МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Дискретного аналізу та інтелектуальних систем
(повна назва кафедри)

Магістерська робота

РОЗРОБКА ЗАСТОСУНКУ ДЛЯ КЕРУВАННЯ ПРОЕКТАМИ

Виконав: студент групи ПМіМ-23с
спеціальності
122 «Комп'ютерні науки»
(шифр і назва спеціальності)

Дяків Ю. В.
(підпис) (прізвище та ініціали)

Керівник Олійник Р. М.
(підпис) (прізвище та ініціали)

Рецензент Гошко Б. М.
(підпис) (прізвище та ініціали)

Львів – 2022

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет Прикладної математики та інформатики

Кафедра Дискретного аналізу та інтелектуальних систем

Спеціальність 122 «Комп'ютерні науки»
(шифр і назва)

«ЗАТВЕРДЖУЮ»

Завідувач кафедри Притула М. М.

"31" серпня 2022 року

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Ляківу Юрію Володимировичу
(прізвище, ім'я по батькові)

1 .Тема роботи "Розробка застосунку для керування проектами"

керівник роботи Олійник Роман Миколайович, кандидат фіз.-мат. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені Вченою радою факультету від **"13" вересня 2022 року №15**

2. Строк подання студентом роботи **12.12.2022р.**

3. Вихідні дані до роботи
Документація та книги по JavaScript, TypeScript, ReactJS, Redux, PostgreSQL, Node.js, Fastify, Knex.js, Objection.js. Статті та інтернет-ресурси по REST архітектурі та патерні Repository. Середовище розробки - Visual Studio Code

4. Зміст магістерської роботи (перелік питань, які потрібно розробити)
- Аналіз предметної галузі
- Використані технології
- Огляд веб застосунку

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
- Схема бази даних
- Демонстрація роботи програми

6. Консультанти розділів роботи

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

7. Дата видачі завдання **31 серпня 2022 р.**

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної галузі	01.09 - 07.09	Виконано
2	Пошук та вивчення необхідних технологій	08.09 - 14.09	Виконано
3	Огляд та вивчення необхідних патернів і архітектури	15.09 - 21.09	Виконано
4	Розробка дизайну для застосунку	22.09 - 29.09	Виконано
5	Розробка застосунку	30.09 - 06.11	Виконано
6	Тестування та виправлення помилок	07.11 - 13.11	Виконано
7	Оформлення магістерської роботи	14.11 - 30.11	Виконано

Студент _____ Дяків Ю. І
(підпис) (прізвище та ініціа

Керівник роботи _____ Олійник Р. М
(підпис) (прізвище та ініціа

РЕФЕРАТ

У даній роботі було створено застосунок для керування проектами. Його суть полягає в тому, що користувачі можуть планувати, візуалізувати, структурувати завдання та ефективно менеджити час. Застосунок можна використовувати як для великих команд на проектах, так і для особистих цілей та планів.

Ключовим завданням веб застосунку є можливість створення дошок для менеджменту проектів. Кожна дошка має колонки, створені користувачем, які можна редагувати та видаляти. Користувач має можливість додавати картки в колонки, редагувати, видаляти та переміщувати їх між колонками. Також можна додати нових членів в команди і призначити їм картки.

Магістерська робота складається зі вступу, п'ятьох розділів, висновків та списку використаних джерел. У вступі вказано загальну інформацію про менеджмент проектів, дошки для завдань, мету проекту та основні технології, які були використані. У чотирьох розділах є інформація про завдання роботи, Всесвітню мережу, архітектуру проект та використані технології. У п’ятому розділі описано базу даних та роботу застосунку з відповідними рисунками.

Обсяг роботи складає 40 сторінок.

ЗМІСТ

ВСТУП

Менеджмент є дуже важливою частиною сучасних проектів. Менеджмент проектів – це набір процесів, методів, навичок, знань та досвіду для досягнення специфічних цілей конкретного проекту [1]. Його головним завданням є керування роботою команди для вчасного виконання всіх завдань та цілей, залежного від узгоджених дедлайнів та бюджету. Ключовою відмінністю від простого менеджменту є те, що проекти мають кінцеві результати, які потрібно досягти та є обмежені у часі. А простий менеджмент є постійним процесом.

Для роботи над проектом потрібна команда, яка протягом визначеного часу фокусується та працює над його цілями. Ефективна командна робота є ключовим аспектом в успішних проектах. В основному менеджмент потребують проекти, котрі надають нові послуги на ринку, мають обмеження у часі, тобто початок та кінець, та можуть бути досить складними з точки зору роботи або кількості працівників.

Тому для ефективного менеджменту проектів потрібний широкий спектр навичок, зокрема технічних, звичайно навички управління командою та хороше розуміння бізнесу загалом.

Він включає такі основні моменти:

- Планування;
- Менеджмент часу;
- Менеджмент змін;
- Управління командою;
- Візуалізацію завдань;
- Структурування завдань;
- Оцінку ресурсів;
- Оцінку ризиків;
- Розробку плану проекту;
- Моніторинг прогресу та бюджету;
- Підтримка комунукації з замовниками;

Вкладання ресурсів та коштів у ефективний проджект менеджмент має ряд переваг, як-от:

- Успішне досягнення цілей;
- Вчасне завершення завдань;
- Краща продуктивность;
- Збільшення мотивації команди;
- Оптимізаця процесів;
- Висока імовірність досягнення бажаних цілей;
- Забезпечення ефективного використання ресурів;

Робота над проектом інколи є малоефективною, оскільки вона не організована, не розставлені правильні пріоритети, не організовані відповідні завдання та цілі. Для вирішення цих проблем доцільно використовувати дошки для завдань [2]. Якщо правильно та ефективно використовувати такі дошки, то можна швидко покращити роботу навіть на великих та складних проектах.

Майже кожна команда чи проджект менеджер може використовувати такі дошки. Зазвичай вони є кастомізованими, і тому їх можна налаштувати майже під будь-який проект. За статистикою, люди з більшою імовірністю зроблять збережене та розписане завдання. І це ще більш важливо для великих проектів та команд.

Зазвичай на проектах завдання ділять на кілька категорій, як-от:

- Ще не початі завдання;
- Завдання у процесі;
- Завдання, які на перевірці;
- Завершені завдання;

Для цього створюють окремі колонки під кожну категорію та призначають окремі завдання членам каманди.

Основні переваги таких дошок:

- Легке інтегрування каманди в проект та його процеси;
- Візуалізація завдань;
- Організація завдань;

- Легке розставлення пріоритетів;
- Легкий розподіл завдань;

Метою цієї магістерської роботи якраз було створення дошки для завдань. Цей застосунок допомагає легко та ефективно керувати проектами та завданнями.

Основним завданням застосунку є можливість створення дошок для менеджменту проектів. Кожна дошка має колонки, створені користувачем, які можна редагувати та видаляти. Користувач має можливість додавати картки в колонки, редагувати, видаляти та переміщувати їх між колонками. Також можна додати нових членів в команди і призначити їм картки.

Також додаток можна використовувати, як менеджер завдань для особистих цілей та планів.

На сьогодні існує безліч технологій для створення веб застосунків, даний веб застосунок написаний за допомогою сучасних та ефективних технологій. Для цього було використано мову програмування JavaScript, бібліотеку React, платформу Node.js, фреймворк Fastify та базу даних PostgreSQL.

РОЗДІЛ 1 ФОРМУЛЮВАННЯ ЗАВДАННЯ

1.1 Цільові користувачі

Для правильного визначення потреб користувачів та особливостей застосунку потрібно визначити його цільових користувачів. В основному додаток має використовуватися проджект менеджерами чи командою розробників на різних проектах. Однак його також можуть використовувати студенти для планування завдань в університеті, збереження дедлайнів чи відслідковування вже виконаних завдань. Іншими користувачами також можуть бути звичані люди для повсякденних задач та чеклістів.

Отже можна виділити такі три групи:

- Працівники на проектах;
- Студенти;
- Інші користувачі;

До потреб користувачів також можна виділити наступні пункти:

- Візуалізація та структурування завдань;
- Планування;
- Менеджмент часу;
- Збільшення продуктивності та мотивації;
- Успішне досягнення цілей;

1.2 Аналоги

Для кращого аналізу потреб користувачів потрібно виділити аналоги таких проектів. На ринку можна побачити такі 3 основні: Jira, Trello та Microsoft To Do.

Щодо проекту Jira, то там реалізований основний функціонал таких дошок. Сама дошка розрахована на великі проекти, які зазвичай використовують Agile методологію та Scrum підхід [3]. На ній можна ефективно розділити завдання на оркемі спрінти зі складною статистикою, графіками, фільтрацією та звітами. Також можна підключити різні додаткові сервіси, як-от: GitHub, Confluence і багато інших. Але така дошка є досить громіздка і не зовсім підходить для невеликих проектів, де мало людей в команді, або коли розробник взагалі один.

Наступний аналог – Trello. Цей сервіс використовує вже трохи інший підхід до керування проектів – Kanban. Він розрахований на менші проекти з меншою кількістю працівників [4]. Без вбудований Agile принципів, без спринтів тощо. До Trello також можна підключити додаткові сервіси, як-от: Google Drive, Slack, DropBox, Teams та інші.

Microsoft To Do – це вже не зовсім дошка, він розрахований для чек лістів та планінгу свого дня [5]. Загалом для менеджменту різних завдань, які можуть бути пов’язані з роботою, навчанням, повсякденним життям і так далі.

1.3 Основні можливості

Проаналізувавши цільових користувачів, їх потреби, аналоги на ринку та іншу інформацію, можна виділити наступні основні можливості, які подібні проекти мають реалізовувати.

Основною можливістю таких проектів є створення дошок для менеджменту проектів. До дошок можна додавати нових членів команди. Кожна дошка має колонки, створені користувачем, які можна редагувати та видаляти. Користувач має можливість додавати картки в колонки, редагувати, видаляти та переміщувати їх між колонками. Також можна призначити картки окремим членам команди.

Перелічені можливості можна відобразити за допомогою UML Use Case діаграми.

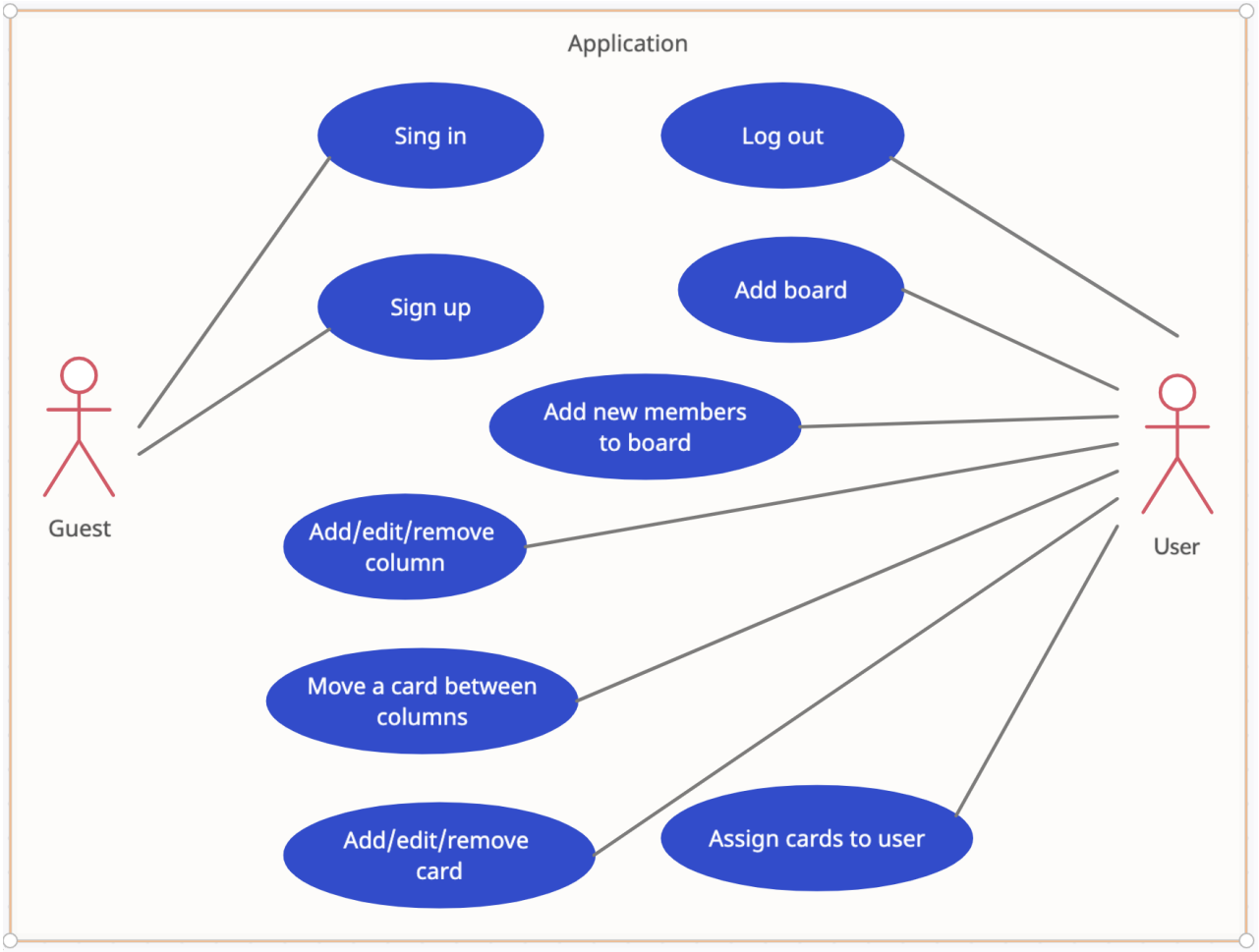


Рисунок 1.3.1 – UML Use Case діаграма

РОЗДІЛ 2 ВСЕСВІТНЯ МЕРЕЖА

2.1 Принцип роботи

World Wilde Web – інформаційна система, яка надає доступ до незліченної кількості документів через інтернет [6]. До цих документів чи інших ресурсів можна доступитися за допомогою браузерів з будь-якої частини світу, використовуючи Uniform Resource Locators (URL) посилання. Користувачі використовують різні пристрої для цього: комп’ютери, планшети, телефони та ін.

Основним і досі найбільш поширеним типом документів є веб-сторінки, які складаються з Hypertext Markup Language (HTML) розмітки та містять текст, зображення, графіку, аудіо, відео, тощо. Інформація передається по Hypertext Transfer Protocol (HTTP) протоколі.

Перегляд сторіки зазвичай починається з введення URL у браузер. Комунікація відбувається за принципом клієнт-сервер. Браузер надсилає запити на сервер та отримує необхідну йому інформацію. Основними та найбільш поширеними браузерами зараз є: Google Chrome, Safari, Firefox.

2.2 HTML

Hypertext Markup Language (HTML) – ствндартизована мова розмітки для документів, які відображаються у браузері для представлення інформації [7]. Ця мова розмітки описує струкуру документу, яка складаєтсья з окремих елементів: заголовків, абзаців, списків, посилань тощо. Самі елементи окреслені в теги, огорнуті в кутові дужки.

Також разом з HTML часто використовується Cascading Style Sheets (CSS), які використовуються для покращення вигляду та формування матеку сторінки. Та разом з JavaScript, який може змінювати анімації на сторінці та динамічно додавати туди нові елементи.

2.3 CSS

Cascading Style Sheets (CSS) – каскадні таблиці стилів, що використовуються для опису документів, які написані на мовах розмітки, таких як HTML чи XML. CSS одна з основних технологій у всесвітній мережі разом з HTML та JavaScript.

CSS розроблені для розділення контенту на сторінці, зміни її вигляду тобто макету, кольорів, шрифтів тощо. Розділення дозволяє побудувати більш зручну та доступну сторінку. Також правильне розділення стилів надає можливість полегшити розробку сторніки.

2.4 URL

Uniform Resource Locator (URL) – писилання на веб-ресурс у всесвітній мережі, часто його просто називають веб-адресою. Найчастіше ці посилання використовують для доступу до веб-сторінок, але також для передачі файлів (FTP), електронної пошти, доступу до БД та інших ресурсів.

Схема URL виглядає так:

scheme ":" ["//" authority] path ["?" query] ["#" fragment], де:

- **scheme** – зазвичай це протоколи, або інші схеми доступу до ресурсів (http, https, mailto, file ftp);
- **authority** – містить необов’язкову інформацію про користувача, хост та порт;
- **path** – додаткова інформація про шлях до ресурсу;
- **query** – атрибут - значення параметри, які починаються зі символу ? та розділені заком &;
- **fragment** – починається з символу # та вказує шлях на додатковий ресурс

2.5 HTTP

Hypertext Transfer Protocol (HTTP) – основний протокол передачі інформації у мережі [8]. Захищений варіант цього протоколу, який називається HTTPS, використовується у більше, ніж 80% веб-сайтів.

HTTP працює за принципом запит - відповідь у моделі клієнт-сервер, де клієнт відправляє HTTP запит до сервера. Сервер повертає відповідь зі статусом запиту та інформацією, яка може складатися з HTML файлів чи іншого контенту.

У цьому протоколі використовуються декілька схем для аутентифікації, наприклад basic access authentication та digest access authentication. Тому серевер перед поверненням даних ідентифікує та перевіряє запит.

Клієнт відправляє запит на сервер, який складається з таких частин:

- Рядка запиту, який містить метод запиту, URL та версію протоколу;
- Нуля або кількох заголовків запиту, що складається з назви поля та його значення;
- Необов'якового повідомлення;

HTTP має декілька визначених методів, які позначають відповідну дію, яку користувач хоче зробити за допомогою запиту. Кожен клієнт може використовувати будь-які методи, але сервер, залежно від налаштувань, може підтримувати не всі. Немає обмежень на кількість визначених методів, що дозволяє додавати нові методи до сервера, не ламаючи попередні.

Користувачі можуть використовувати наступні методи:

- **GET** – використовується, для отримання даних з сервера. Він повинен тільки отримати дані без інших побічних ефектів. Оскільки GET запит не повинен вносити ніякі зміни, його можна використовувати одразу з адресного рядка браузера. У GET запиті також можна передавати додаткові параметри у URL посиланні після символу ?. Кожен GET запит повинен повертати однакові дані, якщо вони не змінилися на сервері;
- **HEAD** – повертає ту ж саму інформацію, що і GET запит, але без даних у тілі відповіді. Використовується для отримання метаданих у заголовку

відповіді, бо не потрібно повертати всі дані з сервера. Також можна перевірити чи дані доступні на сервері, перевіривши статус та розмір фалу;

- **POST** – використовується для передачі даних на сервер, які поміщають у тіло запиту. Наприклад для збереження даних з форми, підписки на розсилку чи підтвердження якоїсь операції;
- **PUT** – використовується для оновлення об’єкта на сервері;
- **DELETE** – використовується для видалення об’єкта;
- **CONNECT** – використовується для втсановлення TCP/IP з’єднання з сервером. Часто для зв’язку через один чи декілька проксі серверів;
- **OPTIONS** – використовується для отримання HTTP методів, які підримує сервер. Часто використовується, щоб перевірити функціонал сервера;
- **TRACE** – використовується, щоб побачити зміни посередників;
- **PATCH** – використовується, щоб оновити якусь частину об’єкта на сервері;

Найчастіше використовують GET, POST, PATCH, PUT та DELETE методи.

Для того, щоб повернути статус відповіді використовуються наступні коди:

- 1xx (інформаційний) – запит отриманий та обробляється;
- 2xx (успішний) – запит успішно отриманий, оброблений та прийнятий;
- 3xx (перенаправлення) – необхідно вжити подальших запитів, щоб завершити запит;
- 4xx (помилка користувача) – запит містить помилку, або не може бути виконаний;
- 5xx (помилка сервера) – сервер не зміг виконати запит;

РОЗДІЛ 3 ВИКОРИСТАНІ ТЕХНОЛОГІЇ

3.1 JavaScript

JavaScript – динамічна мова програмування високого рівня [9]. ЇЇ можна віднести до кількох парадигм програмування: об’єктно орієнтованої, функціональної, імперативної та подійно-орієнтованої. JS є однією з основних технологій разом з HTML та CSS, які використовуються в браузерах. Приблизно 98% вебсайтів використовують цю мову для надання інтерактивності сторінкам.

Всі сучасні браузери на комп’ютерах, планшетах та телефонах використовують JavaScript інтерпретатор. Протягом останніх десяти років активно розробляється Node.js, який використовується поза браузерами. І це все робить JS найбільш поширеною мовою програмування.

Для підтримування версій використовують стандарти ECMA (European Computer Manufacturer’s Assocation), які називають ECMAScript. Останньою стабільною версією є ES2022.

Використання JavaScript:

- У браузерах (завантаження нового контенту для сторінки без її перезавантаження, додавання анімацій, стрімінг медіа контенту);
- Створення SPA (React, Vue.js, Angular);
- Створення серверів (Node.js, Deno);
- Створення десктопних застосунків (NW.js, Electron);
- Створення мобільних додатків (Cordova, React Native);
- Створення вбудованих систем;
- У PDF документах;
- Створення іншого програмного забезпечення;

3.2 TypeScript

TypeScript – безкоштовна мова програмування з відкрити кодом, розроблена та підтримується компанією Microsoft. Вона є додатковим синтаксисом для мови JavaScript. Основною перевагою використання TS є строга типізація, якої немає в мові JS [10]. Розробники позиціонують її для великих проектів, щоб знаходити та виправляти непередбачувані помилки ще на етапі компілювання.

Оскільки TS є додадковим синтаксимом для JavaScript, то всі програми, написані на JS, є валідними для цієї мови. Це робить перехід на неї досить зручним і безболісним для більшості старих проектів. TypeScript транскомпілюється в JavaScript, таким чином на виході отримується еквівалентний код на JavaScript.

TS можна використовувати у будь-яких середовищах, де можливе використання JavaScript. Тому в сучасних проектах досить часно він використовуєтьс як для клієнтської частини, так і для серверної.

TypeScript розширює мову JS та надає нові можливості, як от:

- Явне вказування типів;
- Вивід типів, де компілятор може автоматично визначити тип змінної на основі введеного значення;
- Видалення типів;
- Інтерфейси;
- Перелічувані типи даних;
- Узагальнення;
- Простір назв;
- Кортежі;

Також розширення отримали наступні можливості JS:

- Класи;
- Модулі;
- Анонімні функції;
- Необов’язкові параметри та параметри за замовчуванням;

3.3 ReactJS

React – JavaScript бібліотека з відкритим кодом для створення користувацьких інтерфейсів або UI компонентів [11]. Розробляється і підтримується компанією Facebook, а також частково індивідуальними розробниками та іншими компаніями.

React використовується як для розробки односторінкових застосунків, так і для мобільних додатків. Він дозволяє створювати великі та важкі користувацькі інтерфейси, які складаються з маленьких, ізольованих частинок коду під назвою “компоненти”.

3.4 Redux

Redux – JavaScript бібліотека та патерн для зберігання, оновлення та підтримування стану додатків, часто використовується у різних JS фрейморках, як Angualr, React, View.js. Взаємодія зі станом відбувається за допомогою подій, які називаються “actions”.

Redux використовується, як централізоване сховище даних, які використовуються по всьому додатку [12]. Redux є окремою бібліотекою, але також часто використовується в деяких інших бібліотеках, як React-Redux та Redux Toolkit.

Цю бібліотеку доцільно використовувати у таких випадках:

- Є багато даних, які потрібні у різних частинах застосунку;
- Дані часто оновлюються;
- Логіка оновлення даних є досить складною;
- Проект є середнім чи великим та складним;

Переваги Redux:

- Допомагає зручно зберігати та оновлювати дані в застосунку;
- Робить тестування зручним, оскільки допомагає легко побачити коли, де і як змінилися дані;

3.5 PostgreSQL

PostgreSQL – безкоштовна об’єктно-реляційна система керування базами даних [13]. Спочатку мала назву POSTGRES, але потім її перейменували на PostgreSQL, щоб підкреслити її підтримку SQL.

Особливостями PostgreSQL є транзакції, які гарантуються ACID властивостями (атомарність, узгодженість, ізольованість, довговічність), тригери, збережені процедури. Ця СКБД розроблена, щоб обробляти різні типи навантаження, від малих до великих об’ємних проектів, з багатьма користувачами.

3.6 Node.js

Node.js – JavaScript платформа, заснована на рушію V8, який транслює JavaScript у машинний код [14]. Вона надає можливість використовувати JavaScript не тільки у веб-браузерах, розробляючи клієнтську частину, а й на стороні сервера.

За допомогою Node.js можна розробляти програми для читання і запису файлів, виконувати дочірні процеси, передавати дані через мережу.

Вагомою відмінністю Node.js від інших мов програмування є те, що вона використовує паралелізм з одним потоком, який базується на подіях.

3.7 Fastify

Fastify – це веб-фреймворк, який орієнтований на покращення роботи розробника з найменшими витратами та потужною архітектурою [15]. Він є одинм із найшвидших веб-фреймворків для Node.js.

Основні особливості та принципи, на яких побудовано Fastify:

- Швидкодія, залежно від складності коду він може обслуговувати до 30 тисяч запитів на секунду;
- Легке розширення за допомогою хуків, плагінів і декораторів;

- Зручність у щоденному використанні, не жертвуючи продуктивністю та безпекою;
- Підтримка TypeScript;

3.8 Knex.js

Knex.js – це констуктор SQL запитів для PostgreSQL, CockroachDB, MSSQL, MySQL, MariaDB, SQLite3, Better-SQLite3, Oracle, and Amazon Redshift [16]. Він дозволяє легко доступитися до БД та взаємодіяти з нею.

3.9 Objection.js

Objection.js – це ORM та потужний конструктор реляційних запитів для Node.js, який максимально спрощує використання SQL та базового механізму баз даних [17]. Також він надає набір інструментів для роботи з відношеннями.

Objection.js побудований на основі конструктора SQL запитів Knex.js та підтримує всі бази даних, які підтримує Knex.js. Його можливості:

- Простий декларативний спосіб визначення моделей та зв’язків між ними;
- Проста взаємодія з БД;
- Простота у використанні транзакцій;
- Офіційна підтримка TypeScript;
- Зберігання складних документів у вигляді окремих рядків;

РОЗДІЛ 4 АРХІТЕКТУРА ПРОЕКТУ

4.1 REST

Representational state transfer (REST) – стиль архітектури програмного забезпечення, який описує інтерфейс між фізично розділеними компонентами, часто через інтернет у клієнт-серверній архітектурі [18]. В загальному REST описує взаємодію машини з машиною.

REST описує чотири основні обмеження для інтерфейсу:

- Ідентифікація ресурсів;
- Маніпулювання ресурсами;
- Самоописові повідомлення;
- Hypermedia as the Engine of Application State (HATEOAS);

У веб-розробці це дозволяє динамічно відображати контент. REST підхід досить широко використовується у розробці програмного забезпечення. Оскільки використовується, як набір настанов для написання надійних API. Якщо ми говоримо про веб API, то він базується на HTTP методах, які використовуються для доступу до даних. Відподь до запитів часто формується у JSON або XML форматі.

4.2 Трирівнева архітектура

Трирівнева архітектура – це клієнт-серверна архітекутра, де фунцкії відображення, обробки та збереження даних фізично розділяються. Ця архітектура є однією з найбільш поширених серед багаторівневих архітектур. Отже арітектуру поділяють на такі три рівні:

- Рівень відображення;
- Рівень бізнес логіки;
- Рівень обробки даних;

В цьому застосунку я розділив архітектуру на 3 рівні, використовуючи у рівні відображення front-end сервер, написаний на мові JavaScript, та бібліотеку React. Для рівня бізнес логіки я використав back-end сервер, написаний на Node.js, та Fastify фрейморк. Та третій рівень, де використана база даних PostgreSQL. Для доступу до бази даних з 2 рівня використовуються бібліотеки Knex.js та Objection.js.

4.3 Front-end частина

На front-end частині проекту використовується React бібліотека. Для створення компонетів я використовував функціональний підхід. Також використав модульні стилі у цих компонентах.

```
const Card = ({ card }) => {
    const [showAssigneeSelect, setShowAssigneeSelect] = useState(false);
    const dispatch = useDispatch();

    const [{ isDragging }, drag] = useDrag(() => ({
        type: 'card',
        item: card,
        collect: (monitor) => ({
            isDragging: !!monitor.isDragging()
        })
    }))

    const { currentBoard } = useSelector(({ boards }) => ({
        currentBoard: boards.currentBoard
    }));

    const handleCardTextChange = e => dispatch(CardsActionCreator.updateCard(card.id, { title: e.target.value }));
    const handleCardTitleChange = e => dispatch(CardsActionCreator.updateCard(card.id, { text: e.target.value }));
    const handleRemoveCard = () => dispatch(CardsActionCreator.removeCard(card.id, card.columnId));

    const handleAssigneeChange = assigneeId => () => {
        setShowAssigneeSelect();
        dispatch(CardsActionCreator.updateCard(card.id, { assigneeId }));
    };

    const handleShowAssigneeSelect = () => setShowAssigneeSelect(!showAssigneeSelect);

    return (
        <div ref={drag} className={clsx(styles.card, isDragging && styles.dragOpacity)}>
            <div className={styles.header}>
                <input
                    type='text'
                    placeholder='Title'
                    className={styles.title}
                    onBlur={handleCardTextChange}
                    defaultValue={card.title}
                />
```

Рисунок 4.3.1, аркуш 1 – Card компонент

```
                <div className={styles.actions}>
                    <Button
                        label={<FontAwesomeIcon icon={faTrashAlt} />}
                        type='button'
                        round
                        onClick={handleRemoveCard}
                    />
                    <Button
                        label={card.assignee ? `${card.assignee.name[0]}${card.assignee.surname[0]}` : '+'}
                        color={card.assignee && 'gray-light'}
                        type='button'
                        round
                        onClick={handleShowAssigneeSelect}
                    />
                    {showAssigneeSelect &&
                        <div className={styles.memberSelect}>
                            {currentBoard.users.filter(member => member.id !== card.assignee?.id).map(member =>
                                <div
                                    key={member.id}
                                    className={styles.option}
                                    onClick={handleAssigneeChange(member.id)}
                                >
                                    {`${member.name} ${member.surname}`}
                                </div>
                            )}
                        </div>
                    }
                </div>
            </div>
            <textarea className={styles.text} defaultValue={card.text} onBlur={handleCardTitleChange} placeholder='Text' />
        </div>
    );
};
```

Рисунок 4.3.1, аркуш 2 – Card компонент

Для обробки даних використовуєть Redux Toolkit. У сторі створені Action Creators, які використовуються для відправлення, отримання та обробки даних у Reducers.

```
const { reducer, actions } = createSlice({
    name: ReducerName.CARDS,
    initialState,
    reducers: {}
});

const addCard = card => async (dispatch) => {
    try {
        const response = await cardApi.createCard(card);
        dispatch(ColumnsActionCreator.addCard(response));
    } catch (error) {
        if (error instanceof HttpError) {
            return notificationService.error(`Error ${error.status}`, error.messages);
        }
        throw error;
    }
};

const updateCard = (id, payload, oldColumnId) => async dispatch => {
    try {
        const response = await cardApi.updateCard(id, payload);
        dispatch(ColumnsActionCreator.updateCard({ card: response, oldColumnId }));
    } catch (error) {
        if (error instanceof HttpError) {
            return notificationService.error(`Error ${error.status}`, error.messages);
        }
        throw error;
    }
};

const removeCard = (id, columnId) => async dispatch => {
    try {
        await cardApi.removeCard(id);
        dispatch(ColumnsActionCreator.removeCard({ id, columnId }));
    } catch (error) {
        if (error instanceof HttpError) {
            return notificationService.error(`Error ${error.status}`, error.messages);
        }
        throw error;
    }
};

const CardsActionCreator = { ...actions, addCard, updateCard, removeCard };
```

Рисунок 4.3.2 – Cards Reducer

Для відправлення запитів на back-end частину проекту створені API.

```
class CardApi {
    #http;
    #apiPrefix;

    constructor({ http, apiPrefix }) {
        this.#http = http;
        this.#apiPrefix = apiPrefix;
    }

    updateCard(id, payload) {
        return this.#http.load(`${this.#apiPrefix}${ApiPath.CARDS}${CardsApiPath.ROOT}${id}`, {
            method: HttpMethod.PUT,
            contentType: ContentType.JSON,
            payload
        });
    }

    createCard(payload) {
        return this.#http.load(`${this.#apiPrefix}${ApiPath.CARDS}${CardsApiPath.ROOT}`, {
            method: HttpMethod.POST,
            contentType: ContentType.JSON,
            payload
        });
    }

    removeCard(id) {
        return this.#http.load(`${this.#apiPrefix}${ApiPath.CARDS}${CardsApiPath.ROOT}${id}`, {
            method: HttpMethod.DELETE,
        });
    }
}
```

Рисунок 4.3.3 – Cards API, що використовується у Cards Reducer

4.4 Back-end частина

На beck-end частині проекту я реалізував петерн Repository. Патерн репозиторій використовується для роділення бізнес логіки сервера та рівня бази даних. Репозиторій можна назвати контейнером, який зберігає логіку доступу до даних, яку він приховує від рівня бізнес логіки. Цей підхід досить зручний та ефективний, оскільки дозволяє бізнес логіці доступатися до даних, навіть не знаючи архітектури у репозиторії.

Розділення доступу до даних та бізнес логіки має досить багато переваг, як-от:

- Зосередження логіки доступу до даних в одному місці дозволяє легко підтримувати код;
- Бізнес логіку та логіку доступу до даних можна тестувати окремо;
- Зменшує повторення коду;

- Зменшує шанс отримання помилок у логіці;

У більшості випадків реалізують наступні методи для доступу до даних:

- Отримання всіх записів;
- Отримання запису за унікальним ідентифікатором;
- Створення нового запису;
- Оновлення запису;
- Видалення запису;

Щодо реалізації цього патрену у проекті, то створив API файли. Вони використовується для отримання запитів та передавання даних у рівень бізнес логіки.

```
const initCardApi: FastifyPluginAsync<Options> = async (fastify, opts) => {
  const { card: cardService } = opts.services;

  fastify.route({
    method: HttpMethod.POST, url: CardsApiPath.ROOT,
    async handler(req: FastifyRequest<{ Body: CardCreateRequestDto }>, rep: FastifyReply,) {
      const card = await cardService.create(req.body);

      return rep.send(card).status(HttpCode.CREATED);
    }
  });

  fastify.route({
    method: HttpMethod.PUT, url: CardsApiPath.$ID,
    async handler(req: FastifyRequest<{ Params: UpdateRequestParamsDto; Body: CardUpdateRequestDto; }>, rep: FastifyReply,) {
      const { id } = req.params;
      const group = await cardService.updateById(id, req.body);

      return rep.send(group).status(HttpCode.OK);
    },
  });

  fastify.route({
    method: HttpMethod.DELETE, url: CardsApiPath.$ID,
    async handler(req: FastifyRequest<{ Params: DeleteRequestParamsDto }>, rep: FastifyReply,) {
      const { id } = req.params;

      await cardService.delete(id);

      return rep.send(true).status(HttpCode.OK);
    },
  });
};
```

Рисунок 4.4.1 – API для карток

Далі дані передаються у рівень бізнес логіки, де відбуваються різні операції з ними та їх підготовка до передачі у репозиторій. Для цього створені сервіси. Приклад Cards sercive, наразі тут тільки 3 методи для оновлення створення та видалення даних.

```
class Card {
  #cardRepository: typeof cardRep;

  constructor({ cardRepository }: Constructor) {
    this.#cardRepository = cardRepository;
  }

  public async updateById(id: string, payload: CardUpdateRequestDto): Promise<CardResponseDto | null> {
    const card = await this.#cardRepository.update(id, payload);

    return card;
  }

  public async create({ text, title, columnId, assigneeId }: CardCreateRequestDto): Promise<CardResponseDto> {
    const card = CardEntity.createNew({ text, title, columnId, assigneeId });

    const createdCard = await this.#cardRepository.create(card);

    return createdCard;
  }

  public async delete(id: string): Promise<void> {
    await this.#cardRepository.delete(id);
  }
}
```

Рисунок 4.4.2 – Cards сервіс

Наступник крок - це передача даних у репозиторій для операцій з базою даних. Тобто для подальшого збереження, овновлення, видалення чи отримання даних. Для цього створені репозиторії.

```
class Card {
  #CardModel: typeof CardM;

  constructor({ CardModel }: Constructor) {
    this.#CardModel = CardModel;
  }

  async update(id: string, payload: CardUpdateRequestDto): Promise<CardEntity | null> {
    const card = await this.#CardModel.query().patchAndFetchById(id, payload).withGraphFetched('[assignee]');

    if (!card) {
      return null;
    }

    return card;
  }

  async create(card: CardEntity): Promise<CardM> {
    const { id, text, title, columnId, assigneeId } = card;

    return this.#CardModel.query().insert({ id, text, title, columnId, assigneeId });
  }

  public async delete(id: string): Promise<void> {
    await this.#CardModel.query().where({ id }).del();
  }
}
```

Рисунок 4.4.3 – Cards репозиторій

РОЗДІЛ 5 ОГЛЯД ЗАСТОСУНКУ

Схема бази даних застосунку містить настпупні таблиці: users, cards, columns, boards, users_boards.

У таблиці users зберігається вся необхідна іфнормація про користувачів. Оскільки користувачі можуть мати багато дошок, та дошки можуть мати багато членів команди, то для цього реалізоване відношення many-to-many з users та boards таблицями. Для цього зв’язку використовується проміжна таблиця users_boards. Також реалізовани зв’язок one-to-many між таблицями users та cards, оскільки користувачам можна призначити багато карток.

Дошки можуть мати багато колонок, тому реалізований зв’язок one-to-many між boards та columns таблицями. Такий же самий зв’язок також реалізовано між columns та cards таблицями, оскільки колонки можуть мати багато карток.

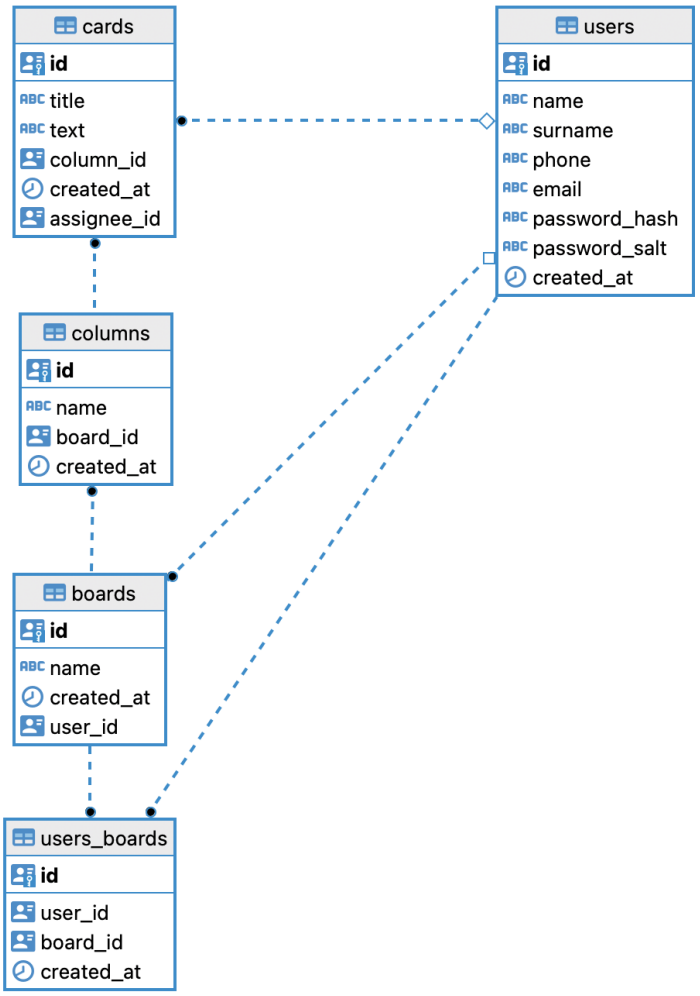


Рисунок 5.1 – Схема бази даних

Використовуючи сторінку для реєстрації, користувач може зареєструватися у системі. Усі поля є обов’язковими та мають відповідну валідацію, залежно від даних, які повині бути у конкретному полі.

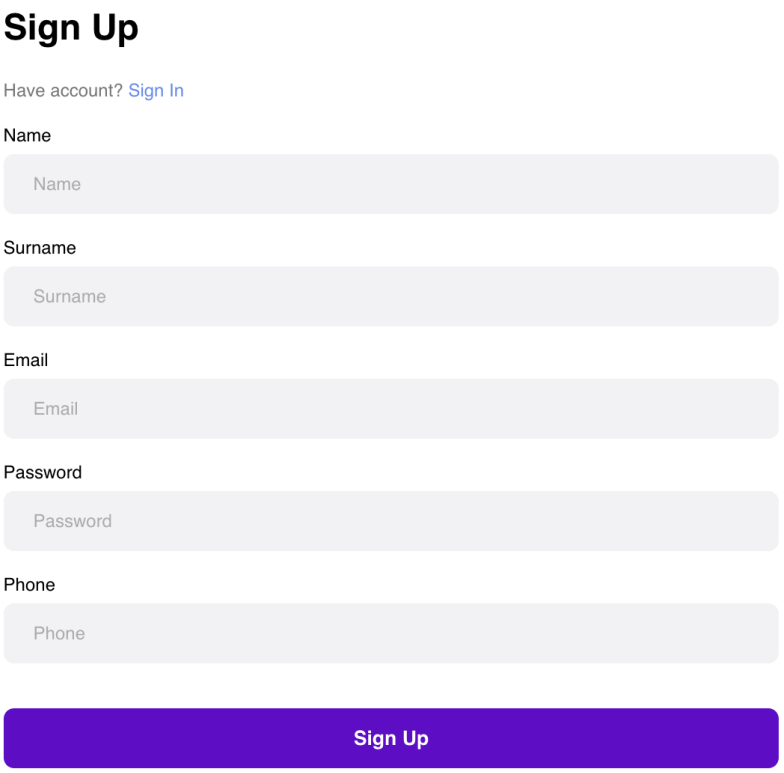


Рисунок 5.2 – Сторінка для реєстрації

За допомогою сторінки для входу, користувач може увійти в систему. Всі поля є обов’язковими та мають відповідну валідацію.

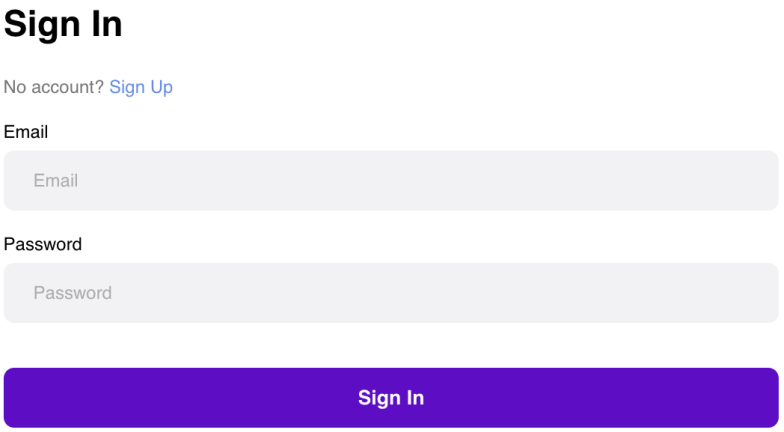


Рисунок 5.3 – Сторінка для входу

Після входу користувача у систему, застосунок перенаправить його на початкову сторінку, де він може додати нову дошку, вибрати з раніше доданих дошок чи вийти з застосунку.

Рисунок 5.4 – Початкова сторінка

Натиснувши на “Add Board” кнопку, користувач може додати нову дошку, використовуючи спливаюче вікно з відповідним полем для назви нової дошки.

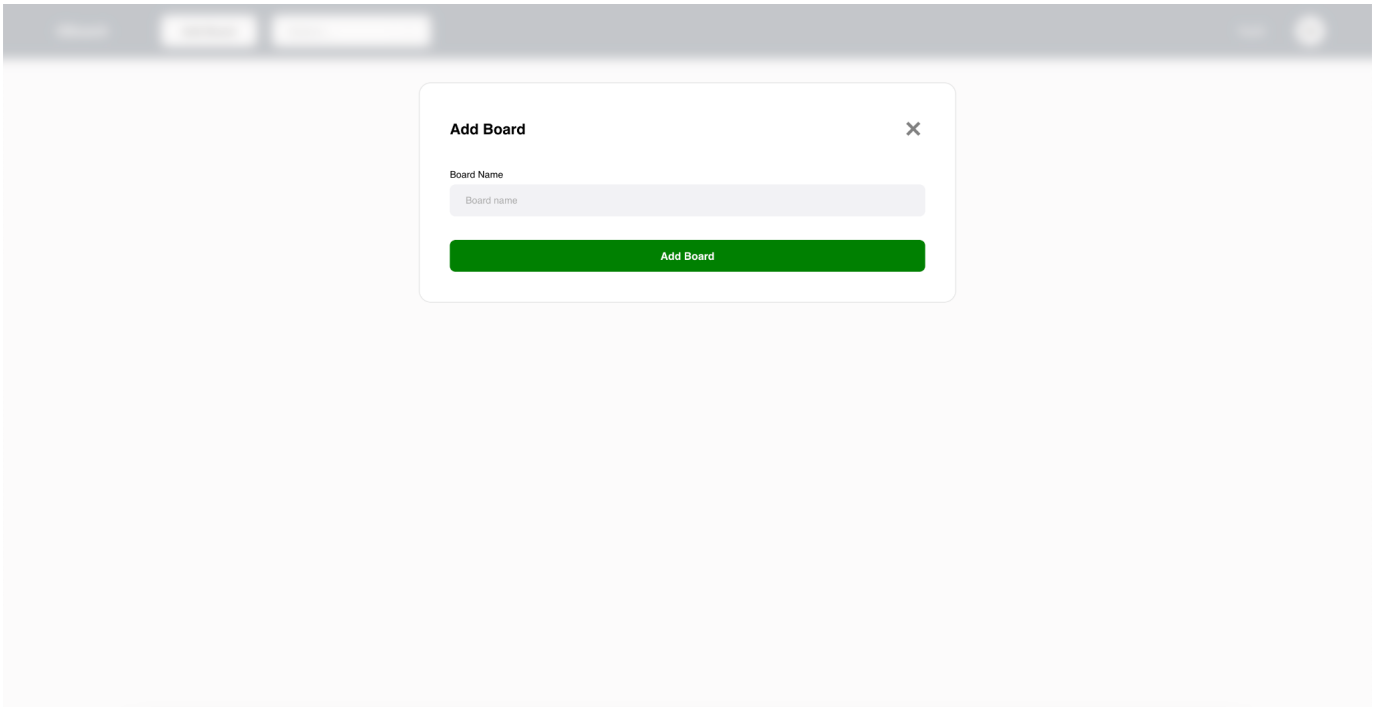


Рисунок 5.5 – Додавання дошки

Після входу в застосунок чи додавання нової дошки користувач може вибрати одну з доступних дошок.

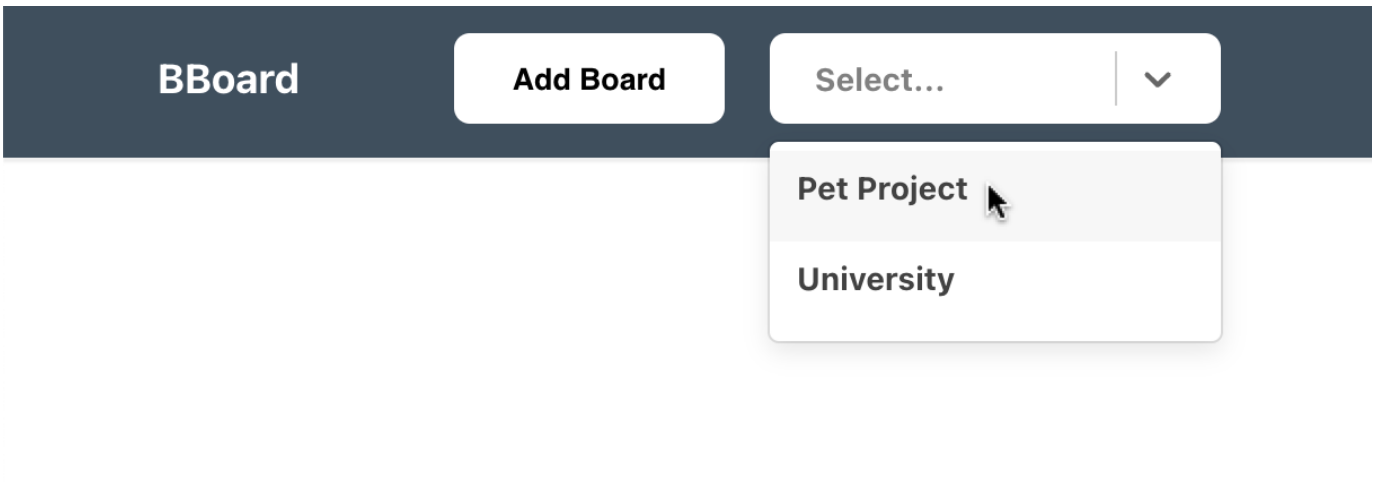


Рисунок 5.6 – Вибір дошки

Вибравши дошку, користувач може дадати нові колонки до неї, використовуючи відповідне поле для назви та кнопку “+”.

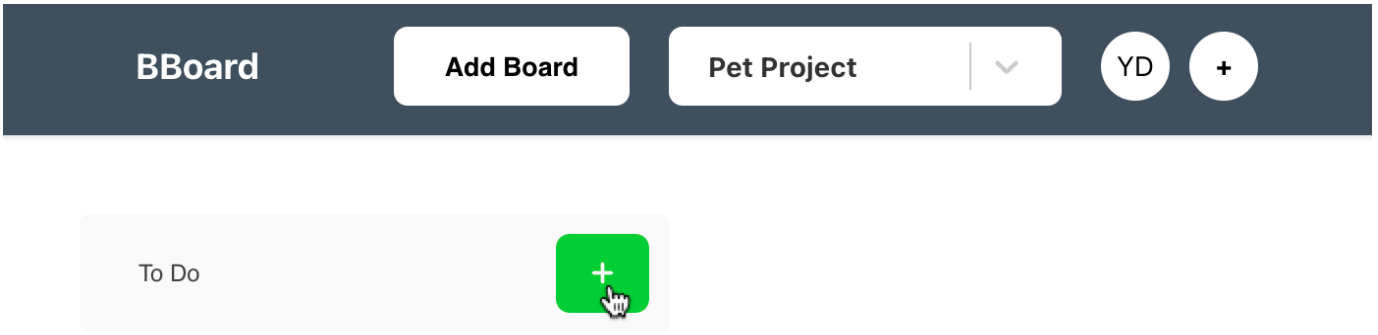


Рисунок 5.7 – Додавання колонки

Додані колонки можна редагувати, використовуючи поле, яке розміщене в заголовку колонки.

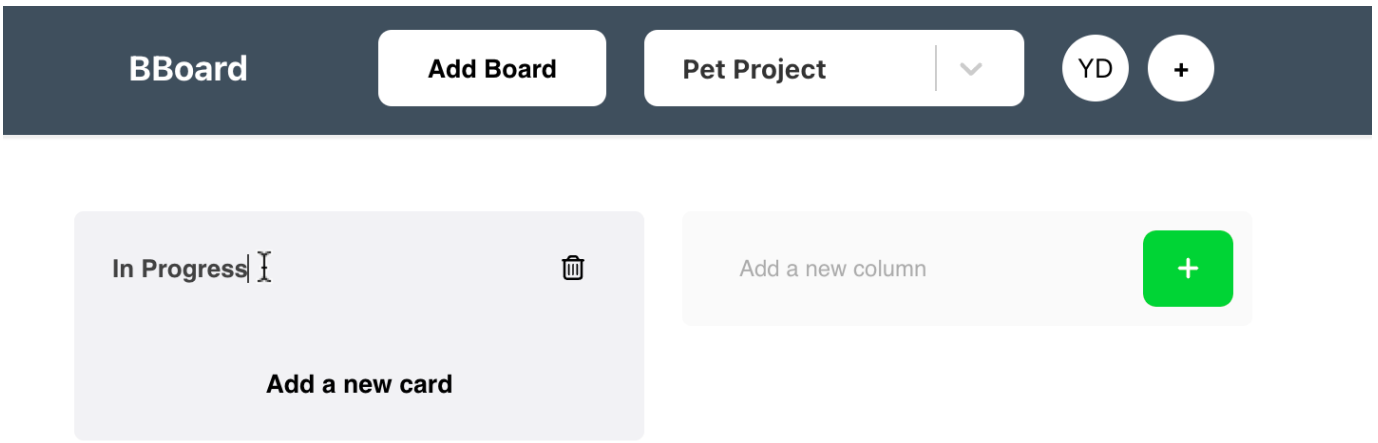
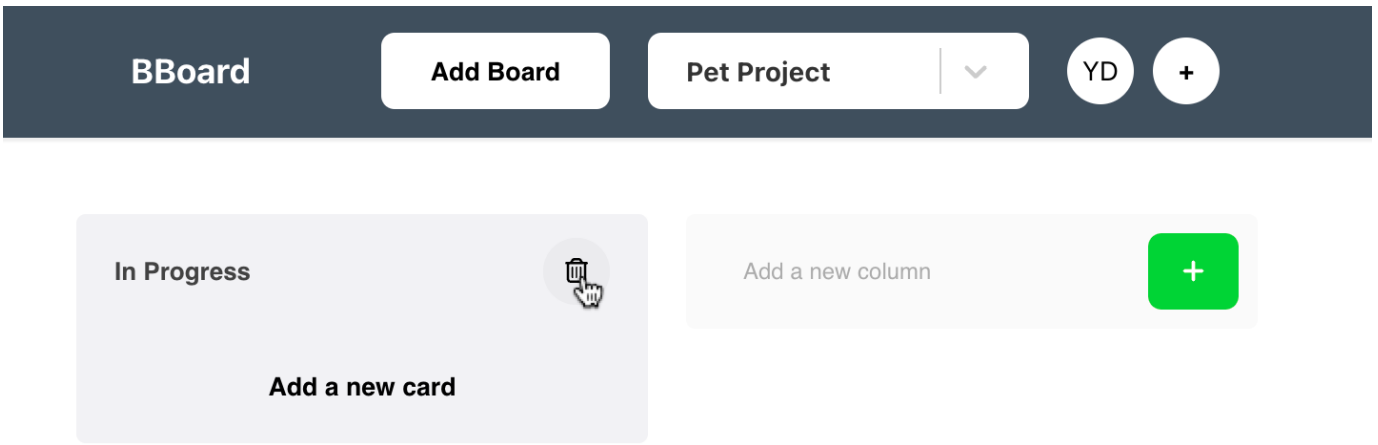


Рисунок 5.8 – Редагування колонки

Кожну колонку можна видалити, використовуючи відповідну кнопку в заголовку колонки.



Рисунко 5.9 – Видалення колонки

Використовуючи “Add a new card” кнопку, можна додати нову картку до колонки.

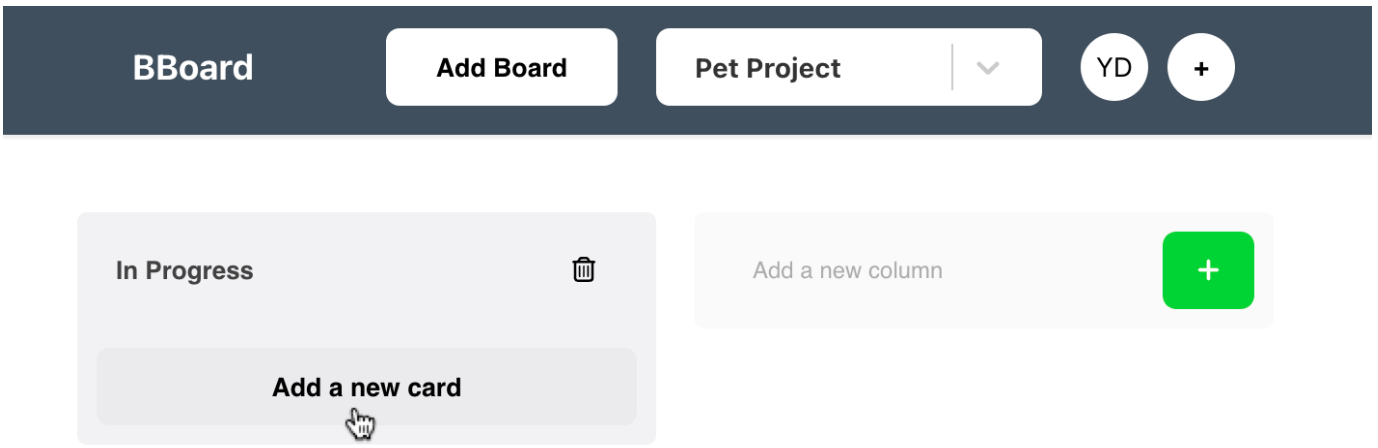


Рисунок 5.10 – Додавання картки

Кожну картку можна редагувати, змінюючи її опис чи заголовок.

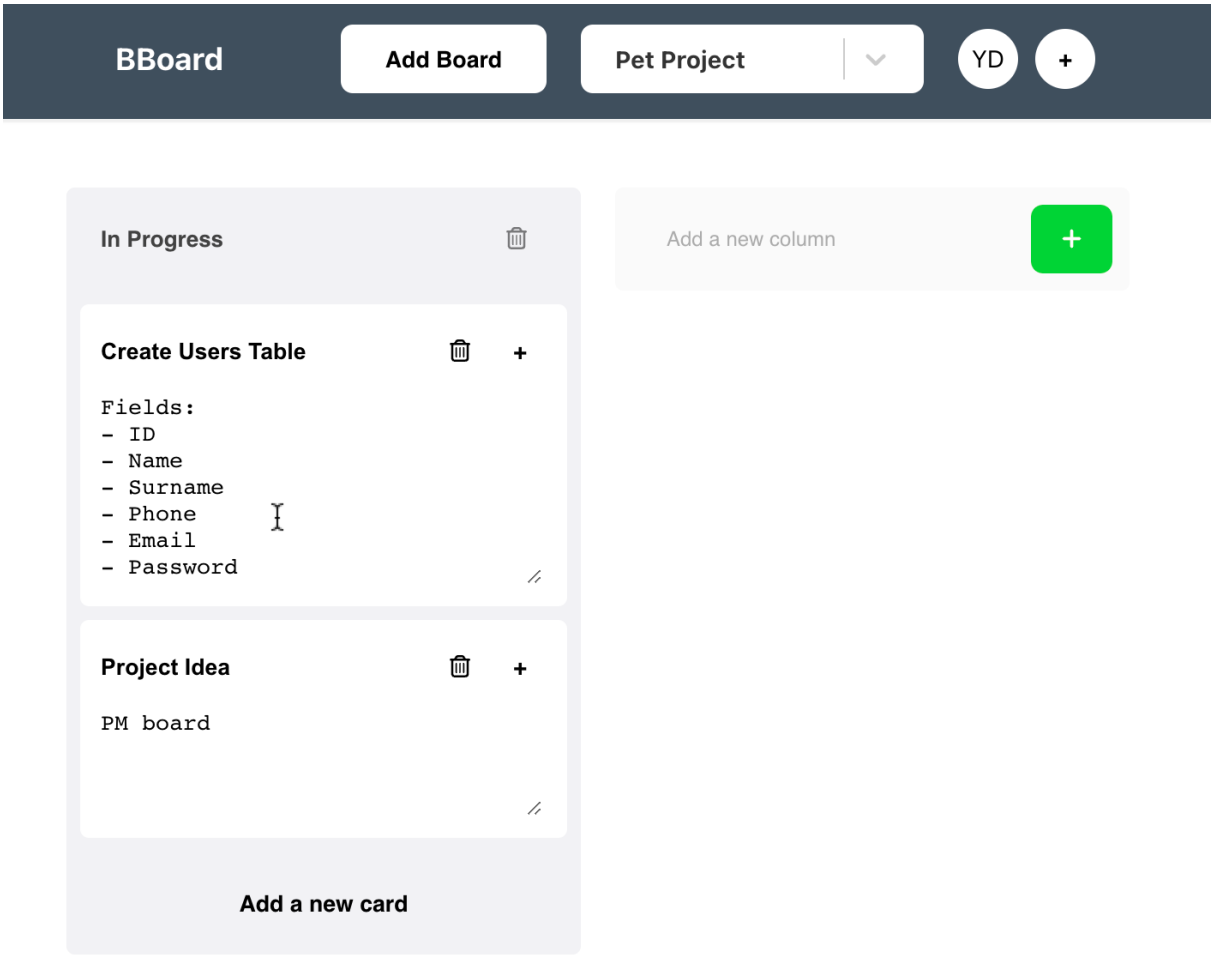


Рисунок 5.11 – Редагування картки

Картки можна видаляти, використовуючи відповіднку кнопку у заголовку картки.

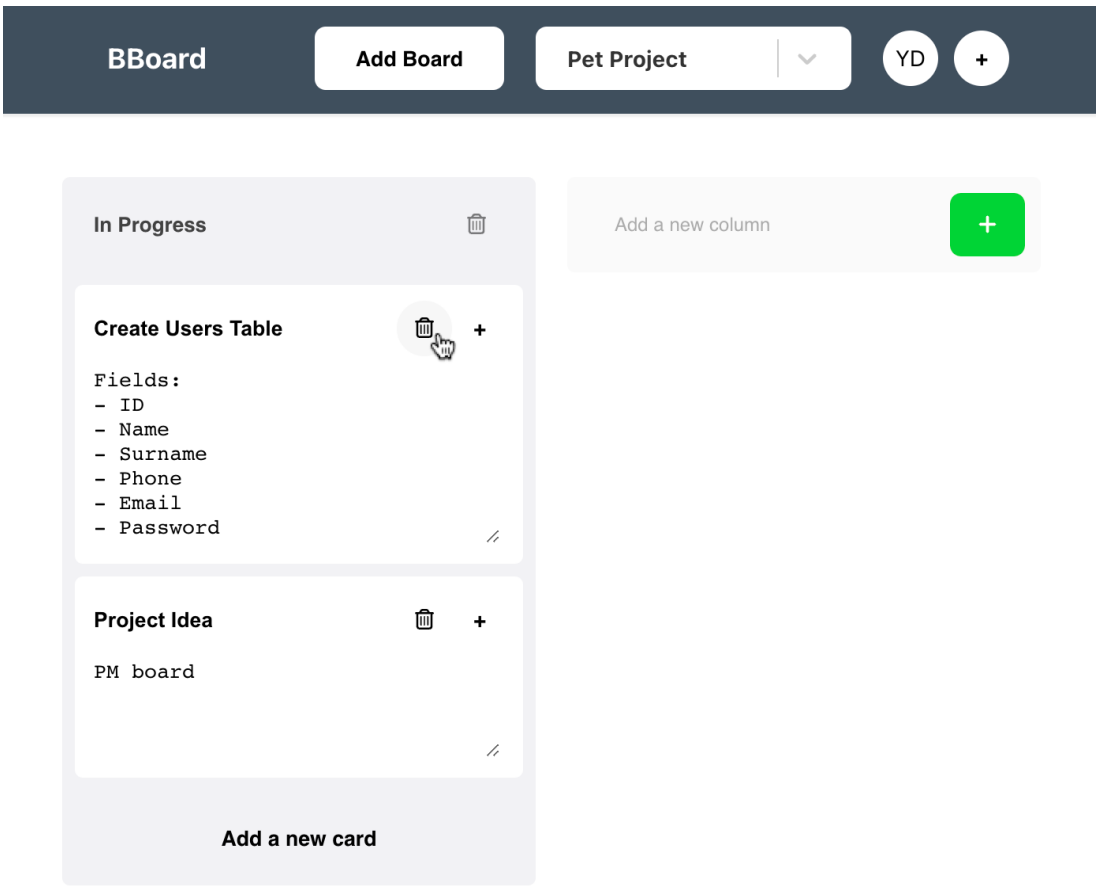


Рисунок 5.12 – Видалення картки

Натиснувши на кнопку “+”, яка розміщена біля власника дошки у верхній панелі застосунку, можна відкрити спливаюче вікно, щоб додати до дошки нових членів команди.

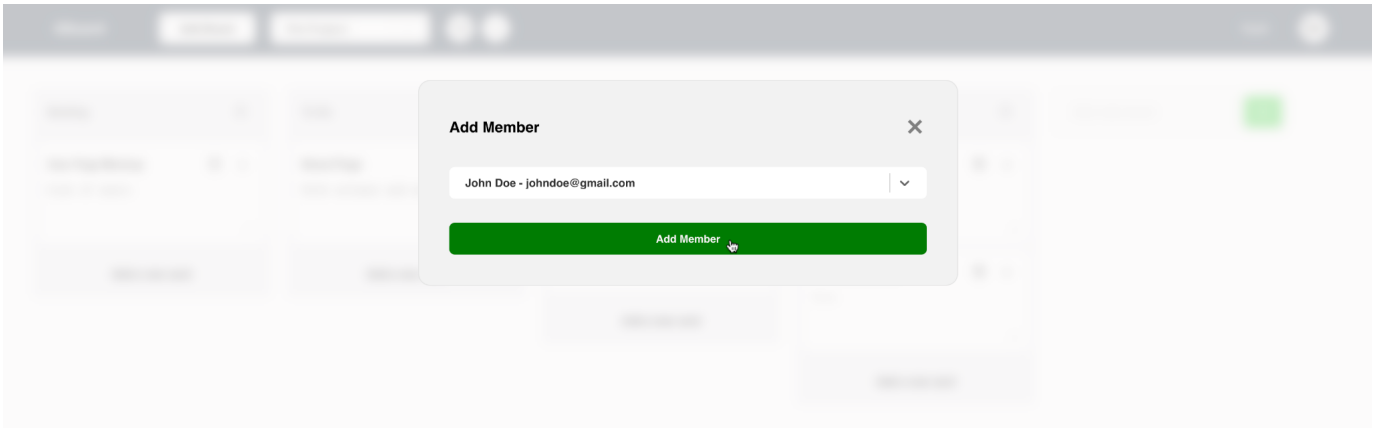


Рисунок 5.13 – Додавання члена команди

Використовуючи кнопку “+” у заголовку картки, мржна призначити цю картку одному з членів команди. Також після цього, використовуючи цю ж саму кнопку, можна змінити користувача, якому призначена відповідна картка.

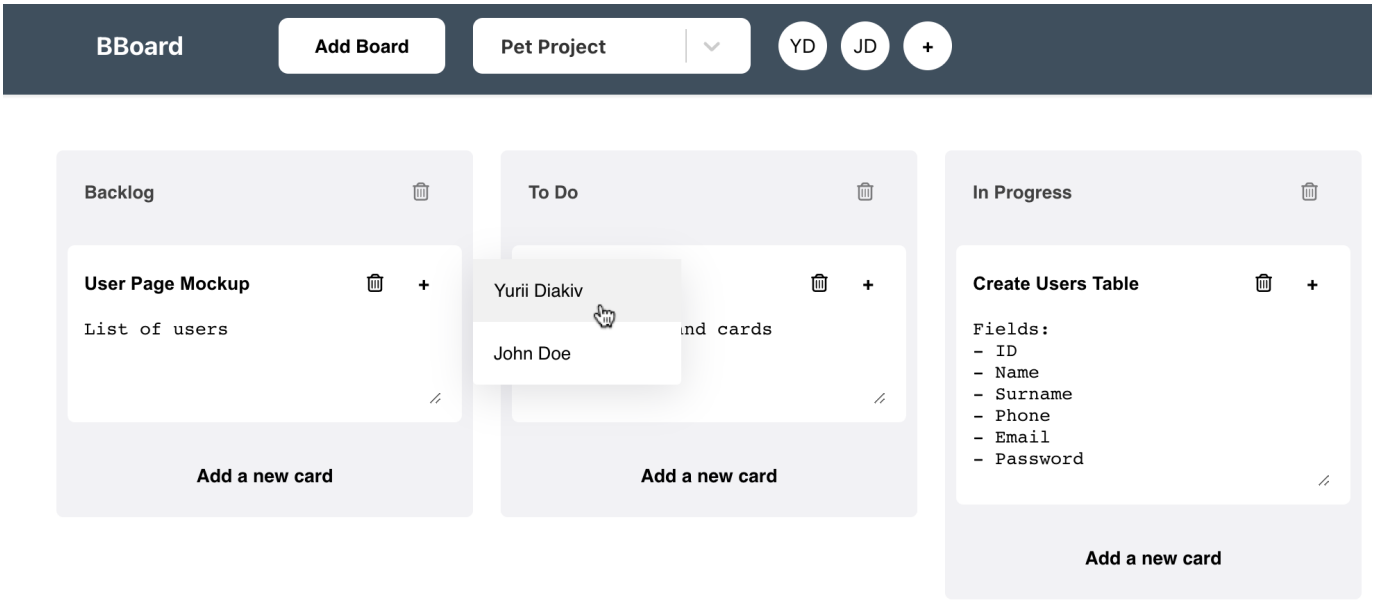


Рисунок 5.14 – Призначення картки члену команди

Кожну картку можна переміщувати між колонками дошки. Для цього потрібно натиснти на неї та, не відпускаючи кнопку миші, перенести її у потрібну колонку.

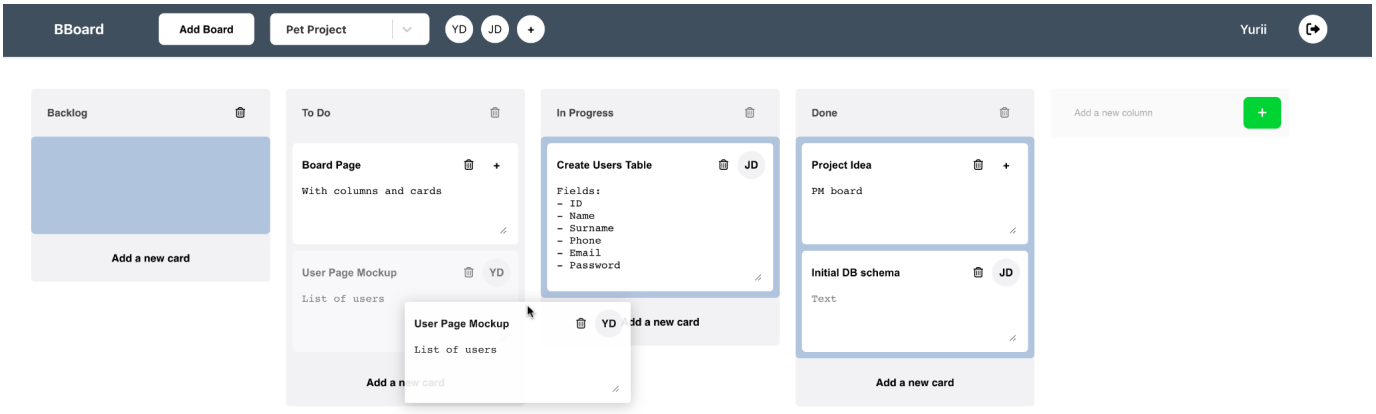


Рисунок 5.15 – Переміщення картки

ВИСНОВКИ

У магістерській роботі був розроблений застосунок, повністю готовий до роботи. За його допомогою можна керувати проектами, а саме розподіляти завдання між працівниками та контролювати час і виконання завдань. Застосунок розроблений для проджект менеджерів або команд на проектах, але він також підходить для студентів для відслідковування завдань в університеті або звичайних людей для особистих цілей.

У веб застосунку реалізовано основний функціонал дошок для завдань. Кожна дошка має колонки, створені користувачем, які можна редагувати та видаляти. Користувач має можливість додавати картки в колонки, редагувати, видаляти та переміщувати їх між колонками. Також можна додати нових членів в команди і призначити їм картки.

Веб застосунок вирішує основі завдання менеджменту проектів, а саме:

- Планування;
- Менеджменту часу та змін;
- Управління командою;
- Візуалізацію та структурування завдань;
- Оцінку ресурсів та ризиків;
- Розробку плану проекту;
- Моніторингу прогресу;

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is project management? [Electronic resource]. – Available from: https://www.apm.org.uk/resources/what-is-project-management/
2. Task boards: what they are and how to master them [Electronic resource]. – 2022 – Available from: https://monday.com/blog/project-management/task-boards-what-they-are-and-how-to-master-them/
3. Atlassian Jira – issue tracking product [Electronic resource]. – Available from: https://www.atlassian.com/software/jira
4. Trello – a web-based, kanban-style, list-making application [Electronic resource]. – Available from: https://trello.com/
5. Microsoft To Do – a cloud-based task management application [Electronic resource]. – Available from: https://todo.microsoft.com/
6. World Wide Web (WWW) [Electronic resource]. – Available from: https://en.wikipedia.org/wiki/World_Wide_Web
7. Duckett J. HTML & CSS: Design and Build Web Sites [Electronic resource] / J. Duckett. − 2011. − 490 p. – Available from: https://books.google.com.ua/books?id=aGjaBTbT0o0C&printsec=frontcover
8. Hypertext Transfer Protocol (HTTP) [Electronic resource]. – Available from: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
9. Flanagan D. JavaScript: The Definitive Guide: Master The World's Most-Used Programming Language [Electronic resource] / D. Flanagan. – 2020 – 706 p. – Available from: https://books.google.com.ua/books?id=NPbkDwAAQBAJ&printsec=frontcover
10. TypeScript is JavaScript with syntax for types [Electronic resource]. – Available from: https://www.typescriptlang.org/
11. Blanks A. Learning React: Functional Web Development with React and Redux [Electronic resource] / A. Blanks. – 2017 – 350 p. – Available from: https://books.google.com.ua/books?id=ycTADgAAQBAJ&printsec=frontcover

12. A Predictable State Container for JS Apps [Electronic resource]. – Available from: https://redux.js.org/
13. PostgreSQL: The World's Most Advanced Open Source Relational Database [Electronic resource]. – Available from: https://www.postgresql.org/
14. Casciaro M. Node.js Design Patterns: Design and implement production-grade Node.js applications using proven patterns and techniques, 3rd Edition [Electronic resource] / M. Casciaro. – 2020 – 660 p. – Available from: https://books.google.com.ua/books?id=xBr0DwAAQBAJ&printsec=frontcover
15. Fast and low overhead web framework, for Node.js [Electronic resource]. – Available from: https://www.fastify.io/
16. Knex.js is a "batteries included" SQL query builder [Electronic resource]. – Available from: https://knexjs.org/
17. An SQL-friendly ORM for Node.js [Electronic resource]. – Available from: https://vincit.github.io/objection.js/
18. Representational state transfer (REST) [Electronic resource]. – Available from: https://en.wikipedia.org/wiki/Representational_state_transfer
19. Beaulieu A. Learning SQL [Electronic resource] / A. Beaulieu. − 2005. – 336 p. – Available from: https://books.google.co.ck/books?id=YqubAgAAQBAJ&printsec=frontcover
20. Haverbeke M. Eloquent JavaScript: A Modern Introduction to Programming [Electronic resource] / M. Haverbeke. – 2011. − 224 p. – Available from: https://eloquentjavascript.net/