

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Вебдодаток управління ІТ-проектами»

Виконала студентка групи ІПЗ-20-3
ІЩУК Дарія Вікторівна

Керівник роботи:
ГОРДІЄНКО Ірина Валеріївна

Консультант роботи:
ЧИЖМОТРИЯ Олена Геннадіївна

Рецензент:
ЄФРЕМОВ Юрій Миколайович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Завідувач кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ІЩУК Дарія Вікторівна**

Керівник роботи: **ГОРДІЄНКО Ірина Валеріївна**

Тема роботи: **«Вебдодаток управління ІТ-проектами»**,

затверджена Наказом закладу вищої освіти від «23» лютого 2024 р., №74с

Вихідні дані для роботи: **процес управління ІТ-проектами, що включає в себе всі аспекти планування, виконання та контролю проєктів в області інформаційних технологій.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Гордієнко І. В., Чижмотря О. Г.	27.02.2024р.	27.02.2024р.
2	Гордієнко І. В., Чижмотря О. Г.	27.02.2024р.	27.02.2024р.
3	Гордієнко І. В., Чижмотря О. Г.	27.02.2024р.	27.02.2024р.

РЕФЕРАТ

Кваліфікаційна робота бакалавра складається з програмного комплексу вебдодатку управління ІТ-проектами та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи містить текстову частину, викладену на 66 сторінках друкованого тексту, має 14 сторінок додатків. Список використаних джерел містить 30 найменувань і займає 3 сторінки. В роботі наведено 36 рисунків та 9 таблиць. Загальний обсяг роботи – 80 сторінок.

Метою роботи є розробка вебдодатку управління ІТ-проектами, спрямованого на підвищення продуктивності та зниження ризиків під час реалізації ІТ-проектів із використанням сучасних інструментів та засобів розробки.

У роботі було визначено вимоги користувача та обґрунтовано створення концептуальної моделі. Також було проаналізовано аналоги програмного забезпечення продуктів та обрано засоби реалізації, проведено розробку структури бази даних. Було розроблено алгоритм роботи, обрано архітектуру та розроблено об'єктно-орієнтовану структуру системи. Також описано програмну реалізацію сайту.

КЛЮЧОВІ СЛОВА: ВЕБДОДАТОК, УПРАВЛІННЯ, ПРОЕКТ, КЛІЄНТ, СЕРВЕР.

					ІПЗ.КР.Б – 121 – 24 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Вебдодаток управління ІТ-проектами	Літ.	Арк.	Аркушів
Розроб.	Д. В. Іщук						3	75
Керівник	І. В. Гордієнко					Житомирська політехніка, група ІПЗ-20-3		
Рецензент	Ю. М. Єфремов							
Зав. каф.	Т. А. Вакалюк							

ABSTRACT

The bachelor's thesis consists of an IT project management web application software package and an explanatory note. The explanatory note to the qualification work contains the text part, set out on 66 printed pages, and has 14 appendices. The list of references contains 30 titles and occupies 3 pages. The paper contains 36 figures and 9 tables. The total volume of the work is 80 pages.

The work aims to develop a web-based IT project management application that uses modern development tools and techniques to increase productivity and reduce risks during IT project implementation.

The paper identifies user requirements and justifies the creation of a conceptual model. Analogs of software products were also analyzed and implementation tools were selected, the database structure was developed, and an algorithm was developed, the architecture was selected, and the object-oriented structure of the system was developed. The site's software implementation is also described.

KEYWORDS: WEB APPLICATION, MANAGEMENT, PROJECT, CLIENT, SERVER.

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						4
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ.....	7
ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБ-ДОДАТКУ ДЛЯ УПРАВЛІННЯ ІТ- ПРОЄКТАМИ.....	10
1.1. Постанова задачі.....	10
1.2. Аналіз аналогів програмного продукту	12
1.3. Вибір архітектури веборієнтованої системи управління проєктами	16
1.3.1. Визначення архітектури клієнтської частини	17
1.3.2. Визначення архітектури серверної частини	18
1.3.3. Визначення способу зберігання даних.....	18
1.4. Обґрунтування вибору засобів реалізації	19
Висновки до першого розділу	20
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБОРІЄНОВАНОЇ СИСТЕМИ	22
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи	22
2.2. Розробка бази даних системи.....	28
2.3. Проектування та реалізація алгоритмів роботи системи	32
2.4. Реалізація додатку веборієнтованої системи управління ІТ- проєктами.....	34
Висновки до другого розділу.....	45
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБДОДАТКОМ ДЛЯ УПРАВЛІННЯ ІТ-ПРОЄКТАМИ.....	46
3.1. Порядок встановлення та налаштування параметрів системи	46
3.2. Структура інтерфейсу вебдодатку та порядок роботи	47
3.3. Тестування роботи програмного комплексу	60

Висновки до третього розділу	62
ВИСНОВКИ.....	63
СПИСОК ЛІТЕРАТУРИ.....	64
ДОДАТКИ.....	67

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПЗ – програмне забезпечення

БД – База даних

СУБД – Система управління базами даних

SQL – Structure Query Language

ПК – Первинний ключ

ЗК – Зовнішній ключ

HTML – Hyper Text Markup Language

UUID – Universally Unique Identifier

JSON – JavaScript Object Notation

JDBC – Java Database Connectivity

JWT – JSON Web Tokens

API – Application Programming Interface

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. Сучасні ІТ-проекти стають все складнішими, тому традиційні методи управління ними вже не ефективні. Для успішного їх ведення потрібен централізований та інтегрований підхід, який забезпечує спільний доступ до інформації, ефективну комунікацію, автоматизацію рутинних завдань, моніторинг та контроль ходу виконання, а також звітність та аналітику.

Вебдодатки управління ІТ-проектами пропонують ряд переваг, які роблять їх ідеальним рішенням для цього завдання. Їх можна використовувати з будь-якого місця та в будь-який час, вони зручні у користуванні, мають широкий спектр функцій, легко масштабуються та інтегруються з іншими інструментами та системами.

Розробка вебдодатку управління ІТ-проектами може допомогти організаціям значно покращити їх ефективність. Завдяки централізованому та інтегрованому підходу вебдодатки допомагають командам краще співпрацювати, економити час та ресурси, а також підвищити якість та результативність ІТ-проектів.

Метою роботи є розробка вебдодатку управління ІТ-проектами, спрямованого на полегшення планування, координацію завдань, відстеження прогресу та аналіз результатів з метою підвищення продуктивності та зниження ризиків при реалізації ІТ-проектів.

Встановлена мета обумовлює наступні завдання:

- Аналіз потреб та вимог користувачів вебдодатку управління ІТ-проектами.
- Визначення функціональних вимог до вебдодатка з урахуванням специфіки управління ІТ-проектами.
- Розробка архітектури вебдодатку, що забезпечує ефективне планування, координацію завдань та моніторинг прогресу проектів.
- Реалізація функціонала додатка з використанням сучасних технологій та інструментів розробки програмного забезпечення.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

- Проведення тестування вебдодатку для перевірки коректності його роботи та відповідності вимогам.
- Виправлення виявлених недоліків.

Об'єктом дослідження є процес управління ІТ-проєктами, що охоплює всі аспекти планування, виконання та контролю проєктів в області інформаційних технологій.

Предметом дослідження є розробка вебдодатку для управління ІТ-проєктами з використанням сучасних веборієнтованих технологій, які спрямовані на полегшення планування, координації завдань, моніторингу прогресу та аналізу результатів управління ІТ-проєктами. Предмет дослідження розглядається в межах процесу управління ІТ-проєктами та включає конкретні аспекти розробки та впровадження вебдодатку для оптимізації цього процесу.

Розроблена система може слугувати практичним інструментом для управління ІТ-проєктами у реальних умовах різних підприємств і організацій, які займаються розробкою або управлінням інформаційними технологіями, стартапами, а також для особистого використання.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБ- ДОДАТКУ ДЛЯ УПРАВЛІННЯ ІТ-ПРОЄКТАМИ

1.1. Постановва задачі

Робота над будь-яким завданням завжди вимагає правильно підходу та планування, щоб досягнути поставлених цілей. Управління проєктами є тяжким завданням не залежно чи це особистий проєкт, де бере участь одна людина, чи бізнес-проєкт націлений на виконання великою командою. Необхідно правильно виконувати планування, розподіляти завдання, відстежувати прогрес. Неправильна організація проєкту може призвести до затримок, нерозуміння у якому напрямку рухатися далі чи навіть до провалу проєкту. У зв'язку з цим завжди є потреба у використанні зручних інструментів для управління проєктами.

Вебдодаток управління ІТ-управління проєктами може допомогти розв'язати цю проблему надаючи усі необхідні інструменти для ефективної роботи над проєктами.

Ціллю створення вебдодатку для управління проєктами є спрощення процесу управління проєктами, створення програмних засобів що дозволять організувати робочий процес у проєктах та що сприятимуть ефективному виконанню проєктів, плануванню, розподілу та моніторингу виконання завдань, надання засобів для відстеження та аналізу прогресу проєктів.

Цільовою аудиторією вебдодатку є організації, команди, невеликі групи людей або поодинокі люди, що прагнуть:

- зручно керувати проєктами різної складності та масштабу;
- збільшити продуктивність та ефективність роботи над проєктами;
- поліпшити співпрацю між учасниками проєкту;

Для досягнення постановленої задачі, потрібно виконати наступні завдання:

- 1) Аналіз вимог;

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

- а) Визначення та аналіз існуючого програмного забезпечення з управління ІТ-проєктами;
 - б) Збір та аналіз потреб користувачів систем управління ІТ-проєктами;
 - с) Визначення функціональних та не функціональних вимог до системи;
 - д) Розробка сценаріїв використання та визначення основних потреб користувачів;
- 2) Проєктування інтерфейсу;
- а) Визначення структури та навігації системи;
 - б) Розробка макетів та дизайну користувацького інтерфейсу;
- 3) Проєктування бази даних;
- а) Визначення головних сутностей системи;
 - б) Реалізація механізмів забезпечення цілісності та безпеки даних;
- 4) Реалізація бізнес-логіки;
- а) Налаштування роботи з базою даних;
 - б) Розробка функціонала для виконання вимог;
 - с) Розробка засобів безпеки (авторизація та аутентифікація);
 - д) Оптимізація продуктивності;
- 5) Тестування;
- а) Проведення модульних та інтеграційних тестів
 - б) Корегування помилок та виправлення недоліків.
 - с) Проведення тестування вебдодатка в реальних умовах з використання різних сценаріїв використання.
- 6) Написання документації;
- а) Оформлення пояснювальної записки.

Кінцевим результатом виконання наведених задач повинна стати реалізована функціональна та ефективна система, яка надає засоби для роботи над проєктами: планування, виконання та відстежування.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

1.2. Аналіз аналогів програмного продукту

Під час проведення аналізу аналогів було знайдено велику кількість популярних сервісів, що мають свої особливості. Це зумовлено великим попитом на дане програмне забезпечення, особливо у сфері бізнесу

Одним найпопулярніших сервісів є Trello (рис.1.1) [1]. Це легкий інструмент для управління проєктами, що підходить для контролю роботи в невеликих компаніях та стартапах, або ж для особистого використання. Trello використовує систему дощок, карток та списків, що організована за принципом Kanban. Це робить його дуже зручним для візуалізації завдань та ходу роботи над проєктом.

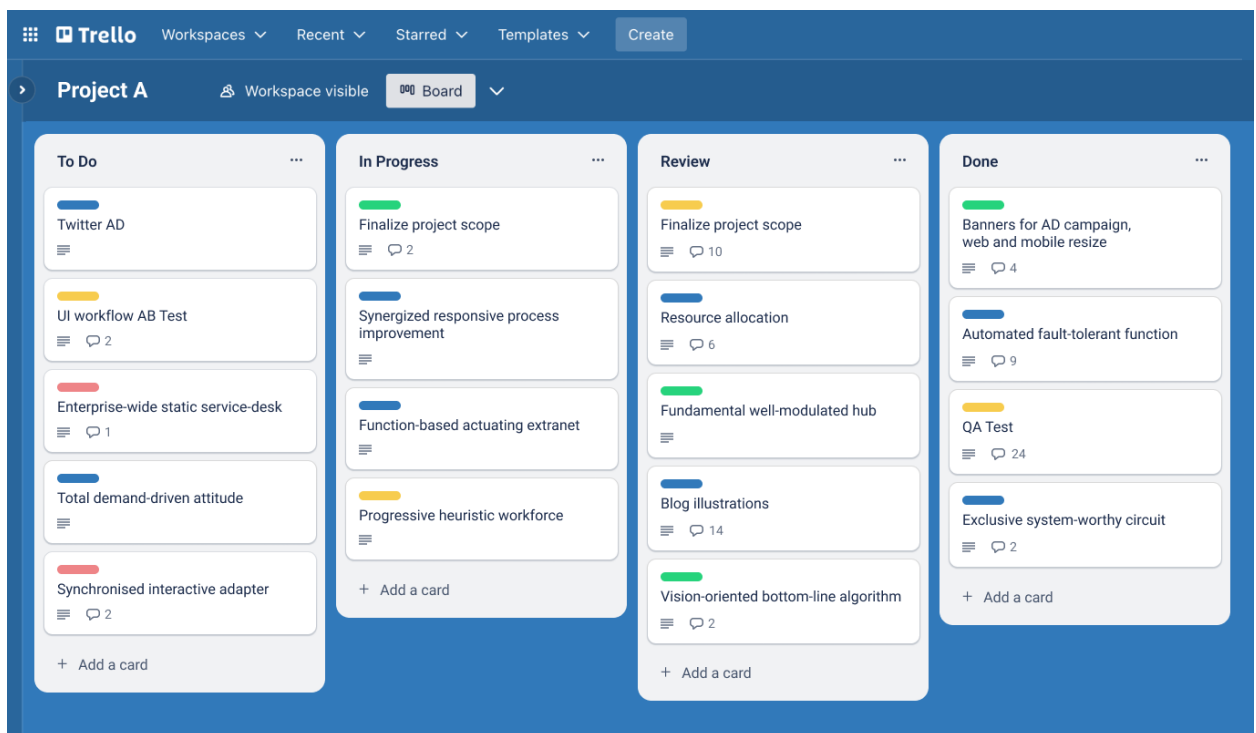


Рисунок 1.1 – Інтерфейс Trello

Переваги:

- *Гнучкість*: Trello можна легко налаштувати під особисті потреби, створюючи власні дошки, картки, списки та мітки.
- *Простота використання*: Trello дуже простий у вивченні та використанні, навіть для людей без досвіду роботи з інструментами для управління проєктами.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

Недоліки:

- *Малий функціонал:* Trello не має такого широкого спектра функцій, як деякі інші інструменти для управління проектами.
- *Може бути занадто простим:* Trello може бути занадто простим для складних проектів або команд, які потребують більш потужного інструменту.
- *Відсутність деяких важливих функцій:* Trello не має деяких важливих функцій, таких як можливість створення залежностей між завданнями або відстеження часу.

Іншим популярним сервісом є Asana (рис.1.2) [2]. Цей сервіс також працює за системою Kanban, але на відміну від Trello, орієнтований на команди середніх розмірів. Він побудований продумано, зосереджуючись на масштабованості та надійності своєї платформи, щоб користувачі завжди були готові до роботи.

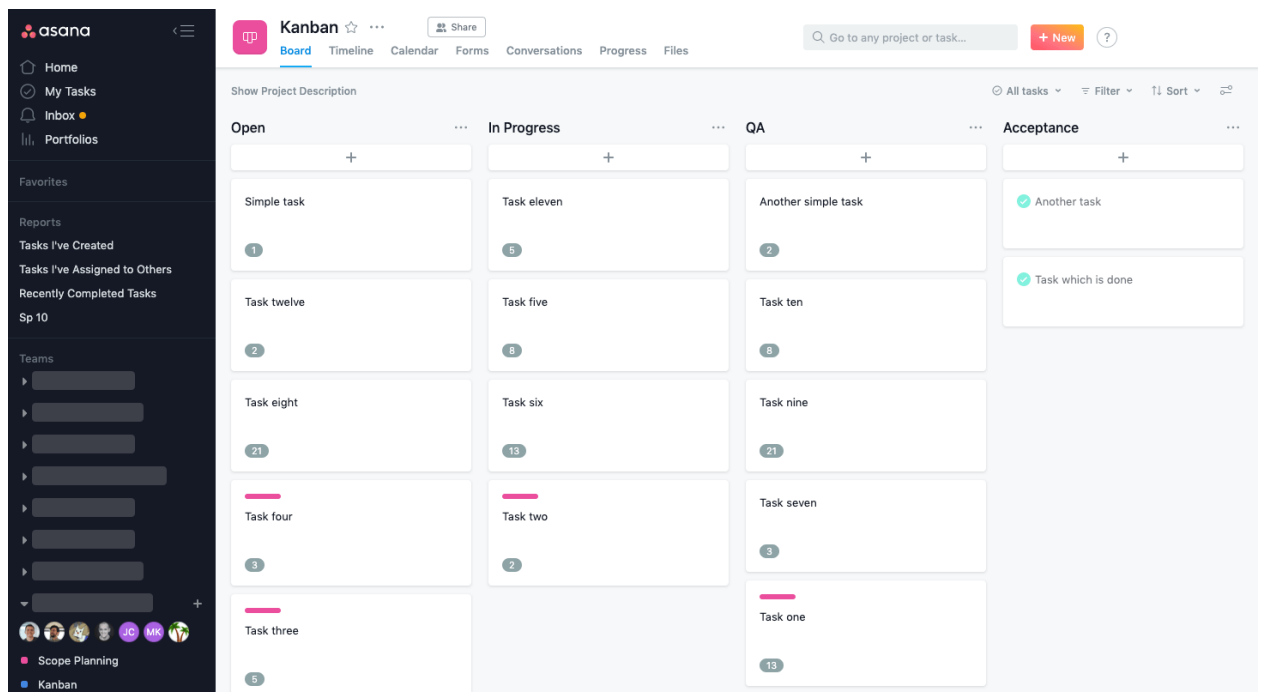


Рисунок 1.2 – Інтерфейс Asana

Переваги:

- *Широкий спектр функцій:* Asana пропонує широкий спектр функцій, таких як створення завдань, підзавдань, списків справ, делегування завдань, встановлення термінів, відстеження прогресу, отримання

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						13
Змн.	Арк.	№ докум.	Підпис	Дата		

сповіщень, створення шаблонів проєктів, інтеграція з іншими інструментами та багато іншого.

- *Відображення власних завдань*: Asana пропонує відображення власного списку завдань співробітника на день, тиждень, місяця у різних варіація (список, дошка або календар).
- *Гнучкість*: Asana можна легко налаштувати під особисті потреби, створюючи власні проєкти, секції, завдання, підзавдання, мітки та правила.

Недоліки:

- *Відсутність деяких візуальних функцій*: Asana не має такої ж візуальної привабливості, як Trello, що може ускладнити візуалізацію завдань та ходу роботи над проєктом.
- *Відсутність безкоштовної версії*: Asana надає пробний період використання їхнього сервісу. Також функціонал додатка залежить від типу підписки на сервіс.

Одним з найпопулярніших сервісів на ринку також є Jira (рис.1.3) [3]. Він націлений на розробку проєктів за методологією Agile – Scrum та Kanban. Jira адаптований під проєкти будь-якої складності, але найкраще розкриває при роботі над складними проєктами, що вимагають координацію між декількома командами.

Переваги:

- *Широкий спектр функцій*: Jira пропонує широкий спектр функцій, таких як створення завдань, епіків, відстеження помилок, завдань, створення звітів, створення та налаштування робочих процесів, інтеграція з іншими інструментами та багато іншого.

Недоліки:

- *Складний інтерфейс*: Jira є складною системою для опанування новачками. Потрібен деякий час її на вивчення.
- *Високі тарифи*: Для отримання максимального результату від роботи з Jira потрібний платний для тримання доступу до усього функціоналу.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

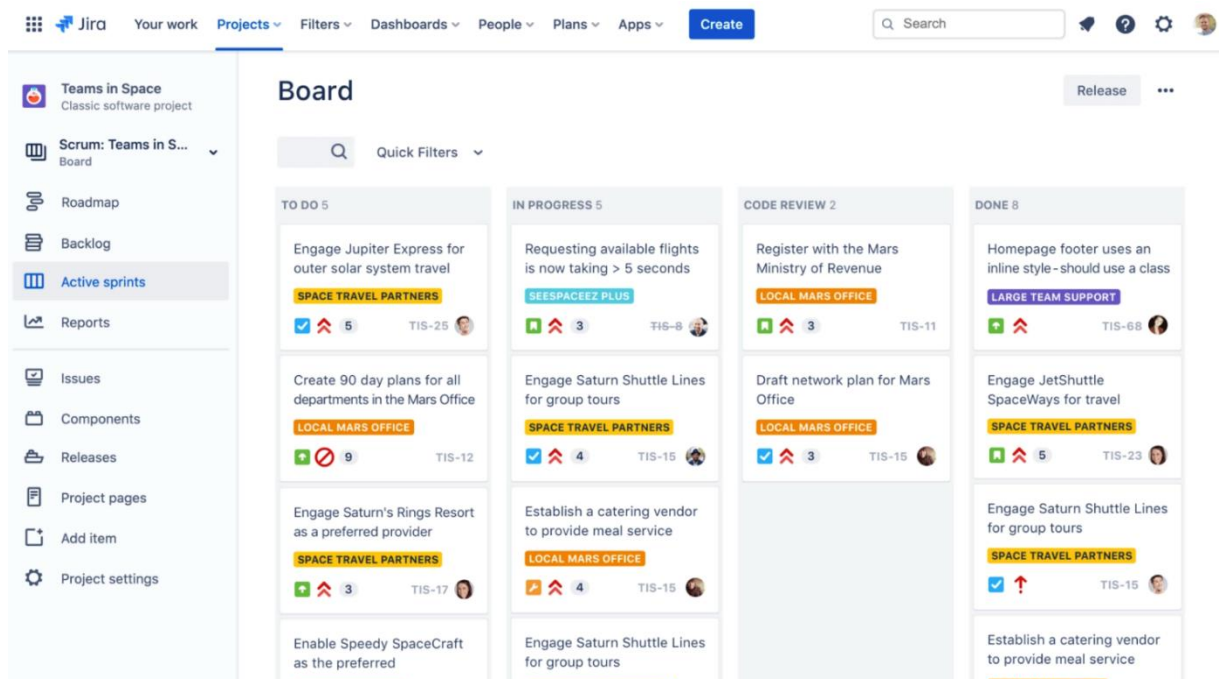


Рисунок 1.3 – Інтерфейс Jira

Усі вище перераховані сервіси націлені на реалізацію проєктів за методологією Agile. Проте, основною відмінністю за функціональність між цими сервісами є зосередженість на певний розмір команд та відповідно складність проєктів. У наслідку, можливості даних інструментів різняться.

Нижче наведено порівняльну таблицю доступних функцій у сервісах сервісів та функції (табл. 1.1).

Таблиця 1.1

Функціональне порівняння інструментів для управління ІТ-проєктами

	 Trello	 asana	 Jira Software
Створення та редагування проєктів	✗	✓	✓
Створення дошок	✓	✗	✗
Конфігурування дошок	✓	✓	✓
Створення і конфігурування завдань	✓	✓	✓
Робота в команді	✓	✓	✓
Статистика по проєкту	✗	✓	✓

<i>Система оповіщень</i>	✓	✓	✓
<i>Історія зміни елементів дошки</i>	✓	✗	✗

Зважаючи на те, що всі сервіси націлені на реалізацію проєктів за методологією Agile, то мінімальний функціонал повинен включати можливість створення та організації завдань, керування дошкою, а також можливість делегування завдань, встановлення пріоритетності та відстеження прогресу. Крім того, мінімальний функціонал повинен включати простий та зрозумілий інтерфейс для легкого використання, можливість візуалізації завдань та ходу роботи над проєктом, а також можливість створення звітів та інтеграції з іншими інструментами.

1.3. Вибір архітектури веборієнтованої системи управління проєктами

Для реалізації вебдодатка управління ІТ-проєктами було обрано трирівневу архітектуру [6]. Це добре налагоджена архітектура, яка організовує програми в три логічні та фізичні рівні: рівень презентації (інтерфейс користувача), прикладний рівень (обробка даних) та рівень даних (зберігання та управління даними).

Клієнтський додаток виконує роль рівня презентації, забезпечуючи інтерфейс користувача, наприклад, веб-сторінки або веб-сайти, де користувачі взаємодіють з системою.

Серверний додаток відповідає за середній рівень, де розміщена бізнес-логіка, яка обробляє введені дані користувачів, як-от запити до бази даних для отримання інформації або оновлення профілю користувача.

Сервер бази даних представляє рівень даних, де зберігаються дані програми та управляються ними, працюючи на програмному забезпеченні для керування базами даних.

Ця архітектура дозволяє чітко розділити обов'язки між усіма компонентами системи. Клієнт відповідає за інтерфейсу користувача та

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

взаємодію з користувачем, в той час, як сервер відповідає за обробку запитів, доступу до бази даних та виконання бізнес-логіки.

Взаємодія між клієнтом та сервером відбувається за допомогою HTTP-запитів та відповідей. Сервер виступає у ролі API та надає кінцеві точки, за якими клієнт насилає запити та отримує відповіді.

1.3.1. Визначення архітектури клієнтської частини

Двома головними архітектурними підходами для побудови веб-додатку є SPA (Single Page Application) та MPA (Multiple Page Application) [7].

SPA – це односторінковий веб-додаток, тобто весь контент завантажується один раз і подальша взаємодія виконується без повторного перезавантаження сторінки за рахунок динамічного перезаписування.

MPA – це багатосторінковий веб-додаток, який завантажує нову сторінку щоразу, коли виконується певна дія. Він складається з декількох окремих статичних HTML сторінок.

SPA пропонує швидку та зручну взаємодію з користувачем, за рахунок початкового завантаження сторінки, великий обсяг даних якої зберігається у кеші. Це є великою перевагою, оскільки дає змогу працювати в офлайн режимі. Проте, початкове завантаження сторінки може зайняти більше часу у порівнянні з MPA. Також SPA знижує навантаження на сервер, так як використовує API для отримання даних з серверу, а не вимагає генерування цілої сторінки кожного запиту. Багатосторінковий підхід також має низку переваг перед односторінковим. Він є класичним способом створення веб-сторінок, тому він сумісний із більшістю старих браузерів і застарілих систем.

Для веб-додатку управління проєктами MPA є кращим вибором. Він є більш безпечнішим, оскільки надсилає окремі запити для кожної сторінки, що полегшує провадження заходів безпеки, такі як автентифікація та авторизація. Це особливо важливо для веб-додатку управління проєктами, де конфіденційність та безпека даних є критичними. Крім того, MPA є більш

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						17
Змн.	Арк.	№ докум.	Підпис	Дата		

сумісним з більшістю старих браузерів і застарілих систем, що може бути важливим для користувачів, які використовують застаріле обладнання.

1.3.2. Визначення архітектури серверної частини

Двома найпоширенішими архітектурами програмного забезпечення є монолітна та мікросервісна [8]. У монолітній архітектурі вся серверна частина реалізована як єдиний компонент. Вона є легкою у розробці та реалізації, проте може бути важко масштабованою, якщо програма стане зовеликою. Ця проблема вирішується за допомогою мікросервісної архітектури, яка розбиває програму на менші та незалежні служби, що спілкуються між собою за допомогою API. Кожна служба відповідає за певну функцію, наприклад, авторизацію. Цей підхід забезпечує велику гнучкість системи, проте є складнішим у розробці та координації.

Для реалізації даного вебдодатка буде використано монолітну архітектуру попри її недоліки, оскільки обсяг функціоналу є невеликим порівняно до корпоративних систем, де використання мікросервісів є оптимальним варіантом через їх складність. Монолітна архітектура дозволяє швидше провести розробку, тестування та розгортання продукту, адже всі компоненти системи інтегровані в єдиній кодовій базі. Це спрощує процес розробки, оскільки не потрібно окремо налаштовувати та підтримувати взаємодію між численними мікросервісами. Для невеликих проєктів монолітна архітектура є більш зручною у підтримці, оскільки всі зміни та оновлення можна здійснювати централізовано.

1.3.3. Визначення способу зберігання даних

Два основних типи систем управління базами даних (СУБД), які широко використовуються, це реляційні та нереляційні [9]. Кожна з них має свої переваги та недоліки, які роблять вибір оптимального рішення залежним від конкретних потреб проєкту.

Реляційні бази даних організовують дані у таблиці з чітко визначеними зв'язками між ними. Ця структура забезпечує високу цілісність та

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						18
Змн.	Арк.	№ докум.	Підпис	Дата		

узгодженість даних, що робить їх ідеальними для застосунків, де потрібна точність та передбачуваність.

Нереляційні бази даних не мають жорсткої структури таблиць. Їх гнучкість робить їх ідеальними для зберігання різноманітних типів даних, включаючи структуровані, неструктуровані та напівструктуровані. Їх горизонтальна масштабованість робить їх придатними для обробки великих обсягів даних з низькими затримками.

У контексті розробки вебдодатку управління ІТ-проектами реляційна СУБД є оптимальним рішенням, оскільки вона ефективно зберігає дані з чіткими структурами та взаємозв'язками. Її структурований підхід до даних, потужні можливості запитів, довгострокова надійність та проста розробка, роблять її ідеальною для цієї мети. Нереляційні СУБД, незважаючи на свою гнучкість та масштабованість, не мають таких переваг, що робить їх менш придатними для вебдодатку, який не працює з великими та складними об'ємами даних.

Фінальний дизайн програмного комплексу має такий вигляд (рис. 1.4):

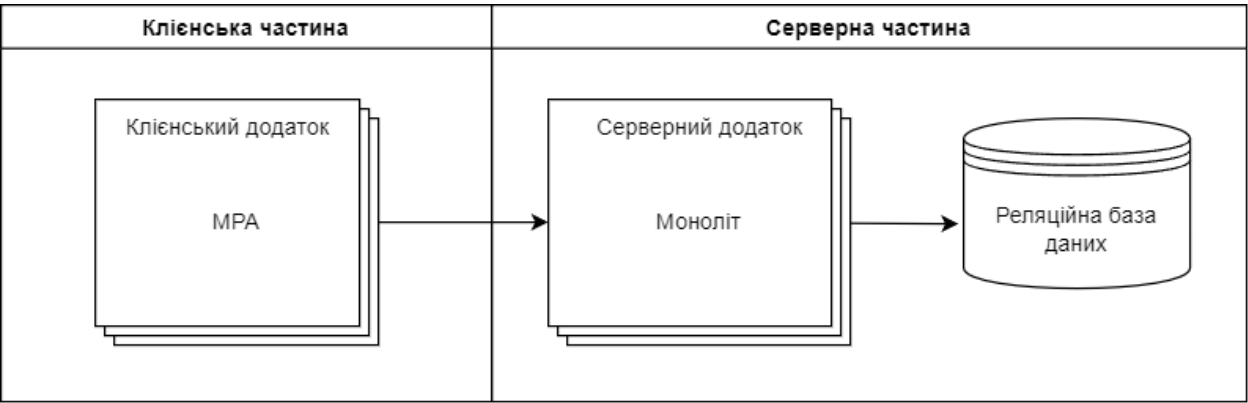


Рисунок 1.4 – Дизайн програмного комплексу

1.4. Обґрунтування вибору засобів реалізації

Для створення клієнтської частини буде використано JavaScript, оскільки це стандартна мова програмування для побудови інтерактивних та динамічних вебдодатків. Вона підтримується усіма сучасними веббраузерами та має велику спільноту серед розробників. Дизайн вебдодатку буде розроблено на основі бібліотек React [10] та Joy UI [11], які дають

можливість побути сучасний та зручний для користувача інтерфейс. React дозволяє створювати інтерфейс з використанням концепції компонентів – самостійних блоків коду, що відповідають за певну частину інтерфейсу. Joy UI, у свою чергу надає вже готові компоненти та шаблони для створення сторінок.

Серверна частина буде написано мовою Java [12] на базі фреймворку Spring [13] з використанням його інструментів, що дозволяють скоротити час розробки та покращити якість коду. Даний фреймворк є популярним для розробки серверних додатків, завдяки своїй великій кількості бібліотек та модулів, які дозволять швидко створювати базовий функціонал, наприклад авторизація та аутентифікація користувачів чи налагодження роботи з базами даних. Також, у Spring наявні інверсія контролю (Inversion of Control, IoC) та впровадження залежностей (Dependency Injection, DI) [14], за рахунок яких фреймворк бере на себе всю відповідальність за створення та управління об'єктами. Це значно полегшує процес розробки через зменшення зав'язків між класами, забезпечуючи гнучкості у конфігурації та зручності у тестуванні системи.

Для роботи з БД буде використано MySQL [15] – це реляційна система управління базами даних, яка відома своєю надійністю та ефективністю. Вона добре оптимізована для швидкої обробки запитів та має великий арсенал функцій, що можна використати при роботі з даними. Через наявність вбудованої підтримки у Spring СУБД можна легко інтегрувати за допомогою JDBC (Java Database Connectivity) [16] до проєкту.

Висновки до першого розділу

У першому розділі було поставлено задачу на основі якої визначено завдання та їх етапи виконання. Було проведено аналіз аналогів програмного продукту на базі найпопулярніших сервісів управління ІТ-проєктами Trello, Asana та Jira. З результатів порівняння було визначено переваги та недоліки

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

кожного сервісів та визначено мінімальний потрібний функціонал для створення програмного продукту.

Також було проаналізовано сучасні методи проектування веб додатків та обрано стек технологій розробки. Для реалізацій вебдодатку для управління проєктами було обрано клієнт-серверну архітектуру для реалізацій вебдодатку управління проєктами. Клієнтська частина буде реалізована як багатосторінковий додаток (MPA) за допомогою React та Joy UI, а серверна за допомогою монолітної архітектури на базі фреймворку Spring. Дані зберігатимуться у релятивній базі даних MySQL.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						21
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБОРІЄНОВАНОЇ СИСТЕМИ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Для забезпечення неперервного виконання необхідних функцій і відповідності встановленим стандартам та специфікаціям, наведено високорівневі вимоги, які повинна задовольняти веборієнтована система управління проєктами. Ці вимоги допомагають забезпечити безперебійну роботу системи та виконання всіх необхідних стандартів і специфікацій. Каталог вимог наведений в додатках.

Бізнес вимоги

1. Ціль проєкту: Створення вебдодатка для ефективного ІТ управління проєктами та завданнями з можливістю спільної роботи та аналізу статистики.
2. Представлення проєкту: Вебдодаток, який надає зручний інтерфейс для створення, керування та відстеження проєктів та завдань. Користувачі можуть реєструватися, створювати облікові записи проєктів, призначати завдання та спілкуватися через систему.

Вимоги користувачів

1. Інтуїтивний інтерфейс користувача.
2. Можливість створювати, редагувати та видаляти проєкти та завдання.
3. Персоналізовані списки завдань для кожного користувача.
4. Зручне управління завданнями з можливістю фільтрації.

Вимоги до користувацьких ролей

1. Роль адміністратора/керівника проєкту: має доступ до управління проєктами, визначення прав доступу та роботи користувачів в проєкті.
2. Роль користувача: має доступ до роботи з дошкою та завданнями.

Характеристика об'єкта комп'ютеризації

Користувач додатку буде мати можливість переглядати інформацію про проєкти, завдання, користувачів та їх взаємодію, а також можливість управління проєктами, призначення завдань, спілкування користувачів.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

Функціональні вимоги

1. Реєстрація та авторизація користувачів.
2. Створення, редагування та видалення проєктів.
3. Додавання та видалення користувачів до проєкту.
4. Зміна ролей користувачів у проєкті.
5. Створення та редагування завдань.
6. Відстеження завдань в межах проєктів.
7. Призначення завдань користувачам.
8. Додавання коментарів до завдання.
9. Налаштування стовпців дошки.
10. Фільтрація дошки.
11. Налаштування особистого профілю.
12. Видалення профілю.

Нефункціональні вимоги

1. Безпека: Захист особистих даних користувачів, захист від атак та збереження даних.
2. Швидкодія: Оптимізована продуктивність та відгук системи на дії користувачів.
3. Сумісність: Підтримка різних браузерів та пристроїв.
4. Зручність: Простий та зрозумілий інтерфейс для користувачів.
5. Масштабованість: Можливість розширення функціональності.

Аналіз потреб користувачів та визначення ключових ролей дали можливість сформулювати діаграми використання вебдодатку управління ІТ-проєктами від сторони користувача (рис. 2.1) та адміністратора проєкту (рис. 2.2).

Роль адміністратора у проєкті є розширеною роллю користувача з додатковими правами на редагування та видалення проєкту, а також керування учасниками проєкту. Адміністратор має змогу додавати нових користувачів до проєкту, призначати їм відповідні ролі, а також видаляти або змінювати існуючих учасників. Крім того, адміністратор може змінювати

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						23
Змн.	Арк.	№ докум.	Підпис	Дата		

налаштування проєкту, такі як його назва, опис та має доступ до видаленню проєкту.



Рисунок 2.1 – Варіанти використання додатку управління проєктами від ролей Користувача



Рисунок 2.2 – Варіанти використання додатку управління проєктами від ролі Адміністратора

На основі сформованих вище варіантів використання системи було створено основні прецеденти – опис конкретних сценаріїв або взаємодій користувачів з системою. Нижче наведено основні та найголовніші прецеденти для вебдодатку для управління проєктами.

Прецедент: Створення проєкту

Опис: Користувач може створити новий проєкт у системі.

Актори: Користувач.

Передумови: Користувач увійшов до системи.

Основний сценарій:

1. Користувач вибирає опцію "Створити проєкт".
2. Система відкриває форму для введення інформації про проєкт.
3. Користувач заповнює форму з необхідними даними (назва проєкту, опис, дедлайн тощо).
4. Користувач підтверджує створення проєкту.
5. Система зберігає інформацію про проєкт у базі даних.
6. Користувач отримує роль адміністратора для створеного проєкту.

Відхилення:

1. Користувач скасовує операцію створення проєкту.

Прецедент: Редагування проєкту

Опис: Можливість користувача або адміністратора проєкту редагувати інформацію про проєкт.

Актори: Адміністратор проєкту.

Передумови:

- Керівник увійшов у систему.
- Має доступ до редагування проєкту.

Основний сценарій:

1. Адміністратор обирає опцію "Редагувати проєкт".
2. Система відображає інформацію про проєкт та дозволяє внесення змін.
3. Адміністратор вносить зміни у назву, опис та іншу інформацію проєкту.
4. Адміністратор підтверджує зміни.
5. Система зберігає оновлені дані та повідомляє про успішне редагування.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						25
Змн.	Арк.	№ докум.	Підпис	Дата		

Відхилення:

1. Адміністратор скасовує операцію редагування проєкту.

Прецедент: Видалення проєкту

Опис: Можливість користувача або адміністратора видалити проєкт.

Актори: Адміністратор проєкту.

Передумови:

- Адміністратор увійшов у систему.
- Має право видаляти проєкти.

Основний сценарій:

1. Адміністратор обирає опцію "Видалити проєкт".
2. Система підтверджує намір керівника видалити проєкт.
3. Адміністратор підтверджує видалення проєкту.
4. Система видаляє проєкт та повідомляє про успішне видалення.

Відхилення:

1. Адміністратор скасовує операцію видалення проєкту.

Прецедент: Перегляд списку проєктів

Опис: Можливість користувача переглядати список усіх проєктів, до яких він має доступ.

Актори: Користувач.

Передумови: Користувач увійшов у систему.

Основний сценарій:

1. Користувач переходить до списку проєктів.
2. Система відображає список проєктів, до яких користувач має доступ.

Прецеденти зі створенню, редагування, та видалення завдань відбуваються аналогічно описаним вище прецедентам операцій над проєктами за участі акторів адміністратора проєкту чи користувача.

Прецедент: Призначення завдання користувачеві

Опис: Користувач може призначити завдання користувачеві в межах проєкту.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						26
Змн.	Арк.	№ докум.	Підпис	Дата		

Актори: Користувач.

Передумови: Користувач ввійшов до системи.

Основний сценарій:

1. Користувач вибирає проєкт, в якому він бажає призначити завдання іншому собі або іншому користувачеві.
2. Система відображає список користувачів, яким можна призначити завдання.
3. Користувач вибирає користувача.
4. Система зберігає інформацію про завдання та повідомляє користувача про нього.

Відхилення:

1. Керівник проєкту скасовує призначення завдання.

Спираючись на дані отримані та сформовані під час проведення аналізу вимог та способів використання програмного продукту, можна сформуванати базову діаграму класів (рис. 2.3). Вона складається з 5 головних класів:

1. User – представляє користувача системи та містить інформацію про користувача, таку як ідентифікатор, ім'я, електронна пошта, пароль, та список проєктів, до яких користувач належить.
2. Project – представляє проєкт та містить інформацію про проєкт, таку як ідентифікатор, назва, опис, ким створено проєкт та список користувачів, що беруть участь у проєкті.
3. ProjectUserRole – представляє роль користувача у проєкті та містить інформацію про роль, таку як ідентифікатор, назва, опис.
4. Task – представляє завдання у проєкті та містить інформацію про завдання, таку як ідентифікатор, назва, опис, дата початку, дата завершення, статус, пріоритет, виконавець та коментарі список коментарів до завдання.
5. Comment – представляє коментар до завдання та містить інформацію про коментар, таку як ідентифікатор, текст, автор та дата.

Ці класи взаємодіють між собою, утворюючи зв'язки:

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						27
Змн.	Арк.	№ докум.	Підпис	Дата		

- Клас User має зв'язок "багато до багатьох" з класом Project через проміжну таблицю, що містить атрибути класу Role.
- Клас Project має зв'язок "один до багатьох" з класом Task.
- Клас Task має зв'язок "один до багатьох" з класом Comment.

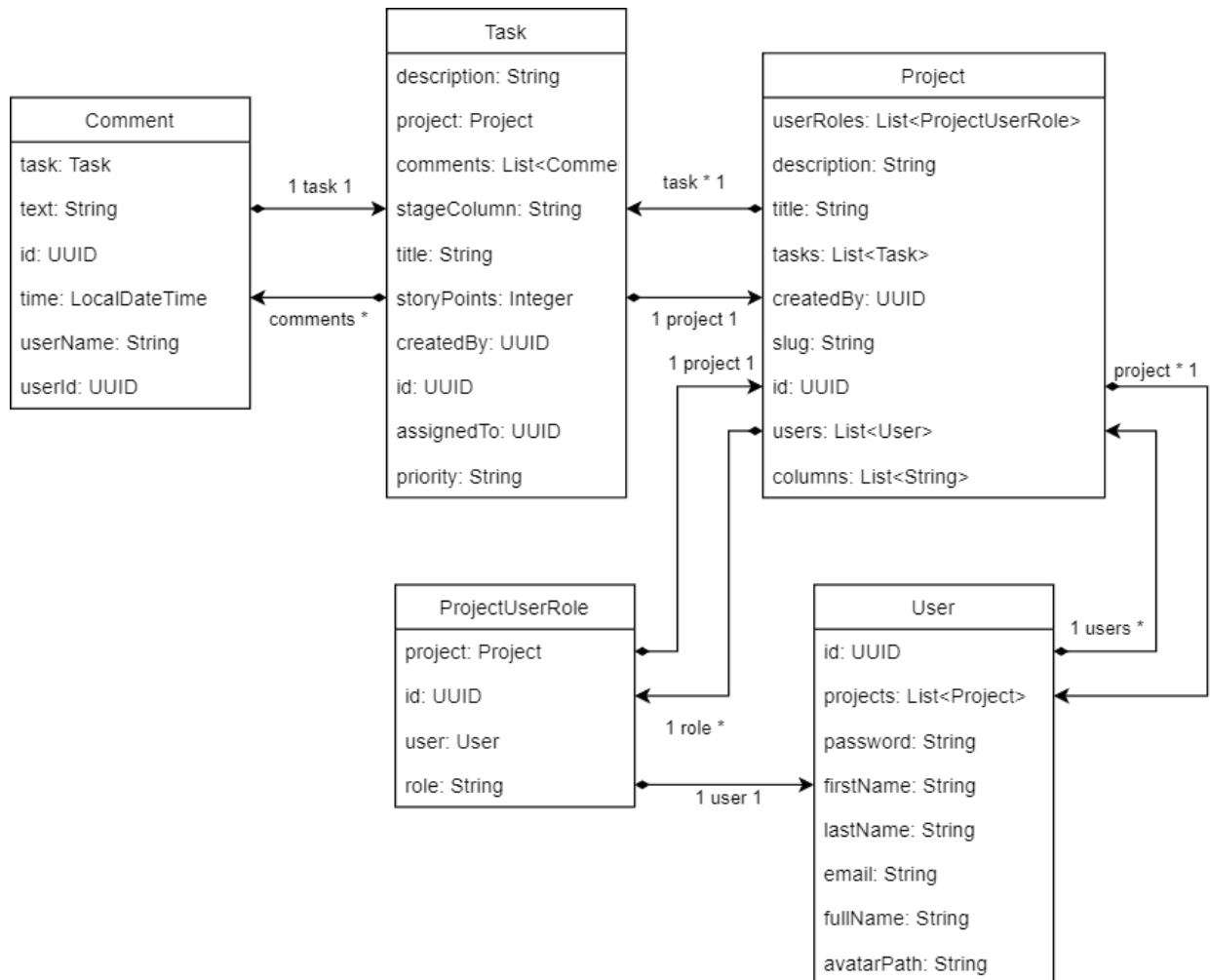


Рисунок 2.3 – Діаграма класів

2.2. Розробка бази даних системи

Розглянемо основні сутності:

- Користувач;
- Проєкт;
- Завдання;
- Коментар.

Першою ключовою сутністю є «Користувачі», оскільки вони представляють основних учасників системи. Кожен користувач має

унікальний ідентифікатор (UUID), електронну пошту та пароль для автентифікації, а також ім'я, прізвище та шлях до збереженого аватару (табл. 2.1).

Таблиця 2.1

Структура таблиці “users”

Назва	Тип даних	ПК	ЗК	Опис поля
id	BINARY	+	-	Унікальний ідентифікатор користувача
email	VARCHAR	-	-	Електронна пошта користувача
password	VARCHAR	-	-	Пароль користувача
first_name	VARCHAR	-	-	Ім'я користувача
last_name	VARCHAR	-	-	Прізвище користувача
avatar_path	VARCHAR	-	-	Шлях до збереженого аватару користувача

Сутність "Проект" представляє конкретний ІТ-проект. Кожен проект має унікальний ідентифікатор, назву, опис, текстовий ідентифікатор (slug) та інформацію про користувача, що створив проект (зовнішній ключ created_by) (табл. 2.2).

Таблиця 2.2

Структура таблиці “projects”

Назва	Тип даних	ПК	ЗК	Опис поля
id	BINARY	+	-	Унікальний ідентифікатор проекту
title	VARCHAR	-	-	Назва проекту
description	VARCHAR	-	-	Опис проекту
slug	VARCHAR	-	-	Текстовий ідентифікатор проекту
created_by	BINARY	-	+	Зовнішній ключ для зв'язку з користувачем

Оскільки проект може мати декілька користувачів, а кожен користувач може мати декілька проектів, між цими сутностями встановлено зв'язок багато-до-багатьох, що реалізовано за допомогою таблиці "project_user", яка зберігає унікальні ідентифікатори проектів та користувачів (табл. 2.3).

Таблиця 2.3

Структура таблиці “project_user”

Назва	Тип даних	ПК	ЗК	Опис поля
project_id	BINARY	-	+	Унікальний ідентифікатор проекту
user_id	VARCHAR	-	+	Назва стовпця

Дошка проекту може мати декілька стовпців, у яких розташовуватимуться завдання. Для збереження цих даних використовується

таблиця "project_columns", що зберігає назви стовпців та ідентифікатори проєктів, до яких вони належать (табл. 2.4).

Таблиця 2.4

Структура таблиці "project_columns"

Назва	Тип даних	ПК	ЗК	Опис поля
project_id	BINARY	+	-	Унікальний ідентифікатор проєкту
column	VARCHAR	-	-	Назва стовпця

Кожен користувач має конкретну роль у кожному з проєктів, у яких бере участь. Ці дані зберігаються у таблиці "project_user_roles", яка містить унікальні ідентифікатори ролей, назви ролей, ідентифікатори проєктів та користувачів (табл. 2.5).

Таблиця 2.5

Структура таблиці "project_user_roles"

Назва	Тип даних	ПК	ЗК	Опис поля
id	BINARY	+	-	Унікальний ідентифікатор ролі
role	VARCHAR	-	-	Назва ролі
project_id	BINARY	+	-	Унікальний ідентифікатор проєкту
user_id	VARCHAR	-	-	Унікальний ідентифікатор користувача

Сутність "Завдання" описує конкретне завдання, що містить унікальний ідентифікатор, назву, опис, етап (стовпець дошки проєкту, наприклад, "Не почато", "В розробці", "На перевірці", "Завершено"), пріоритетність, оцінку складності, відповідального користувача, користувача, що створив завдання, та ідентифікатор проєкту, до якого належить завдання (табл. 2.6).

Таблиця 2.6

Структура таблиці "tasks"

Назва	Тип даних	ПК	ЗК	Опис поля
id	BINARY	+	-	Унікальний ідентифікатор завдання
title	VARCHAR	-	-	Назва проєкту
description	VARCHAR	-	-	Опис проєкту
stage_column	VARCHAR	-	-	Етап, на якому знаходиться завдання
priority	VARCHAR	-	-	Пріоритетність виконання завдання
story_points	INT	-	-	Оцінка складності завдання
assigned_to	BINARY	-	-	Зовнішній ключ для зв'язку з відповідальним користувачем
created_by	BINARY	-	-	Ідентифікатор користувача, що створив завдання
project_id	BINARY	-	+	Зовнішній ключ для зв'язку з проєктом

Сутність "Коментар" описує текстовий запис, який додається до завдання для обговорення, пояснення чи комунікації. Він містить унікальний ідентифікатор, текст коментаря, дату та час створення, інформацію про користувача, що створив коментар, ім'я користувача та ідентифікатор завдання, до якого належить коментар (табл. 2.7).

Таблиця 2.7

Структура таблиці "commnets"

Назва	Тип даних	ПК	ЗК	Опис поля
id	BINARY	+	-	Унікальний ідентифікатор коментарю
text	VARCHAR	-	-	Текст коментаря
time	DATETIME	-	-	Дата та час створення коментарю
user_id	BINARY	-	+	Зовнішній ключ для зв'язку з користувачем, що створив коментар
user_name	VARCHAR	-	-	Ім'я користувача
task_id	BINARY	-	-	Ідентифікатор завдання, якому належить коментар

Загалом, структура бази даних вебдодатку управління ІТ-проектами включає основні сутності користувачів, проектів, завдань та коментарів, що забезпечує ефективне зберігання та обробку даних, необхідних для функціонування системи. ER-діаграму бази даних наведено нижче (рис. 2.4).

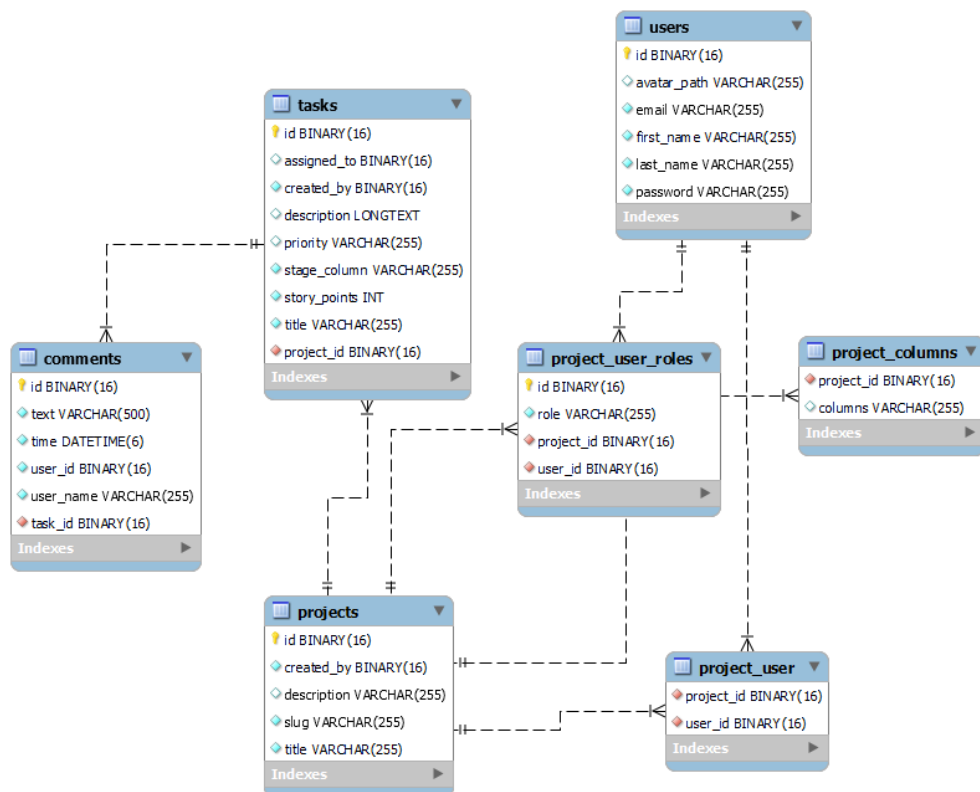


Рисунок 2.4 – ER-діаграма бази даних

2.3. Проектування та реалізація алгоритмів роботи системи

Основними процесами системи є робота з проектами та завданнями. Вони включають такі дії, як створення, редагування та видалення. На діаграмі наведено приклад послідовності дій під час створення нового проекту (рис. 2.5).

Користувач входить до системи, вводючи свої облікові дані (електронну пошту та пароль). Після авторизації користувач вибирає опцію створення нового проекту. Користувач заповнює форму, вводючи необхідну інформацію про проект, та підтверджує введення. Система показує користувачеві форму, в якій він може ввести необхідні дані для створення нового проекту. введених даних система створює новий проект, зберігаючи всі введені дані. Система автоматично надає користувачеві роль керівника (адміністратора) нового проекту, що дозволяє йому керувати проектом та його учасниками. Після успішного створення проекту система показує користувачеві головну сторінку проекту.

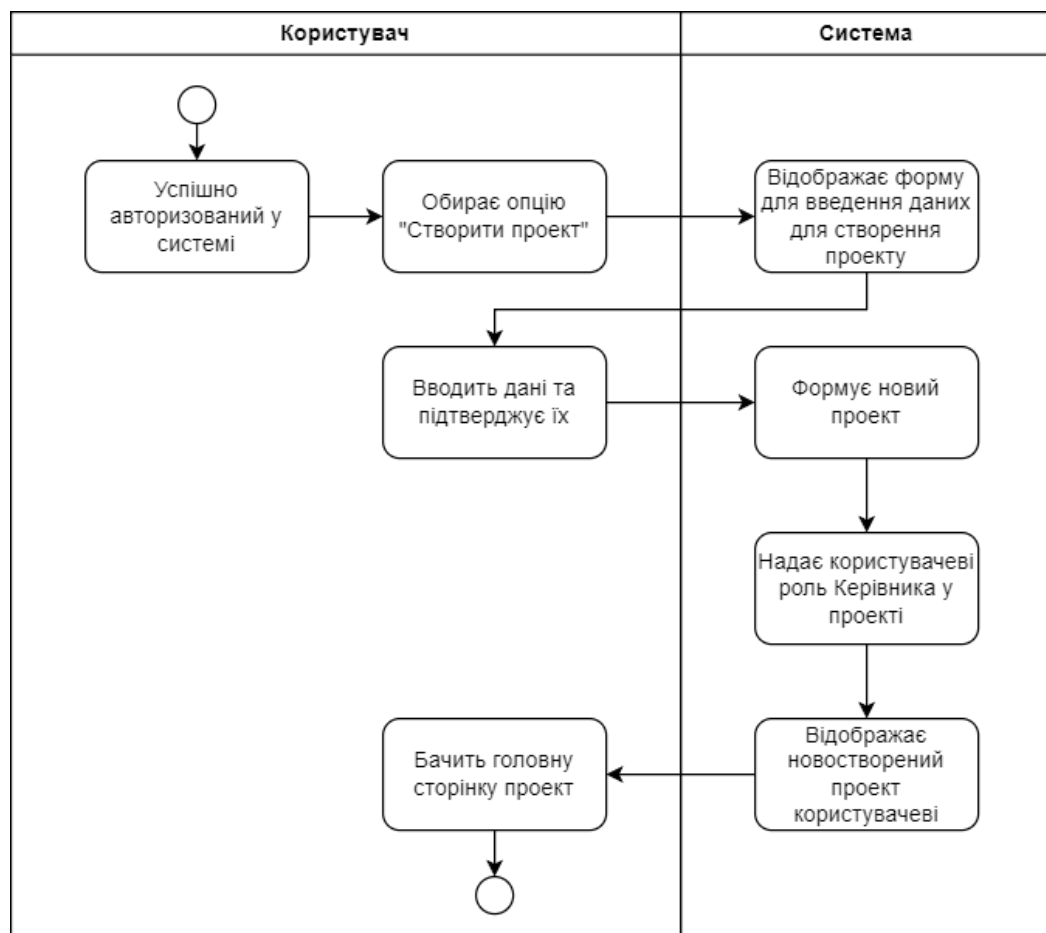


Рисунок 2.5 – Діаграма діяльності для створення проекту

Створення завдання, редагування завдання чи проєкту відбуваються за ідентичним механізмом до створення проєкту – обирається відповідна опція, відображається форма створення або редагування, і завершується дія підтвердженням.

Наступним важливим процесом є взаємодія користувачів з завданнями. Користувач може призначати створені завдання іншим учасникам проєкту. Розглянемо дану дію від особи адміністратора проєкту для кращого прикладу. Дана процедура наведена нижче у діаграмі діяльності призначення завдання (рис. 2.6).

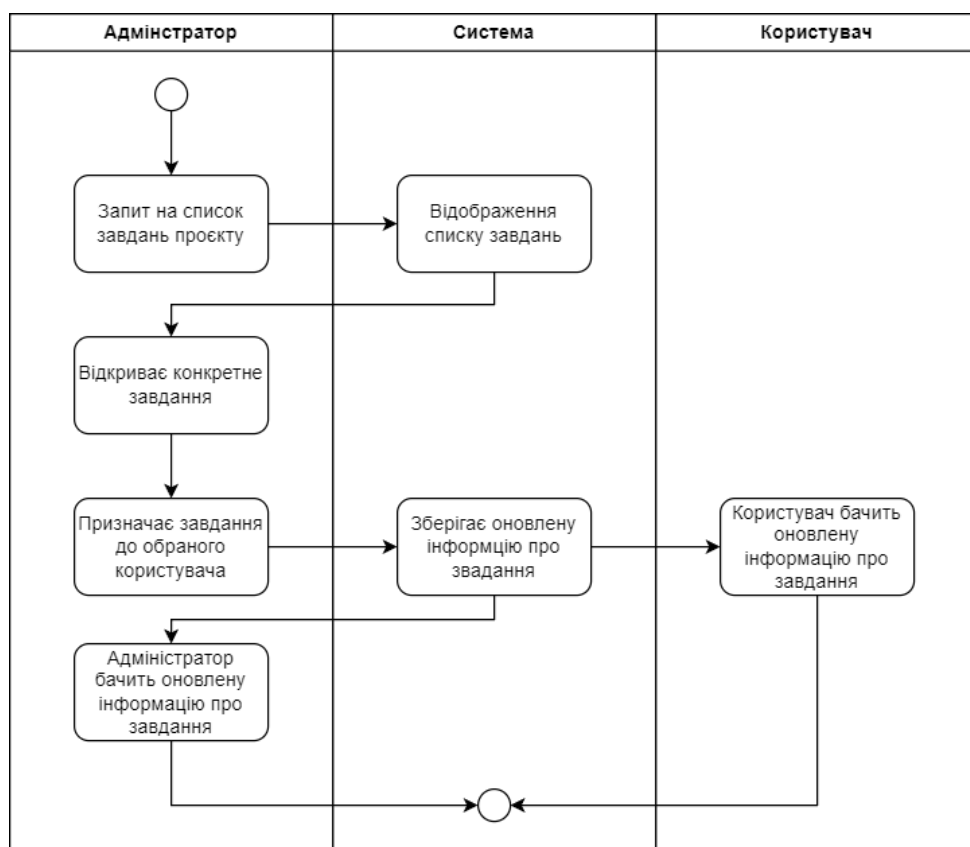


Рисунок 2.6 – Діаграма діяльності для призначення завдань

Адміністратор надсилає запит до системи на отримання списку завдань проєкту. Система відображає йому список завдань проєкту. Адміністратор відкриває будь-яке завдання зі списку, натиснувши на нього. Адміністратор призначає завдання обраному користувачеві. Система зберігає оновлену інформацію про завдання, включаючи призначеного користувача. Адміністратор, користувач, якому було призначено завдання, та решта користувачів проєкту бачить оновлену інформацію про нього.

2.4. Реалізація додатку веборієнтованої системи управління ІТ-проєктами

Програмний комплекс складається з двох компонентів – клієнтського додаток та серверного. Спершу робота була розпочала над створенням базового інтерфейсу системи. Для організації React проєкту було використана компонентна архітектура. Кожен компонент відповідає за певну функціональність системи. Структура проєкту розбита на три модулі:

- Auth – цей модуль відповідає за авторизацію та навігацію по сторінках проєкту.
- Components – цей модуль містить перевикористовувані компоненти, які використовуються в різних частинах додатку. Це може включати компоненти для відображення даних, форм введення та інших елементів інтерфейсу.
- Pages – цей модуль містить компоненти, які відповідають за окремі сторінки додатку.

Ця структура дозволяє розділити відповідальність між компонентами, зробити код більш модульним та легким для підтримки та розширення.

У процесі розробки клієнтської частини вебдодатку управління ІТ-проєктами, було виконано кілька базових етапів, які включають підключення бібліотеки компонентів та налаштування системи навігації.

```
import {CssVarsProvider} from "@mui/joy";
import AuthProvider from "../auth/AuthProvider";
import Routes from "../auth/Routes";

export default function App() {
  return (
    <CssVarsProvider defaultMode="system" disableTransitionOnChange>
      <AuthProvider>
        <Routes />
      </AuthProvider>
    </CssVarsProvider>
  );
}
```

Перш за все, було підключено бібліотеку Joy UI до проєкту. Це бібліотека компонентів, яка надає готові елементи інтерфейсу та стилі для створення привабливого та інтуїтивно зрозумілого користувацького

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						34
Змн.	Арк.	№ докум.	Підпис	Дата		

інтерфейсу. Для цього було використано компонент `CssVarsProvider` з бібліотеки `Joey UI`, який дозволяє налаштувати тему та стилі додатку.

Далі, було налаштовано систему навігації додатку. Для цього було використано компоненти `AuthProvider` та `Routes`. `AuthProvider` відповідає за аутентифікацію користувачів, а `Routes` визначає маршрути додатку та відповідає за навігацію між ними.

```
const AuthContext = createContext();

export default function AuthProvider({ children }) {

  ...

  useEffect(() => {
    if (token) {
      const decodedToken = jwtDecode(token);
      const currentTime = Date.now() / 1000;

      if (decodedToken.exp < currentTime) {
        setToken(null);
      } else {
        axios.defaults.headers.common["Authorization"] = "Bearer " +
token;
        localStorage.setItem('token', token);
      }
    } else {
      delete axios.defaults.headers.common["Authorization"];
      localStorage.removeItem('token');
      localStorage.removeItem('user');
      localStorage.removeItem('avatar');
    }
  }, [token]);

  const contextValue = useMemo(
    () => ({
      token,
      setToken,
      user,
      setUser,
      avatar
    }),
    [token]
  );

  ...

  export const useAuth = () => {
    return useContext(AuthContext);
  };
}
```

У наведеному вище частині коду компоненту `AuthProvider` реалізовано логіку аутентифікації та управління токеном доступу користувача. Компонент `AuthProvider` створює контекст аутентифікації [17], який надає дочірнім компонентам доступ до токена доступу, даних користувача та

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						35
Змн.	Арк.	№ докум.	Підпис	Дата		

аватару користувача. Коли токен доступу змінюється, виконується `useEffect`, який перевіряє, чи токен не прострочений, та встановлює заголовок `Authorization` для `axios`, щоб автоматично додавати токен до запитів. Далі використовується `useMemo` для створення об'єкта `contextValue`, який містить токен доступу, функції для його встановлення, дані користувача та аватар. Цей об'єкт передається дочірнім компонентам через `AuthContext.Provider`. Нарешті, експортується хук `useAuth`, який дозволяє дочірнім компонентам отримувати доступ до контексту аутентифікації.

```
const routesForAuthenticatedOnly = [
  {
    path: "/",
    element: <ProtectedRoute />,
    children: [
      {
        path: "/",
        element: <ProjectsPage />,
      },
      {
        path: "/projects",
        element: <ProjectsPage />,
      },
      {
        path: "/projects/:slug",
        element: <ProjectPage />,
      },
      {
        path: "/profile",
        element: <ProfilePage />,
      },
      {
        path: "/auth",
        element: <ProjectsPage />,
      },
      {
        path: "/logout",
        element: <Logout />,
      },
      {
        path: "*",
        element: <NotFoundPage />,
      },
    ],
  },
];

const routesForNotAuthenticatedOnly = [
  {
    path: "/auth",
    element: <AuthPage />
  },
  {
    path: "*",
    element: <NotFoundPage />,
  }
];
```

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						36
Змн.	Арк.	№ докум.	Підпис	Дата		

У наведеному коді класу Routes реалізовано логіку маршрутизації додатку залежно від статусу аутентифікації користувача. Компонент визначає два типи маршрутів: для аутентифікованих користувачів та для неаутентифікованих користувачів. Маршрути для аутентифікованих користувачів захищаються компонентом ProtectedRoute, який перевіряє наявність токена доступу. Компонент також включає маршрут для сторінки 404, яка відображається, коли користувач намагається доступитися неіснуючої сторінки. Залежно від статусу аутентифікації користувача, компонент створює маршрути та передає їх до компонента RouterProvider для навігації між ними.

Оскільки робота та моніторинг проєктів вимагають роботу з дошкою, було розроблено алгоритм взаємодії з картками, який дозволяє користувачам переміщувати картки з одного стовпчику до іншого. Це було здійснено за допомогою використання бібліотеки React DnD [18].

Розглянемо компоненти Board, Column та TaskCard використовуються для створення інтерфейсу дошки, колонок та картки завдання відповідно. Компонент Board відповідає за отримання даних про завдання та колонки з сервера, а також збереження змін на сервері. Він також містить логіку для фільтрації завдань за користувачами та пріоритетами. Нижче наведено фрагмент коду, що відповідає за переміщення картки з завданням між колонками дошки.

```
const moveCardHandler = (dragIndex, hoverIndex) => {
  const dragItem = items[dragIndex];

  if (dragItem) {
    setItems((prevState) => {
      const copiedStateArray = [...prevState];
      const prevItem = copiedStateArray.splice(hoverIndex, 1,
dragItem);
      copiedStateArray.splice(dragIndex, 1, prevItem[0]);

      return copiedStateArray;
    });
  }
};
```

Компонент TaskCard представляє картку завдання. Він використовує хук useDrag з бібліотеки react-dnd для створення перетягуваного елемента, а також хук useDrop для створення зони, в яку можна переміщати картку

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						37
Змн.	Арк.	№ докум.	Підпис	Дата		

завдання. Компонент також містить логіку для зміни колонки завдання та оновлення даних на сервері. Даний фрагмент коду, що наведений нижче, використовує хук `useDrop` з бібліотеки `react-dnd` для створення зони, в яку можна переміщати картки завдання, і визначення, чи можна перемістити картку завдання в цю зону.

```
const [{ isOver, canDrop }, drop] = useDrop({
  accept: "Our first type",
  drop: () => ({ name: title }),
  collect: (monitor) => ({
    isOver: monitor.isOver(),
    canDrop: monitor.canDrop()
  })
});
```

Компонент `Column` використовує хук `useDrop` з бібліотеки `react-dnd` для створення зони, в яку можна переміщати картки завдання. Компонент також містить логіку для визначення кольору фону зони в залежності від того, чи можна перемістити картку завдання в цю зону. Фрагмент коду, що наведений нижче також використовується для створення зони, в яку можна переміщати картки завдання, але він також містить логіку для переміщення картки завдання між колонками. Коли користувач переміщає картку завдання, функція `hover` визначає, чи можна перемістити картку завдання в цю зону, і якщо так, то викликає функцію `moveCardHandler` для оновлення стану компонента `Board` з новим порядком картки завдання.

```
const [, drop] = useDrop({
  accept: "Our first type",
  hover(item, monitor) {
    if (!ref.current) {
      return;
    }
    const dragIndex = item.index;
    const hoverIndex = index;

    if (dragIndex === hoverIndex) {
      return;
    }

    const hoverBoundingRect = ref.current?.getBoundingClientRect();
    const hoverMiddleY =
      (hoverBoundingRect.bottom - hoverBoundingRect.top) / 2;

    const clientOffset = monitor.getClientOffset();
    const hoverClientY = clientOffset.y - hoverBoundingRect.top;

    if (dragIndex < hoverIndex && hoverClientY < hoverMiddleY) {
      return;
    }
  }
});
```

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						38
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        if (dragIndex > hoverIndex && hoverClientY > hoverMiddleY) {
            return;
        }

        moveCardHandler(dragIndex, hoverIndex);
        item.index = hoverIndex;
    }
});

```

Під час переміщення картки завдання між колонками відбувається наступний алгоритм:

1. Компонент TaskCard визначає, що картка завдання переміщається, і створює об'єкт item, який містить інформацію про картку завдання, включаючи її індекс, назву, поточну колонку та об'єкт завдання.
2. Компонент Column визначає, що картка завдання переміщається над ним, і викликає функцію hover, яка визначає, чи можна перемістити картку завдання в цю колонку.
3. Якщо картка завдання може бути переміщена в цю колонку, компонент Column викликає функцію moveCardHandler з компонента Board, яка оновлює стан компонента Board з новим порядком картки завдання.
4. Компонент TaskCard визначає, що переміщення картки завдання завершено, і викликає функцію end, яка визначає, чи картка завдання була переміщена в нову колонку, і викликає функцію changeItemColumn, яка оновлює колонку завдання та оновлює дані на сервері.

Цей алгоритм дозволяє користувачам легко переміщати картки завдання між колонками, що допомагає їм відстежувати прогрес роботи над завданнями та співпрацювати з іншими членами команди.

Розглянемо сторінку проекту. Повний лістинг коду можна знайти у додатку. Ця сторінка реалізує відображення проекту та його налаштувань. При завантаженні сторінки, вона виконує запит до сервера за допомогою функції fetchProject(), щоб отримати дані проекту за його унікальним ідентифікатором, отриманим з URL-адреси.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						39
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const fetchProject = async () => {
    try {
        const response = await axios.get(process.env.REACT_APP_API_URL +
GET_URL, {
            headers: { Authorization: `Bearer ${token}` },
            params: { slug: slug }
        });
        setProject(response.data);
    } catch (error) {
        navigate("/not_found", { replace: true });
    }
};

const isUserAdmin = async () => {
    try {
        console.log(project.id);
        const response = await axios.get(process.env.REACT_APP_API_URL +
GET_ROLE_URL, {
            headers: { Authorization: `Bearer ${token}` },
            params: {
                projectId: project.id
            }
        });

        console.log(response.data);
        setUserRole(response.data);
        console.info("Project role fetched successfully!", response.data);
    } catch (error) {
        console.error("Failed to fetch project's role data:", error);
    }
};

```

Якщо проєкт існує, далі виконується запит, щоб перевірити роль користувача у цьому проєкті за допомогою функції isUserAdmin().

```

const isUserAdmin = async () => {
    try {
        console.log(project.id);

        const response = await axios.get(process.env.REACT_APP_API_URL +
GET_ROLE_URL, {
            headers: { Authorization: `Bearer ${token}` },
            params: {
                projectId: project.id
            }
        });

        console.log(response.data);
        setUserRole(response.data);
        console.info("Project role fetched successfully!", response.data);
    } catch (error) {
        console.error("Failed to fetch project's role data:", error);
    }
};

```

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

Після завершення запиту дані про проєкт – ім'я та опис відображаються на сторінці, а також вкладки для перемикання між дошкою та налаштуваннями проєкту.

За вище описаним принципом роботи сторінки проєкту, працюють і решта сторінок – коли користувач переходить на відповідно сторінку спрацьовують методи, що звертаються до серверу з конкретним запитами, а той в свою чергу повертає відповідні дані для виведення користувачеві.

Розглянемо тепер роботу серверної частини вебдодатку. У розробці Spring проєкту також використовується модульна архітектура для організації коду. Проєкт розбивається на логічні модулі, кожен з яких відповідає за певну функціональність системи. Структура проєкту розділена на три модулі:

- Auth - цей модуль відповідає за аутентифікацію та авторизацію користувачів. Він включає контролери, сервіси та репозиторії для обробки запитів на вхід, реєстрацію, скидання пароля та захист маршрутів.
- Project - цей модуль відповідає за всю функціональність, пов'язану з проєктами. Він включає контролери, сервіси та репозиторії для обробки запитів на створення, перегляд, редагування та видалення проєктів, а також управління завданнями, коментарями та користувачами в рамках проєкту.
- Profile - цей модуль відповідає за всю функціональність, пов'язану з профілями користувачів. Він включає контролери, сервіси та репозиторії для обробки запитів на перегляд, редагування та зміну пароля профілю користувача.

Ця структура дозволяє розділити відповідальність між модулями, зробити код більш модульним та легким для підтримки та розширення. Кожен модуль має свою власну логіку та взаємодіє з іншими модулями через інтерфейси та залежності, що дозволяє легко додавати нову функціональність до системи, розширювати її та підтримувати код.

У структурі код розділений на логічні шари:

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						41
Змн.	Арк.	№ докум.	Підпис	Дата		

- Моделі (Models): представляють дані та бізнес-логіку додатку.
- Репозиторії (Repositories): відповідають за взаємодію з базою даних.
- Сервіси (Services): містять бізнес-логіку додатку та використовують репозиторії для отримання даних.
- Контролери (Controllers): обробляють HTTP-запити від клієнта та повертають відповідь, використовуючи сервіси для отримання даних та виконуючи необхідні перевірки та валідацію.

Першим функціоналом над яким почалася робота це налагодження комунікації між сервером та клієнтом, а також розробка авторизації та аутентифікації з використанням JWT токенів [21].

Для налагодження комунікації між сервером та клієнтом було налаштовано CORS (Cross-Origin Resource Sharing) [22] за допомогою класу WebConfig. Цей клас містить конфігурацію для дозволу запитів з клієнтського додатку на сервер.

```
@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Value("${client-app.url}") String clientAppUrl;

    @Override
    public void addCorsMappings(CorsRegistry registry) {

        registry.addMapping("/**")
            .allowedOrigins(clientAppUrl)
            .allowedMethods("GET", "POST", "PUT", "DELETE")
            .allowedHeaders("*")
            .allowCredentials(true);
    }
}
```

У методі addCorsMappings() визначається, що всі запити до сервера з будь-якого джерела (allowedOrigins(clientAppUrl)) дозволені для методів GET, POST, PUT та DELETE (allowedMethods()) [23]. Також дозволено використовувати будь-які заголовки (allowedHeaders("*")) та передавати куки (allowCredentials(true)). Це дозволяє клієнтському додатку взаємодіяти з серверною частиною додатку без обмежень з боку браузера.

У проєкті для аутентифікації та авторизації користувачів використовуються JWT (JSON Web Tokens). JWT - це стандарт для створення

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						42
Змн.	Арк.	№ докум.	Підпис	Дата		

токенів доступу, який дозволяє передавати інформацію між клієнтом та сервером в безпечному та зручному форматі.

У проєкті JWT-токені реалізовані за допомогою класу `JwtServiceImpl`, який реалізує інтерфейс `JwtService` [24]. Код наведений у додатках. Клас містить методи для генерації токенів, вилучення інформації з токенів та перевірки їхньої валідності.

Коли користувач успішно проходить аутентифікацію, сервер генерує JWT-токен, який містить інформацію про користувача, таку як ім'я користувача та термін дії токена. Токен потім передається клієнту, який зберігає його у локальному сховищі та включає у заголовок кожного запиту до сервера.

При отриманні запиту від клієнта сервер перевіряє валідність токена, вилучаючи інформацію з нього та перевіряючи термін дії токена. Якщо токен валідний, сервер обробляє запит, інакше повертає помилку аутентифікації.

Ця реалізація JWT-токенів дозволяє нам безпечно та зручно аутентифікувати користувачів та надавати їм доступ до захищених ресурсів на сервері.

Розглянемо реалізацію пошуку користувачів при додаванні до проєкту.

```
@GetMapping("/user_search")
public ResponseEntity<List<UserDto>>
userSearch(@RequestHeader("Authorization") String token,
           @RequestParam String name,
           @RequestParam UUID projectId)
{
    String userEmail = jwtService.extractUserNameByBearerToken(token);

    if (userEmail == null) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
    }

    List<UserDto> filteredUsers = projectService.searchUsers(name, userEmail,
projectId);

    return ResponseEntity.status(HttpStatus.OK).body(filteredUsers);
}
```

Контролер отримує GET-запит на `"/user_search"` з заголовком `"Authorization"`, який містить JWT-токен, та параметрами `"name"` та `"projectId"`. Контролер викликає метод `extractUserNameByBearerToken` з `jwtService`, передаючи йому JWT-

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						43
Змн.	Арк.	№ докум.	Підпис	Дата		

токен, щоб витягнути електронну адресу користувача, який надіслав запит. Якщо електронна адреса користувача не знайдена (тобто, токен недійсний), контролер повертає відповідь з кодом статусу 401 (UNAUTHORIZED). Якщо електронна адреса користувача знайдена, контролер викликає метод searchUsers з projectService, передаючи йому параметри "name", "userEmail" та "projectId", щоб отримати список користувачів, які відповідають заданим критеріям. Контролер повертає відповідь з кодом статусу 200 (OK) та списком користувачів у тілі відповіді.

Перейдемо до методу searchUsers, що здійснює сам пошук користувачів за іменем. Метод викликає findUsersByProjectId з projectRepository, передаючи йому projectId, щоб отримати список користувачів, які вже є учасниками проєкту з заданим projectId. Метод створює список projectUserEmails з електронних адрес користувачів, які вже є учасниками проєкту. Метод створює об'єкт Pageable з номером сторінки 0 та розміром сторінки 15. пошук користувачів в userRepository, імена або прізвища яких містять задане name, а електронна адреса не співпадає з userEmail, викликаючи відповідний метод userRepository та передаючи йому name, userEmail та pageable, щоб отримати список користувачів, імена або прізвища яких містять задане name, а електронна адреса яких не дорівнює userEmail. Метод створює список filteredUsers, який містить тільки тих користувачів з списку users, електронні адреси яких не містяться у списку projectUserEmails. Метод повертає список UserDto, який містить тільки перші 10 елементів з списку filteredUsers, отриманих після перетворення кожного об'єкту User на UserDto за допомогою userMapper. Якщо виникає будь-яка помилка, метод записує повідомлення про помилку у лог та повертає порожній список.

```
public List<UserDto> searchUsers(String name, String userEmail, UUID
projectId) {

    try {
        List<User> projectUsers =
projectRepository.findUsersByProjectId(projectId);

        List<String> projectUserEmails = projectUsers.stream()
            .map(User::getEmail)
```

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						44
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        .collect(Collectors.toList());

Pageable pageable = PageRequest.of(0, 15);

List<User> users =
userRepository.findByFirstNameContainingOrLastNameContainingAndEmailNot(
    name, name, userEmail, pageable);

List<User> filteredUsers = users.stream()
    .filter(user -> !projectUserEmails.contains(user.getEmail()))
    .toList();

return filteredUsers.stream()
    .map(userMapper::userToUserDto)
    .limit(10)
    .collect(Collectors.toList());

} catch (Exception e) {

    log.error("Project | Failed to find users. Reason: [{}].",
e.getMessage());

    return Collections.emptyList();
}
}

```

Висновки до другого розділу

У другому розділі було проведено аналіз вимог та визначено об'єктно-орієнтовану структуру системи. Було визначено функціональні та нефункціональні вимоги, ролі користувачів, варіанти взаємодії з функціоналом в залежності від ролі користувача та визначено основні прецеденти. На основі цих даних було сформовано діаграму класів та розроблено таблиці бази даних.

Під час проєктування та реалізації алгоритмів роботи системи було описано головні процеси системи. На етапі реалізації додатку веборієнтованої системи було описано ключові етапи написання програми з наведенням прикладів реалізації деякого функціоналу системи.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						45
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБДОДАТКОМ ДЛЯ УПРАВЛІННЯ ІТ-ПРОЄКТАМИ

3.1. Порядок встановлення та налаштування параметрів системи

Повноцінний запуск роботи системи вимагає розгортання всіх її компонентів та правильного їх налаштування. Дана система складається з трьох основних частин (рис. 3.1): клієнтської частини на основі React, серверної частини на основі Java та фреймворку Spring, та бази даних MySQL.

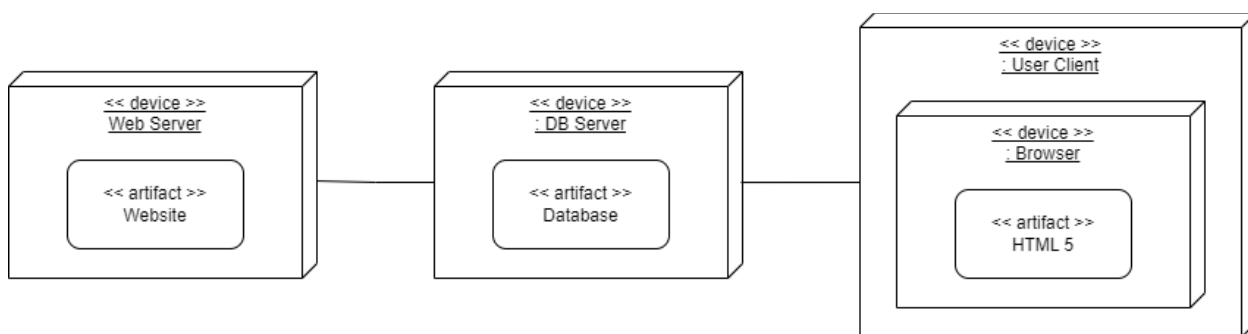


Рисунок 3.1 – Діаграма розгортання

Перед початком роботи необхідно налаштувати робоче середовище, на якому буде здійснено запуск компонентів системи. Робоча машина повинна мати встановлені наступні версії компонентів та засобів:

- Node.js (версія 14.17.0 або вище) [25];
- JDK 17 [26];
- MySQL (версія 8.0 або вище).

Для налаштування параметрів запуску серверного додатку, підключення до бази даних та обробки запитів з клієнтського додатку, необхідно налаштувати конфігураційний файл `application.yml`, де встановити наступні параметри:

- `server.port` – порт, на якому запускається серверний додаток.
- `spring.datasource.url` – ULR-адреса для підключення до бази даних MySQL.

- spring.datasource.username – ім'я користувача бази даних.
- spring.datasource.password – пароль для підключення до бази даних.
- client-app.url – ULR-адреса клієнтського додатку.
- token.signing.key – ключ для підпису JWT токенів.

Для налаштування клієнтського додатку необхідно налаштувати конфігураційний файл .env, а саме:

- PORT – порт, на якому запускається клієнтський додаток.
- HOST – хост, на якому запускається клієнтський додаток
- REACT_APP_API_URL ULR-адреса серверного додатку.

3.2. Структура інтерфейсу вебдодатку та порядок роботи

Щоб розпочати роботу з вебдодатком, користувач повинен авторизуватися в системі. Він може увійти за допомогою форми входу (див. рис. 3.2) або зареєструватися (також див. рис. 3.2), якщо у нього ще немає облікового запису.

Якщо користувач вже зареєстрований у системі, для входу йому потрібно ввести свій логін та пароль у відповідні поля форми входу та натиснути кнопку «Вхід».

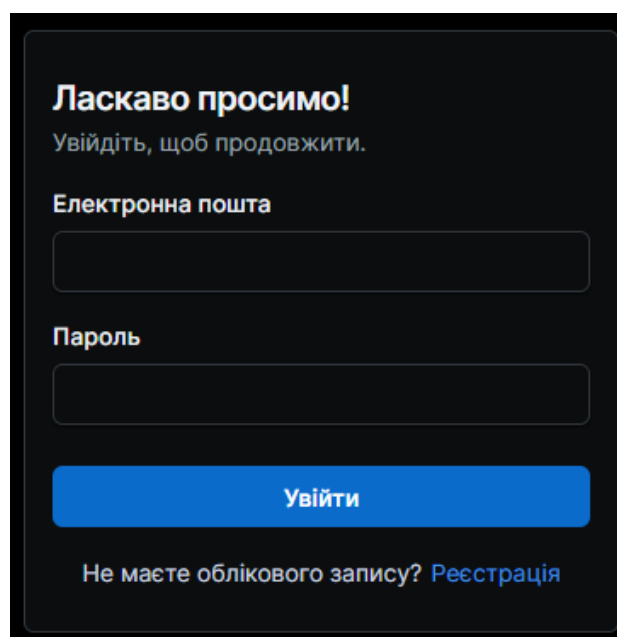


Рисунок 3.2 – Форма авторизації

Для реєстрації користувача необхідно заповнити форму реєстрації. Користувач повинен внести свої дані, такі як прізвище, ім'я, електронна пошта та пароль. Після перевірки введених даних та відправлення форми, користувач буде перенаправлений на головну сторінку (рис. 3.4).

Ласкаво просимо!
Зареєструйтеся, щоб продовжити.

Ім'я

Прізвище

Електронна пошта

Пароль

Повторний пароль

Зареєструватись

Вже маєте акаунт? [Вхід](#)

Рисунок 3.3 – Форма реєстрації

На верхній частині головної сторінки розташована навігаційна панель, яка дозволяє швидко переходити між різними розділами додатку. Нижче розташований блок з проєктами, який може виглядати по-різному, залежно від того, чи є у користувача наявні проєкти чи ні. Якщо він вже приєднався до якихось проєктів, їхні карточки з'являться з відповідною назвою та описом. У випадку, якщо користувач новий у системі, йому буде запропоновано створити свій перший проєкт, що надихне на активність та втягнення у використання сервісу.

Під заголовком блоку проєктів розташована кнопка створення нового проєкту, яка спрямовує користувача на подальші кроки в створенні.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						48
Змн.	Арк.	№ докум.	Підпис	Дата		

Натискання на неї відкриває модальне вікно зі спеціальною формою, де користувач заповнює необхідні дані про новий проєкт, включаючи його назву та, за бажанням, короткий опис, який допоможе легше розібратися в сутності проєкту (рис. 3.5). Цей процес має бути інтуїтивно зрозумілим та простим, сприяючи швидкому старту.

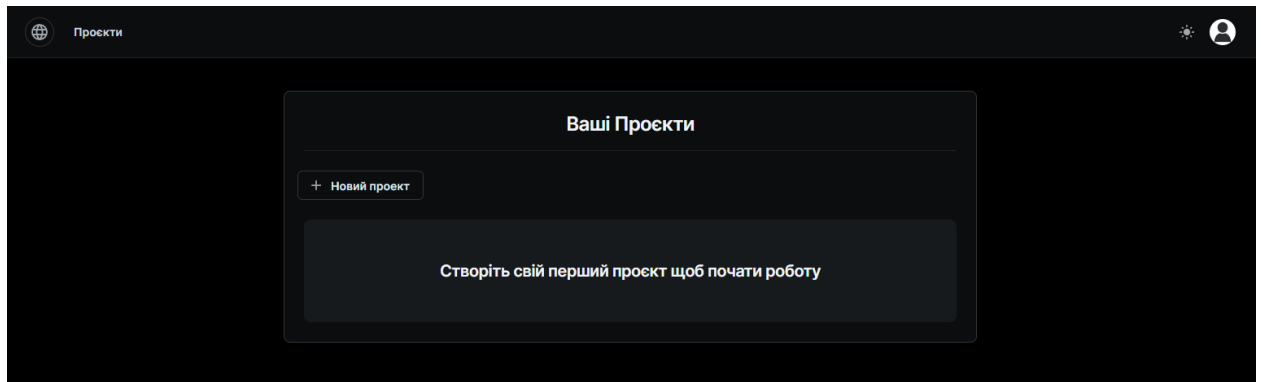


Рисунок 3.4 – Сторінка з проєктами

Рисунок 3.5 – Форма створення нового проєкту

Після успішного заповнення та підтвердження форми, дані про новий проєкт надсилаються на сервер для подальшої обробки. Якщо операція виявиться успішною, новий проєкт з'явиться на головній сторінці вебдодатку, готовий до подальшого редагування та використання. Користувач може одразу перейти до роботи над новим проєктом, натиснувши на відповідну картку або на кнопку переходу (рис 3.6).

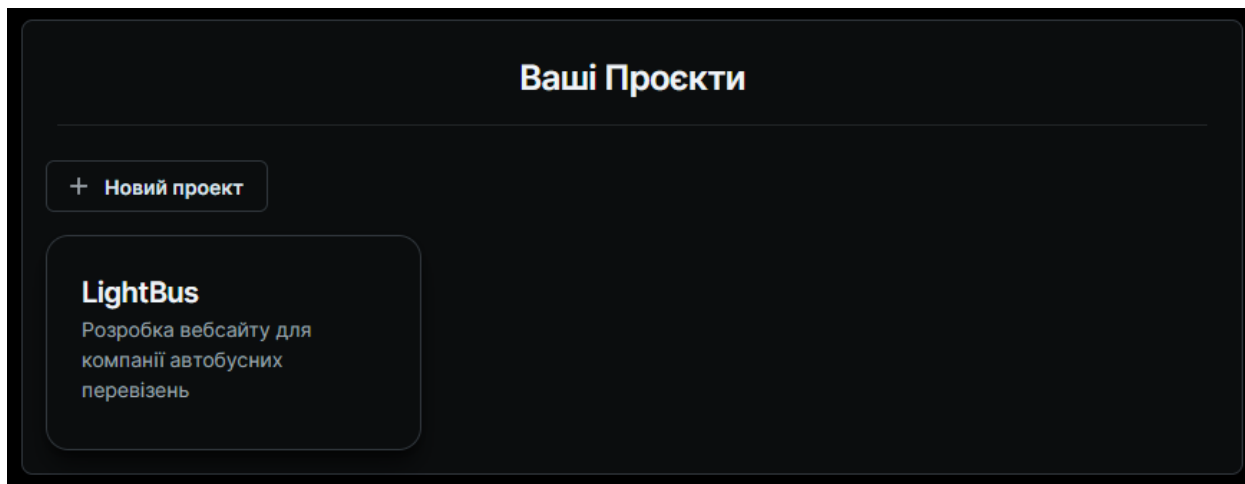


Рисунок 3.6 – Сторінка з проєктами

Сторінка проєкту розподілена на дві основні частини, кожна з яких відіграє свою роль у зручному та ефективному управлінні проєктом (рис. 3.7). Спочатку користувач статичний блок з інформацією про сам проєкт, який може містити загальний опис, необхідні для розуміння сутності проєкту. Цей блок допомагає користувачеві швидко ознайомитися з основною інформацією та контекстом проєкту.

Далі слідує динамічна робоча частина, яка становить серце управління проєктом. Тут розташовані дві вкладки: "робоча дошка" та "налаштування проєкту", кожна з яких спрямована на вирішення конкретних завдань у керуванні проєктом.

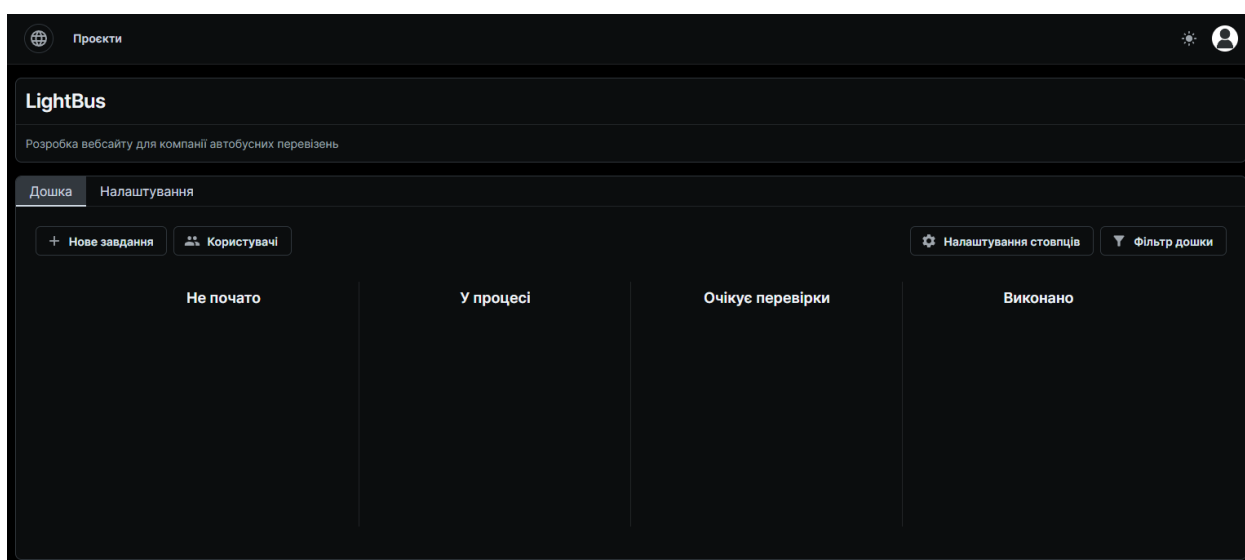


Рисунок 3.7 – Сторінка проєкті з дошку завдань

На верхній частині робочої дошки розміщені різноманітні інструменти, які дозволяють швидко та ефективно працювати з завданнями. Сюди входять кнопки для створення нових завдань, додавання користувачів до проєкту, налаштування стовпців дошки та фільтрація завдань. Цей набір інструментів надає користувачеві повний контроль над процесом роботи, дозволяючи легко організувати та управляти завданнями у межах проєкту.

За замовчуванням, система автоматично створює чотири основні колонки для організації завдань: "Не почато", "У процесі", "Очікує перевірки" та "Виконано". Це стандартна структура, яка допомагає користувачам легко відслідковувати статуси різних завдань та їхній прогрес у рамках проєкту.

Проте, система також надає можливість користувачам налаштувати ці колонки відповідно до своїх потреб. Для цього користувачу потрібно натиснути кнопку "Налаштування стовпців", яка знаходиться у правому верхньому куті дошки. Це відкриє модальне вікно налаштувань, де можна відредагувати розміщення та налаштування колонок (рис 3.8).

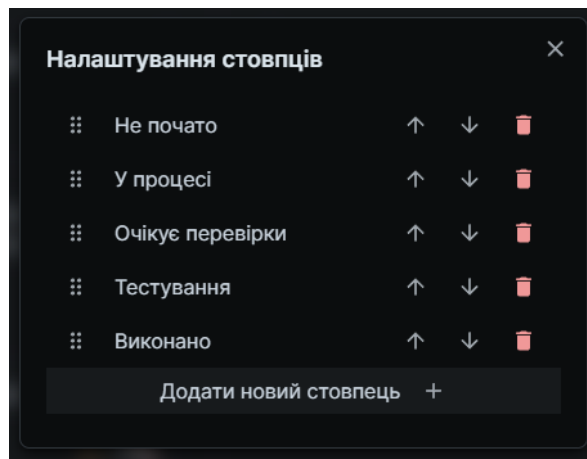


Рисунок 3.8 – Модальне вікно з налаштуваннями стовпців

У цьому модальному вікні користувач може виконати різноманітні дії, такі як змінити назви стовпців подвійним натиском на назву, змінити їх порядок, додати нові або видалити наявні стовпці. Важливо зазначити, що користувач може видаляти лише порожні стовпці, що не містять карток із завданнями, оскільки це дає йому можливість зберігати порядок та організацію завдань у системі (рис. 3.9).

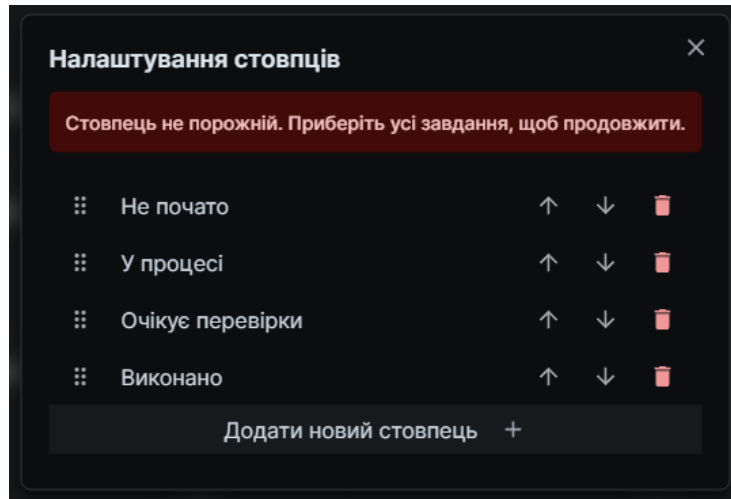


Рисунок 3.9 – Модальне вікно з налаштуваннями стовпців

Створення нового завдання відбувається через виклик модального вікна, де користувачеві пропонується заповнити обов'язкові поля, такі як назва завдання, яка є ключовою для його ідентифікації та організації (рис. 3.10). Крім того, він має можливість додати додатковий опис для більш детального уявлення про суть завдання. Також користувач може вказати оцінку у вигляді "story points", що відображає рівень складності завдання та допомагає у плануванні робіт.

Рисунок 3.10 – Створення нового завдання

Створене завдання негайно додається до дошки у колонці "Not started", щоб користувач міг одразу почати його виконання (рис. 3.11). За

замовчуванням, завдання отримує звичайний пріоритет та не має призначеного виконавця, але це можна легко змінити пізніше.



Рисунок 3.11 – Поява створеного завдання на дошці

Клацнувши на конкретне завдання на дошці, користувач отримує доступ до докладної інформації про нього через модальне вікно (рис. 3.12). Тут він може переглянути всі важливі деталі завдання, такі як назва, опис, оцінка "story points", пріоритет, виконавець тощо. Крім того, користувач має можливість визначити пріоритет завдання, обравши один з трьох варіантів: "високий", "звичайний" або "низький". Це дозволяє чітко визначити важливість та терміновість завдання у контексті загального проєкту та його пріоритетів.

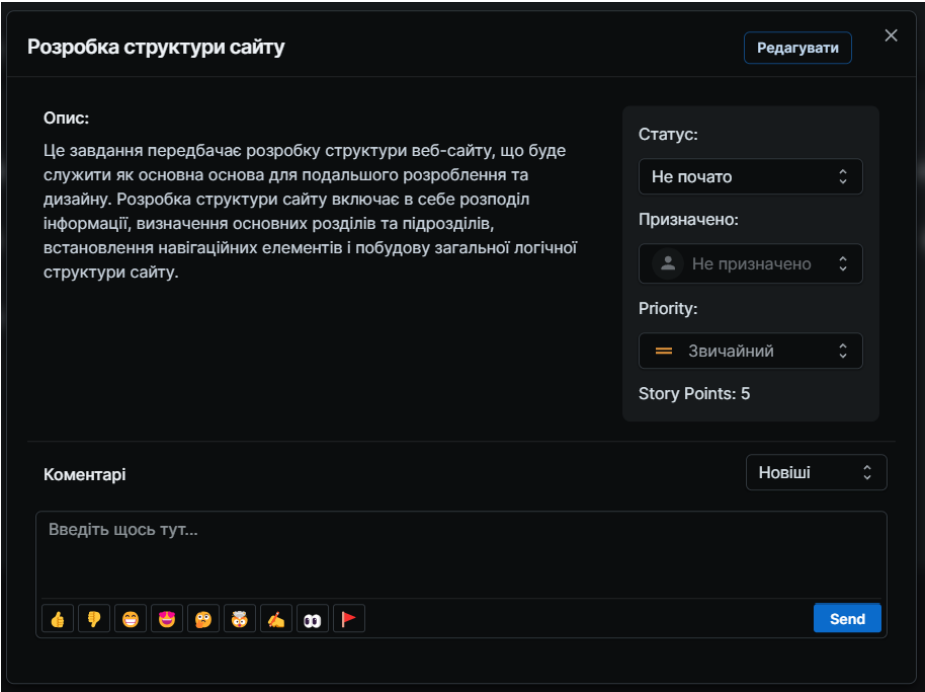


Рисунок 3.12 – Модальне вікно з інформацією про завдання

Якщо користувачу потрібно оновити дані про завдання, у правому верхньому куті модального вікна він може знайти кнопку редагування. Натиснувши її, користувач переходить у режим редагування завдання, де він має можливість змінити заголовок, опис, оцінку та пріоритетність завдання (рис. 3.13).

Завдяки цим можливостям користувач може ефективно управляти завданнями, вносячи необхідні зміни, такі як призначення виконавця, зміна пріоритету або відредагування деталей завдання, щоб краще відповідати поточним вимогам та обставинам проєкту.

Режим редагування

Назва

Розробка структури сайту

Опис

Це завдання передбачає розробку структури веб-сайту, що буде служити як основна основа для подальшого розроблення та дизайну. Розробка структури сайту включає в себе розподіл інформації, визначення основних розділів та підрозділів, встановлення навігаційних елементів і побудову загальної логічної структури сайту.

Story Points

5

Пріоритет

Звичайний

Скасувати

Зберегти

Рисунок 3.13 – Режим редагування завдання

Кожне завдання обов'язково має секцію з коментарями, яка надає користувачам можливість обмінюватися нотатками та зауваженнями щодо нього (рис. 3.14). У цій секції користувачі можуть залишати свої коментарі, додавати нові нотатки та виражати свої думки щодо завдання.

Крім того, дані коментарі можна відсортувати за різними критеріями, такими як "новіші" або "старіші", щоб забезпечити зручну навігацію та швидкий доступ до необхідної інформації. Це дозволяє користувачам

ефективно вести діалог та взаємодіяти у рамках завдання, організовуючи комунікацію відповідно до їхніх потреб та вимог.

Розробити функціонал реєстрації та входу Редагувати ✕

Опис:
Завдання "Розробити функціонал реєстрації та входу" передбачає створення системи, яка забезпечить користувачам можливість створення облікового запису на вебсайті, а також авторизацію для доступу до особистого кабінету. Для досягнення цієї мети, необхідно розробити інтерфейс для реєстрації, де користувачі зможуть ввести свої особисті дані та обрати облікові дані для входу. Крім того, необхідно реалізувати механізми перевірки введених даних на валідність, збереження інформації в базі даних та надання доступу до особистого кабінету для авторизованих користувачів.

Статус:
Очікує перевірки
Призначено: Валентин Дмитр
Priority: Високий
Story Points: 3

Коментарі Новіші

Введіть щось тут...

👍 🗨️ 🤔 🤔 🤔 🤔 🤔 🤔 🚩 Send

Дмитро Павленко 03.06.2024, 22:45:43
Завтра вранці перегляну 🗨️

Валентин Дмитрук 03.06.2024, 22:44:17
Завдання виконано. Потребую, щоб хтось переглянув зміни

Рисунок 3.14 – Модальне вікно завдання з коментарями

Для зручної співпраці в команді адміністратор проєкту має можливість додавати інших користувачів через спеціальне модальне вікно, яке знаходиться правій верхній частині дошки та призначене для управління користувачами (рис. 3.15). У цьому вікні відображається список користувачів, які вже приєдналися до проєкту, а також є пошуковий рядок, за допомогою якого можна швидко знайти та додати потрібного користувача до проєкту.

При додаванні нового користувача до проєкту за замовчуванням він отримує роль звичайного користувача (рис. 3.16). Проте адміністратор має повний контроль над ролями користувачів і може змінювати їх в будь-який час відповідно до потреб проєкту.

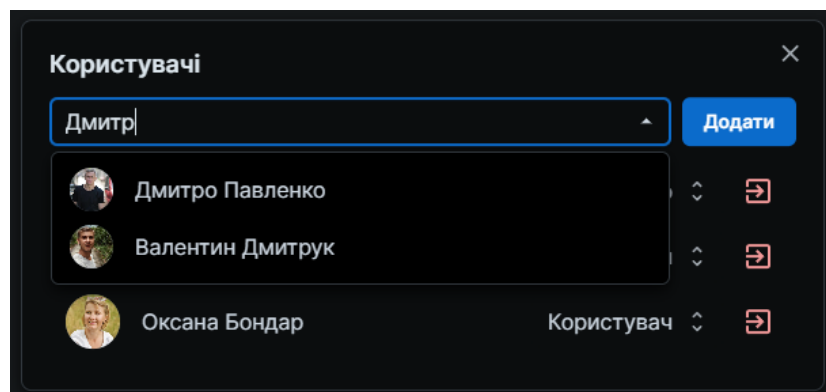


Рисунок 3.15 – Пошук користувачів

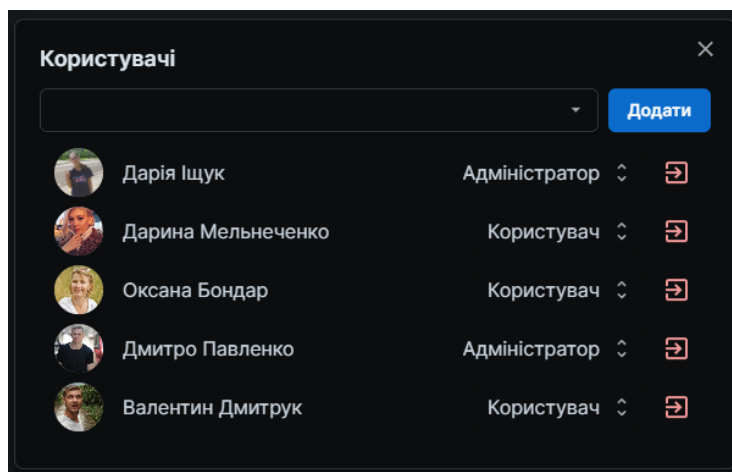


Рисунок 3.16 – Модальне вікно з доданими новими користувачами проєкту

Важливо відзначити, що додавання та видалення користувачів може здійснювати лише адміністратор проєкту (рис. 3.17). Це забезпечує високий рівень безпеки та контролю над доступом до інформації та ресурсів проєкту. Такий підхід сприяє ефективній організації роботи та запобігає можливим конфліктам або порушенням безпеки.

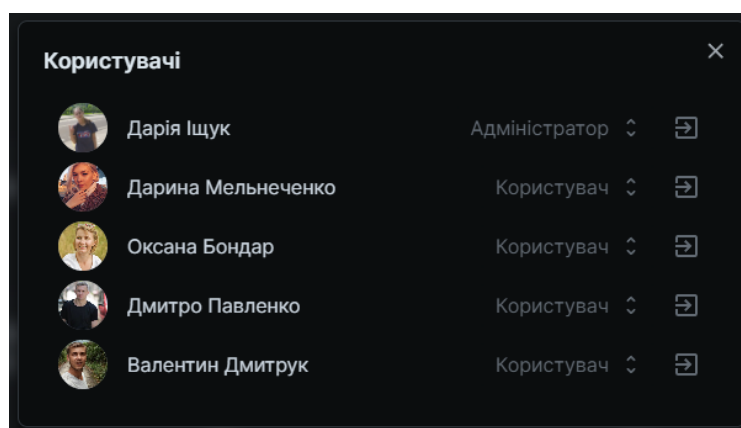


Рисунок 3.17 – Вид модального вікна якщо користувач не є адміністратором

Загалом, наповнена дошка під час роботи в команді має наступний вигляд: у кожному стовпчику завдання сортуються за пріоритетом, з найважливішими завданнями, які розміщуються на початку стовпчика (рис. 3.18). Це дозволяє команді швидко орієнтуватися та визначити, над чим потрібно працювати у першу чергу.

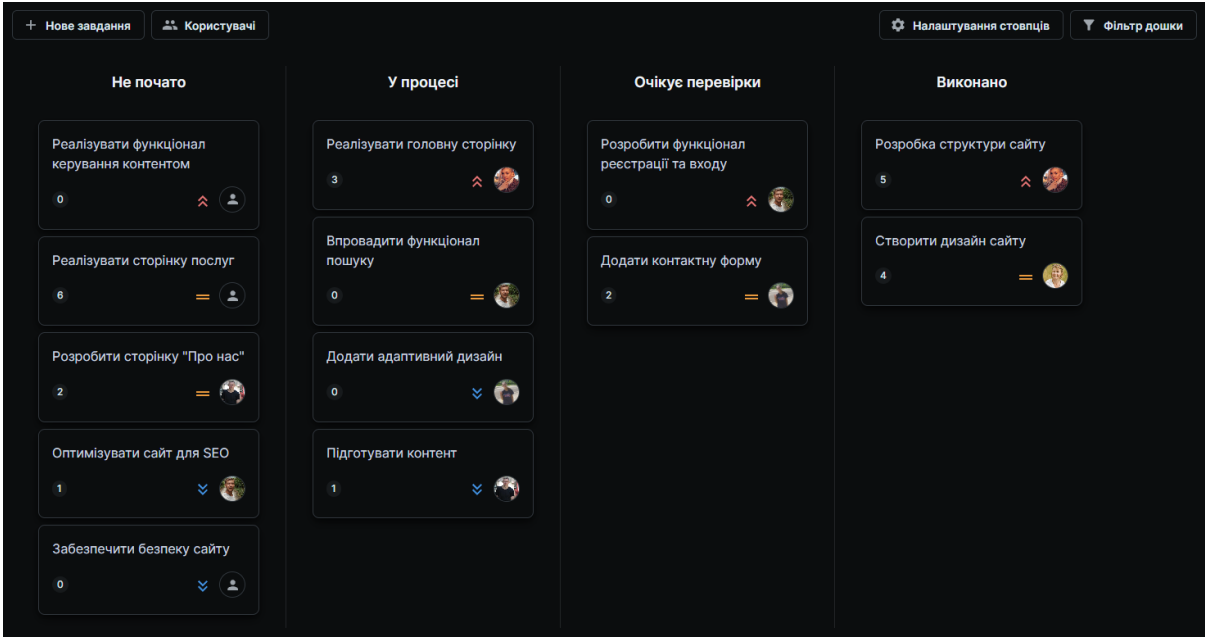


Рисунок 3.18 – Заповнена дошка

Якщо дошка має велику кількість завдань та користувачів (рис. 3.19), які працюють над ними, команда може скористатися функцією фільтрації. Цей інструмент дозволяє відфільтрувати завдання за конкретними критеріями, такими як користувачі, які призначені на завдання, та пріоритетність завдань (рис. 3.20). Такий підхід допомагає зосередитися на найважливіших завданнях та ефективно розподілити ресурси команди.

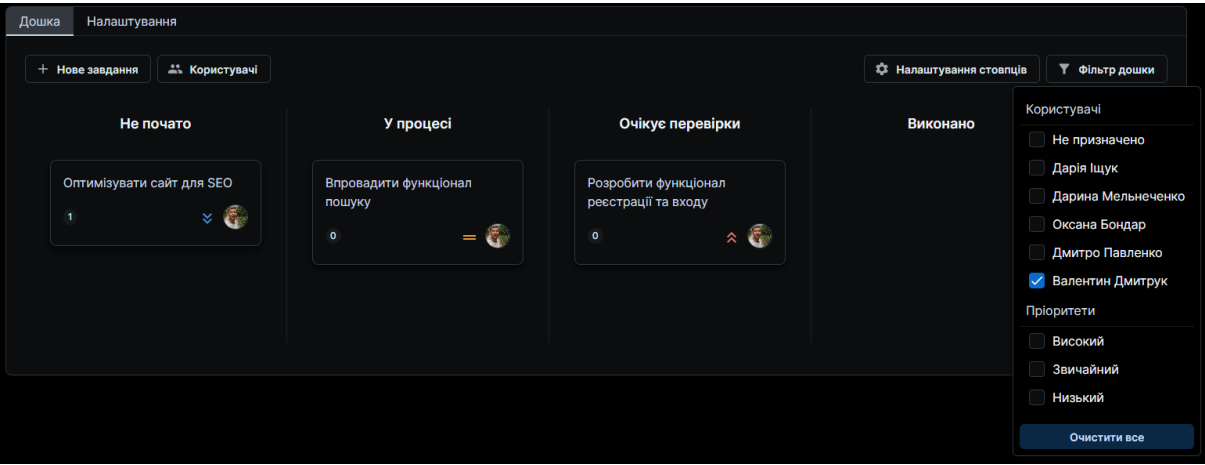


Рисунок 3.19 – Фільтрація дошки лише за користувачем

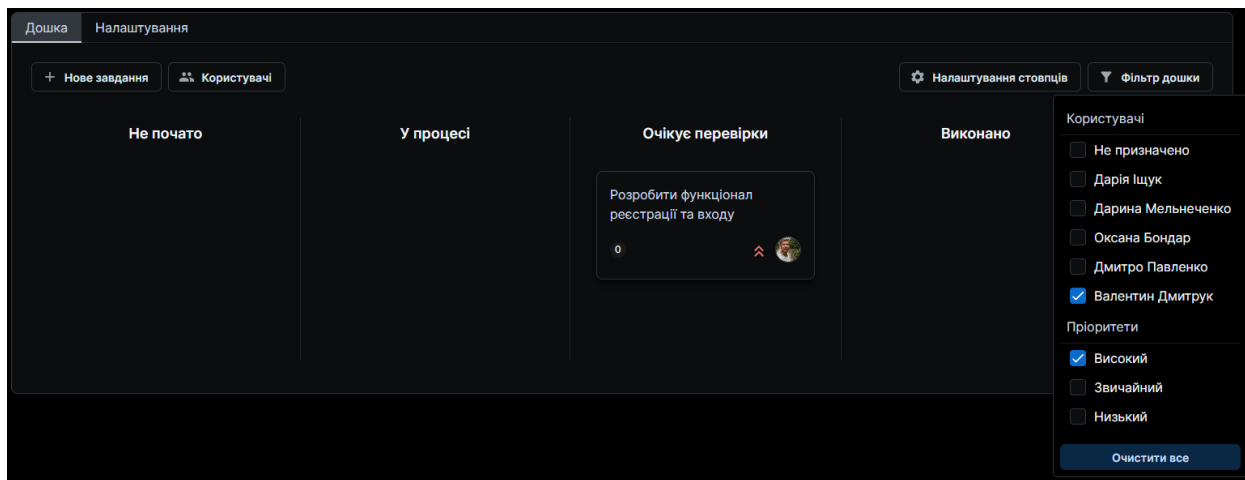


Рисунок 3.20 – Фільтрація дошки за користувачем та пріоритетністю

За допомогою вкладок, розташованих над дошкою з завданнями, користувач може легко перейти до налаштувань проекту (рис. 3.21). Якщо користувач має права адміністратора в проекті, він отримує можливість редагувати дані про проект та, за потреби, навіть видаляти його. Це важлива функціональність, яка дозволяє адміністратору забезпечити потрібні налаштування та керувати проектом відповідно до поточних вимог.

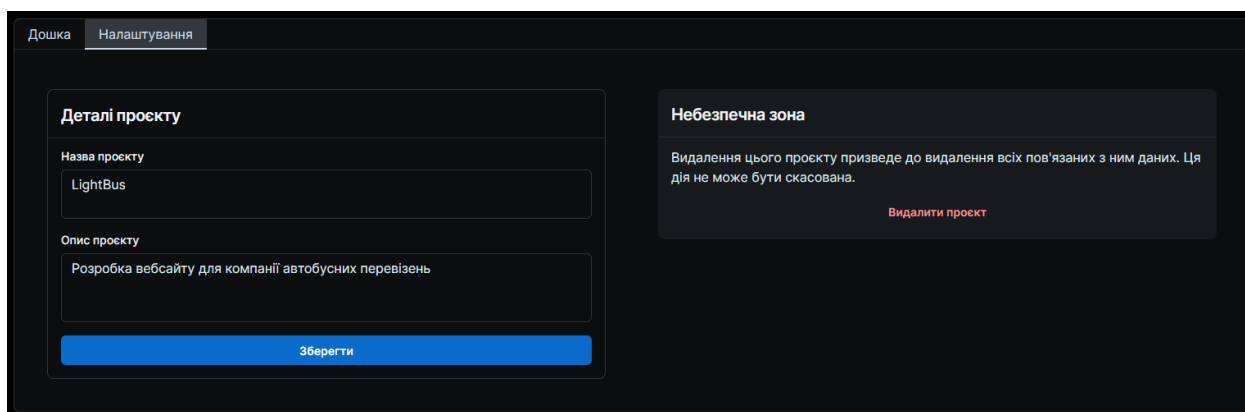


Рисунок 3.21 – Налаштувань проекту, якщо користувач є адміністратор проекту

Однак, якщо користувач не має прав адміністратора в проекті, ці функції будуть недоступні для нього (рис. 3.22). Це захищає проект від неправомірних змін та дозволяє зберегти його цілісність і безпеку. Такий підхід допомагає ефективно керувати доступом та забезпечує контроль над даними та налаштуваннями проекту.

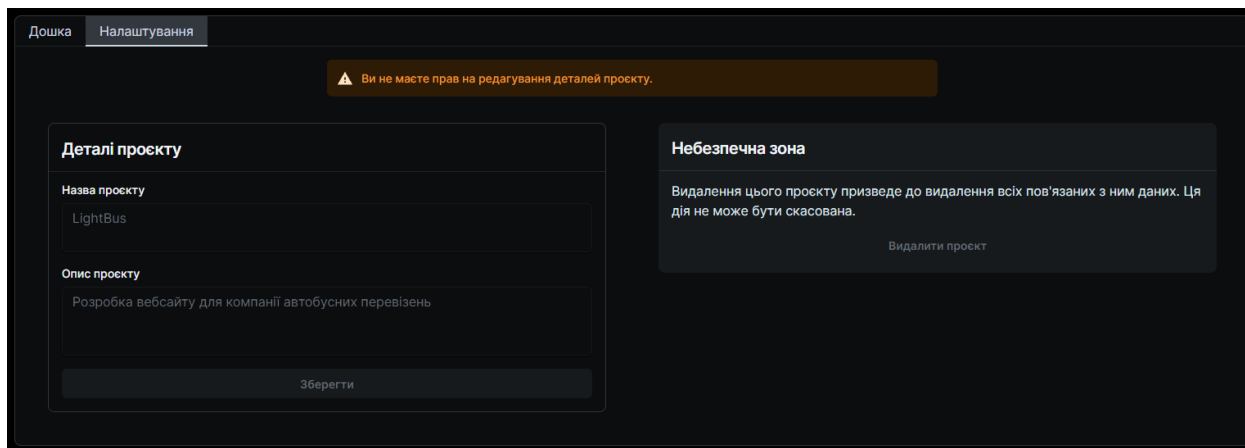


Рисунок 3.22 – Налаштувань проєкту, якщо користувач не має прав адміністратора

На навігаційній панелі в лівій частині розташована кнопка для повернення на головну сторінку з усіма проєктами, що дозволяє користувачеві швидко перейти до списку всіх проєктів. Справа від цієї кнопки розміщений перемикач для зміни теми: світлої або темної. Також тут знаходиться аватар користувача.

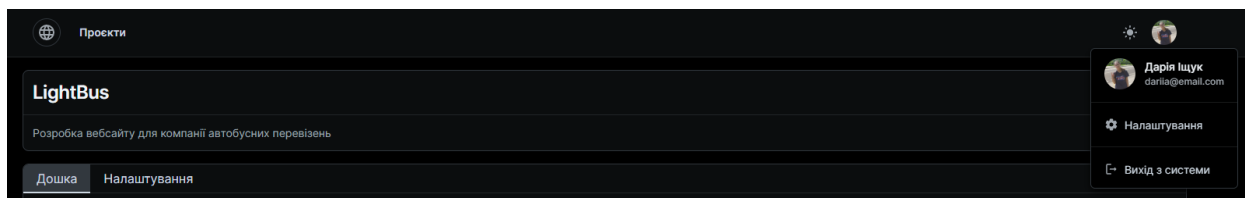


Рисунок 3.23 – Навігаційна панель

Натиснувши на аватар, користувач має можливість перейти до налаштувань особистого профілю або вийти з системи. На сторінці профілю користувач може вносити зміни в особисту інформацію, змінювати аватар, пароль або видалити свій акаунт. Це надає користувачеві контроль над його особистими налаштуваннями та приватними даними.

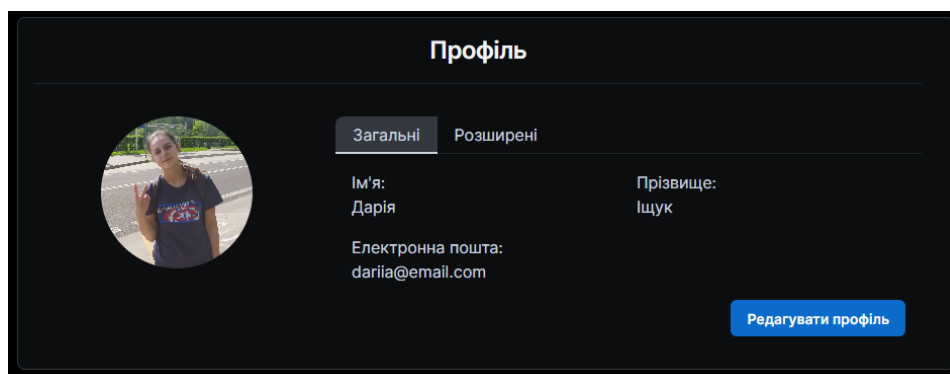


Рисунок 3.24 – Сторінка профілю з загальними налаштуваннями

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						59
Змн.	Арк.	№ докум.	Підпис	Дата		

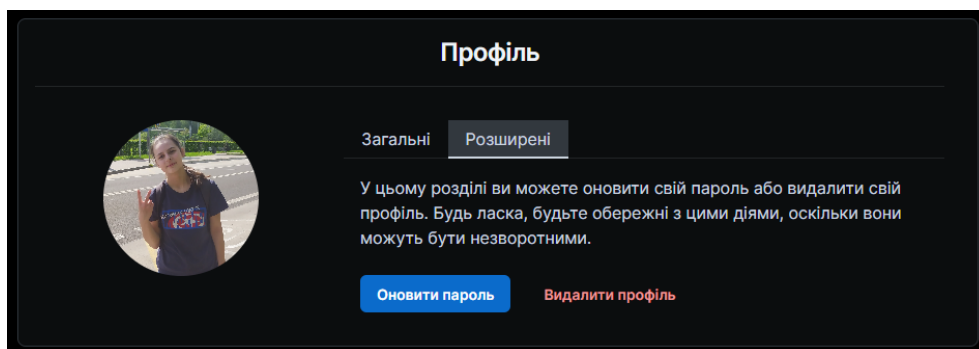


Рисунок 3.25 – Сторінка профілю з розширеними налаштуваннями

3.3. Тестування роботи програмного комплексу

Тестування програмного продукту є невід'ємною частиною розробки та забезпечує надійність, стабільність та відповідність вимогам кінцевого користувача. Метою тестування є виявлення та усунення помилок, неточностей та інших дефектів у програмному коді.

Процес тестування програмного продукту проводився у два етапи. Перший етап був присвячений функціональному тестуванню логіки систем, тобто серверної частини. Для цього було використано метод Unit тестування [28], який дозволяє перевірити кожен окрему функцію або модуль програми. Для реалізації Unit тестування було використано бібліотеки Junit [29] та Mockito [30].

Junit є популярною бібліотекою для тестування Java-коду, яка дозволяє створювати та запускати тести, перевіряти результати та формувати звіти. Вона забезпечує зручні інструменти для організації тестів, визначення очікуваних результатів і аналізу результатів тестування. Mockito, у свою чергу, є інструментом для створення мок-об'єктів, які імітують поведінку реальних об'єктів у тестах. Це дозволяє ізолювати тестовані компоненти від залежностей та зосередитися на їхній внутрішній логіці.

Завдяки використанню цих інструментів вдалося покрити усі ключові класи, такі як контролери та сервіси, що напряму стосуються бізнес-логіки вебдодатку. Було перевірено коректність виконання бізнес-логіки, обробки виключних ситуацій та взаємодії з іншими компонентами системи.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						60
Змн.	Арк.	№ докум.	Підпис	Дата		

Coverage java in project-management-server-app			
Element ^	Class, %	Method, %	Line, %
ztu	66% (32/48)	56% (150/264)	43% (360/826)
edu	66% (32/48)	56% (150/264)	43% (360/826)
ua	66% (32/48)	56% (150/264)	43% (360/826)
projectmanagementserverapp	66% (32/48)	56% (150/264)	43% (360/826)
auth	64% (11/17)	52% (42/80)	50% (86/172)
> config	0% (0/3)	0% (0/9)	0% (0/38)
> controller	100% (1/1)	100% (3/3)	100% (9/9)
> dto	71% (5/7)	51% (15/29)	51% (15/29)
> entity	100% (2/2)	65% (13/20)	66% (14/21)
> repository	100% (0/0)	100% (0/0)	100% (0/0)
> service	75% (3/4)	57% (11/19)	64% (48/75)
profile	33% (2/6)	35% (10/28)	28% (49/169)
> controller	100% (1/1)	100% (8/8)	81% (47/58)
> dto	50% (1/2)	22% (2/9)	22% (2/9)
> service	0% (0/3)	0% (0/11)	0% (0/102)
project	79% (19/24)	63% (98/155)	46% (225/484)
> controller	100% (3/3)	100% (23/23)	85% (78/91)
> dto	87% (7/8)	56% (27/48)	56% (27/48)
> entity	85% (6/7)	65% (32/49)	66% (35/53)
> mapper	0% (0/3)	0% (0/6)	0% (0/54)
> repository	100% (0/0)	100% (0/0)	100% (0/0)
> service	100% (3/3)	55% (16/29)	35% (85/238)
ProjectManagementServerAppApplic	0% (0/1)	0% (0/1)	0% (0/1)

Рисунок 3.26 – Покриття коду unit-тестами

Другий етап тестування зосереджений на перевірці зручності інтерфейсу та функціоналу клієнтської частини. Для цього було вибрано метод тестування, який передбачає створення чек-лістів - списків головних функцій, які потрібно перевірити. Було створено та проведено низку чек-лістів, результати яких наведені нижче.

Таблиця 3.1

Чек-ліст функціоналу для тестування

№	Функціонал	Результат
1	Реєстрація та вхід	Пройдено
2	Отримання проєктів користувача	Пройдено
3	Створення нового проєкту	Пройдено
4	Редагування даних про проєкт	Пройдено
5	Видалення проєкту	Пройдено
6	Отримання дошки проєкту	Пройдено
7	Створення нового завдання	Пройдено

8	Створення коментарів до завдання	Пройдено
9	Редагування завдання	Пройдено
10	Переміщення завдання між колонками	Пройдено
11	Перейменування колонки дошки	Пройдено
12	Видалення колонки	Пройдено
13	Додавання нової колонки	Пройдено
14	Зміна порядку колонок	Пройдено
15	Додавання нового користувача	Пройдено
16	Видалення користувача	Пройдено
17	Зміна ролі користувача	Пройдено
18	Фільтрація дошки	Пройдено

Завдяки проведеному тестуванню, вдалося виявити та усунути дефекти у програмному коді, підвищити надійність та стабільність роботи програмного продукту, а також забезпечити його відповідність вимогам кінцевих користувачів.

Висновки до третього розділу

У третьому розділі було порядок встановлення та налаштування параметрів системи, а налаштування робочого середовища та конфігураційних файлів системи.

Було розглянуто структуру інтерфейсу вебдодатку та продемонстровано його роботу: створення та управління проєктами, робота з дошкою, відслідковування створених задач, принцип роботи у команді та керування особистим профілем.

Також було розглянуто процес тестування програмного комплексу за допомогою юніт-тестування та чек-лістів для перевірки справності роботи функціоналу.

ВИСНОВКИ

В ході дипломного проєкту було розроблено вебдодаток для управління ІТ-проєктами, який може значно покращити ефективність організацій при веденні ІТ-проєктів. Аналіз вимог та визначення функціональних вимог до вебдодатку допомогли визначити необхідний функціонал та структуру системи. Використання сучасних веборієнтованих технологій та стеку технологій дозволило розробити ефективну та інтегровану систему для управління ІТ-проєктами.

Під час розробки вебдодатку було враховано актуальність теми та потреби користувачів, що дозволило створити зручний та функціональний інтерфейс. Використання Drag and Drop функціоналу дозволило користувачам легко переміщати картки завдання між колонками, що допомагає їм відстежувати прогрес роботи над завданнями та співпрацювати з іншими членами команди.

Тестування вебдодатку допомогло перевірити коректність його роботи та відповідність вимогам. Виправлення виявлених недоліків допомогло покращити якість та стабільність роботи системи.

Розроблений вебдодаток може бути використаний в реальних умовах різних підприємств і організацій, які займаються розробкою або управлінням інформаційними технологіями, стартапами, а також для особистого використання. Він може значно покращити ефективність та результативність ІТ-проєктів, допомагаючи командам краще співпрацювати, економити час та ресурси, а також підвищити якість та результативність ІТ-проєктів.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						63
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ЛІТЕРАТУРИ

1. Trello [Електронний ресурс] – Режим доступу до ресурсу: <https://trello.com/home>.
2. Asana [Електронний ресурс] – Режим доступу до ресурсу: <https://asana.com/>.
3. Jira [Електронний ресурс] – Режим доступу до ресурсу: <https://www.atlassian.com/software/jira>.
4. Роберт Мартін, Чиста архітектура. – Харків: Фабула, 2020. – 368 с.
5. Роберт Мартін, Чистий код. – Харків: Фабула, 2020. – 448с.
6. What is three-tier architecture? [Електронний ресурс] // IBM – Режим доступу до ресурсу: <https://www.ibm.com/topics/three-tier-architecture>.
7. Jackson-Barnes S. SPA Vs. MPA Explained: Differences Between Single-Page Applications and Multi-Page Applications [Електронний ресурс] / Shannon Jackson-Barnes // Orient. – 2023. – Режим доступу до ресурсу: <https://www.orientsoftware.com/blog/spa-vs-mpa/>.
8. What’s the Difference Between Monolithic and Microservices Architecture? [Електронний ресурс] – Режим доступу до ресурсу: <https://aws.amazon.com/compare/the-difference-between-monolithic-and-microservices-architecture/>.
9. Pawlan D. Relational vs. Non-Relational Database: Pros & Cons [Електронний ресурс] / David Pawlan // Aloa Blog – Режим доступу до ресурсу: <https://aloe.co/blog/relational-vs-non-relational-database-pros-cons>.
10. React [Електронний ресурс] – Режим доступу до ресурсу: <https://react.dev/reference/react>.
11. Joy UI [Електронний ресурс] – Режим доступу до ресурсу: <https://mui.com/joy-ui/getting-started/>.
12. Java Documentation [Електронний ресурс] // Oracle – Режим доступу до ресурсу: <https://docs.oracle.com/en/java/>.
13. Spring Framework Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.spring.io/spring-framework/reference/index.html>.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						64
Змн.	Арк.	№ докум.	Підпис	Дата		

14. Crusoveanu L. Intro to Inversion of Control and Dependency Injection with Spring [Електронний ресурс] / L. Crusoveanu, K. Gilmore // Baeldung. – 2024. – Режим доступу до ресурсу: <https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring>.
15. MySQL [Електронний ресурс] – Режим доступу до ресурсу: <https://dev.mysql.com/doc/>.
16. Java JDBC API [Електронний ресурс] // Oracle – Режим доступу до ресурсу: <https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/>.
17. React Context Hooks [Електронний ресурс] - Режим доступу до ресурсу: <https://react.dev/reference/react/context-hooks>
18. React DnD [Електронний ресурс] – Режим доступу до ресурсу: <https://react-dnd.github.io/react-dnd/docs/overview>.
19. Axios [Електронний ресурс] – Режим доступу до ресурсу: <https://axios-http.com/docs/intro>.
20. Axios in React [Електронний ресурс] // GeeksForGeeks. – 2024. – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/axios-in-react-a-guide-for-beginners/>.
21. Introduction to JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу: <https://jwt.io/introduction>.
22. Enabling Cross Origin Requests for a RESTful Web Service [Електронний ресурс] – Режим доступу до ресурсу: <https://spring.io/guides/gs/rest-service-cors>.
23. Szrajbman S. CRUD Application With React and Spring Boot [Електронний ресурс] / S. Szrajbman, G. Martin // Baeldung. – 2024. – Режим доступу до ресурсу: <https://www.baeldung.com/spring-boot-react-crud>.
24. Gordadze I. Spring Security With JWT for REST API [Електронний ресурс] / Ioram Gordadze – Режим доступу до ресурсу: <https://www.toptal.com/spring/spring-security-tutorial>.
25. Node.js [Електронний ресурс] – Режим доступу до ресурсу: <https://nodejs.org/docs/latest/api/>.

26. JDK 17 Documentation [Електронний ресурс] – Режим доступу до ресурсу:
<https://docs.oracle.com/en/java/javase/17/>.
27. Accessing data with MySQL [Електронний ресурс] – Режим доступу до ресурсу: <https://spring.io/guides/gs/accessing-data-mysql>. Aibin M. Testing in Spring Boot [Електронний ресурс] / Michal Aibin // Baeldung. – 2024. – Режим доступу до ресурсу: <https://www.baeldung.com/spring-boot-testing>.
28. JUnit 5 User Guide [Електронний ресурс] – Режим доступу до ресурсу:
<https://junit.org/junit5/docs/current/user-guide/>.
29. Mockito [Електронний ресурс] – Режим доступу до ресурсу:
<https://site.mockito.org/>.

ДОДАТКИ

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						67
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг компоненту AuthProvider:

```

import axios from "axios";
import { jwtDecode } from "jwt-decode";
import { createContext, useContext, useEffect, useMemo, useState } from
"react";

const AuthContext = createContext();

export default function AuthProvider({ children }) {
  const [token, setToken_] = useState(localStorage.getItem("token"));
  const [user, setUser_] = useState(() => {
    const savedUser = localStorage.getItem("user");
    return savedUser ? JSON.parse(savedUser) : null;
  });
  const [avatar, setAvatar_] = useState(localStorage.getItem("avatar"));

  const setToken = (newToken) => {
    setToken_(newToken);
  };

  const setUser = (data) => {
    setUser_(data);
    localStorage.setItem('user', JSON.stringify(data));
  };

  useEffect(() => {
    if (token) {
      const decodedToken = jwtDecode(token);
      const currentTime = Date.now() / 1000;

      if (decodedToken.exp < currentTime) {
        setToken(null);
      } else {
        axios.defaults.headers.common["Authorization"] = "Bearer " +
token;
        localStorage.setItem('token', token);
      }
    } else {
      delete axios.defaults.headers.common["Authorization"];
      localStorage.removeItem('token');
      localStorage.removeItem('user');
      localStorage.removeItem('avatar');
    }
  }, [token]);

  const contextValue = useMemo(
    () => ({
      token,
      setToken,
      user,
      setUser,
      avatar
    }),
    [token]
  );

  return (
    <AuthContext.Provider
value={contextValue}>{children}</AuthContext.Provider>
  );
};

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						68
Змн.	Арк.	№ докум.	Підпис	Дата		

```
export const useAuth = () => {
  return useContext(AuthContext);
};
```

Лістинг компоненту ProjectPage:

```
import { useState, useEffect } from "react";
import { Card, CssBaseline, Divider, Grid, TabPanel } from "@mui/joy";
import Layout from "../components/common/Layout";
import Header from "../components/common/Header";
import Stack from "@mui/joy/Stack";
import { useNavigate, useParams } from "react-router-dom";
import Typography from "@mui/joy/Typography";
import TabList from "@mui/joy/TabList";
import Tab from "@mui/joy/Tab";
import Tabs from "@mui/joy/Tabs";
import axios from "axios";
import { useAuth } from "../auth/AuthProvider";
import ProjectSettings from "../components/board/ProjectSettings";
import Board from "../components/board/Board";

const GET_URL = '/project/slug';
const GET_ROLE_URL = '/project/role';

export default function ProjectPage() {
  const { token } = useAuth();
  const { slug } = useParams();
  const [project, setProject] = useState([]);
  const [userRole, setUserRole] = useState(null);
  const navigate = useNavigate();

  useEffect(() => {
    fetchProject();
  }, [slug, token]);

  useEffect(() => {
    if (project && project.id) {
      isAdmin();
    }
  }, [project, token]);

  const fetchProject = async () => {
    try {
      const response = await axios.get(process.env.REACT_APP_API_URL +
GET_URL, {
        headers: { Authorization: `Bearer ${token}` },
        params: { slug: slug }
      });
      console.info("Project fetched successfully!", response.data);
      setProject(response.data);
    } catch (error) {
      console.error("Failed to fetch project's data:", error);
      navigate("/not_found", { replace: true });
    }
  };

  const isAdmin = async () => {
    try {
      console.log(project.id);
      const response = await axios.get(process.env.REACT_APP_API_URL +
GET_ROLE_URL, {
        headers: { Authorization: `Bearer ${token}` },
        params: {
```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						69
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        projectId: project.id
      }
    });
    console.log(response.data);
    setUserRole(response.data);
    console.info("Project role fetched successfully!",
response.data);
  } catch (error) {
    console.error("Failed to fetch project's role data:", error);
  }
};

return (
  <main>
    <CssBaseline />
    <Layout.Root
      sx={{
        gridTemplateColumns: {
          xs: '1fr',
          sm: 'minmax(64px, 200px) minmax(450px, 1fr)',
          md: 'minmax(160px, 300px) minmax(600px, 1fr)
minmax(300px, 420px)',
        }
      }}
    >
      <Layout.Header>
        <Header />
      </Layout.Header>
      <Layout.Main>
        <Stack
          spacing={2}
          direction="column"
          sx={{
            width: {
              xs: '100%',
              sm: '100%',
              md: '100%',
              lg: '90%',
              xl: '80%',
            },
            margin: 'auto',
            display: 'flex',
            justifyContent: 'center',
            alignItems: 'center',
          }}
        >
          <Stack
            spacing={2}
            direction="column"
            sx={{ width: '100%' }}
            justifyContent="center"
          >
            <Card sx={{ padding: 0 }}>
              <Stack
                spacing={0}
                direction="column"
                sx={{ width: '100%' }}
                divider={<Divider />}
              >
                <Stack sx={{ padding: 1.5 }}>
                  <Typography
level="h3">{project.title}</Typography>
                </Stack>
                <Stack sx={{ padding: 1.5 }}>

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        <Typography level="body-sm">{project.description}</Typography>
      </Stack>
    </Stack>
  </Card>

  <Tabs
    variant="outlined"
    defaultValue={0}
    sx={{
      borderRadius: 'md',
      boxShadow: 'sm',
      overflow: 'auto',
    }}
    scrollButtons="auto"
    aria-label="scrollable auto tabs example"
  >
    <TabList>
      <Tab>Дошка</Tab>
      <Tab>Налаштування</Tab>
    </TabList>
    <TabPanel value={0} sx={{ padding: 0 }}>
      <Board slug={slug} userRole={userRole}/>
    </TabPanel>
    <TabPanel value={1}>
      <ProjectSettings project={project}
setProject={setProject} userRole={userRole} />
    </TabPanel>
  </Tabs>
</Stack>
</Stack>
</Layout.Main>
</Layout.Root>
</main>
);
}

```

Лістинг компоненту Board:

```

import React, { useEffect, useState, useCallback } from "react";
import { DndProvider } from "react-dnd";
import { HTML5Backend } from "react-dnd-html5-backend";
import { Card, Divider, Stack, Grid } from "@mui/joy";
import { Column } from "../Column";
import { TaskCard } from "../TaskCard";
import CreateNewTask from "../modals/CreateNewTask";
import ColumnSettingsModal from "../modals/ColumnSettingsModal";
import axios from "axios";
import { useAuth } from "../../auth/AuthProvider";
import UserListModal from "../modals/UserListModal";
import BoardFilter from "../BoardFilter";

const GET_TASKS_URL = '/project/board';
const PUT_TASKS_URL = '/project/board';

export default function Board({ slug, userRole }) {
  const { token } = useAuth();
  const [projectId, setProjectId] = useState([]);
  const [items, setItems] = useState([]);
  const [columns, setColumns] = useState([]);
  const [users, setUsers] = useState([]);
  const [filteredItems, setFilteredItems] = useState([]);

  const priorities = ["high", "medium", "low"]

```

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						71
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    useEffect(() => {
      const intervalId = setInterval(fetchBoard, 10000); // 10000 ms = 10
seconds
      fetchBoard();
      return () => clearInterval(intervalId);
    }, []);

    const fetchBoard = () => {
      axios.get(process.env.REACT_APP_API_URL + GET_TASKS_URL, {
        headers: { Authorization: `Bearer ${token}` },
        params: { slug: slug }
      })
        .then(response => {
          setProjectId(response.data.projectId);
          setItems(response.data.tasks);
          setFilteredItems(response.data.tasks);
          setColumns(response.data.columns);
          setUsers(response.data.users);
        })
        .catch(error => {
          console.error("Failed to fetch project's data:", error);
        });
    };

    const saveBoard = () => {
      axios.put(process.env.REACT_APP_API_URL + PUT_TASKS_URL, {
        projectId: projectId,
        columns: columns,
        tasks: items
      }, {
        headers: { Authorization: `Bearer ${token}` },
      })
        .then(response => {
          console.info("We got saved!", response.data);
          console.info(items);
        })
        .catch(error => {
          console.error("Failed to save project's data:", error);
        });
    };

    const moveCardHandler = (dragIndex, hoverIndex) => {
      const dragItem = items[dragIndex];

      if (dragItem) {
        setItems((prevState) => {
          const coppiedStateArray = [...prevState];

          const prevItem = coppiedStateArray.splice(hoverIndex, 1,
dragItem);

          coppiedStateArray.splice(dragIndex, 1, prevItem[0]);

          return coppiedStateArray;
        });
      }
    };

    const returnItemsForColumn = (columnName) => {
      return filteredItems
        .filter((item) => item.stageColumn === columnName)
        .sort((a, b) => {
          const priorityOrder = { high: 3, medium: 2, low: 1 };

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						72
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        return priorityOrder[b.priority] - priorityOrder[a.priority];
    })
    .map((item, index) => (
        <TaskCard
            task={item}
            key={item.id}
            title={item.title}
            currentColumnName={item.stageColumn}
            setItems={setItems}
            index={index}
            moveCardHandler={moveCardHandler}
            columns={columns}
            users={users}
        />
    ));
};

const updateColumn = (oldName, newName) => {
    const updatedItems = items.map(item => {
        if (item.stageColumn === oldName) {
            return { ...item, stageColumn: newName };
        }
        return item;
    });
    setItems(updatedItems);
}

const isEmptyColumn = (columnName) => {
    const itemsForColumn = items.filter((item) => item.stageColumn ===
columnName);
    return itemsForColumn.length === 0;
};

const handleFilterChange = useCallback(({ users, priorities }) => {
    let filteredTasks = items;

    if (users.length > 0) {
        filteredTasks = filteredTasks.filter(item =>
            (users.includes('Unassigned') && !item.assignedTo) ||
            users.includes(item.assignedTo)
        );
    }

    if (priorities.length > 0) {
        filteredTasks = filteredTasks.filter(item =>
            priorities.includes(item.priority)
        );
    }

    setFilteredItems(filteredTasks);
}, [items]);

return (
    <Grid
        container
        spacing={2}
        sx={{
            margin: 'auto',
            padding: 2,
        }}
    >
        <Grid xs={12} sm={6}>
            <Stack direction="row" spacing={1}>
                <CreateNewTask tasks={items} setTasks={setItems}

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						73
Змн.	Арк.	№ докум.	Підпис	Дата		

```

projectId={projectId} />
      <UserListModal users={users} setUsers={setUsers}
projectId={projectId} userRole={userRole} />
    </Stack>
  </Grid>
  <Grid xs={12} sm={6}>
    <Stack direction="row" spacing={1} justifyContent="flex-end">
      <ColumnSettingsModal columns={columns}
setColumns={setColumns} updateColumn={updateColumn}
saveChanges={saveBoard}
isColumnEmpty={isColumnEmpty} />
      <BoardFilter users={users} priorities={priorities}
onFilterChange={handleFilterChange} />
    </Stack>
  </Grid>
  <Grid item xs={12}>

    <Card
      variant="plain"
      sx={{
        width: '100%',
        overflow: 'auto',
      }}
    >
      <DndProvider backend={HTML5Backend}>
        <Stack
          direction="row"
          alignItems="flex-start"
          spacing={2}
          divider={<Divider orientation="vertical" />}
        >
          {columns.map((column) => (
            <Column key={column} title={column}

columns={columns}>

              {returnItemsForColumn(column)}
            </Column>
          ))}
        </Stack>
      </DndProvider>
    </Card>
  </Grid>
</Grid>
);
}

```

Лістинг компоненту TaskCard:

```

import { useDrag, useDrop } from "react-dnd";
import React, { useRef, useState, useEffect } from "react";
import { Avatar, Box, Card, CardContent, Chip, Link, Sheet, Typography } from
"@mui/joy";
import TaskInfoModal from "../modals/TaskInfoModal";
import { useAuth } from "../../auth/AuthProvider";
import axios from "axios";
import Tooltip from "@mui/joy/Tooltip";
import PriorityIcon from "../modals/task_modal/PriorityIcon";

const PUT_TASK_URL = '/task/update';
const AVATAR_URL = '/profile/avatar';

export const TaskCard = ({
  task,
  title,
  index,

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						74
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        currentColumnName,
        moveCardHandler,
        setItems,
        columns,
        users
    }) => {
const { token } = useAuth();
const [open, setOpen] = useState(false);
const [avatarUrl, setAvatarUrl] = useState('');

useEffect(() => {
    const fetchAvatar = async () => {
        if (task.assignedTo) {
            try {
                const assignedUser = users.find(user => user.id ===
task.assignedTo);
                if (assignedUser) {
                    const response = await
axios.get(process.env.REACT_APP_API_URL + `${AVATAR_URL}/${assignedUser.id}`,
{
                        headers: { Authorization: `Bearer ${token}` },
                        responseType: 'blob'
                    });

                    setAvatarUrl(URL.createObjectURL(response.data));
                }
            } catch (error) {
                console.error('Failed to fetch avatar:', error);
            }
        }
    };

    fetchAvatar();
}, [task.assignedTo, token, users]);

const changeItemColumn = (currentItem, columnName) => {
    setItems((prevState) => {
        return prevState.map((e) => {
            return {
                ...e,
                stageColumn: e.title === currentItem.title ? columnName :
e.stageColumn
            };
        });
    });
    updateStageColumn(currentItem.task, columnName)
};

const updateStageColumn = (task, columnName) => {
    task.stageColumn = columnName
    axios
        .put(process.env.REACT_APP_API_URL + PUT_TASK_URL, task, {
            headers: { Authorization: `Bearer ${token}` }
        })
        .then(function (response) {
            console.info(response.data);
        })
        .catch(function (error) {
            console.error('Failed to update task column:', error);
        });
};

const ref = useRef(null);

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						75
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const [, drop] = useDrop({
  accept: "Our first type",
  hover(item, monitor) {
    if (!ref.current) {
      return;
    }
    const dragIndex = item.index;
    const hoverIndex = index;

    if (dragIndex === hoverIndex) {
      return;
    }

    const hoverBoundingRect = ref.current?.getBoundingClientRect();

    const hoverMiddleY =
      (hoverBoundingRect.bottom - hoverBoundingRect.top) / 2;

    const clientOffset = monitor.getClientOffset();

    const hoverClientY = clientOffset.y - hoverBoundingRect.top;

    if (dragIndex < hoverIndex && hoverClientY < hoverMiddleY) {
      return;
    }

    if (dragIndex > hoverIndex && hoverClientY > hoverMiddleY) {
      return;
    }

    moveCardHandler(dragIndex, hoverIndex);

    item.index = hoverIndex;
  }
});

const [{ isDragging }, drag] = useDrag({
  item: { index, title, currentColumnName, task },
  type: "Our first type",
  end: (item, monitor) => {
    const dropResult = monitor.getDropResult();

    if (dropResult) {
      const { name } = dropResult;
      for (const column of columns) {
        if (column === name) {
          changeItemColumn(item, column);
        }
      }
    }
  },
  collect: (monitor) => ({
    isDragging: monitor.isDragging()
  })
});

drag(drop(ref));

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

return (
  <Sheet>

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						76
Змн.	Арк.	№ докум.	Підпис	Дата		

```

<Card
  ref={ref}
  size="md"
  sx={{
    width: '270px',
    m: 1,
    boxShadow: 'md',
  }}
  onClick={handleOpen}
>
  <Link overlay underline="none">
    <Typography
      sx={{ overflow: 'hidden', textOverflow: 'ellipsis',
display: '-webkit-box', WebkitBoxOrient: 'vertical', WebkitLineClamp: 2 }}
      level="body-md"
    >
      {title}
    </Typography>
  </Link>
  <CardContent orientation="horizontal" sx={{
    display: 'flex',
    justifyContent: 'space-between',
    alignItems: 'center'
  }}>
    <div style={{ display: 'flex', alignItems: 'center' }}>
      <Chip variant="soft" size="sm"
sx={{}}>{task.storyPoints}</Chip>
    </div>
    <div style={{ display: 'flex', alignItems: 'center' }}>
      <Box sx={{paddingTop: 1.2, paddingRight: 1}}>
        <PriorityIcon priority={task.priority}/>
      </Box>
      {task.assignedTo == null ? (
        <Tooltip title="Unassigned" variant="plain">
          <Avatar size="sm" variant="outlined" />
        </Tooltip>
      ) : (
        <Tooltip
          title={users.find((user) => user.id ===
task.assignedTo)?.name}
          variant="plain"
        >
          <Avatar size="sm" variant="outlined"
src={avatarUrl} />
        </Tooltip>
      )}
    </div>
  </CardContent>
</Card>

<TaskInfoModal
  task={task}
  setTasks={setItems}
  open={open}
  setOpen={setOpen}
  users={users}
  columns={columns}
  changeColumn={updateStageColumn}
/>
</Sheet>
);
};

```

Лістинг компоненту Column:

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						77
Змн.	Арк.	№ докум.	Підпис	Дата		

```

import {useDrop} from "react-dnd";
import {Sheet, Typography} from "@mui/joy";

export const Column = ({ children, className, title, columns }) => {
  const [{ isOver, canDrop }, drop] = useDrop({
    accept: "Our first type",
    drop: () => ({ name: title }),
    collect: (monitor) => ({
      isOver: monitor.isOver(),
      canDrop: monitor.canDrop()
    }),
  });

  const getBackgroundColor = () => {
    if (isOver) {
      if (canDrop) {
        // return "rgb(239,239,241)";
      } else if (!canDrop) {
        // return "rgb(255,188,188)";
      }
    } else {
      return "";
    }
  };

  return (
    <>
      <Sheet
        color="neutral"
        sx={{minWidth: '300px' , p: 1, minHeight: 300}}
        ref={drop}
        style={{ backgroundColor: getBackgroundColor() }}
      >
        <Sheet
          variant="plain"
          sx={{height: '50px'}}
          style={{ backgroundColor: getBackgroundColor() }}
        >
          <Typography
            align="center"
            level="title-lg"
          >
            {title}
          </Typography >
        </Sheet>
        {children}
      </Sheet>
    </>
  );
};

```

Лістинг компоненту JwtServiceImpl

```

package ztu.edu.ua.projectmanagementserverapp.auth.service.impl;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.io.Decoders;
import io.jsonwebtoken.security.Keys;
import lombok.extern.log4j.Log4j2;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Service;

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						78
Змн.	Арк.	№ докум.	Підпис	Дата		

```

import ztu.edu.ua.projectmanagementserverapp.auth.service.JwtService;

import java.security.Key;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

@Log4j2
@Service
public class JwtServiceImpl implements JwtService {

    @Value("${token.signing.key}")
    private String jwtSigningKey;

    @Override
    public String extractUserName(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    @Override
    public String extractUserNameByBearerToken(String token) {
        try {
            return extractClaim(token.substring(7), Claims::getSubject);
        } catch (Exception e) {
            log.error("Token validation failed. Reason: [{}].",
e.getMessage());
            return null;
        }
    }

    @Override
    public String generateToken(UserDetails userDetails) {
        return generateToken(new HashMap<>(), userDetails);
    }

    @Override
    public boolean isTokenValid(String token, UserDetails userDetails) {
        final String userName = extractUserName(token);
        return (userName.equals(userDetails.getUsername())) &&
!isTokenExpired(token);
    }

    private <T> T extractClaim(String token, Function<Claims, T>
claimsResolvers) {
        final Claims claims = extractAllClaims(token);
        return claimsResolvers.apply(claims);
    }

    private String generateToken(Map<String, Object> extraClaims, UserDetails
userDetails) {
        return
JwtBuilder().setClaims(extraClaims).setSubject(userDetails.getUsername())
.setIssuedAt(new Date(System.currentTimeMillis()))
.setExpiration(new Date(System.currentTimeMillis() + 1000 *
60 * 60 * 24))
.signWith(getSigningKey(),
SignatureAlgorithm.HS256).compact();
    }

    private boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }
}

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						79
Змн.	Арк.	№ докум.	Підпис	Дата		

```

private Date extractExpiration(String token) {
    return extractClaim(token, Claims::getExpiration);
}

private Claims extractAllClaims(String token) {
    return
Jwts.parserBuilder().setSigningKey(getSigningKey()).build().parseClaimsJws(to
ken)
        .getBody();
}

private Key getSigningKey() {
    byte[] keyBytes = Decoders.BASE64.decode(jwtSigningKey);
    return Keys.hmacShaKeyFor(keyBytes);
}
}

```

					ІПЗ.КР.Б – 121 – 24 – ІЗ	Арк.
						80
Змн.	Арк.	№ докум.	Підпис	Дата		