mapper;
    private final JwtUserService jwtUserService;
    private final JwtUtility jwtUtility;
    private final RefreshTokenService refreshTokenService;
    private final AuthenticationManager authenticationManager;

    @PostMapping("/sign-up")
    public ResponseEntity<UserDto> register(@RequestBody RegisterRequestDto requestDto) {
        String username = requestDto.username();
        try {
            userService.findByUsername(username);
        } catch (UsernameNotFoundException e) {
            var user = mapper.requestToUser(requestDto);
            var registered = userService.register(user);
            var userDto = mapper.userToUserDto(registered);
            return new ResponseEntity<>(userDto, HttpStatus.ACCEPTED);
        }
        throw new UsernameAlreadyExistsException("username already exists");
    }

    @PostMapping("/sign-in")
    public ResponseEntity<AuthResponseDto> login(@RequestBody AuthRequestDto requestDto)
{
        User user;
        try {
            user = userService.findByUsername(requestDto.username());
        } catch (UserNotFoundException e) {
            throw new NotAuthenticatedException("invalid username or password");
        }
        if (user.getUserStatus().equals(UserStatus.BANNED)) {
            throw new NotAuthenticatedException("User is banned");
        }

        try {
```

```
            authenticationManager.authenticate(
                    new UsernamePasswordAuthenticationToken(
                            requestDto.username(),
                            requestDto.password()
                    )
            );
        } catch (AuthenticationException e) {
            throw new NotAuthenticatedException("Invalid username or password");
        }

        final UserDetails userDetails
                = jwtUserService.loadUserByUsername(requestDto.username());

        final String token =
                jwtUtility.generateToken(userDetails);

        final RefreshToken refreshToken =
refreshTokenService.createRefreshToken(user.getId());

        List<RoleDto> roleDtos = user.getRoles().stream()
                .map(mapper::roleToRoleDto).toList();
        return ResponseEntity.ok(new AuthResponseDto(token, refreshToken.getToken(),
roleDtos));
    }

    @PostMapping("/refresh")
    public ResponseEntity<RefreshTokenResponse> refreshToken(@RequestBody
RefreshRequestDto requestDto) {
        String refresh = requestDto.refreshToken();
        try {
            return refreshTokenService.findByToken(refresh)
                    .map(refreshTokenService::verifyExpiration)
                    .map(RefreshToken::getUser)
                    .map(user -> {
                        UserDetails userDetails =
jwtUserService.loadUserByUsername(user.getUsername());
                        String token = jwtUtility.generateToken(userDetails);
                        return new ResponseEntity<>(new RefreshTokenResponse(token,
refresh), HttpStatus.OK);
                    })
                    .orElseThrow(() -> new RefreshTokenExpiredException("refresh token
not in database"));
        } catch (ExpiredJwtException e) {
            throw new RefreshTokenExpiredException("refresh token not in database");
        }
    }

    @DeleteMapping("/logout")
    public ResponseEntity<HttpStatus> logoutUser(Authentication authentication) {
        String username = authentication.getName();
        User user = userService.findByUsername(username);
        refreshTokenService.deleteByUserId(user.getId());
        return new ResponseEntity<>(HttpStatus.OK);
    }
}
```

Додаток Ґ. BoardControllerV1.java

```
package ua.trydmii.trellolo.controller;

import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.*;
import ua.trydmii.trellolo.dto.BoardDto;
import ua.trydmii.trellolo.dto.BoardUsersDto;
import ua.trydmii.trellolo.model.User;
import ua.trydmii.trellolo.repository.UserRepository;
import ua.trydmii.trellolo.service.BoardService;

import java.util.List;

@RestController
@RequestMapping(value = "/api/v1/boards")
@RequiredArgsConstructor
public class BoardControllerV1 {
    private final BoardService boardService;
    private final UserRepository userRepository;

    @GetMapping
    public ResponseEntity<List<BoardDto>> getBoardsByCurrentUser(Authentication
authentication) {
        String username = authentication.getName();
        List<BoardDto> byOwner = boardService.findByOwner(username);
        List<BoardDto> boardsByUsername = boardService.findBoardsByUsername(username);
        List<BoardDto> result = byOwner.isEmpty() ? boardsByUsername : byOwner;
        return new ResponseEntity<>(result, HttpStatus.OK);
    }

    @GetMapping("/{id}")
    public ResponseEntity<BoardDto> getBoardById(@PathVariable Long id) {
        return new ResponseEntity<>(boardService.findById(id), HttpStatus.OK);
    }

    @PostMapping
    public ResponseEntity<BoardDto> createBoard(@RequestBody BoardDto boardDto,
Authentication authentication) {
        String username = authentication.getName();
        return new ResponseEntity<>(boardService.createBoard(boardDto, username),
HttpStatus.CREATED);
    }

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

    @GetMapping("/users/{id}")
    public ResponseEntity<List<BoardUsersDto>> getBoardUsers(@PathVariable Long id) {
        return new ResponseEntity<>(boardService.getBoardUsers(id), HttpStatus.OK);
    }

    @PostMapping("/users/{id}")
    public ResponseEntity<HttpStatus> saveBoardUsers(@PathVariable Long id, @RequestBody
List<String> boardUsersDto) {
        boardService.saveBoardUsers(id, boardUsersDto);
        return new ResponseEntity<>(HttpStatus.OK);
    }

    @GetMapping("/owner/{id}")
    public ResponseEntity<HttpStatus> isOwner(@PathVariable Long id, Authentication
authentication) {
        User user =
userRepository.findByUsername(authentication.getName()).orElseThrow(() -> new
```

```
RuntimeException("User not found"));
        boolean isOwner = boardService.isOwner(id, user.getId());
        if (isOwner) {
            return new ResponseEntity<>(HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.FORBIDDEN);
        }
    }
}
```

Додаток Д. BoardItemControllerV1.java

```
package ua.trydmii.trellolo.controller;

import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ua.trydmii.trellolo.dto.BoardItemCreateRequestDto;
import ua.trydmii.trellolo.dto.BoardItemDto;
import ua.trydmii.trellolo.dto.BoardItemStatusChangeRequestDto;
import ua.trydmii.trellolo.dto.BoardItemUpdateDto;
import ua.trydmii.trellolo.service.BoardItemService;

import java.util.List;

@RestController
@RequestMapping(value = "/api/v1/boardItems")
@RequiredArgsConstructor
public class BoardItemControllerV1 {
    private final BoardItemService boardItemService;

    @GetMapping("/{id}")
    public ResponseEntity<List<BoardItemDto>> getBoardItemsById(@PathVariable Long id) {
        return new ResponseEntity<>(boardItemService.findById(id), HttpStatus.OK);
    }

    @PostMapping("/{id}")
    public ResponseEntity<BoardItemDto> createBoardItem(@PathVariable Long id,
@RequestBody BoardItemCreateRequestDto boardItemDto) {
        return new ResponseEntity<>(boardItemService.createBoardItem(id, boardItemDto),
HttpStatus.CREATED);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<HttpStatus> deleteBoardItemById(@PathVariable String id) {
        boardItemService.deleteBoardItemById(Long.parseLong(id));
        return new ResponseEntity<>(HttpStatus.OK);
    }

    @PatchMapping
    public ResponseEntity<HttpStatus> updateBoardItemStatuses(@RequestBody
List<BoardItemStatusChangeRequestDto> boardItemStatusChangeRequestDto) {
        boardItemService.updateBoardItemStatus(boardItemStatusChangeRequestDto);
        return new ResponseEntity<>(HttpStatus.OK);
    }

    @PutMapping
    public ResponseEntity<HttpStatus> updateBoardItem(@RequestBody BoardItemUpdateDto
boardItemDto) {
        boardItemService.updateBoardItem(boardItemDto);
        return new ResponseEntity<>(HttpStatus.OK);
    }
}
```

Додаток Е. AuthRequestDto.java

```java
package ua.trydmii.trellolo.dto;

public record AuthRequestDto(String username, String password) {
}
```

Додаток Є. AuthResponseDto.java

```
package ua.trydmii.trellolo.dto;

import java.util.List;

public record AuthResponseDto(String accessToken, String refreshToken, List<RoleDto>
roles) {
}
```

Додаток Ж. BoardDto.java

```java
package ua.trydmii.trellolo.dto;

import java.time.LocalDateTime;

public record BoardDto(Long id, String name, String owner, String createdAt) {
}
```

Додаток 3. BoardItemCreateRequestDto.java

```
package ua.trydmii.trellolo.dto;

import com.fasterxml.jackson.annotation.JsonFormat;
import com.fasterxml.jackson.annotation.JsonInclude;
import ua.trydmii.trellolo.model.enums.CardStatus;

import java.time.LocalDateTime;

@JsonInclude(JsonInclude.Include.NON_NULL)
public record BoardItemCreateRequestDto(String title, String text, CardStatus status,
Integer index,
                                        @JsonFormat(pattern = "yyyy-MM-
dd'T'HH:mm:ss.SSSXXX")
                                        LocalDateTime deadline) {
}
```

Додаток И. BoardItemDto.java

```
package ua.trydmii.trellolo.dto;

import com.fasterxml.jackson.annotation.JsonFormat;
import com.fasterxml.jackson.annotation.JsonInclude;
import ua.trydmii.trellolo.model.enums.CardStatus;

import java.time.LocalDateTime;

@JsonInclude(JsonInclude.Include.NON_NULL)
public record BoardItemDto(Long id, String title, String text, CardStatus status, Integer
index,
                           @JsonFormat(pattern = "yyyy-MM-dd'T'HH:mm:ss")
                           LocalDateTime deadline,
                           LocalDateTime createdAt) {
}
```

Додаток І. BoardItemStatusChangeRequestDto.java

```
package ua.trydmii.trellolo.dto;

public record BoardItemStatusChangeRequestDto(Long id, String status, Integer index) {
}
```

Додаток Ї. BoardItemUpdateDto.java

```
package ua.trydmii.trellolo.dto;

import com.fasterxml.jackson.annotation.JsonFormat;
import com.fasterxml.jackson.annotation.JsonInclude;
import ua.trydmii.trellolo.model.enums.CardStatus;

import java.time.LocalDateTime;

@JsonInclude(JsonInclude.Include.NON_NULL)
public record BoardItemUpdateDto(Long id, String title, String text, CardStatus status,
Integer index,
                           @JsonFormat(pattern = "yyyy-MM-dd'T'HH:mm:ss.SSSXXX")
                           LocalDateTime deadline,
                           LocalDateTime createdAt) {
}
```

Додаток Й. BoardUsersDto.java

```
package ua.trydmii.trellolo.dto;

public record BoardUsersDto(String username) {
}
```

Додаток К. RefreshRequestDto.java

```
package ua.trydmii.trellolo.dto;

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

Додаток Л. RefreshResponseDto.java

```
package ua.trydmii.trellolo.dto;

public record RefreshTokenResponse(String accessToken, String refreshToken) {
}
```

Додаток М. RegisterRequestDto.java

```
package ua.trydmii.trellolo.dto;

public record RegisterRequestDto(
        String username,
        String firstName,
        String lastName,
        String password
) {
}
```

Додаток Н. RoleDto.java

```
package ua.trydmii.trellolo.dto;

public record RoleDto(String name) {
}
```

Додаток О. UserDto.java

```
package ua.trydmii.trellolo.dto;

import com.fasterxml.jackson.annotation.JsonInclude;

import java.util.List;

@JsonInclude(JsonInclude.Include.NON_EMPTY)
public record UserDto(
        String username,
        String firstName,
        String lastName,
        String userStatus,
        List<RoleDto> roles
) {
}
```

Додаток П. CustomMapper.java

```java
package ua.trydmii.trellolo.mapper;

import org.mapstruct.Mapper;
import org.mapstruct.Mapping;
import org.mapstruct.Named;
import ua.trydmii.trellolo.dto.*;
import ua.trydmii.trellolo.model.Board;
import ua.trydmii.trellolo.model.BoardItem;
import ua.trydmii.trellolo.model.Role;
import ua.trydmii.trellolo.model.User;

@Mapper(componentModel = "spring")
public interface CustomMapper {
    @Mapping(target = "userStatus", ignore = true)
    UserDto userToUserDto(User user);

    @Mapping(target = "userStatus", ignore = true)
    @Mapping(target = "updated", ignore = true)
    @Mapping(target = "created", ignore = true)
    @Mapping(target = "roles", ignore = true)
    User requestToUser(RegisterRequestDto requestDto);

    RoleDto roleToRoleDto(Role role);

    @Mapping(target = "owner", source = "owner", qualifiedByName = "mapUserToUsername")
    BoardDto boardToBoardDto(Board board);

    @Named("mapUserToUsername")
    default String mapProducts(User user) {
        return user.getUsername();
    }

    @Mapping(target = "id", source = "boardDto.id")
    @Mapping(target = "owner", source = "owner")
    Board boardDtoToBoard(BoardDto boardDto, User owner);

    BoardItemDto boardItemToBoardItemDto(BoardItem boardItem);

    @Mapping(target = "board", ignore = true)
    @Mapping(target = "createdAt", ignore = true)
    @Mapping(target = "id", ignore = true)
    BoardItem boardItemDtoToBoardItem(BoardItemCreateRequestDto boardItemDto);

    BoardUsersDto userToBoardUsersDto(User user);
}
```

Додаток Р. BaseEntity.java

```java
package ua.trydmii.trellolo.model;

import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.MappedSuperclass;
import lombok.*;
import lombok.experimental.SuperBuilder;

@MappedSuperclass
@Getter
@Setter
@ToString
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
public class BaseEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
}
```

Додаток С. Board.java

```java
package ua.trydmii.trellolo.model;

import jakarta.persistence.*;
import lombok.*;
import lombok.experimental.SuperBuilder;

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

@Entity
@Getter
@Setter
@ToString
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "boards")
public class Board extends BaseEntity {
    @Column(name = "name")
    private String name;

    @ManyToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id")
    private User owner;

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

    @ManyToMany(mappedBy = "boards", fetch = FetchType.EAGER)
    private List<User> users;
}
```

Додаток Т. BoardItem.java

```
package ua.trydmii.trellolo.model;

import jakarta.persistence.*;
import lombok.*;
import lombok.experimental.SuperBuilder;
import ua.trydmii.trellolo.model.enums.CardStatus;

import java.time.LocalDateTime;

@Entity
@Getter
@Setter
@ToString
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "boardItems")
public class BoardItem extends BaseEntity {
    @Column(name = "title")
    private String title;

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

    @ManyToOne
    @JoinColumn(name = "board_id", referencedColumnName = "id")
    private Board board;

    @Enumerated(EnumType.STRING)
    private CardStatus status;

    @Column(name = "index")
    private Integer index;

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

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

Додаток У. RefreshToken.java

```java
package ua.trydmii.trellolo.model;

import jakarta.persistence.*;
import lombok.*;
import lombok.experimental.SuperBuilder;

import java.time.Instant;

@EqualsAndHashCode(callSuper = true)
@Entity
@Table(name = "refresh_tokens")
@Data
@AllArgsConstructor
@ToString(exclude = "user")
@SuperBuilder
@NoArgsConstructor
public class RefreshToken extends BaseEntity {
    @OneToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id")
    private User user;

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

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

Додаток Ф. Role.java

```java
package ua.trydmii.trellolo.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.RequiredArgsConstructor;
import lombok.Setter;
import lombok.ToString;
import lombok.experimental.SuperBuilder;

import java.util.List;

@Getter
@Setter
@ToString
@RequiredArgsConstructor
@Entity
@SuperBuilder
@Table(name = "roles")
public class Role extends BaseEntity {
    @Column(name = "name")
    private String name;

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

Додаток X. User.java

```
package ua.trydmii.trellolo.model;

import jakarta.persistence.*;
import lombok.*;
import lombok.experimental.SuperBuilder;

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

@Entity
@Getter
@Setter
@ToString
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "users")
public class User extends BaseEntity {
    @Column(name = "username")
    private String username;

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

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

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

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

    @Column(name = "user_status")
    @Enumerated(EnumType.STRING)
    private UserStatus userStatus;

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

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

    @ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @JoinTable(
            name = "board_users",
            joinColumns = { @JoinColumn(name = "user_id", referencedColumnName = "id") },
            inverseJoinColumns = { @JoinColumn(name = "board_id", referencedColumnName =
"id") }
    )
    private List<Board> boards;
}
```

Додаток Ц. BoardItemRepository.java

```java
package ua.trydmii.trellolo.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import ua.trydmii.trellolo.model.BoardItem;

import java.util.List;

public interface BoardItemRepository extends JpaRepository<BoardItem, Long> {
    List<BoardItem> findByBoardId(Long boardId);
}
```

Додаток Ч. BoardRepository.java

```
package ua.trydmii.trellolo.repository;

import jakarta.transaction.Transactional;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import ua.trydmii.trellolo.model.Board;
import ua.trydmii.trellolo.model.User;

import java.util.List;

public interface BoardRepository extends JpaRepository<Board, Long> {
    List<Board> findBoardByOwner(User user);

    Board findBoardById(Long id);

    @Query("SELECT u FROM User u JOIN u.boards b WHERE b.id = :boardId")
    List<User> findAllUsersByBoardId(@Param("boardId") Long boardId);

    @Query("SELECT b FROM Board b JOIN b.users u WHERE u.username = :username")
    List<Board> findBoardsByUsername(@Param("username") String username);

    @Transactional
    @Modifying
    @Query(value = "INSERT INTO board_users (board_id, user_id) VALUES (:boardId,
:userId)",
            nativeQuery = true)
    void saveUserToBoard(@Param("boardId") Long boardId, @Param("userId") Long userId);

    @Transactional
    @Modifying
    @Query(value = "DELETE FROM board_users WHERE board_id = :boardId", nativeQuery =
true)
    void deleteAllUsersFromBoard(@Param("boardId") Long boardId);
}
```

Додаток Ш. RefreshTokenRepository.java

```java
package ua.trydmii.trellolo.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import ua.trydmii.trellolo.model.RefreshToken;
import ua.trydmii.trellolo.model.User;

import java.util.Optional;

public interface RefreshTokenRepository extends JpaRepository<RefreshToken, Long> {
    Optional<RefreshToken> findByToken(String token);

    @Modifying
    int deleteByUser(User user);
}
```

Додаток Щ. RoleRepository.java

```java
package ua.trydmii.trellolo.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import ua.trydmii.trellolo.model.Role;

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

Додаток Ь. UserRepository.java

```java
package ua.trydmii.trellolo.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import ua.trydmii.trellolo.model.User;

import java.util.Optional;

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

Додаток Ю. JwtFilter.java

```
package ua.trydmii.trellolo.security.jwt;

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

import java.io.IOException;

@Component
@RequiredArgsConstructor
public class JwtFilter extends OncePerRequestFilter {
    private final JwtUtility jwtUtility;
    private final JwtUserService userService;

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

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

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

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

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


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

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

```
        }
}
```

Додаток Ю. JwtUserService.java

```
package ua.trydmii.trellolo.security.jwt;

import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import ua.trydmii.trellolo.model.Role;
import ua.trydmii.trellolo.service.UserService;

import java.util.Collection;

@Service
@Slf4j
@RequiredArgsConstructor
public class JwtUserService implements UserDetailsService {
    private final UserService userService;

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

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

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

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

Додаток Я. JwtUtility.java

```
package ua.trydmii.trellolo.security.jwt;

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

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

@Component
public class JwtUtility implements Serializable {

    @Serial
    private static final long serialVersionUID = 234234523523L;

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

    @Value("${jwt.secret}")
    private String secretKey;

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

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

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

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

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

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

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

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

}
```

Додаток А1. BoardItemServiceImpl.java

```
package ua.trydmii.trellolo.service.impl;

import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;
import ua.trydmii.trellolo.dto.BoardItemCreateRequestDto;
import ua.trydmii.trellolo.dto.BoardItemDto;
import ua.trydmii.trellolo.dto.BoardItemStatusChangeRequestDto;
import ua.trydmii.trellolo.dto.BoardItemUpdateDto;
import ua.trydmii.trellolo.mapper.CustomMapper;
import ua.trydmii.trellolo.model.BoardItem;
import ua.trydmii.trellolo.model.enums.CardStatus;
import ua.trydmii.trellolo.repository.BoardItemRepository;
import ua.trydmii.trellolo.repository.BoardRepository;
import ua.trydmii.trellolo.service.BoardItemService;

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

@Service
@Slf4j
@RequiredArgsConstructor
public class BoardItemServiceImpl implements BoardItemService {
    private final BoardItemRepository boardItemRepository;
    private final BoardRepository boardRepository;
    private final CustomMapper mapper;

    @Override
    public List<BoardItemDto> findById(Long boardId) {
        List<BoardItemDto> boardItems =
boardItemRepository.findByBoardId(boardId).stream().map(mapper::boardItemToBoardItemDto).
toList();
        log.info("found board items by id: {}: {}", boardId, boardItems);
        return boardItems;
    }

    @Override
    @Transactional
    public BoardItemDto createBoardItem(Long boardId, BoardItemCreateRequestDto
boardItemDto) {
        BoardItem boardItem = mapper.boardItemDtoToBoardItem(boardItemDto);
        boardItem.setCreatedAt(LocalDateTime.now());
        boardRepository.findById(boardId).ifPresent(boardItem::setBoard);
        return mapper.boardItemToBoardItemDto(boardItemRepository.save(boardItem));
    }

    @Override
    public void deleteBoardItemById(Long id) {
        boardItemRepository.deleteById(id);
        log.info("tried to delete board item with id: {}", id);
    }

    @Override
    @Transactional
    public void updateBoardItemStatus(List<BoardItemStatusChangeRequestDto>
boardItemStatusChangeRequestDto) {
        boardItemStatusChangeRequestDto.forEach(requestDto -> {
            BoardItem boardItem = boardItemRepository.findById(requestDto.id())
                    .orElseThrow(() -> new IllegalArgumentException("Board item not
found"));
            boardItem.setStatus(CardStatus.valueOf(requestDto.status()));
            boardItem.setIndex(requestDto.index());
        });
    }

    @Override
    @Transactional
```

```
    public void updateBoardItem(BoardItemUpdateDto boardItemDto) {
        BoardItem boardItem = boardItemRepository.findById(boardItemDto.id())
                .orElseThrow(() -> new IllegalArgumentException("Board item not found"));
        boardItem.setTitle(boardItemDto.title());
        boardItem.setText(boardItemDto.text());
        boardItem.setDeadline(boardItemDto.deadline());
        boardItem.setStatus(CardStatus.valueOf(boardItemDto.status().name()));
    }
}
```

Додаток Б1. BoardServiceImpl.java

```
package ua.trydmii.trellolo.service.impl;

import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;
import ua.trydmii.trellolo.dto.BoardDto;
import ua.trydmii.trellolo.dto.BoardUsersDto;
import ua.trydmii.trellolo.exception.BoardNotFoundException;
import ua.trydmii.trellolo.exception.UserNotFoundException;
import ua.trydmii.trellolo.mapper.CustomMapper;
import ua.trydmii.trellolo.model.BaseEntity;
import ua.trydmii.trellolo.model.Board;
import ua.trydmii.trellolo.model.BoardItem;
import ua.trydmii.trellolo.model.User;
import ua.trydmii.trellolo.repository.BoardItemRepository;
import ua.trydmii.trellolo.repository.BoardRepository;
import ua.trydmii.trellolo.repository.UserRepository;
import ua.trydmii.trellolo.service.BoardService;

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

@Service
@Slf4j
@RequiredArgsConstructor
public class BoardServiceImpl implements BoardService {
    private final BoardRepository boardRepository;
    private final BoardItemRepository boardItemRepository;
    private final UserRepository userRepository;
    private final CustomMapper mapper;

    @Override
    public List<BoardDto> findByOwner(String username) {
        User user = userRepository.findByUsername(username)
                .orElseThrow(() -> new UserNotFoundException("User not found"));
        return
boardRepository.findBoardByOwner(user).stream().map(mapper::boardToBoardDto).toList();
    }

    @Override
    public BoardDto findById(Long id) {
        return boardRepository.findById(id)
                .map(mapper::boardToBoardDto)
                .orElseThrow(() -> new BoardNotFoundException("Board not found"));
    }

    @Override
    public BoardDto createBoard(BoardDto boardDto, String username) {
        User user = userRepository.findByUsername(username)
                .orElseThrow(() -> new UserNotFoundException("User not found"));
        Board board = mapper.boardDtoToBoard(boardDto, user);
        board.setCreatedAt(LocalDateTime.now());
        return mapper.boardToBoardDto(boardRepository.save(board));
    }

    @Override
    @Transactional
    public void deleteBoardById(Long id) {
        List<BoardItem> boardItems = boardItemRepository.findByBoardId(id);
        boardItemRepository.deleteAll(boardItems);
        boardRepository.deleteById(id);
        log.info("board id: {} deleted with items ids: {}", id,
boardItems.stream().map(BaseEntity::getId).toList());
    }

    @Override
    public List<BoardUsersDto> getBoardUsers(Long id) {
```

```
        log.info("get board users by board id: {}", id);
        return
boardRepository.findAllUsersByBoardId(id).stream().map(mapper::userToBoardUsersDto).toLis
t();
    }

    @Override
    @Transactional
    public void saveBoardUsers(Long id, List<String> boardUsersDto) {
        boardRepository.deleteAllUsersFromBoard(id);
        boardUsersDto.forEach(username -> boardRepository.saveUserToBoard(id,
userRepository.findByUsername(username)
                .orElseThrow(() -> new UserNotFoundException("User not found"))
                .getId()));
    }


    @Override
    public List<BoardDto> findBoardsByUsername(String username) {
        return
boardRepository.findBoardsByUsername(username).stream().map(mapper::boardToBoardDto).toLi
st();
    }

    @Override
    public boolean isOwner(Long boardId, Long userId) {
        Board boardById = boardRepository.findBoardById(boardId);
        return boardById.getOwner().getId().equals(userId);
    }
}
```

Додаток В1. RefreshTokenServiceImpl.java

```
package ua.trydmii.trellolo.service.impl;

import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import ua.trydmii.trellolo.exception.RefreshTokenExpiredException;
import ua.trydmii.trellolo.exception.UserNotFoundException;
import ua.trydmii.trellolo.model.RefreshToken;
import ua.trydmii.trellolo.repository.RefreshTokenRepository;
import ua.trydmii.trellolo.repository.UserRepository;
import ua.trydmii.trellolo.service.RefreshTokenService;

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

@Service
@Slf4j
@RequiredArgsConstructor
public class RefreshTokenServiceImpl implements RefreshTokenService {

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

    private final UserRepository userRepository;
    private final RefreshTokenRepository refreshTokenRepository;

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

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

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

        refreshToken = refreshTokenRepository.save(refreshToken);
        log.info("IN RefreshTokenServiceImpl.createRefreshToken - refreshToken: {}
successfully created", refreshToken);
        return refreshToken;
    }

    public RefreshToken verifyExpiration(RefreshToken token) {
        if (token.getExpiryDate().compareTo(Instant.now()) < 0) {
            refreshTokenRepository.delete(token);
            log.warn("IN RefreshTokenServiceImpl.verifyExpiration - refreshToken: {}
expired", token);
            throw new RefreshTokenExpiredException("refresh token expired");
        }
        return token;
    }

    @Transactional
    public void deleteByUserId(Long userId) {

refreshTokenRepository.deleteByUser(userRepository.findById(userId).orElseThrow(() -> new
UserNotFoundException("user not found")));
        log.warn("IN RefreshTokenServiceImpl.deleteByUserId - refreshTokens for user with
id {} successfully deleted", userId);
    }

}
```

Додаток Г1. UserServiceImpl.java

```
package ua.trydmii.trellolo.service.impl;

import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import ua.trydmii.trellolo.model.Role;
import ua.trydmii.trellolo.model.User;
import ua.trydmii.trellolo.model.UserStatus;
import ua.trydmii.trellolo.repository.RoleRepository;
import ua.trydmii.trellolo.repository.UserRepository;
import ua.trydmii.trellolo.service.UserService;

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

@Service
@Slf4j
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {
    private static final String USER_NOT_FOUND_MESSAGE = "user not found";

    private final UserRepository userRepository;
    private final RoleRepository roleRepository;
    private final PasswordEncoder passwordEncoder;

    @Override
    public User findByUsername(String username) {
        return userRepository.findByUsername(username)
                .orElseThrow(() -> new
UsernameNotFoundException(USER_NOT_FOUND_MESSAGE));
    }

    @Override
    public User register(User user) {
        Role roleUser = roleRepository.findByName("ROLE_USER");

        user.setPassword(passwordEncoder.encode(user.getPassword()));
        user.setRoles(List.of(roleUser));
        user.setCreated(LocalDateTime.now());
        user.setUpdated(LocalDateTime.now());
        user.setUserStatus(UserStatus.ACTIVE);

        User registeredUser = userRepository.save(user);

        log.info("IN UserServiceImpl.register - user: {} successfully registered",
registeredUser);
        return registeredUser;
    }
}
```

Додаток Д1. V1__initial_script.sql

```
CREATE TABLE users
(
    id          BIGSERIAL    NOT NULL PRIMARY KEY,
    username    varchar(50)  NOT NULL UNIQUE,
    first_name  varchar(50)  NOT NULL,
    last_name   varchar(50)  NOT NULL,
    password    varchar(150) NOT NULL,
    user_status varchar(50)  NOT NULL DEFAULT 'ACTIVE',
    created_at  TIMESTAMP    NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at  TIMESTAMP    NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE roles
(
    id   BIGSERIAL   NOT NULL PRIMARY KEY,
    name varchar(50) NOT NULL UNIQUE
);

CREATE TABLE user_roles
(
    user_id bigint NOT NULL,
    role_id bigint NOT NULL,
    FOREIGN KEY (role_id) REFERENCES roles (id) ON DELETE CASCADE ON UPDATE RESTRICT,
    FOREIGN KEY (user_id) REFERENCES users (id) ON DELETE CASCADE ON UPDATE RESTRICT
);

INSERT INTO users (username, first_name, last_name, password)
VALUES ('admin', 'Dmytro', 'Pushkar',
'$2b$08$Qe.7Yq/NWGUhDWMkb6VjuO73YKzvhWTVexwF4YGB/uXlZXnQJgdra');

INSERT INTO roles (name)
VALUES ('ROLE_USER'),
       ('ROLE_ADMIN');

INSERT INTO user_roles (user_id, role_id)
VALUES (1, 2),
       (1, 1);
```

Додаток Е1. V1__refresh_tokens_table.sql

```
CREATE TABLE IF NOT EXISTS refresh_tokens
(
    id         BIGSERIAL PRIMARY KEY,
    user_id    BIGINT       NOT NULL,
    token      VARCHAR(255) NOT NULL UNIQUE,
    expiration TIMESTAMP    NOT NULL,
    foreign key (user_id) references users (id)
);
```

Додаток Є1. V1__boards.sql

```
CREATE TABLE boards
(
    id         BIGSERIAL   NOT NULL PRIMARY KEY,
    name       varchar(50) NOT NULL,
    user_id    BIGINT      NOT NULL,
    created_at TIMESTAMP   NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users (id)
);

CREATE TABLE board_items
(
    id         BIGSERIAL     NOT NULL PRIMARY KEY,
    title      varchar(50)   NOT NULL,
    text       varchar(1500) NOT NULL,
    board_id   BIGINT        NOT NULL,
    created_at TIMESTAMP     NOT NULL DEFAULT CURRENT_TIMESTAMP,
    status     varchar(50)   NOT NULL,
    index      INT           NOT NULL,
    FOREIGN KEY (board_id) REFERENCES boards (id)
);
```

Додаток Ж1. V1__board_items_deadlines_comments_teammates.sql

```
ALTER TABLE board_items
    ADD COLUMN deadline TIMESTAMP;

CREATE TABLE board_users
(
    user_id  bigint not null,
    board_id bigint not null,
    FOREIGN KEY (user_id) REFERENCES users (id) ON DELETE CASCADE ON UPDATE RESTRICT,
    FOREIGN KEY (board_id) REFERENCES boards (id) ON DELETE CASCADE ON UPDATE RESTRICT
);

CREATE TABLE board_item_comments
(
    id            BIGSERIAL    NOT NULL PRIMARY KEY,
    text          varchar(500) not null,
    user_id       bigint       not null,
    board_item_id bigint       not null,
    created_at    TIMESTAMP    NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

Додаток 31. application-dev.yml

```yaml
server:
  port: 8075

spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/postgres
    username: postgres
    password: postgres
    driver-class-name: org.postgresql.Driver

jwt:
  secret: myjwtsecretword
  expiration: 900000
  refreshExpiration: 1209600000
```

Додаток И1. pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.5</version>
        <relativePath/>
    </parent>

    <groupId>ua.trydmii</groupId>
    <artifactId>trellolo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>trellolo</name>
    <description>trellolo</description>

    <properties>
        <java.version>17</java.version>
        <mapstruct.version>1.5.5.Final</mapstruct.version>
        <jwt.version>0.9.1</jwt.version>
        <javax.xml.bind.version>2.3.1</javax.xml.bind.version>
        <quartz.version>2.3.2</quartz.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.flywaydb</groupId>
            <artifactId>flyway-core</artifactId>
        </dependency>

        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <scope>runtime</scope>
        </dependency>

        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
```

```
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.mapstruct</groupId>
            <artifactId>mapstruct</artifactId>
            <version>${mapstruct.version}</version>
        </dependency>

        <dependency>
            <groupId>org.mapstruct</groupId>
            <artifactId>mapstruct-processor</artifactId>
            <version>${mapstruct.version}</version>
        </dependency>

        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>${jwt.version}</version>
        </dependency>

        <dependency>
            <groupId>javax.xml.bind</groupId>
            <artifactId>jaxb-api</artifactId>
            <version>${javax.xml.bind.version}</version>
        </dependency>

        <dependency>
            <groupId>org.quartz-scheduler</groupId>
            <artifactId>quartz</artifactId>
            <version>${quartz.version}</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

Додаток І1. cardActions.js

```
import { ACTION_TYPES } from './index';

export const addCard = (listID, title, text) => {
  return {
    type: ACTION_TYPES.ADD_CARD,
    payload: { listID, title, text }
  };
};

export const deleteCard = (listID, cardID) => {
  return {
    type: ACTION_TYPES.DELETE_CARD,
    payload: { listID, cardID }
  };
};

export const updateCard = (listID, cardID, title, text, deadline) => {
  return {
    type: ACTION_TYPES.UPDATE_CARD,
    payload: { listID, cardID, title, text, deadline }
  };
};
```

Додаток Ї1. Index.js

```
export * from './cardActions';
export * from './listActions';

export const ACTION_TYPES = {
  ADD_CARD: 'ADD_CARD',
  DELETE_CARD: 'DELETE_CARD',
  UPDATE_CARD: 'UPDATE_CARD',
  DRAG_CARD: 'DRAG_CARD',
  FETCH_CARDS_SUCCESS: 'FETCH_CARDS_SUCCESS',
  FETCH_CARDS_FAILURE: 'FETCH_CARDS_FAILURE',
};
```

Додаток Й1. BoardCard.jsx

```
import {Button, Card, CardActions, CardContent, Grid, Typography} from "@mui/material";
import React from "react";
import moment from "moment";
import {useNavigate} from "react-router-dom";
import {endpoints, fetchWrapper} from "../util/api.js";
import {getAccessToken} from "../util/auth.js";

export default function BoardCard(props) {
  const {board, onDeleteBoard} = props;
  let navigate = useNavigate();
  const accessToken = getAccessToken() || null;

  const handleOpenBoard = () => {
    navigate(/board/${board.id});
  };

  const handleDeleteBoard = async () => {
    try {
      await fetchWrapper(endpoints.board + /${board.id}, {
        method: 'DELETE',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': Bearer ${accessToken},
        }
      });
      onDeleteBoard(board.id);
    } catch (error) {
      console.log("Error deleting board", error);
    }
  };

  return (
    <>
      <Grid>
        <Card sx={{minWidth: 230, display: 'inline-block', mr: '20px', mt: '15px'}}>
          <CardContent>
            <Typography sx={{fontSize: 24, fontWeight: 'bold'}} color="text.secondary"
gutterBottom>
              {board.name}
            </Typography>
            <Typography variant="body2">
              Owner: {board.owner}
            </Typography>
            <Typography variant="body2">
              Created: {moment(board.createdAt).format('MMMM Do YYYY')}
            </Typography>
          </CardContent>
          <CardActions>
            <Button size="large" onClick={handleOpenBoard}>
              Open
            </Button>
            <Button size="large" color="warning" onClick={handleDeleteBoard}>
              Delete
            </Button>
          </CardActions>
        </Card>
      </Grid>
    </>
  );
}
```

Додаток К1. BoardModal.jsx

```
import {useState} from "react";
import {Box, Button, Grid, Modal, TextField} from "@mui/material";
import CancelIcon from "@material-ui/icons/Cancel.js";
import SaveIcon from "@material-ui/icons/Save.js";
import {endpoints, fetchWrapper} from "../util/api.js";
import {getAccessToken} from "../util/auth.js";

const style = {
  position: 'absolute',
  top: '50%',
  left: '50%',
  transform: 'translate(-50%, -50%)',
  width: 400,
  bgcolor: 'background.paper',
  border: '2px solid #000',
  boxShadow: 24,
  p: 4,
};

const BoardModal = ({open, handleClose}) => {
  const [name, setName] = useState('');
  const accessToken = getAccessToken() || null;

  const handleCreate = async () => {
    const response = await fetchWrapper(endpoints.board, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': Bearer ${accessToken},
      },
      body: ({
        name: name
      })
    });
    await response.json();

    handleClose();
  };

  return (
    <Modal
      open={open}
      onClose={handleClose}
      aria-labelledby="modal-modal-title"
      aria-describedby="modal-modal-description"
    >
      <Box sx={style}>
        <TextField
          label="Name"
          variant="outlined"
          fullWidth
          value={name}
          onChange={(e) => setName(e.target.value)}
          margin="normal"
        />
        <Grid item>
          <Button
            onClick={handleClose}
            variant="text"
            startIcon={<CancelIcon/>}>
            Cancel
          </Button>

          <Button
            onClick={handleCreate}
            variant="text"
            color="primary"
            startIcon={<SaveIcon/>}
```

```
            disabled={name === ''}>
            Create
          </Button>
        </Grid>
      </Box>
    </Modal>
  );
};

export default BoardModal;
```

Додаток Л1. TrelloActionButton.jsx

```
import {useState} from "react";
import {Button, Grid} from "@mui/material";
import TrelloModal from "./TrelloModal.jsx";
import AddIcon from '@material-ui/icons/Add';
import {ACTION_TYPES} from "../../actions/index.js";

const TrelloActionButton = (props) => {
  const { listID, cardsLength } = props;
  const [open, setOpen] = useState(false);
  const buttonText = cardsLength > 0 ? 'Add another card' : 'Add a card';

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

  return (
    <Grid item>
      <Button
        color="primary"
        variant="text"
        startIcon={<AddIcon/>}
        onMouseDown={handleOpenModal}>
        {buttonText}
      </Button>

      <TrelloModal

        listID={listID}
        open={open}
        setOpen={setOpen}
        type={ACTION_TYPES.ADD_CARD}
      />
    </Grid>
  );
};

export default TrelloActionButton;
```

Додаток М1. TrelloBoard.jsx

```
import {useEffect, useState} from "react";
import {Button, Grid, Typography} from "@mui/material";
import {makeStyles} from "@mui/styles";
import TrelloList from "./TrelloList.jsx";
import {DragDropContext} from "@hello-pangea/dnd";
import {connect} from "react-redux";
import {sort} from "../../actions/";
import {addCard, fetchCards} from "../../reducers/listReducer.js";
import {useNavigate, useParams} from "react-router-dom";
import ArrowBackIosNewOutlinedIcon from '@mui/icons-material/ArrowBackIosNewOutlined';
import {endpoints, fetchWrapper} from "../util/api.js";
import {getAccessToken} from "../util/auth.js";
import UsersModal from "./UsersModal.jsx";

const useStyles = makeStyles({
  container: {
    display: 'flex',
    flexDirection: 'column',
    padding: '0px 16px'
  },
  row: {
    display: 'flex',
    alignItems: 'flex-start',
    overflowX: 'scroll',
    width: '100%',
    scrollbarWidth: 'none',
    msOverflowStyle: 'none',
    '&::-webkit-scrollbar': {
      width: 0,
      height: 0
    }
  },
  title: {
    textAlign: 'center',
    color: '#000000',
    padding: 16
  }
});

const TrelloBoard = (props) => {
  const {boardId} = useParams();
  const accessToken = getAccessToken() || null;
  const {board, sort, fetchCards} = props;
  const classes = useStyles();
  let navigate = useNavigate();

  const [boardInfo, setBoardInfo] = useState({});
  const [showUsersModal, setShowUsersModal] = useState(false);
  const [usernames, setUsernames] = useState([]);
  const [badUsername, setBadUsername] = useState(false);

  useEffect(() => {
    fetchCards(boardId);
  }, [boardId, fetchCards]);

  const getBoard = async () => {
    const response = await fetchWrapper(endpoints.board + /${boardId}, {
      method: 'GET',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': Bearer ${accessToken},
      }
    });
    if (response.ok) {
      const userData = await response.json();
      setBoardInfo(userData);
    }
  };
```

```
const getBoardUsers = async () => {
  const response = await fetchWrapper(endpoints.boardUsers + /${boardId}, {
    method: 'GET',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': Bearer ${accessToken},
    }
  });
  if (response.ok) {
    const userData = await response.json();
    setUsernames(userData);
  }
}

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

const handleSaveButton = async (boardUsersDto) => {
  const response = await fetchWrapper(endpoints.boardUsers + /${boardId}, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': Bearer ${accessToken}
    },
    body: boardUsersDto
  });
  if (response.ok) {
    return true;
  } else {
    setBadUsername(true);
    setTimeout(() => {
      setBadUsername(false)
    }, 5000)
  }
}

useEffect(() => {
  getBoard();
}, [])

const onDragEnd = (result) => {
  const {destination, source, draggableId} = result;
  if (!destination) {
    return;
  }

  sort(source, destination, draggableId);
};

const handleBackButton = () => {
  navigate(/boards);
}

const handleUsersButton = () => {
  setShowUsersModal(true);
}

const trelloLists = board.lists.map(({id, title, cards}) => (
  <TrelloList
    key={id}
    listID={id}
    title={title}
```

```
      cards={cards}/>
  ));

  return (
    <Grid container className={classes.container}>
      <Grid container direction="column" justifyContent="space-between"
alignItems="start">
        <Grid container direction="row" justifyContent="space-between"
alignItems="start">
          <Button variant="outlined" color="secondary" sx={{mb: "20px", maxWidth:
'350px'}} onClick={handleBackButton}>
            <ArrowBackIosNewOutlinedIcon/>
            Back
          </Button>
          <Button variant="outlined" color="secondary" sx={{mb: "20px", maxWidth:
'350px'}} onClick={handleUsersButton}>
            Users
          </Button>
        </Grid>
        <Typography variant="h6" component="div">
          {boardInfo.name}
        </Typography>
      </Grid>
      <Grid container sx={{marginTop: '20px'}}>
        <DragDropContext onDragEnd={onDragEnd}>
          <Grid item className={classes.row}>
            {trelloLists}
          </Grid>
        </DragDropContext>
      </Grid>
      <UsersModal open={showUsersModal}
                  setOpen={setShowUsersModal}
                  getBoardUsers={getBoardUsers}
                  usernames={usernames}
                  setUsernames={setUsernames}
                  handleSaveButton={handleSaveButton}
                  badUsername={badUsername}
                  isBoardOwner={isBoardOwner}
      />
    </Grid>
  );
};

const mapStateToProps = (state) => ({
  board: state.board,
});

const mapDispatchToProps = (dispatch) => ({
  sort: (source, destination, draggableId) => dispatch(sort(
    source.droppableId,
    destination.droppableId,
    source.index,
    destination.index,
    draggableId
  )),
  fetchCards: (boardId) => dispatch(fetchCards(boardId)),
  addCard: (state, payload, boardId) => dispatch(addCard(state, payload, boardId))
});

export default connect(mapStateToProps, mapDispatchToProps)(TrelloBoard);
```

Додаток Н1. TrelloCard.jsx

```
import {makeStyles} from "@mui/styles";
import {Button, Card, CardActions, CardContent, Grid, Typography} from "@mui/material";
import {Draggable} from "@hello-pangea/dnd";
import DeleteIcon from '@material-ui/icons/Delete';
import EditIcon from '@material-ui/icons/Edit';
import {ACTION_TYPES} from "../../actions";
import {deleteCard} from "../../reducers/listReducer.js";
import TrelloModal from "./TrelloModal.jsx";
import {connect} from "react-redux";
import {useState} from "react";

const useStyles = makeStyles({
  root: {
    marginBottom: 8,
    transition: 'all .4s',
  },
  title: {
    fontWeight: 'bold',
  },
  buttons: {
    display: 'flex',
    justifyContent: 'space-between',
  }
});

const TrelloCard = (props) => {
  const {listID, card, index, deleteCard} = props;
  const classes = useStyles();
  const [open, setOpen] = useState(false);

  const handleOpenModalModal = () => {
    // TODO: Send card title and text
    setOpen(true);
  };

  const handleTextLength = (text) => {
    const CHAR_LIMIT = 200;
    return text.length > CHAR_LIMIT ? ${text.substring(0, CHAR_LIMIT - 3)}... : text;
  };

  const handleDeleteCard = () => {
    deleteCard(listID, card.id);
  };

  return (
    <Grid item>
      <Draggable
        draggableId={String(card.id)}
        index={index}>
        {(provided) => (
          <Grid
            ref={provided.innerRef}
            {...provided.draggableProps}
            {...provided.dragHandleProps}>
            <Card className={classes.root}>
              <CardContent>
                <Typography
                  className={classes.title}
                  variant="h6"
                  component="h4"
                  gutterBottom>
                  {card.title}
                </Typography>

                <Typography
                  variant="body2"
                  component="p">
                  {handleTextLength(card.text)}
```

```
                </Typography>

                {
                  card.deadline !== undefined
                    ?
                    <Typography
                      variant="body2"
                      component="p">
                      {handleTextLength('Deadline:')}
                    </Typography>
                    : ""
                }

                <Typography
                  variant="body2"
                  component="p">
                  {handleTextLength(card.deadline !== undefined ? card.deadline : 'No
deadline')}
                </Typography>
              </CardContent>

              <CardActions className={classes.buttons}>
                <Button
                  color="primary"
                  size="small"
                  startIcon={<EditIcon/>}
                  onMouseDown={handleOpenModalModal}>
                  Edit
                </Button>

                <Button
                  color="secondary"
                  size="small"
                  startIcon={<DeleteIcon/>}
                  onClick={handleDeleteCard}>
                  Delete
                </Button>
              </CardActions>
            </Card>
          </Grid>
        )}
      </Draggable>

      <TrelloModal
        listID={listID}
        cardID={card.id}
        open={open}
        setOpen={setOpen}
        cardTitle={card.title}
        cardText={card.text}
        cardDeadline={card.deadline}
        type={ACTION_TYPES.UPDATE_CARD}/>
    </Grid>
  );
};

const mapDispatchToProps = (dispatch) => ({
  deleteCard: (listID, cardID) => dispatch(deleteCard(listID, cardID))
});

export default connect(null, mapDispatchToProps)(TrelloCard);
```

Додаток О1. TrelloList.jsx

```
import {makeStyles} from "@mui/styles";
import {Grid, Typography} from "@mui/material";
import TrelloCard from "./TrelloCard.jsx";
import TrelloActionButton from "./TrelloActionButton.jsx";
import {Droppable} from "@hello-pangea/dnd";

const useStyles = makeStyles({
  root: {
    position: 'relative',
    backgroundColor: '#ebecf0',
    borderRadius: 3,
    padding: 8,
    flex: '0 0 300px',
  },
  container: {
    overflowY: 'scroll',
    maxHeight: 600,
  },
  title: {
    padding: '4px 8px'
  }
});

const TrelloList = (props) => {
  const {listID, title, cards} = props;
  const classes = useStyles();

  const trelloCards = cards.map((card, index) => (
    <TrelloCard
      key={card.id}
      card={card}
      listID={listID}
      index={index}/>
  ));

  const handleShowActionButton = () => {
    const LIST_NAME = 'TODO';
    return title === LIST_NAME ? <TrelloActionButton listID={listID}
cardsLength={cards.length}/> : null;
  };

  return (
    <Grid
      item
      className={classes.root}>
      <Droppable droppableId={String(listID)}>
        {(provided) => (
          <Grid
            item
            {...provided.droppableProps}
            ref={provided.innerRef}>
            <Typography
              className={classes.title}
              variant="h6"
              component="h3"
              gutterBottom>
              {title}
            </Typography>

            <Grid
              item
              className={classes.container}>
              {trelloCards}
              {provided.placeholder}
            </Grid>
          </Grid>
        )}
      </Droppable>
```

```
      {handleShowActionButton()}
    </Grid>
  );
};

export default TrelloList;
```

Додаток П1. UsersModal.jsx

```
import {makeStyles} from "@mui/styles";
import {
  Alert,
  Button,
  Grid,
  IconButton,
  List,
  ListItem,
  ListItemText,
  Modal,
  TextField,
  Typography
} from "@mui/material";
import SaveIcon from '@material-ui/icons/Save';
import CancelIcon from '@material-ui/icons/Cancel';
import CloseIcon from '@material-ui/icons/Close';
import {connect} from "react-redux";
import {updateCard} from "../../actions";
import {addCard, deleteCard} from "../../reducers/listReducer.js"
import {useEffect, useState} from "react";
import DeleteIcon from "@material-ui/icons/Delete";

const useStyles = makeStyles({
  root: {
    position: 'absolute',
    top: '50%',
    left: '50%',
    transform: 'translate(-50%, -50%)',
    width: 500,
    maxWidth: '90vw',
    backgroundColor: '#fff',
    padding: 16,
  },
  textfield: {
    marginBottom: 16,
    width: 220
  },
  grid: {
    display: 'flex',
    paddingTop: 16
  },
  closeButton: {
    position: 'absolute',
    right: 8,
    top: 8,
  },
});

const UsersModal = (props) => {
  const classes = useStyles();
  const {open, setOpen, getBoardUsers, usernames, handleSaveButton, badUsername,
isBoardOwner} = props;

  const [usernameList, setUsernameList] = useState([]);
  const [username, setUsernameLocal] = useState('');
  const [owner, setOwner] = useState(false);

  useEffect(() => {
    getBoardUsers();
    setUsernameList(usernames);
  }, [open]);

  useEffect(() => {
    isBoardOwner().then((response) => {
      setOwner(response)
    });
  }, []);
```

```
  const handleCloseModal = () => {
    setOpen(false);
  };

  const handleDeleteIcon = (index) => {
    const updatedList = [...usernameList];
    updatedList.splice(index, 1);
    setUsernameList(updatedList);
  }

  const handleSaveButtonLocal = () => {
    const response = handleSaveButton(usernameList);
    if (response) {
      getBoardUsers();
      setUsernameList(usernames);
    }
  }

  return (
    <Modal
      open={open}
      onClose={handleCloseModal}
      aria-labelledby="simple-modal-title"
      aria-describedby="simple-modal-description"
    >
      <Grid
        container
        direction="column"
        className={classes.root}>
        <IconButton
          className={classes.closeButton}
          onClick={handleCloseModal}>
          <CloseIcon/>
        </IconButton>

        <Grid item xs={12} md={6}>
          <Typography sx={{mb: 2}} variant="h6" component="div">
            Users
          </Typography>
          <List dense={true}>
            {
              usernameList.length === 0 ?
                <Typography variant="body1" component="div">
                  No users
                </Typography>
                :
                usernameList.map((username, index) => (
                  <ListItem
                    key={index}
                    secondaryAction={
                      <IconButton edge="end" aria-label="delete" onClick={() =>
handleDeleteIcon(index)} disabled={!owner}>
                        <DeleteIcon />
                      </IconButton>
                    }
                  >
                    <ListItemText
                      primary={username.username ? username.username : username}
                    />
                  </ListItem>
                ))
            }
          </List>
        </Grid>

        <Grid
          container
          justify="space-between"
          alignItems="center">
          <TextField
            placeholder="Enter username"
```

```
            onChange={(event) => setUsernameLocal(event.target.value)}
            // error={title === ''}
            label="Username"
            sx={{mt: 4}}
          />
          <Button
            variant="text"
            color='primary'
            onClick={() => setUsernameList([...usernameList, username])}
            disabled={!owner}
          >
            Add user
          </Button>
        </Grid>

        <Grid
          container
          justify="space-between"
          alignItems="center"
          className={classes.grid}>

          <Grid item>
            <Button
              onClick={handleCloseModal}
              variant="text"
              startIcon={<CancelIcon/>}>
              Cancel
            </Button>

            <Button
              onClick={handleSaveButtonLocal}
              variant="text"
              color="primary"
              startIcon={<SaveIcon/>}>
              Save
            </Button>
          </Grid>
        </Grid>
        {badUsername ?
          <Alert sx={{
            mt: 4,
            width: '40%',
          }}
                 variant="filled" severity="error">
            Wrong username
          </Alert>
          :
          ""
        }
      </Grid>
    </Modal>
  );
};

const mapDispatchToProps = (dispatch) => ({
  addCard: (listID, title, text, boardId, lists, deadline) => dispatch(addCard(listID,
title, text, boardId, lists, deadline)),
  deleteCard: (listID, cardID) => dispatch(deleteCard(listID, cardID)),
  updateCard: (listID, cardID, title, text, deadline) => dispatch(updateCard(listID,
cardID, title, text, deadline))
});

export default connect(null, mapDispatchToProps)(UsersModal);
```

Додаток Р1. NavBar.jsx

```
import {
  AppBar,
  Avatar,
  Box,
  Button,
  Container,
  IconButton,
  Menu,
  MenuItem,
  Toolbar,
  Tooltip,
  Typography
} from "@mui/material";
import MenuIcon from '@mui/icons-material/Menu';
import React from "react";
import {getAccessToken, useAuth} from "../util/auth.js";
import GradingOutlinedIcon from '@mui/icons-material/GradingOutlined';

function Navbar() {
  const {token, onLogout} = useAuth();
  const [anchorElNav, setAnchorElNav] = React.useState(null);
  const [anchorElUser, setAnchorElUser] = React.useState(null);

  const accessToken = getAccessToken() || null;

  const handleOpenNavMenu = (event) => {
    setAnchorElNav(event.currentTarget);
  };
  const handleOpenUserMenu = (event) => {
    setAnchorElUser(event.currentTarget);
  };

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

  const handleCloseNavMenu = () => {
    setAnchorElNav(null);
  };

  const handleCloseUserMenu = () => {
    setAnchorElUser(null);
  };

  return (
    <AppBar position="static" sx={{backgroundColor: 'rgba(128, 128, 128, 0.70)'}}>
      <Container maxWidth="xl">
        <Toolbar disableGutters>
          <GradingOutlinedIcon sx={{display: {xs: 'none', md: 'flex'}, mr: 1}}/>
          <Typography
            variant="h6"
            noWrap
            component="a"
            href="/"
            sx={{
              mr: 2,
              display: {xs: 'none', md: 'flex'},
              fontFamily: 'monospace',
              fontWeight: 700,
              letterSpacing: '.3rem',
              color: 'inherit',
              textDecoration: 'none',
            }}
          >
            TRELLOLO
          </Typography>
```

```
          <Box sx={{flexGrow: 1, display: {xs: 'flex', md: 'none'}}}>
            <IconButton
              size="large"
              aria-label="account of current user"
              aria-controls="menu-appbar"
              aria-haspopup="true"
              onClick={handleOpenNavMenu}
              color="inherit"
            >
              <MenuIcon/>
            </IconButton>
            {/*todo ? logged in or no*/}
            <Menu
              id="menu-appbar"
              anchorEl={anchorElNav}
              anchorOrigin={{
                vertical: 'bottom',
                horizontal: 'left',
              }}
              keepMounted
              transformOrigin={{
                vertical: 'top',
                horizontal: 'left',
              }}
              open={Boolean(anchorElNav)}
              onClose={handleCloseNavMenu}
              sx={{
                display: {xs: 'block', md: 'none'},
              }}
            >
                    <MenuItem key={"HOME"} onClick={handleCloseNavMenu}>
                      <Typography textAlign="center">{"HOME"}</Typography>
                    </MenuItem>
                    <MenuItem key={"BOARDS"} onClick={handleCloseNavMenu}>
                      <Typography textAlign="center">{"BOARDS"}</Typography>
                    </MenuItem>
                    <MenuItem key={"LOGIN"} onClick={handleCloseNavMenu}>
                      <Typography textAlign="center">{"LOGIN"}</Typography>
                    </MenuItem>
                    <MenuItem key={"SIGNUP"} onClick={handleCloseNavMenu}>
                      <Typography textAlign="center">{"SIGNUP"}</Typography>
                    </MenuItem>
            </Menu>
          </Box>
          <Typography
            variant="h5"
            noWrap
            component="a"
            href="/"
            sx={{
              mr: 2,
              display: {xs: 'flex', md: 'none'},
              flexGrow: 1,
              fontFamily: 'monospace',
              fontWeight: 700,
              letterSpacing: '.3rem',
              color: 'inherit',
              textDecoration: 'none',
            }}
          >
            TRELLOLO
          </Typography>
          {
            accessToken ?
              <Box sx={{flexGrow: 1, display: {xs: 'none', md: 'flex'}}}>
                <Button
                  key={"HOME"}
                  href={"/"}
                  onClick={handleCloseNavMenu}
                  sx={{my: 2, color: 'white', display: 'block'}}
                >
```

```
                {"HOME"}
              </Button>
              <Button
                key={"BOARDS"}
                href={"/boards"}
                onClick={handleCloseNavMenu}
                sx={{my: 2, color: 'white', display: 'block'}}
              >
                {"BOARDS"}
              </Button>
            </Box>
            :
            ""
        }

        {
          accessToken ?
            ""
            :
            <Box sx={{flexGrow: 1, display: {xs: 'none', md: 'flex'}, justifyContent:
'flex-end'}}>
              <Button
                key={"LOGIN"}
                href={"/login"}
                onClick={handleCloseNavMenu}
                sx={{my: 2, color: 'white', display: 'block'}}
              >
                {"LOGIN"}
              </Button>
              <Button
                key={"SIGNUP"}
                href={"/signup"}
                onClick={handleCloseNavMenu}
                sx={{my: 2, color: 'white', display: 'block'}}
              >
                {"SIGNUP"}
              </Button>
            </Box>
        }

        <Box sx={{flexGrow: 0}}>
          {
            accessToken ?
              <Tooltip title="Open settings">
                <IconButton onClick={handleOpenUserMenu} sx={{p: 0}}>
                  <Avatar alt="Remy Sharp" src="/static/images/avatar/2.jpg"/>
                </IconButton>
              </Tooltip>
              :
              ""
          }
          <Menu
            sx={{mt: '45px'}}
            id="menu-appbar"
            anchorOrigin={{
              vertical: 'top',
              horizontal: 'right',
            }}
            anchorEl={anchorElUser}
            keepMounted
            transformOrigin={{
              vertical: 'top',
              horizontal: 'right',
            }}
            open={Boolean(anchorElUser)}
            onClose={handleCloseUserMenu}
          >
            <MenuItem key={"PROFILE"} onClick={handleCloseUserMenu}>
              <Typography textAlign="center">{"PROFILE"}</Typography>
            </MenuItem>
            <MenuItem key={"ACCOUNT"} onClick={handleCloseUserMenu}>
```

```
                <Typography textAlign="center">{"ACCOUNT"}</Typography>
              </MenuItem>
              <MenuItem key={"DASHBOARD"} onClick={handleCloseUserMenu}>
                <Typography textAlign="center">{"DASHBOARD"}</Typography>
              </MenuItem>
              <MenuItem key={"LOGOUT"} onClick={handleLogout}>
                <Typography textAlign="center">{"LOGOUT"}</Typography>
              </MenuItem>
            </Menu>
          </Box>
        </Toolbar>
      </Container>
    </AppBar>
  )
    ;
}

export default Navbar;
```

Додаток С1. Boards.jsx

```
import {endpoints, fetchWrapper} from "../util/api.js";
import {getAccessToken, getRefreshToken} from "../util/auth.js";
import React, {useEffect, useState} from "react";
import BoardCard from "../boards/BoardCard.jsx";
import {Button, Grid} from "@mui/material";
import BoardModal from "../boards/BoardModal.jsx";

function Boards() {
  const refreshToken = getRefreshToken() || null;
  const accessToken = getAccessToken() || null;

  const [boards, setBoards] = useState([]);

  const [openModal, setOpenModal] = useState(false);

  const handleOpenModal = () => {
    setOpenModal(true);
  };

  const handleCloseModal = () => {
    setOpenModal(false);
  };

  useEffect(() => {
    getOwnBoards();
  }, []);

  useEffect(() => {
    getOwnBoards();
  }, [openModal]);

  const getOwnBoards = async (e) => {
    // e.preventDefault();
    const response = await fetchWrapper(endpoints.board, {
      method: 'GET',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': Bearer ${accessToken},
      }
    });
    if (response.ok) {
      const userData = await response.json();
      setBoards(userData);
    }
  };

  const handleDeleteBoard = (boardId) => {
    setBoards((prevBoards) => prevBoards.filter((board) => board.id !== boardId));
  };


  return (
    <div>
      <h3>Your boards</h3>
      <Button variant="outlined" onClick={handleOpenModal}>
        Create board
      </Button>
      <Grid container spacing={2}> {/* Use Grid container to organize BoardCards in rows
*/}
        {boards.map((board) => (
          <Grid item key={board.id} xs={12} sm={6} md={4} lg={3}> {/* Set Grid item size
for responsiveness */}
            <BoardCard board={board} onDeleteBoard={handleDeleteBoard}/>
          </Grid>
        ))}
      </Grid>
```

```
      <BoardModal open={openModal} handleClose={handleCloseModal}/>
    </div>
  );
}

export default Boards;
```

Додаток Т1. Home.jsx

```
import {makeStyles} from "@mui/styles";
import {Button, Grid, Typography} from "@mui/material";
import React from "react";
import {getAccessToken} from "../util/auth.js";

const useStyles = makeStyles((theme) => ({
  root: {
    flexGrow: 1,
    padding: theme.spacing(4),
  },
  header: {
    marginBottom: theme.spacing(4),
  },
  features: {
    marginBottom: theme.spacing(4),
  },
  cta: {
    marginBottom: theme.spacing(4),
  },
  testimonials: {
    marginBottom: theme.spacing(4),
  },
  contact: {
    marginBottom: theme.spacing(4),
  },
}));

function Home() {
  const classes = useStyles();
  const accessToken = getAccessToken() || null;

  return (
    <div className={classes.root}>
      <Grid container spacing={3}>
        <Grid item xs={12} className={classes.header}>
          <Typography variant="h3" gutterBottom>Welcome to Trellolo</Typography>
          <Typography variant="body1" gutterBottom>Your ultimate tool for efficient task
management!</Typography>
        </Grid>
        <Grid item xs={12} className={classes.features}>
          <Typography variant="h4" gutterBottom>Key Features</Typography>
          <ul>
            <li>Create boards to organize your tasks</li>
            <li>Add, edit, and delete tasks/cards</li>
            <li>Assign tasks to team members</li>
            <li>Set due dates and deadlines</li>
          </ul>
        </Grid>
        {
          accessToken === null
            ?
            <Grid item xs={12} className={classes.cta}>
              <Typography variant="h4" gutterBottom>Get Started Today!</Typography>
              <Button variant="outlined" color="primary" to="/signup">Sign Up
Now</Button>
            </Grid>
            :
            ""
        }
      </Grid>
    </div>
  );
}

export default Home;
```

Додаток У1. Login.jsx

```
import {useState} from 'react';
import {useAuth} from '../util/auth';
import {endpoints, fetchWrapper} from '../util/api';
import {
  Alert,
  Avatar,
  Box,
  Button,
  Container,
  createTheme,
  CssBaseline,
  Grid,
  Link,
  TextField,
  ThemeProvider,
  Typography
} from "@mui/material";

const defaultTheme = createTheme();

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

  const {onLogin} = useAuth();

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

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

  return (
    <ThemeProvider theme={defaultTheme}>
      <Container component="main" maxWidth="xs">
        <CssBaseline/>
        <Box
          sx={{
            marginTop: 8,
            display: 'flex',
            flexDirection: 'column',
            alignItems: 'center',
          }}
        >
          <Avatar sx={{m: 1, bgcolor: 'secondary.main'}}>
          </Avatar>
```

```
          <Typography component="h1" variant="h5">
            Sign in
          </Typography>
          <Box component="form" onSubmit={handleLogin} noValidate sx={{mt: 1}}>
            <TextField
              margin="normal"
              required
              fullWidth
              id="username"
              label="Username"
              name="username"
              autoComplete="username"
              autoFocus
              value={username}
              onChange={(e) => setUsername(e.target.value)}
            />
            <TextField
              margin="normal"
              required
              fullWidth
              name="password"
              label="Password"
              type="password"
              id="password"
              autoComplete="current-password"
              value={password}
              onChange={(e) => setPassword(e.target.value)}
            />
            <Button
              type="submit"
              fullWidth
              variant="contained"
              sx={{mt: 2, mb: 2}}
            >
              Sign In
            </Button>
            <Grid container>
              <Grid item xs>
                <Link href="#" variant="body2">
                  Forgot password?
                </Link>
              </Grid>
              <Grid item>
                <Link href="#" variant="body2">
                  {"Don't have an account? Sign Up"}
                </Link>
              </Grid>
            </Grid>
          </Box>
        </Box>
      </Container>

      <Box style={{
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center',
      }}>
        {loginError ?
          <Alert sx={{
            mt: 4,
            width: '40%',
          }}
                 variant="filled" severity="error" onClick={handleAlertClose}>
            Wrong username or password
          </Alert>
          :
          ""
        }
      </Box>
    </ThemeProvider>
  );
```

```
}

export default Login;
```

Додаток Ф1. Signup.jsx

```
import { useState } from 'react';
import { clearUserData } from '../util/auth';
import { endpoints, fetchWrapper } from '../util/api';
import {
  Alert,
  Avatar,
  Box,
  Button,
  Container, createTheme,
  CssBaseline,
  Grid,
  Link,
  TextField,
  ThemeProvider,
  Typography
} from "@mui/material";
import {useNavigate} from "react-router-dom";

const defaultTheme = createTheme();

function Signup() {
  const [firstName, setFirstName] = useState('');
  const [lastName, setLastName] = useState('');
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [loginError, setLoginError] = useState(false);
  const navigate = useNavigate();

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

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

  return (
    <ThemeProvider theme={defaultTheme}>
      <Container component="main" maxWidth="xs">
        <CssBaseline/>
        <Box
          sx={{
            marginTop: 8,
            display: 'flex',
            flexDirection: 'column',
            alignItems: 'center',
          }}
        >
          <Avatar sx={{m: 1, bgcolor: 'secondary.main'}}>
```

```
          </Avatar>
          <Typography component="h1" variant="h5">
            Sign up
          </Typography>
          <Box component="form" onSubmit={handleLogin} noValidate sx={{mt: 1}}>
            <TextField
              margin="normal"
              required
              fullWidth
              id="firstname"
              label="First name"
              name="firstName"
              autoComplete="firstName"
              autoFocus
              value={firstName}
              onChange={(e) => setFirstName(e.target.value)}
            />
            <TextField
              margin="normal"
              required
              fullWidth
              id="lastname"
              label="Last name"
              name="lastName"
              autoComplete="lastName"
              autoFocus
              value={lastName}
              onChange={(e) => setLastName(e.target.value)}
            />
            <TextField
              margin="normal"
              required
              fullWidth
              id="username"
              label="Username"
              name="username"
              autoComplete="username"
              autoFocus
              value={username}
              onChange={(e) => setUsername(e.target.value)}
            />
            <TextField
              margin="normal"
              required
              fullWidth
              name="password"
              label="Password"
              type="password"
              id="password"
              autoComplete="current-password"
              value={password}
              onChange={(e) => setPassword(e.target.value)}
            />
            <Button
              type="submit"
              fullWidth
              variant="contained"
              sx={{mt: 2, mb: 2}}
            >
              Sign Up
            </Button>
            <Grid container>
              <Grid item xs>
                <Link href="#" variant="body2">
                  Forgot password?
                </Link>
              </Grid>
              <Grid item>
                <Link href="#" variant="body2">
                  {"Don't have an account? Sign Up"}
                </Link>
```

```
              </Grid>
            </Grid>
          </Box>
        </Box>
      </Container>

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

export default Signup;
```

Додаток Х1. AppRoutes.jsx

```
import {BrowserRouter as Router, Route, Routes,} from 'react-router-dom';
import {ContentArea} from '../pages/style';
import Navbar from '../navBar/NavBar';
import Home from '../pages/Home';
import Login from '../pages/Login';
import Boards from '../pages/Boards.jsx';
import ProtectedRoute from './ProtectedRoute';
import AuthProvider from './AuthProvider';
import Signup from "../pages/Signup.jsx";
import TrelloBoard from "../boards/TrelloBoard.jsx";

function AppRoutes() {
  return (
    <Router>
      <AuthProvider>
        <Navbar/>
        <ContentArea>
          <Routes>
            <Route path="/" element={<Home/>}/>
            <Route path="/home" element={<Home/>}/>
            <Route
              path="/BOARDS"
              element={(
                <ProtectedRoute>
                  <Boards/>
                </ProtectedRoute>
              )}
            />
            <Route path="/board/:boardId" element={(
              <ProtectedRoute>
                <TrelloBoard/>
              </ProtectedRoute>
            )}/>

            <Route path="/login" element={<Login/>}/>
            <Route path="/signup" element={<Signup/>}/>
            <Route path="*" element={<p>There&apos;s nothing here: 404!</p>}/>
          </Routes>
        </ContentArea>
      </AuthProvider>
    </Router>
  );
}

export default AppRoutes;
```

Додаток Ц1. api.js

```
export const API_HOST = 'http://localhost:8075';

export const endpoints = {
  login: '/api/v1/auth/sign-in',
  logout: '/api/v1/auth/logout',
  signup: '/api/v1/auth/sign-up',
  token: '/api/v1/auth/refresh',
  user: '/api/v1/user',
  board: '/api/v1/boards',
  boardUsers: '/api/v1/boards/users',
  boardItems: '/api/v1/boardItems',
  boardOwner: '/api/v1/boards/owner',
};

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

Додаток Ч1. auth,js

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

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

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

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

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

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

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

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

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

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