

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,

Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ОРГАНІЗАЦІЇ РОБОЧИХ
ПРОЦЕСІВ
WEB APPLICATION DEVELOPMENT FOR ARRANGING THE WORKFLOW

Виконав: студент(ка) 4 курсу, групи КНТ-127

Спеціальності 121 Інженерія програмного

забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Волович П.О.

(прізвище та ініціали)

Керівник Гладкова О.М.

(прізвище та ініціали)

Рецензент Морщавка С.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Воловича Павла Олександровича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка вебзастосунку для організації робочих процесів

Web Application Development for Arranging the Workflow

керівник проєкту (роботи) Гладкова Ольга Миколаївна, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “30” березня 2021 року № 103

2. Строк подання студентом проєкту (роботи) травень 2021 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Вибір засобів розробки. 3. Розробка програми. 4. Інструкція по використанню вебзастосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Гладкова О.М., доцент		
Нормоконтролер	Дейнега Л.Ю., старший викладач		

7. Дата видачі завдання “ 16 ” травня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз предметної області	1-2 тижні	Розділ 1
2	Вибір засобів розробки	3 тиждень	Розділ 2
3	Розробка програми	4-5 тижні	Розділ 3
4	Інструкція по використанню вебзастосунку	6 тиждень	Розділ 4
5	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	7 тиждень	Додатки
6	Захист роботи	8 тиждень	

Студент(ка)

(підпис)

Волович П.О.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Гладкова О.М.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 80 с., 5 табл., 21 рис., 2 дод., 27 джерел.

ПЛАНУВАННЯ, ОРГАНІЗАЦІЯ РОБОЧОГО ПРОЦЕСУ, ВЕБЗАСТОСУНОК, МОВА TYPESCRIPT, ФРЕЙМВОРКИ: REACT.JS, EXPRESS.JS.

Об'єкт дослідження – процес організації робочого процесу.

Предмет дослідження – вебзастосунок планування та організації робочого процесу.

Мета роботи – розроблення вебзастосунку планування та організації робочого процесу для автоматизації, зручності, прискорення ведення та подальшого поліпшення або зміни.

Матеріали, методи та технічні засоби: мова програмування TypeScript, фреймворк для клієнтської частини React.js, для серверної частини Express.js, система керування базами даних MongoDB, персональний комп'ютер з процесором Intel Core 2 Duo під управлінням операційної системи Microsoft Windows 7.

Результати. Аналізуючи аналоги для рішення організації робочого процесу і ґрунтуючись на їх засадах розроблено уявлення та концепія вебзастосунку. Створено вебзастосунок.

Висновки. Розроблен веб-додаток для спрощення ведення та організації робочого процесу.

Галузь використання – планування та організація робочих процесів в різних сферах діяльності.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 80 pages, 21 figures, 5 tables, 2 appendixes, 27 sources.

ARRANGING THE WORKFLOW, WEB APPLICATION, TYPESCRIPT LANGUAGE, FRAMEWORKS: REACT.JS, EXPRESS.JS

Object of study is the process of organizing the work process.

Subject of study is the web application of workflow planning and organization.

The purpose of the work is to develop a web application for planning and organizing the workflow for automation, convenience, acceleration of management and further improvement or change.

Materials, methods, and tools: TypeScript programming language, framework for the client part React.js, for the server part Express.js, MongoDB database management system, personal computer with Intel Core 2 Duo processor running Microsoft Windows 7 operating system.

Results. Analyzing analogues for the decision of the organization of the working process and based on their principles developed the idea and concept of the web application. Web application was implemented.

Conclusions. Developed a web application to simplify the management and organization of the workflow.

The field of use is planning and organization of work processes in various fields.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ.....	9
1 Аналіз предметної області.....	11
1.1 Вебзастосунки планування та організації робочих процесів	11
1.2 Висновки за розділом 1	15
2 Вибір засобів розробки	16
2.1 Важливість вибору засобів розробки	16
2.2 Вибір засобів проєктування та розробки	18
2.2.1 Засоби розробки клієнтської частини	18
2.2.2 Засоби розробки серверної частини	21
2.3 Висновки за розділом 2	24
3 Розробка вебзастосунку	25
3.1 Визначення функціональної поведінки вебзастосунку	25
3.2 Архітектура вебзастосунку	27
3.2.1 Архітектура процесу відображення	27
3.2.2 Архітектура процесу роботи з даними	30
3.3 Структура бази даних	35
3.4 Висновки за розділом 3	37
4 Керівництво користувача і програміста	38
4.1 Призначення програми	38
4.2 Умови виконання програми	38
4.3 Характеристики програми.....	39
4.4 Звернення до програми.....	39
4.5 Вхідні та вихідні дані.....	40
4.6 Повідомлення користувачу	41

Висновки	46
Перелік джерел посилання	47
Додаток А Текст програми.....	49
Додаток Б Слайди презентації	75

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
DOM	– Document Object Model;
НОС	– High Order Component;
JSX	– JavaScript XML;
MVC	– Model-View-Controller;
ORM	– Object-Relational Mapping;
SPA	– Single Page Application;
TSX	– TypeScript XML;
UI	– User Interface;
XML	– eXtensible Markup Language;
БД	– База Даних;
ПЗ	– Програмне Забезпечення.

ВСТУП

В наш час майже не можливо уявити життя людини без процесів, які пов'язані з робочою діяльністю, де слід дотримуватися певних поставлених завдань, додержання правил, принципів чи вимог. Незважаючи на ще те, що у людини може бути декілька справ, йому потрібно постійно враховувати і стежити за всіма діями, що він робить. Людина вибудовуючи приблизний план дій, має на меті дотримуватися цього плану, виконуючи певні процеси, функції. Крім того, будь-який процес повинен проходити систематично зрозуміло, плановірно, з мінімальною кількістю виникнення проблем або помилок. На цьому етапі і виникають проблеми зі згуртованістю і організацією робочого процесу.

Незважаючи на розмір роботи, проблем або цілей, в будь-якому випадку з'являться причини або необхідності для збільшення їх в обсязі. У таких моментах було б не погано не просто тримати це в голові, але і зафіксувати це, припустимо, на папері або макеті, що, власне кажучи, вже не погано. Але як тільки час прийде для модифікації або видалення певного процесу, то виникнуть невеликі труднощі: доведеться закреслювати або писати на новому аркуші, макеті. У таких випадках ідеально підійде який-небудь вебзастосунок для фіксації та організації певних етапів, кроків і матеріалів, адже вже в 21 столітті набагато зручніше що-небудь складати і зберігати онлайн, ніж на папері, а то і в голові. Слідуючи з цього виникає значущість і актуальність про створення веб-додатку для організації робочих процесів.

Об'єкт дослідження – організація робочого процесу.

Предмет дослідження – вебзастосунок організації робочого процесу.

Мета роботи полягає в розробці вебзастосунку організації робочого процесу для поліпшення його ведення, можливості зручної модифікації та уникнення значних непорозумінь та складнощів.

Для виконання мети роботи було визначено її завдання:

- аналіз подібних вебзастосунків;
- визначення вимог до вебзастосунку організації робочих процесів;
- проєктування вебзастосунку;
- реалізація вебзастосунку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Вебзастосунки планування та організації робочих процесів

У наш час в інтернеті існує багато різних варіантів вебзастосунків, які допомагають впоратися з поточною проблемою. Але можна виділити декілька найпоширеніших і часто використовуваних на практиці.

Trello [1] (рис. 1.1) – це застосунок, що працює за принципом канбан [2] – популярної методики управління проектами, яка допомагає контролювати організаційні процеси проекту та певних завдань. Розглядаючи цей вебзастосунок, можна відзначити:

- в основному весь базовий функціонал безкоштовний;
- багатомовність;
- легкість функціоналу в створенні проекту і завдань до нього;
- є мобільний додаток;
- можливість створювати дошки, в яких можна додавати список завдання стільки, скільки буде визначено користувачем;
- можливість придумати дошкам будь-які назви, призначення, розташувати їх в будь-якому порядку. Це означає, що система, яку ви для себе визначили або придумали, може бути абсолютно будь-якою, і Trello буде під неї підходити;
- Trello дозволяє організувати як приватні, так і публічні проекти, куди можна додати як свою команду, так і певних користувачів;
- при створенні картки всередині дошки можна призначити користувача на це завдання і зафіксувати дедлайн, до якого ця картка повинна бути зроблена або реалізована, також є можливість додати опис до картки, який можна відформатувати у будь-який вигляд з додаванням певних медіа ресурсів, які дозволяють людині більш детально ознайомитися з проблемою або візуально побачити, що слід очікувати від його роботи;

– однією з дуже зручних функцій є інтегрування додаткових вебзастосунків, таких як Slack, Google Drive, OneDrive, Dropbox, Github, Twitter. Крім того, цей список постійно збільшується;

– платна версія залежить від варіанту вибору, але в основному вона дозволяє користувачам завантажувати файли до 250 МБ (в безкоштовній версії тільки до 10 Мб), необмежене створення дощок (у безкоштовній тільки 10), можливість створювати шаблони для робочих просторів, створювати колекції, дозволяє бачити історію пошуку, а так само дозволяє робити вхід в додатки Google, але платна версія не може вплинути на робочий процес і організацію проекту в Trello [3], [4].

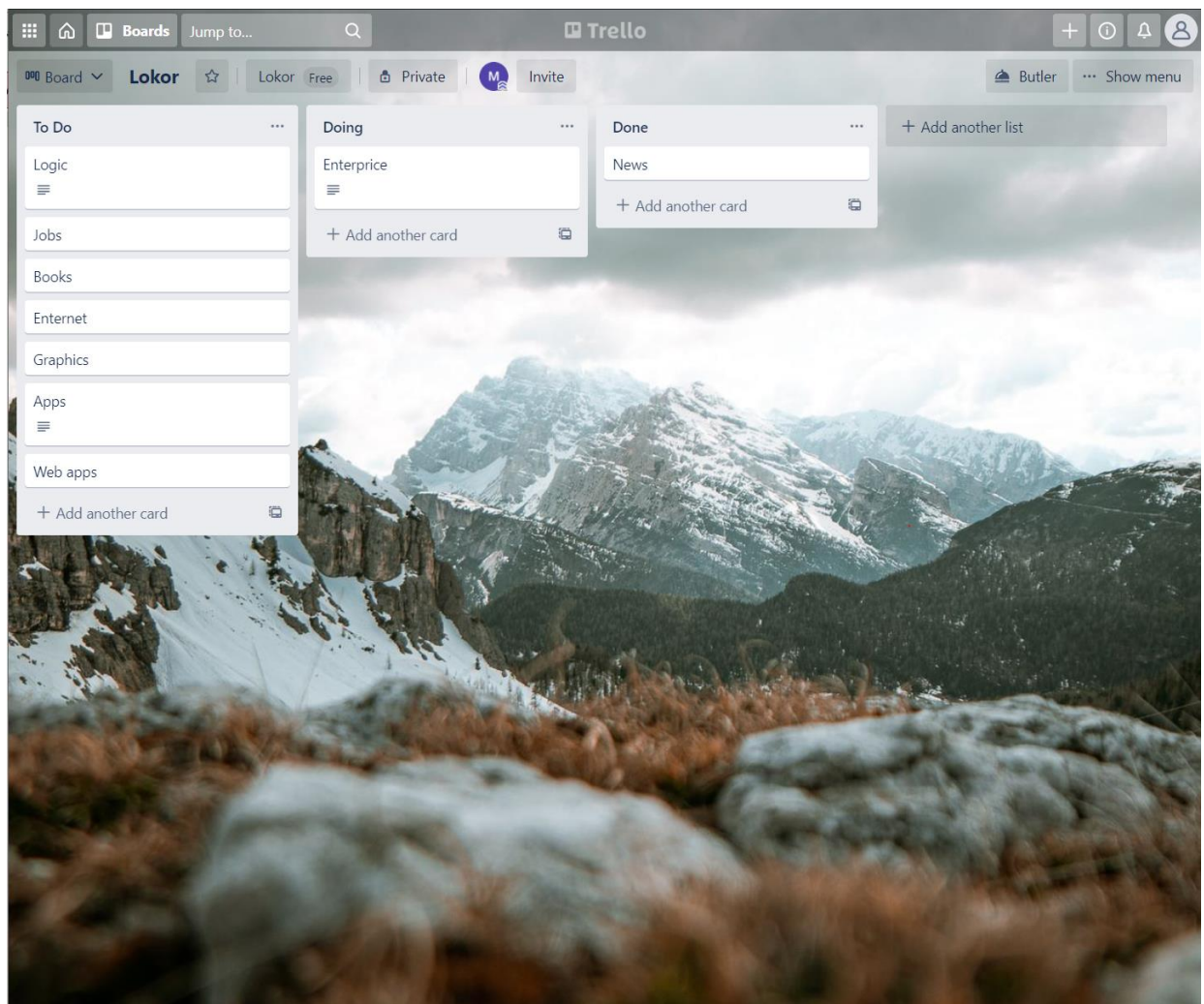


Рисунок 1.1 – Trello [1]

А також вебзастосунок дозволяє вибрати варіант використання: безкоштовний, business class, enterprise.

Безкоштовний варіант підійде як для команди не більше п'яти осіб, так і для однієї людини, яка зможе без проблем вести свій проект по досягненню певних цілей або завдань.

Business class підходить для команди менше 100 чоловік, у якої є велика кількість проектів і їх постійно потрібно оновлювати, стежити за ними, змінювати і так далі. Коштує такий варіант 10 американських доларів США за користувача в місяць при оплаті за рік (12,50 долара при щомісячній оплаті) [3].

Enterprise найкраще йде для компаній, які хочуть організувати спільну роботу команд, поліпшити методи захисту, безпеку і підтримку корпоративного рівня. Ідеальне рішення для організацій від 100 осіб. Вартість даного варіанту 210 долара за користувача в місяць при оплаті за рік [4].

Отже даний веб ресурс дуже корисний як для стартапів, підприємств, які мають у себе на рахунку безліч задумів або проектів, які необхідно постійно моніторити і поліпшувати, так і для людини, яка займається саморозвитком і плануванням своїх планів.

Miro [5] (рис. 1.2) – онлайн платформа для спільної віддаленої роботи, в основі якої лежить онлайн-дошка. Дошка чудово підходить як для одного користувача, так і для команд зі зборами і конференціями, де слід в режимі реального часу показати цілі і план робочого процесу. Дана дошка надає можливість додавання файлів і документів, малювання, створення заміток і стікерів. Так само слід зазначити, що при великій кількості учасників можна включити функцію «відстеження курсору», де користувач буде бачити дії співрозмовників.

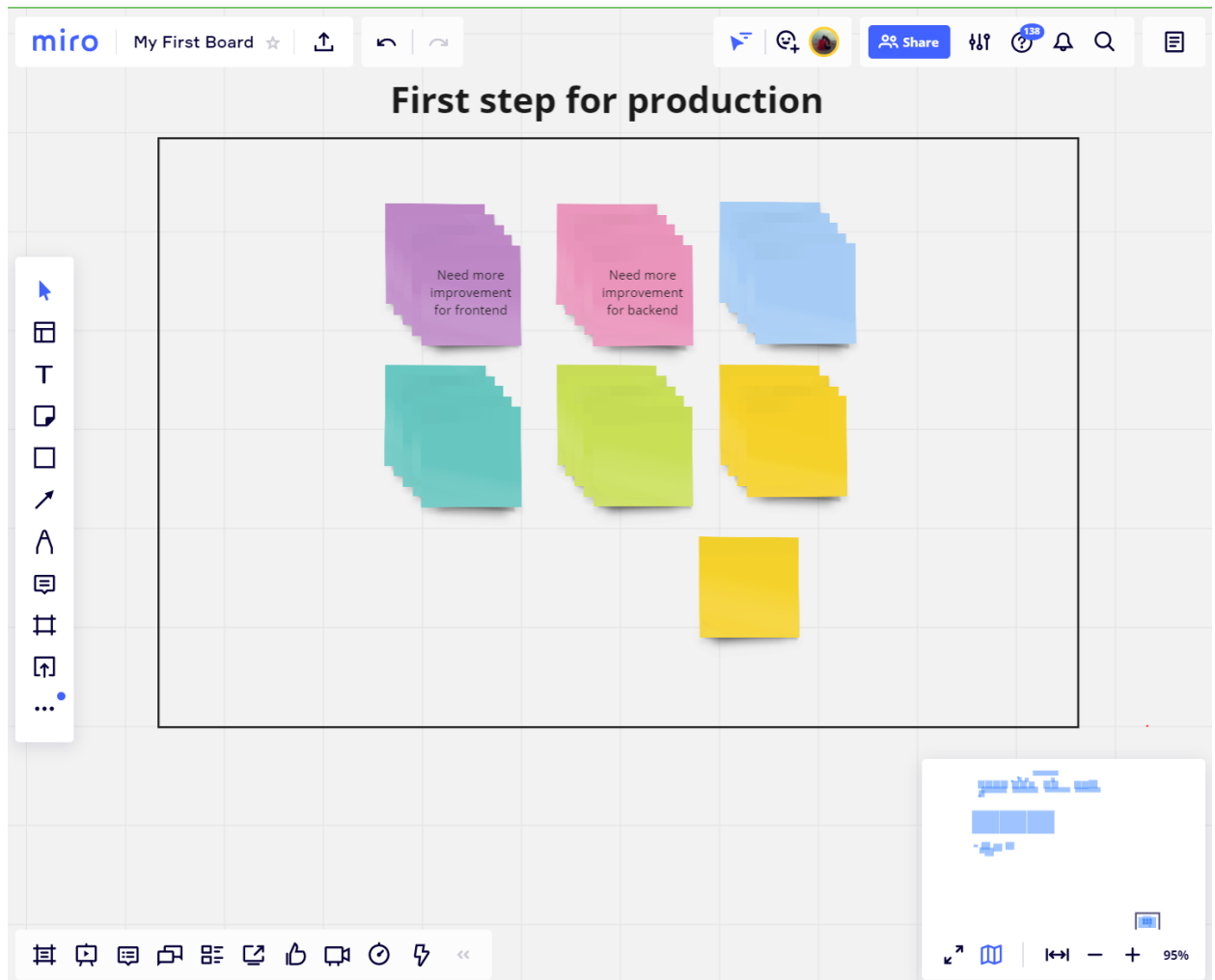


Рисунок 1.2 – Miro [5]

Дана онлайн платформа є як у безкоштовній, так і в платній версії.

З приводу безкоштовного функціоналу:

- дозволено 3 редаговані дошки;
- дозволений анонімний перегляд на дошці з загальнодоступною посиланням, де вхід в систему не потрібен;
- базове управління увагою: можливість залучати окремих співробітників до певної області дошки і стежити за тим, що вони роблять.

З приводу платного функціоналу:

- доступне створення необмеженої кількості дощок;
- спільне використання приватної дошки;

- можливість запросити будь-якого в якості анонімного користувача для спільної роботи на загальнодоступних дошках, де вхід в систему не потрібен;
- доступно розширене керування увагою: можливість використовувати функціонал «Привести до мене», щоб залучити всіх співавторів до певної області дошки;
- можлива розширена безпека, включаючи списки дозволених доменів і журнали аудиту;
- панель управління на рівні організації, що дають уявлення про аналітику користувачів, дощок, проектів та шаблонів [6].

1.2 Висновки за розділом 1

Аналізуючи предметну область, варто відзначити, що для поліпшення робочого процесу варто використовувати вебзастосунки, як в команді, так і для власних цілей. Для цього найбільш відповідними варіантами будуть Trello, Miro. Кожний застосунок підходить для різних цілей по своєму, але обидва мають можливість допомагати в організації та плануванні робочого процесу, уникаючи мінімальних втрат і непорозумінь.

2 ВИБІР ЗАСОБІВ РОЗРОБКИ

2.1 Важливість вибору засобів розробки

Безумовно важливим кроком в розробці вебзастосунку є прийняття вірних рішень щодо вибору інструментів розробки, технологічного стека. Це основний критерій для створення додатка (відповідно до вимог та потреб користувача), який допомагає підвищити показник ефективності та продуктивності робочого процесу розробника, що створює цей вебзастосунок.

Це досить важливий етап в розробці та розвитку всього робочого процесу. Добре підібраний технічний стек гарантує, що продукт буде якісним і відповідає очікуванням клієнтів та користувачів, а також запобігає перед розробкою величезну кількість всіх майбутніх проблем, пов'язаних з виправленням або подальшим поліпшенням. Це безумовно може заощадити досить великий об'єм як часу для розробника, так і коштів для замовника.

Фактори, які слід враховувати при виборі набору технологій:

а) розмір веб-додатку: майбутній розмір проекту впливає на процес вибору стека технологій. Наприклад, у міру зростання вебзастосунку збільшується і складність розробки (рис 2.1);

б) масштабованість: при збільшенні кількості користувачів, технологічний стек повинен дозволяти вебзастосунку справлятися з ростом. Отже, розробник повинен вибрати кращу технологію для веб-розробки, яка може підтримувати як горизонтальний, так і вертикальний ріст (рис 2.1). Горизонтальне зростання взагалі означає здатність веб-додатку справлятися з ростом користувачів. З іншого боку, вертикальний ріст означає можливість додавати нові функції і можливості на вимогу користувачів;

в) можливість обслуговування: це вирішальний фактор при створенні веб-додатків. Простіше кажучи, обслуговування веб-додатку повинно виконуватися без особливих зусиль навіть при масштабуванні. Зокрема, є два фактори, від яких залежить

ремонтпридатність вебзастосунку: архітектура та кодова база. Архітектура веб-додатку повинна підтримувати можливість повторного використання, переносимість і масштабованість. Простіше кажучи, технологічний стек повинен дозволяти включати додаткові інструменти і фреймворки, коли прийде час для зростання. Кодова база веб-додатку не повинна бути ні занадто довгою, ні занадто вузькою. Тому що, для обробки довгого коду буде потрібно більше часу, а стислий код може викликати помилки налагодження;

г) вартість розробки: важливо заздалегідь вирішити, скільки потрібно витратити на створення вебзастосунку. Хоча більшість технологій, інструментів і фреймворків, які використовуються в веб-розробці, мають відкритий код і безкоштовні для використання, але, можливо, доведеться купити підписку будь-якого інструменту, щоб отримати доступ до його розширених функцій;

г) безпека: у наш час будь-який онлайн ресурс схильний до злому або кібератак. Більш того, кількість кібератак з кожним роком зростає. Наприклад, згідно з прогнозом Center for Strategic & International Studies [7], в березні 2021 року китайські хакери атакували програмне забезпечення корпоративної електронної пошти Microsoft, щоб вкрати дані більш ніж 30 000 організацій по всьому світу, включаючи урядові установи, законодавчі органи, юридичні фірми, підрядників до питань оборони, дослідників інфекційних захворювань і аналітичні центри [8].

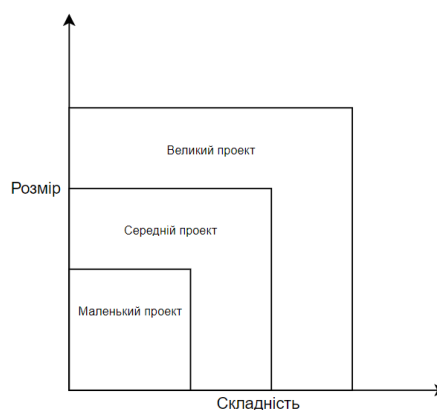


Рисунок 2.1 – Розширення проекту

Тому вибору технологічного стека повинно приділятися рівно стільки ж уваги, як і до самої розробці програми, при цьому він повинен бути ретельно продуманий і визначений.

2.2 Вибір засобів проєктування та розробки

2.2.1 Засоби розробки клієнтської частини

При розробці клієнтської частини було визначено використовувати одну з найпопулярніших мов програмування JavaScript [9]-[10]. На відміну від ще однієї популярної клієнтської мови VBScript [11]-[12], JavaScript є за замовчуванням мовою сценаріїв для більшості браузерів, тоді як перший не є таким і виступає в ролі мови сценаріїв, а також інші порівняння були виявлені (табл. 2.1).

Таблиця 2.1 – Порівняння клієнтських мов програмування JavaScript і VBScript

Критерій	JavaScript	VBScript
Підтримка браузерів	+	+—
Гнучкість синтаксису	+	—
Широкий спектр навчання	+	—
Підтримка функціональності	+	+—
Безкоштовність	+	+

Завдяки великій кількості вирішених проблем і налаштувань, а також широкій спільноті програмістів та великій кількості підтримки браузерів, мова JavaScript є кращим вибором для вебзастосунків.

На додаток до клієнтської частини до JavaScript була обрана бібліотека React.js [13]-[14], яка дуже конкурує з такими фреймворками, як: Angular.js [15-16], за який відповідає компанія Google [17], Vue.js [18-19], який зробив колишній дуже амбітний співробітник Google (табл. 2.2).

Таблиця 2.2 – Порівняння бібліотеки React.js з фреймворками: Angular.js, Vue.js

Критерій	React	Angular	Vue
Мова	JavaScript	TypeScript	JavaScript
Бажана мова розробки	JSX/TSX	TypeScript	HTML-шаблони і JavaScript
Підтримка спільнотою	Спільнота розробників Facebook	Велике співтовариство розробників Google	Проект з відкритим кодом, підтримуваний співтовариством
Тип	Бібліотека	Фреймворк	Фреймворк
Розробник	Facebook	Google	Колишній співробітник Google
Модель	DOM	MVC	DOM
Поріг входження	Середній	Малий	Великий
Типи застосунків	Нативні, гібридні, веб-додатки	SPA, мобільні додатки	Просунуті SPA, нативні веб-додатки

При всіх своїх сильних аспектах інших фреймворків, бібліотека React.js залишається найпопулярнішою в розробці відносно до не великих, середніх, компонентно-орієнтованих веб-додатків, з можливістю легко змінювати компоненти і перевикористовувати код. До того ж, компоненти, які були

створені під час розробки, не мають додаткових залежностей, тому React.js являється не фреймворком, а бібліотекою, в якій використовуються залежності тоді, коли вони потрібні. Так само, використовуючи можливість Virtual DOM, React.js дає можливість з високою швидкістю створювати високопродуктивні додатки, незважаючи на рівень їх складності.

Для реалізації вебзастосунку разом з React.js було обрано використовувати мову TypeScript, яка дає додаткові можливості і переваги при розробці:

- TypeScript дозволяє впоратися зі зростаючим розміром коду;
- типізація дає здатність підвищувати якість і зрозумілість коду;
- типізація підвищує гнучкість при рефакторингу;
- компілятор краще відловлює помилки;
- типізація – одна з кращих форм документації, яку розробник можете мати;
- TypeScript дає уявлення про майбутнє JavaScript;
- впровадження типів робить код надзвичайно передбачуваним і зручним в обслуговуванні;
- код не компілюватиметься, якщо отримає те, чого не очікує;
- допомагає реалізувати шаблон проектування SOLID в мові, яка насправді його не підтримує;
- можливість компілювати до версії JavaScript, яка працює у всіх браузерах;
- автозавершення коду і IntelliSense [20];
- Intellisense надає активні підказки при додаванні коду;
- статична типізація, яка виявляє помилки при написанні скриптів;
- рефакторинг за допомогою інструментів TypeScript стає простіше і швидше;
- підтримує зміни, вживаючи заходи безпеки, які гарантують повну перевірку стану.

TypeScript стає критично важливим вибором для проекту, який в подальшому збирається розширюватися і підтримуватися. Структура об'єктно-орієнтованої мови програмування забезпечує чистоту, масштабованість, узгодженість і простоту налагодження коду.

Отже слід сказати, що при виборі мови програмування для клієнтської частини варто краще використовувати JavaScript, який підтримується абсолютно у всіх браузерах і бажано з яким небудь фреймворком, хоча існують багатообіцяючі способи розробки складних додатків без фреймворків, проте вони, як і раніше, роблять багато дивовижних речей і можуть зробити розробку більш гладкою. Вони спонукають створювати зручну в обслуговуванні модульну автономну архітектуру, яка спрощує створення програми та співробітництво з іншими розробниками.

2.2.2 Засоби розробки серверної частини

Для серверної частини вебзастосунку була обрана програмна платформа Node.js, заснована на движку V8, що перетворює JavaScript з вузькоспеціалізованого мови в мову загального призначення [21]. З огляду на те, що клієнтська частина обрана на мові JavaScript, краще вибрати для бекенду таку ж мову. Даний вибір універсальний, особливо коли з'являється необхідність зробити великий функціонал, який ділиться на front- і back-частини. В такому випадку бекенд може взяти на себе всі обов'язки і ділити завдання на свій розсуд. Це зручно і прискорює процес, так як зникає додаткове узгодження API і поведінки. Так само іноді бувають проблеми зі зміщенням навантаження з бекенду на фронтенд і навпаки.

Тому бекенд з Node.js може маневрувати і перетягувати на себе частину завдань з фронтенду, що робить з цього гнучкість в розробці.

Що б прискорити процес створення серверної частини вебзастосунку, для Node.js існує чимало фреймворків. Одними з найпопулярніших є Express.js [22] і Nest.js [23], порівняння яких було виявлено (табл. 2.3).

Таблиця 2.3 – Порівняння фреймворків Express.js і Nest.js

Критерій	Express.js	Nest.js
Суспільство	Численне	Середнє
Персоналізація	Проста за рахунок модулів та роутінгу	Важкувата, але структурована завдяки фільтрам, каналам, перехоплювачам
Оновлення версій	Регулярно	Регулярно
Документація	Детальна, розділена по модулях	Спрощена, з великою кількістю розділів
Наявна типізація	Немає	Використовує TypeScript
Незалежність	По мірі необхідності	Використовує проміжне ПЗ Express.js
Поріг входження	Великий	Середній

Для бекенду вебзастосунку був обраний фреймворк Express.js у сумісності з TypeScript, адже він краще підходить для вирішення завдань типу роутінга, парсинга тіла та іше, окрім того Express.js немає додаткових залежностей і стороннього проміжного ПЗ і при необхідності легко поєднує в собі різні підключені модулі.

В якості системи управління бази даних було вибрано MongoDB [24]. Цей вибір обумовлений безпосередньо такими перевагами:

- а) модель даних документа – це потужний спосіб зберігання і витягання даних, що дозволяє розробникам діяти швидко;
- б) горизонтально масштабована архітектура MongoDB може підтримувати величезні обсяги як даних, так і трафіку;
- в) MongoDB має відмінний користувальницький інтерфейс для розробників, які можуть встановити MongoDB і відразу ж почати писати код;

г) MongoDB може використовуватися всюди ким завгодно:

- безкоштовно через версію спільноти з відкритим вихідним кодом;
- у найбільших дата-центрах через корпоративну версію;
- у будь-якому з основних публічних хмар через MongoDB Atlas;

г) MongoDB розробила велику і зрілу екосистему платформи, що означає:

- MongoDB має всесвітнє співтовариство розробників і консультантів, тому отримати допомогу легко;
- MongoDB працює на всіх типах обчислювальних платформ, як локально, так і в хмарі (як в приватних, так і в публічних хмарах, таких як AWS [25], Azure [26] і Google Cloud [27]);
- MongoDB можна використовувати на всіх основних мовах;
- доступ до MongoDB можна отримати з усіх основних систем ETL і управління даними;
- MongoDB має підтримку корпоративного рівня;

д) представляє дані з природними кластерами і мінливістю в часі або за своєю структурою;

- е) підтримує швидку ітеративну розробку;
- є) можливість спільної роботи великої кількості команд;
- ж) масштабування до високих рівнів трафіку читання і запису;
- з) можливість змінювати тип розгортання в міру зміни проекту;
- и) зберігання, управління даними за допомогою текстових, геопросторових вимірювань або вимірювань часових рядів.

2.3 Висновки за розділом 2

У даному розділі було виявлено важливість вибору технологічного стека для розроблення вебзастосунку. Визначено мову програмування JavaScript для клієнтської частини з фреймворком React.js і серверної частини з фреймворком Express.js, та MongoDB, як систему управління базою даних.

3 РОЗРОБКА ВЕБЗАСТОСУНКУ

3.1 Визначення функціональної поведінки вебзастосунок

Для успішного ведення робочого процесу веб-додаток надає можливість використовувати систему організації процесу, завдяки веденню проекту і створення карток з описом завдання цього проекту.

Виділено 2 основні частини системи:

- система управління проектами. Дана система передбачає повне управління проектом, з можливістю контролювати внутрішні картки;
- система управління картками проекту. Система повинна надавати повний контроль користувачу над картою у визначенні теми конкретного завдання, опису проблеми завдання, а також можливість визначення категорії картки: вихідні картки (Resources), які потрібно зробити (To do), в процесі (In progress), готові (Done).

Основні функції програми, які може виконувати користувач (рис. 3.1):

- перегляд всіх створених проектів;
- створення проекту;
- визначення теми проекту;
- створення спеціального символу (ключа) проекту, для класифікації та нумерації внутрішніх карток;
- редагування теми проекту;
- зміна ключа проекту;
- повне видалення проекту;
- можливість завантажувати певний проект;
- перегляд всіх створених карток проекту;
- створення картки певного проекту;
- можливість визначення заголовка картки;
- можливість опису проблеми картки;

- редагування опису проблеми картки;
- видалення картки;
- можливість завантажувати картку;
- зміна теми картки;
- зміна опису картки;
- можливість переміщення карток за категорією;
- можливість переміщення картки всередині певної категорії.

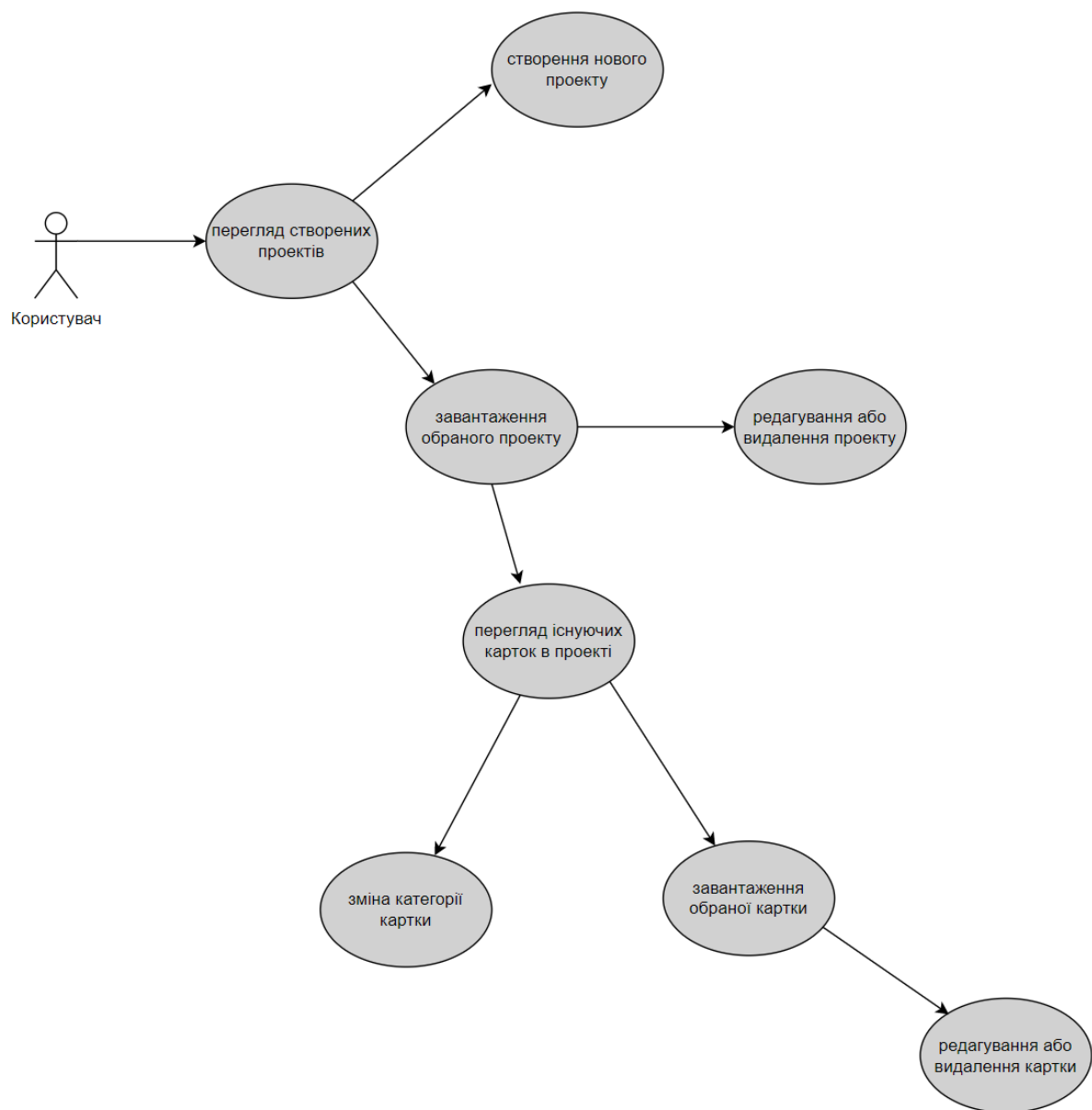


Рисунок 3.1 – Діаграма використання вебзастосунку користувачем

3.2 Архітектура вебзастосунку

3.2.1 Архітектура процесу відображення

Щоб дані відображалися безпомилково і правильно, з подальшим додаванням нових елементів і подальшою підтримкою, для будь-якої програми потрібна правильно складена логіка відображення даних і їх обробка.

У вебзастосунку використана наступна архітектура для контролю відображення:

а) components – тека, яка містить компоненти відображення (презентаційні компоненти) UI, в яких приймаються різні параметри для відображення, візуальної обробки даних або зворотних викликів. Даний розділ містить такі компоненти:

- компонента CurrentTicketContent відображає інформацію конкретної завантаженої картки певного проекту;
- компонента KanbanColumn, яка відображає колонку з назвою певної категорії для карток;
- компонента KanbanColumnContent виконує функцію відображення карток, які містяться в певній колонці категорії;
- компонента KanbanHeader відображає верхню частину всієї канбан дошки проекту, де присутня інформація про назву проекту, його ключа і налаштування проекту;
- компонента NewProjectContent, яка відповідає за введення інформації в поля форми нового проекту;
- компонента NewProjectHeader відображає шлях, де користувач знаходиться, починаючи з головної сторінки проектів, і закінчуючи поточною сторінкою;
- компонента NewTicketContent відповідає за введення інформації в поля форми для створення нової картки;

- компонента `NotFoundContent` відображає візуальну частину не знайденої сторінки, маршруту веб-додатку;

- компонента `ProjectSettingsContent` відповідає за відображення поточної інформації проекту, а також за введення або зміну інформації проекту;

- компонента `ProjectSettingsHeader` відображає шлях, де користувач знаходиться, починаючи з головної сторінки проектів, і закінчуючи поточною сторінкою;

- компонента `ProjectsHeader` відображає заголовок проекту та інформацію про створення нового проекту;

- компонента `ProjectsTable` займається візуалізацією всіх існуючих проектів;

б) `containers` - тека, яка містить в собі компоненти з функціоналом і логікою презентаційних компонент (контейнерні компоненти), в якій містяться дані компоненти:

- компонента `CurrentTicket`, яка відповідає за логіку роботи компоненти конкретної картки;

- компонента `Kanban`, на яку покладено відповідальність за роботу всієї канбан дошки проекту, з можливістю створювати нову картку;

- компонента `NewProject`, яка несе відповідальність за функціонал створення нового проекту;

- компонента `NotFound`, в якій міститься презентаційна компонента не знайденої сторінки або маршруту веб-додатку;

- компонента `ProjectSettings` займається обробкою логіки зміни інформації поточного проекту;

- компонента `Projects` стежить за логікою завантаження всіх проектів;

в) `layouts` - містить в собі компоненти вищого порядку (НОС), які працюють за принципом обгортки: приймають у якості параметра інші

компоненти і повертають на результаті їх. Слід зазначити, що такі компоненти не потребують часткової або повної зміни. Дана тека має такий зміст:

- шаблон MainLayout, в який обертається весь контент кожної сторінки проекту;
- шаблон NavBar, котрий є загальним заголовком для кожної сторінки;

г) pages – тека, для компонентів сторінок, в якій містяться:

- сторінка CurrentTicket з вмістом компонентів логіки і візуальної частини конкретної картки;
- сторінка Kanban, яка вміщує в собі компоненти роботи і відображення канбан дошки проекту;
- сторінка NewProject відповідає за зміст компонентів логіки і відображення форм створення нового проекту;
- сторінка NotFound з компонентами відображення неіснуючої сторінки;
- сторінка ProjectSettings з логікою і візуальною частиною налаштувань проекту, а також Projects з аналогічним вмістом;

г) shared – розділ із загальними матеріалами, які будуть перевикористати кілька разів і постійно повторяться. До цієї теці належать такі сутності, як:

- button, яка представлена у вигляді кнопки для виклику будь-якого дії;
- link, що перенаправляє користувача на інші сторінки;
- loader, яка візуалізується як іконка, для відображення завантаження при виконанні будь-якого запиту до серверу;
- modalDelete відповідає за відображення модального вікна, з інформацією підтвердження або скасування;
- textEditor, яка відповідає за відображення введення у текстове поле.

3.2.2 Архітектура процесу роботи з даними

Для архітектури роботи з даними у вебзастосунку було використана методологія Flux [28], яка в більшій мірі надання і присутня в такому стейт менеджері, як Redux [29]. У той час, як React.js найчастіше називають поданням до структури MVC [30], насправді React.js підходить більше для структури Flux, що беззастережно підтвердила сама компанія Facebook, яка розробила React.js і дану методологію [31]. Проблема зі структурою MVC полягає в тому, що це двонаправлений зв'язок, яку, виявилось, дуже складно налагодити і зрозуміти, коли зміна в одній сутності викликає каскадний ефект у всій кодової базі. Особливо це помітно, коли додаток масштабується до більш крупного. В такому випадку потік даних буде недостатньо добре визначен.

Основні складові структури Flux:

а) уявлення (view). Вони складають візуальну частину програми. Особливий вид уявлень – controller-view, який представляє компонент самого верхнього рівня, який містить всі інші компоненти. Controller-view прослуховує події, які виходять від сховища. Отримавши подію, controller-view передає дані, отримані від сховища, іншим компонентам. Коли controller-view отримує подію від сховища, то спочатку він запитує у сховища всі необхідні дані. Потім він викликає свій метод, який призводить до виконання у компонента методу відтворення, який в свою чергу призводить до оновлення дочірніх компонентів. Дія може бути викликана обробниками подій в компонентах, наприклад, після натискання на кнопку, або ініціатором дій може бути якесь інше зовнішнє джерело, наприклад, сервер. Через диспетчер сховище отримує дію і відповідним чином реагує на нього;

б) сховище даних (store). Воно містить стан програми та його логіку. За своєю дією воно може нагадувати модель з паттерна MVC, в той же час воно не представляє один об'єкт, а керує групою об'єктів. Кожне окреме сховище керує певною областю застосунку. Кожне сховище реєструється в диспетчері разом зі своїми зворотними викликами. Коли диспетчер отримує дію, то він виконує зворотний виклик, передаючи йому дію, яке надійшло в якості параметра. Залежно від типу дії викликається той чи інший метод всередині сховища, в якому відбувається оновлення стану сховища. Після поновлення сховища генерується подія, яке вказує, що сховище було оновлено. І через цю подію уявлення (тобто компонент React) дізнаються, що сховище було оновлено, і самі оновлюють свій стан;

в) диспетчер (dispatcher). Він уявляє в усій цій схемі центральну ланку, яка керує потоком даних у Flux архітектурі, а також обробляє зареєстровані дії і зворотні виклики. Коли диспетчер отримує ззовні деяку дію, то через коллбеки сховищ диспетчер повідомляє ці сховища про дію, яка надійшла.

Диспетчер може бути викликаний дією не тільки з сервера, він так само може спрацювати від дії з клієнтської частини, коли необхідно в режимі реального часу продемонструвати зміни візуальної частини без затримок, в той час, поки виконується запит на сервер і передаються необхідні дані.

Для цього був виконан метод `changeCardColumn({projectId, draggableId, srcIndex, destIndex, srcDroppableId, destDroppableId, enqueueSnackbar})`, для того, щоб переставити картку з однієї колонки категорії в іншу.

Спочатку ми відправляємо інформацію у сховище про те, що відбувається запит на сервер:

```
dispatch({
  type: SET_CARD_DESTINATION_REQUEST,
});
```

Це робиться для того, щоб при виникненні помилки розуміти, на якій стадії була визначена помилка ми дістаємо всі існуючі колонки з категоріями з нашого сховища.

Потім ми дістаємо всі існуючі колонки з категоріями з нашого сховища:

```
const { columns } = getState().kanban;
```

Далі ми проходимо через масив з колонками категорій через цикл, щоб знайти і обробити потрібну картку, та повертаємо вже новий масив з модифікованими картками:

```
const updatedTickets = columns
  .map((item) => {
```

Після чого, ми шукаємо ту саму перетягнуту карту и змінюємо ее позицію та колонку:

```
    if (item._id === draggableId) {
      return { ...item, columnId: destDroppableId, index: destIndex };
    }
  }
```

Наступною перевіркою ми шукаємо картки, які належать до колонки з категорією, з якої картка була видалена, та міняємо позиції на одну менше тих карток, які по позиції більше, ніж та, яку перемістили:

```
    if (item.columnId === srcDroppableId && item.index > srcIndex) {
      return { ...item, index: item.index - 1 };
    }
  }
```

Остання перевірка шукає картки, які належать колонці, в яку була переміщена поточного картка, і змінює позиції цих карток на одну більше:

```
if (
    item.columnId === destDroppableId &&
    item._id !== draggableId &&
    item.index >= destIndex
) {
    return { ...item, index: item.index + 1 };
}
```

І якщо всі перевірки або виконалися, або не задовольнилися, то в кінці просто повертається елемент у своєму незмінному стані:

```
return item;
```

Як тільки новий масив був створений з зміненими елементами, після цього робиться сортування позицій карток по спадаючій:

```
}).sort((a, b) => a.index - b.index);
```

Наступною дією відправляється інформація з новим масивом карток в сховище, і відразу ж після цього виконується запит до серверу з цим масивом:

```
dispatch({
    type: SET_CARD_DESTINATION,
    payload: { modifiedColumns: updatedTickets },
});
```

```
const headers = getFetchHeaders ();
const body = JSON.stringify ({
  projectId,
  updatedTickets,
});

await request ('/ api / ticket / position', 'PATCH', body, headers);
```

І кінцевий успішний результат, по виконанню успішного запиту на сервер, передається у сховище:

```
dispatch ({
  type: SET_CARD_DESTINATION_SUCCESS,
});
```

Весь Flux механізм взаємодії (рис. 3.2) з браузером і базою даних у вебзастосунку можна уявити у вигляді односпрямованого потоку:

- користувач через браузер звертається до уявлень;
- уявлення у свій час відстежують взаємодії користувача;
- на результатах взаємодії створюється дія, в котрій йде звернення до баз даних на основі використання ORM;
- база даних повертає дані;
- дія перенаправляє отримані дані до диспетчеру;
- диспетчер викликає коллбек;
- коллбек в свою чергу отримані дані віддає сховищу;
- сховище приймає дані і змінює їх або додає, після чого дає сигнал зміну діям;
- зміну дій вловлює уявлення, а воно вже безпосередньо дає браузеру зрозуміти, що змінилося.

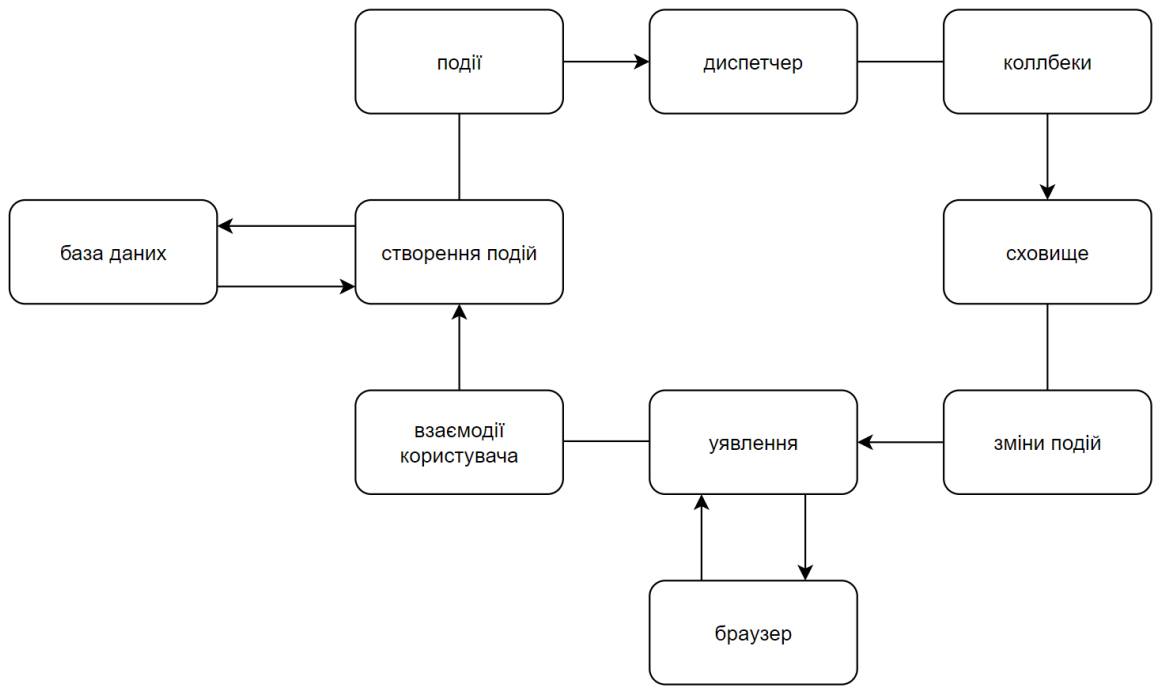


Рисунок 3.2 – Схема односпрямованого потоку взаємодії Flux

3.3 Структура бази даних

База даних MongoDB є основним джерелом отримання необхідної інформації у вебзастосунку, а також засобом для зберігання і зміни необхідних даних. Уся інформації у БД поділена на таблиці, які є сутностями того чи іншого бізнес-об'єкта або процесу.

Короткий опис таблиць БД зазначено у таблиці 3.1.

Таблиця 3.1 – Опис таблиць БД

Назва таблиці у БД	Призначення таблиці
Projects	Список створених проектів
Tickets	Список створених карток проектів
Columns	Колонки категорій для карток

Щоб зрозуміти, які функції виконує кожна таблиця БД, розглянемо детальніше їх.

Таблиця Projects представляє створені проекти, які безпосередньо пов'язані з картками, які були створені у проектах з наступними полями:

- `_id` – унікальний номер кожного проекту;
- `name` – назва проекту;
- `key` – ключ, за яким класифікуються та нумеруються картки;
- `createdAt` – дата створення проекту.

Таблиця Tickets представляє кожен картку певного проекту, яка зберігає такі поля:

- `_id` – унікальний номер кожної картки;
- `descr` – опис картки;
- `projectId` – номер проекту, у якому карта створена;
- `title` – заголовок картки;
- `index` – місце знаходження картки у певній категорії;
- `keyNumber` – нумерація за загальною кількістю карток;
- `columnId` – номер категорії, за якою картка закріплена.

Таблиця Columns представляє список колонок категорій, в яких розміщені картки.

Таблиця включає поля:

- `_id` – унікальний номер категорії;
- `title` – назва категорії.

Схема БД представлена на рис. 3.3.

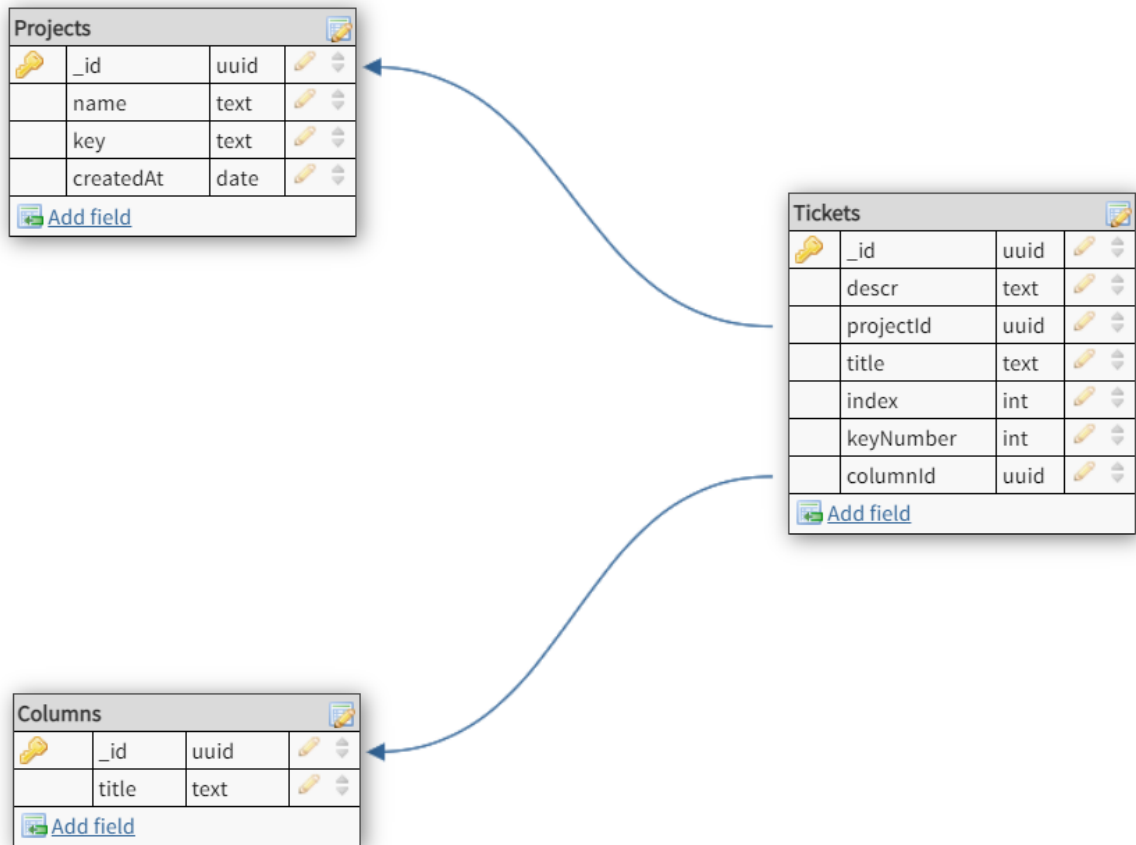


Рисунок 3.3 – Схема БД

3.4 Висновки за розділом 3

В даному розділі було визначено модель архітектури роботи з даними, архітектура відображення даних, структура БД і функціональне поведінка вебзастосунку.

4 КЕРІВНИЦТВО КОРИСТУВАЧА І ПРОГРАМІСТА

4.1 Призначення програми

Призначення програми є організація робочого процесу і можливість оптимізувати ланцюжок планування виробничих потужностей, починаючи від прогнозу попиту, планування виробничих завдань і балансування цих завдань за виробничими потужностями з оптимізацією їх завантаження.

4.2 Умови виконання програми

Програмне забезпечення складається з 2 частин: серверна частина і клієнтська.

Клієнтська частина відповідає за відображення візуальної частини програми і даних, які прийшли з серверної частини, і розроблена на мові JavaScript з розширенням TypeScript та використанням фреймворку React.js.

Серверна частина відповідає за зберігання даних і передачу їх клієнтській частині, та розроблена на програмній платформі Node.js, в основі якої лежить JavaScript, та фреймворком для неї Express.js разом з TypeScript.

Для роботи клієнтської частини потрібно мати доступ до сучасних браузерів:

- Firefox;
- Opera;
- Chrome;
- Microsoft Edge;
- Internet Explorer.

Для роботи серверної частини, потрібно мати доступ до мережі інтернет і вимоги:

- двохядерний процесор або більше;
- оперативна пам'ять 1 ГБ або більше.

4.3 Характеристики програми

Розроблений вебзастосунок повинен забезпечувати такі функціональні характеристики:

- створення нового проекту;
- перегляд створених проектів;
- зміна, видалення створених проектів;
- завантаження обраного проекту;
- створення нової картки проекту;
- зміна, видалення картки проекту;
- перегляд списку створених карток проекту;
- переміщення карток всередині певної категорії;
- переміщення карток між різними категоріями.

4.4 Звернення до програми

Щоб запустити сервер, необхідно:

- а) зареєструватися на сайті ресурсу документоорієнтованої системи управління базами даних MongoDB;
- б) створити проект;
- в) створити кластер (хост бази даних);
- г) в конфігураціях кластера знайти рядок підключення БД і скопіювати;
- г) розмістити папку сервера в будь-яку директорію комп'ютера;
- д) створити файл конфігурації і ввести такі змінні:
 - API_PORT – порт, на якому буде запущено сервер;
 - DEV_CLIENT_URL – адреса клієнтської частини;
 - MONGO_URI – рядок підключення БД, який скопіювали кроками вище;

е) далі необхідно встановити на комп'ютер програмну платформу Node.js;

є) відкрити консоль всередині папки сервера і ввести наступні команди:

- npm i – встановити залежні пакети сервера;
- npm start – запустити проект.

Для роботи клієнтської частини, необхідно:

а) розмістити папку клієнта в будь-яку директорію комп'ютера;

б) створити файл конфігурації і ввести такі змінні:

- REACT_APP_API_URL – адреса серверної частини;

в) після чого відкрити консоль всередині папки клієнта і ввести наступні команди:

- npm i – встановити залежні пакети;
- npm start – запустити проект;
- відкрити в браузері адресу клієнтської частини.

4.5 Вхідні та вихідні дані

Вхідними даними клієнтської частини є набір тексту для створення, зміни проекту, картки. Вихідними даними клієнтської частини є результат обробленого UI.

Вхідними даними серверної частини є тіло запитів різного типу:

- GET – запит для отримання даних;
- POST – запит для відправки даних;
- PATCH – запит для поновлення об'єкта;
- DELETE – запит для видалення об'єкта.

Вихідними даними серверної частини є відповідь запитів з даними.

4.6 Повідомлення користувачу

Вебзастосунок дає стабільну можливість організації робочого процесу і при різних діях видає деякі сповіщення для користувача і програміста, наприклад: при створенні проекту (рис. 4.1), при створенні картки (рис. 4.2), при видаленні проекту (рис. 4.3).

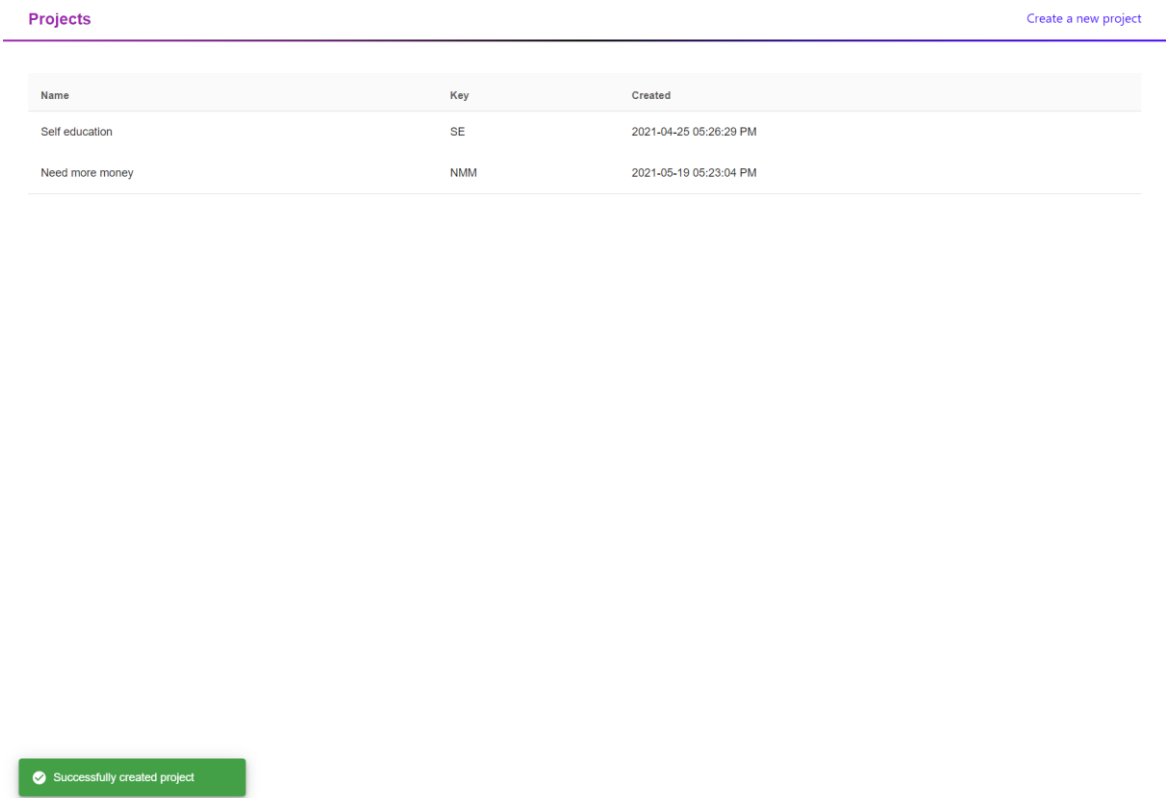


Рисунок 4.1 – Повідомлення при створенні проекту

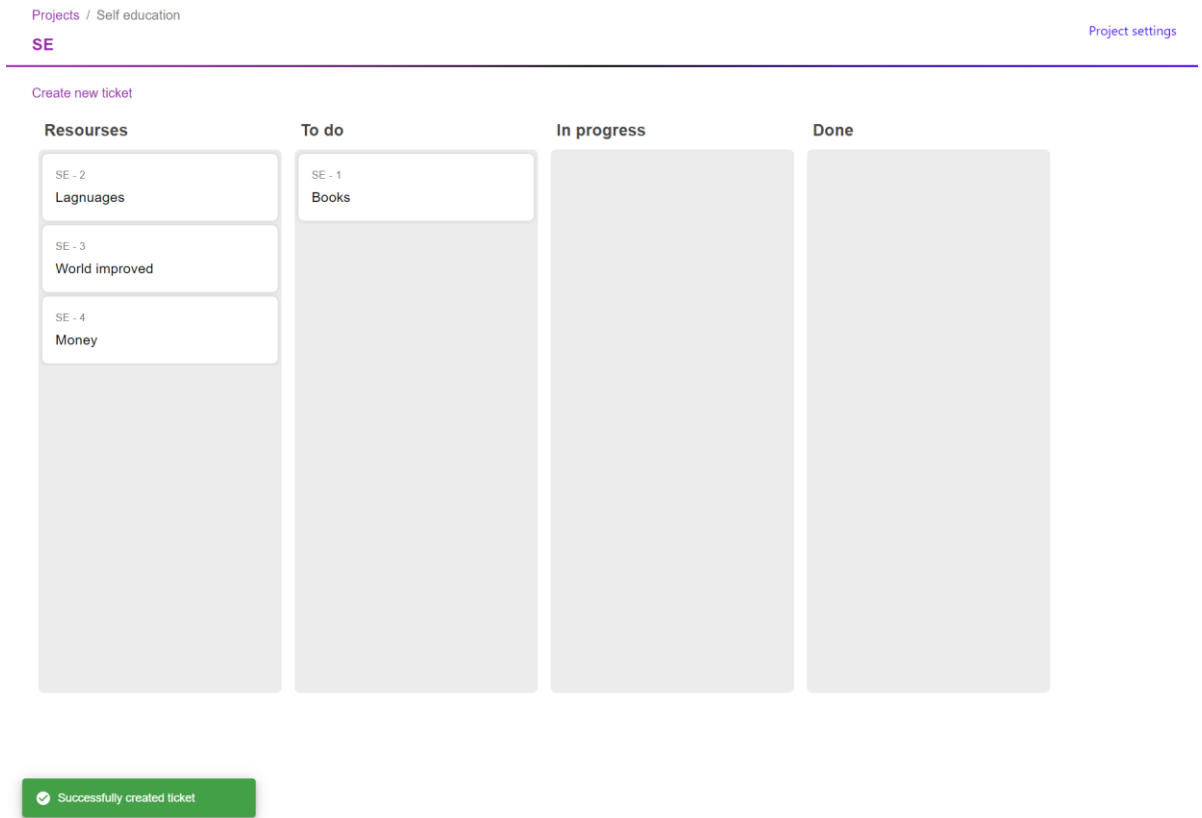


Рисунок 4.2 – Повідомлення при створенні картки

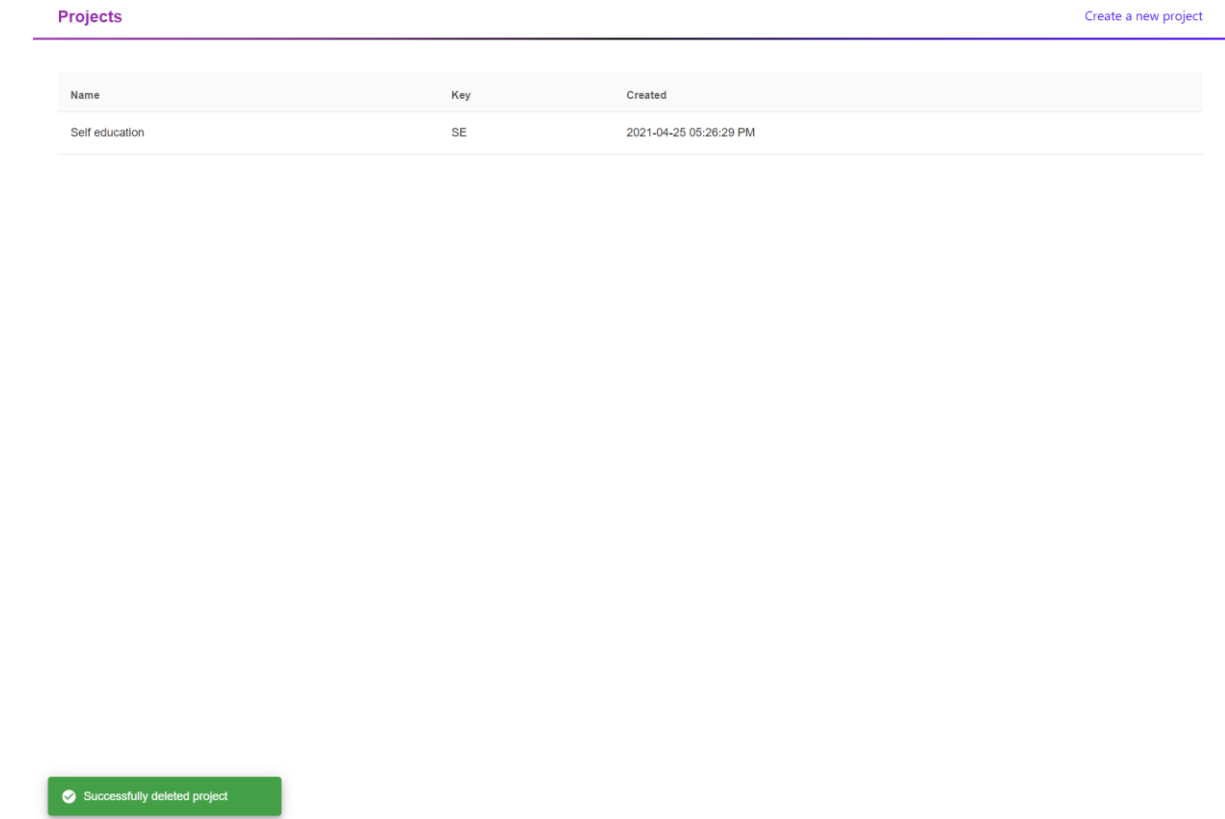


Рисунок 4.3 – Повідомлення при видаленні проекту

Також є сповіщення при видаленні картки (рис. 4.4) та при незнайденому проекті (рис. 4.5).

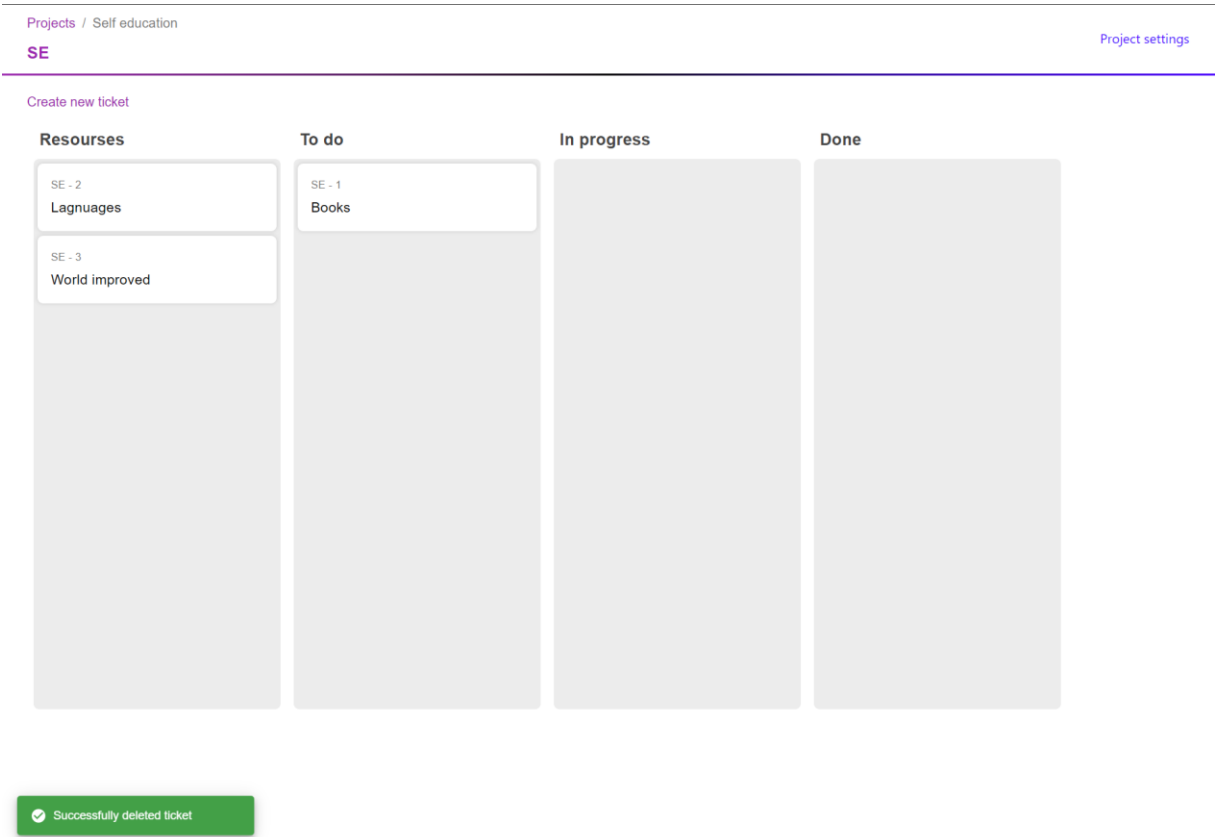


Рисунок 4.4 – Повідомлення при видаленні картки

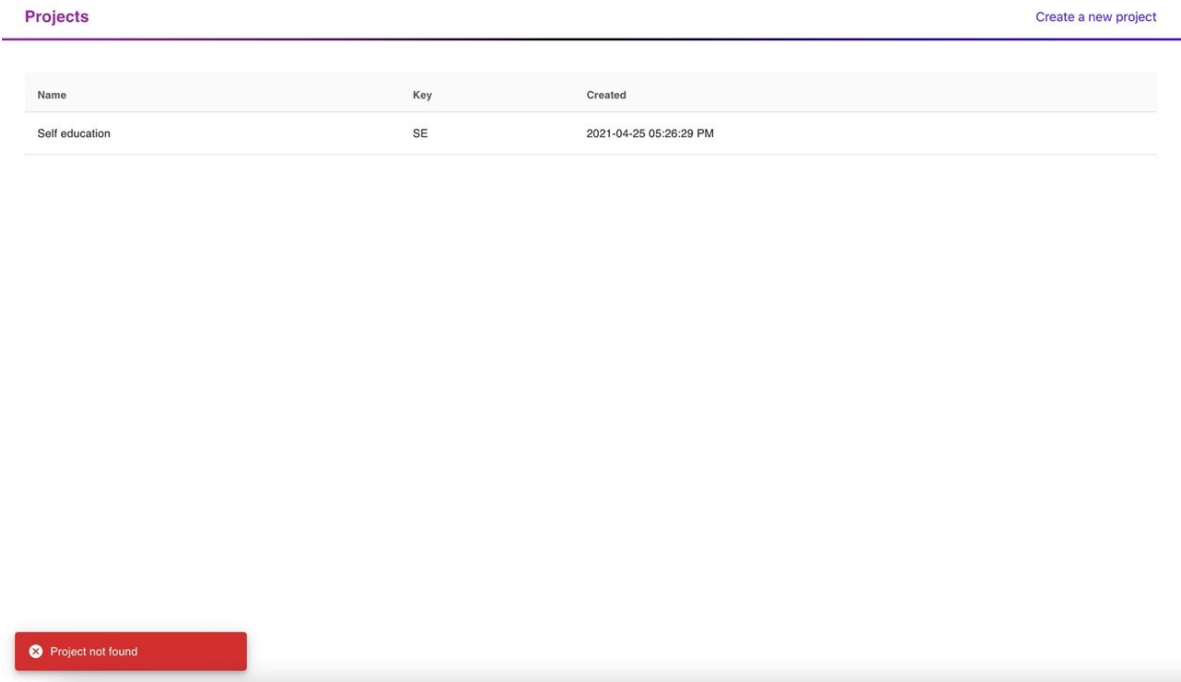


Рисунок 4.5 – Повідомлення, коли проект було не знайдено

Більш детальніше про сповіщення вебзастосунку позначено у таблиці 4.1.

Таблиця 4.1 – Повідомлення програмісту і користувачеві

Повідомлення	Опис	Дії користувача	Дії програміста
Successfully created	Проект був успішно створений	Перевірити на присутність нового проекту в списку проектів	Перевірити у БД наявність нового проекту
Successfully deleted	Успішно видалений проект	Перевірити у списку проектів відсутність видаленого проекту	Перевірити у БД відсутність видаленого проекту
Successfully created	Вдало створена картка	Перевірити у списку карток в проекті наявність нової картки	Перевірити у БД наявність нової картки
Successfully deleted	Вдало видалена картка	Перевірити у списку карток відсутність видаленої картки	Перевірити у БД відсутність видаленої картки
Successfully updated	Картка успішно оновлена	Перевірити оновлення картки	Перевірити у БД оновлена картка чи ні
Successfully updated	Проект успішно оновлен	Перевірити оновлення проекту	Перевірити у БД оновлення полів проекту

Продовження таблиці 4.1

Повідомлення	Опис	Дії користувача	Дії програміста
Something went	Помилка в якомусь запиті	Оновити сторінку	Перевірити статус і результат помилки у інспекторі браузеру
Project not found	Не було знайдено поточного проекту	Вибрати інший існуючий проект або створити новий	Перевірити у БД існування номера проекту, за яким йде запит
Ticket not found	Не було знайдено поточної картки	Відкрити іншу картку чи створити нову	Перевірити у БД існування номера картки, за якою йде запит

ВИСНОВКИ

В результаті виконання дипломної кваліфікаційної роботи бакалавра виконано аналіз проблеми організації та планування робочого процесу і детальний огляд веб-ресурсів, які використовуються як великими компаніями, так і людьми, які потребують поліпшення ведення своїх проектів тощо.

Вебзастосунок розроблений на мовах програмування JavaScript, TypeScript, React.js для клієнтської частини і Node.js, JavaScript, Express.js, TypeScript для серверної частини, а також система керування базами даних MongoDB.

Виконано проектування вебзастосунку на основі створення діаграм дій користувача, послідовностей, архітектури серверної та клієнтської частин, схеми бази даних.

Створений вебзастосунок дозволяє керувати організацією проекту, як окремий проект, з призначеними завданнями, до яких можна додати повноцінний опис з форматуванням для загального розуміння, та вимогами, які можна призначати на різні категорії виконання.

Вебзастосунок має підключене доменне ім'я та налаштований хостинг і працює стабільно у реальному часі.

Всі завдання, поставлені у дипломній кваліфікаційній роботі бакалавра, були повністю виконані.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Trello [Electronic resource]. – Access mode : <https://trello.com/en>.
2. Kanban [Electronic resource]. – Access mode : <https://en.wikipedia.org/wiki/Kanban>.
3. Trello pricing [Electronic resource]. – Access mode : <https://trello.com/ru/pricing>.
4. Trello enterprise [Electronic resource]. – Access mode : <https://trello.com/enterprise>.
5. Miro [Electronic resource]. – Access mode: <https://miro.com/index/>.
6. Miro pricing [Electronic resource]. – Access mode: <https://miro.com/pricing/>.
7. Center for Strategic & International Studies [Electronic resource]. – Access mode: <https://www.csis.org/>.
8. An article on cyber attacks by Chinese hackers [Electronic resource]. – Access mode: <https://www.csis.org/programs/strategic-technologies-program/significant-cyber-incidents>.
9. JavaScript [Electronic resource]. – Access mode : <https://en.wikipedia.org/wiki/JavaScript>.
10. Eloquent JavaScript [Electronic resource]. – Access mode : <https://eloquentjavascript.net/>.
11. VBScript [Electronic resource]. – Access mode : <https://en.wikipedia.org/wiki/VBScript>.
12. VBScript in a Nutshell [Electronic resource]. – Access mode : <https://www.oreilly.com/library/view/vbscript-in-a/1565927206/>.
13. React.js [Electronic resource]. – Access mode : [https://en.wikipedia.org/wiki/React_\(JavaScript_library\)](https://en.wikipedia.org/wiki/React_(JavaScript_library)).

14. Learn React.js [Electronic resource]. – Access mode :
<https://reactjs.org/docs/getting-started.html>.
15. Angular.js [Electronic resource]. – Access mode :
[https://en.wikipedia.org/wiki/Angular_\(web_framework\)](https://en.wikipedia.org/wiki/Angular_(web_framework)).
16. Learn Angular.js [Electronic resource]. – Access mode :
<https://angular.io/docs>.
17. Google [Electronic resource]. – Access mode :
<https://en.wikipedia.org/wiki/Google>.
18. Vue.js [Electronic resource]. – Access mode :
<https://en.wikipedia.org/wiki/Vue.js>.
19. Learn Vue.js [Electronic resource]. – Access mode :
<https://vuejs.org/v2/guide/>.
20. IntelliSense [Electronic resource]. – Access mode :
<https://code.visualstudio.com/docs/editor/intellisense>.
21. Node.js [Electronic resource]. – Access mode :
<https://nodejs.org/en/about/>.
22. Express.js [Electronic resource]. – Access mode :
<http://expressjs.com/>.
23. Nest.js [Electronic resource]. – Access mode : <https://nestjs.com/>.
24. MongoDB [Electronic resource]. – Access mode :
<https://www.mongodb.com/>.
25. AWS [Electronic resource]. – Access mode :
https://en.wikipedia.org/wiki/Amazon_Web_Services.
26. Azure [Electronic resource]. – Access mode :
https://en.wikipedia.org/wiki/Microsoft_Azure.
27. Google Cloud [Electronic resource]. – Access mode :
https://en.wikipedia.org/wiki/Google_Cloud_Platform.

ДОДАТОК А
Текст програми

A.1 Текст файла store/kanban/action.ts

```
import handleError from 'utils/actionErrors';
import getFetchHeaders from 'utils/getFetchHeaders';

import request from 'services/axios';

import { RootThunkAction, ProviderContextNotistack } from
'store/types';

import {
  SET_CARD_DESTINATION,
  SET_CARD_DESTINATION_REQUEST,
  SET_CARD_DESTINATION_SUCCESS,
  CREATE_NEW_TICKET_REQUEST,
  CREATE_NEW_TICKET_SUCCESS,
  GET_TICKETS_REQUEST,
  GET_TICKETS_SUCCESS,
  SET_TEXT_EDITOR_DESCR,
  CHANGE_CURRENT_TICKET_REQUEST,
  CHANGE_CURRENT_TICKET_SUCCESS,
  DELETE_CURRENT_TICKET_REQUEST,
  DELETE_CURRENT_TICKET_SUCCESS,
  KANBAN_FAILURE,
  ParamsChangeCardColumn,
  ParamsChangeCardPosition,
  ParamsCreateNewTicket,
  ParamsGetTickets,
  ColumnsType,
  ChangeTexEditorValueProps,
  ChangeCurrentTicketValueProps,
  DeleteCurrentTicketProps,
} from './types';

export const changeCardColumn = ({
  projectId,
  draggableId,
  srcIndex,
  destIndex,
  srcDraggableId,
  destDraggableId,
  enqueueSnackbar,
}: ParamsChangeCardColumn & ProviderContextNotistack): RootThunkAction
=> async (
  dispatch,
  getState
) => {
  try {
    dispatch({
      type: SET_CARD_DESTINATION_REQUEST,
```

```

    });

    const { columns } = getState().kanban;

    const updatedTickets = columns
      .map((item) => {
        // looking for the same dragged card and change its position
        and board
        if (item._id === draggableId) {
          return { ...item, columnId: destDroppableId, index:
destIndex };
        }

        // looking for cards belonging to the board from which the
        card
        // was removed, in order to change their positions. Then
        change positions
        // by one less of all those cards that are larger than the
        index of the
        // card that was removed.
        if (item.columnId === srcDroppableId && item.index > srcIndex)
        {
          return { ...item, index: item.index - 1 };
        }

        // looking for cards that belong to the board to which the
        card
        // was moved and change these cards to one more position
        if (
          item.columnId === destDroppableId &&
          item._id !== draggableId &&
          item.index >= destIndex
        ) {
          return { ...item, index: item.index + 1 };
        }

        return item;
      })
      .sort((a, b) => a.index - b.index);

    dispatch({
      type: SET_CARD_DESTINATION,
      payload: { modifiedColumns: updatedTickets },
    });

    const headers = getFetchHeaders();
    const body = JSON.stringify({
      projectId,
      updatedTickets,
    });
  });

```

```

    await request('/api/ticket/position', 'PATCH', body, headers);

    dispatch({
      type: SET_CARD_DESTINATION_SUCCESS,
    });
  } catch (error) {
    const errors = handleErrors(error);

    errors.map(({ msg, severity }) =>
      enqueueSnackbar(msg, {
        variant: severity,
      })
    );

    dispatch({
      type: KANBAN_FAILURE,
    });
  }
};

export const changeCardPosition = ({
  projectId,
  draggableId,
  srcIndex,
  destIndex,
  srcDroppableId,
  enqueueSnackbar,
}: ParamsChangeCardPosition & ProviderContextNotistack):
RootThunkAction => async (
  dispatch,
  getState
) => {
  try {
    dispatch({
      type: SET_CARD_DESTINATION_REQUEST,
    });

    const { columns } = getState().kanban;

    const updatedTickets = columns
      .map((item) => {
        if (item.columnId === srcDroppableId && item._id ===
draggableId) {
          return { ...item, index: destIndex };
        }

        if (
          destIndex > srcIndex &&
          item.columnId === srcDroppableId &&
          item._id !== draggableId &&
          item.index >= srcIndex + 1 &&

```

```

        item.index <= destIndex
    ) {
        return { ...item, index: item.index - 1 };
    }

    if (
        destIndex < srcIndex &&
        item.columnId === srcDroppableId &&
        item._id !== draggableId &&
        item.index <= srcIndex - 1 &&
        item.index >= destIndex
    ) {
        return { ...item, index: item.index + 1 };
    }

    return item;
})
.sort((a, b) => a.index - b.index);

dispatch({
  type: SET_CARD_DESTINATION,
  payload: {
    modifiedColumns: updatedTickets,
  },
});

const headers = getFetchHeaders();
const body = JSON.stringify({
  projectId,
  updatedTickets,
});

await request('/api/ticket/position', 'PATCH', body, headers);

dispatch({
  type: SET_CARD_DESTINATION_SUCCESS,
});
} catch (error) {
  const errors = handleErrors(error);

  errors.map(({ msg, severity }) =>
    enqueueSnackbar(msg, {
      variant: severity,
    })
  );
};

dispatch({
  type: KANBAN_FAILURE,
});
}
};

```

```

export const createNewTicket = ({
  projectId,
  title,
  descr,
  columnId,
  index,
  keyNumber,
  enqueueSnackbar,
}: ParamsCreateNewTicket & ProviderContextNotistack): RootThunkAction
=> async (dispatch) => {
  try {
    dispatch({
      type: CREATE_NEW_TICKET_REQUEST,
    });

    const headers = getFetchHeaders();
    const body = JSON.stringify({
      projectId,
      title,
      descr,
      columnId,
      index,
      keyNumber,
    });
    const { msg, severity } = await request('/api/ticket', 'POST',
body, headers);

    enqueueSnackbar(msg, {
      variant: severity,
    });

    dispatch({
      type: CREATE_NEW_TICKET_SUCCESS,
    });
  } catch (error) {
    const errors = handleErrors(error);

    errors.map(({ msg, severity }) =>
      enqueueSnackbar(msg, {
        variant: severity,
      })
    );

    dispatch({
      type: KANBAN_FAILURE,
    });
  }
};

export const getTickets = ({

```

```

    projectId,
    enqueueSnackbar,
  }: ParamsGetTickets & ProviderContextNotistack): RootThunkAction =>
  async (dispatch) => {
    try {
      dispatch({
        type: GET_TICKETS_REQUEST,
      });

      const headers = getFetchHeaders();
      const tickets = await request(`/api/ticket/${projectId}`, 'GET',
        null, headers);

      dispatch({
        type: GET_TICKETS_SUCCESS,
        payload: { tickets: tickets.sort((a: ColumnsType, b:
ColumnsType) => a.index - b.index) },
      });
    } catch (error) {
      const errors = handleErrors(error);

      errors.map(({ msg, severity }) =>
        enqueueSnackbar(msg, {
          variant: severity,
        })
      );

      dispatch({
        type: KANBAN_FAILURE,
      });
    }
  };

export const changeTexEditorValue = ({
  textEditorValue,
}: ChangeTexEditorValueProps): RootThunkAction => async (dispatch) =>
{
  dispatch({
    type: SET_TEXT_EDITOR_DESCR,
    payload: { textEditorValue },
  });
};

export const changeCurrentTicketValue = ({
  projectId,
  ticketId,
  title,
  descr,
  enqueueSnackbar,
}: ChangeCurrentTicketValueProps & ProviderContextNotistack):
RootThunkAction => async (

```

```

    dispatch
  ) => {
    try {
      dispatch({
        type: CHANGE_CURRENT_TICKET_REQUEST,
      });

      const headers = getFetchHeaders();
      const body = JSON.stringify({
        projectId,
        ticketId,
        title,
        descr,
      });
      const { msg, severity } = await request('/api/ticket', 'PATCH',
body, headers);

      enqueueSnackbar(msg, {
        variant: severity,
      });

      dispatch({
        type: CHANGE_CURRENT_TICKET_SUCCESS,
      });
    } catch (error) {
      const errors = handleErrors(error);

      errors.map(({ msg, severity }) =>
        enqueueSnackbar(msg, {
          variant: severity,
        })
      );

      dispatch({
        type: KANBAN_FAILURE,
      });
    }
  };
};

export const deleteCurrentTicket = ({
  projectId,
  ticketId,
  enqueueSnackbar,
}: DeleteCurrentTicketProps & ProviderContextNotistack):
RootThunkAction => async (dispatch) => {
  try {
    dispatch({
      type: DELETE_CURRENT_TICKET_REQUEST,
    });

    const headers = getFetchHeaders();

```

```

    const body = JSON.stringify({
      projectId,
      ticketId,
    });
    const { msg, severity } = await request('/api/ticket', 'DELETE',
body, headers);

    enqueueSnackbar(msg, {
      variant: severity,
    });

    dispatch({
      type: DELETE_CURRENT_TICKET_SUCCESS,
    });
  } catch (error) {
    const errors = handleErrors(error);

    errors.map(({ msg, severity }) =>
      enqueueSnackbar(msg, {
        variant: severity,
      })
    );

    dispatch({
      type: KANBAN_FAILURE,
    });
  }
};

```

A.2 Текст файлу containers/Kanban/Kanban.tsx

```

import { useEffect, useState, useCallback } from 'react';
import { withRouter, RouteComponentProps, useHistory } from 'react-
router-dom';
import { useDispatch, useSelector } from 'react-redux';
import { DragDropContext, DropResult } from 'react-beautiful-dnd';
import { useSnackbar } from 'notistack';
import { useFormik } from 'formik';
import * as yup from 'yup';

import Modal from '@material-ui/core/Modal';
import Backdrop from '@material-ui/core/Backdrop';
import Fade from '@material-ui/core/Fade';
import { makeStyles, createStyles } from '@material-ui/core/styles';

import Loader from 'shared/Loader/Loader';

import KanbanColumn from 'components/KanbanColumn/KanbanColumn';
import KanbanColumnContent from
'components/KanbanColumnContent/KanbanColumnContent';

```

```

import NewTicketContent from
'components/NewTicketContent/NewTicketContent';

import { kanbanTables } from 'assets/code/kanbanTables';

import { RootState } from 'store/types';
import {
  changeCardColumn,
  changeCardPosition,
  changeTexEditorValue,
  createNewTicket,
  getTickets,
} from 'store/kanban/action';
import { clearProjectErrors, getOneProject } from
'store/projects/action';
import { initialStateKanban } from 'store/kanban/reducer';
import { TicketDescrType } from 'store/kanban/types';

import { RouteInfo, FormikParamsNewTicket } from './types';

import './Kanban.scss';

const useStyles = makeStyles(() =>
  createStyles({
    modal: {
      display: 'flex',
      alignItems: 'center',
      justifyContent: 'center',
    },
  })
);

const KanbanContainer: React.FC<RouteComponentProps<RouteInfo>> = ({
match }) => {
  const classes = useStyles();
  const { enqueueSnackbar } = useSnackbar();
  const history = useHistory();
  const dispatch = useDispatch();

  const { columns, ticketDescr } = useSelector((state: RootState) =>
state.kanban);

  const { loading: loadingProject, error, currentProject } =
useSelector(
  (state: RootState) => state.projects
);

  const [fetchIsEndCurrentProject, setFetchIsEndCurrentProject] =
useState(false);
  const [fetchIsEndProjectTickets, setFetchIsEndProjectTickets] =
useState(false);

```

```

const [open, setOpen] = useState(false);

const handleSubmit = async ({ title }: FormikParamsNewTicket) => {
  setFetchIsEndProjectTickets(false);

  const index = columns.filter((item) => item.columnId ===
kanbanTables[0].id).length;

  await dispatch(
    createNewTicket({
      projectId: match.params.projectId,
      title,
      index,
      columnId: kanbanTables[0].id,
      descr: ticketDescr,
      keyNumber: columns.length + 1,
      enqueueSnackbar,
    })
  );
  setFetchIsEndProjectTickets(true);

  formik.resetForm();
  handleCloseModal();
  fetchProjectTickets();
};

const validationSchema = yup.object({
  title: yup.string().required('Title is required').trim().max(70),
});

const formik = useFormik({
  initialValues: {
    title: '',
  },
  validationSchema,
  onSubmit: handleSubmit,
});

const fetchCurrentProject = useCallback(async () => {
  if (!fetchIsEndCurrentProject && !error) {
    await dispatch(getOneProject({ id: match.params.projectId,
enqueueSnackbar }));

    setFetchIsEndCurrentProject(true);
  }
}, [dispatch, enqueueSnackbar, fetchIsEndCurrentProject, error,
match.params.projectId]);

const fetchProjectTickets = useCallback(async () => {
  if (fetchIsEndCurrentProject && !error) {

```

```

    await dispatch(getTickets({ projectId: match.params.projectId,
enqueueSnackbar }));

    setFetchIsEndProjectTickets(true);
  }
}, [dispatch, enqueueSnackbar, match.params.projectId,
fetchIsEndCurrentProject, error]);

useEffect(() => {
  fetchCurrentProject();
  fetchProjectTickets();
}, [fetchCurrentProject, fetchProjectTickets]);

useEffect(() => {
  if (error) {
    dispatch(clearProjectErrors());

    history.push('/projects');
  }
}, [error, history, dispatch]);

const onDragEnd = async (result: DropResult) => {
  if (!result.destination) return;

  const { source, destination, draggableId } = result;

  if (source.droppableId !== destination.droppableId) {
    dispatch(
      changeCardColumn({
        projectId: match.params.projectId,
        draggableId,
        srcIndex: source.index,
        destIndex: destination.index,
        srcDroppableId: source.droppableId,
        destDroppableId: destination.droppableId,
        enqueueSnackbar,
      })
    );
  } else {
    dispatch(
      changeCardPosition({
        projectId: match.params.projectId,
        draggableId,
        srcIndex: source.index,
        destIndex: destination.index,
        srcDroppableId: source.droppableId,
        enqueueSnackbar,
      })
    );
  }
};

```

```

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

const handleCloseModal = () => {
  dispatch(changeTexEditorValue({ textEditorValue:
initialStateKanban.ticketDescr }));
  formik.resetForm();

  setOpen(false);
};

const handleTikectClick = (ticketId: string) => {
history.push(`/projects/${match.params.projectId}/ticket/${ticketId}`)
;
};

const handleChange = (textEditorValue: TicketDescrType) => {
  dispatch(changeTexEditorValue({ textEditorValue }));
};

if (
  !fetchIsEndCurrentProject ||
  !fetchIsEndProjectTickets ||
  loadingProject ||
  !currentProject._id
) {
  return <Loader />;
}

return (
  <div className="kanban__wrapper shared--wrapper">
    <button type="button" className="link link--purple"
onClick={handleOpenModal}>
      Create new ticket
    </button>

    <div className="kanban__content">
      <DragDropContext onDragEnd={onDragEnd}>
        {kanbanTables.map(({ id, name }) => (
          <KanbanColumn key={id} columnId={id} name={name}>
            <KanbanColumnContent
              columnId={id}
              columns={columns}
              keyProject={currentProject.key}
              handleTikectClick={handleTikectClick}
            />
          </KanbanColumn>
        ))}
      </DragDropContext>
    </div>
  </div>
)

```

```

    </DragDropContext>
  </div>

  <Modal
    aria-labelledby="transition-modal-title"
    aria-describedby="transition-modal-description"
    className={classes.modal}
    open={open}
    onClose={handleCloseModal}
    closeAfterTransition
    BackdropComponent={Backdrop}
    BackdropProps={{
      timeout: 500,
    }}
  >
    <Fade in={open}>
      <div className="new-ticket__inner">
        <NewTicketContent
          descrTicket={ticketDescr}
          keyProject={currentProject.key}
          handleClose={handleCloseModal}
          handleChange={handleChange}
          formik={formik}
        />
      </div>
    </Fade>
  </Modal>
</div>
);
};

export default withRouter(KanbanContainer);

```

A.3 Текст файла containers/ProjectSettings/ProjectSettings.tsx

```

import { useEffect, useState } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { withRouter, RouteComponentProps, useHistory } from 'react-router-dom';
import { useSnackbar } from 'notistack';
import { useFormik } from 'formik';
import * as yup from 'yup';

import Modal from '@material-ui/core/Modal';
import Backdrop from '@material-ui/core/Backdrop';
import Fade from '@material-ui/core/Fade';
import { makeStyles, createStyles } from '@material-ui/core/styles';

import Loader from 'shared/Loader/Loader';

```

```

import ProjectSettingsContent from
'components/ProjectSettingsContent/ProjectSettingsContent';

import { RouteInfo } from 'containers/Kanban/types';
import { FormikParamsNewProject } from 'containers/NewProject/types';

import { RootState } from 'store/types';
import {
  clearProjectErrors,
  getOneProject,
  updateProject,
  deleteCurrentProject,
} from 'store/projects/action';
import ModalDelete from 'shared/ModalDelete/ModalDelete';

import './ProjectSettings.scss';

const useStyles = makeStyles(() =>
  createStyles({
    modal: {
      display: 'flex',
      alignItems: 'center',
      justifyContent: 'center',
    },
  })
);

const ProjectSettings: React.FC<RouteComponentProps<RouteInfo>> = ({
  match }) => {
  const dispatch = useDispatch();
  const history = useHistory();
  const { enqueueSnackbar } = useSnackbar();
  const classes = useStyles();

  const { currentProject, loading, error } = useSelector((state:
  RootState) => state.projects);

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

  useEffect(() => {
    if (error) {
      dispatch(clearProjectErrors());

      history.push('/projects');
    }
  }, [error, history, dispatch]);

  useEffect(() => {
    if (!currentProject._id) {

```

```

    dispatch(getOneProject({ id: match.params.projectId,
enqueueSnackbar }));
  }
}, [dispatch, enqueueSnackbar, match.params.projectId,
currentProject._id]);

const handleSumbit = async ({ name, key }: FormikParamsNewProject)
=> {
  if (name !== currentProject.name || key !== currentProject.key) {
    await dispatch(
      updateProject({ id: match.params.projectId, name, key:
key.toUpperCase(), enqueueSnackbar })
    );
  }

  history.push(`/projects/${match.params.projectId}`);
};

const validationSchema = yup.object({
  name: yup.string().required('Name is required').trim().max(25),
  key: yup.string().required('Key is required (for example: KISS,
sas, Gign)').trim().max(9),
});

const formik = useFormik({
  initialValues: {
    name: currentProject.name,
    key: currentProject.key,
  },
  validationSchema,
  onSubmit: handleSumbit,
});

useEffect(() => {
  if (currentProject._id && !formik.values.name) {
    formik.setValues({ name: currentProject.name, key:
currentProject.key });
  }
  // eslint-disable-next-line react-hooks/exhaustive-deps
}, [currentProject._id, currentProject.name, currentProject.key]);

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

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

const deleteProject = async () => {

```

```

    await dispatch(deleteCurrentProject({ id: match.params.projectId,
enqueueSnackbar }));

    history.push('/projects');
  };

  if (loading || !currentProject._id) {
    return <Loader />;
  }

  return (
    <div className="project-settings__wrapper shared--wrapper">
      <ProjectSettingsContent formik={formik}
handleOpenModal={handleOpenModal} />

      <Modal
        aria-labelledby="transition-modal-title"
        aria-describedby="transition-modal-description"
        className={classes.modal}
        open={openModal}
        onClose={handleCloseModal}
        closeAfterTransition
        BackdropComponent={Backdrop}
        BackdropProps={{
          timeout: 500,
        }}
      >
        <Fade in={openModal}>
          <div className="delete-project__inner">
            <ModalDelete
              fetchDelete={deleteProject}
              cancelDelete={handleCloseModal}
              text="You definitely want to delete this project?"
            />
          </div>
        </Fade>
      </Modal>
    </div>
  );
};

export default withRouter(ProjectSettings);

```

A.4 Текст файла containers/CurrentTicket/CurrentTicket.tsx

```

import { useState, useEffect } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { RouteComponentProps, useHistory, withRouter } from 'react-
router-dom';

```

```

import { useFormik } from 'formik';
import * as yup from 'yup';
import { useSnackbar } from 'notistack';

import Fade from '@material-ui/core/Fade';
import Modal from '@material-ui/core/Modal';
import Backdrop from '@material-ui/core/Backdrop';

import { makeStyles, createStyles } from '@material-ui/core/styles';

import Loader from 'shared/Loader/Loader';

import { RouteInfo } from 'containers/Kanban/types';

import CurrentTicketContent from
'components/CurrentTicketContent/CurrentTicketContent';
import ModalDelete from 'shared/ModalDelete/ModalDelete';

import {
  changeCurrentTicketValue,
  changeTexEditorValue,
  getTickets,
  deleteCurrentTicket,
} from 'store/kanban/action';
import { initialStateKanban } from 'store/kanban/reducer';
import { TicketDescrType } from 'store/kanban/types';
import { RootState } from 'store/types';

import './CurrentTicket.scss';

const useStyles = makeStyles(() =>
  createStyles({
    modal: {
      display: 'flex',
      alignItems: 'center',
      justifyContent: 'center',
    },
  })
);

const CurrentTicket: React.FC<RouteComponentProps<RouteInfo>> = ({
  match }) => {
  const history = useHistory();
  const dispatch = useDispatch();
  const { enqueueSnackbar } = useSnackbar();
  const classes = useStyles();

  const [clickOutsideDown, setClickOutsideDown] = useState(false);
  const [editTextEditor, setEditTextEditor] = useState(false);
  const [updates, setUpdates] = useState(false);
  const [openModal, setOpenModal] = useState(false);

```

```

    const { columns, ticketDescr, loading } = useSelector((state:
RootState) => state.kanban);

    const currentTicket = columns.find((item) => item._id ===
match.params.ticketId);

    const handleSumbit = async ({ title }: { title: string }) => {
        setUpdates(true);

        await dispatch(
            changeCurrentTicketValue({
                projectId: match.params.projectId,
                ticketId: String(currentTicket?._id),
                title,
                descr: ticketDescr,
                enqueueSnackbar,
            })
        );

        await dispatch(getTickets({ projectId: match.params.projectId,
enqueueSnackbar }));

        formik.resetForm();
        changeSetEditTextEditor(false);
        setUpdates(false);
    };

    const validationSchema = yup.object({
        title: yup.string().required('Title is required').trim().max(70),
    });

    const formik = useFormik({
        initialValues: {
            title: String(currentTicket?.title),
        },
        validationSchema,
        onSubmit: handleSumbit,
    });

    useEffect(() => {
        if (currentTicket?._id) {
            formik.setValues({
                title: String(currentTicket?.title),
            });
        }
        // eslint-disable-next-line react-hooks/exhaustive-deps
    }, [currentTicket?._id, updates]);

    const handleClose = () => {
        if (editTextEditor) {

```

```

        dispatch(changeTexEditorValue({ textEditorValue:
initialStateKanban.ticketDescr }));
    }

    history.push(`/projects/${match.params.projectId}`);
};

const handleOutsideUp = () => {
    if (clickOutsideDown) {
        handleClose();
    }
};

const handleOutsideDown = () => {
    setClickOutsideDown(true);
};

const clickDown = (e: React.MouseEvent<HTMLDivElement>) => {
    e.stopPropagation();
};

const handleChange = (descr: TicketDescrType) => {
    dispatch(changeTexEditorValue({ textEditorValue: descr }));
};

const changeSetEditTextEditor = (value: boolean) => {
    if (!value) {
        dispatch(changeTexEditorValue({ textEditorValue:
initialStateKanban.ticketDescr }));

        formik.setValues({
            title: String(currentTicket?.title),
        });
    } else {
        dispatch(changeTexEditorValue({ textEditorValue:
currentTicket?.descr }));
    }

    setEditTextEditor(value);
};

const fetchDeleteTicket = async () => {
    setUpdates(true);

    await dispatch(
        deleteCurrentTicket({
            projectId: match.params.projectId,
            ticketId: String(currentTicket?._id),
            enqueueSnackbar,
        })
    );
};

```

```

    await dispatch(getTickets({ projectId: match.params.projectId,
enqueueSnackbar }));

    setUpdates(false);
    handleClose();
  };

  const deleteTicket = () => {
    handleOpenModal();
  };

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

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

  if (loading || updates) {
    return <Loader />;
  }

  return (
    <>
      <Fade in>
        <div
          className="current-ticket__inner"
          onMouseDown={handleOutsideDown}
          onMouseUp={handleOutsideUp}
        >
          <div className="current__ticket-wrapper"
onMouseDown={clickDown}>
            <CurrentTicketContent
              formik={formik}
              editTextEditor={editTextEditor}
              setEditTextEditor={changeSetEditTextEditor}
              title={String(currentTicket?.title)}
              descr={!editTextEditor ? currentTicket?.descr :
ticketDescr}
              handleClose={handleClose}
              handleChange={handleChange}
              deleteTicket={deleteTicket}
            />
          </div>
        </div>
      </Fade>

      <Modal
        aria-labelledby="transition-modal-title"

```

```

        aria-describedby="transition-modal-description"
        className={classes.modal}
        open={openModal}
        onClose={handleCloseModal}
        closeAfterTransition
        BackdropComponent={Backdrop}
        BackdropProps={{
            timeout: 500,
        }}
    >
    <Fade in={openModal}>
        <div className="delete-ticket__inner">
            <ModalDelete
                fetchDelete={fetchDeleteTicket}
                cancelDelete={handleCloseModal}
                text="You definitely want to delete this ticket?"
            />
        </div>
    </Fade>
</Modal>
</>
);
};

export default withRouter(CurrentTicket);

```

A.5 Текст файла store/projects/actions.ts

```

import request from 'services/axios';
import getFetchHeaders from 'utils/getFetchHeaders';
import handleErrors from 'utils/actionErrors';
import createUpdateProject from 'utils/actionCUPProject';

import { RootThunkAction, ProviderContextNotistack } from
'store/types';

import {
    PROJECTS_LOADING_REQUEST,
    PROJECTS_LOADING_SUCCESS,
    CREATE_NEW_PROJECT_REQUEST,
    CREATE_NEW_PROJECT_SUCCESS,
    UPDATE_PROJECT_REQUEST,
    UPDATE_PROJECT_SUCCESS,
    GET_ONE_PROJECT_REQUEST,
    GET_ONE_PROJECT_SUCCESS,
    DELETE_CURRENT_PROJECT_REQUEST,
    DELETE_CURRENT_PROJECT_SUCCESS,
    CLEAR_ROWS,
    PROJECTS_ONE_PROJECT_FAILURE,
    PROJECTS_FAILURE,
    CLEAR_ERRORS,

```

```

    ParamsCreateProject,
    ParamsUpdateProject,
    ParamsGetOneProject,
  } from './types';

export const getProjects = ({
  enqueueSnackbar,
}: ProviderContextNotistack): RootThunkAction => async (dispatch) => {
  try {
    dispatch({
      type: PROJECTS_LOADING_REQUEST,
    });

    const headers = getFetchHeaders();
    const rows = await request('/api/project', 'GET', null, headers);

    dispatch({
      type: PROJECTS_LOADING_SUCCESS,
      payload: { rows },
    });
  } catch (error) {
    const errors = handleErrors(error);

    errors.map(({ msg, severity }) =>
      enqueueSnackbar(msg, {
        variant: severity,
      })
    );

    dispatch({
      type: PROJECTS_FAILURE,
    });
  }
};

export const createNewProject = ({
  enqueueSnackbar,
  name,
  key,
}: ParamsCreateProject & ProviderContextNotistack): RootThunkAction =>
async (dispatch) => {
  const body = JSON.stringify({ name, key });

  createUpdateProject({
    dispatch,
    dispatchTypeRequest: CREATE_NEW_PROJECT_REQUEST,
    dispatchTypeSuccess: CREATE_NEW_PROJECT_SUCCESS,
    body,
    method: 'POST',
    enqueueSnackbar,
  });
};

```

```

};

export const updateProject = ({
  enqueueSnackbar,
  name,
  key,
  id,
}: ParamsUpdateProject & ProviderContextNotistack): RootThunkAction =>
async (dispatch) => {
  const body = JSON.stringify({ name, key, id });

  createUpdateProject({
    dispatch,
    dispatchTypeRequest: UPDATE_PROJECT_REQUEST,
    dispatchTypeSuccess: UPDATE_PROJECT_SUCCESS,
    body,
    method: 'PATCH',
    enqueueSnackbar,
  });
};

export const getOneProject = ({
  id,
  enqueueSnackbar,
}: ParamsGetOneProject & ProviderContextNotistack): RootThunkAction =>
async (dispatch) => {
  try {
    dispatch({
      type: GET_ONE_PROJECT_REQUEST,
    });

    const headers = getFetchHeaders();
    const project = await request(`/api/project/${id}`, 'GET', null,
headers);

    dispatch({
      type: GET_ONE_PROJECT_SUCCESS,
      payload: { project },
    });
  } catch (error) {
    const errors = handleErrors(error);

    errors.map(({ msg, severity }) =>
      enqueueSnackbar(msg, {
        variant: severity,
      })
    );

    dispatch({
      type: PROJECTS_ONE_PROJECT_FAILURE,
    });
  }
};

```

```

    }
  };

export const deleteCurrentProject = ({
  id,
  enqueueSnackbar,
}: ParamsGetOneProject & ProviderContextNotistack): RootThunkAction =>
  async (dispatch) => {
    try {
      dispatch({
        type: DELETE_CURRENT_PROJECT_REQUEST,
      });

      const headers = getFetchHeaders();
      const body = JSON.stringify({ id });

      const { msg, severity } = await request('/api/project', 'DELETE',
        body, headers);

      enqueueSnackbar(msg, {
        variant: severity,
      });

      dispatch({
        type: DELETE_CURRENT_PROJECT_SUCCESS,
      });
    } catch (error) {
      const errors = handleErrors(error);

      errors.map(({ msg, severity }) =>
        enqueueSnackbar(msg, {
          variant: severity,
        })
      );

      dispatch({
        type: PROJECTS_FAILURE,
      });
    }
  };

export const clearProjectRows = (): RootThunkAction => async
  (dispatch) => {
    dispatch({
      type: CLEAR_ROWS,
    });
  };

```

```
export const clearProjectErrors = (): RootThunkAction => async
(dispatch) => {
  dispatch({
    type: CLEAR_ERRORS,
  });
};
```

ДОДАТОК Б
Слайди презентації

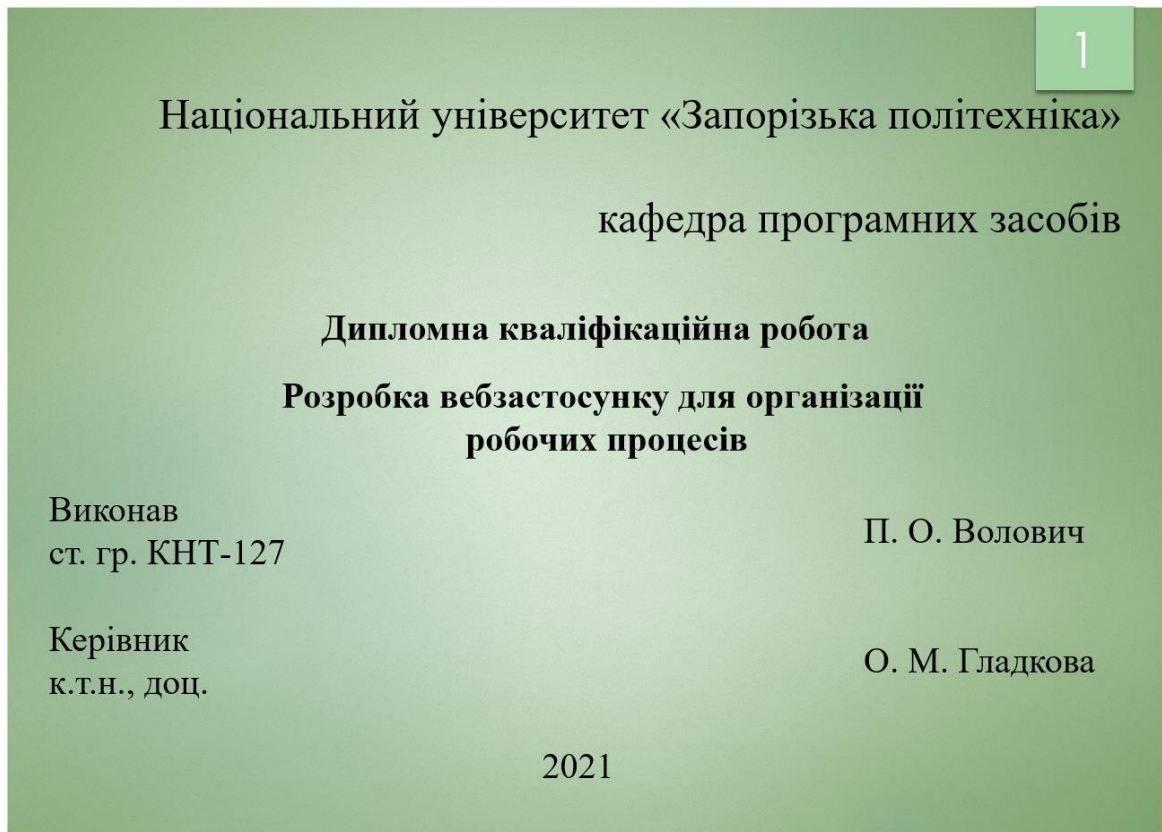
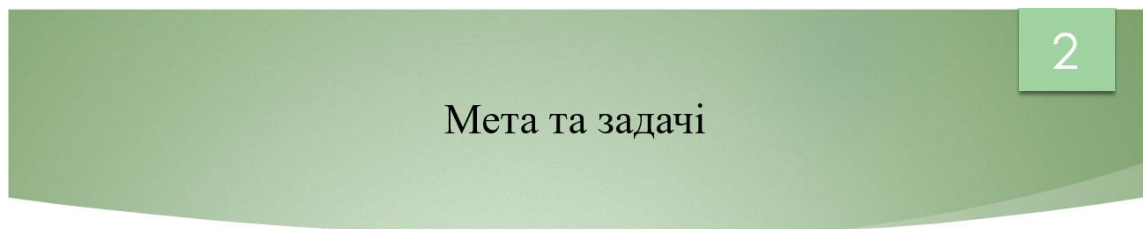


Рисунок Б.1 – Слайд № 1



Мета роботи полягає в розробленні вебзастосунку організації робочого процесу для поліпшення його ведення, можливості зручної модифікації та уникнення значних непорозумінь та складнощів.

Задачі роботи:

- ▶ аналіз подібних вебзастосунків;
- ▶ визначення вимог до вебзастосунку організації робочих процесів;
- ▶ проектування вебзастосунку;
- ▶ реалізація вебзастосунку.

Рисунок Б.2 – Слайд № 2

Важливість вибору засобів розробки

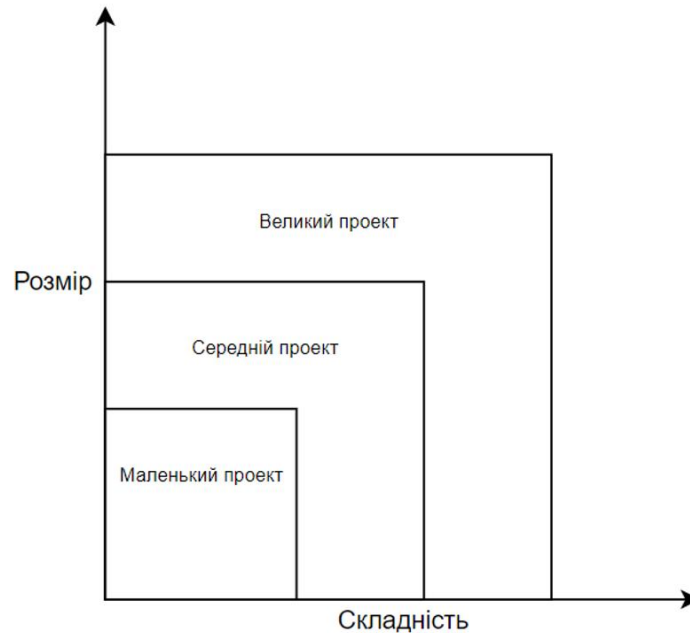


Рисунок Б.3 – Слайд № 3

Порівняння бібліотеки React.js з фреймворками: Angular.js, Vue.js

Критерій	React	Angular	Vue
Мова	JavaScript	TypeScript	JavaScript
Бажана мова розробки	JSX/TSX	TypeScript	HTML-шаблони і JavaScript
Підтримка спільнотою	Розробники Facebook	Розробники Google	Підтримка співтовариства
Тип	Бібліотека	Фреймворк	Фреймворк
Розробник	Facebook	Google	Колишній співробітник Google
Модель	DOM	MVC	DOM
Поріг входження	Середній	Малий	Великий
Типи застосунків	Нативні, гібридні, веб-додатки	SPA, мобільні додатки	Просунуті SPA, нативні веб-додатки

Рисунок Б.4 – Слайд № 4

Основні функціональні частини системи

Система управління проектами	Система управління картками проекту
Дана система передбачає повне управління проектом, з можливістю контролювати внутрішні картки	Система повинна надавати повний контроль користувачу над картою у визначенні теми конкретного завдання, опису проблеми завдання, а також можливість визначення категорії картки: вихідні картки (Resources), які потрібно зробити (To do), в процесі (In progress), готові (Done)

Рисунок Б.5 – Слайд № 5

Діаграма використання вебзастосунку користувачем

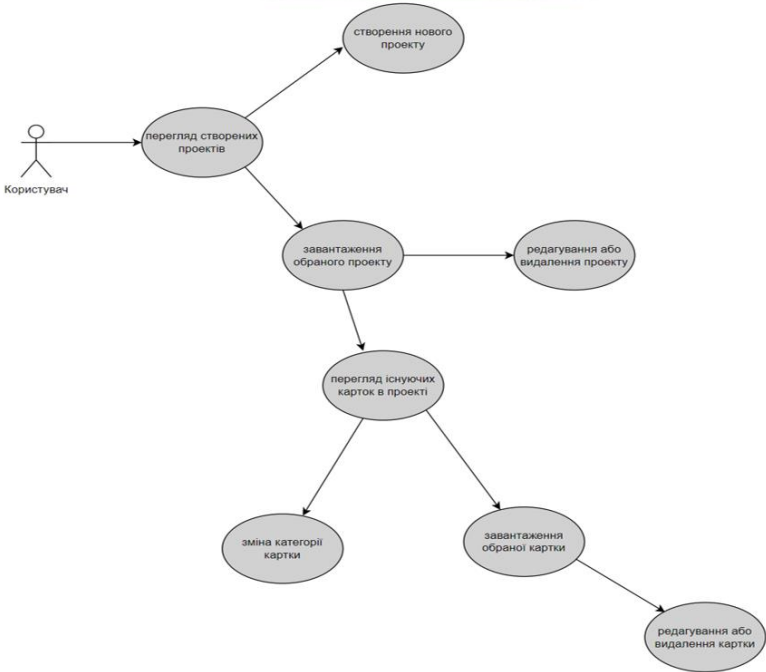


Рисунок Б.6 – Слайд № 6

7

Відображення повідомлень про дії користувача

Projects		
Name	Key	Created
Self education	SE	2021-04-25 05:26:29 PM
Need more money	NMM	2021-05-19 05:23:04 PM

Projects		
Name	Key	Created
Self education	SE	2021-04-25 05:26:29 PM

Successfully created project
Project not found

Рисунок Б.7 – Слайд № 7

8

Схема односпрямованого потоку взаємодії Flux у вебзастосунку

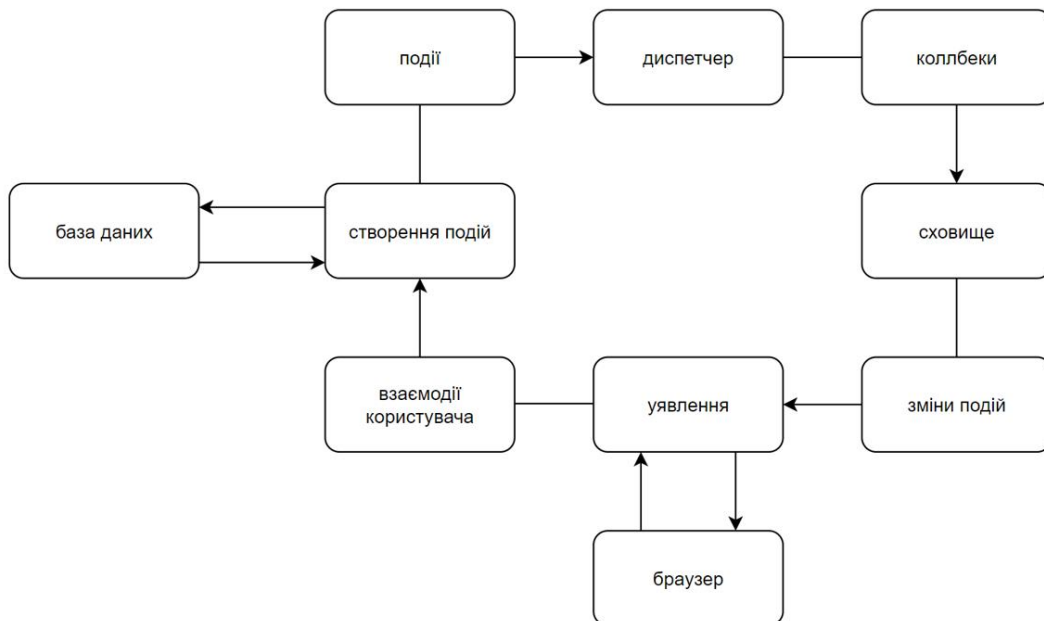


Рисунок Б.8 – Слайд № 8

Схема бази даних

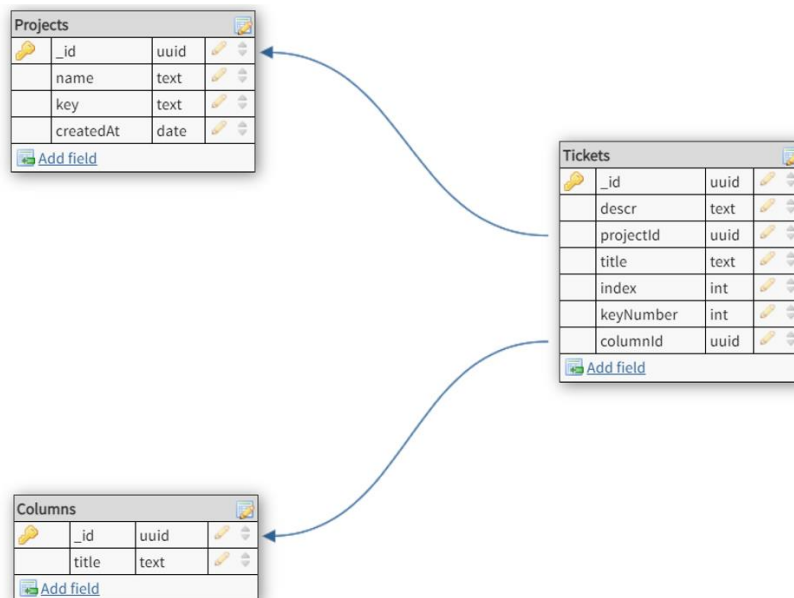


Рисунок Б.9 – Слайд № 9

Висновки

В результаті виконання дипломної кваліфікаційної роботи бакалавра виконано аналіз проблеми організації та планування робочого процесу і детальний огляд веб-ресурсів, які використовуються як великими компаніями, так і людьми, які потребують поліпшення ведення своїх проектів тощо.

Вебзастосунок розроблений на мовах програмування JavaScript, TypeScript, React.js для клієнтської частини і Node.js, JavaScript, Express.js, TypeScript для серверної частини, а також система керування базами даних MongoDB.

Виконано проектування вебзастосунку на основі створення діаграм дій користувача, послідовностей, архітектури серверної та клієнтської частин, схеми бази даних.

Створений вебзастосунок дозволяє керувати організацією проекту, як окремий проект, з призначеними завданнями, до яких можна додати повноцінний опис з форматкуванням для загального розуміння, та вимогами, які можна призначати на різні категорії виконання.

Вебзастосунок має підключене доменне ім'я та налаштований хостинг і працює стабільно у реальному часі.

Всі завдання, поставлені у дипломній кваліфікаційній роботі бакалавра, були повністю виконані.

Рисунок Б.10 – Слайд № 10