МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

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

«До захисту допущено»
В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ
(підпис)
червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Веб-орієнтована інформаційна система управління задачами розробки програмного забезпечення»
здобувача групи ІН - 91 Виганяйла Олександра Володимировича

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

Олександр
_____ ВИГАНЯЙЛО
(підпис)

Керівник,
асистент кафедри комп'ютерних
наук Ольга ШУТИЛЄВА _____
(підпис)

Суми – 2023

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

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

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-91 Виганяйла Олександра Володимировича

1. Тема роботи: «Веб-орієнтована інформаційна система управління задачами розробки програмного забезпечення»_____
затверджую наказом по СумДУ від _____
2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*
3. Вхідні дані до кваліфікаційної роботи_____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для розробки веб-орієнтованих інформаційних систем. 3) Розробка веб-орієнтованої інформаційної системи управління задачами розробки програмного забезпечення. 4) Тестування основного функціоналу веб-додатку.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

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

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

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	18.03.2023	
2	*Огляд технологій, що використовуються для розробки веб-орієнтованих інформаційних систем.*	25.04.2023	
3	*Розробка веб-орієнтованої інформаційної системи управління задачами розробки програмного забезпечення*	01.05.2023	
4	*Тестування основного функціоналу веб-додатку*	30.05.2023	

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

АНОТАЦІЯ

Записка: 72 стр., 23 рис., 4 таблиці, 6 додатків, 16 літературних джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною, оскільки присвячена розробці веб-орієнтованої інформаційної системи, яка допомагає організувати роботу над створенням програмних продуктів у командах та відслідковувати статус виконання завдань.

Об’єкт дослідження – управління задачами розробки програмного забезпечення.

Мета роботи – розробка веб-орієнтована інформаційна система управління задачами розробки програмного забезпечення.

Методи дослідження – обробка теоретичної бази з проблематики та огляд необхідного функціоналу.

Результати – проаналізовано вже готові рішення для керування проєктами. На основі цього аналізу були сформовані вимоги до програмного продукту, який повинен допомагати в управлінні створення завдань проєкту. Розроблено веб-орієнтовану інформаційну систему, яка дозволяє керувати станом розроблення проєкту шляхом переміщення завдань у відповідну колонку, яка відповідає за його поточний стан.

ІНФОРМАЦІЙНА ВЕБ-ОРІЄНТОВАНА СИСТЕМА, JAVASCRIPT, REACT, MATERIALUI, NODEJS, MONGODB, EXPRESS, REDUX

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

JWT – JSON Web Token

HTML – HyperText Markup Language

W3C – World Wide Web Consortium

WHATWG – Web Hypertext Application Technology Working Group

CSS – Cascading Style Sheets

DOM – Document Object Model

API – Application Programming Interface

REST – Representation State Transfer

SPA – Single Page Application

СУБД – Система управління базами даних

BSON – Binary JSON

UMD – Universal Module Definition

ESM – EcmaScript Module

ВСТУП

Застосування сучасних інформаційних технологій стало надзвичайно популярним у різних сферах особистого, соціального та національного життя. Ця популярність зумовлена впровадженням локальних та глобальних інформаційних систем, які спрямовані на прискорення обміну інформацією та забезпечення доступу до різноманітних інформаційних ресурсів.

Однією з основних задач всесвітньої мережі є забезпечення доступу кінцевим користувачам до веб-ресурсів та вирішення поставлених завдань. Логіка багатьох додатків централізована на серверній частині, тоді як веб-браузер часто виконує роль простої оболонки для відтворення інформації, завантаженої з сервера, і передачі даних користувача на серверну частину додатку. Однією із всіх переваг такого підходу є те, що клієнтські пристрої не мають прямої залежності від конкретної операційної системи, що дає змогу розробляти веб-додатки, які працюють на різних платформах.

Актуальність роботи визначається у тому , що кількість проєктів, над якими працює велика кількість команд та людей, зростає з кожним днем і для бізнесу дуже важливо оптимально розподіляти задачі між співробітниками та бачити прогрес виконання тих чи інших задач. Кожна компанія зацікавлена у розробленні свого унікального продукту для рішення проблем делегування та розподілення задач в межах своєї компанії, з урахуванням особливостей використання та специфіки розробки саме тому такий продукт є актуальний на даний час.

Об’єкт дослідження – процес розробки веб-орієнтованої системи для управління задачами розробки програмного забезпечення.

Метою роботи є розробка власної веб-орієнтовної інформаційної системи, яка буде: допомагати у керуванні проєктами та його задачами, мати простий та зрозумілий інтерфейс та реалізовувати основні функціональні можливості для створення та видалення задач. Інтернет є надзвичайно привабливим засобом комунікації, що пояснює його експоненційне зростання

протягом останнього десятиріччя. Новітні засоби зв'язку дозволяють об'єднувати різноманітні комунікаційні системи в єдину глобальну мережу.

Гіпотеза дослідження була сформована на основі зростаючої кількості команд , для яких потрібний функціонал для ефективного управління внутрішніми процесами розробки програмних продуктів.

Новизна фінального результату полягає у розробленні клієнто-орієнтованої веб-системи , яка буде допомагати в управлінні процесу розробки програмного забезпечення. Веб-додаток може бути використаний як для персонального контролю робочого процесу , так і для команд з великою кількістю розробників.

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

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Сучасні технології управління проєктами та задачами

Зростаюча кількість команд і співробітників, що працюють над проєктами, ставить перед бізнесом завдання оптимального розподілу завдань і контролю за їх виконанням. Кожна компанія має інтерес до розробки свого продукту, який вирішує проблеми виконання та менеджменту завдань всередині організації, з урахуванням особливостей й потреб розробки програмних продуктів.

У сучасному світі, де ефективне управління проєктами стає все більш важливим, існує безліч веб-застосунків, які пропонують інструменти для створення та управління задачами. Але обрання даного інструментарію вимагає окремих вимог до його функціоналу, наприклад:

- простота використання програмного забезпечення серед працівників, що дозволить швидко опанувати весь функціонал та приступити до його інтеграції у робочий проєкт;
- підтримка та отримання нових версій, для впровадження додаткового функціоналу та виправлення помилок існуючої версії програми
- функціональні можливості та специфічні кейси для вирішення тих чи інших задач на кожному з проектів;
- масштабованість для роботи над великими проєктами, що забезпечить ефективне управління як при малих, так і при великих обсягах роботи.

Останнім часом набирає популярність стандартизація управління командами і в цьому дуже допомагають системи управління проєктами та задачами, які виконують важливу роль імплементації логічних методологій керування проєктами.

Одним з основних та найбільш використовуваних методологій є Waterfall. Цей підхід є каскадним, що в свою чергу включає в себе поділ

розробки продукту на фази та етапи. Основним критерієм є виконання кожного етапу один за одним, що дозволяє логічно описати прогресс виконання. Зазвичай такий підхід використовується для проєктів з чітко визначеним планом, і потрібно тільки слідкувати щоб були дотримані план та сроки, які чітко визначені замовником. Перевагами такого методу є послідовність, об'єктивна оцінка прогресу, низький ризик недотримання плану та конкретна собівартість розробки. Але сам процес розробки не має гнучкої системи, тому зміни у продукті не є простою задачею і попередньо обговорюються з замовником.

Наступною методологією яка також є популярною це Agile. Ця система представляє собою систему управління, головною метою якої є представлення готових рішень на кожному етапі розробки, але при цьому технічне завдання може змінюватись замовником вже під час реалізації, і тому кінцевий продукт може видозмінюватись не один раз. Планування проходить перед початком спринту, де обговорюються строки, план реалізації та команду яка потрібна для виконання. Такий підхід добре зарекомендував себе для реалізації невеликих проєктів та стартапів, так як у цьому випадку зміни, гнучкість та прогнозованість допомагають команді працювати ефективно. Та попри всі переваги, використання цього методу в деяких випадках може перетворити розробку у низку великої кількості змін, що не завжди можна спрогнозувати через досвід команди та її професіоналізм. [1]

Останнім з підходів до організації розробки є Scrum, що в свою чергу є різновидністю Agile. Така методологія розроблена у такій формі, при якій всі внутрішні процеси скомпоновані найбільш оптимально, і їх використання окремо, не дасть такої ефективності. Головний тезис закладається в тому, що цей підхід залучає абсолютно всіх розробників до реалізації проєкту, а у разі виникнення проблем, будь хто з інших колег може замінити один одного, саме тому відповідальність за кінцевий продукт відчуває кожен член команди і вирішення тих чи інших питань по архітектурі приймаються командою. Зазвичай при використанні цієї методології, в команді повинен бути скрам-

майстер, який плідно слідкує за виконанням плану та контролює процес роботи. Як результат, це дає абсолютну прозорість роботи команди, при цьому відбувається взаємо обмін знаннями та досвідом і замовник навіть при внесенні правок, отримує позитивну відповідь і терміни закінчення розробки.

Дивлячись на загальні підходи до організації роботи у колективі, стає зрозумілим і цілком обґрунтованим створення програмного забезпечення для керування проєктами та його задачами, так як на даний час це є актуальним вирішенням проблем при роботі в команді.

1.2 Аналіз програмних продуктів-аналогів

Визначивши основні критерії, які потрібні для програмних продуктів керування проєктами та їх задачами, можна розглянути вже існуючі рішення. Той чи інший варіант реалізований більше під потреби конкретної робочої групи і доданий окремий функціонал, але головна ідея це імплементація веб-додатку, з допомогою якого можна реалізовувати різні методології управління проєктами.

1.2.1 Хмарна програма для керування проєктами Trello

Trello – це хмарна система для керування проєктами та задачами розроблена компанією Atlassian та Glitch. Дане програмне забезпечення дозволяє відстежувати робочі процеси, створювати нові задачі та організовувати роботу команди. Основний принцип полягає у дошках та картках, що позитивно впливає на враження від користування та має досить простий функціонал, що дозволяє легко розібратись в основних функціях [2].

Домашня сторінка особистого аккаунта користувача має досить інформативний склад та відображує основні проєкти до яких підключений користувач та найбільш популярні шаблони (рис. 1.1).

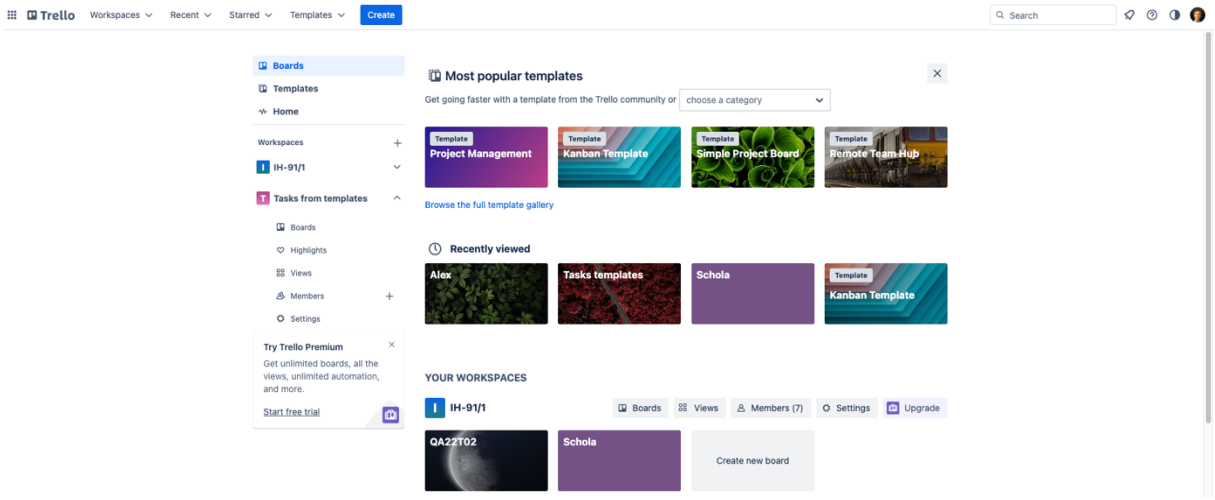

Рисунок 1.1 – Домашня сторінка [2]

Є декілька варіантів створення дошки для керування задачами, це створити пусту дошку і додавати туди свої зміни. Дошки в Trello можуть містити всю потрібну інформацію для користувача (рис. 1.2).

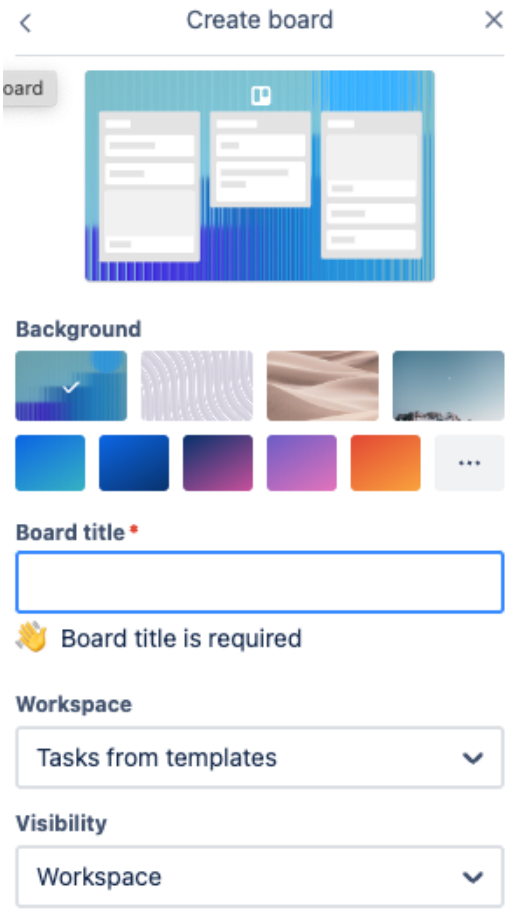

Рисунок 1.2 – Створення пустої дошки

У такому випадку буде представлена дошка без жодних заготовок для роботи. Але якщо обрати вже готовий шаблон за методологією яка потрібна конкретній команді, тоді буде створена дошка з підготовленими налаштуваннями та колонками (рис. 1.3).

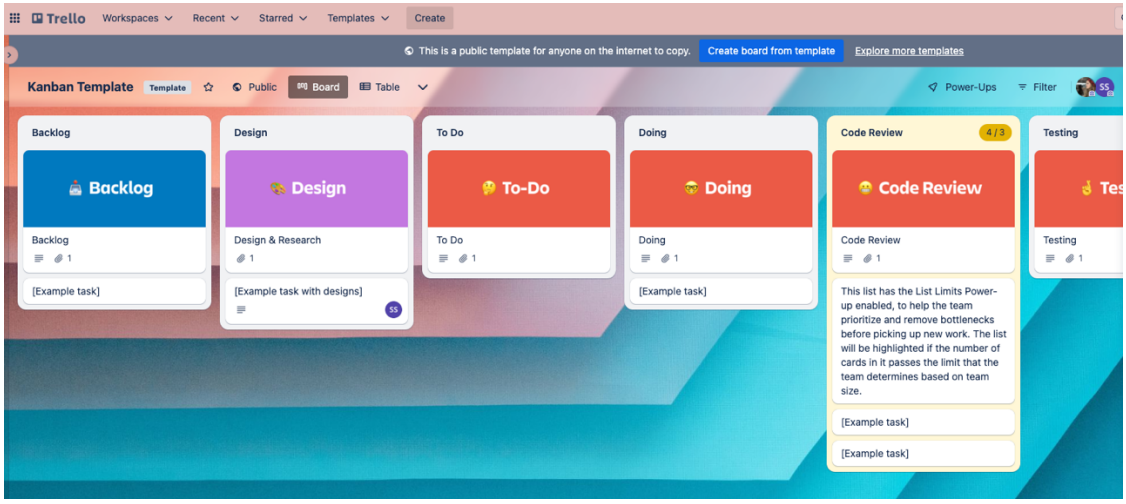

Рисунок 1.3 – Шаблон за потрібною методологією

Також у веб-застосунку представлений дуже великий список функціоналу для роботи з завданнями, завдяки якому можна контролювати абсолютно всі процеси які відбуваються в команді (рис. 1.4).

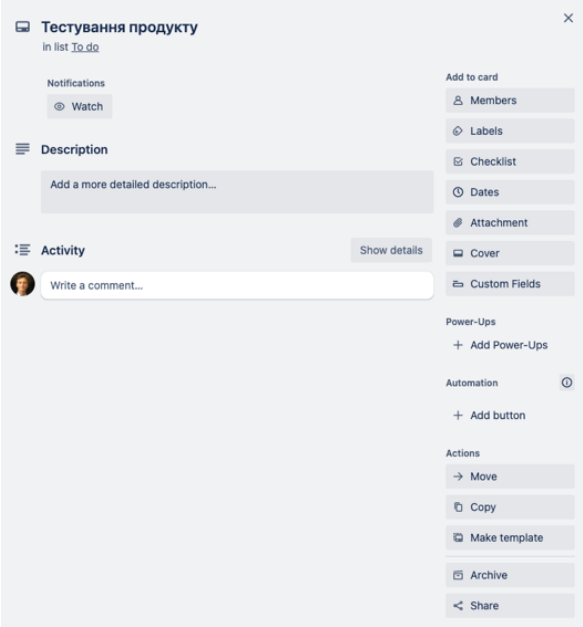

Рисунок 1.4 – Редагування задачі

Також для зручності та контролю команди, є запис попередніх дій які були зроблені конкретною людиною з конкретною задачею (рис. 1.5).

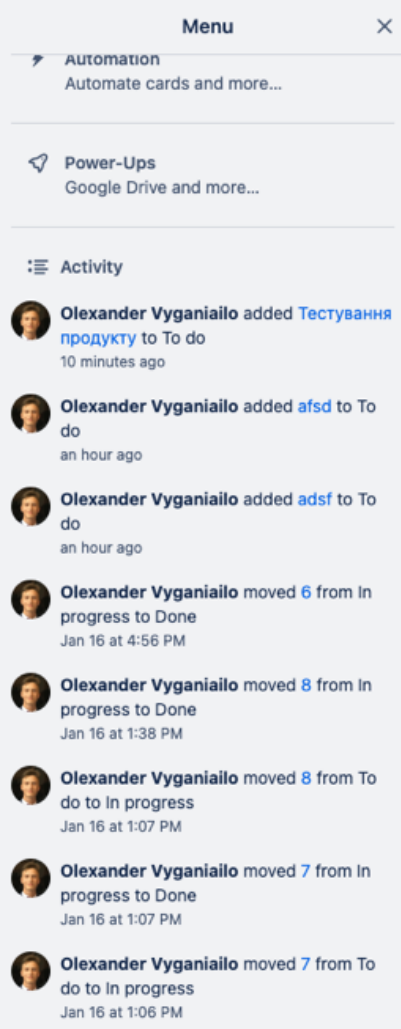

Рисунок 1.5 – Запис попередніх дій

Якщо взяти загальні потреби які потрібні для реалізації ведення проєкту та створення задач, то вони реалізовані. Також цей веб-застосунок надає функціонал спілкування та співпраці у вигляді коментаря до карток, але це більше використовується для нотаток і позбавляє можливості загальної комунікації між співробітниками.

1.2.2 Система управління проєктами Jira

Jira – це система управління проєктами, була розроблена компанією Atlassian. Веб-застосунок дозволяє ефективно працювати в команді, націлений більше на широкий спектр функцій і можливостей для організації процесів розробки, він має систему стеження за помилками та надає гнучкі можливості використання його для управління задачами [3].

На головній сторінці веб-додатку присутні менший функціонал ніж у Trello (рис. 1.6).

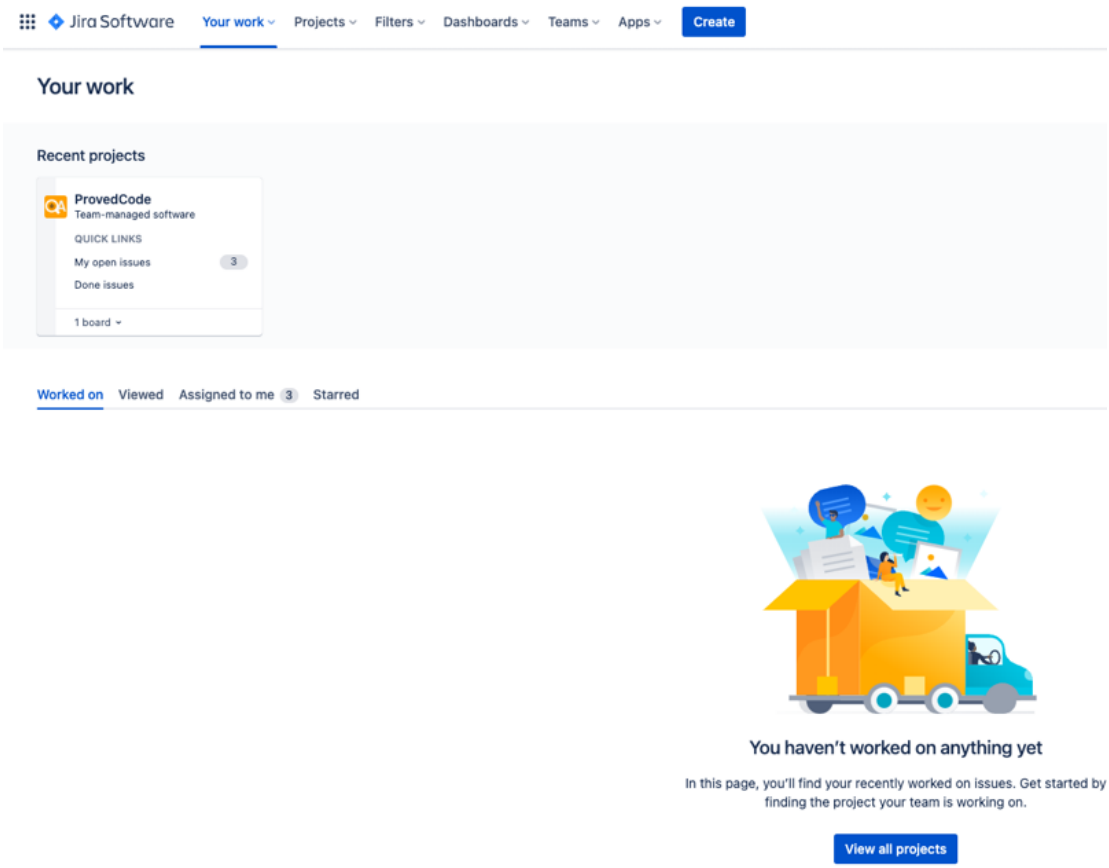

Рисунок 1.6 – Головна сторінка Jira

Але при створенні нового проєкту буде представлений весь функціонал. Це не тільки вже готові шаблони для роботи з такими методологіями як Scrum та Kanban, але і розділи для менеджменту, маркетингу та фінансів.

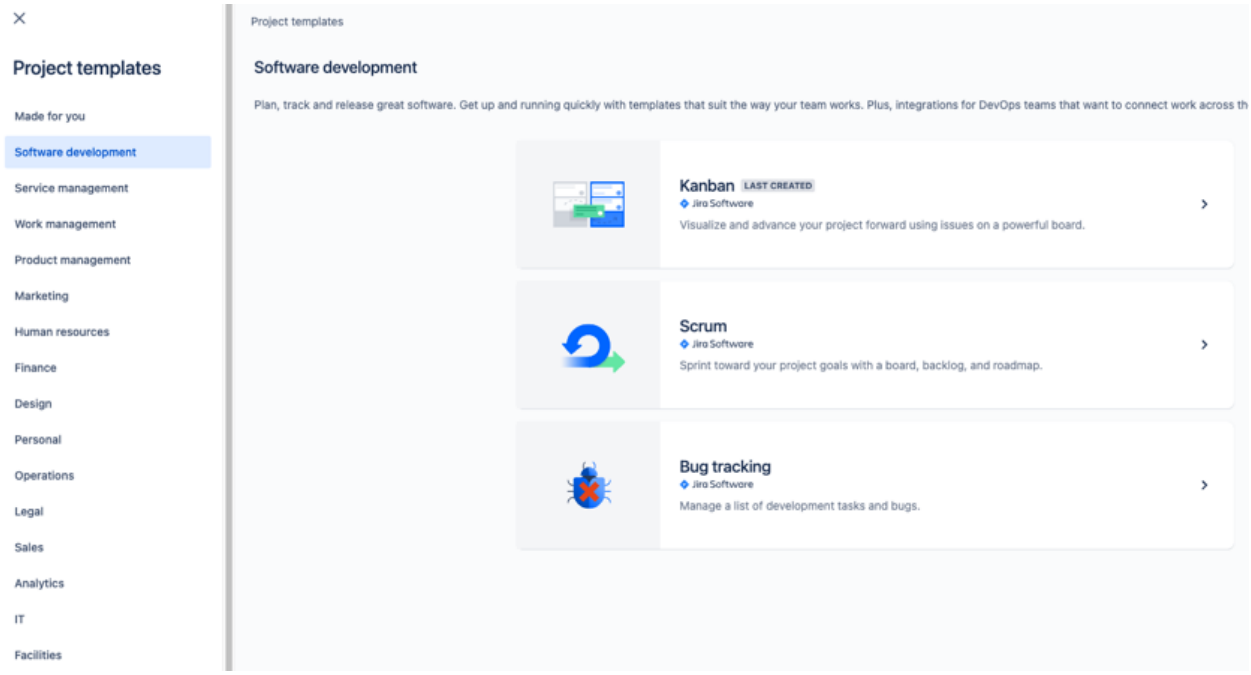

Рисунок 1.7 – Сторінка створення проєкту

Також на сторінці проєкту є колонки з задачами та статусом розробки проєкту. Присутня як і фільтрація по типу задачі, так і різне відображення прогресу (рис. 1.8).

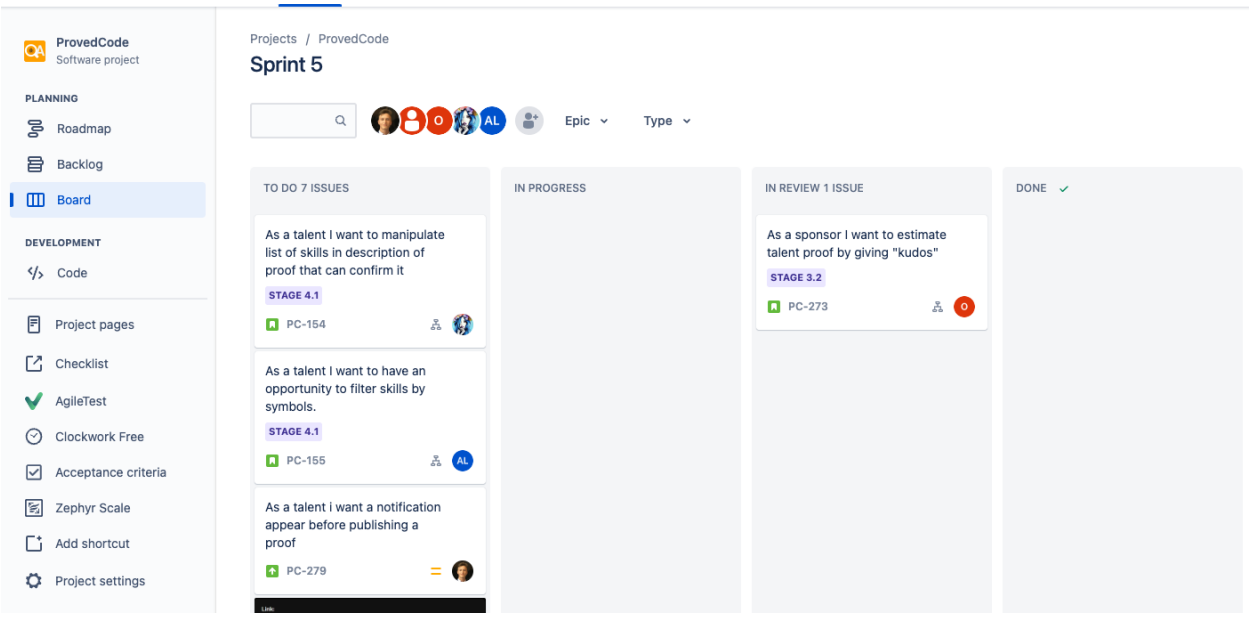

Рисунок 1.8 – Головна сторінка проєкту

1.3 Постановка задачі

Головним завданням є побудова веб-орієнтованої системи керування проєктами і задачами. Зробивши аналіз вже існуючих веб-додатків, було зроблено висновки щодо функціоналу та інтерфейсу програми. Повинна бути можливість створювати окремі дошки проєктів. В середині кожної дошки має бути функціонал додавання нових колонок та створення задач.

Для повноцінної реалізації проєкту потрібно дотримуватись таких пунктів розробки:

1. Визначення вимог до функціоналу додатку
2. Налаштування середовища розробки з використанням React, React router, JWT token, Node JS, Express, MongoDB.
3. Створення серверної сторони додатку з використанням Node JS та Express.
4. Налаштування бази даних MongoDB та підключення її до серверної сторони.
5. Створення API для взаємодії клієнтської сторони з сервером.
6. Створення компонентів React для користувацького інтерфейсу додатку.
7. Розробка функціоналу авторизації користувача з використанням JWT token.
8. Розробка функціоналу створення та відстеження задач проєкту.
9. Розробка функціоналу створення та відстеження проєктів.
10. Розробка функціоналу збереження та відновлення стану додатку під час перезавантаження сторінки.
11. Проведення тестування додатку та виправлення помилок.

2 ВИБІР МЕТОДІВ ВИРІШЕННЯ ЗАДАЧІ

2.1 Вибір мов програмування

Проаналізувавши основні вимоги до кінцевої веб-орієнтовної системи, для створення такого функціоналу потрібно використовувати різні технології, високорівневу мову програмування та нові фреймворки які дозволять реалізувати актуальні рішення.

2.1.1 HTML

На даний момент прослідковується висока актуальність використання мови гіпертекстової розмітки як HTML. Формат або мова HTML, розроблений W3C і WHATWG, з’явився в 1990-х рр. Він поступово зазнавав модифікацій і з 2014 року пропонує більш успішну версію HTML5.

HTML – це те, що дозволяє розробникам веб-сайтів керувати тим, як вміст їхніх веб-сторінок відображатиметься на екрані через браузер. Він заснований на системі тегів, що дозволяє додавати заголовки, субтитри, жирний шрифт тощо до тексту та вводити інтерактивні елементи, такі як зображення, посилання, відео, HTML легше сприймається роботами пошукових систем, ніж мова JavaScript, також використовується, щоб зробити сторінки більш інтерактивними [4].

Система розмітки використовується для опису браузеру того, що він має відображати користувачам Інтернету. Використання різноманніття тегів вказує браузеру, які елементи використовуються на сторінці.

Окрім форматування вмісту, ви можете вставляти інтерактивні елементи, такі як посилання, зображення чи відео, за допомогою HTML, а також сценарії з різних мов (PHP, JavaScript тощо).

Але потрібно розуміти, що мова гіпертекстової розмітки співпрацює з CSS, який використовується для надання стилю елементам (кольору, розташуванню, типу шрифту тощо).

2.1.2 CSS

Наступною технологією яка є основою як нових бібліотек для стилів, так і різних підходів до стилізації веб-додатків є CSS. З кожним днем все більше використовуються нові практики у оформленні зовнішнього вигляду, але такі бібліотеки як Styled Component та Emotion лише полегшують використання стилів і все одно перетворюють стилі у CSS код. [5]

Дана мова каскадних стилей працює за технологією обирання елементів всередині DOM дерева та застосування стилей до них. Кожне правило складається з селектора та властивостей стилей для нього.

2.1.3 JavaScript

JavaScript – це високорівнева мова програмування, створена для додавання функціоналу до веб-сторінок. Вона відрізняється від інших мов тим, що виконання коду виконується самим браузером, на стороні користувача, а не веб-сервером. Таким чином, вона реалізовує загальник стандарт ECMA Script, що позначає клієнт-орієнтовані мови сценаріїв.

У мови JavaScript є багато сценаріїв використання:

- Реагування на дії користувача, надсилання форм та створення інтерактивних елементів.
- Обширний API для маніпуляції з DOM деревом, що дозволяє динамічно змінювати вміст, артибути та стилі HTML документу.
- Підримка асинхронних запитів через зворотні виклики та Event Loop, що дозволяє паралельно виконувати різну кількість запитів і не зупиняти роботу програми.
- JavaScript є основою для розроблення веб-додатків і такі відомі фреймворки як React, Angular та Vue побудовані на базі цієї мови.

2.1.4 NodeJS

Також JavaScript набула популярності у вигляді NodeJS, що є серверною різновидністю її використання. Загалом, це зовсім інше середовище

виконання, яке дозволяє виконувати JavaScript поза браузером, на серверній стороні. Він забезпечує середовище, яке може запускати код JavaScript як окрему програму, дозволяючи розробникам створювати програми на стороні сервера, інструменти командного рядка, API тощо. [6]

Будучи середовищем виконання на стороні серверу, воно може використовуватись для створення серверних програм, мікросервісів та REST API.

2.2 Вибір фреймворків та бібліотек

При розробці програмного продукту, розробники почали помічати, що рішення, які вони імплементують, вже були розроблені до цього. І замість повторної реалізації існуючого функціоналу, почали використовувати вже готові рішення.

2.2.1 React

React – це JavaScript бібліотка для створення користувацьких інтерфейсів, яка змінює відображення без перезавантаження сторінки за допомогою технології SPA. Завдяки чому веб-додатки швидко реагують на дії користувача, і хоча React досить часто порівнюється з такими технологіями як Angular та Vue.js, він є лише бібліотекою і потребує додаткових бібліотек та пакетів для повноцінної роботи. Дана бібліотека базується на конструюванні масштабних інтерфейсів, шляхом створення більш менших за розміром та їх комбінування. [7]

Такий підхід до створення веб-додатків є декларативним, тобто розробка полягає в тому, щоб описати те, що потрібно вивести користувачу, а не створювати інструкції, як це зробити. (рис. 2.1)

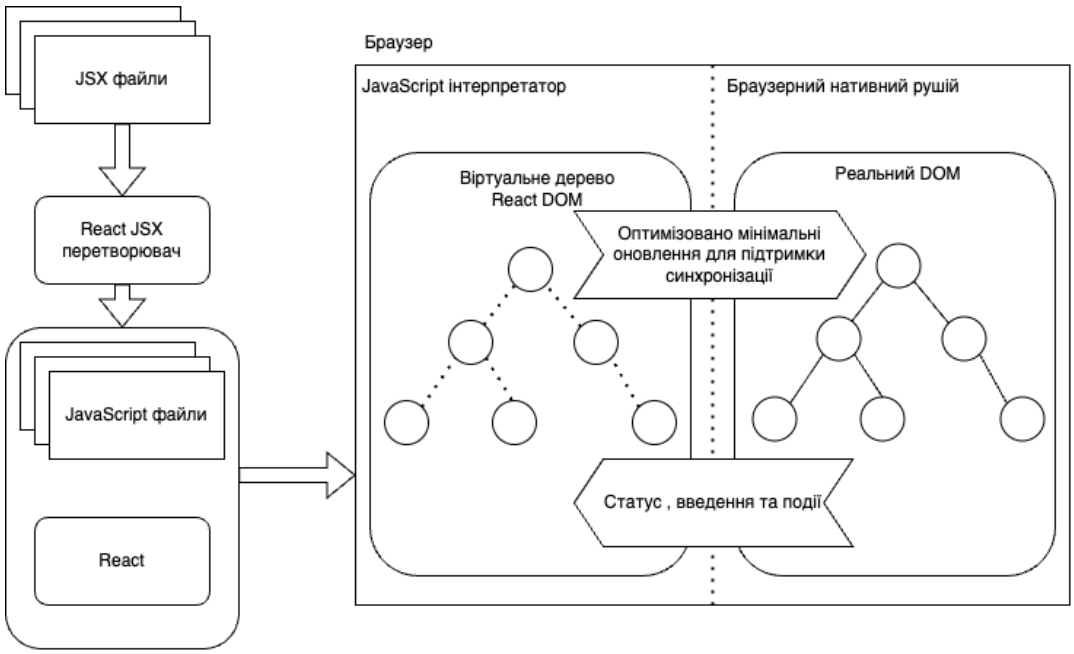

Рисунок 2.1 – Структура виконання веб-додатків

Також React є дуже цінним ресурсом з боку бізнесу та замовника, так як він допомагає у таких речах:

- Технологія Virtual DOM збільшує результативність веб-додатків, з великою навантаженістю, що в свою чергу вирішує проблему довгої роботи при великих навантаженнях на сайт
- Прискорення рендерингу сторінки за допомогою використання деревовидної структури. Завдяки можливості використання одного коду як на клієнтській так і на серверній стороні, уникнуто дублювання функціоналу.

2.2.2 MaterialUI

Сучасним підходом до стилізації компонентів у React є бібліотека MaterialUI. Вона дотримується стандартів щодо матеріального дизайну веб-додатків розроблених компанією Google. Ця бібліотека надає широкий спектр вже готових компонентів, які мають в тому числі інтерфейс для налаштування та взаємодії з елементами.

Головною метою розроблення MaterialUI було дотримання концепції Material Design, що означає акцентування уваги на інтуївитному інтерфейсі

додатку та використання типографіки та руху компонентів для збереження концентрації користувача.

Material-UI також пропонує потужний інструментарій для створення користувацьких тем та стилей. Він надає рішення для стилізації завдяки використанню CSS-in-JS, дозволяючи розробникам писати стилі безпосередньо в JavaScript. Цей підхід дозволяє легко налаштовувати компоненти та теми, забезпечуючи повну взаємодію користувача із загальним дизайном програми. Також розробники додали підтримку адаптивних макетів, що полегшує створення адаптивних програм і забезпечує якісну взаємодію з користувачами на різних пристроях і розмірах екрана. Він також містить готові функції доступності, гарантуючи, що компоненти відповідають стандартам, необхідним для відповідності доступності. [8]

2.2.3 Vite

З кожним роком веб-додатки масштабуються, до них додається новий функціонал, і таким чином вони стають значно складними в розумінні та розростанні. Досить актуальним виходом з цієї проблеми є використання збиральника проєкту. Такий інструмент використовується для компіляції та пакування ресурсів, шляхом перетворення їх у окремі модулі та запускання скриптів для їх відтворення. Така модульна система дозволяє досягти найефективнішої організації проєкту та його масштабуванню.

Найбільш популярним на даний час є Webpack, але його ефективність та швидкодія залишаються невтішними. На вирішення проблем даного збирача з’явився інший – Vite. Перший реліз був нещодавно, але швидкість його роботи значно переважає перед Webpack, і цей збиральник пропонує найшвидший інструмент для збирання веб-додатків.

Підтримка браузерами модулів ES6 зазвичай не була ефективною, але це було зроблено для більшої підтримка в різних браузерах.Vite значно покращує час запуску сервера для режиму розробки, і робить це у вигляді поділу модулів

веб-додатку на категорії залежностей та вихідного коду. Залежності – це JS код, який змінюється дуже рідко під час розробки,

Vite робить попередній збір залежностей для сумісності CommonJS та UMD в режимі розробки, як власний ESM, для даної операції потрібно спочатку перетворити залежності, які зберігаються як CommonJS та UMD.

```
var dep1 = require('./dep1');
var dep2 = require('./dep2');
module.exports = function(){
  // ...
}
```

Рисунок 2.2 – CommonJS формат

```
(function (root, factory) {
    if (typeof define === 'function' && define.amd) {
        define(['b'], factory);
    } else if (typeof module === 'object' && module.exports) {
```

Рисунок 2.3 – UMD формат

Для підвищення продуктивності, Vite перетворює залежності ESM модулей в один, щоб пришвидшити завантаження сторінки. Деякі пакети реалізують процес збирання шляхом підключення великої кількості окремих файлів, і в такій ситуації може траплятись таке, що модуль одночасно робить близько 600 HTTP запитів, що звісно причиняє перенавантаження на стороні браузера і через це сторінка перезавантажується довго. А завдяки попередньому об’єднанню в один модуль, замість 600 запитів, буде виконуватись лише один.

Хешування файлової системи попередньо хешує пов’язані між собою залежності та визначає, чи потрібно повторно запускати етап попереднього збирання проєкту на основі таких відомостей:

- Склад файлу блокування менеджера пакетів, таких як package-lock.json, yarn.lock, bun.lockb тощо;
- Час зміни папки програмних патчів;
- Відповідні поля в файлі vite.config.js;
- Цінність файлу оточення.

ESBuild використовує мову програмування Go, і це пришвидшує пов’язування залежностей у 10-100 разів швидше. [9]

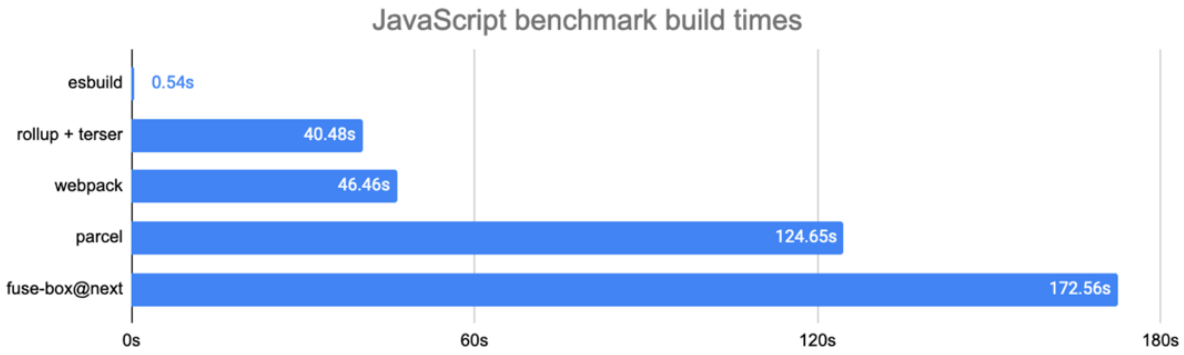

Рисунок 2.4 – Порівняння швидкості пов’язування залежностей

Вихідний код часто містить не звичайний JS код, а JSX, CSS або Vue. Vite допомагає організувати вихідний код таким чином, що замість браузера, бере роботу на себе у перетворенні та обслуговуванні коду запиту.

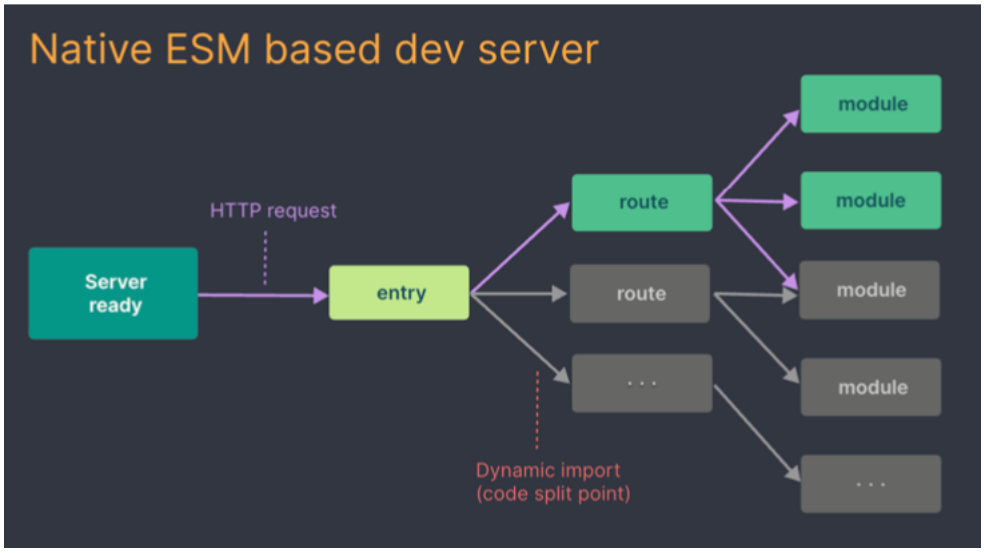

Рисунок 2.5 – Структура збірки проєкта

Таким чином, проаналізувавши швидкість та практичність підходу Vite до збірки проєкту, більш доцільно використовувати його при розробці програмного продукту.

2.3 Вибір СУБД

СУБД – це система управління базами даних, таке програмне забезпечення дозволяє користувачам створювати, маніпулювати та керувати базами даних. СУБД надає користувачам інтерфейс для взаємодії з базою даних, дозволяючи їм зберігати, отримувати, оновлювати та видаляти дані. Він вирішує такі завдання, як організація даних, зберігання, пошук, безпека та контроль паралельності. СУБД широко використовуються в різних галузях і додатках для ефективного керування великими обсягами структурованих і неструктурованих даних.[11]

2.3.1 SQLite

Розробка такої легкої системи контролю бази даних було зумовлене необхідністю зробити самодостатню без серверну СУБД без конфігурації. Дивлячись на стандартні механізми баз даних які використовують архітектуру клієнт-сервер, SQLite вбудовується в саму програму, що дозволяє використовувати її без окремого серверного процесу та інсталяції. [10]

Зазвичай цю СУБД використовують у мобільних та невеликих десктоп додатках, де ключову роль грають легкість та ефективність бази даних, саме це і пропонує SQLite.

2.3.2 PostgreSQL

PostgreSQL є досить функціональною об’єктно-реляційною СУБД, що також дозволяє розширювати та стандартизувати базу даних. Вона підтримує складні типи даних, має досить гнучку схему і також реалізовані транзакції та реплікація [13]. Функціонал даної системи дуже обширний, наприклад:

- Дотримується реляційної моделі, що в свою чергу дозволяє робити таблиці, стовпці, зв’язки та обмеження.
- Обробляє одночасний доступ до бази даних за допомогою MVCC, що дозволяє виконувати декілька транзакцій одночасно без конфліктів. Це забезпечує високу паралельність і узгодженість даних.
- Розроблено для роботи з великими обсягами даних і високим навантаженням трафіку. Він підтримує різні методи оптимізації продуктивності, такі як індексування, оптимізація запитів і паралельне виконання. Крім того, він пропонує такі розширені функції, як розділення таблиці та реплікація для горизонтального масштабування.
- Надає механізми для забезпечення цілісності даних, включаючи підтримку обмежень, тригерів і зовнішніх ключів. Він також забезпечує цілісність транзакцій, дозволяючи групувати кілька операцій бази даних у атомарні одиниці.
- Включає вбудовану підтримку повнотекстового пошуку, що забезпечує ефективний і точний пошук у текстових даних. Він підтримує розширені можливості пошуку, такі як пошук по основі, ранжирування та пошук по фразах.

2.3.3 MongoDB

MongoDB – це популярна нереляційна система управління базами даних із відкритим вихідним кодом, яка забезпечує гнучке та масштабоване рішення для зберігання та пошуку даних. Вона призначена для обробки великого обсягу даних і підтримує різноманітні моделі даних, що робить її добре придатною для сучасних додатків з динамічними та змінними вимогами до даних.

Ось деякі ключові особливості MongoDB:

- Зберігає дані в гнучких самоописуваних JSON-документах, які називаються BSON-документами. Ця модель документів полегшує представлення складних ієрархічних зав’язків та забезпечує гнучкість схеми, оскільки документи в колекції можуть мати різні структури.

- Вона призначена для горизонтального масштабування, що дозволяє розподіляти дані між кількома серверами та забезпечує високу масштабованість. Підтримує розподіл даних між кількома серверами для обробки збільшеного робочого навантаження та забезпечення ефективного доступу до даних.
- Має підтримку великої кількість можливих запитів різної складності тому, що використовує гнучку мову запитів на основі JSON-подібного синтаксису, що дозволяє створювати потужні та динамічні запити документів включаючи фільтрацію, сортування та агрегацію.
- Забезпечує вбудовану реплікацію, що дозволяє створювати набори реплік, які забезпечують резервне зберігання даних і автоматичне перемикання після відмови, забезпечуючи високу доступність бази даних. Також є функціонал асинхронної реплікації, коли дані копіюються на вторинні вузли, забезпечуючи відмово стійкість і довговічність даних.
- Пропонує потужні можливості повнотекстового пошуку за допомогою власних текстових індексів. Підтримує лінгвістичний аналіз, оцінку тексту та ранжування на основі релевантних правил для ефективного та точного пошуку тексту в базі даних.
- Включає гнучку та потужну структуру агрегації, яка дозволяє виконувати комплексну обробку даних і аналітику. Надає набір операторів для групування, фільтрації, перетворення та обчислення агрегацій даних, що зберігаються в базі даних.
- Пропонує вбудовану підтримку для різних мов програмування та фреймворків, що полегшує інтеграцію в різні стеки програм. Він надає офіційні драйвери для таких мов, як Java, Python, Node.js та багатьох інших. [12]

MongoDB зазвичай використовується в різних програмах, включаючи системи керування контентом, аналітику в реальному часі, соціальні мережі, платформи IoT і мобільні програми. Його гнучкість, масштабованість і простота використання роблять його популярним вибором для розробників і організацій, яким потрібна масштабована та динамічна база даних, само тому було вирішено використовувати MongoDB.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Програмна реалізація

Беручи за основу базові технології проєктування та створення веб-додатків стає очевидним розділення функціоналу на серверну та клієнтську частини. Клієнтська частина відповідає за взаємодію з користувачем, відображення базового інтерфейсу додатка, виконання запитів на серверну частину та збирання і збереження результатів. Серверна частина відповідальна за обробку запитів з клієнтської, представлення актуального стану даних на сервері, створення, оновлення та збереження даних.

3.1.1 Серверна частина

Для створення серверної частини веб-орієнтованої інформаційної системи було використано такі інструменти як NodeJS, MongoDB, Express. MongoDB на відміну від реляційних баз даних використовує підхід без SQL-запитів та таблиць, замість цього використовується документо-орієнтована модель даних, що дозволяє зробити роботу з базою даних масштабованою та простою.

Документи в MongoDB моделюються за форматом JSON, але насправді зберігаються у BSON, іншими словами це означає, що документ MongoDB є словником пар ключ-значення, де значення може бути одного з кількох типів [14]:

- Примітивні типи JSON (наприклад, число, рядок, логічний)
- Примітивні типи BSON (наприклад, datetime, ObjectId, UUID, regex)
- Масиви значень
- Об’єкти, що складаються з пар ключ-значення
- Нуль

Загальноприйнятим форматом зберігання у MongoDB є колекції. У виконаному веб-додатку присутні такі колекції:

Таблиця 3.1 – Поля колекції User

Тип	Назва	Обмеження	Призначення
Object	_id	unique	Ідентифікатор користувача
String	fullName		Пошта користувача
String	email	unique	Електронна пошта
String	passwordHash		Пароль користувача у зашифрованому вигляді
String	avatarUrl		Посилання на особистий аватар користувача

Таблиця 3.2 – Поля колекції Project

Тип	Назва	Обмеження	Призначення
Object	_id	unique	Ідентифікатор проєкту
String	boardTitle		Назва проєкту
Array	assignees		Масив учасників проєкту
Array	tables		Масив таблиць у проєкті
Date	dateStart		Дата початку проєкту
Date	dateFinish		Дата закінчення проєкту

Таблиця 3.3 – Поля колекції Task

Тип	Назва	Обмеження	Призначення
Object	_id	unique	Ідентифікатор задачі
String	taskTitle		Назва завдання
String	project		Ідентифікатор проєкту до якого відноситься дане завдання
String	taskDescription		Опис завдання
String	status		Стан завдання

Таким чином для повноцінного функціонування серверної частини достатньо вищеперерахованих колекцій.

Для ефективного та логічного використання серверної частини було вирішено реалізовувати підхід REST API. REST – це набір архітектурних обмежень, а не протокол чи стандарт [15]. Розробники API можуть реалізувати REST різними способами, але основною концепцією такого рішення являє собою стандартизований та гнучкий підхід до розроблення веб-додатків, ці результати досягаються завдяки такому функціоналу:

- Кожного раз при запиті на сервер, у відповідь записується уся необхідна та актуальна інформація яка міститься на сервері.
- Строго типізований інтерфейс, який використовує доступ до ресурсів через кінцеві точки . Це допомагає у зручності при роботі з даним сервером.
- Підтримка хешування на всіх рівнях відповідей зменшує кількість непотрібних запитів з однаковими даними, що позитивно впливає на швидкість реагування системи та мінімізує трафік серверу.

Притримуючись базових правил REST API у реалізації серверної частини було створено ряд кінцевих точок, які дозволяють взаємодіяти з різними ресурсами, такими як користувачі, проєкти та задачі:

Таблиця 3.4 – Таблиця запитів по правилам REST API

Запит	Тип запиту	Призначення
auth		
/auth/login	POST	Авторизація користувача
/auth/register	POST	Реєстрація користувача
/auth/me	GET	Перевірка реєстрації користувача
projects		
/projects	GET	Отримання списку всіх проєктів
/projects/:id	GET	Отримання проєкту по унікальному ідентифікатору
/projects	POST	Створити проєкт

Продовження таблиці 3.4

/projects/:id	DELETE	Видалити проєкт по унікальному ідентифікатору
/projects/:id	PATCH	Змінити проєкт
tasks		
/tasks/:id	GET	Отримання завдання по унікальному ідентифікатору
/tasks	GET	Отримання списку всіх завдань
/tasks	POST	Створення завдання
/tasks/:id	PATCH	Редагування завдання за унікальний ідентифікатором
/tasks/:id	DELETE	Видалення завдання

REST API використовує HTTP методи, такі як GET, POST, PATCH та DELETE, для здійснення операцій CRUD (створення, отримання, оновлення та видалення) над ресурсами. При успішних операціях з сервером, він повертає статус коди запитів, що дає змогу зрозуміти чи успішно виконався запит, чи ні.

3.1.2 Клієнтська частина

З самого початку для створення сучасних веб-додатків з бібліотекою React було використано збиральником проєкту Vite. Для цього було використано команду: **yarn create vite.**

Результатом виконання цієї команди буде створення проєкту, із стандартним набором папок та стилей (рис. 3.1)

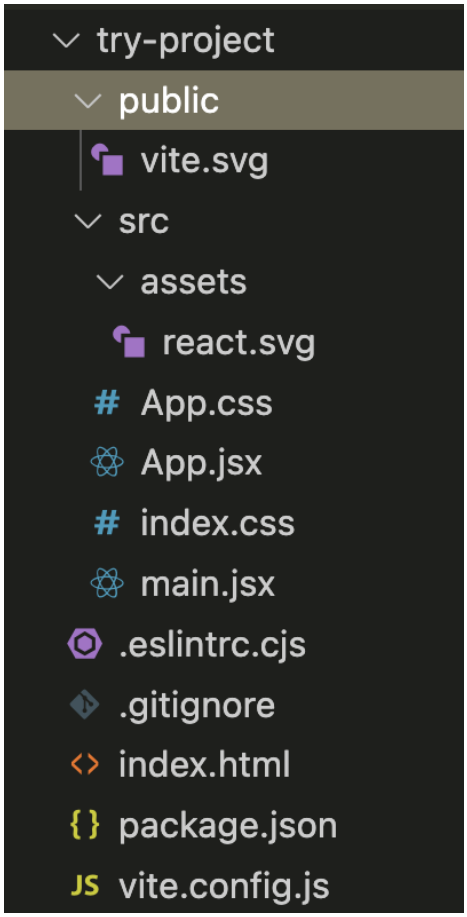

Рисунок 3.1 – Початковий стан створення проєкту

Після того як стартова версія веб-додатку створена, було обрано таку структуру папок проєкту, яка містить у собі:

- assets – папка для зберігання всіх додаткових файлі, зображень, стилів тощо.
- public – основні файли для роботи з додатком, дану папку було використано для зберігання об’єктів для перекладу сайту та основних зображень.
- components – компоненти програми, які можуть використовуватись у всіх інших компонентах.
- pages – компоненти сторінок, які використовуються тільки у головній компоненті App і лише один раз.
- redux – папка для зберігання конфігураційних файлів для Redux Toolkit

- services – папка для зберігання сервісів, за допомогою яких можна буде виконувати HTTP запити до серверної частини та шляхом простого імпорту до функціональних компонентів React, отримувати дані.
- dist – папка з файлами, які згенеровані збиральником проєкту та трансформовані зі зручного формату для розробника у формат, який буде більш ефективним для браузерного двигуна.

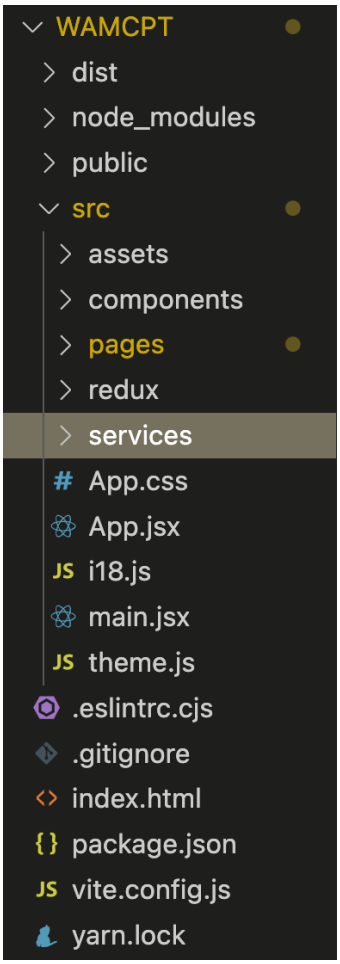

Рисунок 3.2 – Структура проєкту

Для побудови користувацького інтерфейсу та спілкування з серверною частиною було використано React та JavaScript які працюють в парі і їх комбінація дозволяє ефективно робити запити та відображувати актуальну інформацію. На серверну частину потрібно надсилати інформацію для пошуку, видалення та виконання всіх операцій зі станом веб-додатку, і для цього нам потрібне глобальне місце зберігання даних на клієнтській частині.

Для вирішення цього питання було використано пакет Redux Toolkit. Для реалізації даного підходу було використано такий архітектурний паттерн як FLUX. Особливість полягає в тому, що потрібно зберігати стан додатку в одному місці, туди ж записувати нові дані з серверу і тим самим контролювати обмін даними.

Flux – це архітектура додатків, яку розробники компанії Facebook використовують для створення клієнтських веб-додатків за допомогою React [16]. У парі з React така архітектура утворює концепцію однонаправленого потоку даних. Основні компонентами, які потрібні для FLUX – архітектури:

- Dispatcher – обробник дій який керує потоком даних веб-додатку та розподіляє дії до відповідних сховищ.
- Action – метод для зміни стану та отримання даних. Такі методи є звичайними об’єктами, які складаються з необхідної інформації про дію, яку потрібно виконати.
- Store – об’єкт для збереження стану всього додатку. Redux Toolkit спрощує створення та керування базовими функціями.
- View – компоненти які відображають дані зі сховища. Їх оновлення відбувається після зміни даних і об’єкті Store.

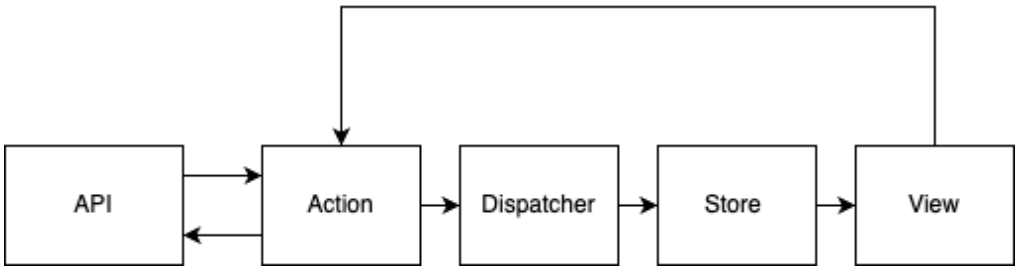

Рисунок 3.3 – Архітектура FLUX

Завдяки використанню FLUX-архітектури та Redux Toolkit ми можемо ефективно керувати станом нашого веб-додатку та забезпечити його оптимальну роботу. Також цей підхід полегшує етап розробки, зберігання та оновлення додатку.

Для виконання HTTP-запитів до серверної частини було використано бібліотеку Axios. З допомогою цього інструменту було розроблено ефективний функціонал з перехопленням стану запиту, та запис його у глобальний об’єкт Store. Таким чином значно спрощується робота над запитами, так як у відповідь на запит приходить Promise з конкретним станом та даними.

3.2 Використання програмного додатку

Основною метою використання веб-додатку є полегшення контролю розробки програмних продуктів шляхом розділення завдань на категорії та їх контроль. При вході у веб-додаток, користувача зустрічає ознайомлююча сторінка, яка інформує його для чого потрібен даний веб-додаток. Також освітлені основні функції системи.

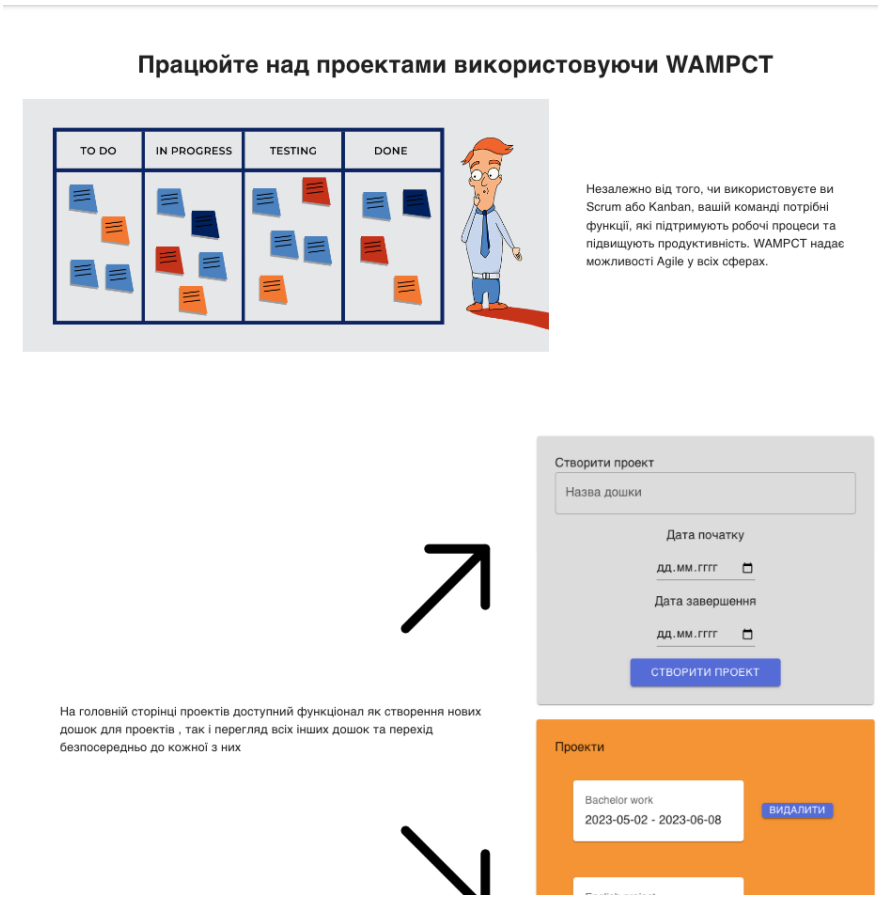

Рисунок 3.4 – Головна сторінка веб-додатку

Для зручності користування було проведено інтернаціоналізацію веб-додатку, а саме переклад на українську та англійську мови. Для зміни мови інтерфейсу достатньо використати функціонал, який знаходиться в шапці сайту.

Рисунок 3.5 – Вибір англійської мови

Рисунок 3.6 – Вибір української мови

Зміна мови у React дозволяє адаптувати інтерфейс веб-додатку для підтримки кількох мов. Вбудованої функції для цього не передбачено, тому було використано бібліотеку i18n, яка вирішила проблему керування мовами. Для того щоб реалізувати такий функціонал було створено файли з перекладами на кожну мову яка присутня в інтерфейсі.

Наступним логічним кроком для користувача буде або реєстрація у системі, або вхід у систему використовуючи особисті дані для авторизації.

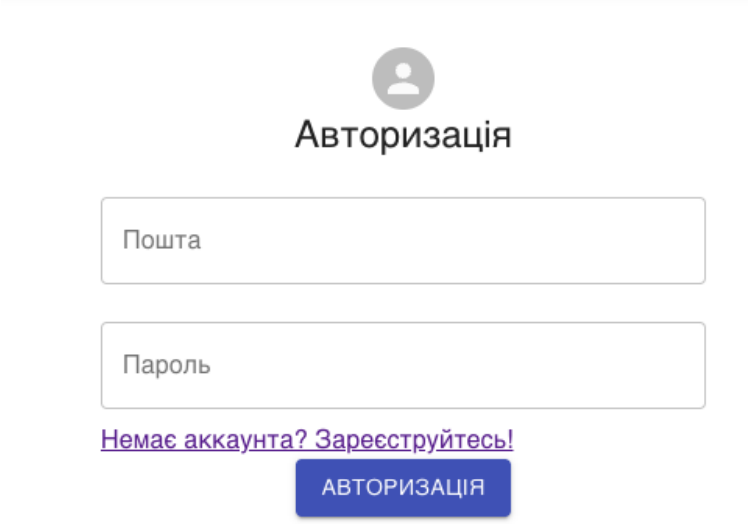

Рисунок 3.7 – Форма авторизації

Для успішної авторизації у системі, потрібно ввести коректну пошту та пароль. У випадку вказання невалідого паролю або пошти при спробі авторизації система попереджає, що користувачу було відмовлено у доступі в систему (рис. 3.6):

Рисунок 3.8 – Невдала спроба авторизації

Якщо користувач не зареєстрований, то він може потрапити на сторінку реєстрації шляхом натискання на допоміжний текст на сторінці авторизації, або в шапці веб-додатку натиснувши відповідну кнопку.

На сторінці реєстрації, система має такі поля для реєстрації як Ім’я та прізвище, Пошта та Пароль. Цих даних достатньо для створення аккаунту та переходу до використання функціоналу додатку.

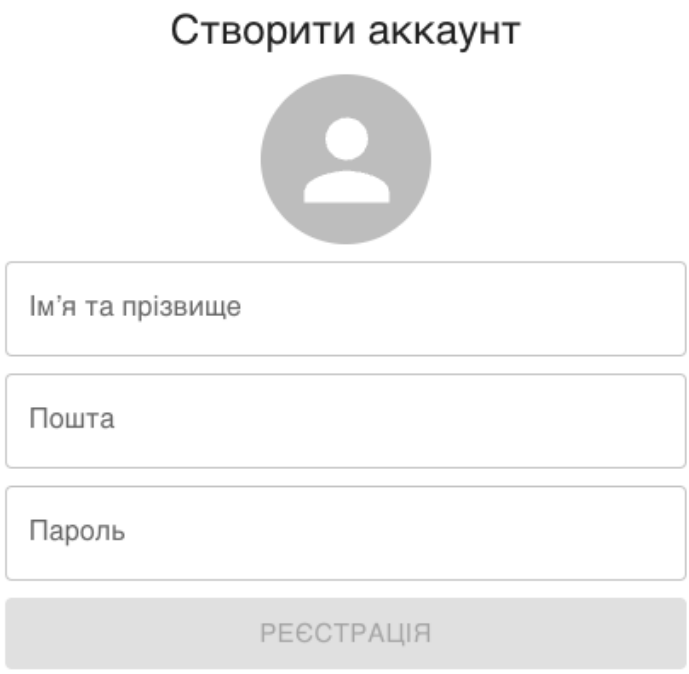

Рисунок 3.9 – Форма реєстрації без вхідних даних

Форма реєстрації буде неактивна до тих пір, доки користувач не введе валідні дані. З боку користувача такий функціонал додає зручності, так як зрозуміло що реєстрація неможлива, доки форма не заповнена.

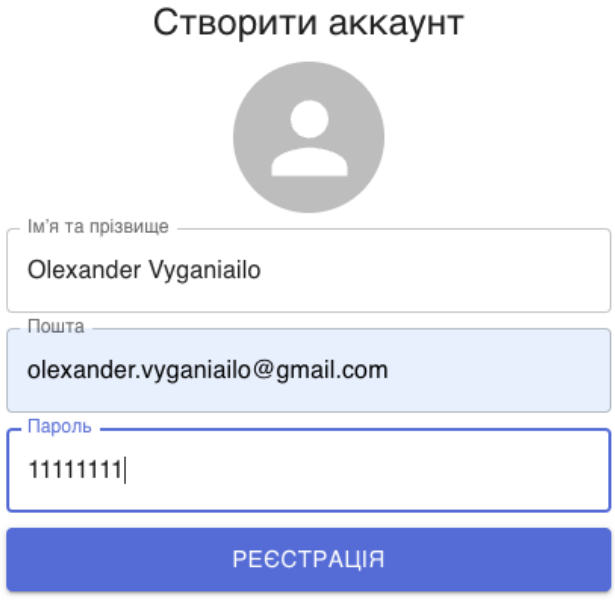

Рисунок 3.10 – Форма реєстрації при введені всіх даних

Після натискання на кнопку реєстрації, користувач автоматично реєструється у системі та потрапляє на голову сторінку проєкту, на якій він може ознайомитись з основним призначення веб-додатку та перейти до списку всіх дошок для керування проєктами.

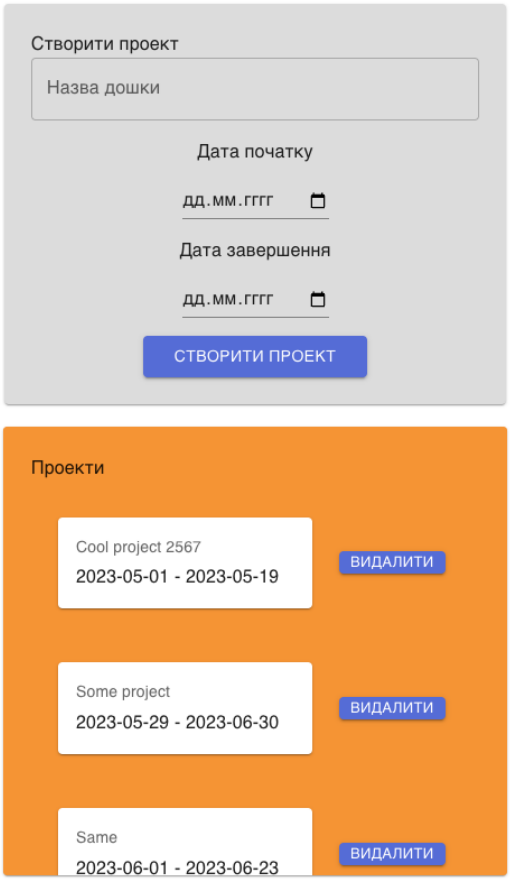

Рисунок 3.11 – Сторінка перегляду та створення дошок проєктів

У першій частині сторінки знаходиться форма для створення нової дошки завдань проєкту. Дана форма містить у собі поля для введення назви проєкту, дати початку та кінця. Під полями вводу знаходиться кнопка для створення дошки . Після заповнення форми необхідними даними і натискання кнопки "Створити проект", його буде створено і він з’явиться і другій частині сторінки.

Друга частина сторінки містить список проєктів. Кожен проєкт представлений як елемент списку, який включає назву проєкту, дату початку та дату кінця. Крім того, кожен елемент списку має декілька опцій, таких як

"Перейти до проекту" шляхом простого натискання на назву проєкту і "Видалити". Функціонал переходу до проєкту дозволяє вам перейти до сторінки конкретної дошки проєкту, де ви можете переглянути більше деталей, виконувати завдання та здійснювати інші дії, пов'язані з проєктом. Кнопка "Видалити" дозволяє видалити проєкт зі списку.

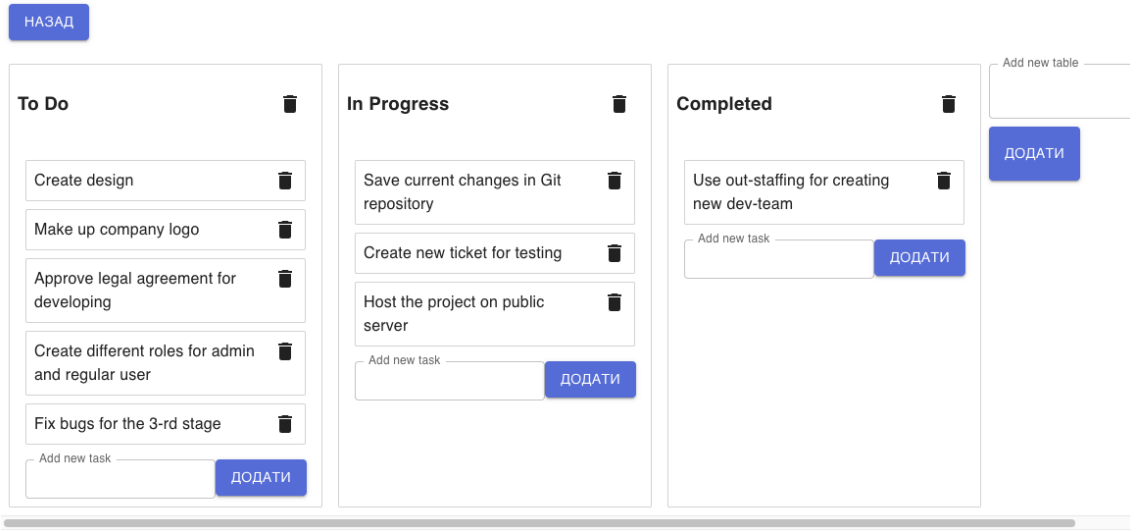

Рисунок 3.12 – Сторінка окремої дошки проєкту

На верхній частині дошки розташовуються назви стовпців. Наприклад, у першому стовпці може бути назва "To Do" (що означає "ще не розпочаті завдання"), другий стовпець може мати назву "In Progress" (означає "завдання в процесі виконання"), а третій стовпець - "Completed" (означає "завершені завдання"). Користувач можете мати будь-яку кількість стовпців і називати їх відповідно до конкретної потреби проєкту.

Кожен стовпець містить список завдань. Завдання можуть бути представлені як окремі картки, де кожна карта представляє окреме завдання. Картки містять опис завдання та функціонал видалення цього завдання зі списку.

Окрім того, на цій сторінці є функціонал додавання та видалення стовпців. Якщо користувач бажає додати новий стовпець, для цього присутня кнопка та поле для вводу назви, на яку можна натиснути. Після натискання на

неї користувач зможе ввести назву нового стовпця. Так само, для видалення стовпця присутня іконка видалення, яку користувач можете використовувати для видалення непотрібних стовпців.

Також, на кожній картці може є функціонал видалення окремого завдання. Якщо користувач хоче видалити конкретне завдання, то для цього присутня кнопка видалення, на він ви можете натиснути. Після цього, завдання буде вилучено зі списку і видалено з системи.

Дана сторінка веб-орієнтованої інформаційної системи надає зручний інтерфейс для організації завдань у вигляді дошки проєкту з відповідними стовпцями та можливістю додавання, редагування та видалення як стовпців, так і окремих завдань. Користувач системи може легко переміщати завдання між стовпцями, оновлювати їх статус та ефективно керувати проєктами, використовуючи цей веб-додаток.

ВИСНОВКИ

У результаті виконання дипломної роботи було виконано огляд необхідного інструментарію для ефективного керування великою кількістю проєктів. На основі зібраних даних було сформовано основні вимоги для розробки веб-орієнтованої інформаційної системи для управління задачами розробки програмного забезпечення, такі як: функціонал створення окремих дошок для проєктів, додавання стовпців які характеризують статус завдання та створення завдань у відповідних стовпцях.

Було розглянуто загальний принцип створення веб-додатків, які використовують сучасні фреймворки, ефективні підходи, архітектури та визначено саме ті, які будуть оптимально працювати у поєднанні одна з одною та виконувати поставлені задачі.

Для створення веб-додатку було реалізовано окремо серверну та клієнтську частини для відділення бізнес логіки від відображення користувацького інтерфейсу. У ході виконання серверної частини, було створено додаток з використанням NodeJS та Express, реалізовано REST API та налаштовано базу даних MongoDB. На клієнтській частині було створено основні компоненти використовуючи React, розроблено функціонал авторизації користувача за використанням JWT токену, додано функціонал створення та відстеження задач проєкту, збереження та відновлення стану додатку під час перезавантаження сторінки та перехід між різними сторінками за основною концепцією SPA.

Результатом виконання випускної кваліфікаційної роботи є веб-орієнтовна інформаційна система, яка дозволяє керувати задачами розробки програмного забезпечення у командах та забезпечує зручне використання системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методології управління проєктами, або Що таке Waterfall, Agile та Scrum. URL: https://devisu.ua/uk/stattia/metodologii-upravlinnya-proktami-abo-shcho-take-waterfall-agile-ta-scrum (дата звернення: 01.05.2023)

2. Trello brings all your tasks, teammates, and tools together. URL: https://trello.com/en (дата зверення: 01.05.2023)

3. Jira Software Cloud Premium. URL: https://www.atlassian.com/en/software/jira (дата звернення: 03.05.2023)

4. Що таке html? : веб-сайт. URL: https://css.in.ua/article/shcho-take-css_3 (дата звернення: 02.05.2023)

5. What is CSS? URL: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS (дата звернення: 05.05.2023)

6. About Node.js. URL: https://nodejs.org/en/about (дата звернення: 07.05.2023)

7. John Larsen. React Hooks in Action: With Suspense and Concurrent Mode. Вид-во Manning Publications Co. 2021. 376 с.

8. Adam Boduch. React Material-UI Cookbook. Вид-во PACKT Publishing. 2019. 534 с.

9. Vite JS - Next Generation Frontend Tool. Maitray Gadhavi. URL : https://radixweb.com/blog/vite-js-latest-front-end-development-tool (дата звернення: 03.06.2023)

10. Database management system. URL: https://www.techtarget.com/searchdatamanagement/definition/database-management-system (дата звернення: 10.05.2023)

11. Введення в базу даних SQLite. URL: http://yoip.com.ua/vvedennya-v-bazu-danih-sqlite/ (дата звернення:15.05.2023)

12. Розробка ECOMMERCE проектів PostgreSQL. URL: https://brander.ua/technologies/postgresql (дата звернення: 16.05.2023)

13. What is MongoDB? URL: https://www.ibm.com/topics/mongodb (дата звернення: 18.05.2023)

14. Rick Copeland. MongoDB Applied Design Patterns: Practical Use Cases with the Leading NoSQL Database. Вид-во O'Reilly Media. 2013. 155 с.

15. What is a REST API? URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api (дата звернення: 31.05.2023)

16. React Flux Concept. URL: https://www.javatpoint.com/react-flux-concept (дата звернення: 01.06.2023)

ДОДАТОК А

index.js – головний файл серверної частини

```
import express from "express";
import mongoose from "mongoose";
import cors from "cors";

import {
    registerValidation,
    loginValidation,
    projectCreateValidation,
} from "./validations/validation.js";

import { checkAuth, handleValidationErrors } from "./utils/index.js";
import {
    UserController,
    ProjectController,
    TasksController,
} from "./contollers/index.js";

mongoose
    .connect(

"mongodb+srv://olexandervyganiailo:whoisalex16@cluster0.oocnttz.mongodb.net/tasks?retr
yWrites=true&w=majority"
    )
    .then(() => console.log("DB OK"))
    .catch((err) => console.log("DB error", err));

const app = express();

app.use(express.json());
app.use(cors());
app.use("/uploads", express.static("uploads"));

app.post("/uploads", checkAuth, upload.single("image"), (request, response) => {
    response.json({
        url: `/uploads/${request.file.originalname}`,
    });
});

app.post(
    "/auth/login",
    loginValidation,
    handleValidationErrors,
    UserController.login
);
app.post(
    "/auth/register",
    registerValidation,
    handleValidationErrors,
    UserController.register
);
```

```
app.get("/auth/me", checkAuth, UserController.getMe);

app.get("/projects", ProjectController.getAll);
app.get("/projects/:id", ProjectController.getOne);
app.post(
    "/projects",
    checkAuth,
    projectCreateValidation,
    handleValidationErrors,
    ProjectController.create
);
app.delete("/projects/:id", checkAuth, ProjectController.deleteOne);
app.patch(
    "/projects/:id",
    checkAuth,
    handleValidationErrors,
    ProjectController.update
);

app.get("/tasks/:id", TasksController.getOneTask);
app.get("/tasks", TasksController.getAllTasks);
app.post("/tasks", TasksController.createTask);
app.patch("/tasks/:id", TasksController.updateTask);
app.delete("/tasks/:id", TasksController.deleteTask);

app.listen(4001, (error) => {
    if (error) {
        return console.log(error);
    }
    console.log("Server OK");
});
```

ДОДАТОК Б

Контроллери колекцій

ProjectsController.js

```
import ProjectModel from "../models/Project.js";

export const getAll = async (request, response) => {
    try {
        const projects = await ProjectModel.find();

        response.json(projects);
    } catch (error) {
        console.log(error);
        response.status(500).json({
            message: "Failed to get all projects",
        });
    }
};

export const getOne = async (request, response) => {
    try {
        const document = await ProjectModel.findOne({ _id: request.params.id });
        if (!document) {
            return response.status(404).json({
                message: "The project didn't found",
            });
        }
        response.json(document);
    } catch (error) {
        console.log(error);
        response.status(500).json({
            message: "Failed to get the project",
        });
    }
};

export const deleteOne = async (request, response) => {
    try {
        const document = await ProjectModel.findOneAndDelete({ _id: request.params.id
});
        if (!document) {
            return response.status(404).json({
                message: "The project didn't delete",
            });
        }
        response.json({ success: true });
    } catch (error) {
        console.log(error);
        response.status(500).json({
            message: "Failed to delete the project",
        });
    }
```

```
};

export const create = async (request, response) => {
    try {
        const doc = new ProjectModel({
            boardTitle: request.body.boardTitle,
            assignees: request.body.assignees,
            dateStart: request.body.dateStart,
            dateFinish: request.body.dateFinish,
        });

        const project = await doc.save();

        response.json(project);
    } catch (error) {
        console.log(error);
        response.status(500).json({
            message: "Failed to create project",
        });
    }
};

export const update = async (request, response) => {
    try {
        await ProjectModel.updateOne(
            {
                _id: request.params.id,
            },
            {
                boardTitle: request.body.boardTitle,
                assignees: request.body.assignees,
                dateStart: request.body.dateStart,
                dateFinish: request.body.dateFinish,
                tables: request.body.tables,
            }
        );

        response.json({
            success: true,
        });
    } catch (error) {
        console.log(error);
        response.status(500).json({
            message: "Failed to update the project",
        });
    }
};
```

TasksContoroller.js

```
import TasksModel from "../models/Tasks.js";

export const getAllTasks = async (request, response) => {
```

```
    try {
        const tasks = await TasksModel.find();

        response.json(tasks);
    } catch (error) {
        console.log(error);
        response.status(500).json({
            message: "Failed to get all tasks",
        });
    }
};

export const getOneTask = async (request, response) => {
    try {
        const document = await TasksModel.findOne({ _id: request.params.id });
        if (!document) {
            return response.status(404).json({
                message: "The tasks didn't found",
            });
        }
        response.json(document);
    } catch (error) {
        console.log(error);
        response.status(500).json({
            message: "Failed to get the task",
        });
    }
};

export const createTask = async (request, response) => {
    try {
        const doc = new TasksModel({
            taskTitle: request.body.taskTitle,
            project: request.body.project,
            taskDescription: request.body.taskDescription,
            status: request.body.status,
        });

        const task = await doc.save();

        response.json(task);
    } catch (error) {
        console.log(error);
        response.status(500).json({
            message: "Failed to create task",
        });
    }
};

export const updateTask = async (request, response) => {
    try {
        await TasksModel.updateOne(
            {
```

```
                _id: request.params.id,
            },
            {
                taskTitle: request.body.taskTitle,
                project: request.body.project,
                taskDescription: request.body.taskDescription,
                status: request.body.status,
            }
        );

        response.json({
            success: true,
        });
    } catch (error) {
        console.log(error);
        response.status(500).json({
            message: "Failed to update the tasks",
        });
    }
};

export const deleteTask = async (request, response) => {
    try {
        console.log(request.params);
        const document = await TasksModel.findOneAndDelete({
            _id: request.params.id,
        });
        if (!document) {
            return response.status(404).json({
                message: "The tasks didn't delete",
            });
        }
        response.json({ success: true });
    } catch (error) {
        console.log(error);
        response.status(500).json({
            message: "Failed to delete the task",
        });
    }
};
```

UserController.js

```
import jwt from "jsonwebtoken";
import bcrypt, { hash } from "bcrypt";

import UserModel from "../models/User.js";

export const register = async (request, response) => {
    try {
        const password = request.body.password;
        const salt = await bcrypt.genSalt(10);
        const hash = await bcrypt.hash(password.toString(), salt);
```

```
        const document = new UserModel({
            email: request.body.email,
            fullName: request.body.fullName,
            avatarUrl: request.body.avatarUrl,
            passwordHash: hash,
        });

        const user = await document.save();

        const token = jwt.sign(
            {
                _id: user._id,
            },
            "secret123",
            {
                expiresIn: "30d",
            }
        );

        const { passwordHash, ...userData } = user._doc;

        response.json({
            ...userData,
            token,
        });
    } catch (error) {
        console.log(error);
        response.status(500).json({
            message: "Fail of registration",
        });
    }
};

export const login = async (request, response) => {
    try {
        const user = await UserModel.findOne({ email: request.body.email });

        if (!user) {
            return response.status(404).json({
                message: "Unknown user or password",
            });
        }

        const isValidPass = await bcrypt.compare(
            request.body.password.toString(),
            user._doc.passwordHash.toString()
        );

        if (!isValidPass) {
            return response.status(403).json({
                message: "Wrong login or password",
            });
```

```
        }

        const token = jwt.sign(
            {
                _id: user._id,
            },
            "secret123",
            {
                expiresIn: "30d",
            }
        );

        const { passwordHash, ...userData } = user._doc;

        response.json({
            ...userData,
            token,
        });
    } catch (error) {
        console.log(error);
        response.status(500).json({
            message: "Fail to login",
        });
    }
};

export const getMe = async (request, response) => {
    try {
        const user = await UserModel.findById(request.userId);

        if (!user) {
            return response.status(404).json({
                message: "User not found",
            });
        }

        const { passwordHash, ...userData } = user._doc;

        response.json(userData);
    } catch (error) {
        console.log(error);
        response.status(500).json({
            message: "No permission",
        });
    }
};
```

ДОДАТОК В

Моделі колекцій

UserModel.js

```
import mongoose from "mongoose";

const UserSchema = new mongoose.Schema(
    {
        fullName: {
            type: String,
            required: true,
        },
        email: {
            type: String,
            required: true,
            unique: true,
        },
        passwordHash: {
            type: String,
            required: true,
        },
        avatarUrl: String,
    },
    {
        timestamps: true,
    }
);

export default mongoose.model("User", UserSchema);
```

Tasks.js

```
import mongoose from "mongoose";

const TasksSchema = new mongoose.Schema(
    {
        taskTitle: {
            type: String,
            required: true,
        },
        project: {
            type: String,
            required: true,
        },
        taskDescription: {
            type: String,
            required: true,
        },
        status: {
            type: String,
            required: true,
        },
```

```
    },
    {
        timestamps: true,
    }
);

export default mongoose.model("Tasks", TasksSchema);
```

Project.js

```
import mongoose from "mongoose";

const ProjectSchema = new mongoose.Schema(
    {
        boardTitle: {
            type: String,
            required: true,
        },
        assignees: {
            type: Array,
            required: true,
        },
        tables: {
            type: Array,
            required: true,
        },
        dateStart: {
            type: Date,
            required: true,
        },
        dateFinish: {
            type: Date,
            required: true,
        },
    },
    {
        timestamps: true,
    }
);

export default mongoose.model("Project", ProjectSchema);
```

ДОДАТОК Г

Файл з сервісами для запитів до серверної частини

```
import axios from "axios";
const BASE_URL = "http://localhost:4001";

const axiosInstance = axios.create({
    baseURL: BASE_URL,
});

axiosInstance.interceptors.request.use((config) => {
    config.headers.Authorization = window.localStorage.getItem("token");
    return config;
});

export const ProjectsRervice = {
    async getAllProjects() {
        try {
            const response = await axiosInstance.get("/projects");

            return response?.data;
        } catch (error) {
            console.log(error);
            return error;
        }
    },
    async postCreateProject(params) {
        try {
            console.log(params);
            const response = await axiosInstance.post("/projects", params);

            return response?.data;
        } catch (error) {
            console.log(error);
        }
    },
    async deleteProject(params) {
        try {
            console.log(params);
            await axiosInstance.delete(`/projects/${params}`);

            // return response?.data;
        } catch (error) {
            console.log(error);
        }
    },
    async updateProject(id, params) {
        console.log(params);
        try {
            const response = await axiosInstance.patch(`/projects/${id}`, {
                tables: params,
            });
```

```
            return response?.data;
        } catch (error) {
            console.log(error);
            return error;
        }
    },
};

export const TaskService = {
    async getAllTasks() {
        try {
            const response = await axiosInstance.get("/tasks");

            return response?.data;
        } catch (error) {
            console.log(error);
            return error;
        }
    },
    async updateTask(id, params) {
        try {
            const response = await axiosInstance.patch(`/tasks/${id}`, {
                status: params,
            });

            return response?.data;
        } catch (error) {
            console.log(error);
            return error;
        }
    },
    async createTask(params) {
        console.log(params);
        try {
            const response = await axiosInstance.post("/tasks", params);

            return response?.data;
        } catch (error) {
            console.log(error);
        }
    },
    async deleteTask(id) {
        try {
            await axiosInstance.delete(`/tasks/${id}`);
        } catch (error) {
            console.log(error);
        }
    },
};

export const UserService = {
    async postLoginUser(params) {
        try {
```

```
            const response = await axiosInstance.post("/auth/login", params);

            return response?.data;
        } catch (error) {
            console.log(error);
        }
    },
    async postRegisterUser(params) {
        try {
            const response = await axiosInstance.post("/auth/register", params);

            return response?.data;
        } catch (error) {
            alert("Fail to register");
            console.log(error);
        }
    },
    async getAuthMe() {
        try {
            const response = await axiosInstance.get("/auth/me");

            return response?.data;
        } catch (error) {
            console.log(error);
        }
    },
};
```

ДОДАТОК Ґ

Основні компоненти відображення

App.jsx

```
import { Container, CssBaseline } from "@mui/material";
import { Route, Routes } from "react-router-dom";
import "./App.css";
import Header from "./components/Header/Header";
import { Register } from "./pages/Register/Register";
import { Login } from "./pages/Login/Login";
import { Projects } from "./pages/Projects/Projects";
import { selectIsAuth, fetchAuthMe } from "./redux/slices/auth";
import { ProjectPage } from "./components/ProjectPage/ProjectPage";
import { useDispatch } from "react-redux";
import { useEffect } from "react";
import { useSelector } from "react-redux";
import { Home } from "./pages/Home/Home";

function App() {
    const dispatch = useDispatch();
    const isAuth = useSelector(selectIsAuth);
    useEffect(() => {
        dispatch(fetchAuthMe(isAuth));
    }, []);

    return (
        <>
            <CssBaseline />
            <Header></Header>
            <Container maxWidth="lg" style={{ paddingTop: "100px" }}>
                <Routes>
                    <Route path="/" element={<Home />} />
                    <Route path="/projects" element={<Projects />} />
                    <Route path="/projects/:id" element={<ProjectPage />} />
                    <Route path="/login" element={<Login />}></Route>
                    <Route
                        path="/register"
                        element={<Register>Register</Register>}
                    ></Route>
                </Routes>
            </Container>
        </>
    );
}

export default App;
```

Home.jsx

```
import { Button } from "@mui/material";
import { Typography } from "@mui/material";
import styled from "styled-components";
```

```
import { Link } from "react-router-dom";
import { useTranslation } from "react-i18next";

const Head = styled.h1`
    text-align: center;
`;

const Div = styled.div`
    display: flex;
    align-items: center;
    justify-content: space-around;
    margin-bottom: 100px;
`;

const DivReverse = styled(Div)`
    flex-direction: row-reverse;
`;

const Img = styled.img`
    max-width: 700px;
`;

export const Home = () => {
    const { t } = useTranslation();

    return (
        <>
            <Head>Працюйте над проектами використовуючи WAMPCT</Head>

            <Div>
                <Img src="./images/Kanban.png" />
                <Typography sx={{ padding: "50px" }}>
                    Незалежно від того, чи використовуєте ви Scrum або Kanban,
                    вашій команді потрібні функції, які підтримують робочі
                    процеси та підвищують продуктивність. WAMPCT надає
                    можливості Agile у всіх сферах.
                </Typography>
            </Div>
            <DivReverse>
                <Img src="/images/Projects.png" />
                <div
                    style={{
                        display: "flex",
                        flexDirection: "column",
                        alignItems: "end",
                    }}
                >
                    <Img
                        style={{ maxWidth: "200px" }}
                        src="/images/Arrow.png"
                    />
                    <Typography sx={{ padding: "50px" }}>
                        На головній сторінці проектів доступний функціонал як
```

```
                        створення нових дошок для проектів, так і перегляд всіх
                        інших дошок та перехід безпосередньо до кожної з них
                    </Typography>
                    <Img
                        style={{ maxWidth: "200px", rotate: "90deg" }}
                        src="/images/Arrow.png"
                    />
                </div>
            </DivReverse>
            <Div>
                <Img src="/images/Main.png" />
                <Typography sx={{ padding: "50px" }}>
                    Перетягування – це поширений метод взаємодії, доданий для
                    того, щоб люди могли інтуїтивно переміщати елементи на
                    сторінці. Це може бути зміна порядку в списку або навіть
                    складання пазла.
                </Typography>
            </Div>
            <Div>
                <Link
                    style={{
                        textDecoration: "none",
                        color: "inherit",
                    }}
                    to={"/projects"}
                >
                    <Button color="inherit" variant="outlined">
                        {t("project.start")}
                    </Button>
                </Link>
            </Div>
        </>
    );
};
```

Login.jsx

```
import { Link, Navigate } from "react-router-dom";
import { useForm } from "react-hook-form";
import { useDispatch, useSelector } from "react-redux";
import { fetchUser, selectIsAuth } from "../../redux/slices/auth";

import "../../i18.js";
import { useTranslation } from "react-i18next";

import {
    Button,
    Typography,
    Container,
    Box,
    Avatar,
    TextField,
    Grid,
```

```
} from "@material-ui/core";

export const Login = () => {
    const { t } = useTranslation();
    const isAuth = useSelector(selectIsAuth);
    const dispatch = useDispatch();

    const {
        register,
        handleSubmit,
        setError,
        formState: { errors, isValid },
    } = useForm({
        defaultValues: {
            email: "test2@gmail.com",
            password: "123456789",
        },
        mode: "onChange",
    });

    const onSubmit = async (values) => {
        const data = await dispatch(fetchUser(values));

        if (!data.payload) {
            return alert("Fail to authorize");
        }

        if ("token" in data.payload) {
            window.localStorage.setItem("token", data.payload.token);
        }
    };

    if (isAuth) {
        return <Navigate to="/" />;
    }

    return (
        <Container component="main" maxWidth="xs">
            <Box
                component="form"
                onSubmit={handleSubmit(onSubmit)}
                sx={{
                    marginTop: 8,
                    display: "flex",
                    flexDirection: "column",
                    alignItems: "center",
                }}
            >
                <Avatar sx={{ m: 1, bgcolor: "secondary.main" }}></Avatar>
                <Typography component="h1" variant="h5">
                    {t("login.title")}
                </Typography>
                <Box noValidate sx={{ mt: 1 }}>
```

```
                    <TextField
                        variant="outlined"
                        margin="normal"
                        fullWidth
                        label={t("login.email")}
                        autoFocus
                        error={Boolean(errors.email?.message)}
                        helperText={errors.email?.message}
                        {...register("email", {
                            required: "Please enter the email",
                        })}
                    />
                    <TextField
                        variant="outlined"
                        margin="normal"
                        fullWidth
                        label={t("login.password")}
                        type="password"
                        error={Boolean(errors.password?.message)}
                        helperText={errors.password?.message}
                        {...register("password", {
                            required: "Please enter the password",
                        })}
                    />
                    <Grid container>
                        <Grid item>
                            <Link to="/register">{t("login.noAccount")}</Link>
                        </Grid>
                    </Grid>
                </Box>
                <Button type="submit" color="primary" variant="contained">
                    {t("login.title")}
                </Button>
            </Box>
        </Container>
    );
};
```

Register.jsx

```
import { Stack, Typography, Avatar, TextField, Button } from "@mui/material";
import { useForm } from "react-hook-form";
import { useSelector, useDispatch } from "react-redux";
import { Navigate } from "react-router-dom";
import { fetchRegister, selectIsAuth } from "../../redux/slices/auth";
import { useTranslation } from "react-i18next";

export const Register = () => {
    const { t } = useTranslation();
    const isAuth = useSelector(selectIsAuth);
    const dispatch = useDispatch();

    const {
```

```
    register,
    handleSubmit,
    formState: { errors, isValid },
} = useForm({
    mode: "onChange",
});

const onSubmit = async (values) => {
    const data = dispatch(fetchRegister(values));

    if ("token" in data.payload) {
        window.localStorage.setItem("token", data.payload.token);
    }
};

if (isAuth) {
    return <Navigate to="/" />;
}

return (
    <Stack
        display="flex"
        flexDirection="column"
        gap="10px"
        justifyContent="center"
        maxWidth="400px"
        margin="0 auto"
        alignItems="center"
        onSubmit={handleSubmit(onSubmit)}
        component="form"
    >
        <Typography variant="h5">{t("register.create")}</Typography>
        <div>
            <Avatar sx={{ width: 100, height: 100 }} />
        </div>
        <TextField
            error={Boolean(errors.fullName?.message)}
            helperText={errors.fullName?.message}
            {...register("fullName", {
                required: "Please enter the full name",
            })}
            label={t("register.name")}
            fullWidth
        />
        <TextField
            error={Boolean(errors.email?.message)}
            helperText={errors.email?.message}
            {...register("email", {
                required: "Please enter the email",
            })}
            label={t("register.email")}
            fullWidth
        />
```

```
            <TextField
                error={Boolean(errors.password?.message)}
                helperText={errors.password?.message}
                {...register("password", {
                    required: "Please enter the password",
                })}
                label={t("register.password")}
                fullWidth
            />
            <Button
                disabled={!isValid}
                type="submit"
                size="large"
                variant="contained"
                fullWidth
            >
                {t("register.title")}
            </Button>
        </Stack>
    );
};
```

Projects.jsx

```
import { useDispatch, useSelector } from "react-redux";
import { useEffect } from "react";
import {
    deleteProject,
    fetchProjects,
    fetchTasks,
} from "../../redux/slices/project";
import {
    Typography,
    Button,
    Box,
    Card,
    TextField,
    Paper,
    List,
    Input,
    Stack,
} from "@mui/material";
import { ProjectCard } from "../../components/ProjectCard/ProjectCard";
import { Link } from "react-router-dom";
import { useForm } from "react-hook-form";
import { createProject } from "../../redux/slices/project";
import { useTranslation } from "react-i18next";
import { selectIsAuth } from "../../redux/slices/auth";
import { Navigate } from "react-router-dom";

export const Projects = () => {
    const isAuth = useSelector(selectIsAuth);
    const { t } = useTranslation();
```

```
const dispatch = useDispatch();
const { projects } = useSelector((state) => state.projects);

useEffect(() => {
    dispatch(fetchProjects());
    dispatch(fetchTasks());
}, []);

const {
    register,
    handleSubmit,
    formState: { errors },
} = useForm({
    mode: "onChange",
});

const onCreateProject = async (values, e) => {
    await dispatch(createProject(values));
    e.target.reset();
    dispatch(fetchTasks());
    dispatch(fetchProjects());
};

const onDeleteProject = async (values) => {
    await dispatch(deleteProject(values));
    dispatch(fetchProjects());
    dispatch(fetchTasks());
};

if (!isAuth) {
    return <Navigate to="/register" />;
}

return (
    <>
        <Stack alignItems={"center"}>
            <Card
                sx={{
                    maxWidth: "500px",
                    marginBottom: "20px",
                    backgroundColor: "#DCDCDC",
                    padding: 3,
                }}
            >
                <Box>{t("header.create")}</Box>
                <Stack
                    onSubmit={handleSubmit(onCreateProject)}
                    component="form"
                    sx={{
                        alignItems: "center",
                        width: "400px",
                    }}
                    spacing={2}
```

```
        >
            <TextField
                error={Boolean(errors.boardTitle?.message)}
                helperText={errors.boardTitle?.message}
                {...register("boardTitle", {
                    required: "Please enter the full name",
                })}
                label={t("header.boardTitle")}
                fullWidth
            />
            <label>{t("header.start")}</label>
            <Input
                error={Boolean(errors.dateStart?.message)}
                {...register("dateStart", {
                    required: "Please enter the full name",
                })}
                type="date"
            ></Input>
            <label>{t("header.finish")}</label>
            <Input
                error={Boolean(errors.dateFinish?.message)}
                {...register("dateFinish", {
                    required: "Please enter the full name",
                })}
                type="date"
            ></Input>
            <Button
                sx={{ maxWidth: "200px" }}
                type="submit"
                size="medium"
                variant="contained"
                fullWidth
            >
                {t("header.create")}
            </Button>
        </Stack>
    </Card>

    <Paper
        style={{
            maxHeight: 400,
            overflow: "auto",
            backgroundColor: "#f59434",
            padding: 25,
        }}
        sx={{ "&::-webkit-scrollbar": { display: "none" } }}
    >
        <Typography>{t("header.projects")}</Typography>
        <List>
            {projects.items.map((el) => (
                <Stack
                    direction={"row"}
                    alignItems="center"
```

```
                                key={el._id}
                                sx={{ width: "400px" }}
                            >
                                <Link
                                    to={`/projects/${el._id}`}
                                    style={{
                                        textDecoration: "none",
                                    }}
                                >
                                    <ProjectCard
                                        key={el}
                                        props={el}
                                    ></ProjectCard>
                                </Link>
                                <Button
                                    size="small"
                                    variant="contained"
                                    onClick={() => onDeleteProject(el._id)}
                                    sx={{ maxHeight: "20px" }}
                                >
                                    {t("header.delete")}
                                </Button>
                            </Stack>
                        ))}
                    </List>
                </Paper>
            </Stack>
        </>
    );
};
```

Header.jsx

```
import { AppBar, Box, Button, Stack, Toolbar, Typography } from "@mui/material";
import { Link } from "react-router-dom";
import { useDispatch, useSelector } from "react-redux";
import { useTranslation } from "react-i18next";
import { logout, selectIsAuth } from "../../redux/slices/auth";
import { useNavigate } from "react-router-dom";

const Header = () => {
    const { data } = useSelector((state) => state.auth);
    let navigate = useNavigate();

    const { t, i18n } = useTranslation();
    const changeLanguage = (lang) => {
        i18n.changeLanguage(lang);
    };

    const dispatch = useDispatch();
    const isAuth = useSelector(selectIsAuth);

    const logoutHandler = () => {
```

```
    if (window.confirm("Are you sure you want to log out")) {
        dispatch(logout());
        window.localStorage.removeItem("token");
        navigate("/login", { replace: true });
    }
};

return (
    <>
        <AppBar position="fixed" style={{ marginBottom: "100px" }}>
            <Toolbar>
                <Typography>
                    <Link
                        style={{ textDecoration: "none", color: "inherit" }}
                        to={"/"}
                    >
                        WAMPCT
                    </Link>
                </Typography>

                <Stack
                    direction={"row"}
                    spacing={2}
                    sx={{
                        marginLeft: "auto",
                        marginRight: "10px",
                        alignItems: "center",
                    }}
                >
                    {data?.fullName ? <Box>{data.fullName}</Box> : <></>}
                    <Button
                        onClick={() => {
                            changeLanguage("en");
                        }}
                        variant="contained"
                    >
                        EN
                    </Button>
                    <Button
                        onClick={() => {
                            changeLanguage("ua");
                        }}
                        variant="contained"
                    >
                        UA
                    </Button>
                </Stack>

                {!isAuth ? (
                    <>
                        <Box sx={{ margin: 2 }}>
                            <Link
                                style={{
```

```
                                        textDecoration: "none",
                                        color: "inherit",
                                    }}
                                    to={"/login"}
                                >
                                    <Button color="inherit" variant="outlined">
                                        {t("login.title")}
                                    </Button>
                                </Link>
                            </Box>

                            <Link
                                style={{
                                    textDecoration: "none",
                                    color: "inherit",
                                }}
                                to={"/register"}
                            >
                                <Button color="secondary" variant="contained">
                                    {t("register.title")}
                                </Button>
                            </Link>
                        </>
                    ) : (
                        <>
                            <Box sx={{ spacing: 5 }}>
                                <Link to="/projects">
                                    <Button
                                        color="secondary"
                                        variant="contained"
                                        sx={{ marginRight: "10px" }}
                                    >
                                        {t("header.projects")}
                                    </Button>
                                </Link>
                                <Button
                                    onClick={logoutHandler}
                                    color="secondary"
                                    variant="outlined"
                                >
                                    {t("login.logout")}
                                </Button>
                            </Box>
                        </>
                    )}
                </Toolbar>
            </AppBar>
        </>
    );
};

export default Header;
```

ДОДАТОК Д

Файли для створення Redux store

store.js

```
import { configureStore } from "@reduxjs/toolkit";
import { authReducer } from "./slices/auth";
import { projectsReducer } from "./slices/project";

const store = configureStore({
    reducer: {
        projects: projectsReducer,
        auth: authReducer,
    },
});

export default store;
```

auth.js

```
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";
import { UserService } from "../../services/api-services";

export const fetchUser = createAsyncThunk("auth/fetchUser", async (params) => {
    let user = {};
    await UserService.postLoginUser(params).then((data) => (user = data));
    return user;
});

export const fetchRegister = createAsyncThunk(
    "auth/fetchRegister",
    async (params) => {
        let user = {};
        await UserService.postRegisterUser(params).then(
            (data) => (user = data)
        );
        return user;
    }
);

export const fetchAuthMe = createAsyncThunk("auth/fetchAuthMe", async () => {
    let dataMe = {};
    await UserService.getAuthMe().then((data) => (dataMe = data));
    return dataMe;
});



const initialState = {
    data: null,
    status: "loading",
};
```

```
const authSlice = createSlice({
    name: "auth",
    initialState,
    reducers: {
        logout: (state) => {
            state.data = null;
        },
    },
    extraReducers: {
        [fetchUser.pending]: (state) => {
            state.status = "loading";
            state.data = null;
        },
        [fetchUser.fulfilled]: (state, action) => {
            state.status = "loaded";
            state.data = action.payload;
        },
        [fetchUser.rejected]: (state) => {
            state.status = "error";
            state.data = null;
        },

        [fetchAuthMe.pending]: (state) => {
            state.status = "loading";
            state.data = null;
        },
        [fetchAuthMe.fulfilled]: (state, action) => {
            state.data = action.payload;
            state.status = "loaded";
        },
        [fetchAuthMe.rejected]: (state) => {
            state.status = "error";
            state.data = null;
        },

        [fetchRegister.pending]: (state) => {
            state.status = "loading";
            state.data = null;
        },
        [fetchRegister.fulfilled]: (state, action) => {
            state.data = action.payload;
            state.status = "loaded";
        },
        [fetchRegister.rejected]: (state) => {
            state.status = "error";
            state.data = null;
        },
    },
});

export const selectIsAuth = (state) => Boolean(state.auth.data);

export const authReducer = authSlice.reducer;
```

```
export const { logout } = authSlice.actions;
```

projects.js

```
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";
import { ProjectsRervice, TaskService } from "../../services/api-services";

export const fetchProjects = createAsyncThunk(
    "projects/fetchProjects",
    async () => {
        let project = [];
        await ProjectsRervice.getAllProjects().then((data) => (project = data));
        return project;
    }
);

export const fetchTasks = createAsyncThunk("tasks/fetchTasks", async () => {
    let tasks = [];
    await TaskService.getAllTasks().then((data) => (tasks = data));
    return tasks;
});

export const createProject = createAsyncThunk(
    "projects/createProject",
    async (params) => {
        let project = {};
        await ProjectsRervice.postCreateProject(params).then(
            (data) => (project = data)
        );
        return project;
    }
);

export const deleteProject = createAsyncThunk(
    "projects/deleteProject",
    async (params) => {
        let project = {};
        await ProjectsRervice.deleteProject(params).then(
            (data) => (project = data)
        );
        return project;
    }
);

const initialState = {
    projects: {
        items: [],
        status: "loading",
    },
    tasks: {
        items: [],
        status: "loading",
```

```
    },
};

const projectsSlice = createSlice({
    name: "projects",
    initialState,
    reducers: {},
    extraReducers: {
        [fetchProjects.pending]: (state) => {
            state.projects.status = "loading";
        },
        [fetchProjects.fulfilled]: (state, action) => {
            state.projects.items = action.payload;
            state.projects.status = "loaded";
        },
        [fetchProjects.rejected]: (state) => {
            state.projects.items = [];
            state.projects.status = "error";
        },

        [fetchTasks.pending]: (state) => {
            state.tasks.status = "loading";
        },
        [fetchTasks.fulfilled]: (state, action) => {
            state.tasks.items = action.payload;
            state.tasks.status = "loaded";
        },
        [fetchTasks.rejected]: (state) => {
            state.tasks.status = "error";
        },

        [deleteProject.pending]: (state, action) => {
            state.tasks.items = state.tasks.items.filter(
                (obj) => obj._id === action.payload
            );
        },
    },
});

export const projectsReducer = projectsSlice.reducer;
```