БАКАЛАВРСЬКА РОБОТА

БР.КІ–03.00.00.000 ПЗ

Група КІ-21-1

Відоняк Юрій

2025

Івано-Франківський Національний Технічний Університет Нафти і Газу
Факультет Інформаційних Технологій
Кафедра комп'ютерних систем і мереж

Відоняк Юрій Романович

УДК 004.4'2:005.94:004.738.5

БАКАЛАВРСЬКА РОБОТА

Розробка візуалізатора для організації роботи студента kanban-дошки, засобами ASP.NET та React

Комп'ютерна інженерія
(назва освітньої програми)

123 - Комп'ютерна інженерія
(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня Відоняк Ю. Р.
(підпис, прізвище та ініціали здобувача)

Науковий керівник Пашковський Богдан Васильович, к. т. н., доцент
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту

Завідувач кафедри КСМ

д.т.н., професор Мельничук С. І.
(посада) (підпис) (дата) (прізвище та ініціали)

м. Івано-Франківськ – 2025

Івано-Франківський Національний Технічний Університет Нафти і Газу

Факультет Інформаційних Технологій

Кафедра Комп'ютерних систем і мереж

Освітній рівень Бакалавр

Спеціальність 123 Комп’ютерна інженерія

ЗАТВЕРДЖУЮ

Завідувач кафедри КСМ

Мельничук С. І.

" 05 " *травня* 2025 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Відоняку Юрію Романовичу

(прізвище, ім’я, по батькові)

1. Тема роботи Розробка візуалізатора для організації роботи студента kanban-дошки, засобами ASP.NET та React

керівник роботи Пашковський Богдан Васильович, к. т. н., доцент

затверджені наказом закладу вищої освіти від "05" травня 2025 року № 275/7

2. Термін подання студентом роботи 12 червня 2025 року

3. Вихідні дані до роботи Методичні вказівки, технічна література

4. Зміст пояснювальної записки: 1 Нормативне підґрунтя, огляд аналогів, опис вхідних даних, відображення даних та кінцева постановка задачі; 2 Розробка структури бази даних, розробка UML діаграм; 3 Розробка функціоналу веб-додатку

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання – 29 січня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Термін виконання етапів роботи	Примітка
1	Пошук інформативних аналогів, теоретична підготовка до виконання бакалаврської роботи	06.03.2025 – 15.03.2025	виконав
2	Написання коду	16.03.2025 – 15.04.2025	виконав
3	Розробка діаграм UML	05.04.2025 – 15.04.2025	виконав
4	Оформлення пояснювальної записки	11.03.2025 – 31.05.2025	виконав

Студент _____ Відоняк Ю. Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Пашковський Б. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська робота присвячена розробці веб-додатку з kanban-дошкою для організації роботи студента засобами ASP.NET Core та React.

У першій частині роботи наведено нормативне та наукове підґрунтя дослідження, огляд існуючих рішень, описано вхідні дані, способи їхнього представлення та сформульовано постановку задачі.

У другій частині описано проектування таблиць, розробку структури бази даних та UML-діаграм взаємодії з користувачем.

У третій частині подано настанову користувача та описано реалізацію веб-додатку.

Для виконання завдання використано мову програмування C# (.NET 9) з фреймворком ASP.NET Core для серверної частини, бібліотеку React для клієнтської та реляційну СУБД PostgreSQL, розробка проводилася в середовищах Visual Studio 2022 та Visual Studio Code.

Ключові слова: ВЕБ-ДОДАТОК, KANBAN-ДОШКА, ОРГАНІЗАЦІЯ РОБОТИ СТУДЕНТА, БАЗА ДАНИХ, СЕРВЕР, КЛІЄНТ, КОРИСТУВАЧ.

ANNOTATION

The bachelor's thesis is devoted to the development of a web application with a kanban board for organizing student work using ASP.NET Core and React.

The first part of the work provides the regulatory and scientific basis of the study, a review of existing solutions, describes the input data, methods of their presentation, and formulates the problem statement.

The second part describes the design of tables, the development of the database structure and UML diagrams of user interaction.

The third part provides user instructions and describes the implementation of the web application.

To perform the task, the C# (.NET 9) programming language with the ASP.NET Core framework for the server part, the React library for the client part, and the PostgreSQL relational DBMS were used; the development was carried out in the Visual Studio 2022 and Visual Studio Code environments.

Keywords: WEB APPLICATION, KANBAN BOARD, ORGANIZATION OF STUDENT WORK, DATABASE, SERVER, CLIENT, USER.

ЗМІСТ

ВСТУП..... 4

1 ОГЛЯД ВЕБ-ДОДАТКІВ АНАЛОГІВ ТА ПОСТАНОВКА ЗАДАЧІ..... 6

1.1 Нормативне підґрунтя дослідження..... 6

1.2 Виділення інформативних атрибутів на основі огляду веб-додатків аналогів 7

1.3 Вхідні дані для опрацювання..... 11

1.4 Структура відображення даних 13

1.5 Постановка задачі..... 14

2 РОЗРОБКА СТРУКТУРИ БАЗИ ДАНИХ ТА UML ДІАГРАМ..... 17

2.1 Розробка таблиць бази даних..... 17

2.2 Розробка структури бази даних 25

2.3 Розробка UML діаграм взаємодії з користувачем 27

3 РОЗРОБКА ФУНКЦІОНАЛУ ДОДАТКУ 29

3.1 Настанова користувача..... 29

3.2 Розробка функціоналу веб-додатку..... 32

ВИСНОВКИ..... 59

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 60

ДОДАТКИ

БІБЛІОГРАФІЧНА ДОВІДКА

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

Web API – реалізація REST-сервісів через HTTP-ендпоінти на стороні сервера.

API (Application Programming Interface) – програмний інтерфейс для взаємодії різних компонентів або сервісів.

REST – архітектурний стиль взаємодії клієнт-сервер із використанням HTTP-методів та ресурсів.

HTTP (HyperText Transfer Protocol) – протокол передачі гіпертексту, основа комунікації в веб.

HTTPS (Hypertext Transfer Protocol Secure) – захищена версія HTTP.

CRUD – базові операції з даними: Create (створення), Read (читання), Update (оновлення), Delete (видалення).

JWT (JSON Web Token) – стандарт відкритої авторизації, що використовує компактні токени у форматі JSON для безпечної передачі інформації між сторонами.

DTO (Data Transfer Object) – об'єкт для передачі даних між шарами застосунку або між клієнтом і сервером.

EF Core – Entity Framework Core

БД – База Даних.

НФ – Нормальна Форма.

СУБД – Система Управління Базами Даних.

ВСТУП

Актуальність – сучасні освітні процеси та самостійна навчальна діяльність вимагають чіткого планування, гнучкого розподілу завдань та своєчасного контролю прогресу. У зв'язку з поширенням дистанційного та змішаного навчання виникає потреба у цифрових інструментах, які дозволяють студенту не лише відстежувати дедлайни й етапи виконання курсових чи проектних робіт, а й швидко адаптувати розклад під нові вимоги. Одним із ефективних рішень є використання kanban-дошок – візуальних трекерів завдань, що забезпечують прозорість процесу та мотивацію користувача [1]. Для реалізації сучасного веб-застосунку доцільно поєднати потужні серверні можливості платформи ASP.NET із динамічним фронтендом на базі React, а зберігання даних організувати в надійній реляційній СУБД PostgreSQL. Використання REST-принципів для побудови API гарантує масштабованість, гнучкість та можливість інтеграції із зовнішніми сервісами.

Об'єкт дослідження – інформаційний процес планування та обліку навчальних і позааудиторних завдань студента за допомогою веб-додатка.

Предмет дослідження – архітектура клієнт-серверного застосунку з реалізацією kanban-дошки, заснована на технологіях ASP.NET, React та REST API, збереженням даних у PostgreSQL і забезпеченням інтуїтивного інтерфейсу для управління студентськими завданнями.

Мета і завдання роботи – метою бакалаврської роботи є розробка веб-додатку для організації навчального процесу студента у вигляді kanban-дошки. Для досягнення поставленої мети необхідно вирішити такі завдання:

– проаналізувати існуючі підходи до візуалізації та управління навчальними завданнями, а також оглянути інструменти ASP.NET Core і React для реалізації інтерфейсу та серверної частини;

– спроектувати архітектуру системи з урахуванням принципів REST для забезпечення чітких та зрозумілих кінцевих точок API;

– розробити модель даних для зберігання інформації про користувачів, дошки, колонки та картки у СУБД PostgreSQL;

– реалізувати бекенд-сервіс на основі ASP.NET Core, який оброблятиме CRUD-операції над ресурсами та забезпечуватиме автентифікацію й авторизацію студентів;

– створити динамічний фронтенд на React із drag-and-drop інтерфейсом для переміщення карточок між колонками та відображенням статусів завдань у реальному часі;

– провести тестування функціональності та зручності використання системи.

Методи дослідження – у роботі застосовуються аналіз вимог і огляд існуючих сервісів kanban-дошок, моделювання даних із ER-діаграмами, проєктування REST-інтерфейсів, розробка Web API, а також використання бібліотеки React для побудови компонентної структури інтерфейсу та реалізації drag-and-drop механізму.

Практичне значення – розроблений веб-додаток дозволить студентам отримати зручний інструмент для візуального планування і контролю виконання навчальних завдань, що сприятиме підвищенню продуктивності та самоорганізації. Інтеграція з REST-сервісом відкриває можливості підключення до сторонніх освітніх платформ та автоматичного імпорту завдань. Застосування ASP.NET Core і PostgreSQL забезпечить стійкість, безпеку та масштабованість системи, а React-фронтенд – комфортну роботу на різних пристроях без втрати функціональності.

Таким чином, реалізоване рішення може бути впроваджене в освітніх установах або як особистий помічник студента для ефективного управління навчальним процесом.

1 ОГЛЯД ВЕБ-ДОДАТКІВ АНАЛОГІВ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Нормативне підґрунтя дослідження

Розробка інформаційної системи управління навчальними завданнями спирається на загальні терміни та визначення, закладені в ДСТУ ISO/IEC 2382:2017 «Інформаційні технології. Словник термінів». Згідно з цим стандартом, інформаційна система – це сукупність технічних засобів, програмного забезпечення, методик і персоналу для збору, збереження, обробки, пошуку й розповсюдження інформації [2].

Розроблювана система повністю відповідає цьому визначенню: вона включає збирання даних про завдання, їх зберігання в реляційній базі, обробку бізнес-логіки на сервері та представлення інформації через веб-інтерфейс. Крім того, система має ознаки системи підтримки прийняття рішень (СППР), оскільки допомагає студентам аналізувати прогрес, розподіляти ресурси та приймати обґрунтовані рішення щодо виконання навчальних завдань.

Також при проєктуванні було враховано вимоги ДСТУ ISO/IEC 25010:2016 «Системи та програмна інженерія. Моделі якості систем та ПЗ», який визначає характеристики якісного програмного забезпечення [3]. Зокрема:

– функціональна придатність – система виконує всі необхідні функції з організації навчального процесу;

– зручність використання (usability) – інтерфейс адаптований до потреб студентів, забезпечує інтуїтивну взаємодію;

– портативність – веб-орієнтованість дає змогу працювати з різних пристроїв без додаткових налаштувань;

– безпека – реалізація автентифікації користувачів і контролю доступу до персональних даних.

1.2 Виділення інформативних атрибутів на основі огляду веб-додатків аналогів

У процесі розробки інформаційної системи надзвичайно важливим є етап аналізу існуючих аналогів. Це дозволяє не лише сформувати уявлення про функціональні та інтерфейсні особливості сучасних рішень, а й виокремити перелік ключових атрибутів, необхідних для моделювання структури бази даних, формування бізнес-логіки та реалізації ефективного інтерфейсу користувача.

Для того щоб з'ясувати, які атрибути потрібно передбачити, розглянемо кілька прикладів реалізації kanban-дошок іншими розробниками.

Найбільш наближений до поставленої задачі аналог – сервіс Trello [4]. На рисунку 1.1 відображено типовий вигляд дошки в Trello із трьома колонками: «To Do», «Doing» та «Done».

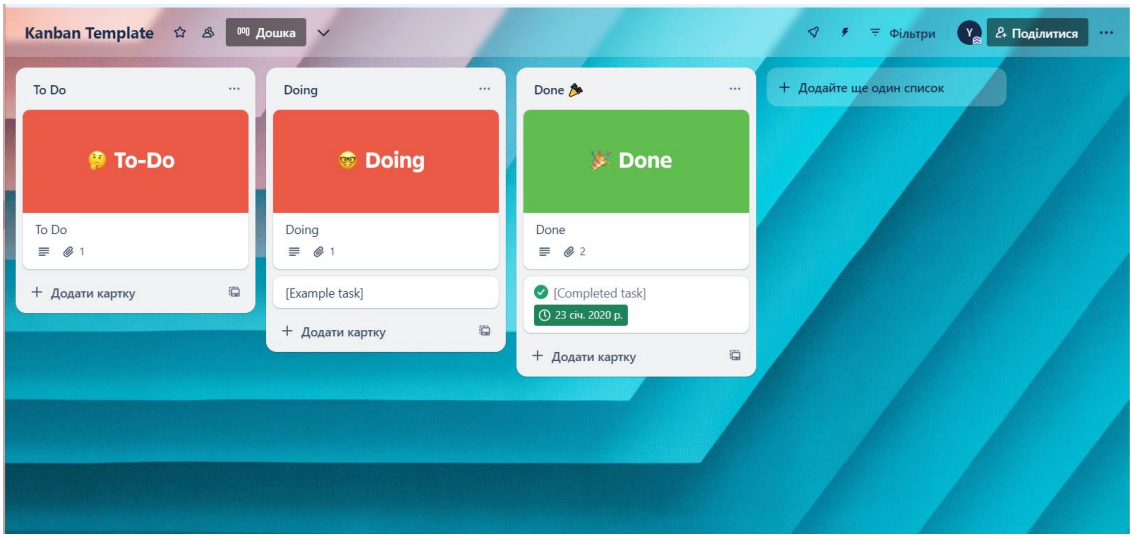


Рисунок 1.1 – Kanban-дошка в Trello

Цей веб-сервіс дозволяє користувачам організовувати особисті або командні завдання шляхом створення віртуальних дошок, де візуально представлено робочий процес у вигляді колонок та карток. З погляду розробника особливо важливо проаналізувати структуру інтерфейсу, механізми взаємодії

користувача із даними, а також зручність доступу до ключової інформації, що безпосередньо впливає на досвід користування системою.

У головному розділі інтерфейсу представлено список доступних дошок. Кожна дошка є логічною одиницею, в межах якої користувач формує власний простір роботи. Дошка складається з колонок (наприклад, «To Do», «Doing», «Done»), які відображають поточний стан завдань. У кожну колонку додаються картки, що відповідають окремим завданням. Самі картки містять розширену інформацію – назву, опис, дедлайни, мітки, чеклісти, коментарі, прикріплені файли. Таким чином, користувачі мають змогу не лише фіксувати завдання, а й управляти деталями виконання на кожному етапі.

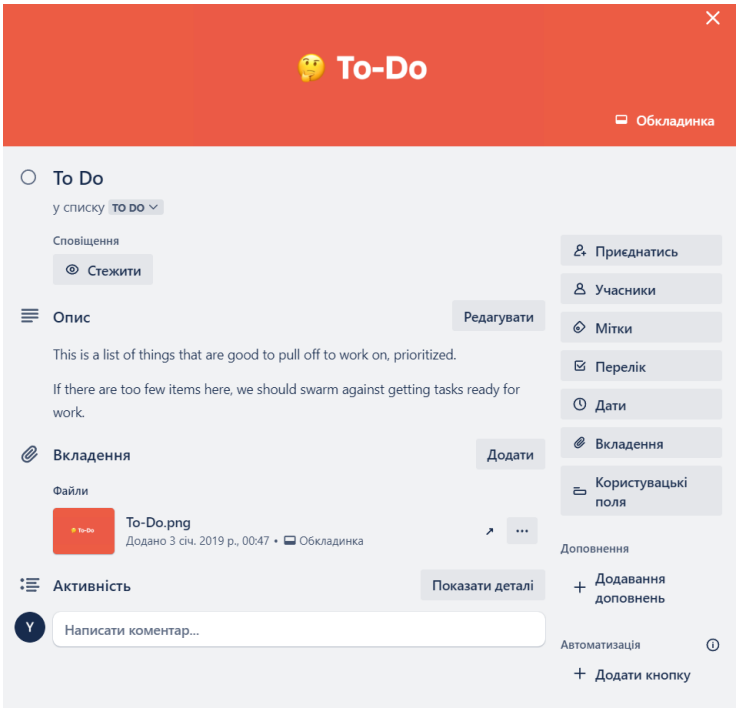


Рисунок 1.2 – Приклад картки у Trello

Додатково, сервіс дозволяє фільтрувати завдання за мітками, датами або виконавцями, що значно полегшує навігацію при великій кількості карток. Реалізація drag-and-drop (перетягування) забезпечує швидку зміну статусу завдання, що робить взаємодію інтуїтивною. Також важливо зазначити, що Trello оптимізовано для мобільних пристроїв – усі елементи адаптуються до розміру

екрану, забезпечуючи однаково зручне користування як на ПК, так і на смартфоні.

Інший приклад – платформа Jira, зображено на рисунку 1.3 [5].

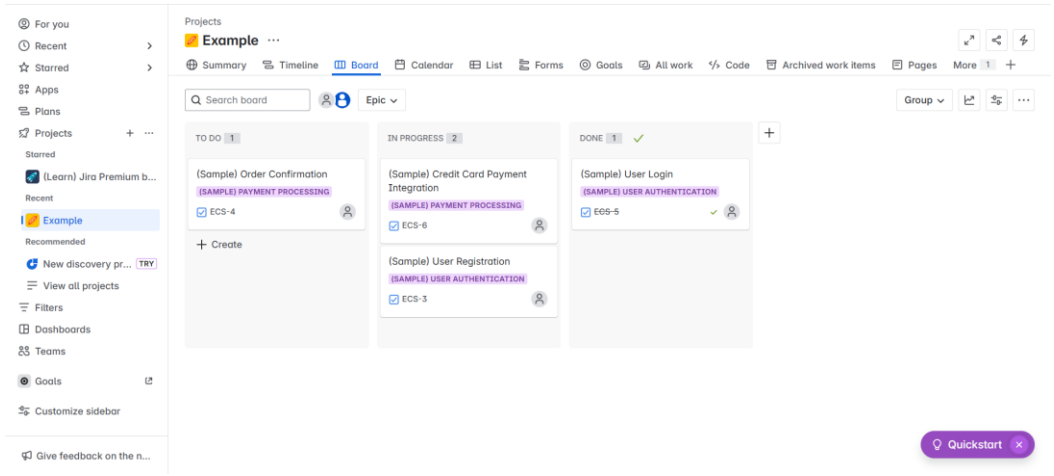


Рисунок 1.3 – Дошка на платформі Jira

Цей сервіс орієнтований переважно на команди розробників, аналітиків та менеджерів, яким необхідно гнучко керувати проєктами з великою кількістю етапів, виконавців та звітів. Інтерфейс платформи побудований навколо проєктів – кожен проєкт має власну панель, набір завдань, схему переходів між статусами та окремі дошки.

Ключовий розділ системи – це kanban або scrum-дошка, де відображаються активні завдання у вигляді карток. Картки містять назву завдання, статус, виконавця, термін виконання, пріоритет, тип завдання (наприклад, помилка або нова функціональність), а також кастомні поля, які можна налаштовувати під специфіку команди або проєкту. Завдяки цьому користувачі мають змогу повністю адаптувати інтерфейс під власні потреби.

Інтерфейс платформи поділений на блоки: меню навігації, фільтри, перегляд задач, інформаційне поле. Така структура забезпечує швидкий доступ до будь-якого розділу системи. Водночас через високу складність конфігурації початкове використання може викликати труднощі, особливо в користувачів без досвіду роботи з подібними платформами.

Останній приклад – сервіс TasksBoard [6], інтерфейс якого зображений на рисунку 1.4.

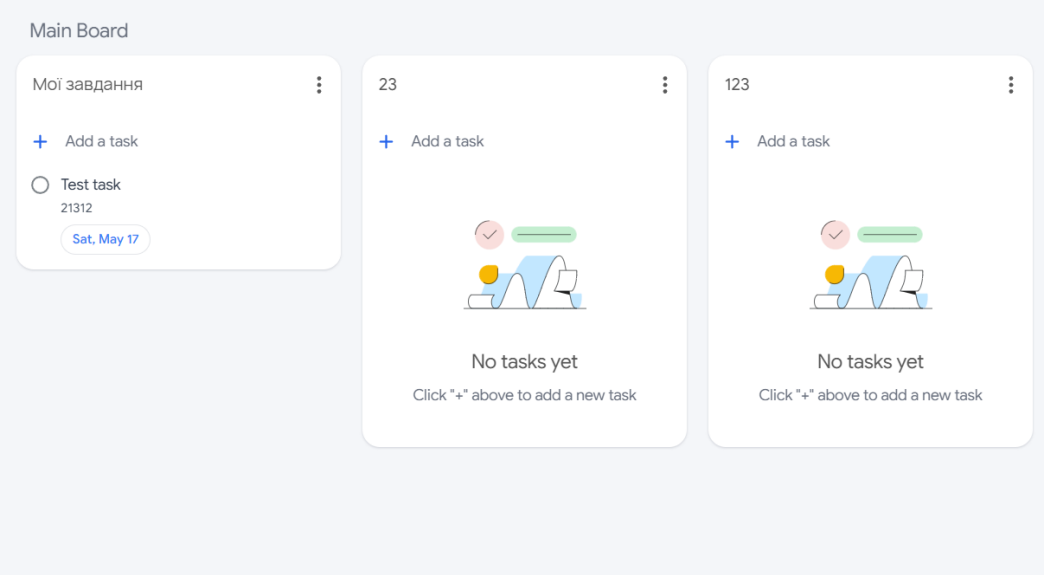


Рисунок 1.4 – Інтерфейс сервісу TasksBoard

Цей сервіс є розширенням до Google Tasks, яке дозволяє відображати списки справ у вигляді kanban-дошки. Такий підхід видається зручним для базового візуального керування завданнями, особливо для тих користувачів, які вже активно використовують екосистему Google. Проте при детальному аналізі можна виявити низку інтерфейсних і функціональних обмежень, які знижують ефективність інструменту, особливо в контексті навчальної або командної роботи.

Інтерфейс складається з вертикальних колонок, які відповідають спискам Google Tasks. У межах кожної колонки знаходяться картки із завданнями, які можна переміщати між колонками шляхом drag-and-drop. Кожна картка містить базову інформацію: назву завдання, дедлайн, а також можливість прикріпити вкладення через Google Drive. Однак варто зазначити, що детальний перегляд самого завдання фактично відсутній: користувач не має можливості відкрити картку у розгорнутому вигляді, щоб переглянути або редагувати розширені

параметри, як це реалізовано в більшості сучасних сервісів управління завданнями.

Крім того, низка корисних функцій – таких як теги, пріоритети або кольорове маркування – доступні лише у платній версії, що знижує доступність ключових інструментів для безкоштовного користувача. Відсутність таких базових елементів у безкоштовній версії значно обмежує персоналізацію та ускладнює візуальне розмежування завдань за важливістю або типом. Навіть ті функції, що є наявними, не підтримують гнучкого налаштування: користувач не може змінювати структуру колонок, додавати підзадачі чи інтегрувати нагадування безпосередньо у TasksBoard – усе це вимагає переходу до самого Google Tasks або сторонніх сервісів.

У результаті такого підходу можуть виникати труднощі у навігації, недостатнє розуміння статусу або змісту завдань, а також загальне враження про незавершеність чи обмеженість функціоналу. Це створює додаткові бар'єри для користувачів, особливо тих, хто прагне чітко структурувати навчальні або особисті задачі. Як наслідок, знижується загальна ефективність використання сервісу, що може призвести до розчарування серед користувачів та пошуку альтернативних, більш гнучких і зручних рішень.

1.3 Вхідні дані для опрацювання

Згідно з оглядом у розділі 1.1, усі розглянуті сервіси мають подібну структуру вхідних даних, необхідних для коректної роботи kanban-дошки. Зокрема, можна виокремити такі загальні групи вхідних даних:

– Kanban-дошка – візуальне представлення завдань у вигляді карток, розміщених у колонках, які відповідають етапам виконання;

– Колонки (списки) – поділ завдань за категоріями або етапами виконання (наприклад, "До виконання", "У процесі", "Завершено");

– Картки (завдання) – окремі одиниці роботи з можливістю додавання опису, дедлайнів, вкладень, коментарів тощо;

– Учасники дошки – автентифіковані особи, які створюють, змінюють або переглядають дошки та завдання.

Нижче наведено декілька текстових прикладів, які були написані на основі ознайомлення з подачею інформації іншими розробниками та послуговуючись одним з основних принципів ООП – абстракцією [7].

Приклад вхідних даних для дошки:

– Назва: Навчальні завдання;

– Опис: Планування та моніторинг виконання курсових і лабораторних робіт, підготовка до лекцій.

Для колонки:

– Назва: В роботі.

Для картки:

– Назва: Написати конспект лекції з БД;

– Опис: Опрацювати лекційні матеріали, виділити ключові поняття та підготувати конспект;

– Мітки: [«Конспект», «База даних»];

– Вкладення: [«lecture_notes.pdf», «erd_diagram.png»];

– Термін виконання: 2025-06-15.

Для учасника дошки:

– Пошта: yuriividoniak@gmail.com;

– Роль: Учасник.

Зважаючи на те, що розроблювана kanban-дошка призначена для індивідуального використання студентом або невеликою навчальною групою, наявність системи користувачів є обов’язковою складовою. Кожен студент повинен мати власний обліковий запис з унікальними обліковими даними, що дозволяють ідентифікувати його в системі та персоналізувати робочий простір.

Система аутентифікації та авторизації забезпечує контроль доступу до особистих дошок, завдань, вкладень та іншої навчальної інформації. Окрім цього, вона закладає основу для подальшого розширення функціоналу –

наприклад, для організації спільної роботи кількох студентів або взаємодії викладач–студент у межах курсу.

Планована реалізація механізму ведення історії активності дозволить у майбутньому відстежувати зміни статусів завдань і підтверджувати виконання ключових етапів роботи, що сприятиме аналізу прогресу та підвищенню ефективності навчального процесу.

Приклад вхідних даних при створені користувача:

– Пошта: yuriividoniak@gmail.com;

– Пароль: 12*34/56QwErTy;

– Ім’я: Yurii;

– Прізвище: Vidoniak;

– Аватарка: avatar.png.

1.4 Структура відображення даних

Враховуючи все описане в попередніх підрозділах, розроблене програмне забезпечення повинно мати таку структуру:

Головна сторінка містить компактний каталог «дошок» (Boards), де студент може побачити назви своїх проєктів та короткі описи або іконки для кожної з них. Такий формат дозволяє швидко зорієнтуватися та перейти до потрібного навчального процесу або курсового завдання. Каталог відображає лише найважливішу інформацію – заголовок дошки, кількість завдань у кожному статусі та дату останньої активності.

Після вибору конкретної дошки користувач потрапляє на її детальну сторінку, де зверху розміщено блок із загальним описом проєкту (мета, дедлайни, рекомендації викладача). Основну площу екрану займає безпосередньо kanban-інтерфейс: набір колонок («Заплановано», «В роботі», «Перевірка», «Завершено»), які чітко розмежовані візуально.

Найбільш значуща частина сторінки – це картки-завдання. Кожна картка відображає заголовок, призначених користувачів та термін виконання, а також індикатори: кольорові мітки, наявність опису, кількість вкладень та коментарів. За потреби студент може швидко переглянути деталі картки у спливаючому вікні, де доступні поля для редагування заголовку чи опису, призначення користувача, зміни терміну, можливість додати чи редагувати мітки, також є панель вкладень та коментарів. Для зручності навігації є можливість фільтрації та сортування карток.

Також інтерфейс передбачає базові елементи керування: кнопка «Додати колонку» розташовані у верхній частині kanban-блоку, а фільтри за мітками, дедлайнами або виконавцями – у лівій панелі. Кнопка «Додати картку» розташована внизу кожної колонки для інтуїтивності. Така організація дає змогу студенту швидко фокусуватися на обраному наборі завдань та мінімізувати кількість кліків під час роботи.

У підсумку, запропонована структура інтерфейсу забезпечує студенту інтуїтивний та адаптивний до різних пристроїв канал роботи з навчальними завданнями. Однорідність елементів, чітка ієрархія та можливість швидкого доступу до деталей картки створюють ефективне середовище організації роботи.

1.5 Постановка задачі

Враховуючи все описане в попередньому підрозділі, розроблене програмне забезпечення повинне мати таку структуру, яка буде забезпечувати максимальну зручність для користувачів та ефективну організацію робочого процесу:

Головна сторінка:

– простий та зрозумілий каталог проєктів (дошок), де користувач бачить лише назви, короткий опис або іконку, що ідентифікує проєкт;

– така структура дозволяє користувачеві швидко знайти потрібну дошку та розпочати роботу.

Сторінка дошки (проєкту):

– додаткова інформація про проєкт: опис, відповідальна особа, дедлайн, статус виконання;

– kanban-дошка із розділенням завдань на типові статуси: «Заплановано», «У процесі», «На перевірці», «Виконано»;

– кожне завдання подано у вигляді картки, що містить назву, короткий опис, дедлайн, пріоритет, відповідального користувача, мітки та інші ключові атрибути;

– користувач може натиснути на картку для перегляду розширеної інформації (детальний опис, вкладення, коментарі тощо);

– реалізована можливість фільтрації завдань за мітками, термінами або виконавцями;

– верхній частині сторінки доступне меню пошуку за назвою.

Каталог користувачів:

– кожен користувач має унікальний обліковий запис, що забезпечує ідентифікацію в системі;

– права доступу розподіляються відповідно до ролі користувача;

– адміністратор має можливість переглядати список усіх користувачів та змінювати їхні права доступу;

– всередині дошки кожен учасник має певну категорію прав (власник, адміністратор, учасник та глядач).

Система створення та редагування завдань:

– забезпечено повноцінне створення нового завдання з усіма необхідними атрибутами: назва, опис, дедлайн, мітки, вкладення;

– завдання можуть бути змінені або переміщені між колонками за допомогою drag-and-drop;

– передбачена можливість додавання коментарів, прикріплення файлів, та оновлення статусу виконання.

Додатково до цього:

– забезпечити інтерфейсну гнучкість: сторінка має коректно відображатися на різних пристроях, включаючи мобільні, без втрати функціональності;

– розробити інтуїтивно зрозумілий механізм створення нових дошок/проєктів із базовими налаштуваннями для швидкого старту користувачів.

2 РОЗРОБКА СТРУКТУРИ БАЗИ ДАНИХ ТА UML ДІАГРАМ

2.1 Розробка таблиць бази даних

Потрібно розробити базу даних забезпечивши третю форму нормалізації (3НФ) [8].

У кожної основній таблиці присутній атрибут id – тобто ідентифікатор запису в таблиці, також він виконує роль первинного ключа. У зв'язувальних таблицях присутні ідентифікатори записів з таблиць, які зв'язуються. Ці ідентифікатори утворюють композитний первинний ключ.

Boards – таблиця дошок (проєктів). Містить інформацію про кожну kanban-дошку. Атрибут Id – первинний ключ. Зовнішні ключі відсутні. Таблиця перебуває в третій нормальній формі (3 НФ), оскільки всі її неключові атрибути функціонально залежать від первинного ключа Id, а між самими неключовими атрибутами немає транзитивних залежностей.

Структуру таблиці Boards подано у таблиці 2.1.

Таблиця 2.1 – Структура таблиці Boards

Ключове поле	Назва стовпця	Тип даних	Довжина	Дозволяє NULL	Вміст
☑	Id	BIGINT	-	-	Містить ідентифікатор дошки
	Title	TEXT	-	-	Назва дошки
	Description	TEXT	-	☑	Опис або мета дошки

Columns – таблиця колонок (статусів) для кожної дошки. Містить інформацію про окремі стани завдань у межах kanban-дошки. Атрибут Id – первинний ключ. Зовнішній ключ BoardId посилається на таблицю Boards(Id) із каскадним видаленням. Таблиця перебуває в третій нормальній формі (3 НФ), оскільки всі її неключові атрибути функціонально залежать лише від первинного ключа Id, а між собою не мають транзитивних залежностей.

Структуру таблиці Columns подано у таблиці 2.2.

Таблиця 2.2 – Структура таблиці Columns

Ключове поле	Назва стовпця	Тип даних	Довжина	Дозволяє NULL	Вміст
☑	Id	BIGINT	-	-	Містить ідентифікатор колонки
	Title	TEXT	-	-	Містить назву колонки
	BoardId	BIGINT	-	-	Містить ідентифікатор дошки, до якої належить колонка
	Position	INTEGER	-	☑	Містить позицію колонки для відображення

Cards – таблиця карток (завдань) у межах колонок. Містить інформацію про окремі завдання на kanban-дошці. Атрибут Id – первинний ключ. Зовнішній ключ ColumnId посилається на таблицю Columns(Id) із каскадним видаленням. Таблиця відповідає третій нормальній формі (3 НФ), оскільки всі неключові атрибути функціонально залежать лише від первинного ключа Id, а також між ними відсутні транзитивні залежності.

Структуру таблиці Cards подано у таблиці 2.3.

Таблиця 2.3 – Структура таблиці Cards

Ключове поле	Назва стовпця	Тип даних	Довжина	Дозволяє NULL	Вміст
☑	Id	BIGINT	-	-	Містить ідентифікатор картки
	Title	TEXT	-	-	Містить назву картки
	DueDate	TIMESTAMP WITH TIMEZONE	-	☑	Містить дату та час дедлайну
	Description	TEXT	-	☑	Містить детальний опис картки
	ColumnId	BIGINT	-	-	Містить ідентифікатор колонки в якій знаходиться картка
	Position	INTEGER	-	-	Містить позицію картки

AspNetUsers – таблиця користувачів. Містить інформацію про облікові записи та профілі користувачів системи. Атрибут Id – первинний ключ. Зовнішніх ключів у таблиці немає. Ця таблиця відповідає третій нормальній формі (3НФ), оскільки всі її неключові атрибути функціонально залежать лише від первинного ключа Id, між неключовими атрибутами немає транзитивних залежностей, а значення в кожному стовпці однотипні.

Структуру таблиці AspNetUsers подано у таблиці 2.4.

Таблиця 2.4 – Структура таблиці AspNetUsers

Ключове поле	Назва стовпця	Тип даних	Довжина	Дозволяє NULL	Вміст
☑	Id	INTEGER	-	-	Містить ідентифікатор користувача
	CreatedAt	TIMESTAMP WITH TIMEZONE	-	-	Містить дату та час створення запису
	UserName	CHARACTER VARYING	256	☑	Містить ім’я користувача у вільному форматі
	Normalized UserName	CHARACTER VARYING	256	☑	Містить нормалізоване ім’я користувача
	Email	CHARACTER VARYING	256	☑	Містить адресу електронної пошти
	Normalized Email	CHARACTER VARYING	256	☑	Містить нормалізовану адресу електронної пошти
	Email Confirmed	BOOLEAN	-	-	Містить прапорець підтвердження електронної пошти
	Password Hash	TEXT	-	☑	Містить хеш пароля
	Security Stamp	TEXT	-	☑	Містить токен безпеки
	Concurrency Stamp	TEXT	-	☑	Містить токен контролю одночасних оновлень
	Phone Number	TEXT	-	☑	Містить номер телефону

Кінець таблиці 2.4

Ключове поле	Назва стовпця	Тип даних	Довжина	Дозволяє NULL	Вміст
	Phone Number Confirmed	BOOLEAN	-	-	Містить прапорець підтвердження номера телефону
	TwoFactor Enabled	BOOLEAN	-	-	Містить прапорець увімкнення двофакторної автентифікації
	Access FailedCount	INTEGER	-	-	Містить кількість невдалих спроб входу
	Lockout Enabled	BOOLEAN	-	-	Містить прапорець дозволу блокування користувача
	FirstName	TEXT	-	-	Містить ім’я користувача
	LastName	TEXT	-	-	Містить прізвище користувача
	Profile PictureUrl	TEXT	-	☑	Містить URL або шлях до зображення профілю

Comments – таблиця коментарів до карток. Містить інформацію про текстові повідомлення, залишені користувачами під завданнями. Атрибут Id – первинний ключ. Зовнішні ключі: UserId → AspNetUsers(Id) та CardId → Cards(Id). Ця таблиця відповідає третій нормальній формі (3НФ), оскільки всі її неключові атрибути функціонально залежать лише від первинного ключа Id, а між собою не мають транзитивних залежностей.

Структуру таблиці Comments подано у таблиці 2.5.

Таблиця 2.5 – Структура таблиці Comments

Ключове поле	Назва стовпця	Тип даних	Довжина	Дозволяє NULL	Вміст
☑	Id	BIGINT	-	-	Містить ідентифікатор коментаря
	Content	TEXT	-	-	Містить текст коментаря
	CreatedAt	TIMESTAMP WITH TIMEZONE	-	-	Містить дату та час створення коментаря

Кінець таблиці 2.5

Ключове поле	Назва стовпця	Тип даних	Довжина	Дозволяє NULL	Вміст
	UpdatedAt	TIMESTAMP WITH TIMEZONE	-	☑	Містить дату та час останнього редагування
	CardId	BIGINT	-	-	Містить ідентифікатор картки
	UserId	INTEGER	-	-	Містить ідентифікатор користувача

BoardMembers – таблиця учасників дошок. Містить інформацію про зв'язок користувачів із конкретними kanban-дошками та їхні ролі. Атрибути BoardId та UserId утворюють складений (композитний) первинний ключ. Зовнішні ключі: BoardId → Boards(Id) та UserId → AspNetUsers(Id) з каскадним видаленням.Таблиця відповідає третій нормальній формі (3 НФ), оскільки всі її неключові атрибути функціонально залежать лише від первинного ключа (комбінації BoardId, UserId), а між ними немає транзитивних залежностей.

Структуру таблиці BoardMembers подано у таблиці 2.6.

Таблиця 2.6 – Структура таблиці BoardMembers

Ключове поле	Назва стовпця	Тип даних	Довжина	Дозволяє NULL	Вміст
☑	BoardId	BIGINT	-	-	Містить ідентифікатор дошки
☑	UserId	INTEGER	-	-	Містить ідентифікатор користувача
	Role	BoardMemberRole	-	-	Містить роль користувача на дошці

BoardMemberRole – перерахований тип даних, який використовується для позначення ролі користувача на конкретній kanban-дошці. Дозволяє забезпечити цілісність і читабельність значень у стовпці Role таблиці BoardMembers, замінюючи рядкові константи на фіксований набір допустимих варіантів.

Визначений чотирма можливими значеннями:

– admin – роль адміністратора дошки з повними правами на керування налаштуваннями та учасниками;

– member – роль активного учасника, який може створювати й редагувати картки;

– observer – роль спостерігача, що має доступ тільки для перегляду без можливості вносити зміни;

– owner – роль власника дошки, яка одночасно охоплює права адміністратора та є її первинним ініціатором.

Attachments – таблиця вкладень для карток. Містить інформацію про файли, прикріплені до завдань на kanban-дошці. Атрибут Id – первинний ключ. Зовнішній ключ CardId посилається на таблицю Cards(Id) із каскадним видаленням. Таблиця відповідає третій нормальній формі (3 НФ), оскільки всі неключові атрибути функціонально залежать лише від первинного ключа Id, а між собою не мають жодних транзитивних або інших залежностей.

Структуру таблиці Attachments подано у таблиці 2.7.

Таблиця 2.7 – Структура таблиці Attachments

Ключове поле	Назва стовпця	Тип даних	Довжина	Дозволяє NULL	Вміст
☑	Id	BIGINT	-	-	Містить ідентифікатор вкладення
	FileName	TEXT	-	-	Містить ім’я файлу
	FileUrl	TEXT	-	-	Містить шлях до файлу
	CardId	BIGINT	-	-	Містить ідентифікатор картки з цим вкладенням

Labels – таблиця міток (тегів) для задач на дошках. Містить інформацію про довільні позначки, які користувач може присвоювати карткам. Атрибут Id – первинний ключ. Зовнішній ключ BoardId посилається на таблицю Boards(Id) із каскадним видаленням. Таблиця перебуває в третій нормальній формі (3 НФ), оскільки всі неключові атрибути функціонально залежать лише від первинного ключа Id, а між собою не мають транзитивних чи інших залежностей.

Структуру таблиці Labels подано у таблиці 2.8.

Таблиця 2.8 – Структура таблиці Labels

Ключове поле	Назва стовпця	Тип даних	Довжина	Дозволяє NULL	Вміст
☑	Id	BIGINT	-	-	Містить ідентифікатор мітки
	Title	TEXT	-	-	Назва мітки
	Color	TEXT	-	-	Колір мітки (HEX-код)
	BoardId	BIGINT	-	-	Містить ідентифікатор дошки, якій належить мітка

ActivityLogs – таблиця журналу активності користувачів. Містить інформацію про дії, виконані на kanban-дошках. Атрибут Id – первинний ключ. Зовнішні ключі: BoardId → Boards(Id) та UserId → AspNetUsers(Id) з каскадним видаленням відповідних записів. Ця таблиця відповідає третій нормальній формі (3 НФ), оскільки всі її неключові атрибути функціонально залежать лише від первинного ключа Id, містять тільки одиночні значення, а взаємозалежностей між ними немає.

Структуру таблиці ActivityLogs подано у таблиці 2.9.

Таблиця 2.9 – Структура таблиці ActivityLogs

Ключове поле	Назва стовпця	Тип даних	Довжина	Дозволяє NULL	Вміст
☑	Id	BIGINT	-	-	Містить ідентифікатор запису
	Action	TEXT	-	-	Містить опис дії
	CreatedAt	TIMESTAMP WITH TIMEZONE	-	-	Містить дату та час фіксації дії
	BoardId	BIGINT	-	-	Містить ідентифікатор дошки, де виконано дію
	UserId	BIGINT	-	-	Містить ідентифікатор користувача, що виконав дію

CardLabels – зв’язувальна таблиця для зв’язку карток із мітками. Містить два поля, які разом утворюють складений первинний ключ: CardId та LabelId, кожне з яких є водночас зовнішнім ключем (вказує на відповідні записи в таблицях Cards та Labels). Оскільки жодних інших атрибутів у таблиці немає, всі її стовпці функціонально залежать лише від первинного ключа (комбінації CardId, LabelId), кожен з них містить тільки одиночне значення, а атрибути не мають жодних взаємозалежностей, таблиця повністю відповідає вимогам третьої нормальної форми (3 НФ).

Структуру таблиці CardLabels подано у таблиці 2.10.

Таблиця 2.10 – Структура таблиці CardLabels

Ключове поле	Назва стовпця	Тип даних	Довжина	Дозволяє NULL	Вміст
☑	CardId	BIGINT	-	-	Містить ідентифікатор картки
☑	LabelId	BIGINT	-	-	Містить ідентифікатор мітки

CardAssignments – проміжна таблиця призначення виконавців карткам. Містить інформацію про те, які користувачі відповідають за виконання конкретних завдань на kanban-дошці. Первинним ключем є поєднання полів CardId і UserId, кожне з яких одночасно є зовнішнім ключем, що посилається на таблиці Cards(Id) та AspNetUsers(Id). У таблиці відсутні додаткові атрибути: обидва поля зберігають лише одиночні атомарні значення і функціонально залежать виключно від складеного ключа, а взаємозалежностей між ними немає. Тому структура CardAssignments відповідає вимогам третьої нормальної форми.

Структуру таблиці CardAssignments подано у таблиці 2.11.

Таблиця 2.11 – Структура таблиці CardAssignments

Ключове поле	Назва стовпця	Тип даних	Довжина	Дозволяє NULL	Вміст
☑	CardId	BIGINT	-	-	Містить ідентифікатор картки
☑	UserId	INTEGER	-	-	Містить ідентифікатор користувача

2.2 Розробка структури бази даних

Для розробки веб-застосунку для організації роботи студента засобами ASP.NET та React необхідно створити діаграму реляційної бази даних у третій формі нормалізації [9].

Отже, у базі даних task_managment_system було створено одинадцять таблиць:

– AspNetUsers;

– Boards;

– Columns;

– Cards;

– Labels;

– Attachments;

– CardLabels;

– CardAssignments;

– Comments;

– BoardMembers;

– ActivityLogs.

Між більшістю таблиць встановлено зв’язок "один-до-багатьох" (наприклад, одна дошка має багато колонок, одна колонка – багато карток тощо), а відношення між картками і мітками реалізовано як "багато-до-багатьох" через зв’язувальну таблицю CardLabels, і аналогічно для призначення виконавців – через CardAssignments. Також у розробці використовуватиметься фреймворк ASP.NET Core Identity для автентифікації та авторизації, який при створенні міграції додає системні таблиці з префіксом AspNet.

Для генерації наочної діаграми сутностей та їх зв’язків використано програму pdAdmin4 [10], яка є графічним клієнтом для управління базою даних. Розроблена діаграма демонструє назви таблиць, перелік полів із зазначенням типів даних і спрямовані стрілки, що позначають зовнішні ключі та типи зв’язків.

Структуру розробленої бази даних наведено на рисунку 2.1.

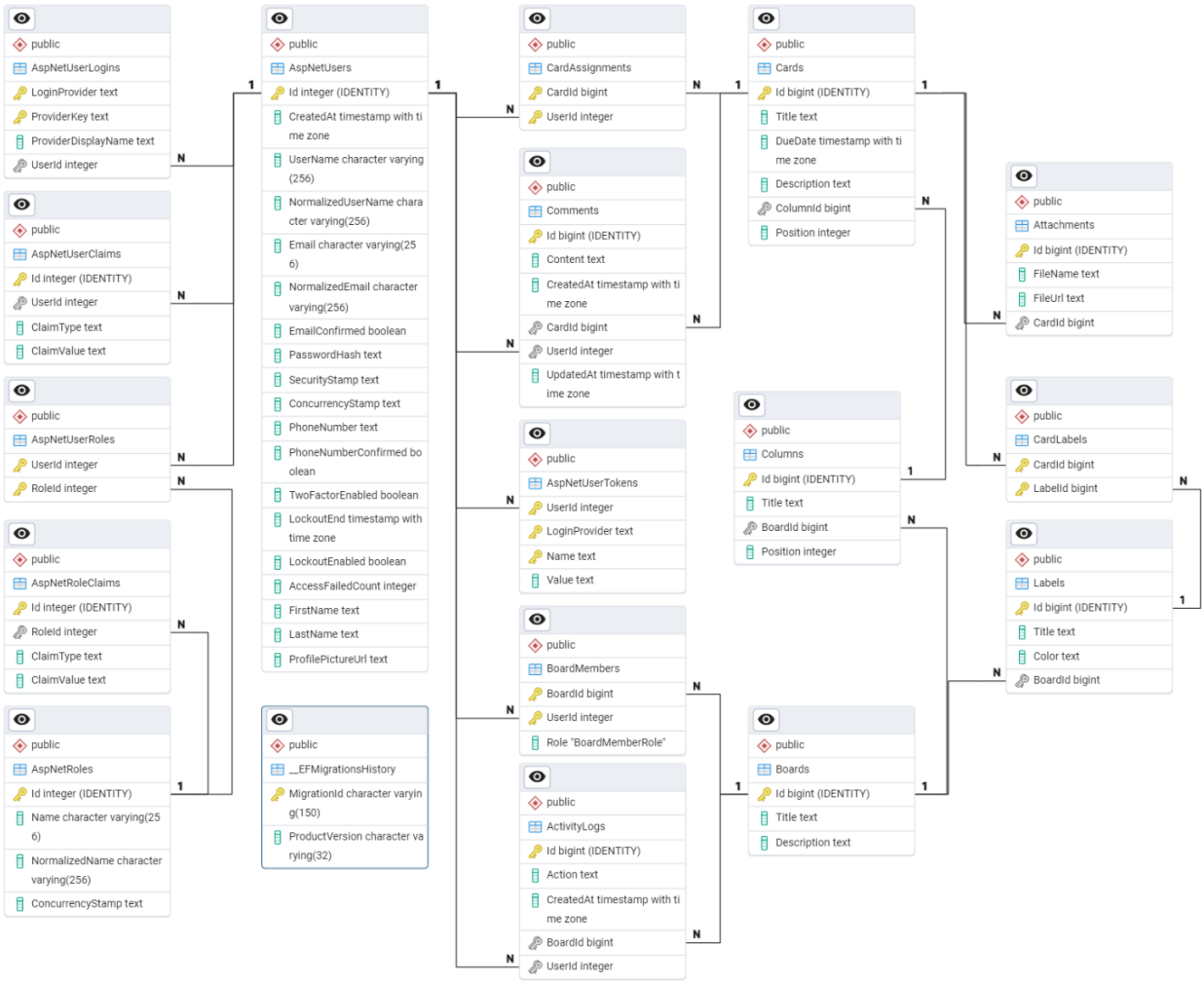


Рисунок 2.1 – Структура бази даних task-management-system

Ця діаграма згенерована за допомогою засобу ERD For Schema [11] застосунку pgadmin.

Діаграма бази даних є критично важливим етапом проєкту, оскільки вона дозволяє всебічно оцінити взаємозв’язки даних і уникнути надлишковості. Завдяки застосуванню третьої форми нормалізації забезпечується відсутність транзитивних та часткових залежностей між атрибутами, що гарантує цілісність і консистентність даних.

2.3 Розробка UML діаграм взаємодії з користувачем

Основними складовими діаграм є учасник (користувач) та прецедент (варіант). Побудова діаграм виконана з використанням онлайн-сервісу Visual Paradigm [12].

На рисунку 2.2 подано діаграму прецедентів.

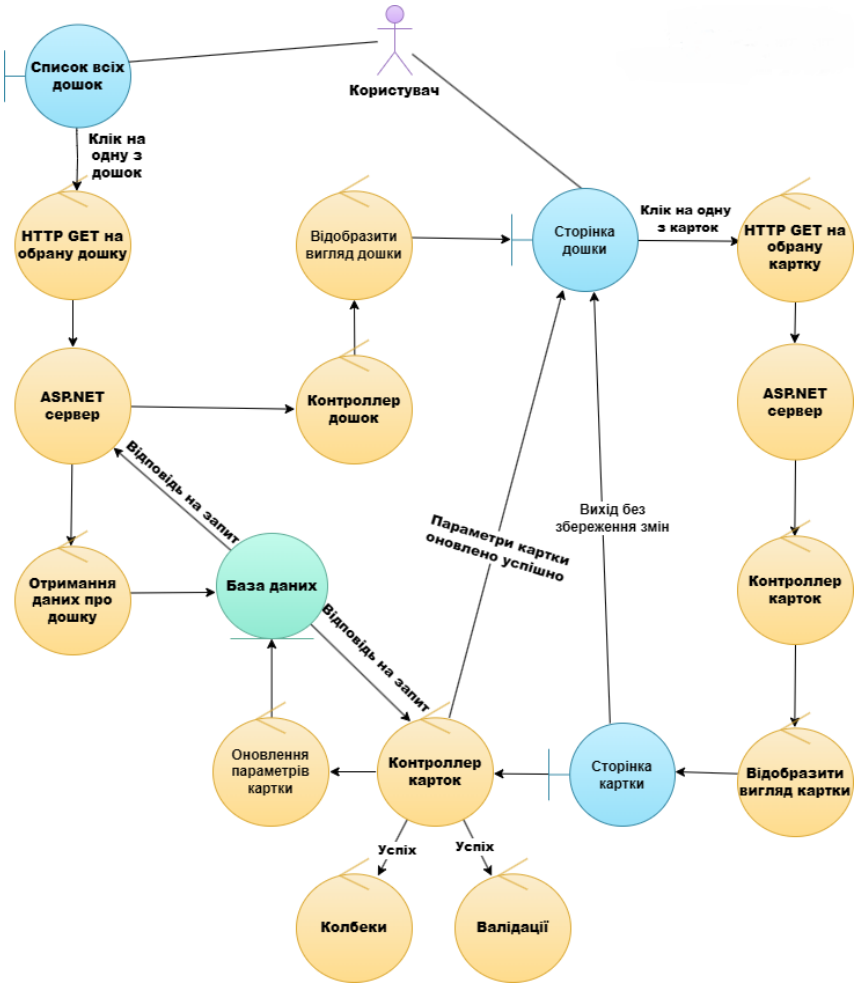


Рисунок 2.2 – Діаграма прецедентів

Учасник зображується у вигляді піктограми людини ("чоловічка") та може представляти як реальну особу, так і зовнішню систему, підсистему або клас. У цьому випадку учасником виступає користувач, що взаємодіє з веб-додатком.

Прецедент (use case) відображає послідовність дій, що виконуються в системі та можуть мати різні варіанти розвитку, результатом яких є певний

підсумок, важливий для учасника. На діаграмі зображено процес перегляду та редагування даних на дошці, що охоплює запити до системи, оновлення бази даних і відображення змін користувачу.

Діаграма послідовностей зображена на рисунку 2.3.

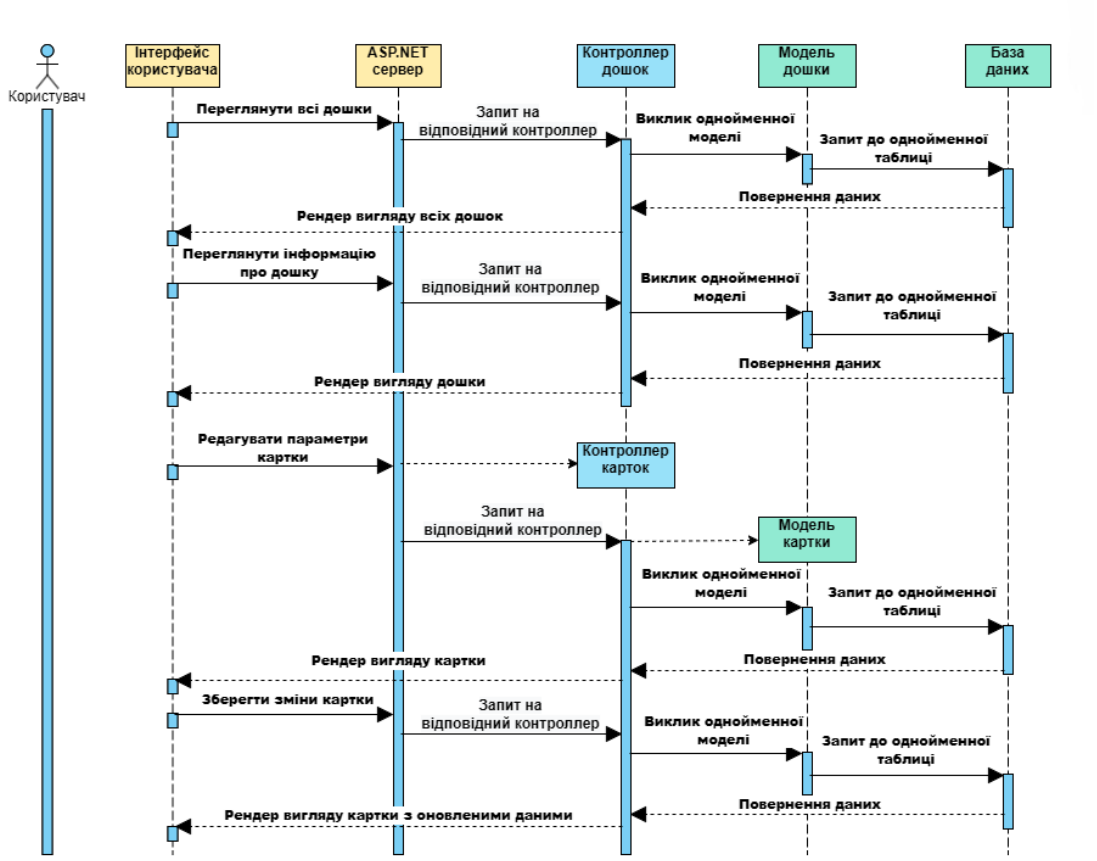


Рисунок 2.3 – Діаграма послідовностей

Згідно діаграми, користувач ініціює дії через інтерфейс — наприклад, перегляд усіх дошок, відкриття конкретної дошки, редагування картки чи збереження змін. Кожна з цих дій призводить до надсилання відповідного запиту до сервера, де обробкою займаються спеціалізовані компоненти, які звертаються до структури даних та здійснюють доступ до бази даних. У відповідь сервер повертає оброблені дані, які клієнтська частина відображає у вигляді оновленого інтерфейсу.

3 РОЗРОБКА ФУНКЦІОНАЛУ ДОДАТКУ

3.1 Настанова користувача

Цей розділ призначений для ознайомлення з основними функціями та можливостями сервісу управління завданнями, а також правилами взаємодії з інтерфейсом.

Сторінка дошок користувача: після входу в систему користувач потрапляє на головну сторінку, де відображається перелік наявних kanban-дошок. Кожна дошка є окремим робочим простором, що відповідає одному проєкту або напрямку роботи. Для кожної дошки відображається її назва, короткий опис, кількість завдань та учасників. Звідси можна перейти до перегляду будь-якої дошки або створити нову (рис. 3.1).

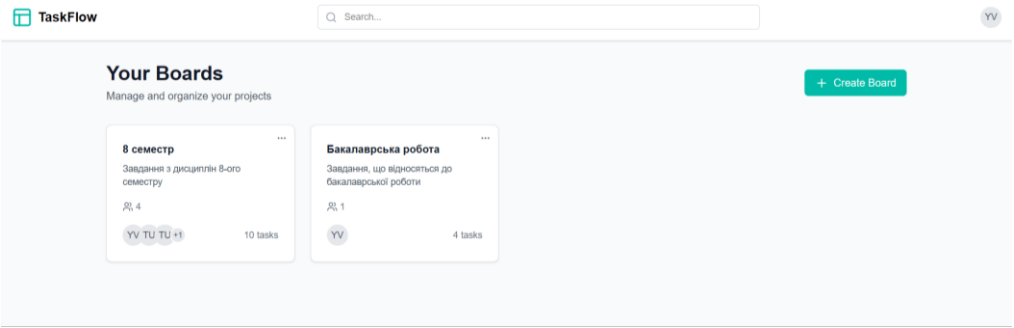


Рисунок 3.1 – Головна сторінка

Сторінка дошки: при відкритті дошки користувач бачить kanban-інтерфейс, поділений на колонки. Кожна колонка містить у собі довільну кількість карток, що відповідають окремим завданням.

Інтерфейс дошки дозволяє:

– додавати нові завдання в будь-яку колонку;

– переміщати завдання між колонками;

– переглядати коротку інформацію про завдання без відкриття картки;

– фільтрувати завдання за мітками;

– шукати завдання за назвою;

– сортувати завдання за назвою або кількістю коментарів.

Користувач також має доступ до функцій управління дошкою: запрошення нових учасників, налаштування заголовку та опису, редагування списку міток тощо.

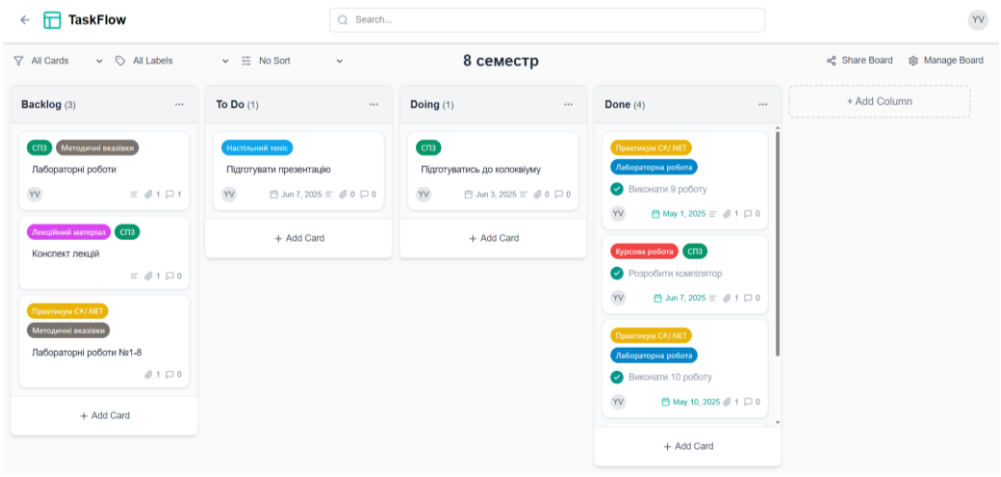


Рисунок 3.2 – Сторінка дошки

Створення завдання: щоб створити нове завдання, користувач натискає кнопку «Додати картку» у потрібній колонці. Після натиску на кнопку у колонці з’явиться картка із назвою «New Card». Кнопка створення завдання наведена на рис. 3.3.

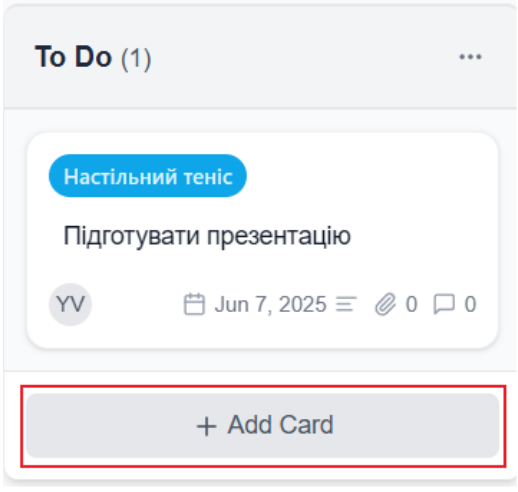


Рисунок 3.3 – Кнопка додавання завдання

Представлення картки: при натисненні на картку відкривається вікно з усіма деталями завдання: повний опис, списки вкладень, коментарі, призначені користувачі і колірні мітки. Тут можна редагувати будь-який атрибут – змінювати статус, термін виконання, призначати виконавців, додавати файли чи коментарі та оновлювати опис:

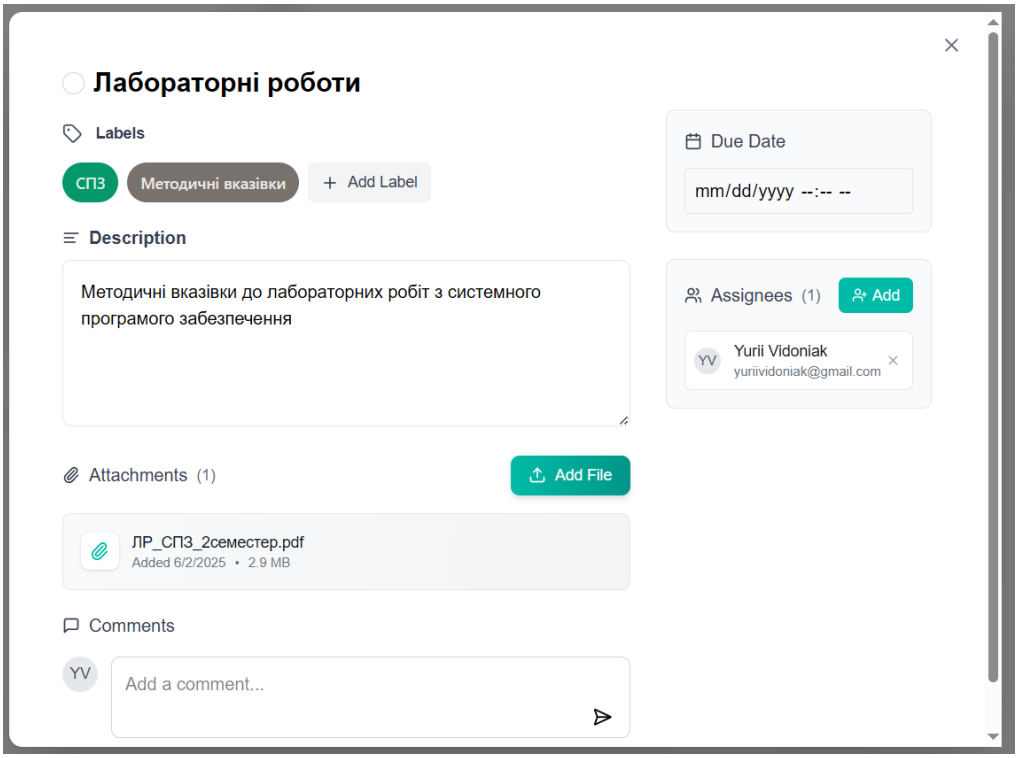


Рисунок 3.4 – Сторінка картки

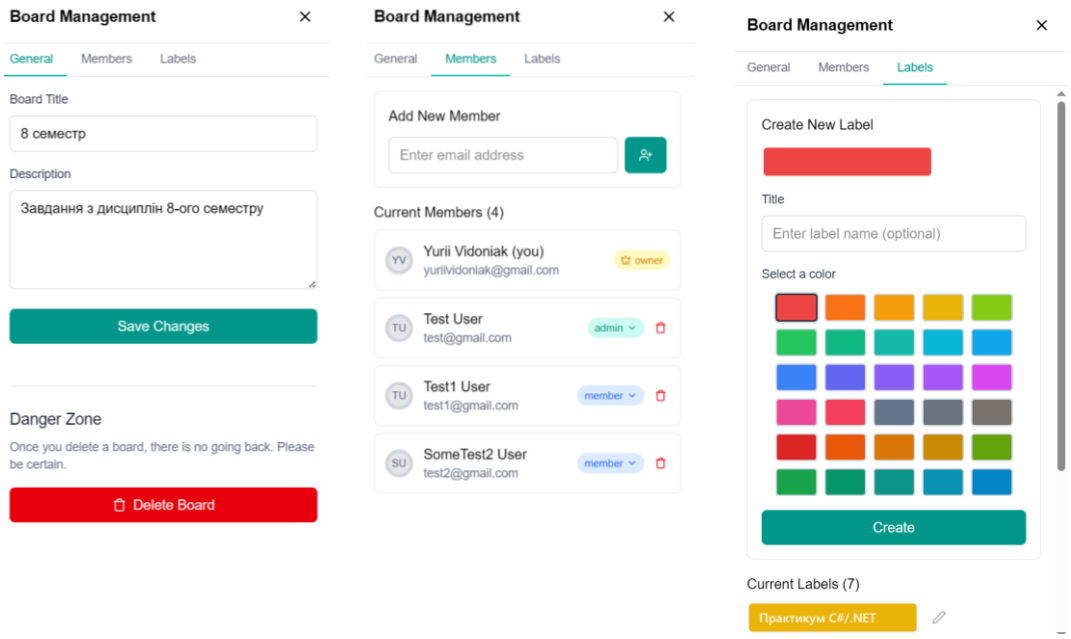
Меню управління дошкою: при натисненні на кнопку «Manage boards» з правої сторони екрану з’являється меню управління дошкою. Тут користувачу доступні такі можливості:

– редагування загальної інформації (назва, опис), а також видалення дошки чи можливість її покинути;

– керування учасниками (додавання, зміна ролі, видалення);

– налаштування доступних міток для позначення завдань;

Якщо користувач є власником дошки, то меню буде виглядати таким чином:



(а) *(б)* *(в)*

Рисунок 3.5 – Вигляд меню управління дошки для власника дошки:
(а) – вкладка загальних параметрів, *(б)* – вкладка учасників, *(в)* – вкладка міток дошки

Доступ до деяких функцій, зокрема оновлення загальної інформації дошки, її видалення, зміна ролей учасників та вилучення їх з дошки, мають лише користувачі з відповідними правами доступу.

3.2 Розробка функціоналу веб-додатку

Розпочато розробку програмного забезпечення, послуговуючись користувацькими інтерфейсами існуючих аналогів. Аналіз аналогічних рішень наведено в першому розділі даної роботи та подано у списку літератури [1–3].

У результаті було обрано архітектуру, яка забезпечує як функціональність, так і масштабованість, використовуючи розділення на клієнтську та серверну частини:

– Серверна частина розташована в директорії API і реалізована на ASP.NET. Вона містить всі основні компоненти, пов’язані з обробкою HTTP-запитів, логікою авторизації, маршрутизацією, доступом до бази даних, сервісами та міграціями.

– Клієнтська частина знаходиться у папці frontend і реалізована на React з використанням TypeScript [13]. Вона відповідає за візуалізацію інтерфейсу, маршрутизацію на клієнті та взаємодію з API.

Структура папки API включає:

– Controllers – реалізація REST-контролерів, які обробляють HTTP-запити;

– DTOs – Data Transfer Object-структури, розбиті за функціональністю;

– Models – базові сутності, що відображають структуру даних;

– Services – логіка обробки даних та взаємодії з базою даних;

– Migrations – скрипти міграцій для оновлення схеми бази даних;

– Middleware – проміжне програмне забезпечення;

– Filters, Exceptions – компоненти для обробки запитів, помилок та фільтрації;

– SeedConfiguration – початкове заповнення бази даних тестовими даними.

Фронтенд проєкту побудовано з урахуванням розділення функціоналу на модулі (features). Основні елементи структури:

– features – модулі, які відповідають за певний функціонал: auth, kanban, кожен з яких містить свої pages, api, components.

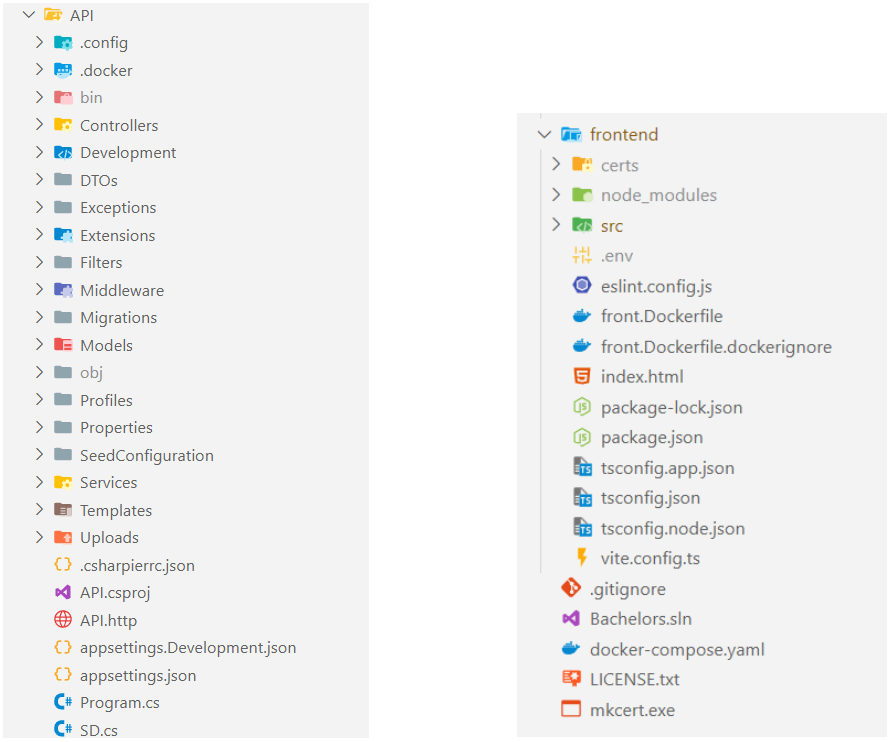
– shared – загальні компоненти, доступні у всіх частинах застосунку: модальні вікна, аватар користувача, маршрути, дата/час дисплеї.

– app – глобальні файли конфігурації, наприклад Redux store [14] або кастомні хелпери.

– utils – утиліти та допоміжні функції.

– certs – сертифікати безпеки (для HTTPS-з'єднань під час розробки).

На рис. 3.6 представлено загальну структуру проєкту, яка відображає логіку розділення відповідальності між клієнтською та серверною частинами.



(а) *(б)*

Рисунок 3.6 – Файлова структура проекту:

(а) – структура папки API, *(б)* – структура папки frontend та вміст корінної папки

Проєкт використовує docker-compose.yaml для запуску фронтенду, API та бази даних у контейнерах [15].

Для налаштування підключення до бази даних використовувався файл appsettings.Development.json, де вказано рядок підключення до PostgreSQL. Застосовано ORM бібліотеку Entity Framework Core [16]. Фрагмент конфігурації:

```
    "ConnectionStrings": {
    "DefaultConnection":
"Host=db;Username=user;Password=password;Database=task_managment_s
ystem;"
    }
```

Розроблено початкову сторінку (рис. 3.7), на якій користувач може ознайомитись із коротким описом веб-додатку, перейти на сторінку входу або реєстрації акаунта. Без акаунта користувач не має можливості скористатись функціоналом сайту.

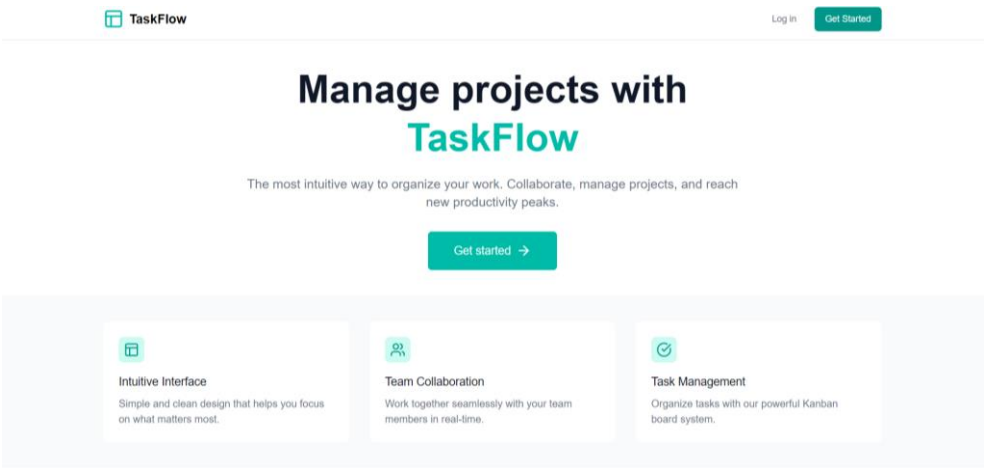


Рисунок 3.7 – Початкова сторінка сайту

Для автентифікації користувачів використано власну реалізацію на основі ASP.NET Core Identity [17] та JWT токенів [18]. Для підключення додано відповідні сервіси до Program.cs та налаштовано конфігурацію Identity:

```
builder
    .Services.AddIdentityCore<AppUser>(o =>
    {o.SignIn.RequireConfirmedAccount = true;
        o.User.RequireUniqueEmail = true;
    })
    .AddRoles<Role>()
    .AddRoleManager<RoleManager<Role>>()
    .AddEntityFrameworkStores<AppDbContext>()
    .AddSignInManager<SignInManager<AppUser>>()
    .AddUserManager<UserManager<AppUser>>()
    .AddDefaultTokenProviders();
builder.Services.AddScoped<JWTService>();
...
```

Для обробки авторизації створено контролер AuthController.cs на серверній частині застосунку. Процес реєстрації реалізовано методом Register:

```
[HttpPost("register")]
 public async Task<IActionResult> Register(RegisterDto model)
    {if (await CheckEmailExistsAsync(model.Email))
{return BadRequest("Account with given email already exists");}
var userToAdd = new AppUser{ FirstName = model.FirstName,
LastName = model.LastName,UserName = model.Email.ToLower(), Email =
model.Email.ToLower(),};
var result = await _userManager.CreateAsync (userToAdd,
model.Password);
try{return await SendConfirmEmailAsync(userToAdd) ? Ok(
new{title = "Account Created", message = "Your account has been
created, please confirm your email address",}) : BadRequest("Failed
to send email. Please contact admin");}
catch (Exception ex) {_logger.LogError(ex.Message); return
BadRequest ("Failed to send email. Please contact admin");}}
```

Форму реєстрації, реалізовану на клієнтській частині, наведено нижче:

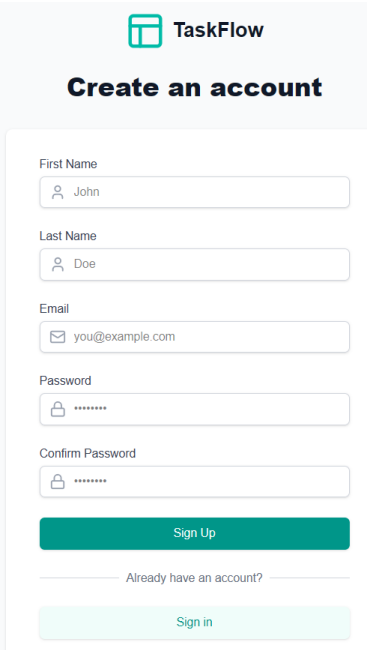


Рисунок 3.8 – Форма реєстрації користувача

Після натиску на кнопку «Sign Up» клієнт перевіряє чи збігаються паролі, введені у двох полях, також чи пошта введена за правильним шаблоном та інші валідації, і відправляє запит на реєстрацію на сервер:

```
const handleSubmit = useCallback(
   async (e: SyntheticEvent) => {e.preventDefault();
const { password, confirmPassword, firstName, lastName, email } = credentials;
if (password !== confirmPassword) {setFormError("Passwords do not match"); return; } try {const res = await register({ firstName, lastName, email, password }).unwrap();
setResponse(res); setIsRegistered(true);
}catch (err) { console.error("Registration failed:", err);}},[credentials, register]);
```

При успішній реєстрації, користувача буде повідомлено про те, що йому надіслано лист про підтвердження реєстрації на пошту:

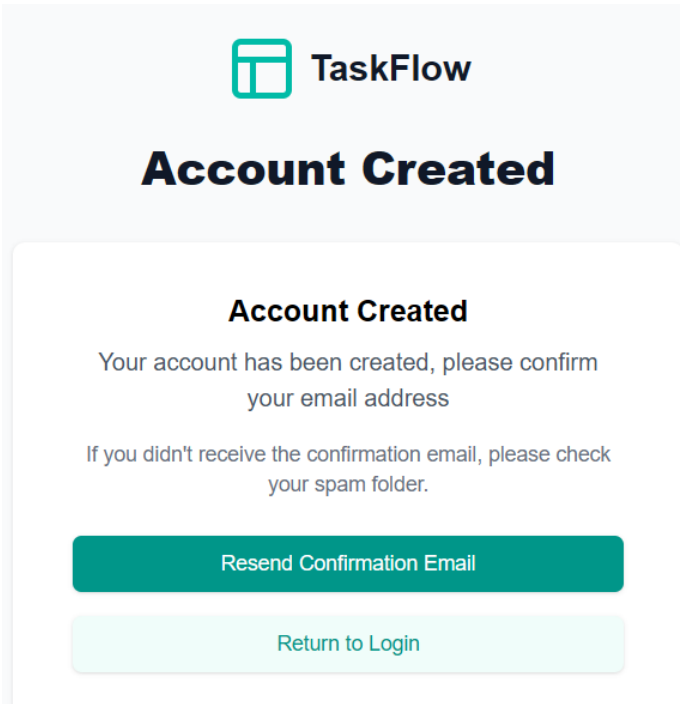


Рисунок 3.9 – Повідомлення користувача

Лист має такий вигляд:

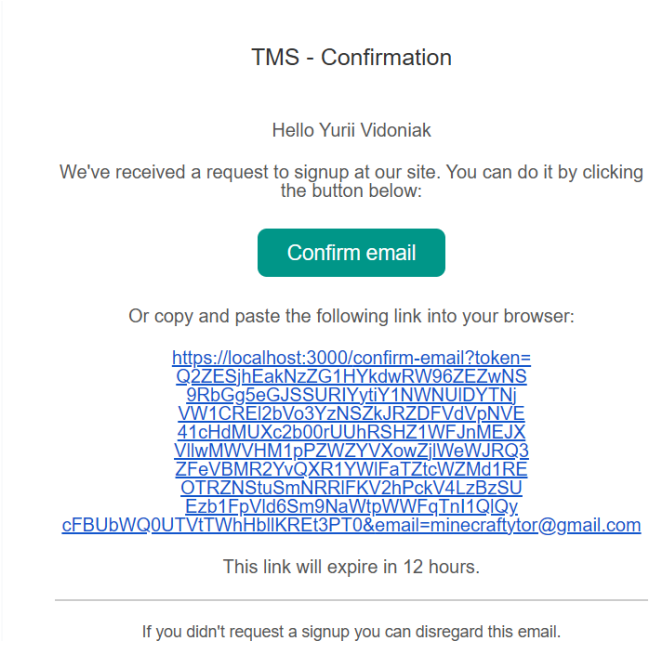


Рисунок 3.10 – Вигляд листа про підтвердження реєстрації

Після переходу за посиланням, вказаним у повідомленні, проводиться процес підтвердження пошти (рис 3.11), після чого користувач може увійти у свій акаунт.

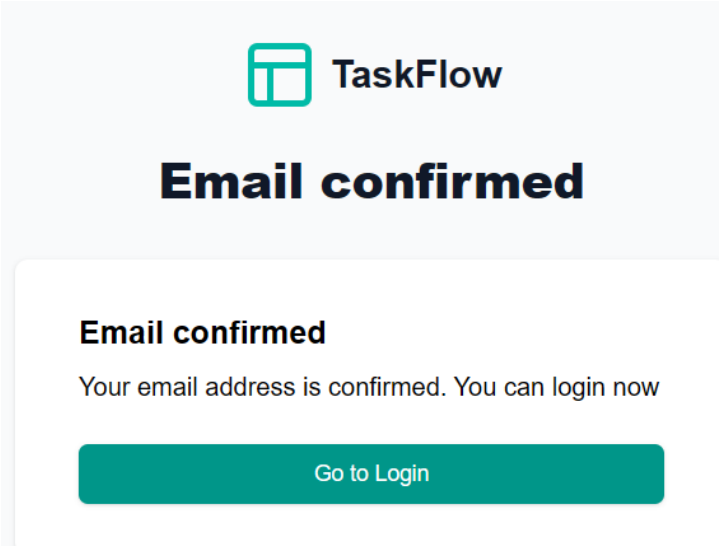


Рисунок 3.11 – Повідомлення про підтвердження реєстрації

Процес входу реалізовано методом Login:

```
[HttpPost("login")]
public async Task<IActionResult> Login(LoginDto model)
{var user = await _userManager.FindByNameAsync (model.UserName);
if (user == null) {return Unauthorized("Invalid username or password");}
if (!user.EmailConfirmed){return Unauthorized("Please confirm your email.");}
var result = await _signInManager.CheckPasswordSignInAsync (user, model.Password, true);
if (result.IsLockedOut) {return Unauthorized($"Your account has been locked due to too many failed attempts. You should wait until {user.LockoutEnd} (UTC time) to be able to login");}
var refresh = await _jwtService.CreateRefreshTokenAsync(user);
return result.Succeeded ? Ok(await CreateAppUserDto(user)) : Unauthorized("Invalid username or password");}
```

Токен, що повертається, містить основні claims (ідентифікатор користувача, роль), підписується симетричним ключем і має обмежений термін дії (наприклад, 15 хв). На фронтенді реалізовано форму входу (рис 3.12).

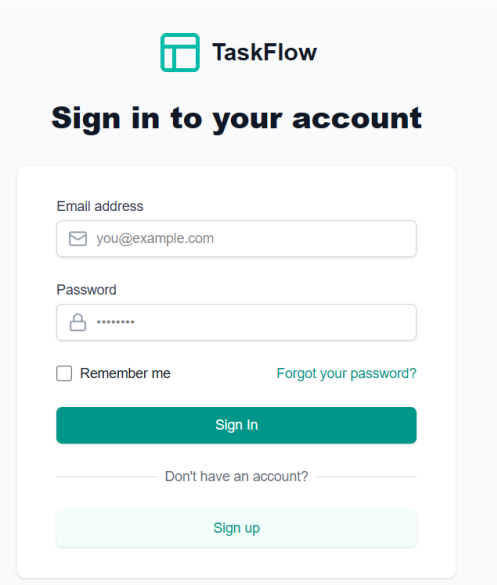


Рисунок 3.12 – Вигляд форми входу в додаток

Ця форма відправляє вхідні дані користувача до API і при успішному вході додає токен (надісланий API) у Redux стан та перенаправляє клієнта на сторінку дошок:

```
const handleSubmit = useCallback(
    async (e: SyntheticEvent) => {
      e.preventDefault();
      try {
        await login(credentials).unwrap();
        navigate("/boards");}
catch (err) {
        console.error("Login failed:", err);}
},
    [login, credentials, navigate]);
```

У подальшому, при кожному запиті на сервер, іде перевірка наявності токена всередині стану і додавання його до заголовку Authorization: Bearer.

Всі контролери серверної частини, що потребують авторизації, позначені атрибутом [Authorize] [19]:

```
[Route("api/boards")]
[Authorize]
[ApiController]
public class BoardsController : Controller
```

У разі спроби доступу без токена або з недійсним токеном повертається HTTP 401 Unauthorized.

Таким чином, реалізована авторизація гарантує, що доступ до ресурсів має лише автентифікований користувач з дійсним токеном. Повні реалізації контролера, сервісу генерації токенів та форми входу і реєстрації наведено у додатку А.

Після успішного входу користувач потрапляє на головну сторінку, де може бачити свої дошки, спершу вона буде пустою (рис. 3.13).

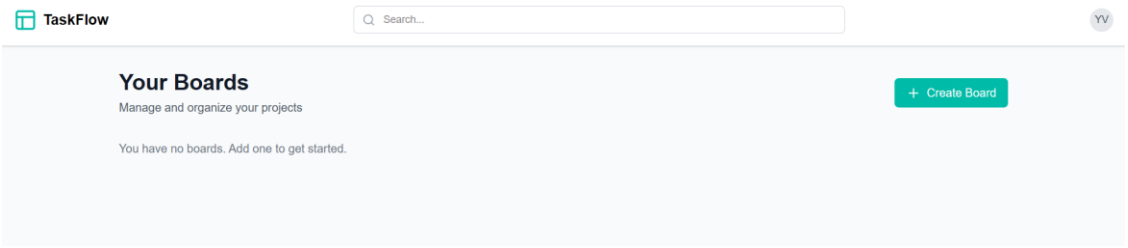


Рисунок 3.13 – Вигляд сторінки дошок

На сторінці видно кнопку з плюсом і надписом «Create board», яка дозволяє користувачу створити дошку. При натиску на неї, на екрані з’являється вікно, у якому можна ввести назву та опис створюваної дошки (рис. 3.14).

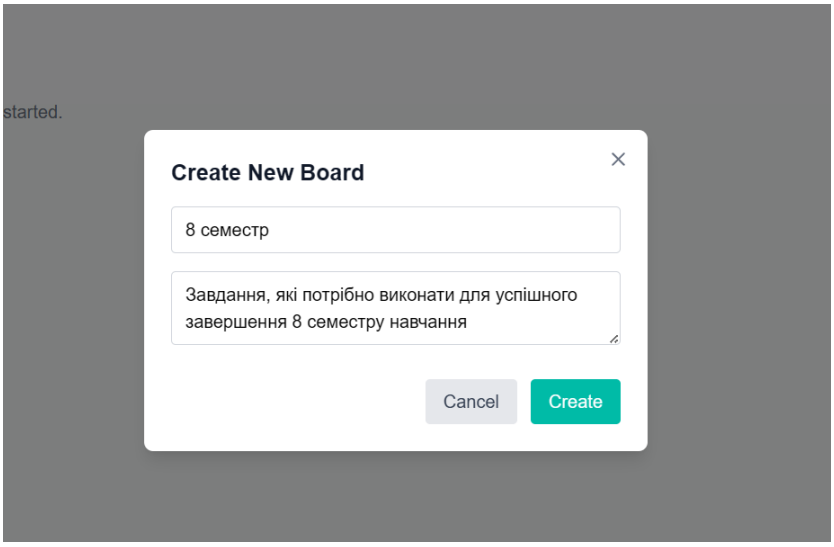


Рисунок 3.14 – Приклад заповнення форми у вікні

Після натиску на кнопку «Create», клієнтська частина відправляє запит на сервер для створення дошки із зазначеними параметрами. Код клієнта:

```
const handleConfirmCreate = async () => {
const title = newBoardTitle.trim(); if (!title) return;
```

```
try {await createBoard({ title, description: newBoardDescription
}).unwrap();
     toast.success("Board created"); setNewBoardTitle("");
     setNewBoardDescription(""); setCreateModalOpen(false);}
     catch (err) {console.error("Failed to create board:", err);}};
     createBoard: build.mutation<BriefBoard, { title: string;
description?: string }>({
           query: (board) => ({url: "/boards", method: "POST", body:
board,}),
```

З цього фрагмента коду видно, що надсилається POST запит на /boards на сервер, що відповідає шляху до контролера дошок. Враховуючи, що контролер реалізований за REST архітектурою, то POST – означає створення нового ресурсу, у цьому випадку – нової дошки. Серверна реалізація методу CreateBoard:

```
     [HttpPost]
         public async Task<ActionResult> CreateBoard(UpsertBoardDto
boardDto)
         { if (!ModelState.IsValid) {return BadRequest(ModelState);}
     var board = new Board { Title = boardDto.Title, Description =
boardDto.Description };
              var currentUserId = User.GetCurrentUserId();
     board.Members.Add(new BoardMember { UserId = currentUserId,
Role = BoardMemberRole.Owner });
     var defaultLabels = GetDefaultLabels();
     foreach (var label in defaultLabels)
      { label.BoardId = board.Id; board.Labels.Add(label);}
     _context.Boards.Add(board); await _context.SaveChangesAsync();
     return CreatedAtAction(nameof(GetBoard), new { id = board.Id },
new { board.Id, board.Title });}
```

Для фільтрації вхідних атрибутів використовується DTO [20] – UpsertBoardDto, що містить у собі поля Title та Description.

Далі, за допомогою системи взаємодії з базою даних (EF Core) створено та надіслано у базу даних відповідний запит:

```
INSERT INTO "public"."Boards" ("Title", "Description") VALUES ('8 семестр', 'Завдання, які потрібно виконати для успішного завершення 8 семестру навчання')
```

При переході на сторінку самої дошки (рис.3.15) помітно, що вона не містить в собі жодної колонки чи картки.

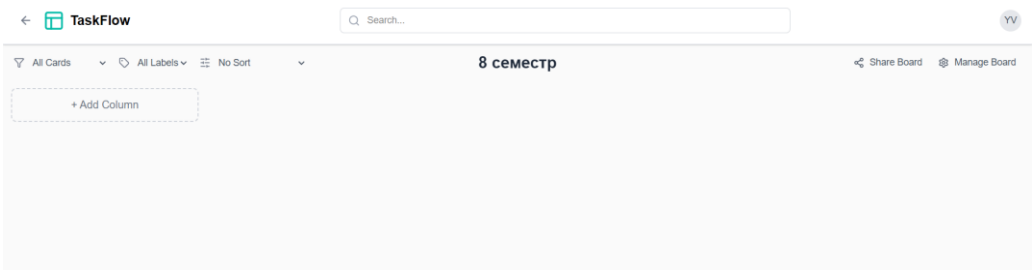


Рисунок 3.15 – Сторінка дошки

При натиску на кнопку «+ Add Column», на сервер поступає відповідний POST запит на створення нового ресурсу, після чого поступає новий GET на ресурс дошки і на сторінці відобразиться колонка із назвою New Column:

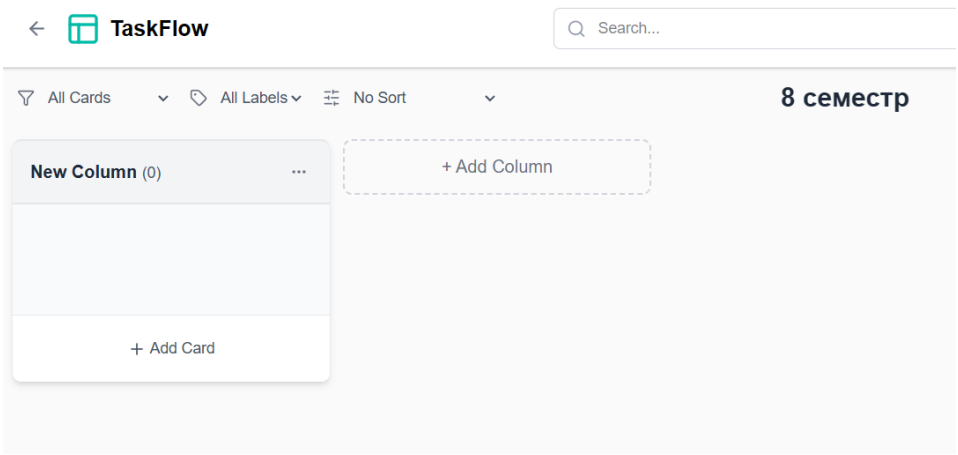


Рисунок 3.16 – Вигляд дошки із створеною колонкою

Якщо натиснуто на три точки, відображається випадаюче меню управління колонкою (рис. 3.17), де можна редагувати її назву, або видалити колонку (разом з усіма картками в ній).

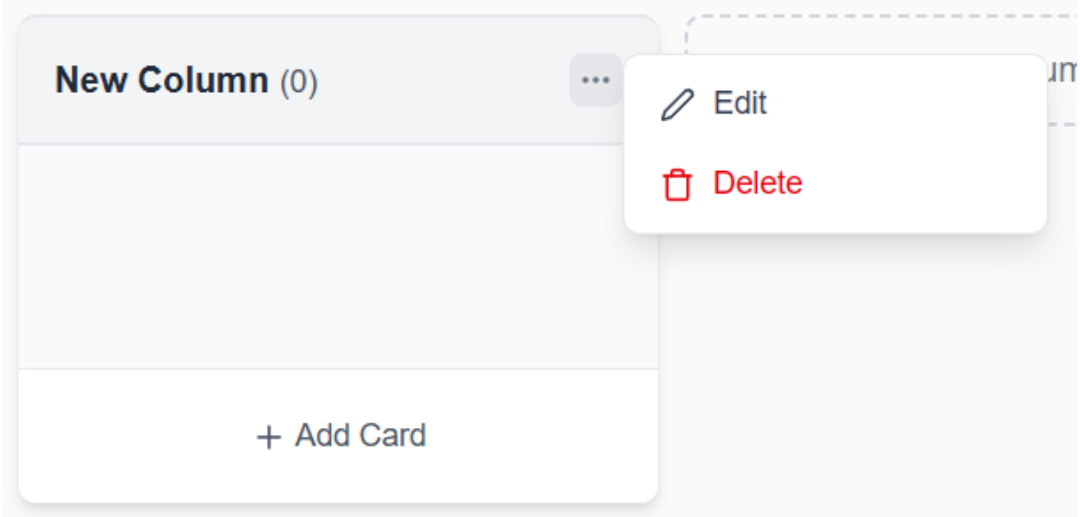


Рисунок 3.17 – Вигляд випадаючого меню

Вибрано кнопку «Edit» та змінено назву колонки на «Backlog»:

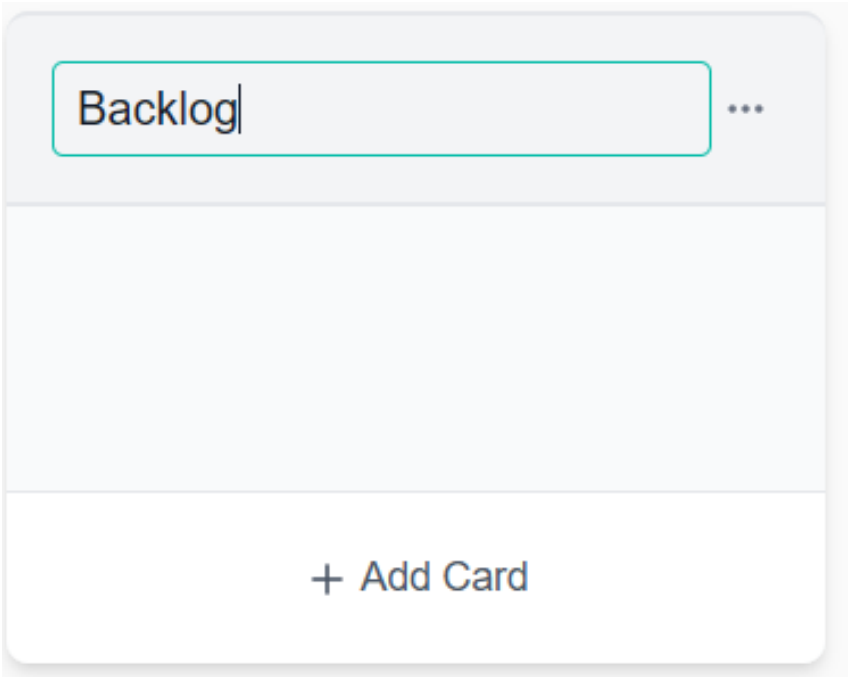


Рисунок 3.18 – Вигляд редагування назви колонки

Далі створено ще 3 колонки та названо їх «To Do», «Doing» та «Done», як в шаблонній kanban-дошці. Після натиснено на кнопку «+ Add Card» в колонці «Backlog»:

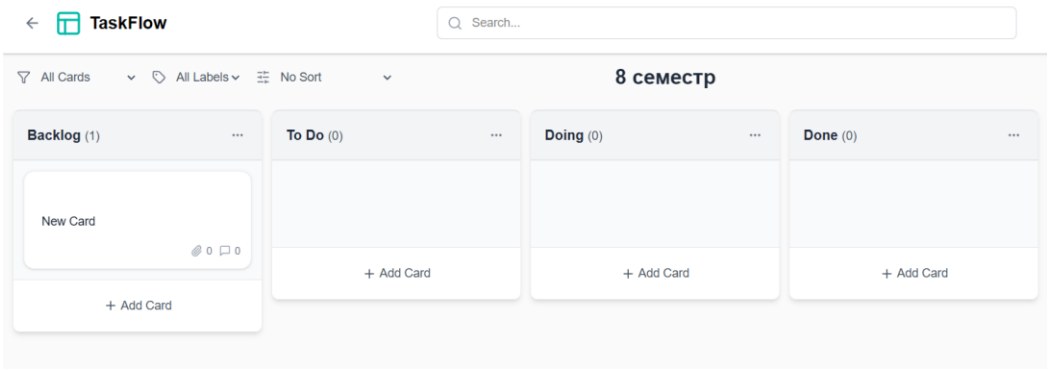


Рисунок 3.19 – Вигляд дошки після додавання колонок і картки

Помітно, що нова картка була створена автоматично з назвою «New Card». Також видно іконки, які відповідають за кількість вкладень та коментарів на картці. При наведенні курсора на картку, на її представленні з’являється три точки, за допомогою яких відкривається випадаюче меню управління карткою, та прапорець, який дозволяє відмітити чи завдання, яке вказане на картці, виконане (рис. 3.20).

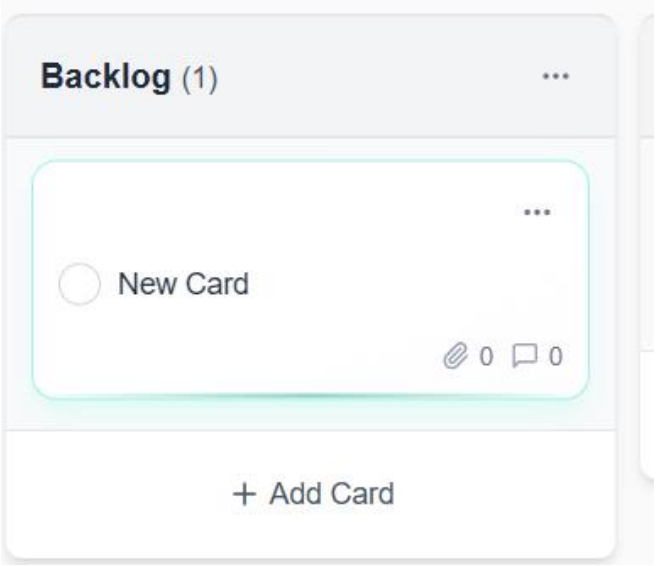


Рисунок 3.20 – Вигляд картки після наведення курсора

При натиску на картку, відкривається вікно з детальною інформацією про картку (рис. 3.21).

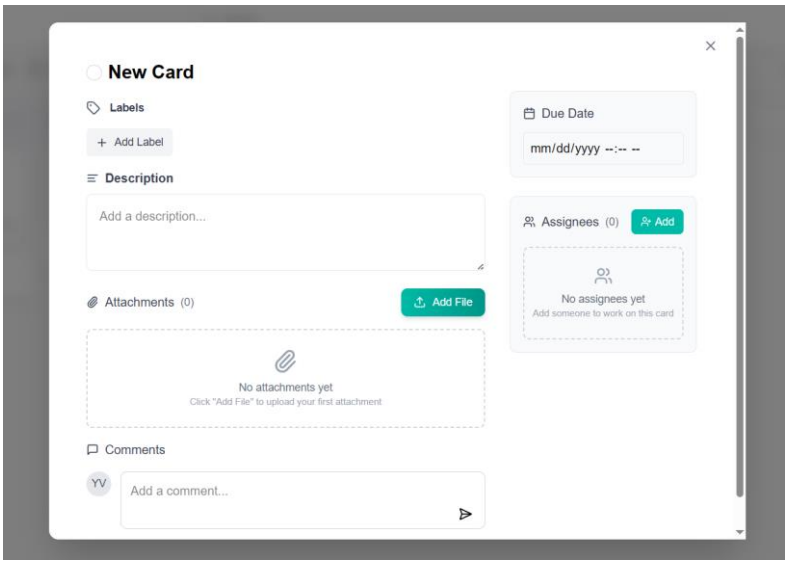


Рисунок 3.21 – Вікно детального вигляду картки

У верхній частині картки розташована назва картки, зліва від неї прапорець виконання завдання. Під ними видно кнопку «+Add Label», при натиску на яку відображається меню доступних міток (рис. 3.22), з можливістю додавати нові та редагувати чи видаляти наявні, а також прикріпляти наявні мітки до картки чи відкріпляти уже прикріплені.

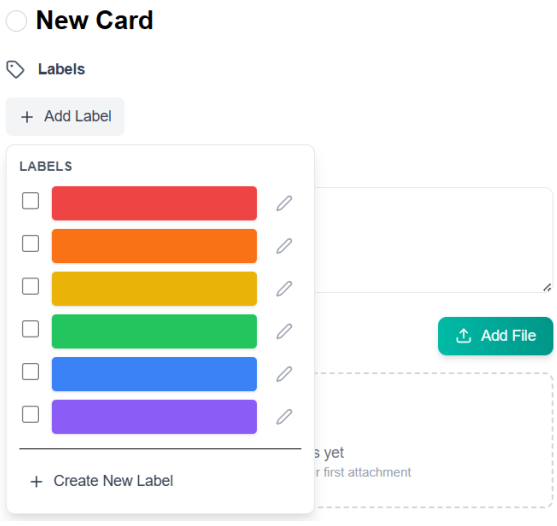


Рисунок 3.22 – Меню доступних міток

Нижче розташований опис картки, під яким знаходиться меню завантаження та роботи з вкладеннями. При натиску на кнопку «Add file» відкривається меню завантаження файлу з пристрою. Під областю вкладень розташовані коментарі, де користувачі мають змогу додавати коментарі, а після редагувати чи видаляти їх. Праворуч від опису знаходиться поле введення терміну виконання, під яким можна додати виконавців з списку учасників дошки. Проведено зміну назви картки на «Лабораторні роботи», додано мітки «Методичні вказівки» та «СПЗ», додано опис та завантажено сам файл з методичними вказівками (рис. 3.23).

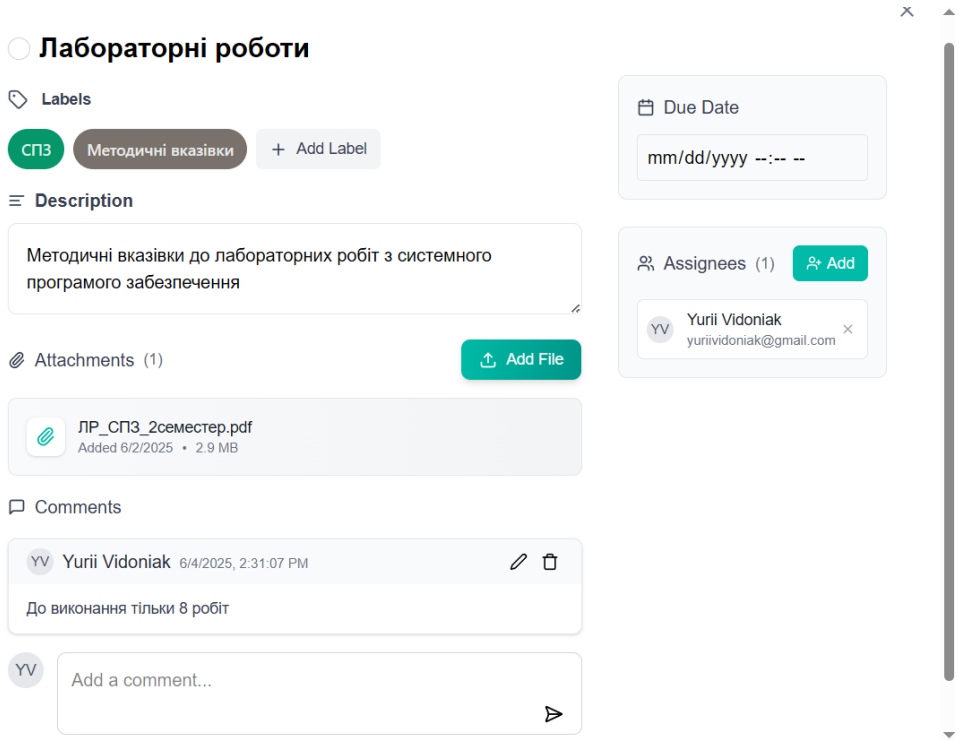


Рисунок 3.23 – Вигляд картки після проведених змін

Поля назви, опису, терміну виконання та прапорець виконання є безпосередніми полями картки і їх зміна відбувається індивідуально, тобто при натиску на конкретне поле, можна зберегти значення в ньому або ж відмінити зміни:

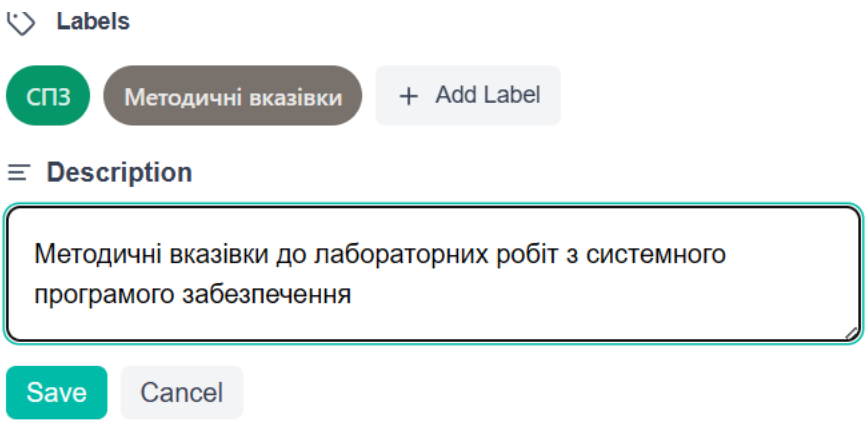


Рисунок 3.24 – Вигляд поля «Description» при редагуванні

Після натискання кнопки «Save» на сервер надсилається PATCH-запит до контролера – CardsController, а саме до методу PatchCard, який використовує JsonPatchDocument для внесення змін лише в окремі атрибути ресурсу:

```
[HttpPatch("{id}")]
    public async Task<IActionResult> PatchCard(long boardId, long id, JsonPatchDocument<UpdateCardDto> patchDoc)
    {if (patchDoc == null){ return BadRequest();}
await _boardValidationService.ValidateBoardAsync(_context, boardId, User.GetCurrentUserId());
var card = await _context.Cards.Include(c => c.Column).Where(c => c.Id == id && c.Column.BoardId == boardId).FirstOrDefaultAsync();
        if (card == null) {return NotFound("Card not found");}
        var dto = _mapper.Map<UpdateCardDto>(card);
        …
        patchDoc.ApplyTo(dto, ModelState);
        if (!TryValidateModel(dto))
        {return BadRequest(ModelState);}
        _mapper.Map(dto, card);
        await _context.SaveChangesAsync();
        return NoContent();}
```

За фільтрацію вхідних даних відповідає UpdateCardDto.

Після застосування JsonPatchDocument, відбувається валідація даних, за успішного завершення якої, оновлений DTO за допомогою AutoMapper перетворюється у сутність Card. Далі EF Core відслідковує зміни у цій сутності і надсилає відповідні Update запити у базу даних.

Змінимо активний акаунт на той, де уже є заповнена kanban-дошка:

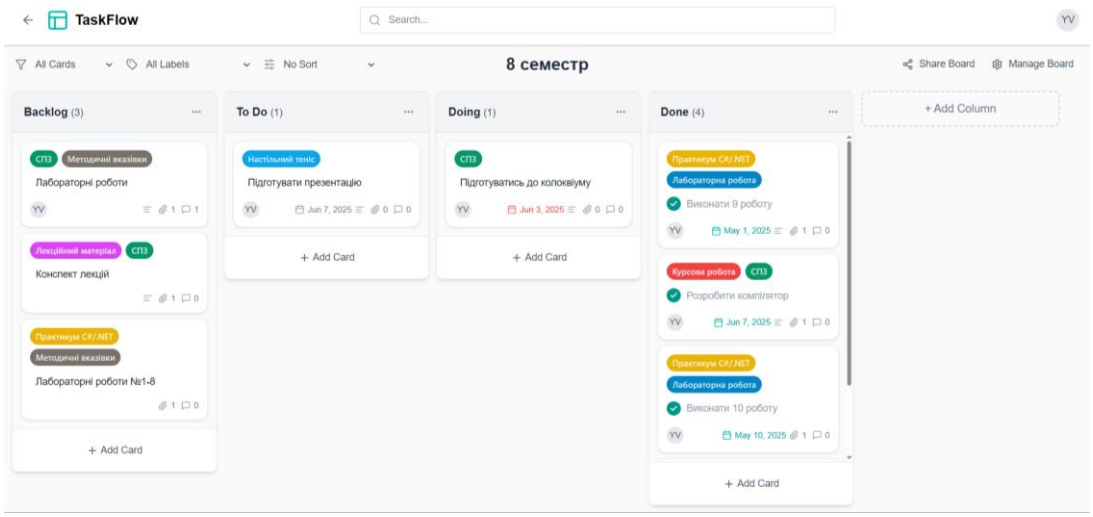


Рисунок 3.25 – Вигляд заповененої kanban-дошки

На прикладі цієї дошки розглянуто функціонал фільтрів та сортування. У верхній частині дошки натиснуто на випадаючий список із назвою «All labels»:

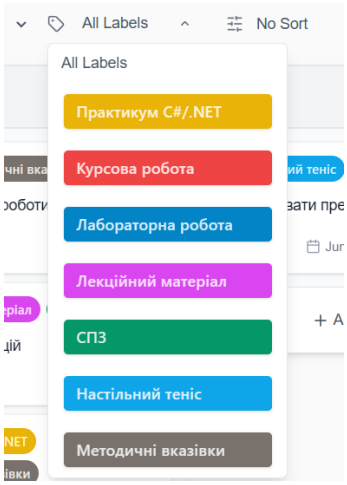


Рисунок 3.26 – Вигляд випадаючого списку

Вигляд дошки після натиску на мітку «Практикум C#/.NET»:

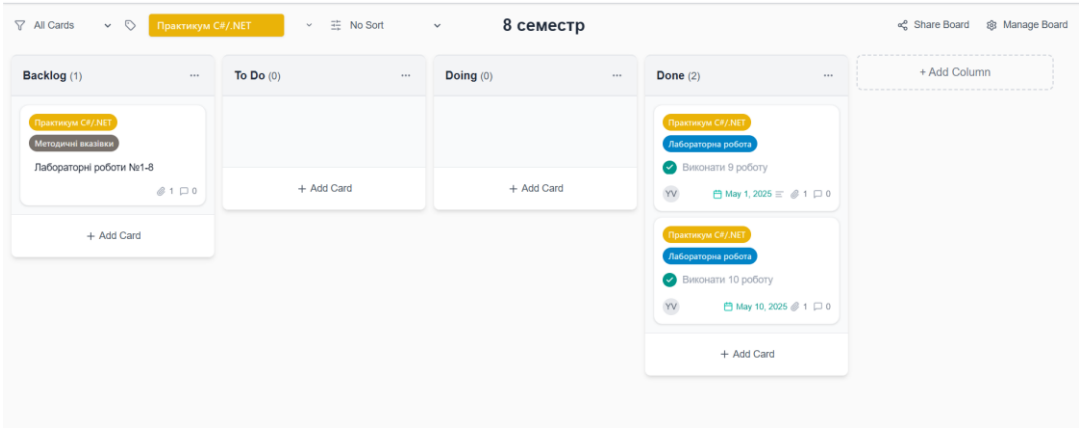


Рисунок 3.27 – Вигляд дошки після застосування фільтру

Обрано картку «Виконати 9 роботу». На її прикладі перевірено функціонал видалення. Відкрито випадаюче меню та вибрано кнопку «Delete». На екрані відображено вікно підтвердження:

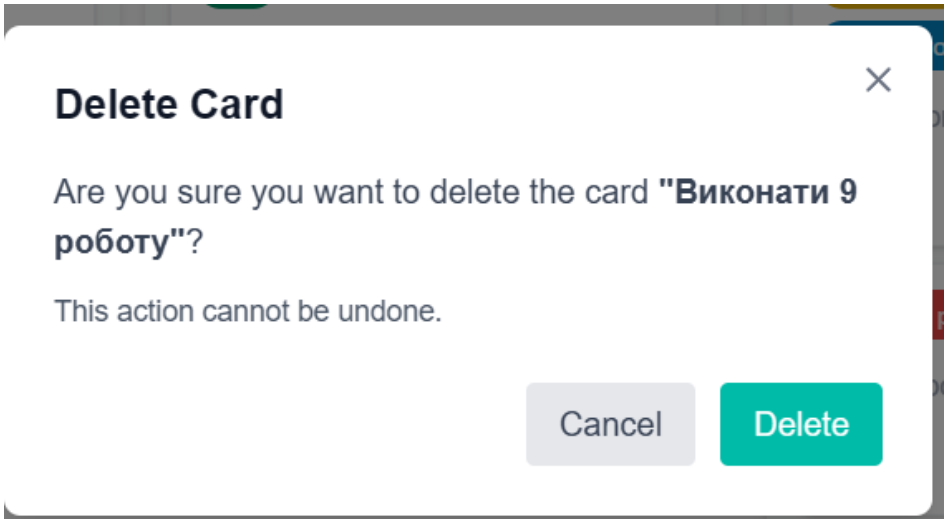


Рисунок 3.28 – Вікно підтвердження видалення картки

Після підтвердження дії натискання кнопки «Delete» у спливаючому вікні, клієнт надсилає на сервер DELETE запит до CardsController, у якому викликається метод DeleteCard:

```
[HttpDelete("{id}")]
        public async Task<IActionResult> DeleteCard(long boardId,
long id)
        {
                await  _boardValidationService.ValidateBoardAsync
(_context, boardId, User.GetCurrentUserId());
             var card = await _context.Cards.Where(c => c.Id == id
&& c.Column.BoardId == boardId).FirstOrDefaultAsync();
             if (card == null){return NotFound("Card not found");}
             _context.Cards.Remove(card);
             await _context.SaveChangesAsync();
             return Ok();}
```

Усередині методу відбувається пошук сутності Card за вказаним ідентифікатором. Якщо картку знайдено, вона видаляється з контексту за допомогою Remove(). Далі Entity Framework Core генерує SQL-команду DELETE і надсилає її до бази даних під час виклику SaveChangesAsync().

Вигляд колонки «Done» після видалення картки «Виконати 9 роботу»:

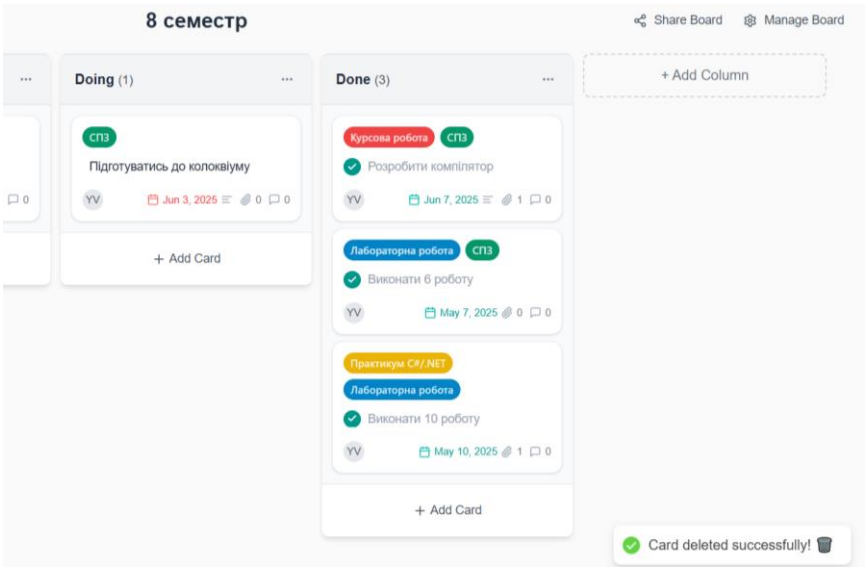


Рисунок 3.29 – Колонка «Done» після видалення картки

Далі проведено випробування функціоналу видалення колонки. Відкрито випадаюче меню для колонки «Doing» та виберемо кнопку «Delete». На екрані буде відображено таке вікно підтвердження:

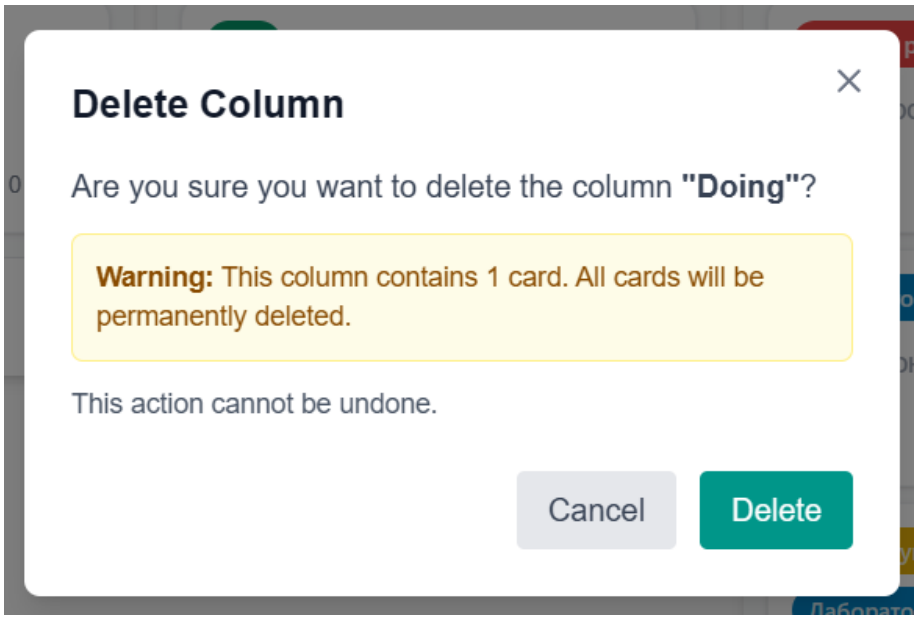


Рисунок 3.30 – Вікно підтвердження видалення колонки

Після підтвердження видалення колонки «Doing», дошка матиме такий вигляд:

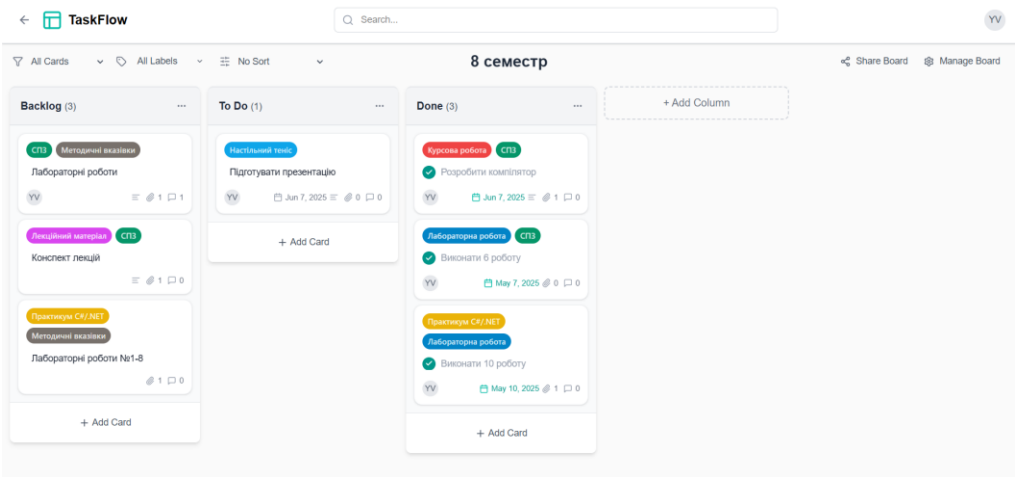


Рисунок 3.31 – Вигляд дошки після видалення колонки

Також розроблено меню управління дошкою. Кнопка «Manage Board», що знаходиться у правому верхньому куті сторінки, при натисненні відкриває бічне меню, яке дозволяє проводити дії над дошкою:

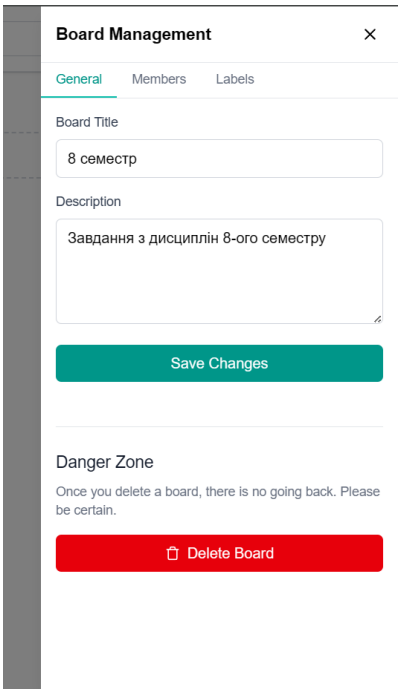


Рисунок 3.32 – Вигляд бічного меню управління дошкою

Змінено опис дошки на «Завдання з дисциплін, які потрібно виконати для успішного завершення 8-ого семестру». При натисненні кнопки «Save Changes», на сервер відправляється PUT запит до BoardsController, де виконується метод UpdateBoard:

```
[HttpPut("{id}")]
public async Task<IActionResult> UpdateBoard(long id, UpsertBoardDto boardDto) {if (!ModelState.IsValid)
{return BadRequest(ModelState); }
var currentUserId = User.GetCurrentUserId();
var board = await _context.Boards.Include(b => b.Members).FirstOrDefaultAsync(b => b.Id == id);
_boardValidationService.ValidateBoard(board, currentUserId);
```

```
            var member = board!.Members.FirstOrDefault(m => m.UserId
== currentUserId);
            if (member!.Role != BoardMemberRole.Admin && member.Role
!= BoardMemberRole.Owner)
             { return Forbid();}
             board.Title = boardDto.Title;
             board.Description = boardDto.Description;
             await _context.SaveChangesAsync();
             return NoContent();}
```

У цьому методі відбувається перевірка валідності вхідної моделі, авторизація користувача, а також перевірка його ролі. Лише користувачі з роллю admin або owner мають право змінювати дошку. Після оновлення заголовка та опису EF Core генерує SQL-команду UPDATE і надсилає її до бази даних під час виклику SaveChangesAsync().

Вигляд вкладки «General» в меню управління дошкою після зміни опису:

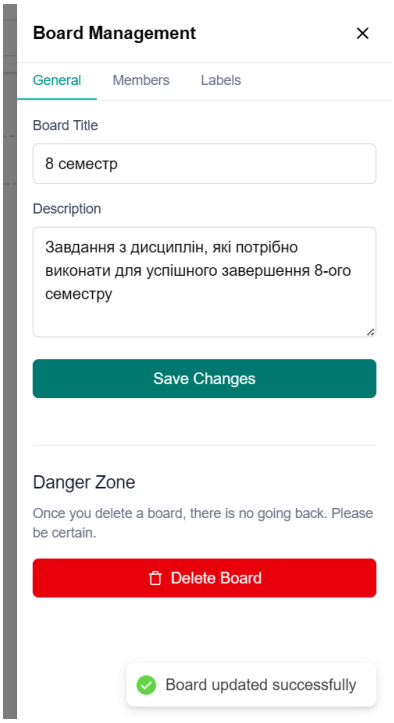


Рисунок 3.33 – Вигляд бічного меню управління дошкою після змін

У вкладці «Members» доступне управління учасниками дошки, зокрема зміна їх ролей, додавання чи видалення учасників (рис. 3.34).

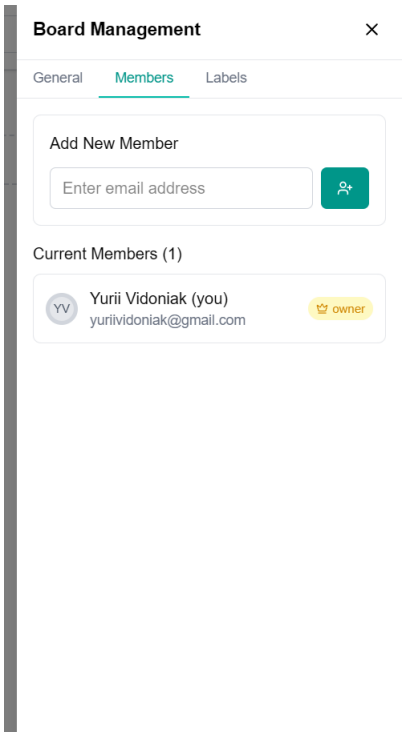


Рисунок 3.34 – Вигляд вкладки «Members»

У поле «Add new member» було введено електронну пошту тестового користувача (test@gmail.com), після чого натиснуто кнопку додавання. Клієнт перевіряє введену адресу на коректність і, якщо вона дійсна, надсилає запит до сервера:

```
const handleAddMember = async () => {
  if (newMemberEmail.trim()) {
    const success = await onAddMember(newMemberEmail);
    if (success) {
      setNewMemberEmail("");
    }
  }
};
```

На сервері опрацьовується POST-запит до BoardMembersController, де викликається метод AddMember:

```
[HttpPost]
    public async Task<IActionResult> AddMember(long boardId, AddBoardMemberDto addMemberDto){var board = await _context.Boards.Include(b => b.Members).FirstOrDefaultAsync(b => b.Id == boardId);
        var currentUserId = User.GetCurrentUserId();
    _boardValidationService.ValidateBoard(board, currentUserId);
    var currentUserMember = board!.Members.FirstOrDefault(m => m.UserId == currentUserId);
    if (currentUserMember!.Role != BoardMemberRole.Owner && currentUserMember.Role != BoardMemberRole.Admin)
    {return BadRequest("Only admins and owners can add members");}
    var userToAdd = await _context.Users.Where(u => u.Email == addMemberDto.Email).FirstOrDefaultAsync();
    if (userToAdd == null)
    {return NotFound("User with this email does not exist");}
    if (board.Members.Any(m => m.UserId == userToAdd.Id))
    { return BadRequest("User is already a member of this board");}
    if (addMemberDto.BoardMemberRole == BoardMemberRole.Admin)
    { if (currentUserMember.Role != BoardMemberRole.Owner)
    {return BadRequest("Only the owner can add admins");}}
    var newMember = new BoardMember {BoardId = boardId,
    UserId = userToAdd.Id,
    Role = addMemberDto.BoardMemberRole, };
    board.Members.Add(newMember);
    await _context.SaveChangesAsync();
    return NoContent();}
```

Метод перевіряє, чи поточний користувач має право додавати інших (доступно лише admin або owner), чи існує користувач з вказаною поштою, і чи ще не є він учасником дошки. Якщо всі умови виконано, новий учасник

додається до колекції Members, і EF Core виконує SQL-запит INSERT до таблиці, коли викликається SaveChangesAsync().

Вигляд меню після додавання користувача:

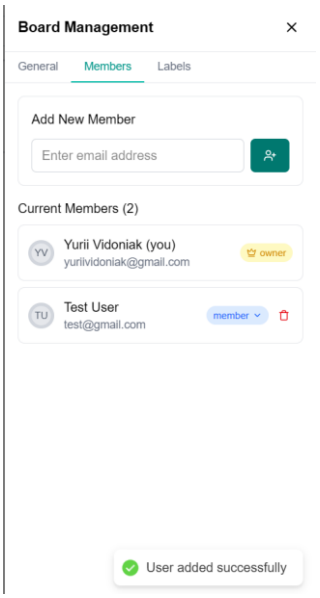


Рисунок 3.35 – Вигляд меню після додавання користувача

У вкладці «Labels» доступне управління мітками дошки, а саме додавання, редагування чи видалення міток. При видаленні мітки, яка прикріплена до карток, вона буде видалена з усіх карток, на яких вона була присутня. Вибрано мітку «Настільний теніс» і відкрито меню її редагування:

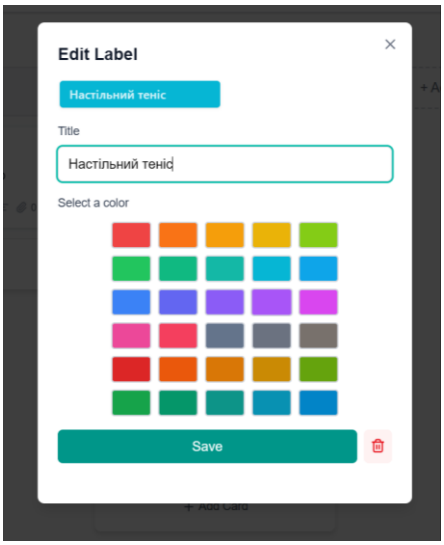


Рисунок 3.36 – Меню редагування мітки

Натиснуто на кнопку видалення мітки, з’являється стандартне спливаюче вікно підтвердження видалення. Після підтвердження видалення, бічне меню закривається, вигляд дошки зміниться таким чином:

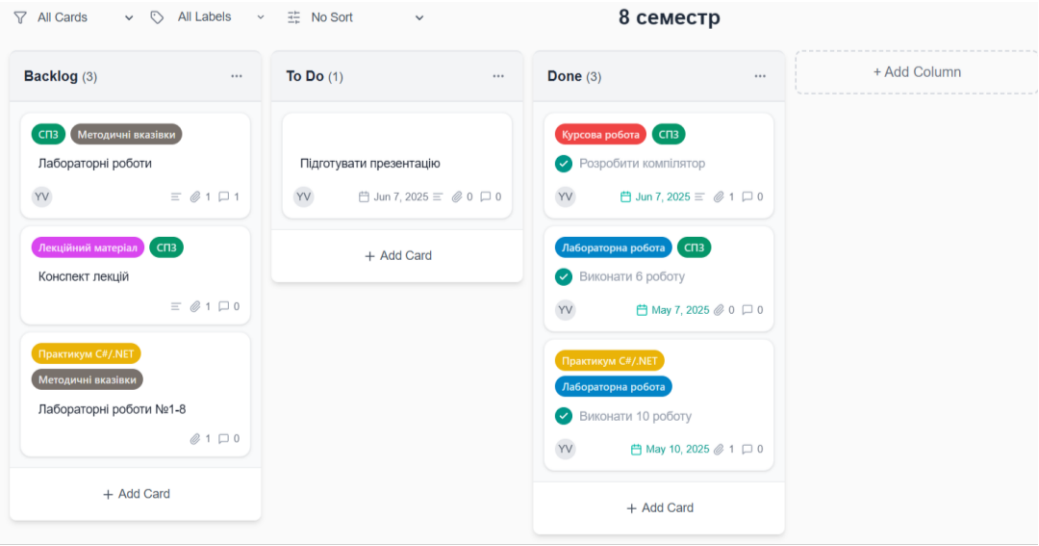


Рисунок 3.37 – Вигляд дошки після видалення мітки

Помітно, що картка «Підготувати презентацію», яка раніше містила мітку «Настільний теніс», зараз не має ніякої мітки, отже видалення пройшло успішно.

Код описаних елементів знаходиться у додатку А. Повний код проекту знаходиться на Github [21].

ВИСНОВКИ

У першому розділі бакалаврської роботи сформульовано постановку задачі, подано теоретичні відомості щодо процесу управління завданнями, здійснено аналіз існуючих аналогів (Trello, Jira, TasksBoard), а також визначено вимоги до структури вхідних даних і принципи їх подальшого відображення в інтерфейсі користувача.

У другому розділі зосереджено на архітектурному плануванні системи. Було розроблено реляційну модель бази даних, побудовано UML-діаграми варіантів використання (use-case) для опису взаємодії між користувачами та функціоналом додатку, а також діаграми послідовностей для ілюстрації внутрішніх процесів.

У третьому розділі, на основі проєктних рішень другого розділу, розроблено повноцінний веб-додаток для керування завданнями студентів із використанням ASP.NET Core на бекенді та React на фронтенді. Реалізовано основні функції системи: реєстрацію та авторизацію користувачів, створення дошок, колонок і карток, керування ролями учасників, фільтрацію, оновлення та видалення даних. Окремо наведено ключові фрагменти коду, що демонструють реалізацію REST API та реактивного UI.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Методологія Kanban: дошки, принципи та сервіси, які вам допоможуть – UASpectr. Новини про технології та бізнес. URL: https://uaspectr.com /2021/01/26/shho-take-kanban/ (дата звернення: 01.03.2025).

2. ДСТУ ISO/IEC 2382:2017 (ISO/IEC 2382:2015, IDT). Інформаційні технології. Словник термінів. На заміну ДСТУ ISO/IEC 2382:2005 ; чинний від 2019-01-01. Вид. офіц. Київ : УкрНДНЦ, 2019. 623 с.

3. ДСТУ ISO/IEC 25010:2016 (ISO/IEC 25010:2011, IDT). Інженерія систем і програмних засобів. Вимоги до якості систем і програмних засобів та її оцінювання (SQuaRE). Моделі якості системи та програмних засобів. На заміну ДСТУ ISO/IEC 25010:2015 ; чинний від 2018-01-01. Вид. офіц. Київ : УкрНДНЦ, 2018. 32 с.

4. Зберігайте, упорядковуйте й виконуйте завдання будь-де | Trello. URL: https://trello.com/uk (дата звернення: 11.03.2025).

5. Jira | Issue & Project Tracking Software | Atlassian. URL: https://www. atlassian.com/software/jira (дата звернення: 11.03.2025).

6. TasksBoard | Desktop app for Google Tasks. URL: https://tasks board.com/ (дата звернення: 11.03.2025).

7. Zang A. ASP.NET Core Basics: Mastering Object-Oriented Programming. URL: https://www.telerik.com/blogs/aspnet-core-basics-mastering-object-oriented-programming-concepts (дата звернення: 15.03.2025).

8. PostgreSQL: Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 22.03.2025).

9. Normalization in SQL DBMS: 1NF, 2NF, 3NF, and BCNF Examples | PopSQL. URL: https://popsql.com/blog/normalization-in-sql (дата звернення: 26.03.2025).

10. pgAdmin - PostgreSQL Tools. URL: https://www.pgadmin.org/ (дата звернення: 07.04.2025).

11. ERD Tool – pgAdmin 4 9.3 documentation. URL: https://www.pgadmin.org /docs/pgadmin4/9.3/erd_tool.html (дата звернення: 07.04.2025).

12. Ideal Modeling & Diagramming Tool for Agile Team Collaboration. URL: https://www.visual-paradigm.com/ (дата звернення: 07.04.2025).

13. Documentation - TypeScript for JavaScript Programmers. URL: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html (дата звернення: 30.04.2025).

14. Store | Redux. URL: https://redux.js.org/api/store (дата звернення: 07.05.2025).

15. Гайд по Docker: концепція, устрій та принцип роботи. URL: https:// spacelab.ua/articles/Docker/ (дата звернення: 12.05.2025).

16. Overview of Entity Framework Core - EF Core. URL: https://learn. microsoft.com/uk-ua/ef/core/ (дата звернення: 15.05.2025).

17. Introduction to Identity on ASP.NET Core. URL: https://learn.microsoft. com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-9.0& tabs=visual-studio (дата звернення: 16.05.2025).

18. Як використовувати JSON Web Tokens (JWT) для автентифікації. URL: https://devzone.org.ua/post/iak-vykorystovuvaty-json-web-tokens-jwt-dlia-avtentyfikatsiyi (дата звернення: 16.05.2025).

19. Simple authorization in ASP.NET Core. URL: https://learn. microsoft.com/en-us/aspnet/core/security/authorization/simple? view=aspnetcore-9.0 (дата звернення: 18.05.2025).

20. Create Data Transfer Objects (DTOs). URL: https://learn.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5 (дата звернення: 20.05.2025).

21. GitHub - YuriiVid/Bachelors. URL: https://github.com/YuriiVid/Bachelors (дата звернення: 29.05.2025).

22. Reynders F. Modern API Design with ASP.NET Core 2: Building Cross-Platform Back-End Systems. Apress, 2018. 260 с.

23. Lauret A. Design of Web APIs. Manning Publications Co. LLC, 2019. 400 с.

24. Lock A. ASP. NET Core in Action, Second Edition. Manning Publications Co. LLC, 2021. 832 c

25. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. O'Reilly Media, 2020. 310 c.

26. Stefanov S. React: Up and Running: Building Web Applications. O'Reilly Media, Incorporated, 2021. 222 c.

27. Griffiths D., David G. React Cookbook: Recipes for Mastering the React Framework. O'Reilly Media, Incorporated, 2021. 500 c.

28. Rippon C. Learn React with TypeScript: A beginner's guide to reactive web development with React 18 and TypeScript, 2nd Edition. Packt Publishing, 2023. 474 c.

ДОДАТКИ

Вміст файлів проекту

API/Controllers/BoardsController.cs

```
using API.DTOs;
using API.Extensions;
using API.Models;
using API.Services;
using AutoMapper;
using AutoMapper.QueryableExtensions;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
namespace API.Controllers;
[Route("api/boards")]
[Authorize]
[ApiController]
public class BoardsController : Controller{    private readonly AppDbContext
_context;
    private readonly IBoardValidationService _boardValidationService;
    private readonly IMapper _mapper;
    public BoardsController(AppDbContext context, IBoardValidationService
boardValidator, IMapper mapper){        _context = context;
        _boardValidationService = boardValidator;
        _mapper = mapper;}
    [HttpGet]
    public async Task<ActionResult<IList<BriefBoardDto>>>
GetBoards(){        var boards = await _context
            .Boards.Include(b => b.Members)
            .AsNoTracking()
            .Where(b => b.Members.Any(m => m.UserId == User.GetCurrentUserId()))
            .ProjectTo<BriefBoardDto>(_mapper.ConfigurationProvider)
            .ToListAsync();
        return Ok(boards);}
    [HttpGet("{id}")]
    public async Task<ActionResult<FullBoardDto>> GetBoard(long
id){        await _boardValidationService.ValidateBoardAsync(_context, id,
User.GetCurrentUserId());
        var fullBoard = await _context
            .Boards.AsNoTracking()
            .Where(b => b.Id == id)
            .ProjectTo<FullBoardDto>(_mapper.ConfigurationProvider)
            .FirstOrDefaultAsync();
        return Ok(fullBoard);}
    [HttpPost]
    public async Task<ActionResult> CreateBoard(UpsertBoardDto
boardDto){        if (!ModelState.IsValid){            return
BadRequest(ModelState);}
        var board = new Board { Title = boardDto.Title, Description =
boardDto.Description };
        var currentUserId = User.GetCurrentUserId();
        board.Members.Add(new BoardMember { UserId = currentUserId, Role =
BoardMemberRole.Owner });
        // Create default labels
        var defaultLabels = GetDefaultLabels();
        foreach (var label in defaultLabels){            label.BoardId =
board.Id;
            board.Labels.Add(label);}
        _context.Boards.Add(board);
        await _context.SaveChangesAsync();
        return CreatedAtAction(nameof(GetBoard), new { id = board.Id }, new {
board.Id, board.Title });}
    [HttpPut("{id}")]
```

```
    public async Task<IActionResult> UpdateBoard(long id, UpsertBoardDto
boardDto){        if (!ModelState.IsValid){            return
BadRequest(ModelState);}
        var currentUserId = User.GetCurrentUserId();
        var board = await _context.Boards.Include(b =>
b.Members).FirstOrDefaultAsync(b => b.Id == id);
        _boardValidationService.ValidateBoard(board, currentUserId);
        var member = board!.Members.FirstOrDefault(m => m.UserId ==
currentUserId);
        if (member!.Role != BoardMemberRole.Admin && member.Role !=
BoardMemberRole.Owner){            return Forbid();}
        board.Title = boardDto.Title;
        board.Description = boardDto.Description;
        await _context.SaveChangesAsync();
        return NoContent();}
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteBoard(long id){        var board =
await _context.Boards.Include(b => b.Members).FirstOrDefaultAsync(b => b.Id ==
id);
        var currentUserId = User.GetCurrentUserId();
        _boardValidationService.ValidateBoard(board, currentUserId);
        var userRole = board!.Members.FirstOrDefault(m => m.UserId ==
currentUserId)!.Role;
        if (userRole != BoardMemberRole.Owner){            return Forbid();}
        _context.Boards.Remove(board);
        await _context.SaveChangesAsync();
        return Ok();}
```

API/Controllers/CardsController.cs

```
using API.DTOs;
using API.Extensions;
using API.Models;
using API.Services;
using AutoMapper;
using AutoMapper.QueryableExtensions;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.JsonPatch;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Newtonsoft.Json;
using NodaTime;
using NodaTime.Text;
namespace API.Controllers;
[Route("api/boards/{boardId}/cards")]
[Authorize]
[ApiController]
public class CardsController : Controller{    private readonly AppDbContext
_context;
    private readonly IBoardValidationService _boardValidationService;
    private readonly ILogger _logger;
    private readonly IMapper _mapper;
    public CardsController(
        AppDbContext context,
        IBoardValidationService boardValidator,
        ILogger<CardsController> logger,
        IMapper mapper){        _logger = logger;
        _boardValidationService = boardValidator;
        _context = context;
        _mapper = mapper;}
    [HttpGet]
    public async Task<ActionResult<List<BriefCardDto>>> GetCards(long
boardId){        await _boardValidationService.ValidateBoardAsync(_context,
boardId, User.GetCurrentUserId());
```

```
        var briefCards = await _context
            .Cards.AsNoTracking()
            .Where(c => c.Column.BoardId == boardId)
            .ProjectTo<BriefCardDto>(_mapper.ConfigurationProvider)
            .ToListAsync();
        return Ok(briefCards);}
    [HttpGet("{id}")]
    public async Task<ActionResult<FullCardDto>> GetCard(long boardId, long
id){        await _boardValidationService.ValidateBoardAsync(_context, boardId,
User.GetCurrentUserId());
        var fullCard = await _context
            .Cards.AsNoTracking()
            .Where(b => b.Id == id)
            .ProjectTo<FullCardDto>(_mapper.ConfigurationProvider)
            .FirstOrDefaultAsync();
        if (fullCard == null){            return NotFound("Card not found");}
        return Ok(fullCard);}
    [HttpPost]
    public async Task<ActionResult> CreateCard(long boardId, AddCardDto
cardDto){        if (!ModelState.IsValid){            return
BadRequest(ModelState);}
        await _boardValidationService.ValidateBoardAsync(_context, boardId,
User.GetCurrentUserId());
        var maxPosition =
            await _context.Cards.Where(c => c.ColumnId ==
cardDto.ColumnId).MaxAsync(c => (int?)c.Position) ?? -1;
        var card = new Card{            Title = cardDto.Title,
            ColumnId = cardDto.ColumnId,
            Position = maxPosition + 1,};
        _context.Cards.Add(card);
        await _context.SaveChangesAsync();
        return CreatedAtAction(nameof(GetCard), new { boardId, id = card.Id },
card);}
    [HttpPut("{id}")]
    public async Task<IActionResult> UpdateCard(long boardId, long id,
UpdateCardDto cardDto){        if (!ModelState.IsValid){            return
BadRequest(ModelState);}
        await _boardValidationService.ValidateBoardAsync(_context, boardId,
User.GetCurrentUserId());
        var card = await _context.Cards.Where(c => c.Id == id &&
c.Column.BoardId == boardId).FirstOrDefaultAsync();
        if (card == null){            return NotFound("Card not found");}
        card.Title = cardDto.Title;
        card.Description = cardDto.Description;
        card.DueDate = cardDto.DueDate;
        await _context.SaveChangesAsync();
        return NoContent();}
    [HttpPatch("{id}")]
    public async Task<IActionResult> PatchCard(long boardId, long id,
JsonPatchDocument<UpdateCardDto> patchDoc){        if (patchDoc ==
null){            return BadRequest();}
        await _boardValidationService.ValidateBoardAsync(_context, boardId,
User.GetCurrentUserId());
        var card = await _context
            .Cards.Include(c => c.Column)
            .Where(c => c.Id == id && c.Column.BoardId == boardId)
            .FirstOrDefaultAsync();
        if (card == null){            return NotFound("Card not found");}
        var dto = _mapper.Map<UpdateCardDto>(card);
        var dueOp = patchDoc.Operations.FirstOrDefault(o =>
            o.path.Equals("/dueDate", StringComparison.OrdinalIgnoreCase));
```

```
        if (dueOp != null && dueOp.value is string s){            var parse =
InstantPattern.ExtendedIso.Parse(s);
            if (!parse.Success)
                ModelState.AddModelError("dueDate", "Invalid ISO-8601 Instant");
            else
                dto.DueDate = parse.Value;
            patchDoc.Operations.Remove(dueOp);}
        patchDoc.ApplyTo(dto, ModelState);
        if (!TryValidateModel(dto)){            return BadRequest(ModelState);}
        _mapper.Map(dto, card);
        await _context.SaveChangesAsync();
        return NoContent();}
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteCard(long boardId, long
id){        await _boardValidationService.ValidateBoardAsync(_context, boardId,
User.GetCurrentUserId());
        var card = await _context.Cards.Where(c => c.Id == id &&
c.Column.BoardId == boardId).FirstOrDefaultAsync();
        if (card == null){            return NotFound("Card not found");}
        _context.Cards.Remove(card);
        await _context.SaveChangesAsync();
        return Ok();}
```

API/Controllers/ColumnsController.cs

```
using System.Security.Claims;
using API.DTOs;
using API.Extensions;
using API.Models;
using API.Services;
using AutoMapper;
using AutoMapper.QueryableExtensions;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
namespace API.Controllers;
[Route("api/boards/{boardId}/columns")]
[Authorize]
[ApiController]
public class ColumnsController : Controller{    private readonly AppDbContext
_context;
    private readonly IBoardValidationService _boardValidationService;
    private readonly IMapper _mapper;
    public ColumnsController(AppDbContext context, IBoardValidationService
boardValidator, IMapper mapper){        _boardValidationService =
boardValidator;
        _context = context;
        _mapper = mapper;}
    [HttpGet]
    public async Task<ActionResult<IList<BriefColumnDto>>> GetColumns(long
boardId){        await _boardValidationService.ValidateBoardAsync(_context,
boardId, User.GetCurrentUserId());
        var briefColumns = await _context
            .Columns.Where(c => c.BoardId == boardId)
            .ProjectTo<BriefColumnDto>(_mapper.ConfigurationProvider)
            .ToListAsync();
        return Ok(briefColumns);}
    [HttpGet("{id}")]
    public async Task<ActionResult<FullColumnDto>> GetColumn(long boardId, long
id){        await _boardValidationService.ValidateBoardAsync(_context, boardId,
User.GetCurrentUserId());
        var fullColumn = await _context
            .Columns.AsNoTracking()
            .Where(c => c.Id == id && c.BoardId == boardId)
```

```
            .ProjectTo<FullColumnDto>(_mapper.ConfigurationProvider)
            .FirstOrDefaultAsync();
        if (fullColumn == null){            return NotFound("Column not
found");}
        return Ok(fullColumn);}
    [HttpPost]
    public async Task<ActionResult<BriefColumnDto>> CreateColumn(long boardId,
UpsertColumnDto dto){        await
_boardValidationService.ValidateBoardAsync(_context, boardId,
User.GetCurrentUserId());
        var maxPosition = await _context.Columns.Where(c => c.BoardId ==
boardId).MaxAsync(c => (int?)c.Position) ?? -1;
        var column = new Column{            Title = dto.Title,
            BoardId = boardId,
            Position = maxPosition + 1,};
        _context.Columns.Add(column);
        await _context.SaveChangesAsync();
        return CreatedAtAction(nameof(GetColumn), new { boardId, id = column.Id
}, new { column.Id, column.Title });}
    [HttpPut("{id}")]
    public async Task<IActionResult> UpdateColumn(long boardId, long id,
UpsertColumnDto dto){        await
_boardValidationService.ValidateBoardAsync(_context, boardId,
User.GetCurrentUserId());
        var column = await _context.Columns.Where(c => c.Id == id && c.BoardId
== boardId).FirstOrDefaultAsync();
        if (column == null){            return NotFound("Column not found");}
        column.Title = dto.Title;
        await _context.SaveChangesAsync();
        return NoContent();}
    [HttpDelete("{id}")]
    public async Task<ActionResult> DeleteColumn(long boardId, long
id){        await _boardValidationService.ValidateBoardAsync(_context, boardId,
User.GetCurrentUserId());
        var column = await _context.Columns.Where(c => c.Id == id && c.BoardId
== boardId).FirstOrDefaultAsync();
        if (column == null){            return NotFound("Column not found");}
        _context.Columns.Remove(column);
        await _context.SaveChangesAsync();
        return NoContent();}
    [HttpPost("{id}/move")]
```

API/Controllers/BoardMemberController.cs

```
using API.DTOs;
using API.Extensions;
using API.Models;
using API.Services;
using AutoMapper;
using AutoMapper.QueryableExtensions;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
namespace API.Controllers;
[Route("api/boards/{boardId}/members")]
[Authorize]
[ApiController]
public class BoardMembersController : Controller{    private readonly
AppDbContext _context;
    private readonly IBoardValidationService _boardValidationService;
    private readonly IMapper _mapper;
    public BoardMembersController(AppDbContext context, IBoardValidationService
boardValidator, IMapper mapper){        _context = context;
        _boardValidationService = boardValidator;
```

```
        _mapper = mapper;}
    [HttpGet]
    public async Task<ActionResult<IList<BoardMemberDto>>> GetBoardMembers(long
boardId){        await _boardValidationService.ValidateBoardAsync(_context,
boardId, User.GetCurrentUserId());
        var members = await _context
            .BoardMembers.AsNoTracking()
            .Where(m => m.BoardId == boardId)
            .ProjectTo<BoardMemberDto>(_mapper.ConfigurationProvider)
            .FirstOrDefaultAsync(b => b.Id == boardId);
        return Ok(members);}
    [HttpPost]
    public async Task<IActionResult> AddMember(long boardId, AddBoardMemberDto
addMemberDto){        var board = await _context.Boards.Include(b =>
b.Members).FirstOrDefaultAsync(b => b.Id == boardId);
        var currentUserId = User.GetCurrentUserId();
        _boardValidationService.ValidateBoard(board, currentUserId);
        var currentUserMember = board!.Members.FirstOrDefault(m => m.UserId ==
currentUserId);
        if (currentUserMember!.Role != BoardMemberRole.Owner &&
currentUserMember.Role != BoardMemberRole.Admin){            return
BadRequest("Only admins and owners can add members");}
        var userToAdd = await _context.Users.Where(u => u.Email ==
addMemberDto.Email).FirstOrDefaultAsync();
        if (userToAdd == null){            return NotFound("User with this email
does not exist");}
        if (board.Members.Any(m => m.UserId == userToAdd.Id)){            return
BadRequest("User is already a member of this board");}
        if (addMemberDto.BoardMemberRole ==
BoardMemberRole.Admin){            if (currentUserMember.Role !=
BoardMemberRole.Owner){                return BadRequest("Only the owner can add
admins");}}
        var newMember = new BoardMember{            BoardId = boardId,
            UserId = userToAdd.Id,
            Role = addMemberDto.BoardMemberRole,};
        board.Members.Add(newMember);
        await _context.SaveChangesAsync();
        return NoContent();}
    [HttpPatch("{userId}/role")]
    public async Task<IActionResult> UpdateMemberRole(long boardId, int userId,
ChangeBoardMemberRoleDto changeRoleDto){        var board = await
_context.Boards.Include(b => b.Members).FirstOrDefaultAsync(b => b.Id ==
boardId);
        var currentUserId = User.GetCurrentUserId();
        _boardValidationService.ValidateBoard(board, currentUserId);
        var currentUserMember = board!.Members.FirstOrDefault(m => m.UserId ==
currentUserId);
        var memberToUpdate = board.Members.FirstOrDefault(m => m.UserId ==
userId);
        if (memberToUpdate == null){            return NotFound("Member not
found");}
        if (changeRoleDto.Role == BoardMemberRole.Owner){            return
BadRequest("Cannot change owner's role. Use transfer ownership instead");}
        // Only owner can manage admins
        if (changeRoleDto.Role == BoardMemberRole.Admin || memberToUpdate.Role
== BoardMemberRole.Admin){            if (currentUserMember!.Role !=
BoardMemberRole.Owner){                return BadRequest("Only the owner can
manage admins");}}
        // Admins can only manage regular users and observers
        else if (currentUserMember!.Role != BoardMemberRole.Owner &&
currentUserMember.Role != BoardMemberRole.Admin){            return
BadRequest("You don't have permission to change roles");}
```

```
        memberToUpdate.Role = changeRoleDto.Role;
        await _context.SaveChangesAsync();
        return NoContent();}
    [HttpDelete("{userId}")]
    public async Task<IActionResult> RemoveMember(long boardId, int
userId){        var board = await _context.Boards.Include(b =>
b.Members).FirstOrDefaultAsync(b => b.Id == boardId);
        var currentUserId = User.GetCurrentUserId();
        _boardValidationService.ValidateBoard(board, currentUserId);
        var currentUserMember = board!.Members.FirstOrDefault(m => m.UserId ==
currentUserId);
        var memberToRemove = board.Members.FirstOrDefault(m => m.UserId ==
userId);
        if (memberToRemove == null){            return NotFound("Member to
remove is not found");}
        if (memberToRemove.Role == BoardMemberRole.Owner){            return
BadRequest("Cannot remove the owner");}
        // Only owner can remove admins
        if (memberToRemove.Role == BoardMemberRole.Admin){            if
(currentUserMember!.Role != BoardMemberRole.Owner){                return
BadRequest("Only the owner can remove admins");}}
        // Admins can only remove regular users and observers
        else if (currentUserMember!.Role != BoardMemberRole.Owner &&
currentUserMember.Role != BoardMemberRole.Admin){            return
BadRequest("You don't have permission to remove members");}
        _context.BoardMembers.Remove(memberToRemove);
        await _context.SaveChangesAsync();
        return NoContent();}
```

API/Controllers/AuthController.cs

```
using System.Security.Claims;
using System.Text;
using API.DTOs;
using API.Extensions;
using API.Models;
using API.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.WebUtilities;
using Microsoft.EntityFrameworkCore;
namespace API.Controllers;
[Route("api/auth")]
[ApiController]
public class AuthController : ControllerBase{    private readonly IConfiguration
_config;
    private readonly ILogger _logger;
    private readonly UserManager<AppUser> _userManager;
    private readonly SignInManager<AppUser> _signInManager;
    private readonly JWTService _jwtService;
    private readonly EmailService _emailService;
    private readonly AppDbContext _context;
    public AuthController(
        IConfiguration config,
        UserManager<AppUser> userManager,
        JWTService jwtService,
        SignInManager<AppUser> signInManager,
        EmailService emailService,
        ILogger<AuthController> logger,
        AppDbContext context){        _config = config;
        _userManager = userManager;
        _jwtService = jwtService;
        _signInManager = signInManager;
```

```
        _emailService = emailService;
        _logger = logger;
        _context = context;}
    [HttpGet("refresh-user-token")]
    public async Task<IActionResult> RefreshUserToken(){        if
(!Request.Cookies.TryGetValue("refreshToken", out var
incomingToken)){            return Unauthorized("No token found");}
        var tokenEntry = await _context.UserTokens.SingleOrDefaultAsync(t =>
            t.LoginProvider == "RefreshToken" && t.Name == "MyAppRefreshToken"
&& t.Value == incomingToken);
        if (tokenEntry == null){            return Unauthorized("Invalid
token");}
        var user = await
_userManager.FindByIdAsync(tokenEntry.UserId.ToString());
        if (user == null){            return Unauthorized("No user found");}
        if (!_jwtService.IsRefreshTokenValid(incomingToken)){            await
_userManager.RemoveAuthenticationTokenAsync(user, "RefreshToken",
"MyAppRefreshToken");
            return Unauthorized();}
        var newRefresh = await _jwtService.CreateRefreshTokenAsync(user);
        Response.Cookies.Append(
            "refreshToken",
            newRefresh,
            new CookieOptions{                HttpOnly = true,
                Secure = true,
                SameSite = SameSiteMode.None, // allow cross-site from your SPA
                Expires =
DateTime.UtcNow.AddDays(_config.GetValue<int>("JWT:RefreshTokenExpiresInDays")),
});
        return await _userManager.IsLockedOutAsync(user) ? Unauthorized() :
Ok(await CreateAppUserDto(user));}
    [HttpPost("register")]
    public async Task<IActionResult> Register(RegisterDto model){        if
(await CheckEmailExistsAsync(model.Email)){            return
BadRequest("Account with given email already exists");}
        var userToAdd = new AppUser{            FirstName = model.FirstName,
            LastName = model.LastName,
            UserName = model.Email.ToLower(),
            Email = model.Email.ToLower(),};
        var result = await _userManager.CreateAsync(userToAdd, model.Password);
        if
(!result.Succeeded){            _logger.LogError(result.Errors.ToString());
            return BadRequest(result.Errors);}
        await _userManager.AddToRoleAsync(userToAdd, SD.UserRole);
        try{            return await SendConfirmEmailAsync(userToAdd)
                ? Ok(
                    new{                        title = "Account Created",
                        message = "Your account has been created, please confirm
your email address",})
                : BadRequest("Failed to send email. Please contact admin");}
        catch (Exception ex){            _logger.LogError(ex.Message);
            return BadRequest("Failed to send email. Please contact admin");}}
    [HttpPost("login")]
    public async Task<IActionResult> Login(LoginDto model){        var user =
await _userManager.FindByNameAsync(model.UserName);
        if (user == null){            return Unauthorized("Invalid username or
password");}
        if (!user.EmailConfirmed){            return Unauthorized("Please
confirm your email.");}
        var result = await _signInManager.CheckPasswordSignInAsync(user,
model.Password, true);
        if (result.IsLockedOut){            return Unauthorized(
```

```
                $"Your account has been locked due to too many failed attempts.
You should wait until {user.LockoutEnd} (UTC time) to be able to login");}
        var refresh = await _jwtService.CreateRefreshTokenAsync(user);
        Response.Cookies.Append(
            "refreshToken",
            refresh,
            new CookieOptions{                HttpOnly = true,
                Secure = true,
                SameSite = SameSiteMode.None,
                Expires =
DateTime.UtcNow.AddDays(_config.GetValue<int>("JWT:RefreshTokenExpiresInDays")),
});
        return result.Succeeded ? Ok(await CreateAppUserDto(user)) :
Unauthorized("Invalid username or password");}
    [HttpPut("confirm-email")]
    public async Task<IActionResult> ConfirmEmail(ConfirmEmailDto
model){        var user = await _userManager.FindByEmailAsync(model.Email);
        if (user == null){            return Unauthorized("This email address
has not been registered yet");}
        if (user.EmailConfirmed == true){            return BadRequest("Your
email was confirmed before. Please login to your account");}
        try{            var decodedTokenBytes =
WebEncoders.Base64UrlDecode(model.Token);
            var decodedToken = Encoding.UTF8.GetString(decodedTokenBytes);
            var result = await _userManager.ConfirmEmailAsync(user,
decodedToken);
            return result.Succeeded
                ? Ok(new { title = "Email confirmed", message = "Your email
address is confirmed. You can login now" })
                : BadRequest("Invalid token. Please try again");}
        catch (Exception ex){            _logger.LogError(ex.Message);
            return BadRequest("Invalid token. Please try again");}}
    [HttpPost("resend-email-confirmation-link/{email}")]
    public async Task<IActionResult> ResendEmailConfirmationLink(string
email){        if (string.IsNullOrEmpty(email))
            return BadRequest("Invalid email");
        var user = await _userManager.FindByEmailAsync(email);
        if (user == null){            return Unauthorized("This email address
has not been registerd yet");}
        if (user.EmailConfirmed == true){            return BadRequest("Your
email address was confirmed before. Please login to your account");}
        try{            return await SendConfirmEmailAsync(user)
                ? Ok(new { title = "Confirmation link sent", message = "Please
confirm your email address" })
                : BadRequest("Failed to send email. Please contact admin");}
        catch (Exception ex){            _logger.LogError(ex.Message);
            return BadRequest("Failed to send email. PLease contact admin");}}
    [HttpPost("forgot-password/{email}")]
    public async Task<IActionResult> ForgotUsernameOrPassword(string
email){        if (string.IsNullOrEmpty(email))
            return BadRequest("Invalid email");
        var user = await _userManager.FindByEmailAsync(email);
        if (user == null){            return Unauthorized("This email address
has not been registerd yet");}
        if (user.EmailConfirmed == false){            return BadRequest("Please
confirm your email address first.");}
        try{            return await SendForgotUsernameOrPasswordEmail(user)
                ? Ok(new { title = "Forgot username or password email sent",
message = "Please check your email" })
                : BadRequest("Failed to send email. Please contact admin");}
        catch (Exception ex){            _logger.LogError(ex.Message);
            return BadRequest("Failed to send email. Please contact admin");}}
```

```
    [HttpPut("reset-password")]
    public async Task<IActionResult> ResetPassword(ResetPasswordDto
model){        var user = await _userManager.FindByEmailAsync(model.Email);
        if (user == null){            return Unauthorized("This email address
has not been registerd yet");}
        if (user.EmailConfirmed == false){            return BadRequest("Please
confirm your email address first");}
        try{            var decodedTokenBytes =
WebEncoders.Base64UrlDecode(model.Token);
            var decodedToken = Encoding.UTF8.GetString(decodedTokenBytes);
            var result = await _userManager.ResetPasswordAsync(user,
decodedToken, model.NewPassword);
            return result.Succeeded
                ? Ok(new { title = "Password reset success", message = "Your
password has been reset" })
                : BadRequest("Invalid token. Please try again");}
        catch (Exception ex){            _logger.LogError(ex.Message);
            return BadRequest("Invalid token. Please try again");}}
    [Authorize]
    [HttpPost("logout")]
    public async Task<IActionResult> Logout(){        var user = await
_userManager.FindByIdAsync(User.GetCurrentUserId().ToString());
        if (user == null){            return Unauthorized("User not found");}
        await _userManager.RemoveAuthenticationTokenAsync(user, "RefreshToken",
"MyAppRefreshToken");
        Response.Cookies.Delete("refreshToken");
        return Ok(new { title = "Success", message = "Logged out" });}
    private async Task<bool> CheckEmailExistsAsync(string email){        return
await _userManager.Users.AnyAsync(x => x.Email == email);}
    private async Task<AuthUserDto> CreateAppUserDto(AppUser
appUser){        return new AuthUserDto{            User = new
UserDto{                Id = appUser.Id,
                FirstName = appUser.FirstName,
                LastName = appUser.LastName,
                Email = appUser.Email,
                UserName = appUser.UserName,
                ProfilePictureUrl = appUser.ProfilePictureUrl,},
            JWT = await _jwtService.CreateJWT(appUser),};}
    private async Task<bool> SendConfirmEmailAsync(AppUser user){        var
token = await _userManager.GenerateEmailConfirmationTokenAsync(user);
        token = WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(token));
        var url =
$"{_config["JWT:ClientUrl"]}/{_config["Email:ConfirmEmailPath"]}?token={token}&e
mail={user.Email}";
        return await _emailService.SendConfirmationEmailAsync(user.Email,
$"{user.FirstName} {user.LastName}", url);}
    private async Task<bool> SendForgotUsernameOrPasswordEmail(AppUser
user){        var token = await
_userManager.GeneratePasswordResetTokenAsync(user);
        token = WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(token));
        var url =
$"{_config["JWT:ClientUrl"]}/{_config["Email:ResetPasswordPath"]}?token={token}&
email={user.Email}";
        return await _emailService.SendResetPasswordEmailAsync(user.Email,
$"{user.FirstName} {user.LastName}", url);}}
```

API/Program.cs

```
using System.Reflection;
using System.Text;
using API.Filters;
using API.Middleware;
using API.Models;
```

```
using API.Profiles;
using API.Services;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.FileProviders;
using Microsoft.IdentityModel.Tokens;
using Newtonsoft.Json;
using Newtonsoft.Json.Converters;
using NodaTime;
using NodaTime.Serialization.JsonNet;
using Scalar.AspNetCore;
var builder = WebApplication.CreateBuilder(args);
builder.Logging.AddConsole();
builder
    .Configuration.AddJsonFile(
        $"appsettings.{Environment.GetEnvironmentVariable("ASPNETCORE_ENVIRONMEN
T")}.json",
        optional: false,
        reloadOnChange: true)
    .AddEnvironmentVariables();
// Enum convertertion to be able to send enums as strings in requests and
receive them as enums in the backend
builder
    .Services.AddControllers(opt => opt.Filters.Add<GlobalJsonResponseFilter>())
    .AddNewtonsoftJson(opt =>{        opt.SerializerSettings.DateParseHandling =
DateParseHandling.None;
        opt.SerializerSettings.ConfigureForNodaTime(DateTimeZoneProviders.Tzdb);
        opt.SerializerSettings.Converters.Add(new StringEnumConverter());});
// OpenAPI Setup for better Scalar managment in dev
builder.Services.AddOpenApi(options =>
options.AddDocumentTransformer<BearerSecuritySchemeTransformer>());
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseNpgsql(
        builder.Configuration.GetConnectionString("DefaultConnection"),
        o => o.MapEnum<BoardMemberRole>("BoardMemberRole").UseNodaTime())
);
builder
    .Services.AddIdentityCore<AppUser>(o
=>{        o.SignIn.RequireConfirmedAccount = true;
        o.User.RequireUniqueEmail = true;})
    .AddRoles<Role>()
    .AddRoleManager<RoleManager<Role>>()
    .AddEntityFrameworkStores<AppDbContext>()
    .AddSignInManager<SignInManager<AppUser>>()
    .AddUserManager<UserManager<AppUser>>()
    .AddDefaultTokenProviders();
builder.Services.AddScoped<JWTService>();
builder.Services.AddScoped<EmailService>();
builder.Services.AddScoped<IBoardValidationService, BoardValidationService>();
builder.Services.AddScoped<IFileStorageService, LocalFileStorageService>();
builder.Services.AddAutoMapper(typeof(BoardMappingProfile).Assembly);
builder.Services.AddCors();
builder
    .Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>{        options.TokenValidationParameters = new
TokenValidationParameters{            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["JWT:Key"])),
            ValidIssuer = builder.Configuration["JWT:Issuer"],
            ValidateIssuer = true,
            ValidateAudience = false,};
```

```
        options.Events = new JwtBearerEvents{            OnChallenge = async
context =>{                context.HandleResponse();
                context.Response.StatusCode = StatusCodes.Status401Unauthorized;
                context.Response.ContentType = "application/json";
                var payload = new { message = "You are not authenticated. Please
log in" };
                await context.Response.WriteAsJsonAsync(payload);},
            OnForbidden = async context
=>{               context.Response.StatusCode = StatusCodes.Status403Forbidden;
                context.Response.ContentType = "application/json";
                var payload = new { message = "You are not authorized to access
this resource" };
                await context.Response.WriteAsJsonAsync(payload);},};});
builder
    .Services.AddAuthorizationBuilder()
    .AddPolicy("AdminPolicy", policy => policy.RequireRole("Admin"))
    .AddDefaultPolicy("UserPolicy", policy => policy.RequireRole("User",
"Admin"));
var app = builder.Build();
app.UseCors(opt
=>{    opt.WithOrigins(builder.Configuration["JWT:ClientUrl"]).AllowAnyHeader().
AllowAnyMethod().AllowCredentials();});
if (app.Environment.IsDevelopment()){    app.MapOpenApi();
    app.MapScalarApiReference(options =>{        options.Servers =
Array.Empty<ScalarServer>();});}
app.UseHttpsRedirection();
app.UseStaticFiles(
    new StaticFileOptions{        FileProvider = new
PhysicalFileProvider(Path.Combine(builder.Environment.ContentRootPath,
"Uploads")),
        RequestPath = "/uploads",}
);
app.UseAuthentication();
app.UseAuthorization();
app.MapControllers();
app.UseMiddleware<ExceptionHandlingMiddleware>();
app.Run();
```

API/appsetings.Development.json

```
{    "Logging": {        "LogLevel": {            "Default": "Information",
            "Microsoft.AspNetCore": "Warning",
            "Microsoft.EntityFrameworkCore.Database.Command": "Warning"}},
    "AllowedHosts": "*",
    "ConnectionStrings": {        "DefaultConnection":
"Host=db;Username=user;Password=password;Database=task_managment_system;"},
    "JWT": {        "Key": "G6ZjGyuKnKabrdrPf27oj7RQHWl2kxl6-
G6ZjGyuKnKabrdrPf27oj7RQHWl2kxl6",
        "AccessTokenExpiresInMinutes": 15,
        "RefreshTokenExpiresInDays": 30,
        "Issuer": "https://localhost:8001",
        "ClientUrl": "https://localhost:3000"},
    "Email": {        "Host": "smtp.gmail.com",
        "Port": 587,
        "Password": "",
        "SenderName": "TMS",
        "SenderEmail": "minecraftytor@gmail.com",
        "TokenExpireHours": 12,
        "ConfirmEmailPath": "confirm-email",
        "ResetPasswordPath": "reset-password"},
    "UploadDirectory": "Uploads"}
```

frontend/src/features/kanban/components/Board/Board.tsx

```
import React, { useEffect, useRef, useState } from "react";
import { Board } from "../../types";
```

```
import { useCreateColumnMutation, useMoveColumnMutation, useMoveCardMutation,
useCreateCardMutation } from "../../api";
import { monitorCardDrop, monitorColumnDrop } from "./dropHandlers";
import { setupAutoScroll } from "./scrollHelpers";
import { bindAll } from "bind-event-listener";
import { BoardHeader, ColumnComponent } from "../../components";
import { CleanupFn } from "@atlaskit/pragmatic-drag-and-drop/dist/types/entry-
point/types";
type FilterOption = "all" | "hasComments" | "hasAssignees";
type SortOption = "none" | "name" | "comments";
interface BoardProps {  board: Board;
  boardId: number;
  searchQuery: string;}
export const KanbanBoard: React.FC<BoardProps> = ({ board, boardId, searchQuery
}) => {  const [sortBy, setSortBy] = useState<SortOption>("none");
  const [filterBy, setFilterBy] = useState<FilterOption>("all");
  const [selectedLabelId, setSelectedLabelId] = useState<number | null>(null);
  const [filteredColumns, setFilteredColumns] = useState(board.columns);
  const scrollableRef = useRef<HTMLDivElement | null>(null);
  const [moveCard] = useMoveCardMutation();
  const [createCard] = useCreateCardMutation();
  const [createColumn] = useCreateColumnMutation();
  const [moveColumn] = useMoveColumnMutation();
 useEffect(() => {    let processedColumns = board.columns.map((col) => ({
...col, cards: [...col.cards] }));
    // Search filter
    if (searchQuery.trim()) {      const q = searchQuery.toLowerCase();
      processedColumns = processedColumns.map((col) => ({        ...col,
        cards: col.cards.filter((card) => card.title.toLowerCase().includes(q)
|| card.description?.toLowerCase().includes(q)),}));}
    // Standard filters
    if (filterBy !== "all") {      processedColumns = processedColumns.map((col)
=> ({        ...col,
        cards: col.cards.filter((card) =>
          filterBy === "hasComments"
            ? card.commentsCount > 0
            : filterBy === "hasAssignees"
            ? card.assignedUsers.length > 0
            : true),}));}
    // Label filter
    if (selectedLabelId) {      processedColumns = processedColumns.map((col) =>
({        ...col,
        cards: col.cards.filter((card) => card.labels?.some((label) => label.id
=== selectedLabelId)),}));}
    // Sorting
    if (sortBy !== "none") {      processedColumns = processedColumns.map((col)
=> ({        ...col,
        cards: [...col.cards].sort((a, b) =>
          sortBy === "name"
            ? a.title.localeCompare(b.title)
            : sortBy === "comments"
            ? b.commentsCount - a.commentsCount
            : 0),}));}
    setFilteredColumns(processedColumns);}, [searchQuery, board.columns, sortBy,
filterBy, selectedLabelId]);
  const handleAddCard = async (columnId: number) => {    try {      await
createCard({ boardId, columnId, title: "New Card" }).unwrap();} catch (error)
{      console.error("Failed to create card:", error);}};
  const handleAddColumn = async () => {    try {      await createColumn({
boardId, title: "New Column" }).unwrap();} catch (error)
{      console.error("Failed to create column:", error);}};
  return (    <>
```

```
      <BoardHeader
        board={board}
        filterBy={filterBy}
        sortBy={sortBy}
        selectedLabel={selectedLabelId}
        onFilterChange={setFilterBy}
        onSortChange={setSortBy}
        onLabelFilterChange={setSelectedLabelId}/>
      <div className="flex-grow relative">
        <div className="absolute inset-0">
          <div className="h-full flex flex-col">
            <div
              className="flex h-full flex-row gap-3 overflow-x-auto p-3
[scrollbar-width:thin] overflow-y-hidden"
              ref={scrollableRef}>{filteredColumns.map((column) =>
(               <ColumnComponent key={column.id} column={column}
boardId={boardId} onAddCard={handleAddCard} />))}
              <button
                onClick={handleAddColumn}
                className="flex-shrink-0 w-[280px] h-[50px] border-2 border-
dashed border-gray-300 rounded-lg
                hover:border-gray-400 transition-colors flex items-center
justify-center text-gray-500
                hover:text-gray-600">
                <span className="flex items-center gap-2">+ Add Column</span>
              </button>
            </div>
          </div>
        </div>
      </div>
    </>);};
```

frontend/src/features/kanban/components/Column/Column.tsx

```
import React, { useCallback, useEffect, useRef, useState, memo } from "react";
import { PlusIcon, Edit2, Archive, Trash } from "lucide-react";
import toast from "react-hot-toast";
import invariant from "tiny-invariant";
import { DropdownMenu } from "../DropdownMenu/DropdownMenu";
import { Modal } from "@shared";
import BriefCard from "../BriefCard/BriefCard";
import { Column as ColumnType } from "../../types";
import { useDeleteColumnMutation, useUpdateColumnMutation } from "../../api";
import { autoScrollForElements } from "@atlaskit/pragmatic-drag-and-drop-auto-
scroll/element";
import { unsafeOverflowAutoScrollForElements } from "@atlaskit/pragmatic-drag-
and-drop-auto-scroll/unsafe-overflow/element";
import { draggable, dropTargetForElements } from "@atlaskit/pragmatic-drag-and-
drop/element/adapter";
import { combine } from "@atlaskit/pragmatic-drag-and-drop/combine";
import { preserveOffsetOnSource } from "@atlaskit/pragmatic-drag-and-
drop/element/preserve-offset-on-source";
import { setCustomNativeDragPreview } from "@atlaskit/pragmatic-drag-and-
drop/element/set-custom-native-drag-preview";
import {  getColumnData,
  isColumnData,
  isCardData,
  isCardDropTargetData,
  isDraggingACard,
  isDraggingAColumn,
  CardData,} from "../../data";
import { isShallowEqual } from "../../utils";
import { BriefCardShadow } from "../../components";
```

```
import { DragLocationHistory } from "@atlaskit/pragmatic-drag-and-
drop/dist/types/internal-types";
// Column-level Drag State
type TColumnState =
  | { type: "idle" }
  | { type: "is-card-over"; isOverChildCard: boolean; dragging: DOMRect }
  | { type: "is-column-over" }
  | { type: "is-dragging" };
const stateStyles: Record<TColumnState["type"], string> = {  idle: "cursor-
grab",
  "is-card-over": "outline outline-2 outline-neutral-50",
  "is-column-over": "bg-gray-200",
  "is-dragging": "opacity-40",};
const idleState = { type: "idle" } satisfies TColumnState;
// Memoized card list to avoid rerenders
const CardList = memo(function CardList({ column, boardId }: { column:
ColumnType; boardId: number }) {  return column.cards.map((card) => <BriefCard
key={card.id} card={card} columnId={column.id} boardId={boardId} />);});
interface ColumnProps {  column: ColumnType;
  boardId: number;
  onAddCard: (columnId: number) => void;}
const ColumnComponent: React.FC<ColumnProps> = ({ column, boardId, onAddCard })
=> {  const [deleteColumn] = useDeleteColumnMutation();
  const [updateColumn] = useUpdateColumnMutation();
  const [isEditing, setIsEditing] = useState(false);
  const [title, setTitle] = useState(column.title);
  const [showDeleteModal, setShowDeleteModal] = useState(false); // Add modal
state
  const [colState, setColState] = useState<TColumnState>(idleState);
  const scrollableRef = useRef<HTMLDivElement | null>(null);
  const outerRef = useRef<HTMLDivElement>(null);
  const headerRef = useRef<HTMLDivElement>(null);
  const innerRef = useRef<HTMLDivElement>(null);
  // Handlers
  const handleUpdateTitle = useCallback(async () => {    if (!title.trim())
return toast.error("Title cannot be empty");
    try {      await updateColumn({ boardId, columnId: column.id, title:
title.trim() }).unwrap();
      setIsEditing(false);
      toast.success("Column updated");} catch {      setTitle(column.title);}},
[title, updateColumn, boardId, column.id, column.title]);
  // Updated delete handler to show modal instead of window.confirm
  const handleDeleteClick = useCallback(() => {    setShowDeleteModal(true);},
[]);
  const handleConfirmDelete = useCallback(async () => {    try {      await
deleteColumn({ boardId, columnId: column.id }).unwrap();
      toast.success("Column deleted");
      setShowDeleteModal(false);} catch (err: any) {      console.log(`Failed to
delete column ${err}`);
      toast.error("Failed to delete column");}}, [deleteColumn, boardId,
column.id]);
  const handleCancelDelete = useCallback(() => {    setShowDeleteModal(false);},
[]);
  return (    <>
      <div ref={outerRef} className="flex flex-col flex-shrink-0 w-72 rounded-lg
shadow-md h-fit">
        <div
          className={`flex h-full flex-col rounded-lg bg-gray-100 text-neutral-
50 ${stateStyles[colState.type]}`}
          ref={innerRef}
          data-block-board-panning>
```

```
          <div className={`flex h-full flex-col ${colState.type === "is-column-
over" ? "invisible" : ""}`}>
            <div
              className="bg-gray-100 rounded-t-lg border-y-2 border-gray-200 p-4
flex flex-row items-center justify-between"
              ref={headerRef}>{isEditing ? (                <input
                  type="text"
                  value={title}
                  onChange={(e) => setTitle(e.target.value)}
                  onBlur={handleUpdateTitle}
                  onKeyDown={(e) => e.key === "Enter" && handleUpdateTitle()}
                  className="px-2 py-1 border rounded w-full focus:outline-none
focus:border-teal-500 text-gray-900"
                  autoFocus/>) : (                <h3 className="font-semibold
text-base text-gray-800 flex-1">{column.title} <span className="text-gray-600
text-sm font-normal">({column.cards.length})</span>
                </h3>)}
            </div>
            <div
              ref={scrollableRef}
              className="flex flex-col p-2 [overflow-anchor:none] overflow-y-
auto bg-gray-50 min-h-[100px] max-h-[calc(100vh-17rem)] [scrollbar-width:thin]">
              <CardList boardId={boardId} column={column} />{colState.type ===
"is-card-over" && !colState.isOverChildCard && (                <BriefCardShadow
dragging={colState.dragging} />)}
            </div>
            <div className="flex flex-row gap-2 p-3 border-t border-gray-200 bg-
white rounded-b-lg">
              <button
                onClick={() => onAddCard(column.id)}
                className="flex flex-grow flex-row items-center justify-center
gap-1 py-2 text-sm text-gray-600 rounded-md hover:bg-gray-200 transition-colors
duration-200 font-medium">
                <PlusIcon size={16} /> Add Card
              </button>
            </div>
          </div>
        </div>
      </div
export default ColumnComponent;
```

frontend/src/features/kanban/components/Card/Card.tsx

```
import React, { useEffect } from "react";
import { CardHeader } from "./CardHeader";
import { CardDescription } from "./CardDescription";
import { CardAttachments } from "./CardAttachments";
import { CardComments } from "./CardComments";
import { CardSidebar } from "./CardSidebar";
import { CardLabels } from "./CardLabels";
import { Board, Card as CardType, Label } from "@features/kanban/types";
import { AppUser } from "@shared";
interface CardProps {  board: Board;
  card: CardType;
  currentUser: AppUser;
  isUploading: boolean;
  onPatch: (patches: any[]) => void;
  onAddAttachment: (file: File) => void;
  onRemoveAttachment: (attachmentId: number) => void;
  onDownloadAttachment: (id: number, fileName: string) => void;
  onAddComment: (content: string) => void;
  onUpdateComment: (commentId: number, content: string) => void;
  onDeleteComment: (commentId: number) => void;
  onAddAssignee: (userId: number) => void;
```

```
  onRemoveAssignee: (userId: number) => void;
  onCreateLabel: (title: string, color: string) => Promise<Label>;
  onUpdateLabel: (labelId: number, title: string, color: string) => void;
  onDeleteLabel: (labelId: number) => void;
  onAttachLabel: (labelId: number) => void;
  onDetachLabel: (labelId: number) => void;
  onClose: () => void;}
const UPLOADS_URL = import.meta.env.VITE_UPLOADS_PATH || "/uploads";
export const Card: React.FC<CardProps> = ({  board,
  card,
  currentUser,
  isUploading,
  onPatch,
  onAddAttachment,
  onRemoveAttachment,
  onDownloadAttachment,
  onAddComment,
  onUpdateComment,
  onDeleteComment,
  onAddAssignee,
  onRemoveAssignee,
  onCreateLabel,
  onUpdateLabel,
  onDeleteLabel,
  onAttachLabel,
  onDetachLabel,
  onClose,}) => {  const availableMembers = board.members.filter((m) =>
!card.assignedUsers.some((u) => u.id === m.id));
  useEffect(() => {    const handleEscape = (e: KeyboardEvent) => {      if
(e.key === "Escape") onClose();};
    document.addEventListener("keydown", handleEscape);
    return () => document.removeEventListener("keydown", handleEscape);},
[onClose]);
  const handleTitleChange = (newTitle: string) => {    onPatch([{ op: "replace",
path: "/title", value: newTitle }]);};
  const handleDescriptionChange = (newDesc: string) => {    onPatch([{ op:
"replace", path: "/description", value: newDesc }]);};
  const handleDueDateChange = (newDate: string) => {    onPatch([{ op:
"replace", path: "/dueDate", value: newDate || null }]);};
  const handleToogleCompleted = (isCompleted: boolean) => {    onPatch([{ op:
"replace", path: "/isCompleted", value: isCompleted }]);};
  const openInNewTab = (url: string): void => {    const newWindow =
window.open(url, "_blank", "noopener,noreferrer");
    if (newWindow) newWindow.opener = null;};
  const handleOverlayClick = (e: React.MouseEvent) => {    if (e.target ===
e.currentTarget) onClose();};
  return (    <div
      className="fixed inset-0 bg-black/50 flex items-center justify-center p-4
backdrop-blur-sm z-50"
      onClick={handleOverlayClick}>
      <div
        className="bg-white rounded-xl p-12 w-full max-w-4xl relative shadow-2xl
ring-1 ring-black/5 max-h-[90vh] overflow-y-auto"
        onClick={(e) => e.stopPropagation()}>
        <CardHeader
          title={card.title}
          onTitleChange={handleTitleChange}
          onClose={onClose}
          isCompleted={card.isCompleted}
          onToogleCompleted={handleToogleCompleted}/>
        <div className="grid grid-cols-1 md:grid-cols-3 gap-8 mt-2">
          <div className="md:col-span-2 space-y-4">
```

```
            <CardLabels
              card={card}
              board={board}
              onCreateLabel={onCreateLabel}
              onUpdateLabel={onUpdateLabel}
              onDeleteLabel={onDeleteLabel}
              onAttachLabel={onAttachLabel}
              onDetachLabel={onDetachLabel}/>
            <CardDescription description={card.description ?? ""}
onDescriptionChange={handleDescriptionChange} />
            <CardAttachments
              attachments={card.attachments}
              isUploading={isUploading}
              onAddAttachment={onAddAttachment}
              onRemoveAttachment={onRemoveAttachment}
              onOpenInNewTab={(url) => openInNewTab(`${UPLOADS_URL}/${url}`)}
              onDownload={onDownloadAttachment}/>
            <CardComments
              comments={card.comments}
              currentUser={currentUser}
              onAddComment={onAddComment}
              onUpdateComment={onUpdateComment}
              onDeleteComment={onDeleteComment}/>
          </div>
          <CardSidebar
            dueDate={card.dueDate || ""}
            assignedUsers={card.assignedUsers}
            availableMembers={availableMembers}
            onDueDateChange={handleDueDateChange}
            onAddAssignee={(id) => onAddAssignee(id)}
            onRemoveAssignee={(id) => onRemoveAssignee(id)}/>
        </div>
      </div>
    </div>);};
```

frontend/features/kanban/pages/UsersBoardsPage.tsx

```
import React, { useState, useMemo, useCallback } from "react";
import { Link } from "react-router-dom";
import { useGetBoardsQuery, useDeleteBoardMutation, useCreateBoardMutation,
useUpdateBoardMutation } from "../api";
import { getErrorMessage } from "@utils";
import { Header, BriefBoard, DropdownMenu } from "../components/";
import { Plus, Edit2, Trash2 } from "lucide-react";
import { toast } from "react-hot-toast";
import { Modal, LoadingScreen, ErrorScreen } from "@shared";
const UserBoardsPage: React.FC = () => {  const { data: boards = [], isLoading,
error } = useGetBoardsQuery();
  const [deleteBoard] = useDeleteBoardMutation();
  const [createBoard] = useCreateBoardMutation();
  const [updateBoard] = useUpdateBoardMutation();
  const [searchQuery, setSearchQuery] = useState("");
  const [confirmState, setConfirmState] = useState<{ open: boolean; boardId:
number | null }>({    open: false,
    boardId: null,});
  const [createModalOpen, setCreateModalOpen] = useState(false);
  const [newBoardTitle, setNewBoardTitle] = useState("");
  const [newBoardDescription, setNewBoardDescription] = useState("");
  const [editModalOpen, setEditModalOpen] = useState(false);
  const [editingBoardId, setEditingBoardId] = useState<number | null>(null);
  const [editBoardTitle, setEditBoardTitle] = useState("");
  const [editBoardDescription, setEditBoardDescription] = useState("");
  const filteredBoards = useMemo(() => {    const q =
searchQuery.trim().toLowerCase();
```

```
    if (!q) return boards;
    return boards.filter((b) => b.title.toLowerCase().includes(q) ||
(b.description && b.description.toLowerCase().includes(q)));}, [boards,
searchQuery]);
  const handleSearch = useCallback((q: string) => {    setSearchQuery(q);}, []);
  const confirmDelete = (boardId: number) => {    setConfirmState({ open: true,
boardId });};
  const handleConfirmDelete = async () => {    const id = confirmState.boardId;
    if (!id) return;
    try {      await deleteBoard(id).unwrap();
      toast.success("Board deleted");} catch (err) {      console.error("Failed
to delete board:", err);} finally {      setConfirmState({ open: false, boardId:
null });}};
  const handleConfirmCreate = async () => {    const title =
newBoardTitle.trim();
    if (!title) return;
    try {      await createBoard({ title, description: newBoardDescription
}).unwrap();
      toast.success("Board created");
      setNewBoardTitle("");
      setNewBoardDescription("");
      setCreateModalOpen(false);} catch (err) {      console.error("Failed to
create board:", err);}};
  const openEditModal = (boardId: number) => {    const board = boards.find((b)
=> b.id === boardId);
    if (!board) return;
    setEditingBoardId(boardId);
    setEditBoardTitle(board.title);
    setEditBoardDescription(board.description || "");
    setEditModalOpen(true);};
  const handleConfirmEdit = async () => {    if (!editingBoardId) return;
    const title = editBoardTitle.trim();
    try {      await updateBoard({ id: editingBoardId, title, description:
editBoardDescription }).unwrap();
      toast.success("Board updated");
      setEditModalOpen(false);
      setEditingBoardId(null);} catch (err) {      console.error("Failed to
update board:", err);}};
  if (isLoading) return <LoadingScreen />;
  if (error) return <ErrorScreen title="Boards Loading Failed"
message={getErrorMessage(error)} type="warning" />;
  return (    <div className="min-h-screen bg-gray-50">
      <Header onSearch={handleSearch} />
      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-8">
        <div className="flex justify-between items-center mb-8">
          <div>
            <h1 className="text-3xl font-bold text-gray-900">Your Boards</h1>
            <p className="text-gray-600 mt-1">Manage and organize your
projects</p>
          </div>
          <button
            onClick={() => setCreateModalOpen(true)}
            className="flex items-center bg-teal-500 hover:bg-teal-600 text-
white px-4 py-2 rounded-md transition-colors">
            <Plus className="w-5 h-5 mr-2" />
            Create Board
          </button>
        </div>
        <div>{filteredBoards.length > 0 ? (           <div className="grid
grid-cols-1 md:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 gap-
6">{filteredBoards.map((board) => (               <div key={board.id}
className="relative">
```

```
                  <Link to={`/boards/${board.id}`} className="block">
                    <BriefBoard board={board} />
                  </Link>
                  <div className="absolute top-2 right-2">
                    <DropdownMenu
                      size="sm"
                      items={[{                          label: "Edit",
                          icon: <Edit2 size={16} />,
                          onClick: () =>
openEditModal(board.id),},{                          label: "Delete",
                          icon: <Trash2 size={16} />,
                          variant: "danger",
                          onClick: () => confirmDelete(board.id),},
                      ]}/>
                  </div>
                </div>))}
            </div>) : (            <p className="text-gray-500">{searchQuery ?
`No boards match “${searchQuery}”.` : "You have no boards. Add one to get
started."}
            </p>)}
        </div>
      </div>
    </div>);};
export default UserBoardsPage;
```

docker-compose.yaml

```
name: tms
services:
  api:
    image: tms_api
    container_name: tms_api
    build:
      context: ./API
      dockerfile: .docker/api.Dockerfile
    ports:
      - 8000:80
      - 8001:443
    depends_on:
      db:
        condition: service_healthy
      migrate:
        condition: service_completed_successfully
    restart: always
    env_file:
      - ./API/.config/api.env
    volumes:
      - ~/.aspnet/https:/https:ro
      - ./API:/source
      - /source/obj/ # <- directory won't be mounted
      - /source/bin/
    logging:
      driver: "json-file"
      options:
        max-size: "200k"
        max-file: "10"
  migrate:
    image: tms_migrate
    container_name: tms_migrate
    build:
      context: ./API
      dockerfile: .docker/migrate.Dockerfile
    depends_on:
      db:
```

```
        condition: service_healthy
    env_file:
      - ./API/.config/api.env
    volumes:
- ./API:/source
      - /source/obj/
      - /source/bin/
    restart: no
  db:
    image: tms_pg_db
    container_name: tms_pg_db
    restart: always
    build:
      context: ./API
      dockerfile: .docker/db.Dockerfile
    volumes:
      - db-data:/var/lib/postgresql/data:rw
    command: "postgres -c max_connections=150 -c shared_buffers=512MB -c
effective_cache_size=1536MB -c maintenance_work_mem=128MB -c
checkpoint_completion_target=0.9 -c wal_buffers=16MB -c
default_statistics_target=100 -c random_page_cost=1.1 -c
effective_io_concurrency=200 -c work_mem=3495kB -c min_wal_size=1GB -c
max_wal_size=4GB -c max_worker_processes=2 -c max_parallel_workers_per_gather=1
-c max_parallel_workers=2 -c max_parallel_maintenance_workers=1"
    env_file:
      - API/.config/db.env
    ports:
      - 5432:5432
    healthcheck:
      test: [ "CMD-SHELL", "pg_isready -d $$POSTGRES_DB -U $$POSTGRES_USER" ]
      interval: 10s
      timeout: 5s
      retries: 5
    logging:
      driver: "json-file"
      options:
        max-size: "200k"
        max-file: "10"
  front:
    image: tms_front
    container_name: tms_front
    build:
      context: ./frontend
      dockerfile: front.Dockerfile
    restart: always
    depends_on:
      - api
    ports:
      - 3000:3000
    env_file:
      - ./frontend/.env
    volumes:
      - ./frontend:/app
      - /app/node_modules
    logging:
      driver: "json-file"
      options:
        max-size: "200k"
        max-file: "10"
volumes:
db-data:
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: ***Розробка візуалізатора для організації роботи студента kanban-дошки, засобами ASP.NET та React***

Обсяг пояснювальної записки 62 аркуші:

11 таблиць;

44 рисунки;

1 додаток.

Дата завершення роботи: *12 червня 2025р*.

Підпис студента- _____*Відоняк Ю.Р.*