МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА СИСТЕМИ МЕНЕДЖМЕНТУ ПРОЄКТНИХ ЗАВДАНЬ ТА ПОМИЛОК З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ REACT»**

Виконав: студент 2 курсу, групи 8.1213-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
В.В. Кістрінь
(ініціали та прізвище)
Керівник професор кафедри комп'ютерних наук, професор, д.т.н. Чопоров С.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри фундаментальної та прикладної математики, доцент, к.ф.-м.н. Панасенко Є.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Кістріню Владиславу Віталійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка системи менеджменту проєктних завдань та помилок з використанням технології React

керівник роботи Чопоров Сергій Вікторович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 25.11.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Проєктування програмного забезпечення.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

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

7. Дата видачі завдання 07.03.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	18.03.2024	
2.	Збір вихідних даних.	22.04.2024	
3.	Обробка методичних та теоретичних	27.05.2024	
	джерел.		
4.	Розробка першого та другого розділу.	29.08.2024	
5.	Розробка третього розділу.	25.10.2024	
6.	Оформлення та нормоконтроль	20.11.2024	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	12.12.2024	

Студент _____ В.В. Кістрінь
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.В. Чопоров
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка системи менеджменту проєктних завдань та помилок з використанням технології React»: 50 с., 23 рис., 1 табл., 30 джерел.

DOCKER, EXPRESS, MONGODB, NODE.JS, REACT.JS, REST API.

Об’єкт дослідження – програмні системи для управління проєктними завданнями та помилками.

Мета роботи: системи менеджменту проєктних завдань та помилок.

Метод дослідження – аналіз вимог до програмного забезпечення, моделювання та проєктування архітектури програмного забезпечення.

У кваліфікаційній роботі розглянуто сучасні технології розробки вебдодатків, зокрема React, Node.js та MongoDB. Проведено порівняння різних підходів до управління проєктами та відстеження помилок. Розроблена система дозволяє значно покращити організацію робочих процесів, підвищити продуктивність команди та забезпечити якість кінцевого продукту.

Результатом виконання кваліфікаційної роботи є система менеджменту проєктних завдань та помилок з використанням технології React та Node.js.

SUMMARY

Master’s qualification paper “Development of a Project Task and Error Management System Using React Technology”: 50 pages, 23 figures, 1 table, 30 references.

DOCKER, EXPRESS, MONGODB, NODE.JS, REACT.JS, REST API.

Object of research: software systems for project task and bug management.

Aim of the work: to develop a project task and bug management system.

Research method: analysis of software requirements, modeling, and design of software architecture.

The qualification paper examines modern technologies for web application development, specifically React, Node.js, and MongoDB. Various approaches to project management and bug tracking are compared. The developed system significantly improves the organization of workflows, enhances team productivity, and ensures the quality of the final product.

The result of the qualification paper is a project task and bug management system developed using React and Node.js technologies.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 8
1 Аналіз предметної області та об’єкти дослідження 10
1.1 Аналіз предметної області 10
1.2 Методології та інструменти управління проєктними 10
1.2.1 Методології управління проєктами 11
1.2.2 Технологічні інструменти 11
1.3 Потреби та вимоги до систем управління завданнями і помилками . 12
1.3.1 Основні концепції та принципи управління 13
1.4 Огляд та порівняльний аналіз альтернативних застосунків..... 14
1.5 Обґрунтування обраного стеку технологій 18
1.5.1 React..... 18
1.5.2 Огляд платформи Node.js..... 20
1.5.3 MongoDB..... 21
1.5.4 Express 22
1.5.5 Огляд інструменту контейнеризації Docker..... 23
1.6 Висновки до розділу 1 24
2 Проєктний аналіз..... 25
2.1 Призначення розробки..... 25
2.2 Вимоги до програмного продукту..... 25
2.2.1 Визначення основних компонентів користувацького інтерфейсу
..... 25
2.2.2 Функціональні вимоги..... 26
2.2.3 Технічні вимоги..... 27
2.3 Діаграма використання..... 28

2.4 Діаграма послідовності 30
2.5 Структура документа для управління проєктами в MongoDB 32
2.6 Опис обраної методики розробки 33
2.7 Висновки до розділу 2 33
3 Реалізація та тестування 34
3.1 Встановлення середовища розробки 34
3.2 Клієнтська частина вебдодатку 36
3.3 Серверна частина вебдодатку 41
3.4 Тестування та відлагодження системи 45
3.5 Висновки до розділу 3 46
Висновки 47
Перелік посилань 48

ВСТУП

Актуальність теми. Сучасний розвиток інформаційних технологій, зокрема у сфері веброзробки, значно підвищує ефективність управління проєктами та контролю якості програмного забезпечення. Однією з важливих задач в цій галузі є ефективне управління проєктними завданнями та відстеження помилок. Реалізація цієї задачі дозволяє значно покращити організацію робочих процесів, підвищити продуктивність команди та забезпечити якість кінцевого продукту.

Існуючі інструменти для управління проєктами, такі як JIRA, Trello та Asana, мають свої переваги, але не завжди задовольняють специфічні потреби окремих команд. В умовах постійного розвитку технологій виникає потреба у створенні нових, більш адаптивних та зручних рішень, які б забезпечували гнучкість, масштабованість та простоту використання.

Однією з перспективних технологій для створення таких рішень є React – бібліотека для побудови користувацьких інтерфейсів, що дозволяє створювати швидкі та інтерактивні вебдодатки. Використання React у поєднанні з іншими сучасними технологіями, такими як Node.js для бекенду, Express.js як ефективний та гнучкий вебфреймворк, а також MongoDB для зберігання даних, дозволяє створювати потужні та масштабовані системи менеджменту проєктних завдань та помилок.

Таким чином, актуальним є створення системи менеджменту проєктних завдань та помилок з використанням технології React, яка б забезпечувала високу продуктивність, зручність у використанні та відповідала сучасним вимогам до програмного забезпечення.

Мета і завдання дослідження. Метою дослідження є розробка системи менеджменту проєктних завдань та помилок з використанням технології React, яка забезпечить ефективне управління та контроль якості програмного забезпечення.

Об’єкт дослідження. Об’єктом дослідження є програмні системи для управління проєктними завданнями та помилками.

Предмет дослідження. Предметом дослідження є методи та технології розробки систем менеджменту проєктних завдань та помилок з використанням технології React.

Методи дослідження. Для вирішення поставлених завдань використовуються наступні методи дослідження:

- аналіз літератури та існуючих рішень у галузі управління проєктами та відстеженя помилок;
- вивчення та порівняння різних підходів до розробки програмних систем з використанням React;
- розробка архітектури системи та створення UML-діаграм;
- моделювання структури бази даних за допомогою ER-діаграм;
- реалізація та тестування системи на практиці.

Наукова новизна одержаних результатів полягає у створенні системи менеджменту проєктних завдань та помилок, що використовує сучасні технології React та Node.js, забезпечуючи високу продуктивність та зручність у використанні. Розроблена система включає інноваційні підходи до управління проєктами, що дозволяє підвищити ефективність робочих процесів.

Практичне значення одержаних результатів дослідження полягає у створенні функціональної та зручної системи для управління проєктними завданнями та помилками. Розроблена система може бути використана в різних галузях для підвищення ефективності управління проєктами та забезпечення високої якості програмного забезпечення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБ'ЄКТИ ДОСЛІДЖЕННЯ

1.1 Аналіз предметної області

Системи менеджменту проєктних завдань та помилок є ключовими інструментами для організації, управління та контролю за розробкою програмного забезпечення. Вони забезпечують платформу для відстеження і управління різними етапами роботи над проєктом, зокрема розробкою, тестуванням, документуванням та підтримкою. Оскільки програмні продукти стають дедалі масштабнішими та складнішими, попит на системи для відстеження завдань і помилок стабільно зростає. Це обумовлює необхідність пошуку оптимальних рішень для управління завданнями в умовах швидкої зміни вимог та значної кількості учасників проєктних команд.

1.2 Методології та інструменти управління проєктними

Наукові дослідження та практичні підходи в галузі менеджменту проєктних завдань зосереджуються на вивченні методів, інструментів та методологій, які сприяють підвищенню ефективності та продуктивності команд розробників. Серед них виділяються різні методології управління проєктами та технологічні інструменти [1].

Водоспадна модель: ця традиційна методологія передбачає послідовне виконання етапів проєкту, де кожен етап завершується перед початком наступного.

Гнучкі методології: такі як Scrum і Kanban, акцентують увагу на ітеративному підході, де завдання поділяються на менші, керовані частини, а робота над проєктом постійно вдосконалюється та адаптується.

Гібридні методології: поєднують елементи водоспадної та гнучких моделей, щоб створити більш гнучкий підхід, який відповідає конкретним потребам проєкту.

До програмного забезпечення для управління проєктами відносяться такі інструменти, такі як Asana, Trello та Jira, допомагають командам планувати, організовувати та відстежувати завдання проєкту.

Інструменти для спільного чату та співпраці такі як Slack, Microsoft Teams та Zoom.

Інструменти, такі як Google Drive, Dropbox та Box, допомагають командам зберігати та спільно використовувати проєктні документи [2].

1.2.1 Методології управління проєктами

Одним із ключових аспектів досліджень є вивчення методологій управління проєктами, таких як Agile, Scrum, Kanban та інші. Дослідження вказують на те, що використання Agile-методологій призводить до підвищення гнучкості та адаптивності команд, що дозволяє швидше реагувати на зміни в вимогах клієнтів та ринку. Scrum, як одна з найпоширеніших Agile-методологій, дозволяє командам організовувати свою роботу в короткі спринти, що сприяє регулярній перевірці результатів та внесенню коригувань. Kanban акцентує увагу на візуалізації всього робочого процесу за допомогою карток на дошці, що підходить для команд, що прагнуть та бажають зменшити кількість одночасно виконуваних завдань [3].

1.2.2 Технологічні інструменти

Інший напрям досліджень фокусується на технологічних інструментах, які підтримують управління проєктами. Зокрема, дослідження показують, що

інтеграція інструментів для управління завданнями, версіями коду та комунікацією може значно підвищити продуктивність команди. Наприклад, інструменти, такі як Jira, Trello та Asana, дозволяють командам ефективно відстежувати прогрес завдань, координувати дії та забезпечувати прозорість у процесі розробки [4].

1.3 Потреби та вимоги до систем управління завданнями і помилками

Системи управління завданнями і помилками відіграють важливу роль у забезпеченні ефективної роботи команд над проєктами різної складності. Їх основною метою є організація злагодженої комунікації, підвищення продуктивності команди, зменшення часу на виправлення помилок і покращення контролю якості. Для досягнення цих цілей системи повинні відповідати певним потребам і вимогам [5].

Централізоване зберігання інформації про завдання і помилки дозволяє членам команди швидко і зручно отримувати доступ до всіх необхідних даних, що мінімізує дублювання інформації і втрати при передачі даних. Прозорість і контроль прогресу є не менш важливими аспектами – система повинна надавати можливість відстеження статусу виконання завдань у реальному часі, що допомагає керівникам проєктів контролювати поточний стан роботи і краще розуміти етапи просування проєкту. Також важливою є можливість оптимізації командної роботи та комунікації, оскільки система має полегшувати співпрацю, дозволяючи учасникам команди взаємодіяти, обговорювати завдання, призначати відповідальних і залишати коментарі [5].

Система повинна забезпечувати можливість створення і редагування завдань та помилок із зазначенням ключових деталей, а також підтримувати відстеження життєвого циклу завдань та помилок на різних етапах, таких як «нове», «в роботі», «тестування» і «завершено». Це дозволяє чітко

відслідковувати етапи виконання. Пріоритизація завдань також є критично важливою, оскільки вона дає команді змогу концентруватися на найбільш значущих завданнях і вчасно вирішувати критичні помилки. Наявність сповіщень і нагадувань допомагає уникати затримок та пропусків, надаючи можливість вчасно реагувати на дедлайни, оновлення статусів або важливі коментарі.

Крім того, важливою вимогою є генерування звітів і аналітики щодо виконаної роботи, обсягу виконаних завдань і часу, витраченого на вирішення помилок. Це допомагає керівникам проєктів оцінювати продуктивність команди і приймати обґрунтовані рішення. Інтеграція з іншими інструментами, такими як системи контролю версій Git, месенджери та календарі.

Серед нефункціональних вимог виділяються швидкодія і стабільність, що дозволяють системі залишатися продуктивною навіть за високих навантажень і великої кількості даних. Зручність користування через інтуїтивний інтерфейс і продуманий UX значно підвищує ефективність роботи команди. Безпека і конфіденційність також є пріоритетними – система повинна забезпечувати захист від несанкціонованого доступу і підтримувати різні рівні доступу для учасників команди [5].

1.3.1 Основні концепції та принципи управління

Основні концепції спрямовані на забезпечення ефективного виконання проєкту, підвищення якості продукту та оптимізацію роботи команди. Ці концепції охоплюють підхід до постановки цілей, розподілу завдань, моніторингу процесів, а також виявлення й усунення помилок.

Прозорість є ключовою концепцією в управлінні завданнями і помилками. Кожен учасник проєкту повинен мати доступ до актуальної

інформації про стан завдань і виявлені помилки. Пріоритизація завдань і помилок допомагає визначити, які з них є критичними і потребують негайної уваги, а які можна відкласти на пізніший термін. Вона базується на таких факторах, як терміни виконання, важливість для клієнта, вплив на роботу системи та ризики для проєкту що допомагає команді зосередитися на найбільш важливих аспектах проєкту відклавши дрібні помилки на потім.

Ефективне управління потребує постійного моніторингу прогресу команди та регулярного зворотного зв’язку. Такий підхід дозволяє швидко виявляти відхилення від запланованих результатів і своєчасно вносити корективи. Цей підхід передбачає регулярне оновлення та коригування планів, аналіз результатів після кожної ітерації та впровадження змін для досягнення кращих результатів [6].

Основні концепції та принципи зосереджуються на забезпеченні прозорості, швидкості реагування на зміни, пріоритизації завдань і безперервному вдосконаленні процесів. Вони формують надійну основу для успішної реалізації проєкту, забезпечуючи максимальну ефективність команди та високу якість кінцевого продукту.

1.4 Огляд та порівняльний аналіз альтернативних застосунків

Серед існуючих рішень виділяються такі системи, як Jira, Trello, Asana, та GitHub Issues. Кожна з цих систем має свої особливості та застосовується в різних контекстах.

Jira (див. рис. 1.1) є одним із найпотужніших інструментів для управління проєктами та задачами, який спеціально розроблено для команд, що працюють у сфері розробки програмного забезпечення. Вона пропонує розширений набір функцій, який дозволяє командам ефективно організовувати свою роботу. Серед ключових можливостей Jira варто виділити

створення та управління задачами, що включає визначення пріоритетів, встановлення дедлайнів і призначення відповідальних осіб. Також система підтримує відстеження витрат часу на виконання завдань, що сприяє точному аналізу продуктивності команди [7].

Jira активно використовується як розробниками програмного забезпечення, так і іншими командами, що працюють у сфері IT. Вона є універсальним рішенням для планування, тестування, контролю за процесами розробки та організації випуску продуктів, що робить її популярним вибором серед компаній у всьому світі.

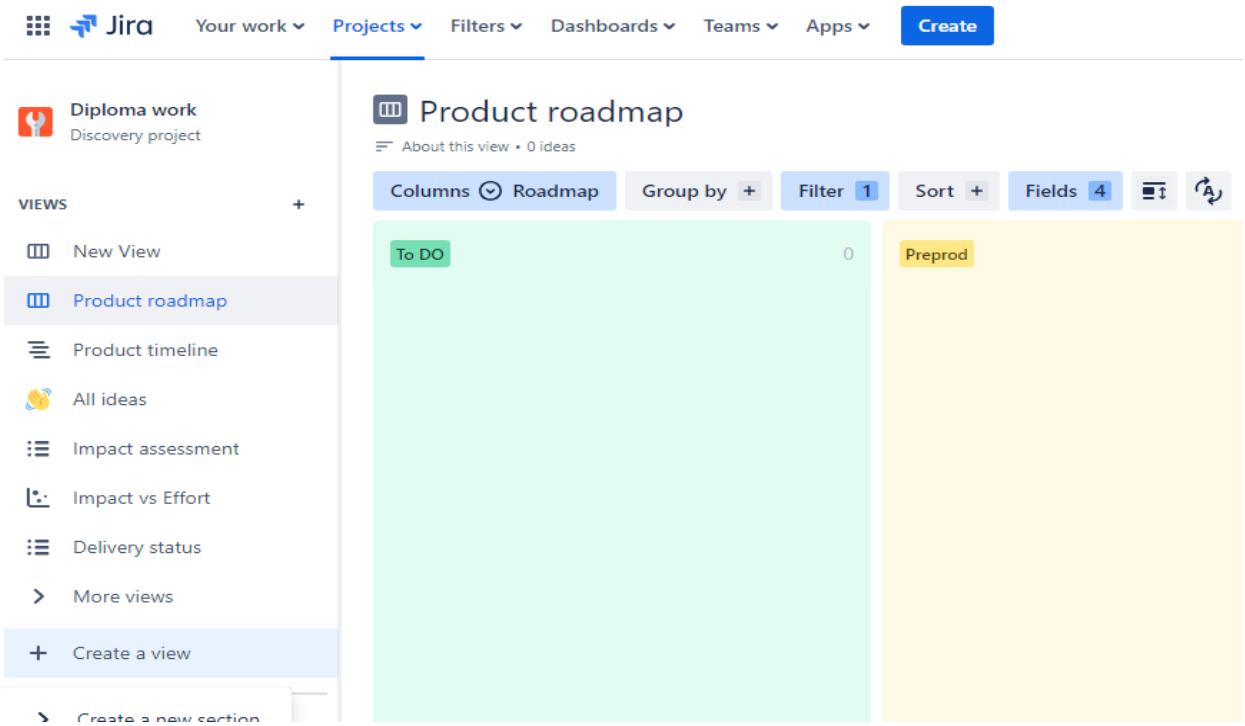


Рисунок 1.1 – Застосунок Jira

Trello (див. рис. 1.2) – це візуальний інструмент для керування проєктами, який пропонує користувачам використовувати дошки, списки та картки для організації своїх завдань. Він відомий своєю простотою використання та можливістю інтеграції з іншими сервісами додатками й службами, наприклад Slack, Outlook, Gmail, Salesforce, InVision, Jira. Trello часто використовується як особистий організатор завдань, а також для невеликих команд та проєктів [8].

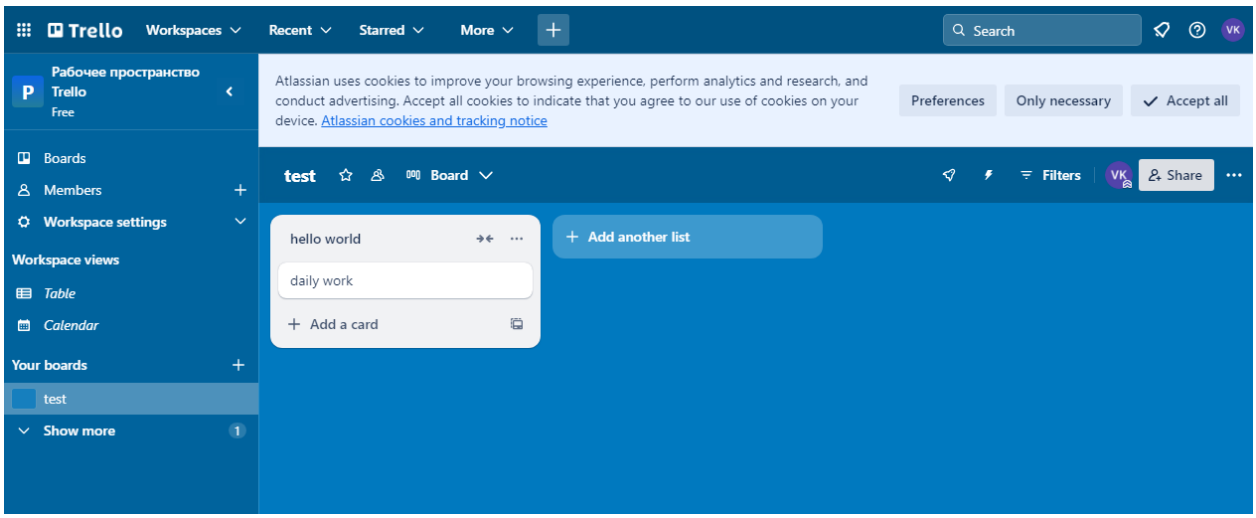


Рисунок 1.2 – Застосунок Trello

Asana (див. рис. 1.3) – це інструмент для керування проєктами, який дозволяє створювати завдання, проєкти, графіки та багато іншого. Він також надає засоби для спільної роботи та комунікації в команді. Asana використовується для різних видів проєктів та команд, від маркетингу та реклами до розробки програмного забезпечення [9].

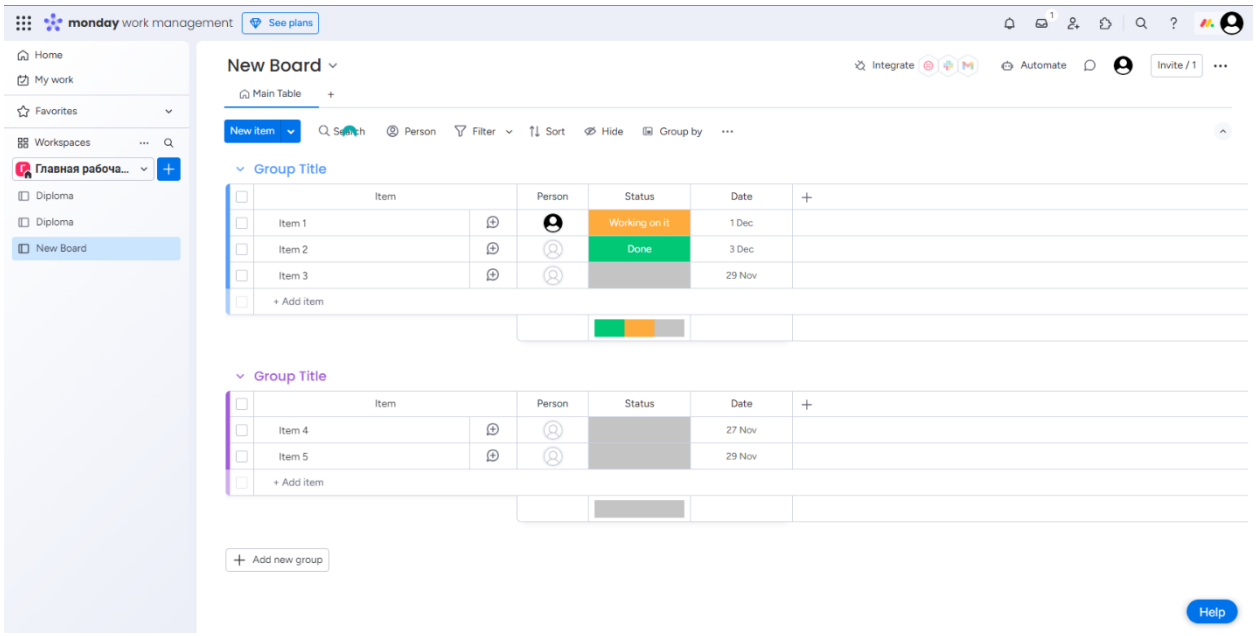


Рисунок 1.3 – Застосунок Asana

GitHub Issues (див. рис. 1.4) – це застосунок від GitHub, яка дозволяє користувачам створювати та відстежувати проблеми, завдання та запити на внесення змін у проєкти. GitHub Issues часто використовується розробниками

програмного забезпечення для спільної роботи над відкритими джерелами проєктами та управління задачами в рамках розробки програмного забезпечення [10].

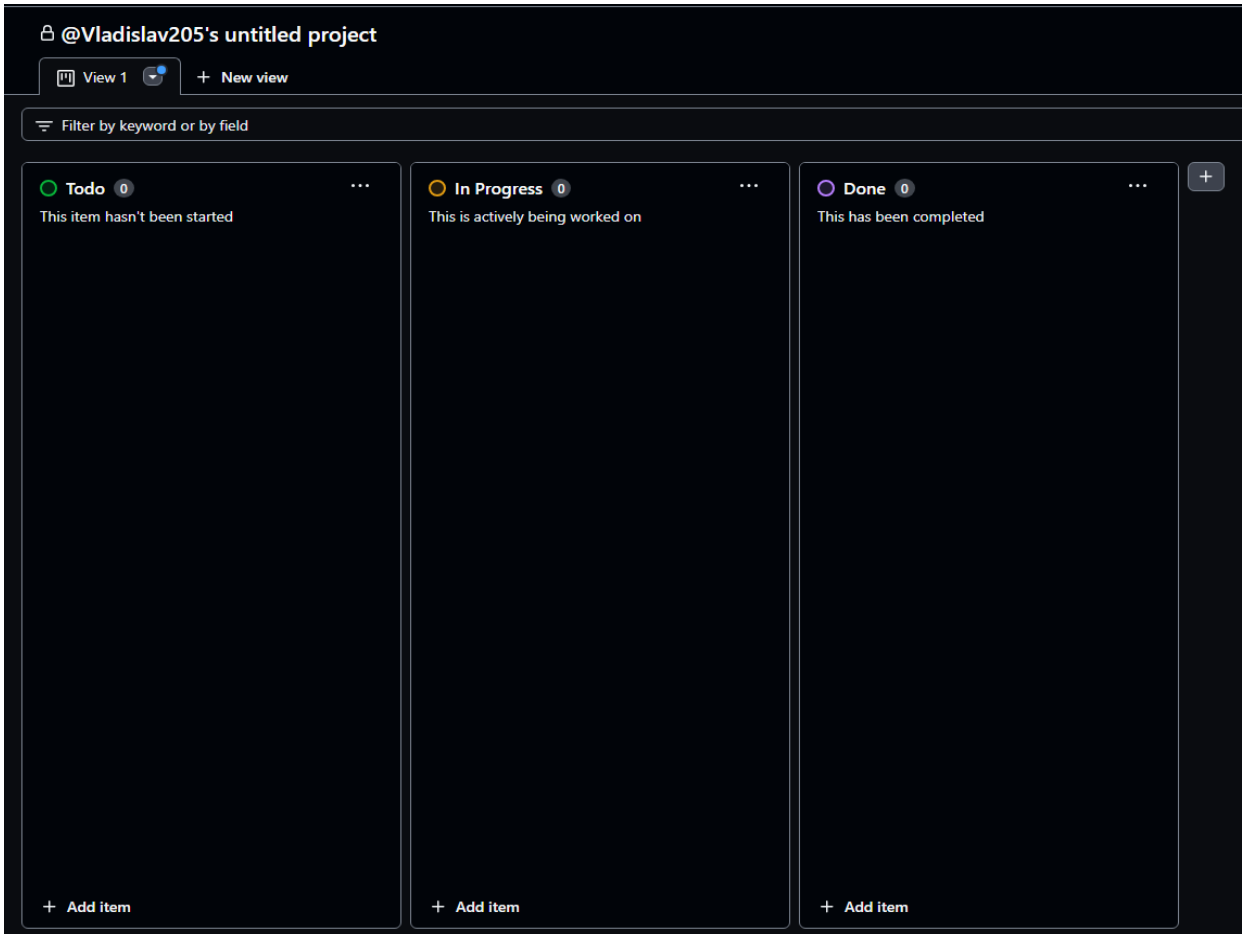


Рисунок 1.4 – Застосунок GitHub Issues

В таблиці 1.1 наведені порівняльні характеристики різних підходів та інструментів.

Таблиця 1.1 – Порівняння підходів та інструментів

Застосунок	**Методології**	**Переваги**	**Недоліки**
Jira	Kanban, Scrum	Потужний функціонал, гнучкі налаштування, інтеграція з іншими інструментами	Висока вартість для великих команд

Продовження табл. 1.1

Застосунок	Методології	Переваги	Недоліки
Trello	Kanban	Простота використання, візуальна привабливість	Обмежений функціонал для великих проєктів
Asana	Kanban, Scrum	Інтуїтивний інтерфейс, можливість управління великими проєктами	Може бути занадто складним для малих команд
GitHub Issues	Custom	Інтеграція з GitHub, простота використання	Обмежені можливості для великих проєктів

Кожна з цих систем має свої переваги і недоліки, тому вибір залежить від конкретних потреб та вимог проєкту чи команди.

1.5 Обґрунтування обраного стеку технологій

1.5.1 React

React.js – це гнучка JavaScript бібліотека, яка призначена для створення динамічних та інтерактивних користувацьких інтерфейсів. Вона була розроблена з метою ефективного часткового оновлення вебсторінок, дозволяючи додавати стільки React, скільки потрібно для конкретного застосування. Це забезпечує високу продуктивність та гнучкість при створенні складних інтерфейсів. Основною перевагою React є його здатність ефективно працювати з оновленням лише тих частин сторінки, які зазнали змін, що значно підвищує продуктивність вебдодатків, особливо на великих проєктах. Вона дозволяє розробникам створювати компоненти – незалежні блоки коду, які можна знову використовувати в різних частинах додатка. Завдяки такій

модульності та компонентному підходу, розробка та підтримка програмного забезпечення стають значно простішими [11].

JSX – це спеціалізований синтаксис, який використовується для опису інтерфейсів у React. Він поєднує в собі можливості HTML і JavaScript, дозволяючи створювати структури інтерфейсу прямо в коді без потреби у додаткових шаблонізаторах. Завдяки цьому розробники можуть швидко і ефективно працювати з елементами інтерфейсу, роблячи код більш інтуїтивно зрозумілим [12].

Порівнюючи React з іншими популярними фреймворками, такими як Angular чи Vue.js, можна зазначити, що React надає більше гнучкості в виборі інструментів для створення проєктів, у той час як Angular є більш структурованим фреймворком з готовими рішеннями для кожної задачі. У порівнянні з Vue.js, React пропонує більше можливостей для кастомізації, хоча останній вважається легшим для початкового освоєння завдяки більш вбудованій екосистемі. Однією з основних переваг React є також велика спільнота розробників і величезна кількість готових бібліотек, що значно прискорює процес розробки [13].

Має більш пологу криву навчання завдяки простій концепції компонентів та JSX, в той час як Angular вимагає глибоких знань TypeScript та додаткових концепцій.

Окрім основної бібліотеки React, для покращення функціональності та зручності розробки часто використовуються популярні бібліотеки. Розглянемо деякі з них.

React router – одна з найпопулярніших бібліотек для маршрутизації в React. Вона дозволяє легко керувати переходами між сторінками додатку, що дає змогу створювати динамічні та багаторазові компоненти з можливістю зміни URL без перезавантаження сторінки.

Redux – бібліотека для централізованого управління станом додатку. Вона забезпечує зберігання всього стану в одному глобальному сховищі, що дозволяє ефективно керувати даними в складних додатках, де стан може змінюватися в різних компонентах.

React-query – бібліотека для роботи з асинхронними запитами та даними. Вона забезпечує автоматичне кешування, синхронізацію і оновлення даних, що знижує навантаження на сервер і підвищує ефективність роботи з API.

Axios – популярна бібліотека для відправки HTTP-запитів. Axios забезпечує простоту у використанні для взаємодії з RESTful API, з підтримкою асинхронних запитів та перехоплення помилок [14].

Styled components – бібліотека для створення стилів безпосередньо в JavaScript-коді. Вона дозволяє застосовувати CSS до окремих компонентів, що сприяє кращій організації стилів, забезпечує їх динамічність і зручність у масштабованих додатках.

Material-UI (MUI) – бібліотека компонентів, заснована на принципах Material Design. Вона включає в себе готові до використання компоненти, що дозволяє швидко створювати привабливі інтерфейси з мінімальними зусиллями.

Formik – одна з найбільш популярних бібліотек для роботи з формами в React. Вона допомагає з управлінням станом форм, валідацією даних та обробкою подій, спрощуючи розробку складних форм з мінімальними зусиллями.

React spring – бібліотека для створення анімацій у React-додатках. Вона використовує фізичні моделі для надання анімаціям більш природних рухів, що значно підвищує зручність використання додатків.

Використання бібліотек у поєднанні з основною бібліотекою React дозволяє розробникам значно прискорити процес створення додатків, забезпечуючи високий рівень ефективності, гнучкості та масштабованості.

1.5.2 Огляд платформи Node.js

Використання Node.js забезпечує високу продуктивність та швидкість обробки запитів завдяки своїй подійній моделі та неблокуючим операціям

вводу/виводу. Node.js дозволяє виконувати JavaScript-код на сервері, що забезпечує єдність мови програмування як на фронтенді, так і на бекенді. Це спрощує обмін кодом та забезпечує узгодженість в дизайні та логіці програми. Крім того, Node.js має велику та активну спільноту розробників, що дозволяє використовувати різноманітні розширення та пакети для швидкої розробки та отримання підтримки [15].

Порівнюючи Node.js з іншими серверними технологіями, такими як PHP чи Python, можна побачити, що Node.js забезпечує вищу ефективність завдяки асинхронній обробці запитів. Наприклад, в той час як PHP кожен запит обробляє окремо, Node.js дозволяє обробляти декілька запитів одночасно, не блокуючи виконання програми. Порівняно з Python і його фреймворком Django, Node.js здатен обробляти більшу кількість запитів при меншому навантаженні на систему [16].

Node.js підходить для масштабованих і швидких додатків, таких як чати, ігри в реальному часі, а також для розробки API для мобільних додатків. Важливою перевагою є також велика кількість модулів, доступних через npm, що дозволяє швидко інтегрувати нові функції в проєкт.

1.5.3 MongoDB

MongoDB є нереляційною базою даних, яка використовує документи в форматі BSON для зберігання даних. Ця база даних ідеально підходить для застосунків, де структура даних часто змінюється або не є фіксованою, таких як проєкти з великим обсягом неструктурованих даних. Оскільки MongoDB дозволяє змінювати структуру даних без необхідності переписувати всю схему, це забезпечує велику гнучкість у роботі з різноманітними типами інформації [17].

MongoDB відома своєю здатністю до горизонтального масштабування за допомогою шардінгу, що дозволяє розподіляти дані між кількома серверами для забезпечення високої доступності та продуктивності навіть при великих

навантаженнях. Завдяки використанню JSON-подібних документів MongoDB оптимально підходить для роботи з вебдодатками, оскільки дані в такому форматі легко інтегруються з іншими системами.

Порівняно з реляційними базами даних, такими як MySQL або PostgreSQL, MongoDB має перевагу в плані масштабування та гнучкості. Вона не потребує чітко визначеної схеми і дозволяє працювати з великими обсягами змінюваних даних без складних налаштувань. Також MongoDB підтримує високу швидкість доступу до даних, що робить її оптимальним вибором для додатків, де потрібно працювати з великими масивами інформації [18].

1.5.4 Express

Express.js є фреймворком для Node.js, який забезпечує швидку розробку серверної частини вебдодатків. Його простий API дозволяє легко створювати RESTful API та обробляти HTTP-запити, що робить його ідеальним інструментом для створення сервісів, які повинні обробляти великий обсяг запитів за короткий час. Основною перевагою Express є гнучкість – він дозволяє вибирати підходи до організації коду та підключати додаткові middleware для конкретних задач.

У порівнянні з іншими фреймворками для Node.js, такими як Koa або Hapi, Express відзначається простотою у використанні та великою кількістю готових до застосування модулів, що спрощує створення складних вебдодатків. Наприклад, Koa є більш мінімалістичним фреймворком, що вимагає більше налаштувань, а Hapi надає більше можливостей для налаштування, але може бути складнішим для початківців.

Завдяки своїй модульній архітектурі, Express дозволяє легко масштабувати додатки і інтегрувати нові функції. Це робить його ідеальним вибором для проєктів, що потребують високої продуктивності та можливості швидко адаптуватися до змінних вимог [19].

1.5.5 Огляд інструменту контейнеризації Docker

Docker є одним із найпопулярніших інструментів для контейнеризації, який надає можливість створення, управління та розгортання контейнеризованих додатків. Контейнеризація дозволяє ізолювати додатки разом із усіма необхідними залежностями в стандартні пакети, які можуть працювати на будь-якому середовищі, що підтримує Docker. Це забезпечує однаковість роботи додатків незалежно від середовища, зменшує ризик помилок, пов’язаних з конфігурацією системи, та спрощує процес розгортання [20].

Docker працює на основі концепції контейнерів, які є легковаговими, автономними одиницями, що містять усе необхідне для виконання додатка: код, бібліотеки, конфігурації та середовище виконання. Відмінною рисою Docker є те, що контейнери запускаються поверх операційної системи хоста, спільно використовуючи її ядро, що робить їх менш ресурсомісткими порівняно з віртуальними машинами.

Docker використовує Docker Images – шаблони, які містять усі необхідні файли й налаштування для запуску контейнера. Ці образи можуть бути завантажені з Docker Hub – централізованого репозиторію, або створені власноруч за допомогою Dockerfile [20].

Docker є лідером серед інструментів контейнеризації завдяки своїй простоті використання, великій кількості документації та активній спільноті. Інші інструменти, такі як Podman або LXC, також пропонують можливості контейнеризації, але їх функціональність і зручність можуть бути обмеженими залежно від конкретного сценарію використання. Наприклад, Podman не потребує демонів для запуску контейнерів, що є перевагою у деяких середовищах, але Docker має більшу інтеграцію з популярними хмарними сервісами [21].

1.6 Висновки до розділу 1

У цьому розділі було проведено аналіз сучасного стану проблеми менеджменту проєктних завдань та помилок, розглянуто існуючі підходи та програмні рішення, а також зроблено порівняльний аналіз альтернативних підходів. На основі цього аналізу було визначено місце майбутньої розробки в системі існуючих рішень, підкреслено її новизну та актуальність. Крім того, обґрунтовано вибір стека технологій для розробки системи, що забезпечить високу продуктивність, модульність та інтеграцію з іншими інструментами.

2 ПРОЄКТНИЙ АНАЛІЗ

2.1 Призначення розробки

Розробка системи менеджменту проєктних завдань та помилок спрямована на автоматизацію та оптимізацію процесів управління завданнями в межах проєктів, а також на вдосконалення процесу моніторингу завдань та виправлення помилок. Система надасть можливість ефективно управляти завданнями, відстежувати їх виконання, швидко реагувати на проблеми, що виникають, та забезпечить прозорість процесів для всіх учасників проєкту.

Призначення цієї системи полягає у створенні зручного інструменту, який дозволить не лише організовувати й планувати завдання, а й забезпечити централізоване місце для реєстрації, відслідковування та виправлення помилок.

2.2 Вимоги до програмного продукту

2.2.1 Визначення основних компонентів користувацького інтерфейсу

Розглянемо основні компоненти користувацького інтерфейсу.

Панель навігації: компонент, який дозволяє виконувати навігацію по проєктам, режимах відображення та особистим кабінетом.

Dashboard: компонент який формує графіки, статистику та інформацію про продуктивність проєкту.

Backlog: компонент, що відображає зміни за певний проміжок часу, дозволяючи переглядати інформацію зміни в проєкті.

Архів: містяться відкладені, заблоковані та видалені баг-репорти певного проєкту з функціями відновлення та видалення з архіву.

Створення нового репорту: компонент, завдяки якому користувач може додавати нові репорти.

Інформація про користувача: зберігає дані про користувача, дату реєстрації, дату підключення та роль в проєкті.

Компонент завдання: компонент у вигляді popup-вікна, який дозволяє переглядати інформацію про обраний репорт, змінювати стан та ін.

2.2.2 Функціональні вимоги

Програмний продукт, що розробляється, повинен відповідати наступним функціональним вимогам:

- реєстрація користувачів: застосунок має дозволяти новим користувачам реєструватися, вказуючи свої облікові дані;
- аутентифікація користувачів: застосунок повинен забезпечувати безпечний вхід користувачів з використанням їхніх облікових даних;
- переглядати та редагувати свій профіль, включаючи зміну пароля та оновлення особистих даних;
- застосунок має підтримувати різні ролі (адміністратор, менеджер, розробник, тестувальник) з відповідними рівнями доступу;
- користувачі з відповідними правами повинні мати можливість створювати нові проєкти, вказуючи їхні назви, описи та інші метадані;
- менеджери проєктів повинні мати можливість редагувати деталі проєкту, додавати та видаляти учасників команди;
- переглядати список доступних їм проєктів та деталі кожного проєкту;
- користувачі з відповідними правами повинні мати можливість створювати нові завдання, вказуючи їхні назви, описи, пріоритети та терміни виконання;
- користувачі повинні мати можливість редагувати, переназначати та змінювати стан завдань;

- переглядати список завдань для кожного проєкту та деталі кожного завдання;
- створювати нові баг-репорти, вказуючи їхні описи, пріоритети та статуси;
- редагувати, переназначати та змінювати статуси баг-репортів;
- переглядати список баг-репортів для кожного проєкту та деталі кожного баг-репорту;
- додавати коментарі до завдань та баг-репортів;
- переглядати коментарі, додані до завдань та баг-репортів;
- користувачі повинні мати можливість редагувати або видаляти свої коментарі;
- користувачі повинні мати можливість архівувати завершені або відмінені завдання та баг-репорти;
- користувачі повинні мати можливість переглядати архівовані завдання та баг-репорти;
- користувачі повинні мати можливість відновлювати архівовані завдання та баг-репорти;
- переглядати список завдань та баг-репортів, що очікують на виконання, із зазначенням змін та відповідальних осіб;
- користувачі повинні отримувати повідомлення про зміни у завданнях та баг-репортах, які їх стосуються;
- користувачі повинні мати можливість налаштовувати, які повідомлення вони бажають отримувати (наприклад, електронною поштою або через інтерфейс застосунку).

2.2.3 Технічні вимоги

Програмний продукт, що розробляється, повинен мати наступні технічні вимоги:

– система повинна бути побудована за допомогою технологій, таких як React для фронтенд-розробки та Node.js для бекенд-розробки;
– підтримка браузерів, включаючи Google Chrome, Mozilla Firefox Microsoft Edge [22];
– використання RESTful API для забезпечення взаємодії між клієнтською та серверною частинами;
– для зберігання даних буде використовуватися нереляційна база даних MongoDB;
– база даних повинна підтримувати горизонтальне масштабування для ефективної роботи при зростанні обсягу даних.

2.3 Діаграма використання

UML-діаграма прецедентів є важливим інструментом для моделювання основних сценаріїв використання системи менеджменту проєктних завдань та помилок. Вона дозволяє чітко візуалізувати, як різні користувачі (актори) взаємодіють із функціональними можливостями системи (прецедентами), що побудована з використанням технологій React та Node.js. Такий підхід дає змогу краще зрозуміти вимоги до системи, а також розподілити функціональність між різними компонентами програми [23].

Графічне відображення цих взаємодій (див. рис. 2.1) дає змогу ідентифікувати ключові елементи системи та визначити пріоритетність їх реалізації.

Актори:

– адміністратор: користувач з повними правами доступу, який може керувати користувачами, проєктами, завданнями та помилками;
– менеджер проєкту: користувач, відповідальний за управління проєктами та завданнями, але з обмеженими правами в порівнянні з адміністратором;

– розробник: користувач, відповідальний за виконання завдань та обробку завдань;
– тестувальник: користувач, відповідальний за створення та обробку завдань.



Рисунок 2.1 – Діаграма прецедентів

Розглянемо основні **прецеденти**.

Управління користувачами:

– реєстрація користувача;
– авторизація користувача;
– редагування профілю користувача;
– видалення користувача.

Управління проєктами:

– створення проєкту;

– редагування проєкту;

– видалення проєкту;

– перегляд проєктів.

Управління завданнями:

– створення завдання;

– редагування завдання;

– видалення завдання;

– перегляд завдань;

– призначення завдання користувачу;

– зміна статусу завдання.

Створена діаграма використання допомагає визначити та структурувати функції, які має надавати система, з точки зору взаємодії з різними типами користувачів.

2.4 Діаграма послідовності

Діаграма послідовності ілюструє взаємодію користувача із системою під час створення та зміни статусу завдання на «виконано». Вона демонструє етапи обміну даними між користувачем та основними компонентами системи, забезпечуючи розуміння динаміки процесу [24].

Сценарій починається з того, що користувач вводить дані для створення нового завдання через інтерфейс платформи, включаючи опис, пріоритет та строк виконання (див. рис. 2.2). Інтерфейс надсилає запит до сервера управління завданнями з введеними даними. Сервер перевіряє коректність отриманої інформації та викликає модуль доступу до бази даних для збереження нового завдання. Після успішного збереження база даних надсилає підтвердження серверу, який у свою чергу повертає відповідь інтерфейсу. Користувач отримує повідомлення про успішне створення завдання.

На наступному етапі користувач виконує завдання і ініціює зміну його статусу через інтерфейс, обираючи опцію «виконано». Інтерфейс формує запит на зміну статусу завдання та передає його серверу. Сервер оновлює відповідний запис у базі даних, змінюючи статус завдання, та надсилає підтвердження оновлення назад до інтерфейсу. Користувач отримує повідомлення про успішну зміну статусу завдання на «виконано».

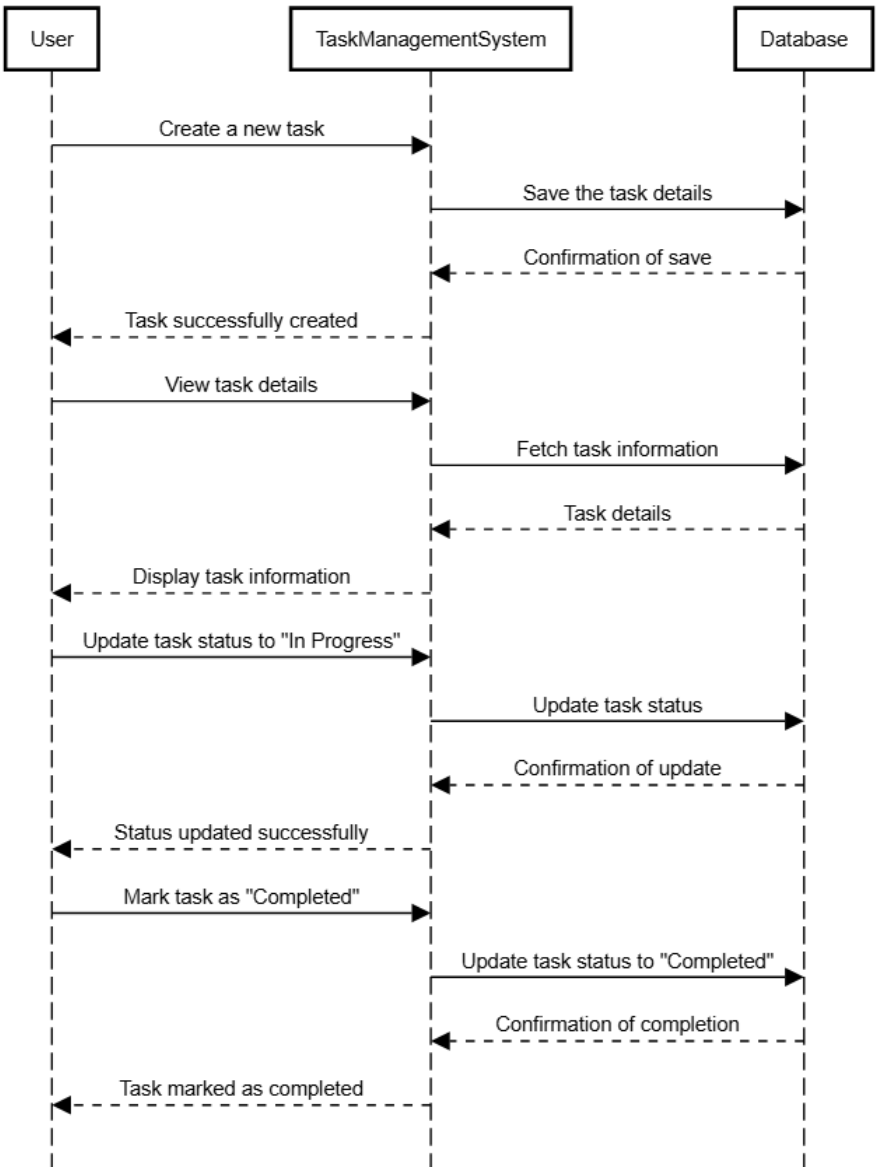


Рисунок 2.2 – Діаграма послідовності

Цей сценарій показує, як компоненти системи взаємодіють для забезпечення ключових функцій, що підвищує прозорість і точність процесу розробки.

2.5 Структура документа для управління проєктами в MongoDB

Нереляційні бази даних (NoSQL) відрізняються від реляційних тим, що не використовують традиційну модель таблиць і зв’язків. Вони забезпечують гнучкішу структуру даних, що дозволяє зберігати дані у різних форматах, таких як документи, графи, колонки чи ключ-значення. Основною перевагою NoSQL баз даних є легкість у зміні структури без необхідності модифікації всієї бази даних [25].

MongoDB є документною базою даних, яка зберігає дані у вигляді документів у форматі JSON. Нижче наведено структуру документа для системи менеджменту проєктних завдань та помилок (див. рис. 2.3) [25].

```
Projects
{_id: ObjectId,
 name: String,
 description: String,
 manager: {
  user_id: ObjectId, name: String},
 tasks: [{_id: ObjectId, name: String,
   description: String,
   status: String,
   responsible: {user_id: ObjectId,
    name: String}}]}
```

Рисунок 2.3 – Структура документа для MongoDB

Проєкт містить унікальний ідентифікатор, назву, опис, учасників проєкту і менеджера проєкту. Менеджер включає унікальний ідентифікатор користувача та його ім’я. Кожен проєкт має масив завдань. Завдання мають унікальний ідентифікатор, назву, опис, статус, особу яка призначила завдання та відповідальну особу. Відповідальна особа включає унікальний ідентифікатор користувача та його ім’я.

2.6 Опис обраної методики розробки

Даний застосунок для менеджменту проєктних завдань та помилок може значно покращити роботу з Agile методологією завдяки своїм функціональним можливостям і структурі.

Agile є однією з найпопулярніших методологій розробки програмного забезпечення, яка забезпечує гнучкий підхід до управління проєктами. Agile підхід дозволяє командам швидко реагувати на зміни, ефективно комунікувати та постійно вдосконалювати процес розробки. Процес розробки розбивається на короткі ітерації, кожна з яких завершується релізом робочого продукту. Кожна ітерація додає нову функціональність до продукту, забезпечуючи поступове зростання його можливостей. Постійна взаємодія з клієнтами та користувачами для врахування їх потреб та зворотного зв’язку. Команди, які працюють за Agile, мають високу автономність і відповідальність за результат [26].

2.7 Висновки до розділу 2

У цьому розділі було проведено проєктний аналіз розробки системи менеджменту проєктних завдань та помилок. Визначено основні сценарії використання системи, розроблено UML-діаграму прецедентів та структуру документу бази даних. Описано обрані технології та методики розробки, представлено алгоритми, функціональні вимоги та зроблені перші кроки до створення застосунку проєктуванням макету.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Встановлення середовища розробки

Першим етапом у створенні системи було налаштування середовища розробки, яке забезпечує зручність і ефективність роботи над проєктом. Цей процес включав вибір необхідних технологій, налаштування інструментів та перевірку їхньої взаємодії.

Для реалізації системи було обрано технології (див. рис. 3.1):

– node.js – для виконання JavaScript на стороні сервера, що забезпечує швидкість і масштабованість;
– express.js – для побудови серверної частини й обробки HTTP-запитів;
– react.js – для створення динамічних і зручних користувацьких інтерфейсів;
– mongoDB – база даних для зберігання даних задач, статусів та інформації про користувачів;
– docker – для контейнеризації додатку та спрощення його розгортання.



Рисунок 3.1 – Технології MERN

Для розробки було обрано останню стабільну версію Node.js. Встановлення виконувалось через офіційний сайт, а для перевірки коректності інсталяції використано команди node -v і npm –v (див. рис. 3.2).

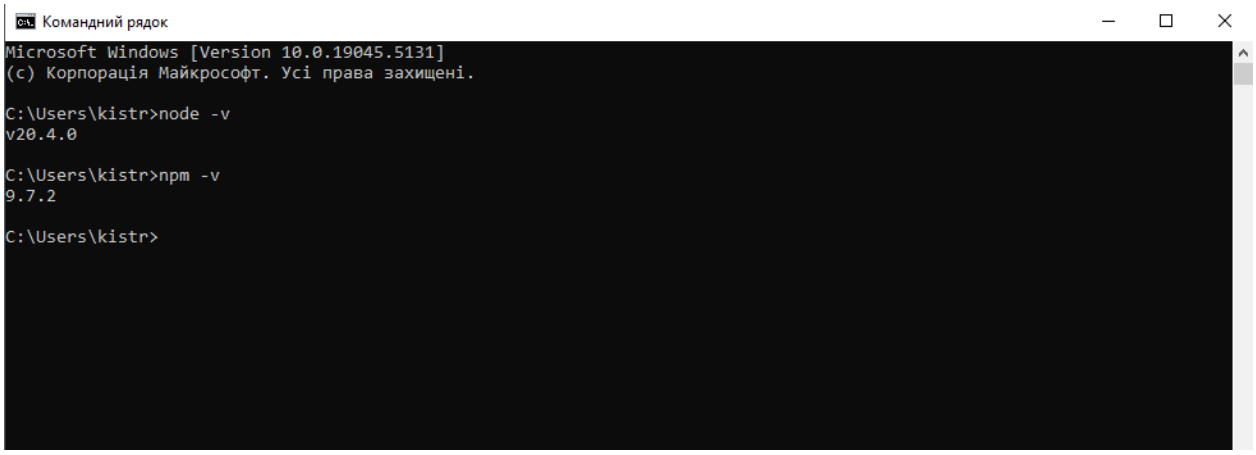


Рисунок 3.2 – Перевірка версій

Для клієнтської частини використано команду npx create-react-app, яка автоматично генерує базову структуру проєкту (див. рис. 3.3) [27].

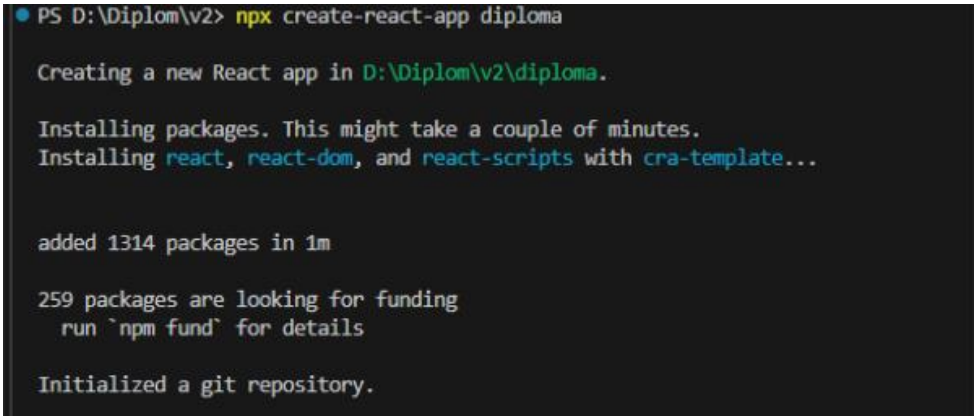


Рисунок 3.3 – Створення npm проєкту

На етапі налаштування додано базові залежності React, Redux і React Router для клієнтської частини та Express.js, Mongoose, bcrypt, jsonwebtoken для серверної частини (див. рис. 3.4).

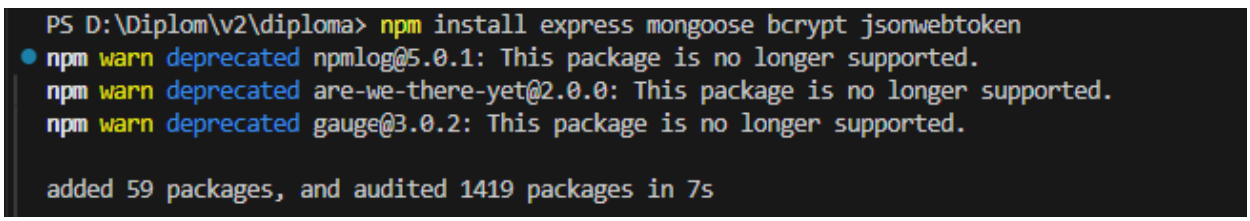


Рисунок 3.4 – Встановлення залежностей

Як основний редактор коду обрано Visual Studio Code (див. рис. 3.5).

Було додано необхідні плагіни:

– ESLint для перевірки якості коду;
– prettier для автоматичного форматування;
– docker для інтеграції роботи з контейнерами;
– mongoDB для взаємодії з базою даних безпосередньо у редакторі.

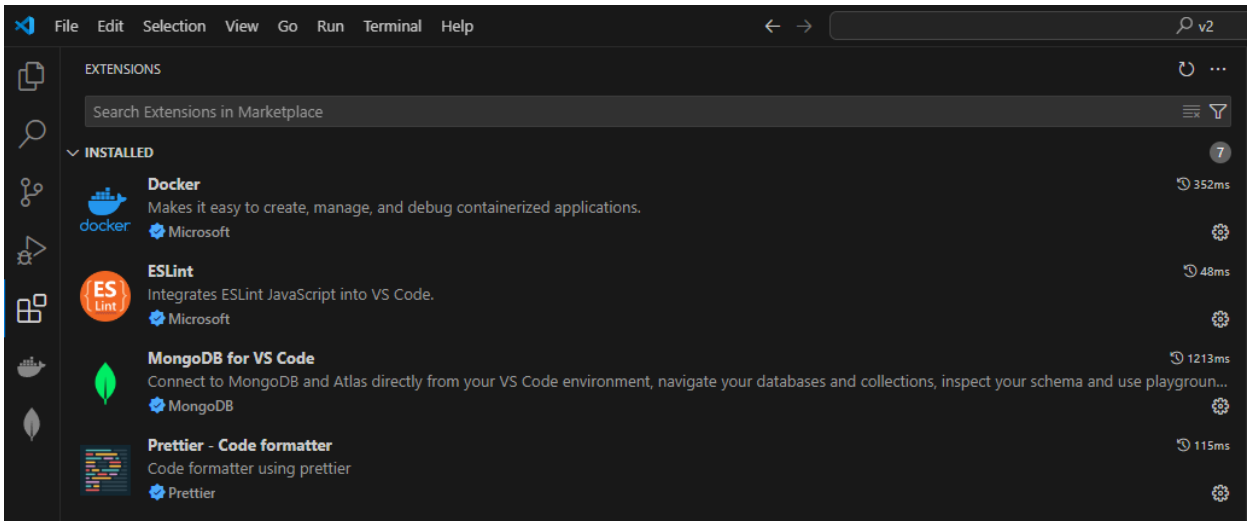


Рисунок 3.5 – Панель плагінів у Visual Studio Code

Таким чином, встановлення середовища створило стабільну та функціональну платформу для подальшої розробки програмного забезпечення.

3.2 Клієнтська частина вебдодатку

Клієнтська частина вебдодатку реалізована з використанням бібліотеки React.js, що дозволяє створювати інтерактивні інтерфейси з динамічним оновленням без перезавантаження сторінки. Це забезпечує безперервний досвід користувача, оскільки зміни в даних автоматично відображаються на сторінці без необхідності перезавантажувати весь контент. Вебдодаток складається з кількох основних сторінок і компонентів, які взаємодіють для забезпечення основних функцій системи [28].

Сторінка Overview надає загальний огляд стану проєкту, містить ключову інформацію, зокрема назву проєкту, поточний статус виконання завдань, список документів для проєкту та стрічку останніх змін і активностей. Це дозволяє користувачам швидко оцінити прогрес проєкту та отримати необхідну інформацію (див. рис. 3.6).

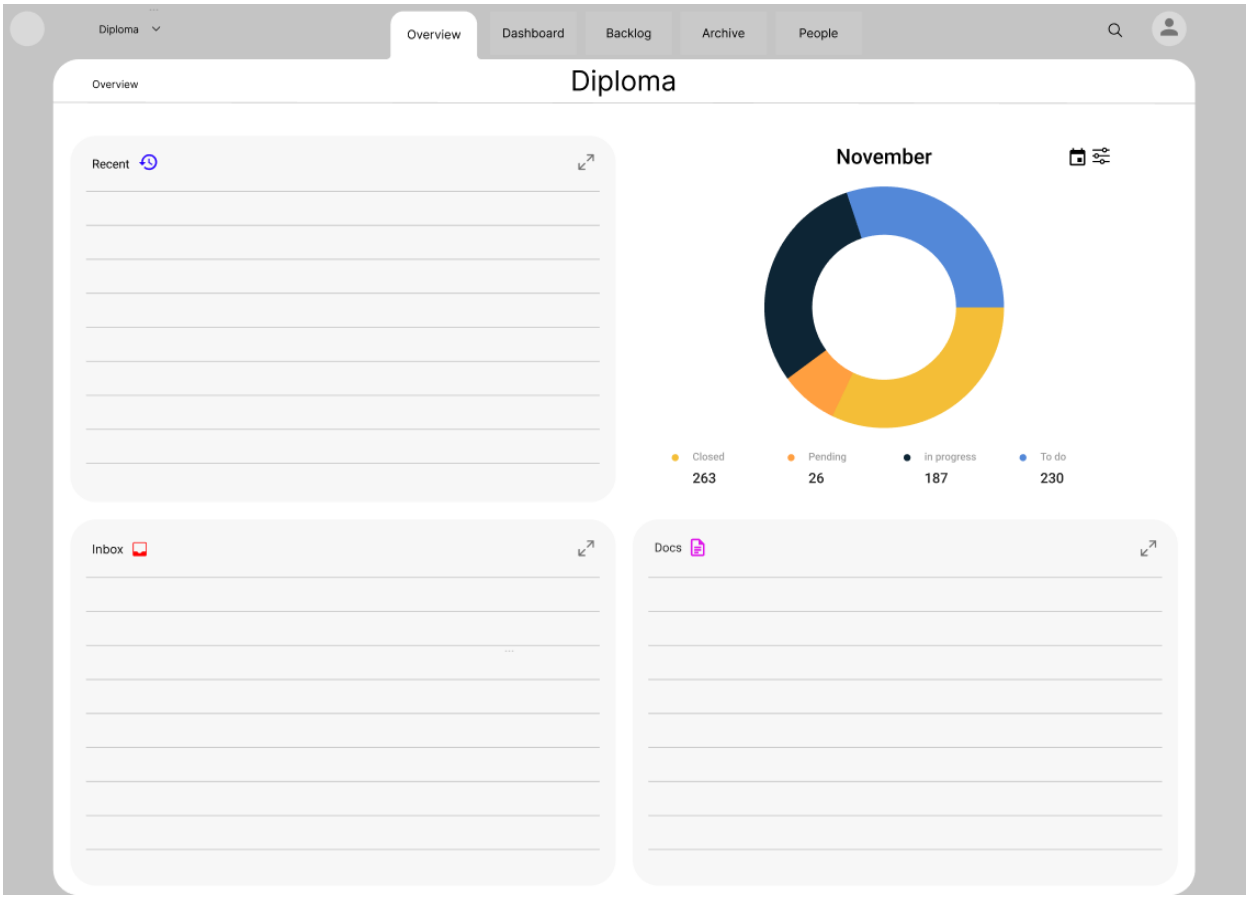


Рисунок 3.6 – Сторінка Overview

На сторінці Dashboard користувач може працювати з основними функціями для управління завданнями, такими як перегляд їхнього списку, фільтрація, сортування та додавання нових завдань. Завдання відображаються у вигляді карток, що містять назву, опис, дедлайн та статус виконання, що дає змогу зручно керувати ними (див. рис. 3.7).

Add Task надає можливість створювати нові завдання з гнучкими налаштуваннями. Користувач вводить назву завдання, яка є обов’язковою для заповнення і обмежена певною кількістю символів, що дозволяє зберігати

назви короткими та інформативними. Завдання можна розмістити в обраній колонці, яка відповідає його статусу, наприклад, «Очікує виконання», «У процесі» або «Готово» або ж лишивши пустим, в такому разі таск автоматично потрапить до “Backlog”. Вибір колонки відбувається через зручний випадаючий список.

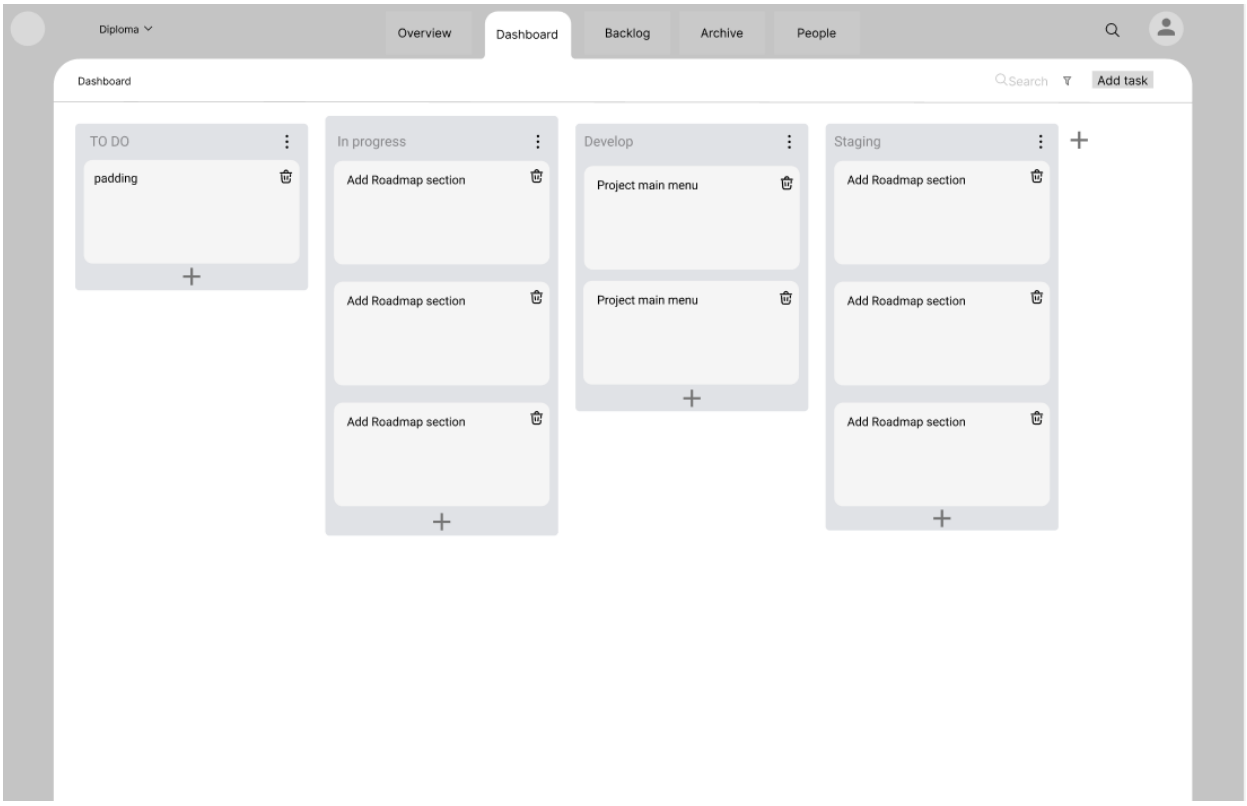


Рисунок 3.7 – Сторінка Dashboard

Для кожного завдання можна вказати його пріоритет: низький, середній високий або ж лишивши без пріоритету. Цей параметр впливає на відображення завдання в списку, де для кожного рівня важливості використовується свій кольоровий маркер. Виконавця (Assignee) призначають із числа учасників команди за допомогою випадаючого списку або пошуку. Можливе призначення одного або декількох виконавців.

Дедлайн вказується через інтегрований календар із вибором дати та часу. Система автоматично підсвічує завдання, у яких дедлайн наближається або вже минув. Тип завдання можна обрати зі списку, наприклад, “Specification”, “Backend” або “Front-end”.

Сторінка Archive призначена для перегляду завершених завдань і проєктів, забезпечуючи збереження історії роботи та можливість аналізу виконаних дій. Усі завдання на сторінці архіву структуровано за кількома категоріями: «Закриті», «Видалені», «Нещодавно додані» та інші.

На сторінці відображається статистика, що включає загальну кількість завдань, які були закриті, а також кількість видалених завдань. Завдання зі статусом «Закриті» доступні для перегляду в повному обсязі: назва, опис, дедлайн, статус виконання, виконавці, прикріплені медіа та інші параметри. Усі завершені завдання можна відновити в список Backlog, що дозволяє повернути їх для подальшої роботи в разі необхідності.

У категорії «Видалені» користувач має доступ лише до короткої інформації про завдання, яка включає його назву, тип і дату останнього оновлення. Для видалених завдань можливість відновлення недоступна, однак ця інформація дозволяє користувачам відстежувати, які завдання були видалені та коли (див. рис. 3.8).

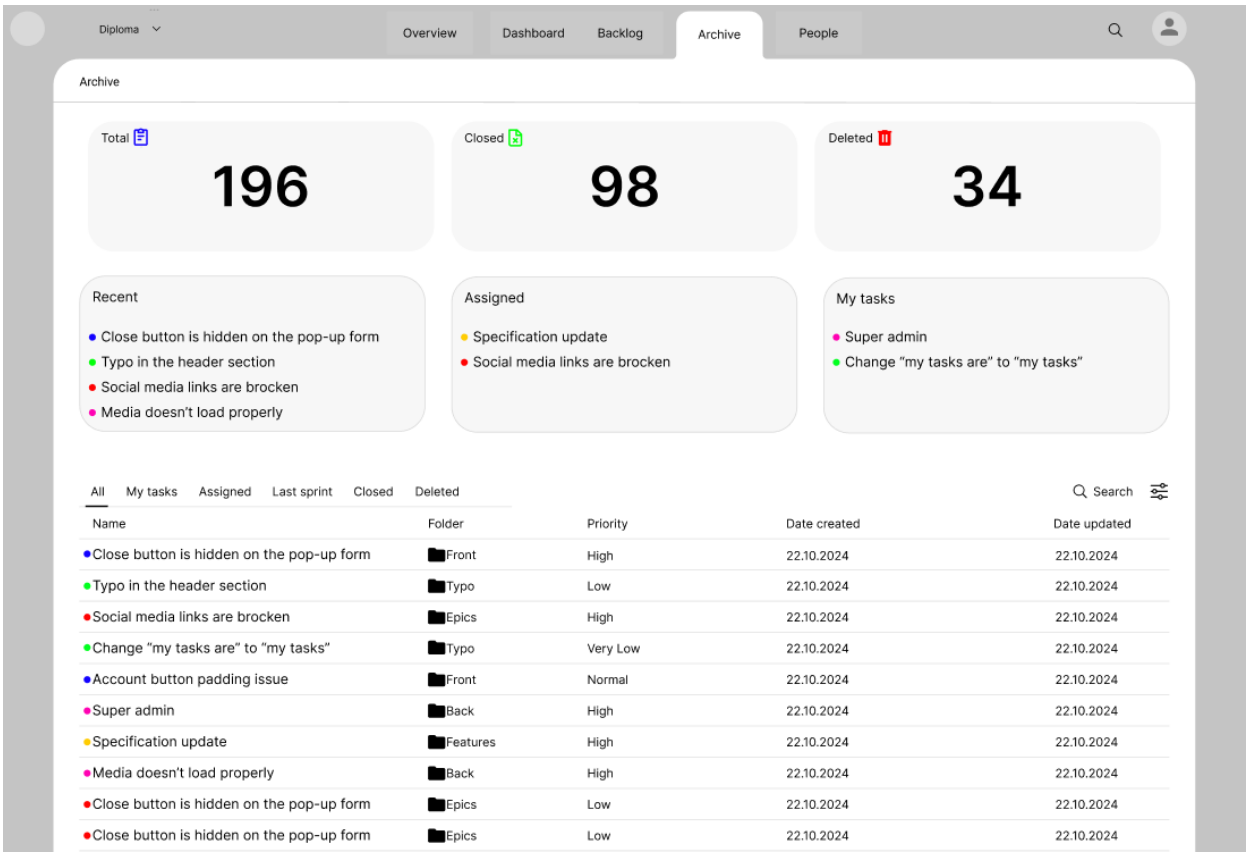


Рисунок 3.8 – Сторінка Archive

На сторінці Backlog зібрані завдання, які ще не розпочаті. Реалізовано функціонал перетягування завдань між різними статусами для планування їх виконання, а також зазначаються пріоритети завдань, що дозволяє ефективно керувати робочим процесом.

Для роботи з особистими даними користувача передбачена сторінка Account Page, де він може переглядати свою особисту інформацію та редагувати її, а також змінювати пароль для підвищення безпеки.

Форма Registration / Log In дає змогу новим користувачам зареєструватися в системі або увійти з уже наявним обліковим записом. Для забезпечення коректності введених даних реалізована перевірка, що дозволяє уникнути помилки у процесі реєстрації та авторизації (див. рис. 3.9).

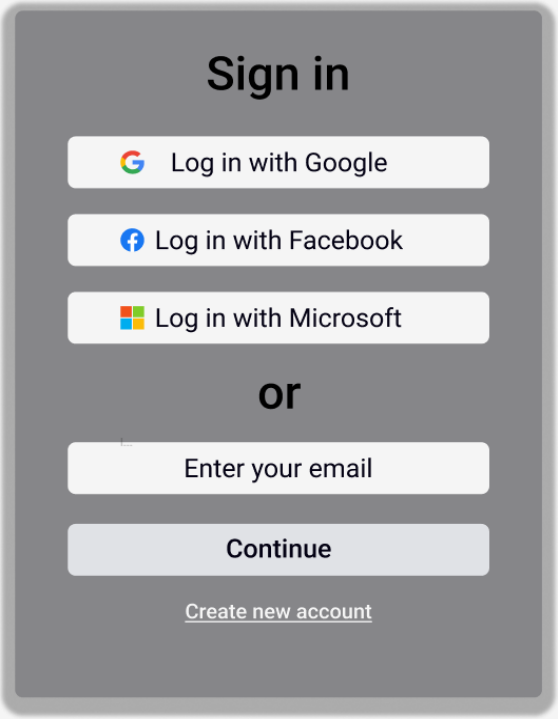


Рисунок 3.9 – Форма входу та реєстрації

Для створення або редагування завдань використовується Popup завдання, який можна викликати на сторінках Dashboard або Backlog. Це спливаюче вікно дозволяє швидко вводити назву, опис, дедлайн, пріоритет завдання та призначеного відповідального, що дає змогу користувачам не відриватися від основної роботи (див. рис. 3.10).

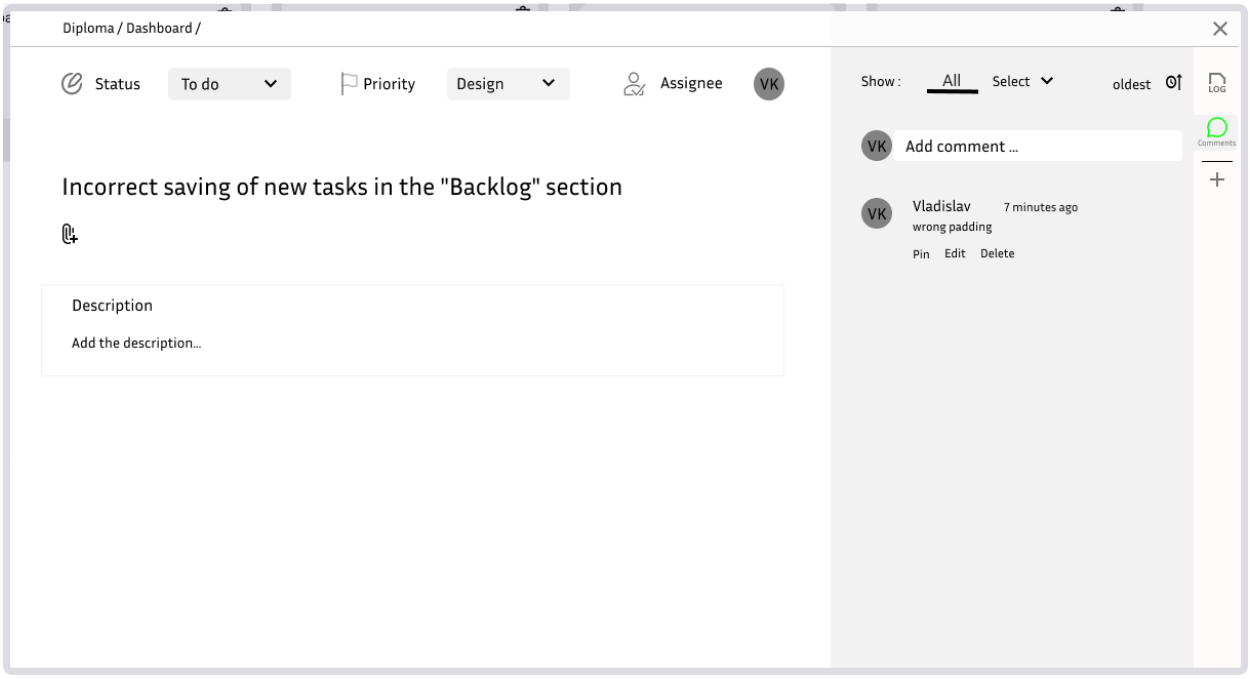


Рисунок 3.10 – Спливаюче вікно завдань

3.3 Серверна частина вебдодатку

Серверна частина вебдодатку базується на платформі Node.js із застосуванням фреймворку Express.js, який забезпечує створення гнучкого й високопродуктивного REST API. Основне завдання серверної частини – обробка запитів клієнтської частини, організація роботи з базою даних і забезпечення безпеки доступу до ресурсів [29].

Щоб забезпечити зрозумілість структури та легкість розширення, серверна частина побудована за модульним підходом (див. рис. 3.11):

– server.js: головний файл для запуску сервера;
– routes: директорія для опису маршрутів REST API;
– controllers: функції для обробки запитів і бізнес-логіки;
– models: схеми даних для взаємодії з MongoDB;
– config: зберігає конфігурацію, наприклад, налаштування бази даних.

Сервер використовує бібліотеку Mongoose для роботи з MongoDB (див. рис. 3.12), що дозволяє виконувати запити до бази й керувати даними через схеми.

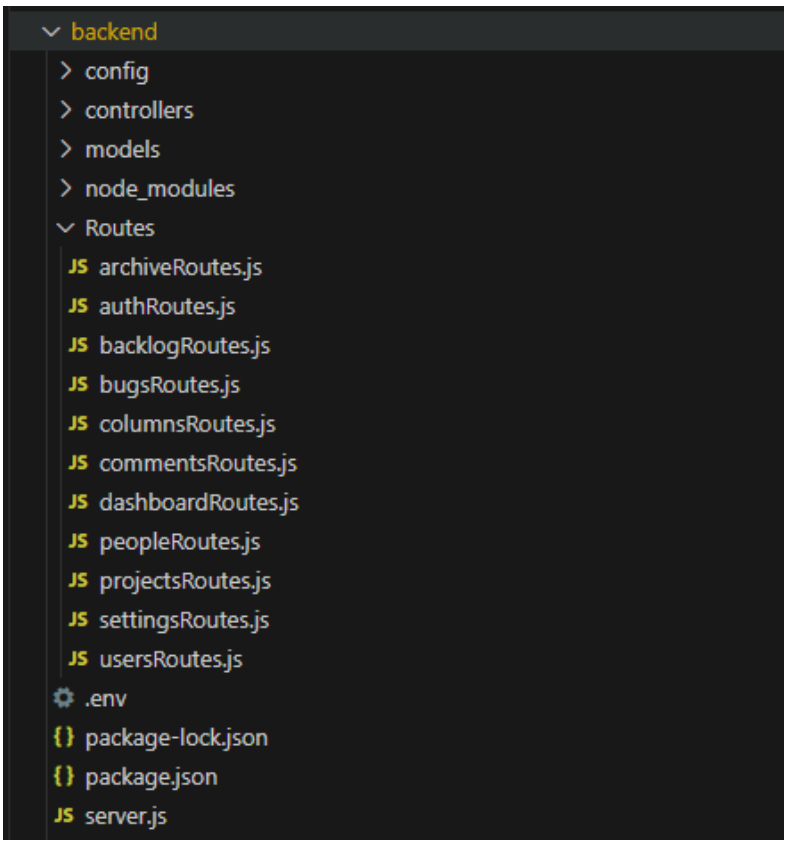


Рисунок 3.11 – Структура серверної частини

```
mongoose.connect(
process.env.MONGOOSECONNECT, {
 useNewUrlParser: true, useUnifiedTopology: true,}).then(() => {
 console.log('Connected to MongoDB');}).catch((error) => {
 console.log('Error connecting to MongoDB:', error); });
```

Рисунок 3.12 – Підключення бази данних

Головний файл server.js відповідає за налаштування сервера, підключення маршрутизаторів та запуск додатку (див. рис. 3.13).

Для кожного ресурсу вебдодатку створюються окремі файли з маршрутами. Це дозволяє організувати код у структурованому вигляді, що спрощує підтримку та розвиток системи. Наприклад, файл routes/tasksRoutes.js містить маршрути для роботи із завданнями. Кожен маршрут відповідає за певну операцію, пов’язану з ресурсом, і викликає відповідну функцію з файлу контролерів.

```
mongoose.connect(
process.env.MONGOOSECONNECT, {  }); //database connection
app.post('/project-add', (req, res) => {
 const newProject = new Project({
  title: generateRandomTitle(),
  description: uuidv4(),
  columns: [{name: 'Column 1', bugs: [{...},],},], });//bugs settings
 newProject.save().then((project) => {
    res.status(201).json({ project });}).catch((error) => {
    console.log('Error creating project:', error);
    res.status(500).json({ error: 'Failed to create project' });});});
const taskRoutes = require('./routes/tasks');
app.use('/api/tasks', taskRoutes);
//Other routes
app.listen(3001, () => {console.log(`Server is running on port ${3001}`);});
```

Рисунок 3.13 – Головний файл server

На рисунку 3.14 представлено логіку маршрутизатора tasks.js. Він відповідає за зв’язок між клієнтськими запитами та відповідними функціями контролера, які взаємодіють із базою даних. Ця структура маршрутизатора дозволяє чітко розділити обробку запитів та бізнес-логіку, спрощуючи налагодження та підтримку коду.

```
import express from 'express';
import {
getAllTasks, createTask, getTaskById, updateTask, deleteTask,
} from '../controllers/taskController.js';
const router = express.Router();
router.get('/', getAllTasks);
router.post('/', createTask);
router.get('/:id', getTaskById);
router.put('/:id', updateTask);
router.delete('/:id', deleteTask);
export default router;
```

Рисунок 3.14 – Маршрутизатор tasks

Контролери виконують основну бізнес-логіку й обробку запитів до серверної частини (див. рис. 3.15). Вони відповідають за обробку даних, отриманих від клієнта, та передають результати відповідним моделям. Завдяки цьому забезпечується чіткий поділ відповідальності між рівнями застосунку, що сприяє зручності підтримки й розширення системи [30].

```
const User = require('../models/User');
exports.getAllUsers = async (req, res) => {
  try {
     const users = await User.find();
     res.status(200).json(users);
  }
catch (err) {
     res.status(500).json({ message: 'Server error' });}};
```

Рисунок 3.15 – Маршрутизатор users

Для роботи з інформацією про завдання (tasks) створюється схема, яка визначає поля, типи даних, правила валідації, а також додаткові властивості.

Схема для задач створюється за допомогою mongoose.Schema. Вона дозволяє чітко вказати, які дані можуть бути збережені в колекції задач. Наприклад, завдання може мати такі поля: title, description, status, priority, dueDate, а також посилання на користувача або проєкт, до якого воно належить. Поле title є обов’язковим і використовується для короткого опису задачі. Поле description призначене для детального опису завдання і є необов’язковим. Поле status визначає стан задачі, а поле priority задає рівень важливості (див. рис. 3.16).

```
const taskSchema = new mongoose.Schema({
 id: {type: String,
 title: {type: String, required: true },
 description: {type: String, required: true, },
 priority: {type: String, required: true, },
 status: { type: String, required: true, },}, {
 timestamps: true,});
export default mongoose.model(taskModel, taskSchema);
```

Рисунок 3.16 – Схема tasks

Схема дозволяє точно визначити, які дані і якого типу можуть бути збережені. Завдяки параметрам, таким як required, unique та іншим, ми можемо перевіряти вхідні дані перед їх збереженням у базі. Їх можна легко розширити, додаючи нові поля чи властивості.

3.4 Тестування та відлагодження системи

Під час тестування та налаштування системи основна увага приділялася мануальному тестуванню, яке допомогло оцінити функціонування додатку в реальних умовах.

Для перевірки серверної частини використовували інструменти, такі як Postman, які дозволяли вручну викликати API-ендпоінти. Це дало змогу детально протестувати кожну функцію системи: отримання списку завдань, створення нового, оновлення інформації або видалення. Кожен запит аналізувався з точки зору коректності відповіді сервера.

Додатково тестування клієнтської частини здійснювалося безпосередньо через браузер. Інструменти розробника, наприклад Chrome DevTools, дозволяли в реальному часі відслідковувати, як дані передаються між клієнтом і сервером. Наприклад, під час створення нового завдання у вебінтерфейсі відразу перевірялося, чи правильно формується запит до сервера, чи зберігаються дані у базі, і чи повертаються вони назад для відображення у списку.

На фінальному етапі відтворювалися сценарії, близькі до реального використання. Наприклад, тестувалося, як користувач створює проєкт, додає завдання, призначає їх відповідальним і змінює статуси. Усі ці дії виконували вручну, паралельно контролюючи запити до сервера через Postman.

Мануальне тестування виявило не лише технічні помилки, а й допомогло побачити, як користувачі будуть взаємодіяти із системою.

3.5 Висновки до розділу 3

У третьому розділі було розглянуто основні аспекти розробки системи менеджменту проєктних завдань та помилок. Детально описано клієнтську та серверну частини вебдодатку, методи їх створення, а також етапи тестування та відлагодження системи.

Таким чином, у розділі було реалізовано основну функціональність вебдодатку, перевірено його працездатність та підтверджено готовність до подальшої експлуатації. Система відповідає початковим вимогам і має достатній потенціал для подальшого вдосконалення.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи була створена система для управління проєктними завданнями та помилками. Її основна мета – забезпечення ефективного керування проєктами, оптимізація робочих процесів та підвищення продуктивності командної роботи.

У межах дослідження було здійснено аналіз предметної області, розглянуто та порівняно існуючі системи, такі як Jira, Trello та GitHub Issues. Визначено ключові функціональні вимоги до розроблюваної системи, а також обґрунтовано вибір стеку технологій. Зокрема, для створення користувацького інтерфейсу було обрано React, серверна частина реалізована за допомогою Node.js та Express.js, MongoDB використано для зберігання даних, а Docker забезпечив контейнеризацію додатку.

Проєктування системи охоплювало визначення ключових функціональних компонентів, формулювання технічних вимог, створення UML-діаграм використання і послідовності, а також розробку структури бази даних на основі MongoDB.

Програмна реалізація була виконана з використанням сучасних технологічних рішень. Клієнтська частина забезпечує зручний користувацький інтерфейс, побудований за допомогою React. Серверна частина, реалізована на базі Node.js і Express.js, ефективно обробляє запити та забезпечує інтеграцію з базою даних MongoDB.

Етап тестування включав перевірку функціональності через виклики API з використанням спеціальних інструментів. Це дало змогу перевірити коректність обробки запитів, інтеграцію між клієнтською та серверною частинами, а також функціональність усіх основних модулів системи.

Система досягла поставлених цілей та повністю відповідає визначеним функціональним і технічним вимогам. У майбутньому її можна вдосконалювати, додаючи нові можливості або інтегруючи з іншими сервісами для покращення процесів управління завданнями.

ПЕРЕЛІК ПОСИЛАНЬ

1. Metodologii Documentation. URL: https://devisu.ua/uk/stattia/metodologii-upravlinnya-proktami-abo-shcho-take-waterfall-agile-ta-scrum (дата звернення: 17.05.2024).

2. Remote tools Documentation. URL: https://worksection.com/ua/blog/best-tools-for-remote-teams.html (дата звернення: 17.05.2024).

3. Kanban and scrum Documentation. URL: https://www.atlassian.com/agile/kanban/kanban-vs-scrum (дата звернення: 20.05.2024).

4. Jira and Documentation. URL: https://worksection.com/ua/blog/best-tools-for-remote-teams.html (дата звернення: 17.05.2024).

5. Project management tool requirements. URL: https://birdviewpsa.com/blog/12-project-management-software-considerations (дата звернення: 10.10.2024).

6. Key concepts defined. URL: https://www.iienstitu.com/en/blog/principles-of-management-key-concepts-defined (дата звернення: 15.07.2024).

7. Jira tool. URL: https://www.atlassian.com/software/jira/agile (дата звернення: 10.09.2024).

8. Trello tool. URL: https://www.simplilearn.com/tutorials/project-management-tutorial/what-is-trello (дата звернення: 12.06.2024).

9. Asana tool. URL: https://elementor.com/blog/the-tools-we-love-asana/ (дата звернення: 12.06.2024).

10. GitHub Issues tool. URL: https://docs.github.com/en/issues/tracking-your-work-with-issues (дата звернення: 12.07.2024).

11. React.js Documentation. URL: https://uk.legacy.reactjs.org/ (дата звернення: 25.05.2024).

12. Markup with JSX. URL: https://react.dev/learn/writing-markup-with-jsx (дата звернення: 25.05.2024).

13. React.js and Angular Documentation. URL: https://blog.udemy.com/ (дата звернення: 21.05.2024).

14. Axios. URL: https://bito.ai/resources/axios-npm-npm-explained/ (дата звернення: 22.06.2024).

15. Node.js Documentation. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 28.05.2024).

16. Comparison of Node.js and Python. URL: https://radixweb.com/blog/nodejs-vs-python (дата звернення: 23.07.2024).

17. MongoDB documentation. URL: https://www.mongodb.com/docs/manual/ (дата звернення: 06.10.2024).

18. MongoDB and MySQL. URL: https://www.knowi.com/blog/mongodb-vs-sql/ (дата звернення: 11.04.2024).

19. Express.js Documentation. URL: https://expressjs.com/ru/advanced/developing-template-engines.html (дата звернення: 01.06.2024).

20. Docker docs. URL: https://docs.docker.com/ (дата звернення: 03.06.2024).

21. Docker alternatives. URL: https://spacelift.io/blog/docker-alternatives (дата звернення: 03.11.2024).

22. Requirements for a Task and Bug Management. URL: https://luxequality.com/blog/how-to-write-a-bug-report/ (дата звернення: 28.07.2024).

23. UML діаграма прецедентів. URL: https://www.lucidchart.com/pages/uml-use-case-diagram (дата звернення: 20.09.2024).

24. UML діаграма послідовності. URL: https://sparxsystems.com/resources/tutorials/uml2/sequence-diagram.html (дата звернення: 16.09.2024).

25. noSQl database Documentation. URL: https://www.ibm.com/topics/nosql-databases (дата звернення: 02.06.2024).

26. Agile Documentation. URL: https://www.atlassian.com/agile (дата звернення: 04.06.2024).

27. React get started. URL: https://react.dev/learn (дата звернення: 04.06.2024).

28. React.js Client-Side Rendering. URL: https://flatirons.com/blog/react-js-server-side-rendering-vs-client-side-rendering/ (дата звернення: 09.10.2024).

29. Node.js and RestAPI. URL: https://www.tutorialspoint.com/nodejs/nodejs_restful_api.htm (дата звернення: 08.10.2024).

30. Routes and controllers. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes (дата звернення: 20.10.2024).