НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

Сергій СТІРЕНКО

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Веб-додаток для управління проектами на основі гнучкої методології створення ПЗ»

Виконала:
студентка IV курсу, групи ІВ-71
Підчасюк Ганна Олександрівна _____

Керівник:
Асистент,
Регіда Павло Геннадійович _____

Консультант з н. контроль:
Професор, д.т.н,
Сімоненко Валерій Павлович _____

Рецензент:
Старший викладач,
Тимофєєва Юлія Сергіївна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студентка _____

Київ – 2021 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

«Комп'ютерні системи та мережі»

спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Сергій СТІРЕНКО

«___»_____20__ р.

ЗАВДАННЯ

на бакалаврський дипломний проєкт студентці

Підчасюк Ганні Олександрівні

1. Тема проєкту «*Веб-додаток для управління проектами на основі гнучкої методології створення ПЗ*», керівник проєкту *Регіда Павло Геннадійович*, *асистент,* затверджені наказом по університету від « 11 » травня 2021 р. № 1139-с

2. Термін здачі студентом закінченого проєкту *27 травня 2021 р.*

3. Вихідні дані до проєкту *технічна документація і теоретичні дані.*

4. Зміст пояснювальної записки: *опис предметної області і сучасних методів розв'язання поставленої задачі, опис та порівняння систем-аналогів для менеджемнту проєктами, загальна структура веб-додатку, демонстрація працездатності продукту.*

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо) : *діаграма моделей бази даних, схема взаємодії клієнту і серверу, схема алгоритму роботи з веб-додатком.*

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Н. контроль	Сімоненко В.П., професор, д.т.н.		

7. Дата видачі завдання *21 вересня 2020р.*

Календарний план

з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Затвердження теми проєкту*	*10.12.2020-15.12.2020*	
2.	*Вивчення та аналіз завдання*	*15.12.2020-15.03.2021*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2021-25.03.2021*	
4.	*Розробка структур окремих підсистем*	*25.03.2021-5.04.2021*	
5.	*Програмна реалізація системи*	*5.04.2021-15.04.2021*	
6.	*Оформлення пояснювальної записки*	*15.04.2021-20.05.2021*	
7.	*Захист програмного продукту*	*25.05.2021*	
8.	*Передзахист*	*27.05.2021*	
9.	*Захист*	*17.06.2021*	

Студентка-дипломниця Ганна ПІДЧАСЮК

Керівник Павло РЕГІДА

АНОТАЦІЯ

У даному бакалаврському проєкті розроблено та реалізовано веб-додаток для управління проєктами на основі гнучкої методології створення ПЗ.

У основні дії які можна виконувати у веб-додатку: робота з командами, робота з проєктами, робота з задачами різних типів. Система може бути використана з метою врегулювання процесу створення нового продукту, не обов’язково в IT-сфері.

Програмна частина реалізована за допомогою стеку технологій MERN: MongoDB, Express.js, React.js та Node.js. Також було застосовано допоміжний фреймворк для формування інтерфейсу користувача Bootstrap та препроцесор SCSS.

ANNOTATION

In this bachelor's project a web application for project management based on an agile software development methodology has been developed and implemented.

The main actions that can be performed in this web application: working with teams, working with projects, working with different types of tasks. The system can be used to regulate the process of creating a new product, not necessarily in the IT.

The software part is implemented using the MERN stack of technologies: MongoDB, Express.js, React.js and Node.js. An additional framework Bootstrap and a SCSS preprocessor were also used to form the user interface and a SCSS preprocessor.

Відомість
до дипломного проєкту
на тему: «Веб-додаток для управління проектами на основі гнучкої методології створення ПЗ»

Київ – 2021 року

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

з/п	Формат	*Значення*	*Найменування*	Кількість листів	*Додаток*
1	***А4***		Завдання на дипломний проєкт	*2*	
2	***А4***	***ІАЛЦ.467200.001 ВП***	Відомість проєкту	*1*	
3	***А4***	***ІАЛЦ.467200.002 ТЗ***	Технічне завдання	*3*	
4	***А4***	***ІАЛЦ.467200.003 ПЗ***	Пояснювальна записка	*80*	
5	***А4***	***ІАЛЦ.467200.004 Д1***	Діаграма моделей БД	*1*	
6	***А4***	***ІАЛЦ.467200.005 Д2***	Схема взаємодії клієнту і серверу	*1*	
7	***А4***	***ІАЛЦ.467200.006 Д3***	Алгоритм роботи додатку	*1*	
8	***А4***	***ІАЛЦ.467200.007 Д4***	Текст програми	*31*	

					ІАЛЦ.467200.001 ОА	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розробила		*Підчасюк Г.О.*			*Веб-додаток для управління проєктами на основі гнучкої методології створення ПЗ. Опис альбому*	*Літ.* / *Аркуш* / *Аркушів*: *1* / *1*
Перевірив		*Регіда П.Г.*				
Н. контр.		*Сімоненко В.П.*				*НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІВ-71*
Затверд.		*Стіренко С.Г.*				

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломної роботи

освітньо-кваліфікаційного рівня бакалавр

на тему: «Веб-додаток для управління проектами на основі гнучкої методології створення ПЗ»

Київ – 2021 рік

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку курсу «Інженерія програмного забезпечення» та курсу «Основи бази даних». Область застосування: практичне використання у бізнесі, що базується на створенні нового програмного забезпечення.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського проєкту освітньо-професійної програми «Комп'ютерні системи та мережі», затверджене кафедрою Обчислювальної техніки Національного Технічного Університету України «Київський Політехнічний Інститут ім. Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даної дипломної роботи є розробка веб-додатку для управління проєктами на основі гнучкої методології створення ПЗ.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література з теорії і практики програмування, обробки даних, наукові статті, публікації в Інтернеті з даних питань.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до продукту, що розробляється

- Система може оброблювати дані лише для того клієнта, який зареєстрований в системі.

5.2. Вимоги до програмного забезпечення

- Наявність на сервері Node.js не нижче версії 7.14.0.
- Наявність на сервері Express.js не нижче версії 4.17.1.
- Наявність на сервері React.js не нижче версії 17.0.2.

5.3. Вимоги до апаратної частини.

- Сервер із підтримкою розгортання додатків на Node.js
- Оперативної пам’яті не менше 512 Мбайт.
- Стабільне підключення до мережі Інтернет.

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення літератури	12.04.2021
Складання і узгодження технічного завдання	19.04.2021
Створення структури системи і окремих модулів	26.04.2021
Тестування окремих модулів системи	03.05.2021
Допрацювання, налагодження і виправлення помилок	17.05.2021
Оформлення документації дипломної роботи	23.05.2021

Пояснювальна записка
до дипломного проєкту
на тему: «Веб-додаток для управління проектами на основі гнучкої методології створення ПЗ»

Київ - 2021 року

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

БД	База даних
ПЗ	Програмне забезпечення
СМ	Скрам майстер
СУБД	Система управління базами даних
API	Application programming interface
CRUD	Create, read, update and delete
CSS	Cascading Style Sheets
HTML	HyperText Markup Language,
HTTP	Hypertext Transfer Protocol
JS	JavaScript
JSON	JavaScript Object Notation
MERN	MongoDB, Express JS, React JS and Node JS
MIT	Massachusetts Institute of Technology
MVC	Model, View, and Controller
SASS	Syntactically awesome style sheets
SQL	Structured Query Language
URL	Uniform Resource Locator

ВСТУП

Протягом останніх 25-30 років гнучкі інноваційні методи для розробки програмного забезпечення значно зросли у використанні. За їх допомогою покращилась якість та швидкість виходу продукту на ринок, а також підвищилась мотивація та продуктивність IT-команд. Зараз ці методи поширюються в широкому діапазоні галузей на всі сектори та більшість функцій в організаціях.

Актуальність теми. Зараз гнучкі методології, які включають нові цінності, принципи, практики та переваги є радикальною альтернативою управлінському стилю менеджменту. Актуальність даного підходу демонструють не тільки IT-компанії, такі як Apple, IBM, Microsoft, Atlassian, а також інші сфери - компанія John Deere (американська машинобудівна компанія) для розробки нових машин, Saab (шведська компанія-виробник легкових автомобілів) - для виробництва нових винищувачів, Intronis (лідер у хмарних службах резервного копіювання) використовує їх у маркетингу, C. H. Robinson (світовий постачальник логістичних послуг) застосовує їх у підборі людських ресурсів, завод Mission Bell використовує їх для всього, від виробництва вина до управління старшою групою керівників.

Мета і задачі дослідження. Метою роботи є розробка веб-додатку для управління проєктами на основі гнучкої методології створення програмного забезпечення, що дозволить легко забезпечити виконання необхідних задач для кожного члена команди.

Для досягнення поставленої мети були поставлені наступні основні задачі:

- Провести аналіз вже існуючих додатків для організації роботи при створенні програмного забезпечення;
- Створити модель та програмну реалізацію веб-додатку;

- Провести моделювання роботи;
- Проаналізувати отримані результати.

Практичне значення. Веб-додаток для управління проєктами призначений для допомоги командам усіх типів в організації роботи. Головні цілі даної роботи - управління задачами та відслідковування помилок.

У розділі 1 наведено огляд існуючих аналогів веб-додатків для керування проєктами за різними методологіями створення ПЗ. Наведено порівняння цих систем, описано переваги та недоліки кожної з них.

У розділі 2 визначено технології, які були обрані для вирішення поставленого завдання.

У розділі 3 описано та реалізовано архітектуру системи із застосуванням технологій, які були визначені в розділі 2.

У розділі 4 наведено інструкцію роботи з веб-додатком для управління проєктами по створенню ПЗ.

Ключові слова. Методологія, веб-додаток, проєкт, менеджмент, Еджайл, Канбан, Скрам, Вотерфол, відслідковування помилок, управління задачами, сервер, клієнт, команда.

РОЗДІЛ 1. ОГЛЯД ГНУЧКИХ МЕТОДОЛОГІЙ ТА ІСНУЮЧИХ ВЕБ-ДОДАТКІВ ДЛЯ КЕРУВАННЯ ПРОЄКТАМИ

Гнучка методологія (далі Еджайл) - це вид процесу управління проєктами, який в основному використовується для розробки програмного забезпечення, де вимоги та рішення розвиваються завдяки спільним зусиллям організованих та міжфункціональних команд і їх клієнтів.

Еджайл - це сукупність принципів, що передбачають пристосованість та гнучкість. Еджайл прагне забезпечити кращу реакцію на мінливі потреби бізнесу, і тому зосереджується на наданні командам можливості виконувати ефективні дії. [1]

Виходячи із цінностей та принципів Еджайл маніфесту, він був створений як відповідь на незручність традиційних методів розвитку, таких як метод Вотерфол. Індустрія програмного забезпечення є висококонкурентним ринком через те, що програмне забезпечення потребує постійного оновлення. Це означає, що розробники повинні постійно вдосконалювати свої продукти, щоб не відставати від «гри», а лінійний і послідовний підхід Вотерфол для цього не підходить. [2]

1.1 Еджайл маніфест

Еджайл маніфест - це декларація цінностей та принципів, виражених у гнучкій методології. Складений з чотирьох основних цінностей та 12 ключових принципів, він має на меті допомогти розкрити кращі способи розробки програмного забезпечення, надаючи чітку та вимірювану структуру процесу, який сприяє ітеративному розвитку, співпраці команд та прийняттю іншої стратегії.

Основні цінності:

- Люди та їх взаємодія важливіші за процеси та інструменти.
- Продукт, який працює, важливіший за деталізовану документацію.
- Співпраця із замовником важливіша, ніж узгодження умов контракту.
- Бути готовим до змін важливіше, ніж слідувати плану[3].

Маніфест визнає важливість процесу та інструментів, враховуючи, що взаємодія кваліфікованих людей має найважливіше значення. Відповідно до цього, детальна документація не обов'язково потрібна, основна увага повинна залишатися на кінцевому продукті - постачанні робочого програмного забезпечення. Тому кожна команда проєкту повинна сама визначити, яка документація є необхідною.

12 ключових принципів Еджайл маніфесту:

- Найпріоритетнішим є задоволення потреб клієнта шляхом постійних демонстрацій програмного забезпечення на ранніх етапах розробки.
- Лояльне ставлення до зміни у вимогах проєкту, навіть на кінцевих стадіях розробки.
- Постійна демонстрація робочого ПЗ, ітераціями від декількох тижнів до декількох місяців.
- Замовник та команда повинні працювати разом протягом усього проєкту.
- Створення проєктів навколо людей, які мотивовані.
- Особиста розмова - найкращий метод передачі інформації та вимог до команди розробників та в рамках самої команди розробників є.

- ПЗ, яке працює, є головним показником прогресу роботи команди.
- Гнучкі процеси сприяють сталому розвитку. Увага до технічної досконалості та гарного дизайну підвищує гнучкість.
- Простота – деталізувати обсяг роботи, яку потрібно зробити. Тобто велике завдання розбивається на багато дрібних.
- Найкращі реалізації вимог виникають у команд, що можуть самостійно організувати свою роботу.
- Через рівні проміжки часу команда переглядає свою роботу, щоб стати ефективнішими.

Маніфест пропонує хороший огляд того, що очікується, коли йдеться про практики життєвого циклу Еджайл розробки. [3]

1.2 Методологія Скрам

Скрам – це вид Еджайл і одна з найпопулярніших систем процесів для реалізації Еджайл. Це ітеративна модель розробки програмного забезпечення, що використовується для управління складними процесами створення продуктів. Ітерації фіксованої довжини, які називаються спринтами тривалістю від одного до двох тижнів, дозволяють команді регулярно демонструвати результат роботи. В кінці кожного спринту зацікавлені сторони та члени команди збираються, щоб спланувати наступні кроки.

Скрам дотримується набору ролей та зустрічей, які ніколи не змінюються. Наприклад, Скрам вимагає проведення чотирьох мітингів, які формують структуру кожного спринту: планування спринту, щоденний мітинг, демонстрація спринту та ретроспектива спринту. Під час кожного спринту команда використовує візуальні артефакти, такі як дошки завдань або діаграми, щоб показати прогрес та отримати додаткові відгуки.

Скрам – це визначена методологія з конкретними ролями та зустрічами. У Скрам є три конкретні ролі.

Власник продукту: Власник продукту Скрам має бачення того, що він хоче створити, і передає це бачення команді. Він фокусується на потребах бізнесу та ринку, надаючи пріоритет всій роботі, яку потрібно виконати. Він створює та управляє завданнями, надає вказівки щодо того, які функції реалізовувати далі, а також взаємодіє з командою та іншими зацікавленими сторонами, щоб переконатися, що всі розуміють елементи створення проєкту. Власник продукту не є менеджером проєкту. Замість управління статусом та прогресом, його або її робота полягає в тому, щоб мотивувати команду ціллю та баченням.

Скрам Майстер(СМ): Часто вважається тренером команди, СМ допомагає команді робити якнайкраще роботу. Це означає організацію зустрічей, боротьбу з перешкодами та проблемами, а також роботу з власником продукту, щоб забезпечити список нових задач до наступного спринту. СМ також стежить за тим, щоб команда дотримувалася Скрам-процесу. Він не має повноважень над членами команди, але має повноваження над процесом. Наприклад, СМ не може сказати комусь, що робити, але може запропонувати нову каденцію спринту.

Скрам Команда: Команда Скрам складається з п’яти-семи членів. Всі учасники проєкту працюють разом, допомагають одне одному. На відміну від традиційних команд розробників, тут немає різних ролей, таких як програміст, дизайнер або тестувальник. Всі разом виконують комплекс робіт. Команда Скрам володіє планом кожного спринту; вони передбачають, скільки роботи вони можуть виконати за кожну ітерацію.

У потоці Скрам існує конкретний, незмінний набір кроків. Розглянемо їх відповідно до рисунку 1.1.

Рис. 1.1 – Схема роботи відповідно до Скрам.

Список задач: власник продукту та команда Скрам збираються, щоб визначити пріоритети елементів зі списку задач (робота над даним списком походить від вимог користувачів). Список задач – це не перелік речей, які потрібно завершити, а це, швидше, список усіх бажаних функцій продукту. Потім команда розробників обирає роботу із перерахованих завдань для завершення під час кожного спринту.

Планування спринту: перед кожним спринтом власник продукту представляє команді основні задачі на зустрічі з планування спринту. Потім команда вибирає, яку роботу вона може виконати під час спринту, і переміщує роботу з загального списку до списку задач, які мають бути реалізовані за один спринт.

Уточнення задач: після закінчення одного спринту команда та власник продукту зустрічаються, щоб переконатися, що задачі готові до наступної ітерації. Команда може видаляти неактуальні стратегії користувачів, створювати нові, переоцінювати пріоритети або розділяти великі завдання на менші. Мета цієї зустрічі полягає у забезпеченні того, щоб задачі містили лише елементи, які є релевантними та детальними та відповідають цілям проєкту.

Щоденні зустрічі Скрам: це 15-хвилинне нарада, на якій кожен член команди розповідає про свої цілі та будь-які проблеми, що виникали. Ця

зустріч відбувається щодня під час спринту і допомагає формувати актуальні пріоритети для команди.

Зустріч з огляду спринту: наприкінці кожного спринту команда представляє роботу, яку вони виконали. Ця зустріч повинна мати демонстрацію, а не звіт або презентацію.

Ретроспектива спринта: також наприкінці кожного спринту команда обмірковує, наскільки добре працює Скрам, та розповідає про будь-які зміни, які потрібно реалізувати у наступній ітерації. Команда може говорити про те, що пройшло добре під час спринту, що пішло не так і що вони могли зробити інакше.

Даний метод має масу переваг. Переваги Скрам включають:

- Більша прозорість проєкту: проводяться щоденні наради, для того щоб вся команда була ознайомлена з тим, хто які завдання виконує, що запобігає уникненню багатьох непорозумінь. Проблеми визначаються заздалегідь, що дозволяє команді вирішити їх до того, як вони вийдуть з-під контролю.
- Контрольована звітність команди: немає жодного менеджера проєкту, який би вказував команді Скрам, що і коли робити. Натомість команда колективно вирішує, яку роботу вони можуть виконати в кожному спринті. Всі вони працюють разом і допомагають один одному, надаючи кожному члену команди можливість бути незалежним.
- Легко впроваджувати зміни: завдяки коротким спринтам та налагодженому зворотному зв’язку легше впровадити зміни до проєкту.
- Економія витрат: постійне спілкування гарантує, що команда знає всі проблеми та зміни, як тільки вони виникають, що допомагає зменшити витрати та покращити якість продукту. Завдяки створенню та тестуванню функцій поступово, є

постійний контроль, і помилки можна виправити на ранніх термінах, перш ніж вони стануть занадто дорогими для виправлення.

Хоча Скрам має деякі конкретні переваги, він також має деякі мінуси. Методологія вимагає від команди високого рівня досвіду та відданості.

Ось недоліки Скрам:

- Ризик розгортання обсягів: деякі проєкти Скрам можуть розширюватися через відсутність конкретної дати завершення. Без дати завершення, зацікавлені сторони можуть продовжувати створювати додаткові функції.
- Команда вимагає досвіду та відданості: маючи визначені ролі та обов'язки, команда повинна знати принципи Скрам, щоб досягти успіху.
- Некваліфікований Скрам Майстер може все зіпсувати. СМ сильно відрізняється від менеджера проєкту. Він не має повноважень над командою; йому або їй потрібно довіряти команді, якою вони керують, і ніколи не говорити їм, що робити. Якщо Скрам Майстер намагається керувати командою, проєкт зазнає невдачі.
- Погано визначені завдання можуть призвести до помилок: витрати та терміни проєкту не будуть точними, якщо завдання не будуть чітко визначені. Якщо початкові цілі незрозумілі, планування стає важким, і спринти можуть зайняти більше часу, ніж передбачалося на початку. [4]

1.3 Методологія Канбан

Канбан - це метод управління робочим процесом для визначення, керування та вдосконалення послуг, що забезпечують результат. Він

спрямований на те, щоб допомогти візуалізувати свою роботу, максимізувати ефективність та постійно вдосконалюватися. З японської, Канбан перекладається як рекламний щит або вивіска. Він виник у виробництві, згодом, став об’єктом інтересу, на який претендували команди розробників програмного забезпечення Еджайл. Нещодавно його почали визнавати бізнес-підрозділи різних галузей.

Девід Дж. Андерсон сформулював метод Канбан як підхід до поступових, еволюційних процесів та змін систем для організацій, що працюють над програмним забезпеченням. Його основи можна розділити на чотири основні принципи.

Принцип 1: Почніть з того, що ви робите зараз

Гнучкість Канбан дозволяє накладати його на існуючі робочі процеси, системи та процеси, не порушуючи того, що вже успішно робиться; природно, це буде висвітлювати проблеми, які потребують вирішення, та допомагати оцінювати та планувати зміни, тому їх впровадження буде якомога непорушним.

Універсальність Канбан дозволяє поступово вводити його в усі типи організацій, не боячись надмірних зобов’язань, оскільки немає необхідності вносити широкі зміни з самого початку.

Принцип 2: Погодьтесь на подальші еволюційні зміни

Методологія Канбан розроблена так, щоб задовольнити вимоги ТЗ. Це покращує постійні незначні поступові та еволюційні зміни поточного процесу. Загалом, радикальні зміни не рекомендуються, оскільки вони, як правило, стикаються з опором через страх чи невизначеність.

Принцип 3: Поважайте поточний процес, ролі та обов'язки

Канбан визнає, що існуючі процеси, ролі, обов'язки та назви мають цінність і, як правило, їх варто зберегти. Метод Канбана не забороняє зміни, але також не прописує їх як «панацею». Він призначений для

просування та заохочення поступових, логічних змін, не викликаючи страху перед самими змінами.

Принцип 4: Заохочуйте акти лідерства на всіх рівнях

Це найновіший принцип Канбан. Це нагадує вам, що лідерство походить від повсякденних дій людей на передовій своїх команд. Кожен повинен сприяти постійному вдосконаленню мислення, щоб досягти оптимальних показників на рівні команди/відділу/компанії. Це не може бути діяльністю на рівні управління. [5]

Канбан - це неруйнівна система управління еволюційними змінами. Це означає, що існуючий процес вдосконалюється малими кроками. Шляхом впровадження багатьох незначних змін, завдяки чому зменшується ризик для загальної стабільності системи.

Першим кроком у впровадженні Канбан є візуалізація робочого процесу. Це робиться у формі дошки, що складається з простих наліпок або карток. Кожна картка на дошці представляє завдання.

Використання дошки Канбан для управління своєю роботою, визначає кожен із кроків у процесі та активно відстежує роботу під час її просування. Дошки Канбан гнучкі та налаштовуються так, щоб редагувати дизайн дошки в міру розвитку процесу. У цьому сенсі існує безліч типів дошок, оскільки кожна дошка з них - унікальна для команди або організації, яка її використовує.

У класичній моделі дошки Канбан використовує три колонки, як показано на рисунку 1.2:

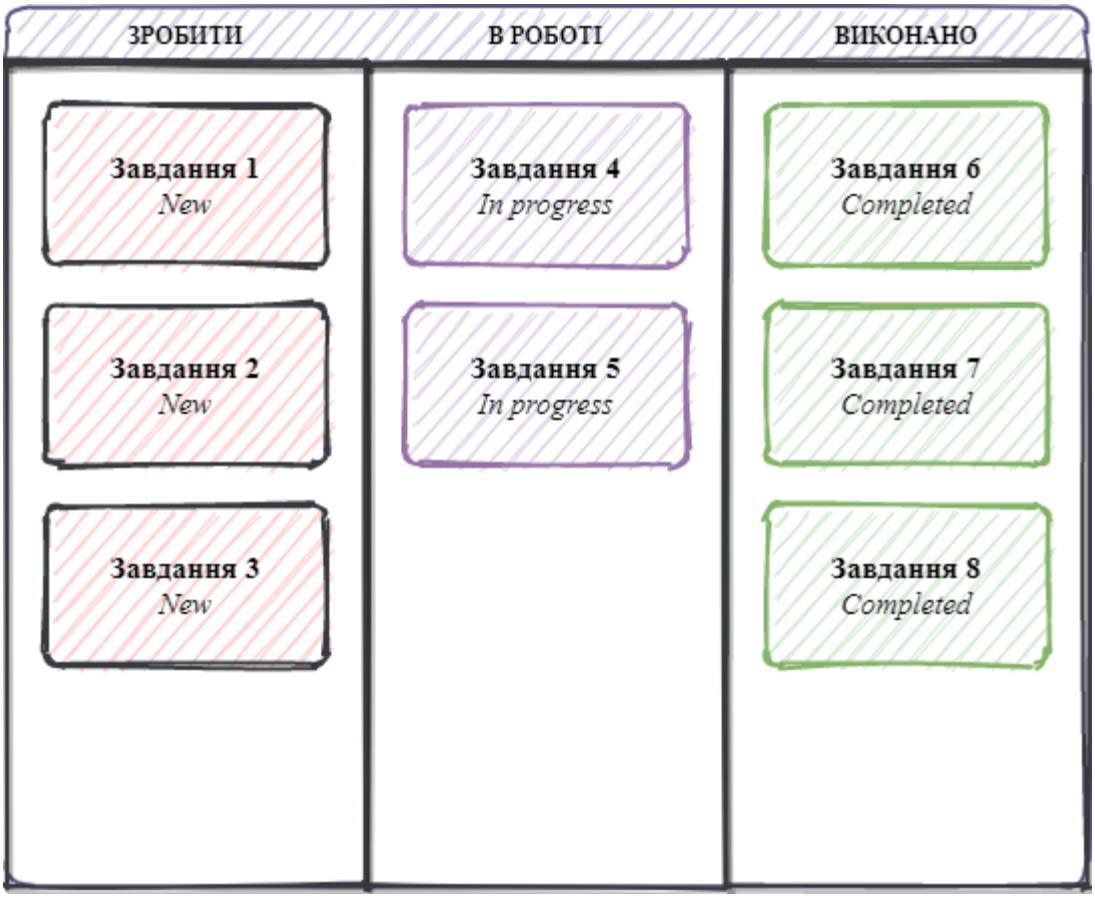


Рис. 1.2 – Класична модель дошки Канбан.

- «Зробити»: у цьому стовпці перелічені завдання, які ще не розпочаті.
- «Виконання»: Складається із задач, які виконуються.
- «Готово»: Складається із задач, які виконано.

Ця проста візуалізація призводить до великої прозорості розподілу роботи. Звичайно, дошки Канбан можуть показувати складні робочі процеси в залежності від складності робочого процесу та потреби у візуалізації та вивченні конкретних частин робочого процесу для виявлення вузьких місць з метою їх усунення. [6]

1.4 Відомі додатки для керування проєктами

Є кілька важливих елементів, якими має володіти будь-яка гнучка програма управління проєктами. Тим не менше, вибираючи гнучкий інструмент, можна додатково встановити власні критерії, оскільки слід

враховувати лише те програмне забезпечення, яке зручне для команди розробників. У цьому випадку інтерфейс відіграє важливу роль, оскільки дозволяє зручніше використовувати всі функції.

Існує перелік елементів, які потрібно шукати в основних інструментах для Еджайл розробки:

- Підходить для команд - наскільки гнучкі інструменти підходять для командної роботи та співпраці розробників;
- Підходить для управління завданнями. Додавання та переміщення завдань повинно бути простим, а весь інтерфейс не повинен бути надто складним для розуміння;
- Наявність аналітики. Спритна система управління проєктами передбачає аналіз даних з метою вдосконалення процесів розробки продукту. Найкращі гнучкі інструменти пропонують такий аналіз та статистику за замовчуванням, що дозволяє менеджеру уникати підрахунку інформації вручну.

Також варто розглянути, наскільки певний інструмент інтегрується у роботу з іншими інструментами, але це залежить від особистої ситуації та інструментів, якими зазвичай користується команда на роботі.

1.4.1 Bitrix24

Bitrix24 гнучке програмне забезпечення для управління проєктами пропонує платформу з широким набором інструментів для між групової комунікації, а також для співпраці з клієнтами. Організовуючи свій робочий процес за допомогою Bitrix24, можна відстежувати будь-які взаємодії з клієнтом, сортувати та зберігати дані, а також створювати звіти, що економить час.

На рисунку 1.3 продемонстровано робочий стіл додатку.

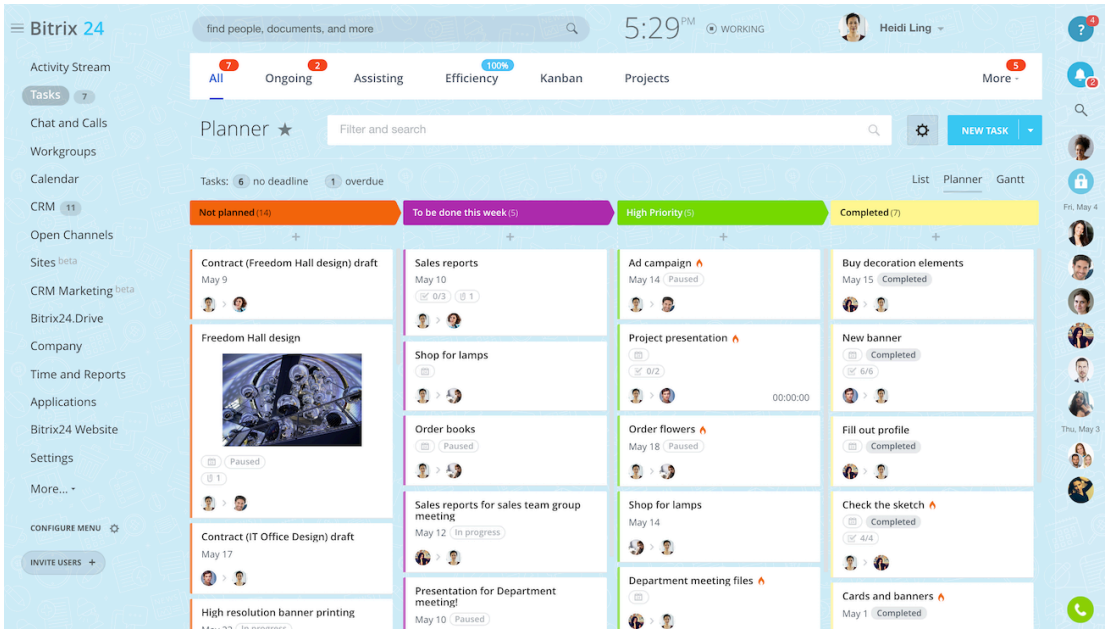


Рис. 1.3 – Вигляд робочого столу веб-додатку Bitrix24

Також присутні основні інструменти для організації роботи розробників, планування зустрічей та встановлення термінів. Завдання можуть бути призначені конкретним працівникам, відстежується час, витрачений на виконання завдання, що дозволяє коригувати процеси, якщо це необхідно.

Користувачі можуть вибрати параметри, які дозволяють передавати всю інформацію зі свого веб-сайту безпосередньо до CRM системи. Це дозволяє зберігати статистичні дані про роботу та замовлення, а також створювати зворотній зв’язок.

Якщо потрібне спілкування з кількома людьми одночасно, можна створити групові електронні листи. Усі члени вашої команди будуть в курсі поточних питань. Вони також можуть слідувати запропонованому графіку та відвідувати збори відповідно до нього.

Bitrix24 пропонує різні діаграми та інструменти для планування роботи над проєктами. Можливо як візуалізувати базовий план, так і залежності між завданнями, що дозволяє розробникам визначити пріоритети одного з них, перш ніж переходити до інших. Також можна розділити більш велике завдання на низку менших завдань і виконувати

роботу покроково, створюючи контрольний список. Для спілкування з працівниками пропонуються також різні інструменти, починаючи від чатів та електронних листів і закінчуючи телефонною інтеграцією.

1.4.2 Atlassian Jira

Jira - ще один відомий гнучкий інструмент розробки програмного забезпечення. Дозволяє керувати всіма етапами розробки ПЗ, починаючи з самого початку, коли планується проєкт, і під час роботи над ним, що включає кілька менших завдань, і до моменту випуску продукту та передачі його клієнту. Jira застосовується до різних гнучких методів управління проєктами, незалежно від того, віддаєте перевагу Скрам чи Канбан.

Такі гнучкі інструменти планування, як Jira, дозволяють мати повну карту процесу розвитку. Все, що вам потрібно зробити, або те, що вже виконується, можна додавати та контролювати в системі. Кожне завдання відокремлене і ним може керувати працівник, який виконує його сам. Створюючи план проєкту, ви можете додавати історії користувачів, а також розбивати основне завдання на менші. Усі проблеми, що виникають у процесі, також відстежуються, що дозволяє звертати увагу та знаходити рішення вчасно.

Спілкування між членами команди в Jira гарно організоване. Можна звернутися до цілої групи робітників, але також зв’язатися з певним членом команди.

Додаткові інструменти, які пропонує Jira, включають інформацію з різних бізнес-додатків. Вони можуть бути використані для підвищення ефективності роботи, але також надають розробникам більше ідей та деталей щодо сфери, в якій вони працюють. Користувачі Jira також мають можливість звернутися до команди підтримки, а також допоміжних матеріалів.

На рисунку 1.4 продемонстровано робочий стіл веб-додатку.

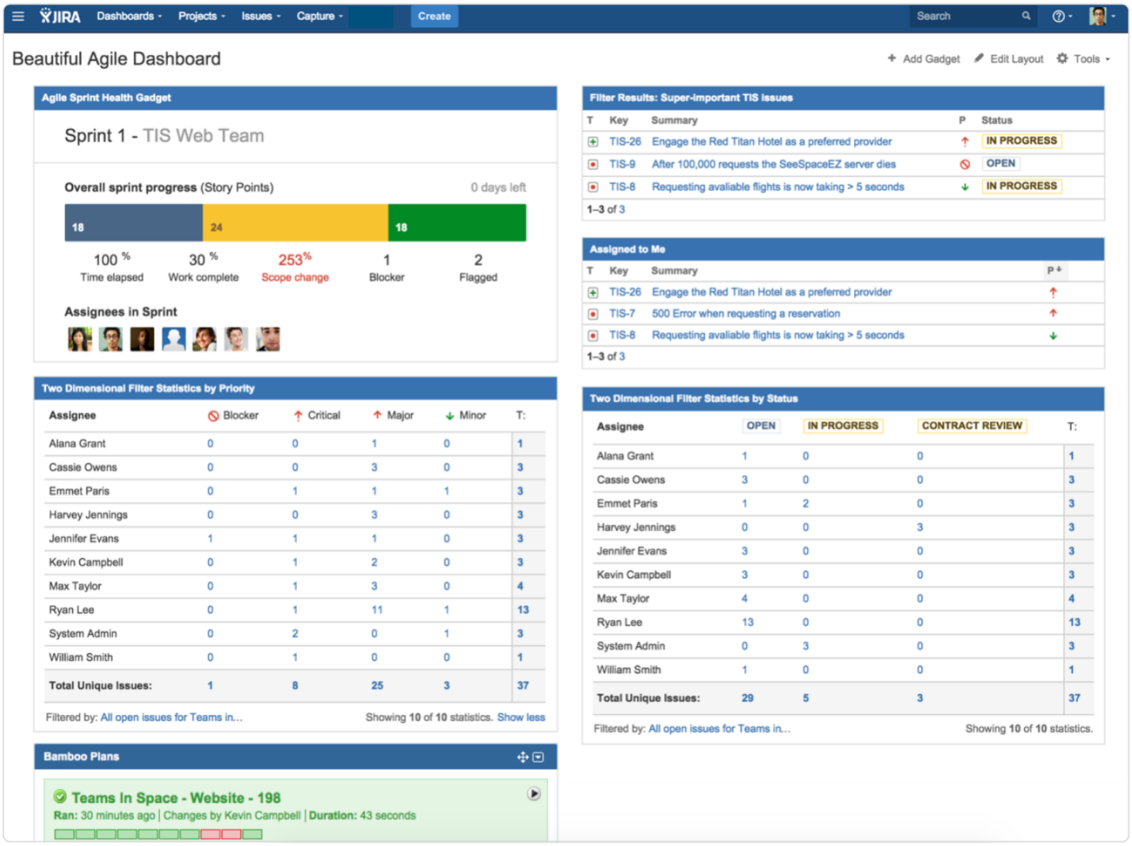


Рис. 1.4 – Приклад робочого столу веб-додатку Jira

Немає конкретного гаджета, який потрібен, якщо ви хочете використовувати Jira. Існує варіант програмного забезпечення, який зручніше використовувати з персонального комп'ютера. Але якщо у вас його немає під рукою, ви можете увійти в систему за допомогою мобільного додатка та відстежувати весь проєкт незалежно від того, де ви знаходитесь.

Хоча це виглядає надзвичайно привабливим гнучким програмним забезпеченням для проєкту, у Jira також є кілька недоліків:

- Вам доведеться витратити досить багато часу, щоб звикнути як правильно з цим ПЗ працювати. Jira також не найпростіший інструмент з точки зору налаштування і може вимагати спеціалістів для встановлення;

- Jira дорога. Незважаючи на те, що великі компанії можуть дозволити собі платити щомісяця запитувану плату, для невеликих підприємств та проєктів це можуть бути витрати, які не варті результату.

Але в цілому виконуються інші вимоги, Jira ідеально підходить для колективної роботи, оскільки вона одночасно покращує командне спілкування та дозволяє визначати пріоритети та розподіляти завдання. Також доступне виявлення проблем, що є надзвичайно важливою функцією, оскільки дозволяє своєчасно їх виправляти та підвищує ефективність.

1.4.3 Kanbanize

Kanbanize - це гнучкий інструмент розробки програмного забезпечення, який підвищує продуктивність праці шляхом створення візуальної карти робочого процесу. Це більше, ніж просто дошка в стилі Канбан, оскільки вона також включає автоматизацію бізнес-процесів і створює середовище для продуктивної та ефективної роботи розробників.

За винятком дошки Канбан, яка може бути встановлена в точності до потреб вашого проєкту та інструментів автоматизації бізнесу, Kanbanize також пропонує аналітичну систему, яка полегшує роботу менеджера проєкту, а також інструменти планування, що оптимізують терміни виконання роботи. Відстежуються різні сторони завершення проєкту та пропонуються дані, які згодом оптимізуються та впроваджуються в робочий процес.

Це також особливо корисно для між функціональних команд, оскільки дозволяє створювати кілька одночасних робочих процесів. Щоденні та довгострокові заходи можна запланувати за допомогою Kanbanize. Кожен з них може бути представлений простим завданням з чіткими вимогами, а також позначеним як найвищий пріоритет і

призначеним конкретному працівникові. Залежності між завданнями також керовані, що дозволяє розробникам звертати увагу на завдання в необхідному порядку.

На рисунку 1.5 продемонстровано головний екран додатку.

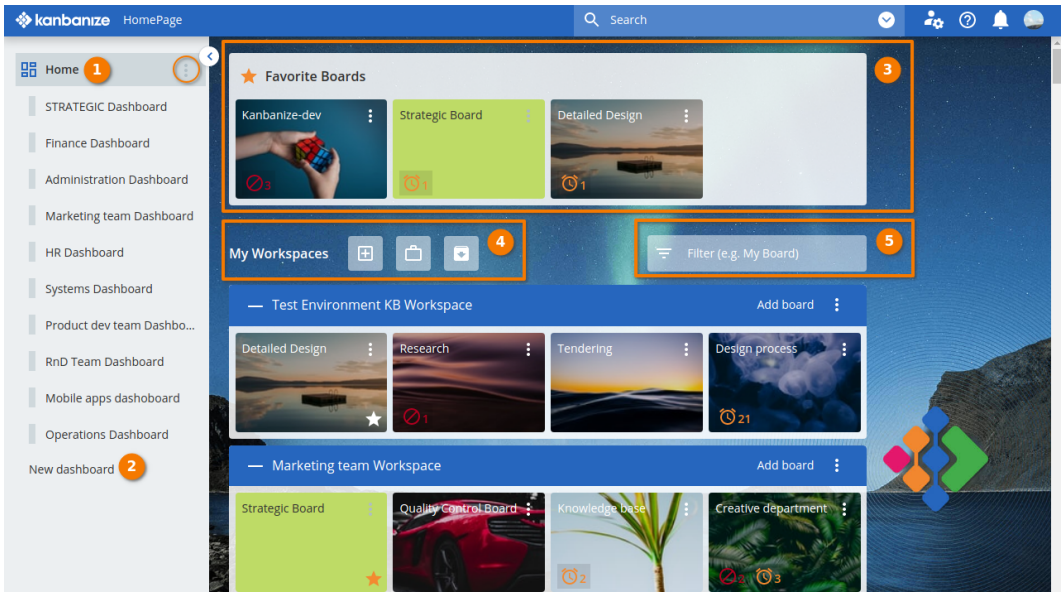


Рис. 1.5 – Вигляд робочого столу веб-додатку Kanbanize

Kanbanize має обмеження. Це означає, що працівники не перевантажені роботою і не спокушаються виконувати кілька завдань одночасно. Також їм набагато простіше приділяти достатньо уваги кожному завданню і виконувати його без помилок.

У той же час, система забезпечує велику прозорість серед команди, а також у компанії в цілому. Користувачам дуже просто змінити робоче поле, налаштувати кількість карток та обмеження роботи в процесі.

Ще однією важливою особливістю Kanbanize є те, що він показує роботу та прогрес у реальному часі. Це означає, що більше не потрібно оновлювати статус задач від членів команди, і тому вони можуть повністю

зосередитися на завданні. Це також означає, що набагато легше передбачити реалізацію проєкту, а також спланувати наступні етапи

завершення розробки продукту. Розробники виконують лише цінну роботу, яка впливає на кінцевий результат.

За необхідності Kanbanize можна інтегрувати з іншими інструментами, з якими працює ваша компанія. До них належать Google Drive, Jira, Dropbox та інші. Це особливо зручно для великих компаній з декількома відділами, які можуть використовувати різне програмне забезпечення.

1.4.4 Asana

Як і багато інших інструментів для гнучкого розвитку, Asana використовується як для організації процесу створення продукту, постановки завдань, так і для спілкування в команді, а також із клієнтом. Asana також застосовується до бізнесу будь-якого розміру та незалежно від кількості членів команди.

На рисунку 1.6 продемонстровано головну сторінку додатку.

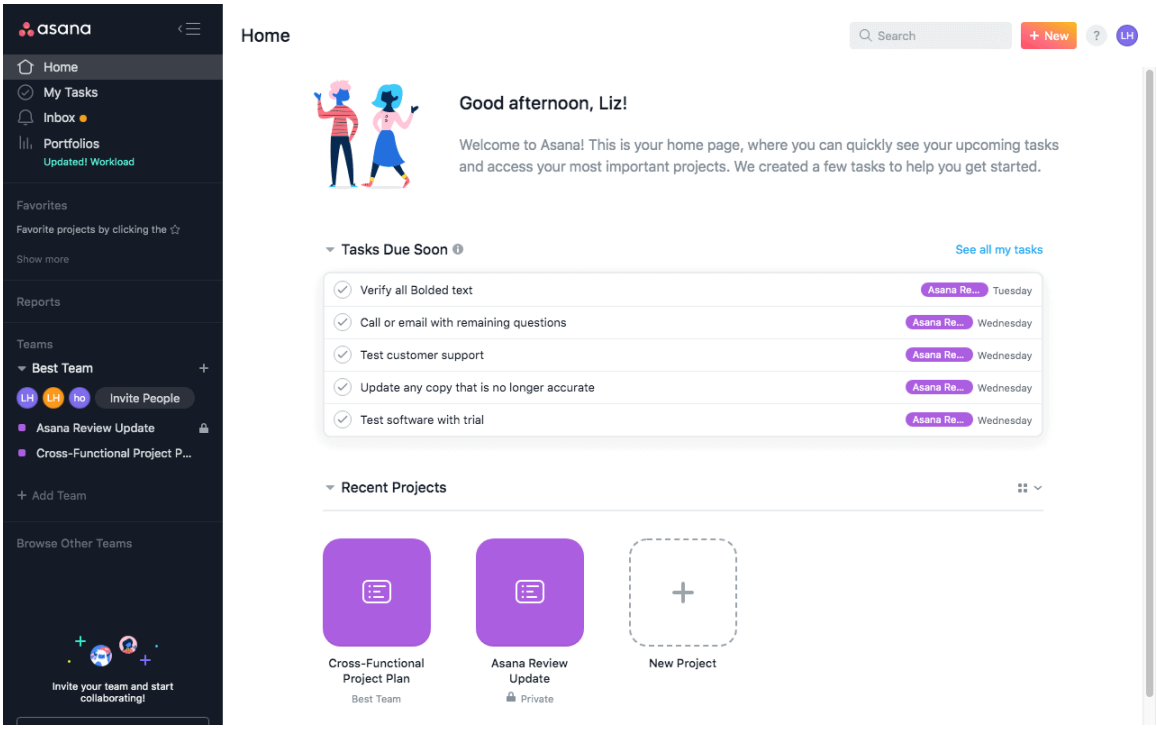


Рис. 1.6 – Вигляд робочого столу веб-додатку Asana

Робота з Asana починається з нанесення на карту проєкту та його розподілу на етапи. Використовуються прості картки, які позначені

різними кольорами, залежно від того, на якому етапі знаходиться ваш проєкт на даний момент. Це робить таку карту процесів надзвичайно наочною, що полегшує розуміння для кожного учасника. В Асані використовується проста система стовпців, яка робить її придатною для багатьох гнучких методологій управління проєктами. Тут можна самостійно вирішити, які колонки потрібні для проєкту. Вони можуть включати список справ, незавершених задач, відставання, перевірку та, звичайно, виконану роботу.

Якщо є певні елементи, які потрібно знайти, є можливість відфільтрувати завдання залежно від потреб. Наприклад, можна встановити термін виконання і відфільтрувати всі завдання відповідно до нього.

Також можна створити план відповідно до термінів та залежностей. Однак, якщо виконання певних завдань залежить від попереднього виконання інших, можна зв’язати їх, щоб розробники знали, на що звернути увагу. І якщо виконання завдання триває довше, завжди можна відредагувати шкалу часу.

Безумовно, є робота, яка може бути нудною для будь-якого менеджера проєкту. І це підрахунок статистики. Такі дані насправді дозволяють вдосконалити процеси розробки продукту, але в той же час займають багато часу. З Asana така робота виконується автоматично, і дані можна знайти у відповідному розділі, що означає, що більше немає необхідності робити це вручну. Також є можливість коригувати потік роботи кожного працівника, залежно від навантаження.

Asana також доступний інструмент для будь-якого типу пристрою. Окрім комп’ютера/ноутбука завжди є можливість підтримувати зв’язок за допомогою телефону, незалежно від операційної системи.

1.4.5 LeanKit

Наступним у списку найкращих гнучких інструментів управління проєктами є LeanKit. Він пропонує більше візуального підходу, який базується на системі Канбан. Він підходить для підприємств різного розміру незалежно від галузі.

Принципи, які підтримує LeanKit, дозволяють забезпечити середовище, де впроваджуються постійні вдосконалення щодо цінностей замовника та розробника.

На рисунку 1.7 продемонстрована головна дошка додатку.

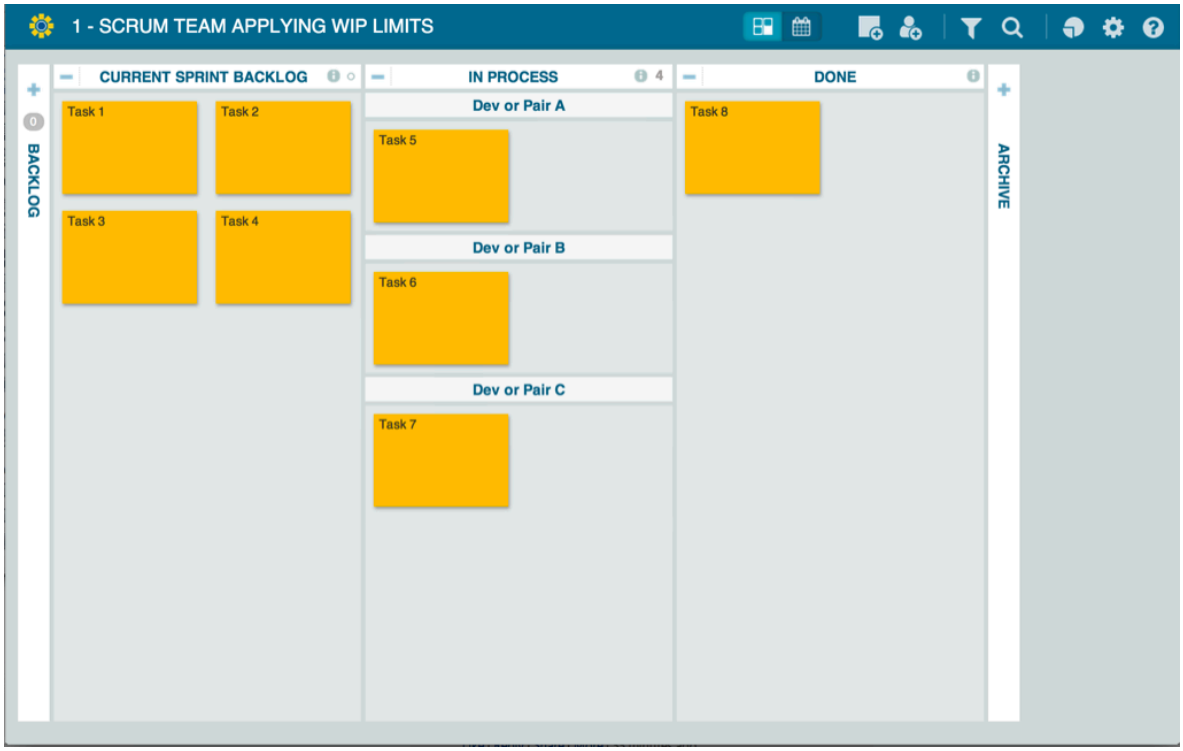


Рис 1.7 – Вигляд робочого столу веб-додатку LeanKit.

Дошку, запропоновану LeanKit, можна легко змінити. Вона включає вертикальну та горизонтальну смуги, що передбачають різні процеси:

- Горизонтально структуровані завдання відбуваються паралельно. У них беруть участь кілька розробників, які одночасно працюють над різними завданнями;

- Вертикальні смуги включають кожен крок, зроблений для виконання даного завдання.

Але оскільки процес роботи над проєктом триває і потрібно робити певні ітерації, можна змінити макет вашої дошки, додаючи або видаляючи завдання та замінюючи їх іншими. Ви також можете перепризначити завдання іншому працівникові. За потреби статус завдання також можна змінити, а якщо певні призначення пов'язані між собою, залежності також можна візуалізувати.

Ще одним корисним інструментом, який менеджери проєктів можуть знайти в програмному забезпеченні LeanKit, є аналітика. Можна відстежувати різні показники, які в цілому відображають ефективність проєкту. Це дозволяє позначити проблеми, які не є настільки очевидними, і скорегувати робочі процеси. Якщо звіт потрібно надіслати клієнту, його також можна створити за допомогою LeanKit. Інше програмне забезпечення для управління проєктами може бути інтегровано, що означає, що дані не будуть втрачені.

Якщо потрібно оновити свій проєкт, але поблизу немає ні ноутбука, ні персонального комп’ютера, ви можете увійти в систему LeanKit за допомогою мобільного телефону. Доступні як мобільний додаток, так і версія браузера. [7]

1.5 Порівняння існуючих веб-додатків для керування проєктами

Після детального огляду кожного з додатків варто порівняти їх між собою. Для зручності, порівняльна характеристика була зібрана в формі таблиці. Наведена таблиця 1.1 демонструє недоліки та переваги кожного з веб-додатків посеред інших.

Табл. 1.1 – Порівняння аналогів

Додаток / Функція	**Bitrix24**	**Jira**	**Kanbanize**	**Asana**	**LeanKit**
Відстеження часу роботи	+	+	+	+	-
Відстеження помилок	+	+	+	+	+
Управління завданнями	+	+	+	+	+
Візуалізація проєкту	+	+	+	+	-
Спілкування	+	-	-	+	-
Сповіщення	-	+	-	-	-
Вартість для новачків	+	+	-	+	-
Вартість для компанії	+	-	-	-	-
Інтеграція додаткового ПЗ	-	+	+	-	+
Календар	+	+	+	+	+

ВИСНОВКИ ДО РОЗДІЛУ 1

В першому розділі були розглянуті такі питання: що таке гнучка методологія Еджайл, які загальні правила (маніфест), найпоширеніші підвиди даної методології – Скрам і Канбан, а також існуючі додатки для керування проєктами та їх відмінності.

Були розглянуті основні принципи організації роботи відповідно до Еджайл маніфесту, які позиції є найголовнішими в роботі та ті, які можуть відійти на задній план.

Були розглянуті етапи розробки програмного забезпечення використовуючи методологію Скрам, які ролі і процеси існують в даному методі. Головною складовою цього методу є постійне спілкування з командою та замовником, постійний огляд роботи та відгуки щодо неї, задля покращення співпраці в майбутньому. Також були описані переваги та недоліки методу, які можуть вплинути на роботу.

Був розглянутий Канбан, як інструмент для створення програмного забезпечення та 4 основні принципи роботи за цією методологією. Головна ідея даної методології – це візуалізація проєкту та сортування задач за їх статусами, для зручності відслідковування роботи.

Були розглянуті такі найпопулярніші існуючі веб-додатки для управління проєктами на основі гнучкої методології як Bitrix24, Jira, Kanbanize, Asana та LeanKit. Зроблено огляд кожної з даних систем, а також їх порівняння.

Таким чином задача створення веб-додатку для управління проєктами на основі гнучкої методології є актуальною.

РОЗДІЛ 2. ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ

Існує багато наборів технологій які можна обрати для створення проєкту. Більшість розробників, як правило, вибирають найновіший і найпростіший варіант на ринку.

На жаль, це такі критерії не завжди є правильними для вибору найкращого набору технологій. Тут вступає в дію багато факторів, таких як:

- тип та розмір проєкту;
- командні знання та навички;
- час виходу на ринок;
- масштабованість;
- адаптивність до змін;
- вартість загального розвитку.

Для даного проєкту було обрано набір технологій MERN [8].

2.1 Стек технологій MERN

MERN - це скорочення, що використовується для опису певного набору технологій на основі JavaScript, які використовуються в процесі розробки веб-додатків. Він створений з ідеєю зробити процес розробки максимально плавним. MERN включає такі компоненти з відкритим кодом:

- MongoDB
- ExpressJS
- React / Redux
- NodeJS

Кожен із цих компонентів відіграє вирішальну роль у процесі розробки веб-додатків. Все це забезпечує наскрізну структуру для роботи

розробників. Наприклад, MongoDB - це система баз даних, NodeJS - це середовище для налаштування серверу, ExpressJS - це внутрішній веб-фреймворк, а React - це інтерфейсний фреймворк.

На рисунку 2.1 зображена взаємодія технологій стеку MERN.

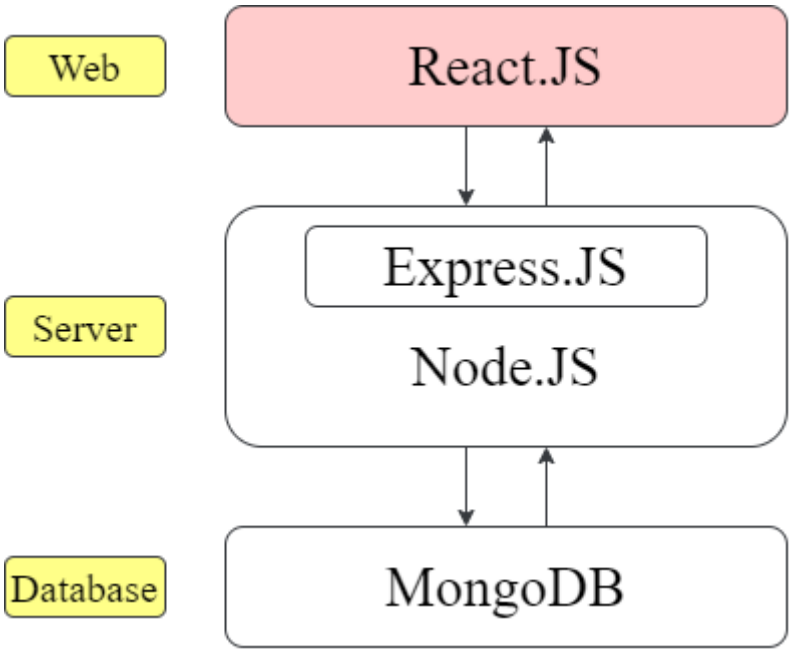


Рис. 2.1 – Взаємодія компонентів набору MERN

Як показано на ілюстрації вище, користувач взаємодіє з компонентами ReactJS у інтерфейсі програми, що знаходиться у браузері. Цей інтерфейс обслуговується сервером програми через ExpressJS, що працює поверх NodeJS.

Будь-яка взаємодія, яка створює запит на зміну даних, надсилається на сервер Express, що базується на NodeJS, який за потреби захоплює дані з бази даних MongoDB і повертає дані до інтерфейсу програми, який потім представляє користувачеві.

Головна перевага для розробників, що використовують стек MERN, полягає в тому, що кожен рядок коду написаний на JavaScript. Це мова програмування, яка використовується скрізь, як для клієнтського коду, так і для коду на стороні сервера. З однією мовою на різних рівнях немає необхідності в переключенні контексту.

Для технологічного стеку з багатьма мовами програмування розробники повинні зрозуміти, як компоненти мають взаємодіяти між собою. За допомогою стека JS розробникам потрібно лише володіти JS та JSON.

Загалом, використання стеку MERN дозволяє розробникам створювати високоефективні веб-додатки. [9]

Далі детальніше розглянемо кожен із цих компонентів.

2.2 React/Redux

React - це декларативна, ефективна та гнучка бібліотека JavaScript для побудови користувальницьких інтерфейсів. Вона дозволяє створювати складні інтерфейси з невеликих та ізольованих фрагментів коду, які називаються компонентами.

На рисунку 2.2 зображено логотип технології.

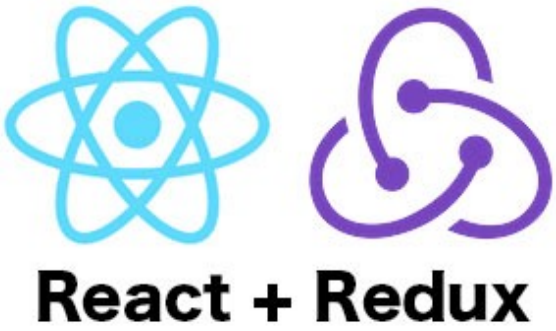


Рис. 2.2 – Логотип React/Redux

Компоненти використовуються, щоб повідомити React про те, що має бути бачити на екрані. Коли наші дані змінюються, React буде ефективно оновлювати та рендерити наші компоненти.

Життєвий цикл компонента React існує для захисту стану компонента. Стан компонента не повинен мутувати, поки React малює компонент. Натомість компонент потрапляє у відомий стан, малює, а

потім відкриває життєвий цикл для ефектів, оновлення стану та подій. [10]

React дозволяє розробникам створювати великі веб-програми, які можуть змінювати дані, не перезавантажуючи сторінку. Основна мета React - бути швидким, масштабованим і простим. Він працює лише на користувальницьких інтерфейсах програми. Це відповідає поданню в шаблоні MVC. Його можна використовувати з комбінацією інших бібліотек або фреймворків JavaScript, таких як Redux.

Redux - це популярна бібліотека JavaScript для управління станом вашої програми. Стан програми - це як глобальний об’єкт, який зберігає інформацію і використовується для різних цілей, наприклад, прийняття рішень щодо того, які компоненти і коли відображати, візуалізація збережених даних тощо.

Великі програми мають великі стани додатків, і управління ними стає все більш незручним у міру зростання програми. [11]

На рисунку 2.3 зображено взаємодію компонентів Redux..

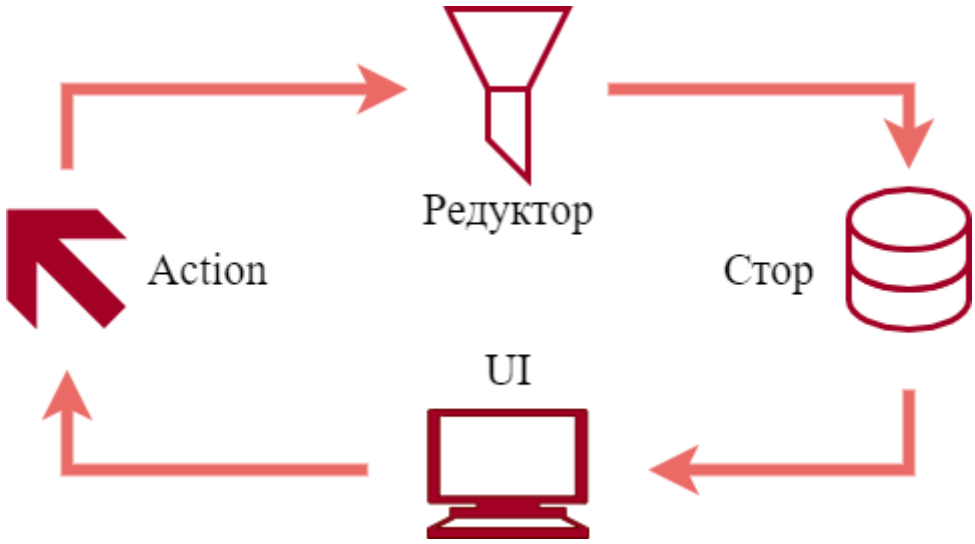


Рис. 2.3 – Три основні блоки Redux

Redux має 3 основні частини:

- Дії
- Редуктори

- Стор

2.2.1 Стор

Стор зберігає стан програми. Стор насправді є об’єктом, а не класом, хоча спочатку він може здаватися таким. Він також містить кілька додаткових речей, крім стану програми (наприклад, функції та інші об’єкти).

Хоча теоретично можна створити кілька сторів, це суперечить шаблону, якого дотримується Redux. За правилом створюється лише один стор для кожного додатка.

Стан у Redux має форму об’єкта JavaScript і його часто називають «деревом стану». В нього можна помістити будь-які значення, які потрібно зберегти, і можна вкласти їх стільки, скільки необхідно.

2.2.2 Дії

Дії - це звичайні об’єкти JavaScript, які описують, що сталося, але не описують, як змінюється стан програми.

Ми просто відправляємо їх у наш екземпляр стору, коли хочемо оновити стан нашого додатку. Рештою займаються редуктори.

Важливо пам’ятати, що Redux вимагає, щоб наші об’єкти дій містили поле типу. Це поле використовується для опису дії, яка відправляється, і зазвичай це константа, яка експортується з файлу.

Усі інші поля в об’єкті дії є необов’язковими і залежать від проєкту.

2.2.3 Редуктори

Редуктори - це чисті функції, які визначають, як змінюється стан програми. Іншими словами, вони використовуються для перерахунку нового стану програми або, принаймні, її частини.

Щоразу, коли ми відправляємо дію до нашого «магазину», вона передається редуктору.

Функція редуктора приймає два параметри: попередній стан програми, дію, що відправляється, і повертає новий стан програми. Іншими словами, редуктор обчислює новий стан додатку на основі надісланої дії (та її типу).

Щоб вирішити питання зі складністю редуктора, краще розділити їх на декілька простіших редукторів, а пізніше поєднати їх допоміжною функцією Redux, яка називається combReducers.

Основний редуктор умовно називають «кореневим редуктором». [12]

2.3 SASS

SASS розшифровується як Syntaxically Awesome Stylesheets. SASS в основному є лише розширенням CSS, яке допомагає нам писати більш гнучкі стилі.

SASS допомагає нам зробити більш складні таблиці стилів зрозумілішими та простішими в обслуговуванні. Завдяки таким функціям, як змінні, комбінації, вкладеність, наслідування, код є більш організованим, що дозволяє нам працювати швидше.

На рисунку 2.4 зображено логотип технології.

Рис. 2.4 – Логотип SASS

Коли ми пишемо в SASS, браузери не розуміють наш код, оскільки ми пишемо не CSS, тому нам потрібно використовувати компілятор для компіляції нашого SASS -коду в CSS.

Змінні в SASS є потужними, оскільки вони дозволяють швидко змінювати код. При визначенні змінної ми зберігаємо всередині неї певне значення, допустимі значення для змінних включають числа, рядки, кольори, null, списки та карти.

Щоб оголосити змінну в SASS, вам потрібен символ $, а потім ім'я змінної. Розглянемо на прикладі:

```
$blue: #3498db;
```

SASS дозволяє нам використовувати правила CSS для вкладання одне в одне. Вкладеність - це чудовий спосіб організувати та структурувати ваш CSS і утримати вас від зайвих повторень. Розглянемо приклад вкладеності:

```
ul {
  list-style: none;
  li {
    padding: 15px;
    display: inline-block;
    a {
      text-decoration: none;
      font-size: 14px;
      color: #fff;
    }
  }
}
```

Наслідування є однією з найкорисніших функцій у SASS, і використовуючи розширення, можна передати набори властивостей CSS від одного об’єкту до іншого. Розширення слід використовувати, коли

нам потрібні подібні стилі елементів, які все ще відрізняються деякими деталями.

Комбінації - ще одна чудова особливість SASS. По суті, вони дозволяють робити групи декларацій CSS, які необхідно повторно використовувати для веб-сайту. Можна передавати значення як аргументи, що дозволяє нашому змішуванню бути більш гнучким. [13]

2.4 Bootstrap

Bootstrap - це безкоштовний фреймворк з відкритим кодом для створення веб-сайтів та веб-додатків. Це найпопулярніший фреймворк HTML, CSS та JS для розробки адаптивних проєктів в Інтернеті.

На рисунку 2.5 зображено логотип фреймворку.

Рис. 2.5 – Логотип Bootstrap

Оскільки Інтернет дедалі більше розвивається до адаптивного дизайну, веб-розробники можуть стати зіткнутись із проблемою адаптивності. Bootstrap може набагато полегшити ситуацію. Фреймворк дозволяє створювати адаптивні веб-сайти без необхідності робити біт "реагування".

Головне правило Bootstrap - один фреймвок для будь-якого пристрою. Це пов’язано з тим, що веб-сайти, побудовані за допомогою Bootstrap, автоматично масштабуватимуться між пристроями - незалежно від того, є пристрій мобільним телефоном, планшетом, ноутбуком, настільним комп’ютером, зчитувачем екрану тощо.

Bootstrap - мобільний. Це означає, що він в першу чергу розроблений для мобільних пристроїв, а потім масштабується (на від міну від того, що призначений для настільних ПК, а потім намагається зменшити його до мобільних пристроїв).

Bootstap включає такі компоненти, як кнопки, панелі навігації, випадаючі меню, поля сповіщень тощо. У більшості випадків можна використовувати компонент, просто використовуючи відповідне ім'я класу.

Однією з головних переваг таких фреймворків, як Bootstrap, є те, що вони можуть сприяти пришвидшенню часу розробки, зберігаючи при цьому якість і послідовність на сайті. Вам більше не потрібно перепроєктувати кожен елемент. І вам не потрібно витрачати години, намагаючись зробити так, щоб усе виглядало та працювало прямо в браузерах, платформах та пристроях. Використовуючи Bootstrap, вся важка робота виконується за вас.

Крім того, хоча Bootstrap поставляється зі своїм набором стилів, їх легко замінити. Ви не заблоковані в "дизайні Bootstrap". Є можливість використовувати будь-які вибрані вами компоненти Bootstrap, одночасно додаючи власні. Є тисячі веб-сайтів, побудованих на Bootstrap, але зі своїм власним дизайном.

Bootstrap можна використовувати для створення веб-сайтів будь-якого масштабу, від невеликих блогів до великих корпоративних веб-сайтів. До організацій, які використовують Bootstrap, належать NASA, FIFA, Newsweek, VOGUE та багато інших. [14]

2.5 NodeJS та ExpressJS

Node.js - це безкоштовне середовище виконання платформ JavaScript з відкритим вихідним кодом, яке дозволяє розробникам писати

інструменти командного рядка та серверні сценарії поза браузером. Це популярний інструмент практично для будь-якого типу проєкту.

На рисунку 2.6 зображено логотип NodeJS.

Рис. 2.6 – Логотип NodeJS

Node.js запускає движок JavaScript V8, та ядро Google Chrome поза браузером. Це дозволяє Node.js бути дуже продуктивним.

Додаток Node.js запускається в одному процесі без створення нового потоку для кожного запиту. Node.js надає набір асинхронних примітивів вводу-виводу у своїй стандартній бібліотеці, які перешкоджають блокуванню коду JavaScript, і загалом бібліотеки в Node.js записуються з використанням неблокуючих парадигм, роблячи поведінку блокування винятком, а не нормою.

Коли Node.js виконує операцію вводу-виводу, наприклад, зчитування з мережі, доступ до бази даних або файлової системи, замість того, щоб блокувати потік і витрачати очікування циклів процесора, Node.js відновить роботу, коли відповідь повернеться. Це дозволяє Node.js обробляти тисячі одночасних з'єднань з одним сервером, не вкладаючи тягар управління паралельністю потоків, що може провокувати виникнення помилок.

Node.js має унікальну перевагу, оскільки мільйони розробників інтерфейсів, які пишуть JavaScript для браузера, тепер можуть писати код на стороні сервера на додаток до коду на стороні клієнта без необхідності вивчати зовсім іншу мову. [15]

У Node.js нові стандарти ECMAScript можна використовувати без проблем, оскільки вам не доведеться чекати, поки всі ваші користувачі оновлять свої браузери - вам належить вирішити, яку версію ECMAScript використовувати, змінивши версію Node.js, а також можна ввімкнути певні експериментальні функції, запустивши Node.js із прапорами.

Node.js має стандартну бібліотеку, включаючи першокласну підтримку роботи в мережі. Метод createServer () http створює новий HTTP-сервер і повертає його. Сервер налаштований на прослуховування вказаного порту та імені хосту. Коли сервер готовий, викликається функція зворотного виклику, в цьому випадку інформуючи нас про те, що сервер працює.

Щоразу, коли отримується новий запит, викликається подія, що забезпечує два об'єкти: запит (об'єкт http.IncomingMessage) та відповідь (об'єкт http.ServerResponse). Ці 2 об'єкти є важливими для обробки виклику HTTP. Перший містить деталі запиту. У цьому простому прикладі це не використовується, але ви можете отримати доступ до заголовків запиту та даних запиту. Другий використовується для повернення даних абоненту.

Нижче наведено деякі важливі функції, які роблять Node.js першим у виборі архітекторів програмного забезпечення:

- Асинхронність та керованість подіями - всі API бібліотеки Node.js є асинхронними, тобто не блокуючими. По суті, це означає, що сервер на основі Node.js ніколи не чекає, поки API поверне дані. Сервер переходить до наступного API після його виклику, і механізм сповіщення про події Node.js допомагає серверу отримати відповідь від попереднього виклику API.
- Однопоточність, але масштабованість - Node.js використовує однопоточну модель із циклічною подією. Механізм подій

допомагає серверу реагувати неблокуючим способом і робить сервер масштабованим на відміну від традиційних серверів, які створюють обмежені потоки для обробки запитів.

- Без буферизації - програми Node.js ніколи не буферизують будь-які дані. Ці програми просто виводять дані шматками.
- Ліцензія - Node.js випускається під ліцензією MIT.

Список, програм та компаній, які використовують Node.js, включає eBay, General Electric, GoDaddy, Microsoft, PayPal, Uber, Wikipins, Yahoo!, та Yammer. [16]

Express - це мінімальна та гнучка структура веб-додатків Node.js, яка забезпечує надійний набір функцій для веб- і мобільних додатків. Це фреймворк з відкритим кодом, розроблений і підтримуваний фондом Node.js.

Express є найпопулярнішим веб-фреймворком Node і є базовою бібліотекою для ряду інших популярних веб-фреймворків Node. Він забезпечує механізми для:

- Написання обробників запитів HTTP за різними шляхами URL.
- Інтеграції з механізмами візуалізації "view", щоб генерувати відповіді, вставляючи дані в шаблони.
- Встановлення загальних налаштувань веб-додатків, такі як порт для підключення та розташування шаблонів, які використовуються для надання відповіді.
- Додавання додаткової обробки запитів "проміжного програмного забезпечення" в будь-яку точку конвеєру для обробки запитів.

Хоча сам Express є досить мінімалістичним, розробники створили сумісні пакети проміжного програмного забезпечення для вирішення

практично будь-якої проблеми веб-розробки. Існують бібліотеки для роботи з файлами кукі, сеансами, логінами користувачів, параметрами посилання, даними POST, заголовками безпеки та багатьма іншими.

Express допомагає відповідати на запити за допомогою підтримки маршруту, щоб мати змогу писати відповіді на конкретні URL-адреси. Він підтримує кілька механізмів шаблонування для спрощення генерації HTML.

Експрес не є безперспективним. Ви можете вставити майже будь-яке сумісне проміжне програмне забезпечення, яке вам подобається, у ланцюжок обробки запитів, майже в будь-якому порядку, який вам подобається. Ви можете структурувати програму в одному файлі або декількох файлах, використовуючи будь-яку структуру каталогів. [17]

Табл. 2.1 – Порівняння NodeJS та ExpressJS

NodeJS	**ExpressJS**
Використовується для створення як інтерфейсу, так і серверної частини веб-програми.	Це фреймворк node.js, який використовується для створення серверної бази веб-програми.
Написаний з використанням різних мов програмування, таких як JavaScript, C та C ++.	Написаний лише з використанням JavaScript.
Це не веб-фреймворк.	Це веб-фреймворк.
Він підтримує інші мови, такі як TypeScript, CoffeeScript та Ruby.	Він підтримує JavaScript.
Він підходить для невеликих проєктів.	
Він підтримує архітектуру MVC.	

2.6 MongoDB

MongoDB - це база даних NoSQL, яка зберігає дані у формі пар ключ-значення. Це база даних з відкритим кодом, яка забезпечує високу продуктивність та масштабованість, а також моделювання даних та управління великими наборами даних у корпоративній програмі.

На рисунку 2.6 зображено логотип MongoDB.

Рис 2.7 – Логотип MongoDB

MongoDB також надає функцію автоматичного масштабування. Оскільки MongoDB є крос-платформною базою даних і може бути встановлена на різних платформах, таких як Windows, Linux тощо.

MongoDB була розроблена Еліотом Горовіцем та Дуайтом Меррiманом у 2007 році, коли вони мали проблеми з масштабованістю реляційної бази даних під час розробки корпоративних веб-додатків у своїй компанії DoubleClick. За словами Дуайта Меррімана, одного з розробників MongoDB, ця назва бази даних походить від слова «humongous» для підтримки ідеї обробки великої кількості даних.

Окрім більшості функцій NoSQL за замовчуванням, MongoDB має деякі інші, дуже важливі та корисні функції:

- MongoDB забезпечує високу продуктивність. Операції вводу-виводу менші за реляційні бази даних завдяки підтримці вбудованих документів (моделі даних), а також запити Select швидші, оскільки індекси в MongoDB підтримують швидкі запити.

- MongoDB має багату мову запитів, що підтримує всі основні операції CRUD. Мова запитів також забезпечує хороші функції пошуку тексту та агрегації.
- Функція автоматичної реплікації MongoDB призводить до високої доступності. Вона забезпечує автоматичний механізм відновлення після відмови, оскільки дані відновлюються за допомогою резервної копії (копії), якщо сервер виходить з ладу.
- Шардінг (метод розподілу даних між кількома машинами) - це основна особливість MongoDB. Горизонтальна масштабованість можлива завдяки шардингу.
- MongoDB підтримує кілька механізмів зберігання. Механізми зберігання керують способом збереження даних у пам'яті тана диску. [18]

Різниця між традиційними базами даних та MongoDB продемострована у таблиці 2.2.

Табл 2.2 - Відмінність між термінами СУБД та MongoDB (початок)

СУБД	**MongoDB**	**Різниця**
База даних	База даних	-
Таблиця	Колекція	У СУБД таблиця містить стовпці та рядки, які використовуються для зберігання даних, тоді як у MongoDB ця сама структура відома як колекція. Колекція містить документи, які в свою чергу містять поля, які в свою чергу є парами ключ-значення.

Табл 2.2 - Відмінність між термінами СУБД та MongoDB

(продовження)

СУБД	MongoDB	Різниця
Рядок	Документ	У СУБД рядок представляє один, неявно структурований елемент даних у таблиці. У MongoDB дані зберігаються в документах.
Стовпець	Поле	У СУБД стовпець позначає набір значень даних. Вони в MongoDB відомі як Поля.
Індекс	Індекс	-

У MongoDB дані зазвичай зберігаються в одній колекції, але розділяються за допомогою вбудованих документів. Отже, в MongoDB немає концепції приєднання.

Дані в MongoDB мають гнучку схему. На відміну від баз даних SQL, де перед вставкою даних вам потрібно оголосити схему таблиці, колекції MongoDB не застосовують структуру документа. Така гнучкість робить MongoDB настільки потужним.

ВИСНОВКИ ДО РОЗДІЛУ 2

В другому розділі були розглянуті технології, які будуть застосовані для створення веб-додатку для керування проєктами. Додаток буде побудований на основі набору технологій MERN, що включає базу даних MongoDB, ExpressJS та NodeJS для серверної частини веб-додатку, React/Redux – для клієнтської частини. Окрім цього, додатково буде застосовано препроцесор SASS та фреймворк Bootstrap.

Було розглянуто технологію React - це гнучка бібліотека JavaScript для побудови користувальницьких інтерфейсів. Також описано технологію Redux, яка є популярною бібліотекою JavaScript для управління станом програми.

Було розглянуто технологію SASS, яка забезпечує наступні механізми: змінні, вкладеність, комбінації та наслідування селектора.

Було розглянуто технологію Bootstrap. Він містить шаблони дизайну на основі CSS та JavaScript для типографіки, форм, кнопок, навігації та інших компонентів інтерфейсу.

Були розгнянуті серверні технології NodeJS та ExpressJS. Node.js - це технологія із відкритим вихідним кодом, міжплатформене та внутрішнє середовище виконання JavaScript, яке працює на рушії V8 і виконує код JavaScript поза веб-браузером. ExpressJS - це внутрішній фреймворк веб-додатків для Node.js, випущений як безкоштовне програмне забезпечення з відкритим кодом під ліцензією MIT. Він призначений для створення веб-додатків та API.

Була розглянута технологія MongoDB - це база даних, яка зберігає дані у схожих на JSON документах з динамічною схемою. Завдяки цьому існує можливість зберігати свої записи, не турбуючись про структуру даних, таку як кількість полів або типи полів для зберігання значень. Також було розглянуто різницю між традиційними СУБД та MongoDB.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

Враховуючи досліджені аналоги та технології у попередніх розділах можна перейти до етапу розробки веб-додатку. З метою створення гнучкого програмного забезпечення необхідно описати основні частини розробленого продукту, основні моделі для бази даних, функціональні блоки та схеми їх взаємодії.

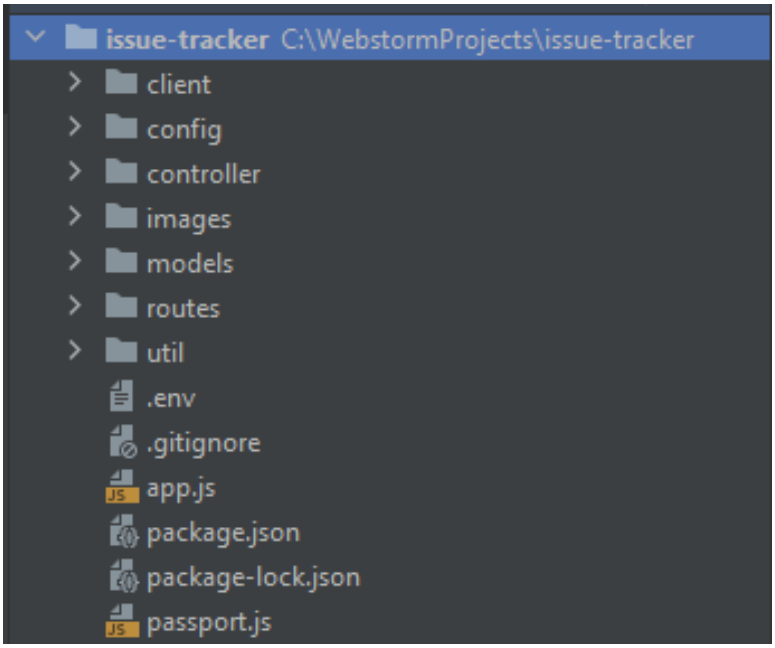


Рис. 3.1 – Структура проєкту

На рисунку 3.1 продемонстровано основні частини з яких складається проєкт. Клієнтська частина програми створена в папці *client*. Основні модулі серверної частини описані в папці *controller*, а їх взаємозв'язок в папці *routes*. База даних описана в папці *models*. Поєднання логіки клієнтської та серверної частини описано в головному файлі програми *app.js*. Далі детальніше розглянемо реалізацію серверної та клієнтської частини веб-додатку.

3.1 Розробка моделей бази даних

Для зручного керування даними в системі було розроблено 6 основних моделей бази даних: модель користувача, модель команди, модель проєкту, модель задачі, модель коментарів та модель чату.

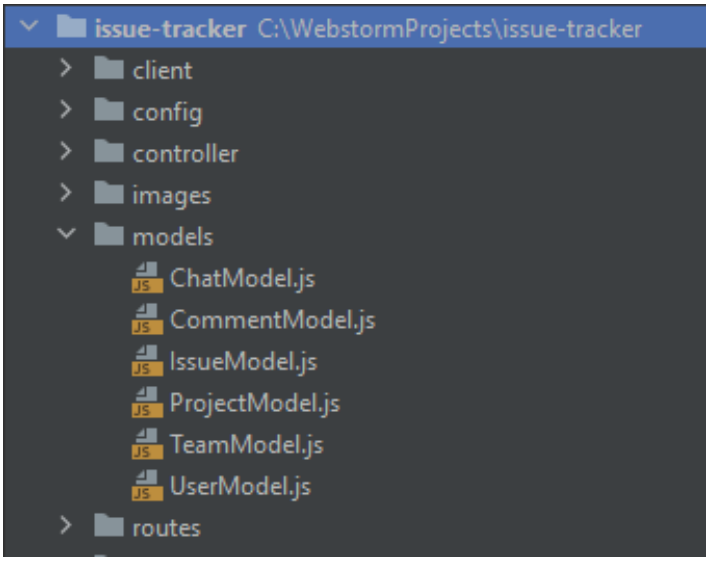


Рис. 3.2 – Файли моделей бази даних

Далі детальніше розглянемо необхідні поля з яких складається кожна з моделей.

Відповідно до рисунку 3.2 поля які необхідні для збереження даних користувача розміщені у файлі під назвою *UserModel.js* та описані в таблиці 3.1.

Табл. 3.1 – Поля моделі користувача (початок)

Назва поля	Тип даних	Опис
userId	*ObjectId*	Зберігає унікальний ідентифікатор користувача
username	*String*	Зберігає ім’я користувача
email	*String*	Зберігає електронну пошту користувача
password	*String*	Зберігає пароль користувача
createdAt	*Date*	Зберігає дату створення акаунту користувача

Табл. 3.1 – Поля моделі користувача (продовження)

Назва поля	Тип даних	Опис
projects	*Array*	Зберігає проєкти, до яких під'єднано користувача
teams	*Array*	Зберігає команди до яких входить користувач

Відповідно до рисунку 3.2 поля які необхідні для збереження даних про команду розміщені у файлі під назвою *TeamModel.js* та описані в таблиці 3.2.

Табл. 3.2 – Поля моделі команди

Назва поля	Тип даних	Опис
teamId	*ObjectId*	Зберігає унікальний ідентифікатор команди
name	*String*	Зберігає назву команди
password	*String*	Зберігає пароль команди
users	*Array*	Зберігає користувачів, які входять у команду
admins	*Array*	Зберігає користувачів, які можуть керувати командою
projects	*Array*	Зберігає проєкти до яких підключено команду
createdBy	*ObjectId*	Зберігає ім'я користувача, яким команду було створено
createdAt	*Date*	Зберігає дату створення команди

Відповідно до рисунку 3.2 поля які необхідні для збереження даних про проєкт розміщені у файлі під назвою *ProjectModel.js* та описані в таблиці 3.3.

Табл. 3.3 – Поля моделі проєкту

Назва поля	Тип даних	Опис
projectId	*ObjectId*	Зберігає унікальний ідентифікатор проєкту
name	*String*	Зберігає назву проєкту
description	*String*	Зберігає опис проєкту
labels	*Array*	Зберігає ключові слова проєкту
issues	*Array*	Зберігає задачі/баги які прив'язані до проєкту
createdBy	*ObjectId*	Зберігає ім'я користувача, яким проєкт було створено
createdAt	*Date*	Зберігає дату створення проєкту
team	*ObjectId*	Зберігає команди, які працюють над проєктом
archived	*Boolean*	Зберігає стан проєкту
updatedAt	*Date*	Зберігає дату оновлення проєкту

Відповідно до рисунку 3.2 поля які необхідні для збереження даних про задачі в проєктах розміщені у файлі під назвою *IssueModel.js* та описані в таблиці 3.4.

Табл. 3.4 – Поля моделі задачі (початок)

Назва поля	Тип даних	Опис
issueId	*ObjectId*	Зберігає унікальний ідентифікатор задачі
name	*String*	Зберігає назву задачі
description	*String*	Зберігає опис задачі
status	*Object*	Зберігає поточний статус задачі
labels	*Array*	Зберігає ключові слова проєкту
comments	*Array*	Зберігає коментарі до задачі

Табл. 3.4 – Поля моделі задачі (продовження)

Назва поля	Тип даних	Опис
assignees	*Array*	Зберігає користувачів, які працюють над задачею
project	*ObjectId*	Зберігає проєкт до якого відноситься задача
createdBy	*ObjectId*	Зберігає ім’я користувача, яким задачу було створено
createdAt	*Date*	Зберігає дату створення проєкту
completedOn	*Date*	Зберігає дату завершення виконання задачі
updates	*Date*	Зберігає дату оновлення задачі

Відповідно до рисунку 3.2 поля які необхідні для збереження даних про коментарі в задачах розміщені у файлі під назвою *CommentModel.js* та описані в таблиці 3.5.

Табл. 3.5 – Поля моделі коментарів

Назва поля	Тип даних	Опис
commentId	*ObjectId*	Зберігає унікальний ідентифікатор коментаря
comment	*String*	Зберігає текст коментаря
issue	*ObjectId*	Зберігає задачу, в якій був залишений коментар
createdBy	*ObjectId*	Зберігає ім’я користувача, яким коментар було створено
createdAt	*Date*	Зберігає дату створення коментаря
updatedAt	*Date*	Зберігає дату оновлення коментаря

Відповідно до рисунку 3.2 поля які необхідні для збереження даних про чати команд розміщені у файлі під назвою *ChatModel.js* та описані в таблиці 3.6.

Табл. 3.6 – Поля моделі чату

Назва поля	Тип даних	Опис
message	*String*	Зберігає текст повідомлення
sender	*ObjectId*	Зберігає опис задачі
teamID	*ObjectId*	Зберігає команду до якої прив'язаний чат
postedAt	*Date*	Зберігає дату створення повідомлення
type	*String*	Зберігає тип повідомлення

Для того щоб більш детально зрозуміти взаємодію моделей БД між собою було розроблено візуальну схему яка представлена у Додатку 1.

3.2 Розробка серверної частини веб-додатку

Бекенд – це основа веб-додатку. Він збирає дані із зовнішніх серверів та програм, фільтрує цю інформацію, після чого повертає назад на веб-сайт для обробки запитів користувачів.

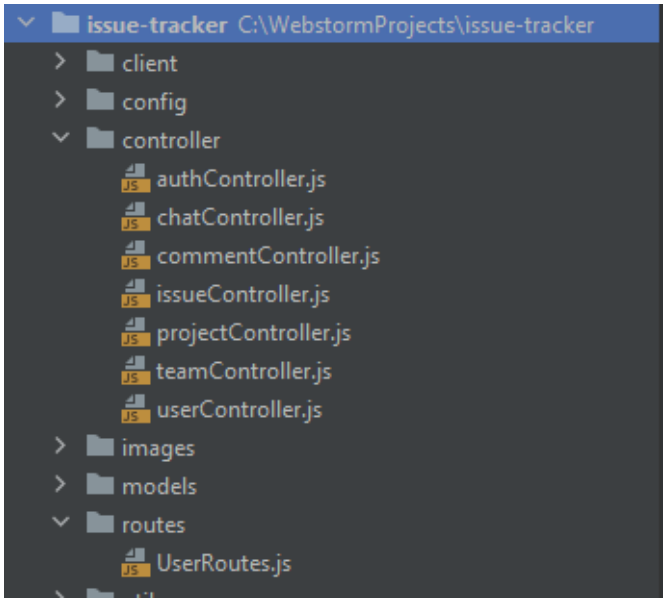


Рис. 3.3 – Файли контролерів API

Для покриття всіх можливих сценаріїв взаємодії було розроблено API який складається із 7 контролерів продемонстрованих на рисунку 3.3: контролер користувача, контролер команди, контролер проєкту, контролер задач, контролер коментарів, контролер чатів та контролер аутентифікації.

Основні ендпоінти, які допомагають взаємодіяти з базою даних веб-додатку описані у файлі *UserRoutes.js*. Нижче розглянемо кожен ендпоінт:

1. Вхід у систему

   POST /login

   Параметри:

   login: string

   password: string

   Результат:

   Статус 200 – успішний вхід в систему

2. Аутентифікація

   POST /token

   Параметри:

   token: string

   user: object

   Результат:

   Статус 200 – успішний вхід в систему

   Статус 401 – помилка a

3. Реєстрація

   POST /signup

   Параметри:

   username: string

   email: string

   password: string

   confirmPassword: string

Результат:

Статус 200 – успішно зареєстрований

Статус 400 – неправильні вхідні дані

Статус 403 – користувач/емейл вже існує

Статус 500 – помилка під час створення користувача

4. Вихід із системи

POST /logout

Параметри:

token: string

Результат:

Статус 200 – успішний вихід із системи

5. Доступ до користувача

GET /user

Параметри:

token: string

Що повертає:

user: object

Результат:

Статус 500 – помилка доступу до даних юзера

6. Оновлення даних користувача

PUT /edit-profile

Параметри:

token: string

userId: objectId

username: string

image: string

email: string

password: string

createdAt: date

projects: object

teams: object

Результат:

Статус 200 – дані оновлено

Статус 500 – помилка оновлення даних юзера

7. Отримання чату

GET /chats/:teamID

Параметри:

token: string

Що повертає:

chats: object

Результат:

Статус 400 – помилка доступу до чату команди

8. Створення задачі

POST /issue

Параметри:

token: string

name: string

description: string

labels: string

projectId: integer

assignees: object

Результат:

Статус 200 – задача успішно створена

Статус 500 – помилка створення задачі

9. Оновлення даних задачі

PUT /issue/:issueId

Параметри:

token: string

name: string

description: string

labels: string

projectId: integer

assignees: object

Результат:

Статус 200 – задача оновлена

Статус 400 – інша помилка

Статус 404 – задачу не знайдено

Статус 500 – помилка оновлення даних задачі

10. Оновлення статусу задачі

PUT /issue/:issueId/status

Параметри:

token: string

status: string

issueId: integer

Результат:

Статус 200 – задача оновлена

Статус 400 – інша помилка

Статус 404 – задачу не знайдено

Статус 500 – помилка оновлення статусу задачі

11. Видалення задачі

DELETE /issue/:issueId

Параметри:

token: string

Результат:

Статус 404 – задачу не знайдено

Статус 500 – помилка видалення задачі

12.Отримання задачі

GET /issue/:issueId

Параметри:

token: string

Що повертає:

issue: object

Результат:

Статус 404 – задачу не знайдено

Статус 500 – помилка видалення задачі

13.Створення коментаря

POST /comment

Параметри:

token: string

issue: integer

comment: string

createdBy: string

createdAt: date

Результат:

Статус 200 – коментар успішно створений

Статус 400 – інша помилка

Статус 500 – помилка додавання коментаря

14.Редагування коментаря

PUT /comment/:commentId

Параметри:

token: string

```
issue: integer
comment: string
createdBy: string
createdAt: date
```

Результат:

```
Статус 200 – коментар успішно оновлений
Статус 400 – інша помилка
Статус 500 – помилка додавання коментаря
```

15.Видалення коментаря

DELETE /comment/:commentId

Параметри:

```
token: string
```

Результат:

```
Статус 404 – коментар не знайдено
Статус 500 – помилка видалення коментаря
```

16.Отримання проєкту

GET /project/:projectId

Параметри:

```
token: string
```

Що повертає:

```
project: object
```

Результат:

```
Статус 404 – проєкт не знайдено
Статус 500 – помилка пошуку прєкту
```

17.Отримання всіх проєктів створених одним користувачем

GET /projects

Параметри:

```
token: string
userId: integer
```

Що повертає:

```
projects: object
```

Результат:

```
Статус 500 – помилка пошуку проєктів
```

18.Отримання проєктів з архіву створених певним користувачем

GET /projects/archived

Параметри:

```
token: string
userId: integer
```

Що повертає:

```
projects: object
```

Результат:

```
Статус 500 – помилка видалення задачі
```

19.Створення проєкту

POST /project

Параметри:

```
token: string
name: string
description: string
labels: string
issues: object
createdBy: integer
createdAt: date
team: integer
archived: Boolean
```

Результат:

```
Статус 200 – проєкт успішно створений
Статус 400 – інша помилка
Статус 500 – помилка створення проєкту
```

20.Додавання ключових слів до проєкту

POST /project/:projectId/label/create

Параметри:

token: string

name: string

fontColor: string

backgroundColor: string

createdAt: date

Результат:

Статус 200 – ключове слово успішно створений

Статус 400 – ключове слово вже існує

Статус 500 – помилка додавання ключового слова

21.Редагування ключових слів проєкту

PUT /project/:projectId/label/:labelId/edit

Параметри:

token: string

name: string

fontColor: string

backgroundColor: string

createdAt: date

Результат:

Статус 200 – ключове слово успішно оновлене

Статус 400 – ключове слово вже існує

Статус 500 – помилка оновлення ключове слово

22.Додавання проєкту в архів

PUT /project/:projectId/archive/add

Параметри:

token: string

Результат:

Статус 200 – проєкт успішно доданий в архів

Статус 400 – інша помилка

Статус 500 – помилка додавання проєкту в архів

23.Повернення проєкту з архіву

PUT /project/:projectId/archive/remove

Параметри:

token: string

Результат:

Статус 200 – проєкт успішно повернений з архів

Статус 400 – інша помилка

Статус 500 – помилка повернення проєкту

24.Редагування проєкту

PUT /project/:projectId

Параметри:

token: string

name: string

description: string

labels: string

issues: object

createdBy: integer

createdAt: date

team: integer

archived: Boolean

Результат:

Статус 200 – проєкт успішно оновлений

Статус 400 – інша помилка

Статус 500 – помилка оновлення проєкту

25.Видалення ключових слів проєкту

DELETE /project/:projectId/label/:labelId

Параметри:

token: string

Результат:

Статус 400 – інша помилка

Статус 500 – помилка видалення ключового слова

26.Видалення проєкту

DELETE /project/:projectId

Параметри:

token: string

Результат:

Статус 401 – немає доступу до видалення

Статус 400 – інша помилка

Статус 500 – помилка видалення коментаря

27.Отримання всіх команд

GET /teams

Параметри:

token: string

userId: integer

Що повертає:

teams: object

Результат:

Статус 500 – помилка видалення задачі

28.Отримання певної команди

GET /team/:teamId

Параметри:

token: string

Що повертає:

team: object

Результат:

Статус 404 – команду не знайдено

Статус 500 – інша помилка

29.Отримання всіх проєктів певної команди

GET /team/projects/:teamId

Параметри:

token: string

Що повертає:

projects: object

Результат:

Статус 500 – помилка пошуку проєктів

30.Отримання всіх архівованих проєктів команди

GET /team/:teamId/projects/archived

Параметри:

token: string

Що повертає:

projects: object

Результат:

Статус 500 – помилка видалення задачі

31.Створення команди

POST /team

Параметри:

token: string

```
name: string
password: string
users: object
admins: object
projects: object
createdBy: integer
createdAt: date
```

Результат:

```
Статус 200 – команда успішно створена
Статус 500 – помилка створення команди
```

32.Додавання учасника в команду

POST /join/team

Параметри:

```
token: string
name: string
password: string
users: object
admins: object
projects: object
createdBy: integer
createdAt: date
```

Результат:

```
Статус 200 – учасник доданий в команду
Статус 400 – інша помилка
Статус 401 – учасник вже є в команді
Статус 500 – помилка додавання учасника
```

33.Видалення учасника з команди

PUT /leave/team/:teamId

Параметри:

```
token: string
name: string
password: string
users: object
admins: object
projects: object
createdBy: integer
createdAt: date
```

Результат:

```
Статус 200 – учасника видалено з команди
Статус 400 – адмін не може покинути команду
Статус 500 – помилка видалення учасника
```

34. Видалення команди

DELETE /team/:teamId

Параметри:

```
token: string
```

Результат:

```
Статус 500 – помилка видалення команди
```

3.3 Реалізація клієнтської частини веб-додатку

Розробка клієнтської частини відноситься до тієї галузі веб-розробки, яка фокусується на тому, що бачать користувачі. Вона передбачає перетворення коду, створеного на серверній частині, у графічний інтерфейс, забезпечуючи представлення даних у зручному для читання та зрозумілому форматі.

Розглянемо рисунок 3.4 на якому представлена структура клієнтської частини. Можна помітити, що вона складається з двох основних папок: *public* та *src*.

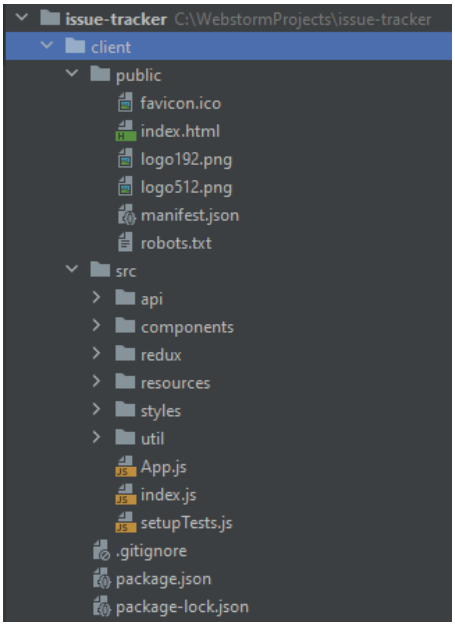


Рис. 3.4 – Структура підпроєкту клієнтської частини веб-додатку

В папці *public* міститься інформація, яку можна не приховувати від потенційних користувачів. Вся логіка клієнтської частини зберігається в папці *src*.

Найбільш важливим елементом клієнтської частини є інтерфейс для користувача. Він має бути зручним та чітко продуманим. Для комфортної взаємодії з веб-додатком було описано компоненти, які продемонстровані на рисунку 3.5. Також для кожного з цих компонентів було створено унікальні стилі, що показано на рисунку 3.6

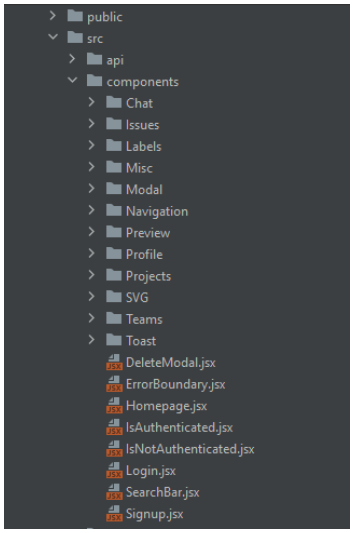


Рис. 3.5 – Файли, що зберігають структуру компонентів

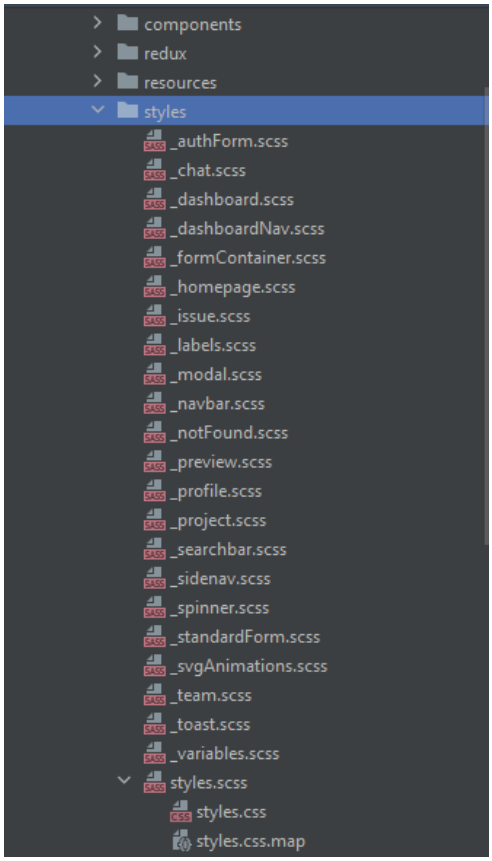


Рис. 3.6 – Файли, що зберігають стилі компонентів

Клієнтська частина веб-додатку повинна взаємодіяти з серверною. Саме це описано в файлі, що зображений на рисунку 3.7, під назвою *axiosApi.js*.

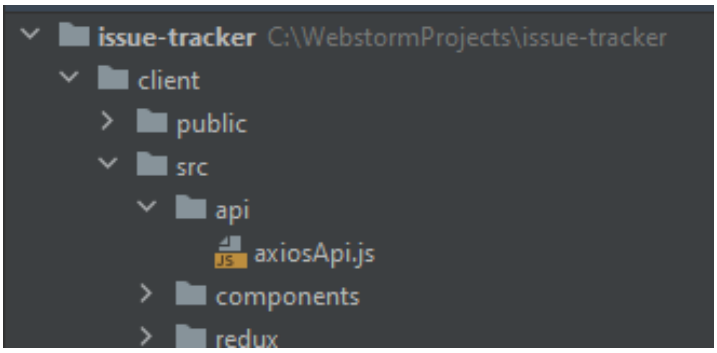


Рис. 3.7 – Файл, що поєднує клієнтську частину з серверною

В другому розділі було розглянуто технологію Redux, були описані основні елементи які необхідні для її реалізації. Дані елементи зберігаються в папці *redux*.

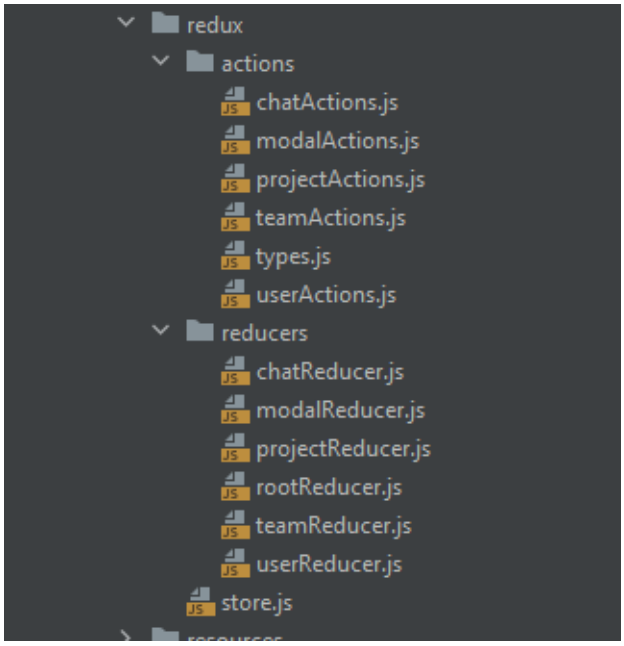


Рис. 3.8 – Опис технології Redux в проєкті

З рисунку 3.8 видно, що в проєкті присутні вище вказані дії (структури, що передають інформацію в стор), редуктори (визначають, як змінюється стан додатку відповідно до дій, які були відправлені в стор) та стор (поєднує дії та редуктори між собою).

Схема взаємодії серверної та клієнтської частини у системі зображена у Додатку 2.

З лістингом програми можна ознайомитись у додатку 4.

ВИСНОВКИ ДО РОЗДІЛУ 3

Під час роботи над третім розділом дипломного проєкту була розроблена база даних, серверна частина та клієнтська частина веб-додатку для керування проєктами по створенню програмного забезпечення по гнучкій методології.

Було розроблено базу даних веб-додатку, яка складається з 6 основних моделей: модель користувача, модель команди, модель проєкту, модель задачі, модель коментарів та модель чату.

Було розроблено серверну частину веб додатку, яка поєднує в собі взаємодію бази даних та API. Було розроблено відповідне API яке складається із 7 контролерів та 34 ендпоінтів: контролер користувача, контролер команди, контролер проєкту, контролер задач, контролер коментарів, контролер чатів та контролер аутентифікації. Також було розроблено взаємодію контролерів з базою даних.

Було розроблено клієнтську частину веб-додатку. Було спроєктовано інтерфейс користувача. Було сформовано і розроблено всі необхідні компоненти для зручної взаємодії з додатком: головні сторінки, модальні та спливаючі вікна, текстові поля, форми, кнопки і тд. Розроблено основні елементи Redux технології - дії, редуктори та стор.

Була протестована працездатність проєкту по загальному сценарію, та виключено основні помилки які може зробити користувач.

РОЗДІЛ 4. ДЕМОНСТРАЦІЯ ВЕБ-ДОДАТКУ

4.1 Розгортання програмного забезпечення

Щоб розгорнути серверну частину програмного забезпечення, попередньо повинно бути встановлено з офіційного сайту для необхідної ОС node.js 7+ та фреймворк express.js.

4.2 Огляд інтерфейсу системи

На рисунку 4.1 продемонстровано головну сторінку системи на яку користувач потрапляє коли заходить на сайт.

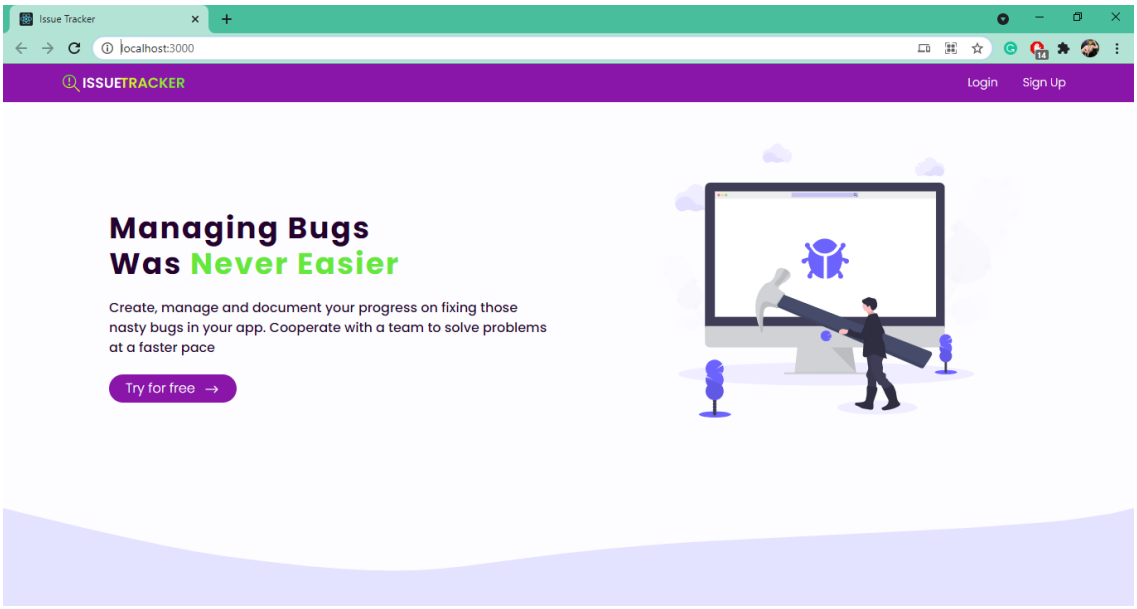


Рис. 4.1 – Головна сторінка веб-додатку

Якщо користувач раніше не користувався сервісом, тоді потрібно створити новий обліковий запис. Спочатку потрібно натиснути кнопку «Sign Up» у верхній правій частині екрану. Користувача буде переадресовано на сторінку реєстрації, де йому необхідно заповнити 4 поля: ім’я користувача, емейл, пароль та підтвердження паролю. Після чого потрібно натиснути кнопку «Sign Up» під формою реєстрації. Далі вхід в систему здійснюється використовуючи емейл і пароль.

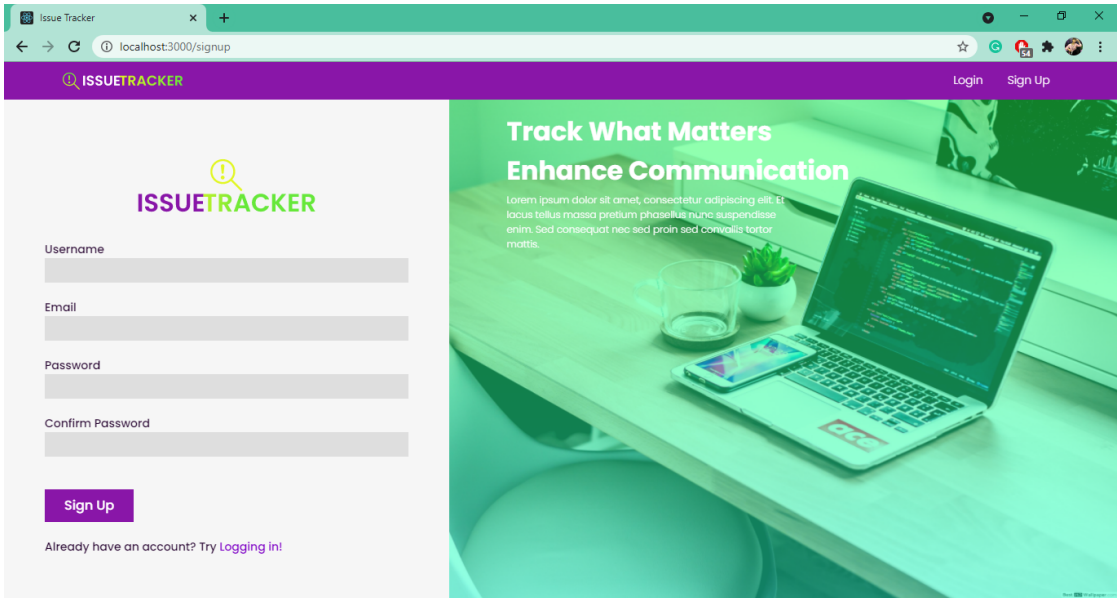


Рис. 4.2 – Сторінка реєстрації нового користувача

У випадку, якщо користувач вже існує в системі він натискає на кнопку «Login» та переадресовується на сторінку входу у систему, що зображена на рисунку 4.3. Після чого необхідно ввести коректний емейл та пароль, щоб аутентифікуватись та отримати доступ до проєктів та команд. У випадку, якщо користувач введе неправильний емейл та пароль, система покаже користувачу спливаюче вікно з помилкою «Wrong credentials» (не правильні облікові дані).

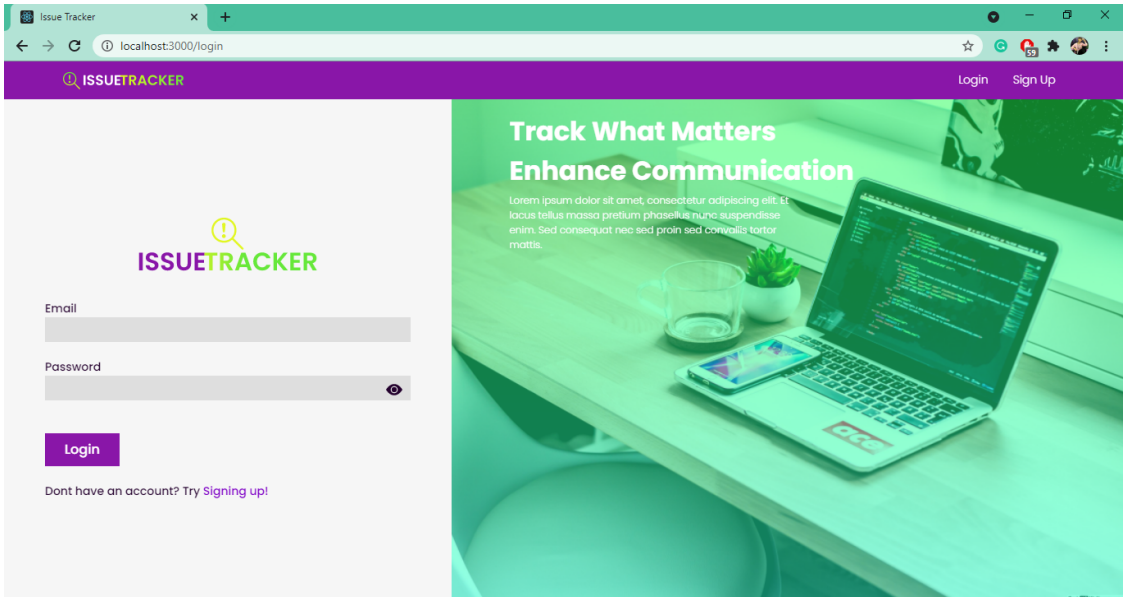


Рис. 4.3 – Сторінка логіну в систему

У випадку успішної авторизації, користувача буде переадресовано на стартову сторінку, що зображена на рисунку 4.4, з якої можна переглянути проєкти та команди.

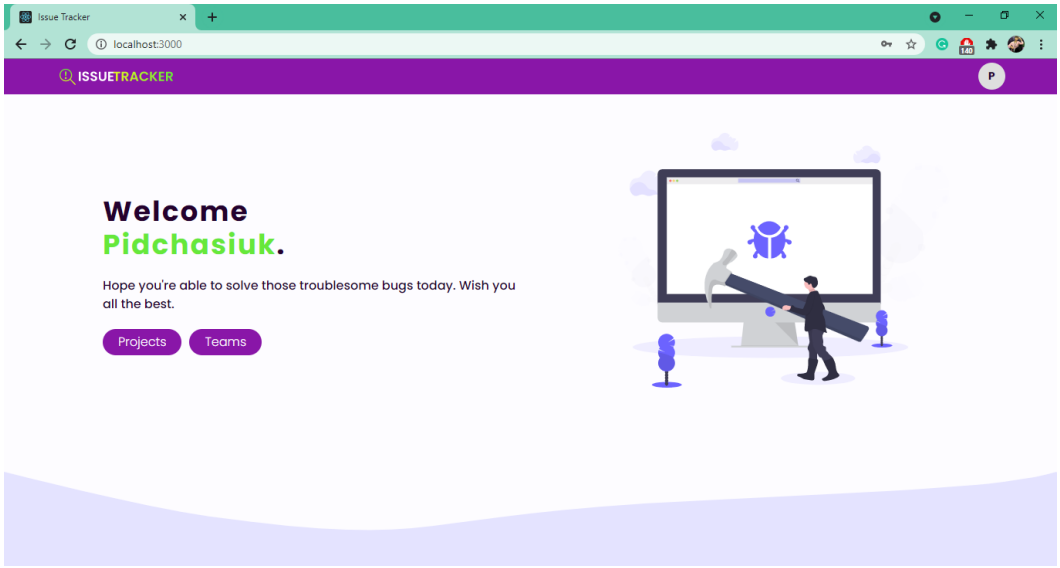


Рис. 4.4 – Стартова сторінка після входу системи

Щоб створити проєкт потрібно натиснути на кнопку «Projects». Користувач буде переадресований сторінку проєктів. Далі потрібно натиснути на кнопку «+» і відкриється форма для створення проєктів, що зображена на рисунку 4.4, де йому необхідно заповнити два поля: назва та опис проєкту. Щоб зберегти проєкт треба натиснути на кнопку «Create Project» внизу екрану.

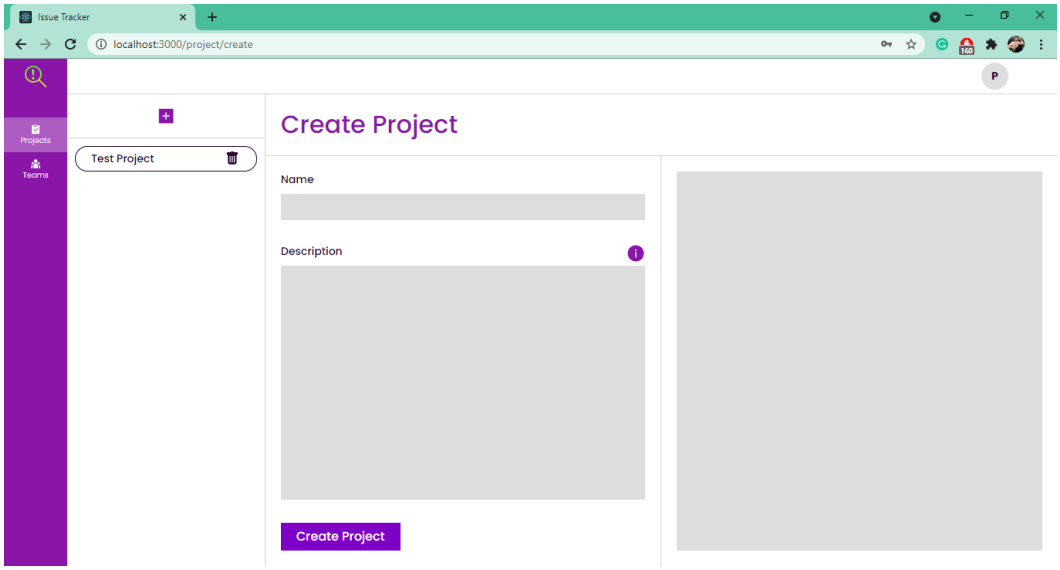


Рис. 4.5 – Сторінка створення нового проєкту

Після того як проєкт успішно створений, потрібно натиснути на назву в правій стороні екрану щоб переглянути інформацію про нього.

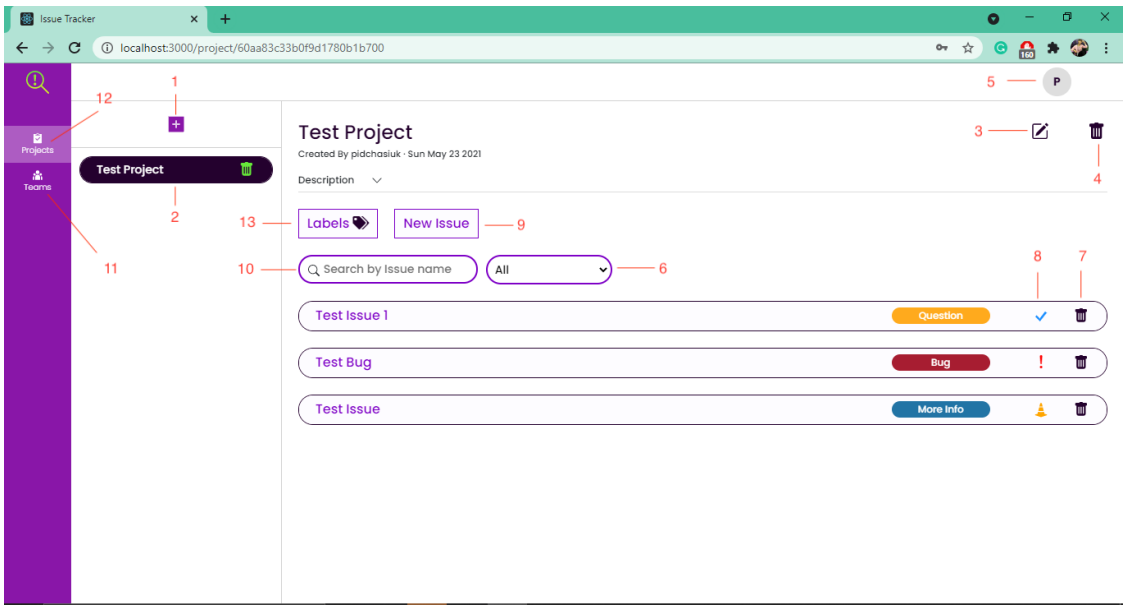


Рис. 4.6 – Перегляд існуючих задач в проєкті

На рисунку 4.6 представлений користувацький інтерфейс сторінки проєкту. Числами позначено головні елементи:

1. Кнопка створення нового проєкту
2. Список існуючих проєктів
3. Редагування проєкту
4. Видалення проєкту
5. Сторінка користувача
6. Фільтр задач по їх статусу
7. Видалити задачу з проєкту
8. Статус задачі
9. Створення нової задачі
10. Пошук по назві задачі
11. Вкладка команд
12. Вкладка проєктів
13. Редагування списку ключових слів

Щоб створити нову задачу потрібно натиснути на кнопку «New Issue». Відкриється форма створення нової задачі, що зображена на рисунку 4.7, з двома обов'язковими полями: назва задачі та її опис. Також представлено список ключових слів, яких характеризує тип задачі, але це поле не є обов'язковим. Список ключових слів можна змінити в налаштуваннях проєкту.

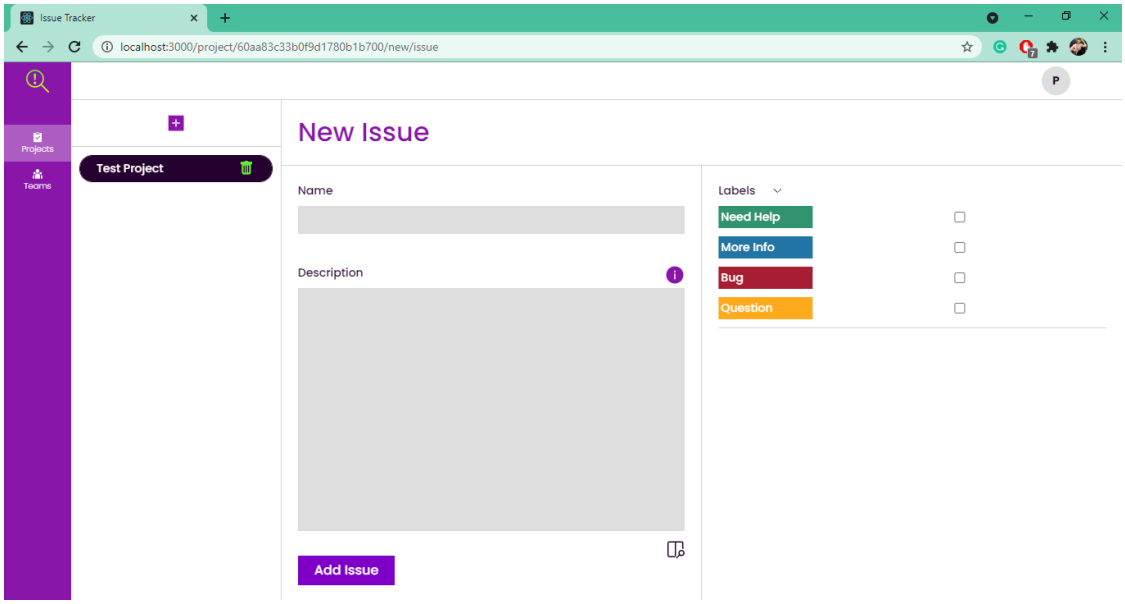


Рис. 4.7 – Сторінка створення нової задачі

Коли задачу успішно створена, її можна переглянути на сторінці всіх проєктів, якщо обрати відповідний проєкт. Сторінка задачі зображена на рисунку 4.8. На даній сторінці можна змінити статус задачі, редагувати задачу, видалити задачу та залишити коментар. Щоб залишити коментар під задачею, необхідно ввести текст в поле «New Comment» та натиснути на кнопку «Comment». Щоб змінити статус задачі потрібно натиснути на один із трьох елементів позначених цифрами:

1 – задача тільки створена, ще не була взята в роботу;

2 – задача знаходиться в роботі;

3 – задача виконана.

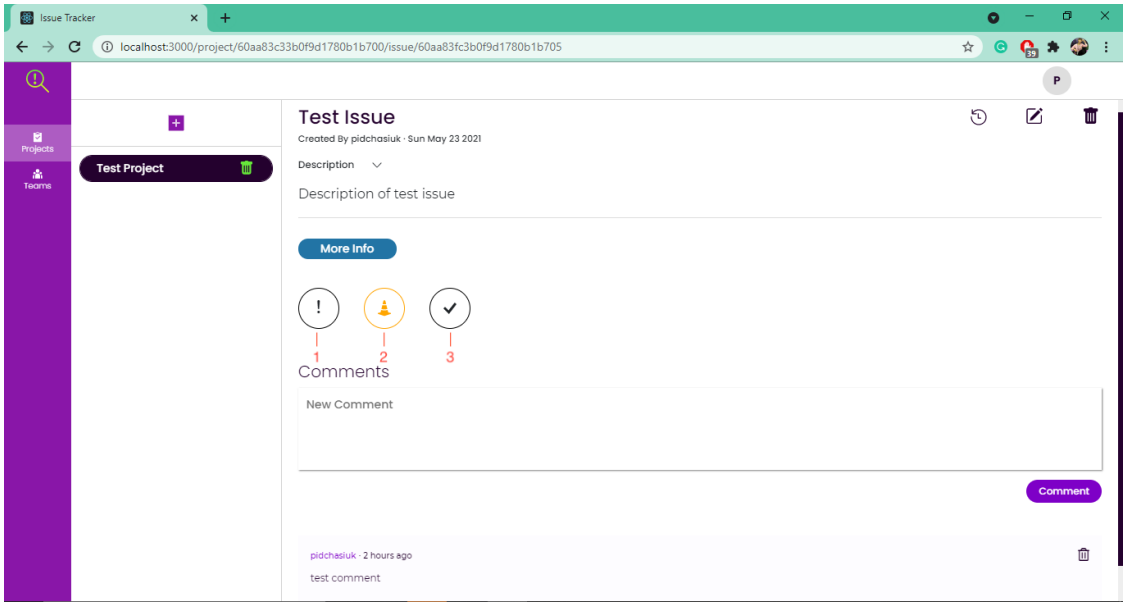


Рис. 4.8 – Сторінка задачі

Щоб створити нову команду потрібно натиснути на вкладку «Teams», далі на кнопку «+». Після цього відкриється форма для створення нової команди, в якій є два необхідних поля: назва команди та пароль, за яким можна буде приєднатись до команди.

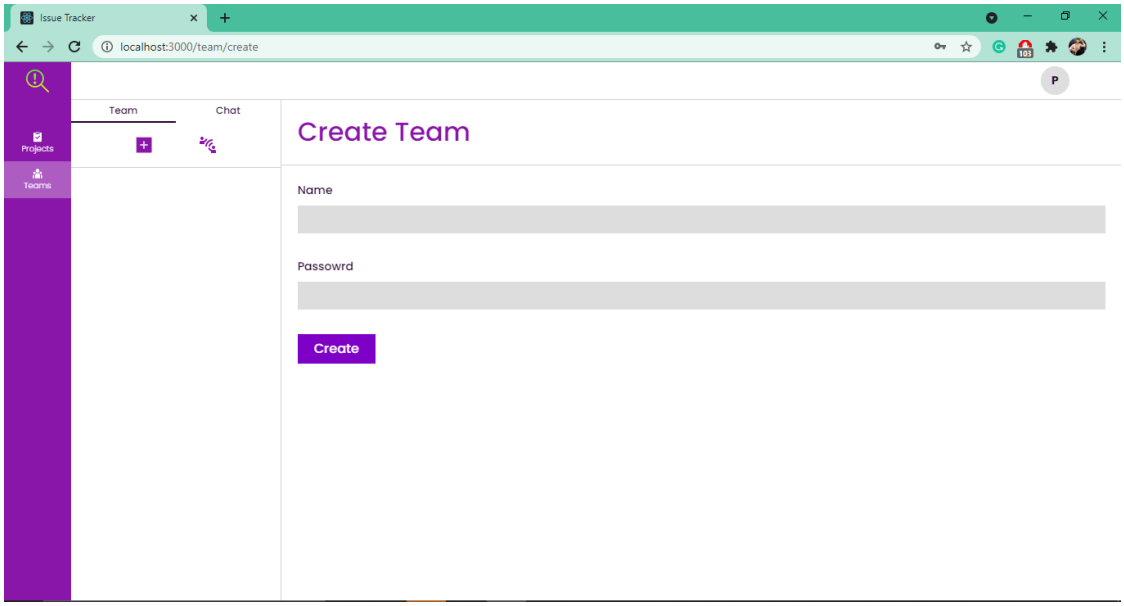


Рис. 4.9 – Сторінка створення нової команди

Щоб переглянути учасників команди потрібно натиснути на назву команди, тоді відкриється сторінка з інформацією про команду та її проєкти. Сторінка зображена на рисунку 4.10.

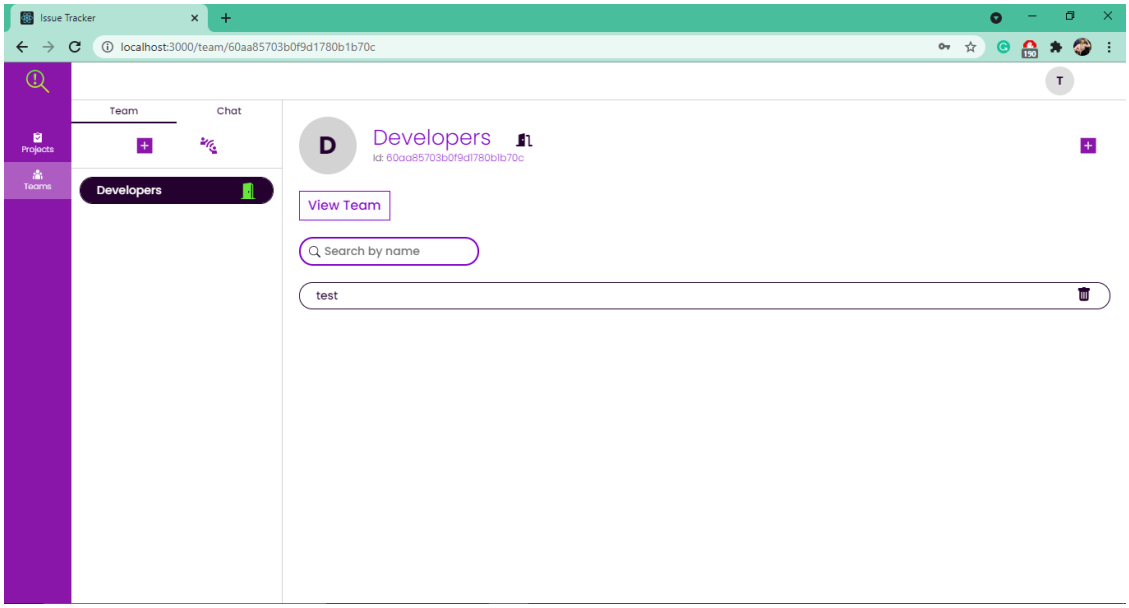


Рис. 4.10 – Сторінка команди

Щоб переглянути список учасників команди, необхідно натиснути на кнопку «View Team». Після цього відкриється сторінка, на якій буде продемонстровано ієрархію членів команди, рисунок 4.11.

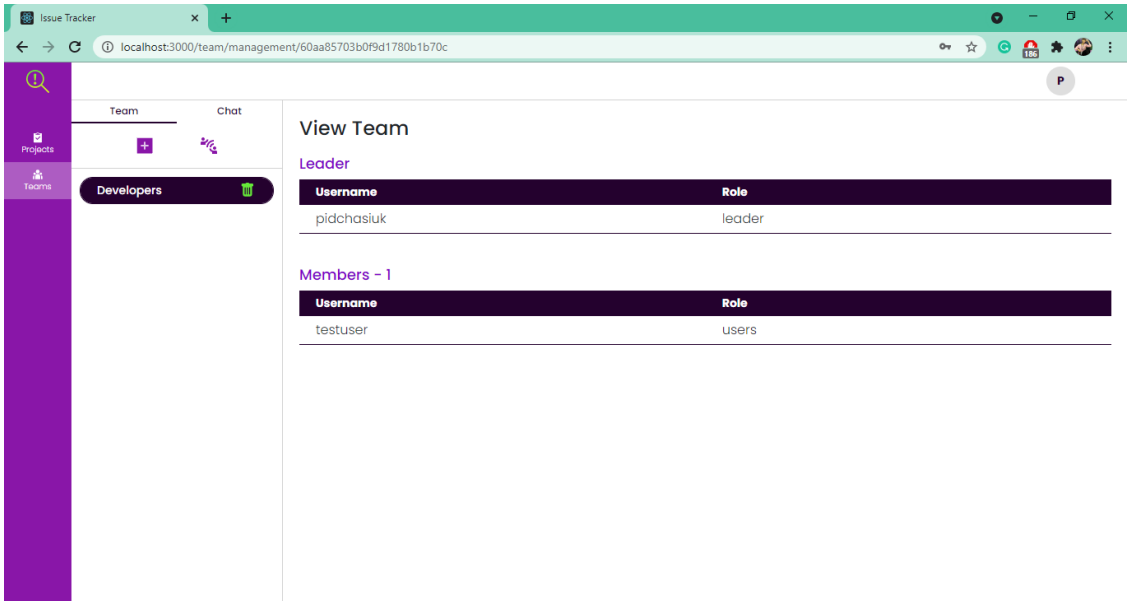


Рис. 4.11 – Сторінка учасників команди

Схематичний алгоритм роботи з веб-сервісом представлений у Додатку 3.

ВИСНОВКИ ДО РОЗДІЛУ 4

У даному розділі було розглянуто роботу веб-додатку по керуванню проєктами різних типів.

Було продемонстровано основні функції та сторінки сайту. Було описано:

- створення нової команди та додавання в неї нових учасників;
- створення нового проєкту;
- створення нової задачі;
- зміна статусу задачі;

Була представлена робота програми на прикладі створення тестової команди, проєкту та задач в цьому проєкті. Був детально описаний алгоритм роботи додатку. Всі зроблені дії були наведені у відповідних знімках екрану.

Даний веб-додаток дозволяє контролювати різні етапи створення проєктів. Розроблена система дає можливість відслідковувати всі задачі, які необхідно виконати, щоб запуск нового продукту був успішним. Користувач може працювати із системою інтуїтивно, тому що розроблений інтерфейс є достатньо зрозумілим та зручним.

ВИСНОВКИ

У даній дипломній роботі була поставлена задача розробки зручного веб-додатку для керування процесом створення нового програмного забезпечення за гнучкою методологією.

Було розглянуто різні підходи до створення програмного забезпечення: Скрам, Канбан та Вотерфол. З яких, до гнучких методологій створення ПЗ, можна віднести перші два методи. Було наведено моделі роботи для Скрам та Канбан. Також було описано позитивні та негативні сторони кожної з них.

Для отримання більш актуального результату було розглянуто декілька найпоаулярніших систем-аналогів зі схожим функціоналом. Серед них: Jira, Asana, Kanbanize, Bitrix24 та LeanKit. Кожен з додатків був детально описаний, було наведено основні позитивні та негативні сторони цих додатків. Також було зроблено порівняльну характеристику всіх аналогів між собою.

Було розглянуто стек технологій для веб-програмування MERN. Який складається з трьох фреймворків мови JavaScript (Node, Express та React) та хмарної бази даних MongoDB. Було розглянуто допоміжні технології для роботи над клієнтською частиною додатку, зокрема з інтерфейсом користувача. Фреймворк Bootstrap, що допомагає зробити сайт більш адаптивним у використанні. Та препроцесор SASS, за допомогою якого було спрощено роботу над стилями елементів.

В ході створення дипломного проєкту було реалізовано веб-додаток для керування процесом створення проєктів різної складності та у різних сферах. Додаток складається із серверної та клієнтської частини, взаємодію яких продемонстровано в додатку 2. Серверна частина побудована на базі 7 контролерів, які відповідають за дії над певними

елементами системи: авторизація, користувач, команда, задача, коментар, чат та проєкт.

В останньому розділі даної дипломної роботи продемонстрована працездатність програми на прикладі створення тестової команди, тестового проєкту та задач у ньому. Дана система дозволяє контролювати створення проєктів на різних етапах. Розроблений додаток дає можливість відслідковувати завдання, які необхідно виконати, щоб запуск нового продукту був успішним. Веб-додаток має простий та інтуїтивно зрозумілий інтерфейс. Було представлено знімки екрану в різних станах роботи додатку.

Варто зазначити, що розроблена система, на даному етапі, ще не може конкурувати на ринку, тому що має ряд не реалізованих функцій на клієнтській частині додатку. Одна з яких – спілкування в командних чатах. Не дивлячись на недоліки, результати виконаної роботи продемонстрували, що даний проєкт має практичне застосування і є актуальним рішенням посеред аналогів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Muslihat D. Agile Methodology: An Overview [Електронний ресурс] / Dinnie Muslihat. – 2018. – Режим доступу до ресурсу: https://zenkit.com/en/blog/agile-methodology-an-overview/.
2. Darrell K. Rigby. Embracing Agile [Електронний ресурс] / Darrell K. Rigby, Jeff Sutherland, Hirotaka Takeuchi / 2016 – Режим доступу до ресурсу: https://hbr.org/2016/05/embracing-agile.
3. Fowler M. The Agile Manifesto [Електронний ресурс] / M. Fowler, J. Highsmith. – 2001. – Режим доступу до ресурсу: http://users.jyu.fi/~mieijala/kandimateriaali/Agile-Manifesto.pdf.
4. Kate Eby. What's the Difference? Agile vs Scrum vs Waterfall vs Kanban [Електронний ресурс] / Kate Eby. – 2017. – Режим доступу до ресурсу: https://www.smartsheet.com/agile-vs-scrum-vs-waterfall-vs-kanban.
5. What is Kanban? [Електронний ресурс] – Режим доступу до ресурсу: digite.com/kanban/what-is-kanban/.
6. Scrum Vs. Kanban: Know the Difference [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://www.guru99.com/scrum-vs-kanban.html.
7. TechnologyAdvice: Agile Project Management Software [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://technologyadvice.com/agile-project-management-software/.
8. Introduction to MERN Technology Stack [Електронний ресурс] – Режим доступу до ресурсу: https://www.shareitsolutions.com/blog/mern-technology-stack.
9. Samikshya A. MERN Stack with Modern Web Practices [Електронний ресурс] / Aryal Samikshya. – 2020. – Режим доступу до ресурсу:

https://www.theseus.fi/bitstream/handle/10024/338237/Samikshya_Aryal_Final_Thesis.pdf?sequence=2.

10. Tutorial: Intro to React [Електронний ресурс] – Режим доступу до ресурсу: https://reactjs.org/tutorial/tutorial.html.

11. Elliott E. The Missing Introduction to React [Електронний ресурс] / Eric Elliott. – 2020. – Режим доступу до ресурсу: https://medium.com/javascript-scene/the-missing-introduction-to-react-62837cb2fd76.

12. Stevanoski H. The only introduction to Redux you'll ever need [Електронний ресурс] / Hristijan Stevanoski. – 2019. – Режим доступу до ресурсу: https://javascript.plainenglish.io/the-only-introduction-to-redux-and-react-redux-youll-ever-need-8ce5da9e53c6.

13. Dronca R. Introduction to Sass for Beginners [Електронний ресурс] / Raul Dronca. – 2017. – Режим доступу до ресурсу: https://designmodo.com/introduction-sass/.

14. Snig B. Bootstrap Essentials [Електронний ресурс] / Bhaumik Snig // Packt Publishing Ltd.. – 2015. – Режим доступу до ресурсу: https://books.google.com.ua/books?hl=en&lr=&id=iktVCgAAQBAJ&oi=fnd&pg=PP1&dq=bootstrap&ots=ZtFK7VsCw3&sig=figj5H4ZJFZtscjirva2ylQt7xw&redir_esc=y#v=onepage&q=bootstrap&f=false.

15. Introduction to Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.dev/learn.

16. Node.js in Action [Електронний ресурс] / M.Cantelon, M. Harter, T. Holowaychuk, N. Rajlich. – 2014. – Режим доступу до ресурсу: http://sd.blackball.lv/library/Node.js_in_Action_(2014).pdf.

17. Express/Node introduction [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction.

18. Manjunath M. An Introduction to MongoDB [Електронний ресурс] / Manjunath M. – 2020. – Режим доступу до ресурсу: https://www.sitepoint.com/an-introduction-to-mongodb/.

ДОДАТОК 1

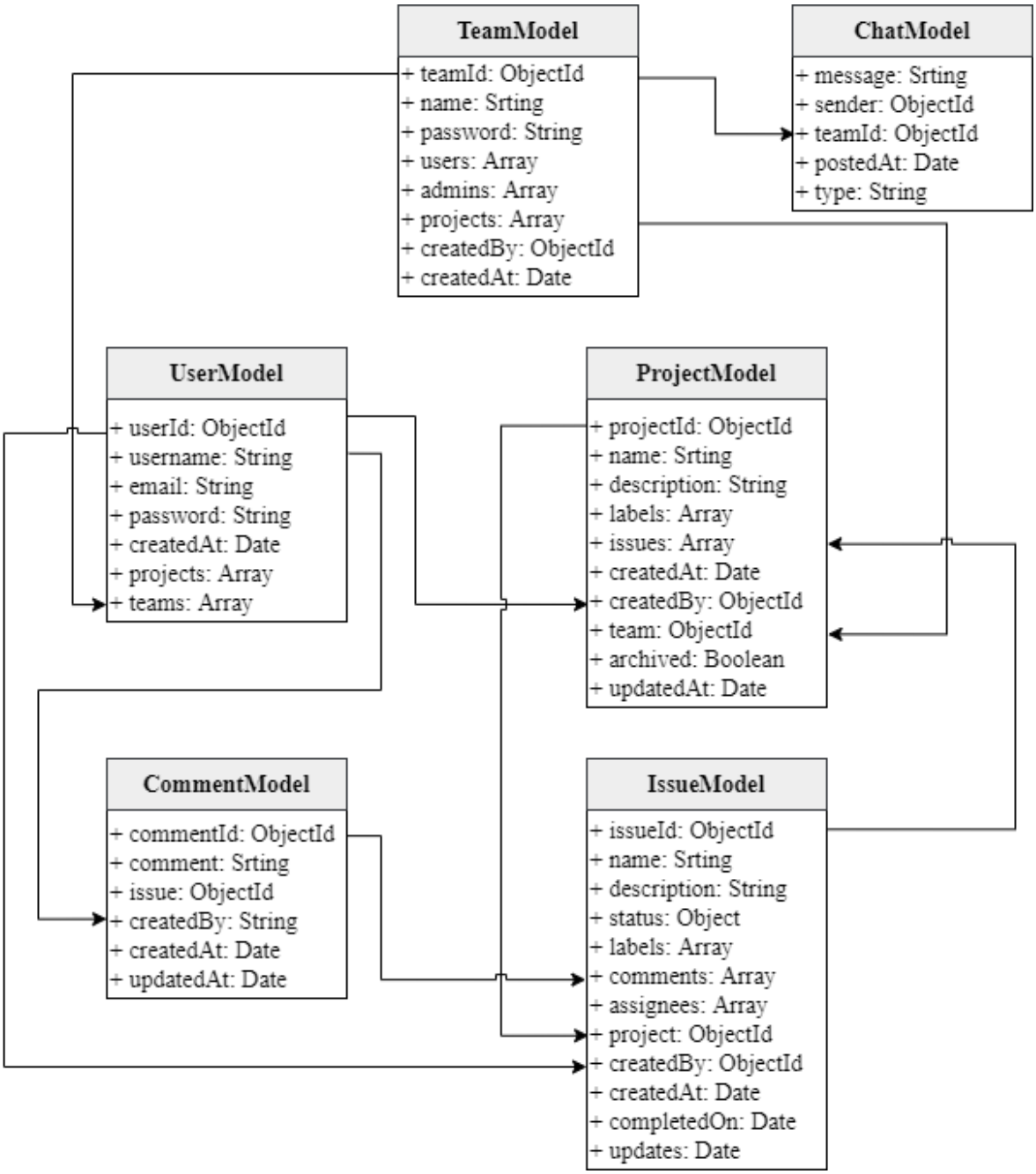
Веб-додаток для управління проектами на основі гнучкої методології створення ПЗ

Діаграма моделей БД

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2021 р.



					ІАЛЦ.467200.004 Д1		
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробила		*Підчасюк Г.О.*			*Веб-додаток для управління проєктами на основі гнучкої методології створення ПЗ. Діаграма моделей БД*	*Літ.*	*Аркуш* / *Аркушів*
Перевірив		*Регіда П.Г.*					*1* / *1*
						НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІВ-71	
Н. контр.		*Сімоненко В.П.*					
Затверд.		*Стіренко С.Г.*					

ДОДАТОК 2

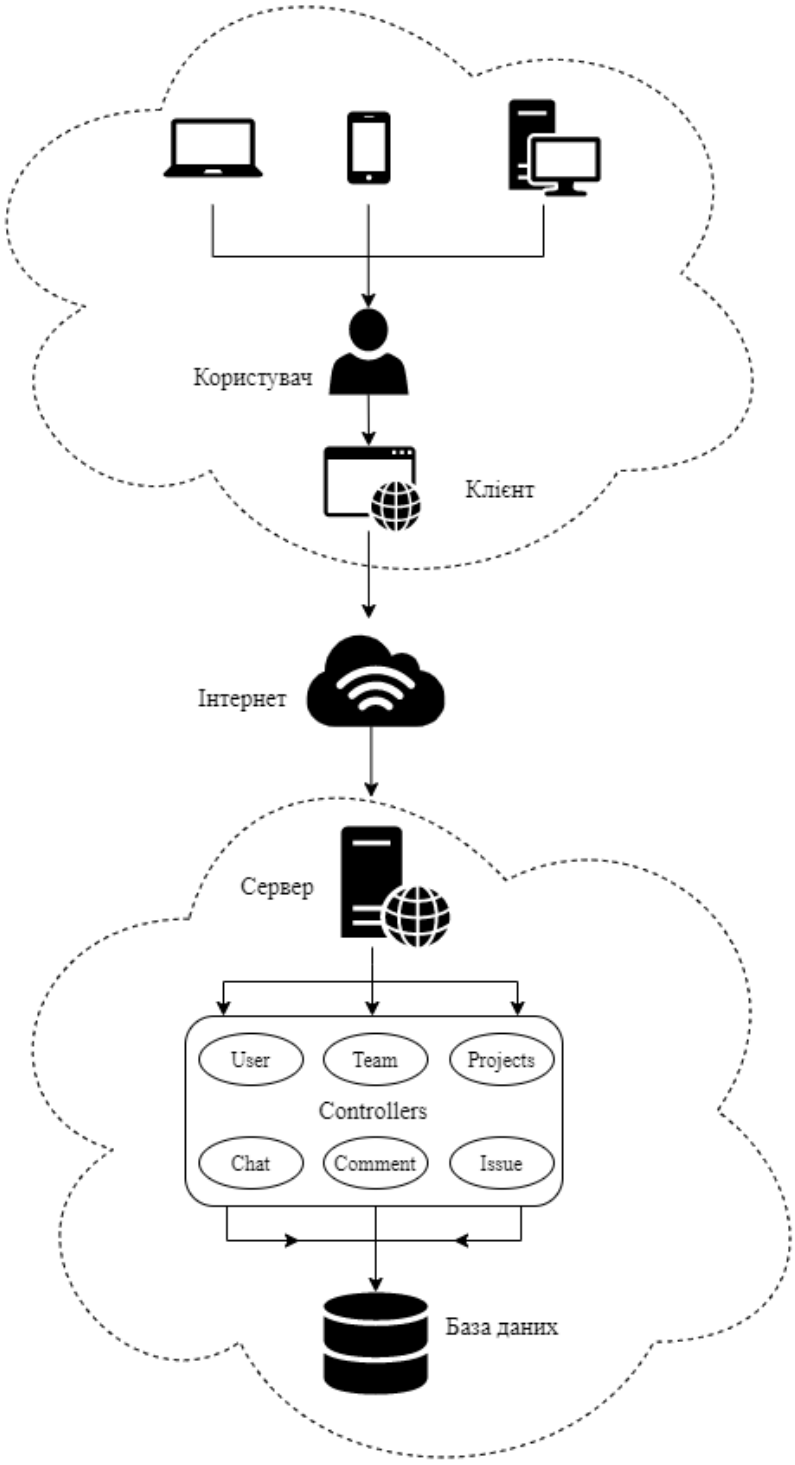
Веб-додаток для управління проектами на основі гнучкої методології створення ПЗ

Схема взаємодії клієнту і серверу

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2021



					ІАЛЦ.467200.005 Д2		
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробила		*Підчасюк Г.О.*			*Веб-додаток для управління проєктами на основі гнучкої методології створення ПЗ. Схема взаємодії клієнту і серверу*	*Літ.*	*Аркуш* / *Аркушів*
Перевірив		*Регіда П.Г.*					*1* / *1*
						НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІВ-71	
Н. контр.		*Сімоненко В.П.*					
Затверд.		*Стіренко С.Г.*					

ДОДАТОК 3

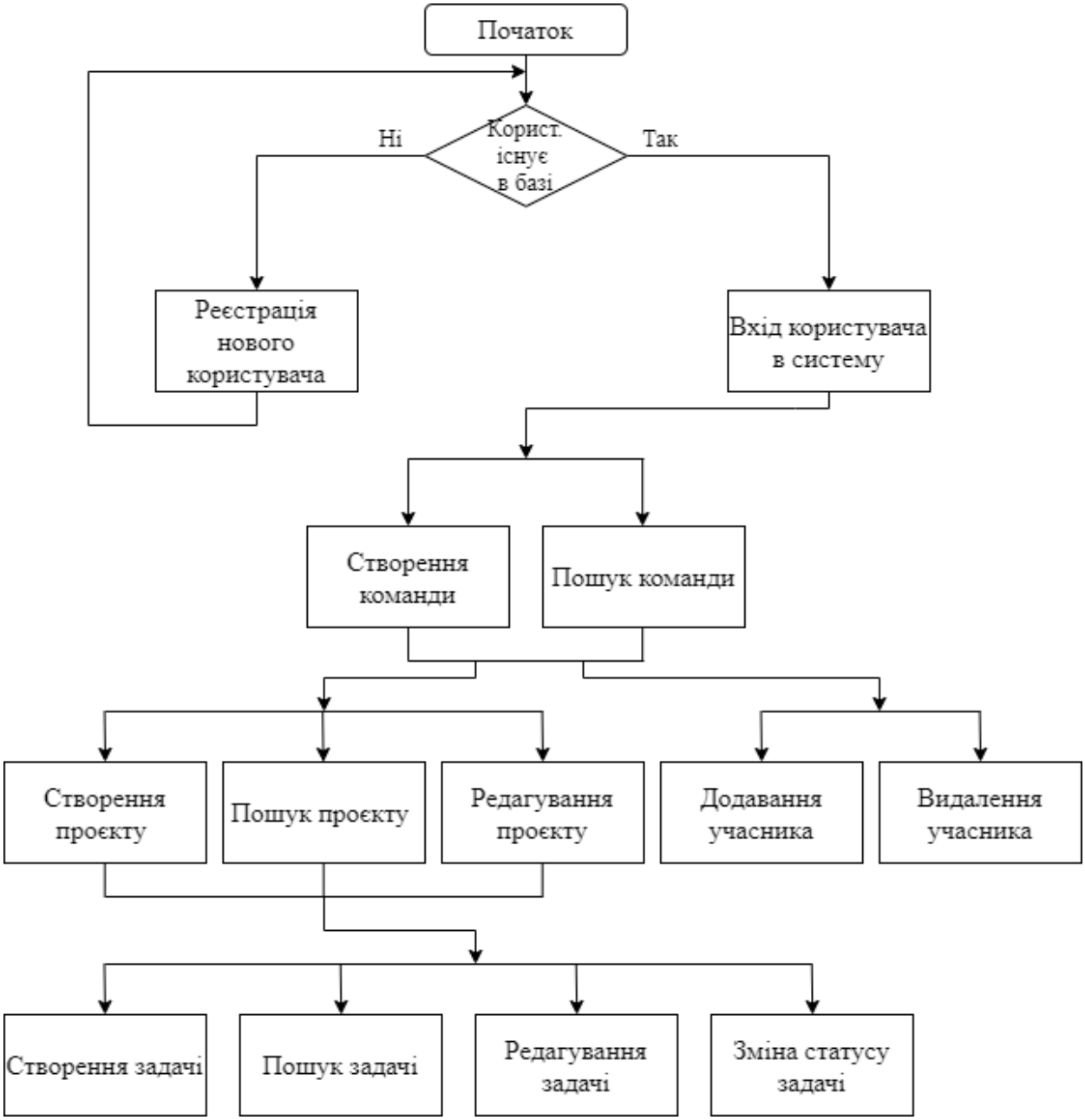
Веб-додаток для управління проектами на основі гнучкої методології створення ПЗ

Алгоритм роботи додатку

ІАЛЦ.467100.006 Д3

Аркушів 1

Київ 202



					ІАЛЦ.467200.006 Д3		
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробила		*Підчасюк Г.О.*			*Веб-додаток для управління проєктами на основі гнучкої методології створення ПЗ. Алгоритм роботи додатку*	*Літ.*	*Аркуш* / *Аркушів*
Перевірив		*Регіда П.Г.*					*1* / *1*
Н. контр.		*Сімоненко В.П.*				*НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІВ-71*	
Затверд.		*Стіренко С.Г.*					

ДОДАТОК 4

Веб-додаток для управління проектами на основі гнучкої методології створення ПЗ

Текст програми

ІАЛЦ.467200.007 Д4

Аркушів 31

Київ 2021

authController.js

```
const {verifyToken} = require("../util/authUtil");

module.exports = {
    refreshToken: (req, res) => {
        const refreshToken = req.cookies["jwtIssRef"];
        const user = req.body.user;

        if (refreshToken === null) return res.sendStatus(401);
        if (user === null || Object.keys(user).length <= 0)
            return res.sendStatus(401);

        try {
            const accessToken = verifyToken(refreshToken, user);
            res.status(200).clearCookie("jwtIss");
            res
                .status(200)
                .cookie("jwtIss", accessToken, {
                    sameSite: "strict",
                    path: "/",
                    httpOnly: true,
                })
                .send("Returning cookie");
        } catch (err) {
            res.sendStatus(401);
        }
    },
};
```

userController.js

```
const {json} = require("express");
const User = require("../models/UserModel");

const          {validateLoginData,          validateSignUpData}          =
require("../util/authUtil");
const {
    signToken,
    decodeToken,
    generateRefreshToken,
} = require("../util/authUtil");

const UserModel = require("../models/UserModel");

module.exports = {
    //Login User
    login: (req, res) => {
        const accessToken = signToken(req.user);
        const refreshToken = generateRefreshToken(req.user);
```

```
                res.status(200).cookie("jwtIss", accessToken, {
                    sameSite: "strict",
                    path: "/",
                    httpOnly: true,
                });

                res
                    .status(200)
                    .cookie("jwtIssRef", refreshToken, {
                        sameSite: "strict",
                        path: "/",
                        httpOnly: true,
                    })
                    .send("Returning cookie");
            },

            //SignUp User
            signUp: async (req, res) => {
                //Check to see if the request has a body
                if (req.body.user) {
                    let messages = [];
                    //Destructor the variables from req.body
                    const   {username,   email,   password,   confirmPassword}   =
req.body.user;

                    //Create a user object
                    const user = {
                        username,
                        email,
                        password,
                        confirmPassword,
                    };

                    //Pass user object for validation and get the returned values
                    const {valid, errors} = validateSignUpData(user);

                    //Check to see if the data passed in was valid
                    if (!valid) return res.status(400).json(errors);

                    //Check to see if the user already exists using email
                    const foundUserByEmail = await UserModel.findOne({email});

                    //If the user already exists, return an error
                    if (foundUserByEmail) {
                        let errors = [];
                        errors.push("Email is already registered");
                        return res.status(403).json(errors);
                    }

                    //Check to see if the user already exists using username
                    const           foundUserByUsername           =           await
UserModel.findOne({username});

                    //If the user already exists, return an error
                    if (foundUserByUsername) {
                        let errors = [];
                        errors.push("Username already exists");
                        return res.status(403).json(errors);
                    }
```

```
                    //Create a new user from req.body
                    const newUser = new UserModel({
                        username,
                        email,
                        password,
                    });

                    //Save the new user in the database
                    await newUser
                        .save()
                        .then(() => {
                            //If user was saved successfully, notify user
                            messages.push("Successfully signed up, try logging
in.");
                            return res.json(messages);
                        })
                        .catch(() => {
                            //If something went wrong when saving user, notify user
                            let errors = [];
                            errors.push("Error occurred when signing user up. Please
try again");
                            return res.status(500).json(errors);
                        });
                }
            },
            // Logout
            logout: (req, res) => {
                if (req.cookies["jwtIss"]) res.status(200).clearCookie("jwtIss");
                if                                          (req.cookies["jwtIssRef"])
res.status(200).clearCookie("jwtIssRef");
                return res.sendStatus(200);
            },
            //Delete User
            deleteUser: (req, res) => {
                if (req.user) {
                    let messages = {};
                    let errors = {};
                    //Get user if from current user
                    const userId = req.user._id;

                    //Find user in database and delete them
                    UserModel.findByIdAndDelete(userId).exec(function (err, user) {
                        //If error occurred, notify user
                        if (err) {
                            console.error(err);
                            errors.user = "Error occurred when attempting to delete
user";
                            return res.status(500).json(errors);
                        }

                        //If everything went well, notify user
                        messages.user = "User successfully deleted";
                        return res.json(messages);
                    });
                }
            },
            getUser: (req, res) => {
                //Get token from cookies
```

```
            const token = req.cookies.jwtIss;
            //Decode token the was sent by user
            const decodedToken = decodeToken(token);
            //Get user id from token sub
            const userId = decodedToken.data;
            //Find user in database
            UserModel.findById(userId).exec(function (err, user) {
                //If error occurred, notify user
                let errors = {};
                if (err) {
                    console.error(err);
                    errors.user = "Error occurred when fetching user data";
                    return res.status(500).json(errors);
                }

                //If everything went well, return user
                return res.json(user);
            });
        },
        editProfile: (req, res) => {
            let errors = [];
            let messages = [];
            let file = req.file;

            const userID = req.user._id;

            User.findByIdAndUpdate(userID, {
                $set: {
                    image: req.file.path,
                },
            }).exec(function (err, user) {
                if (err) {
                    console.error(err);
                    errors.push("Error occurred when updating profile");
                    return res.status(500).json(errors);
                }

                //If everything went right, notify user
                messages.push("Profile successfully updated");
                return res.json(messages);
            });
        },
    };
```

teamController.js

```
const TeamModel = require("../models/TeamModel");
const UserModel = require("../models/UserModel");
const ProjectModel = require("../models/ProjectModel");

module.exports = {
    //Create Team
    createTeam: async (req, res) => {
        if (req.body.team) {
            let messages = [];
            let errors = [];
            //Get name and password from body
            const {name, password} = req.body.team;
```

```
                    //Create a new team using team model
                    const newTeam = new TeamModel({
                        name,
                        password,
                        users: [req.user._id],
                        createdBy: req.user._id,
                    });

                    //Save team in database
                    await newTeam
                        .save()
                        .then(async (team) => {
                            //Add the team to the list of teams the user is in (
adding it by its id )
                            let newTeams = [...req.user.teams, team._id];
                            await UserModel.findByIdAndUpdate(req.user._id, {
                                teams: newTeams,
                            }).exec(function (err, user) {
                                //Error occurred? Notify user
                                if (err) {
                                    console.error(err);
                                    errors.push("Error occurred");
                                    return res.status(500).json(errors);
                                }

                                //Return a message if the team was successfully
saved
                                messages.push("User successfully added to team");
                            });
                            messages.push("Team successfully created");
                            return res.json(messages);
                        })
                        .catch((err) => {
                            //Return an error if the team was not successfully saved
                            errors.push("Error occurred");
                            console.error(err);
                            return res.status(500).json(errors);
                        });
                }
            },
            //Leave Team
            leaveTeam: (req, res) => {
                if (req.params.teamId) {
                    let messages = [];
                    let errors = [];
                    const teamId = req.params.teamId;

                    TeamModel.findById(teamId).exec(function (err, team) {
                        //If an error occurred when fetching team, return it
                        if (err) {
                            console.error(err);
                            errors.push("Opps! Something went wrong");
                            return res.status(500).json(errors);
                        }
                        if (team.createdBy.toString() !== req.user._id.toString())
{
                            //Get all the users in the team and filter out the one
that left
                            var users = team.users;
```

```
                            users = users.filter(
                                (user)        =>        user.toString()        !==
req.user._id.toString()
                            );

                            //Update the team with the new user data
                            TeamModel.findByIdAndUpdate(teamId,             {users:
users}).exec(function (
                                err,
                                team
                            ) {
                                //Error occurred? Notify User
                                if (err) {
                                    console.error(err);
                                    let errors = [];
                                    errors.push("Error occurred");
                                    return res.status(500).json(errors);
                                }

                                //Get all the teams that the user is in and filter
out the one the user left
                                var teams = req.user.teams;
                                teams = teams.filter(
                                    (team) => team.toString() !== teamId.toString()
                                );

                                //Update the user with the new teams
                                UserModel.findByIdAndUpdate(req.user._id, {
                                    teams: teams,
                                }).exec(function (err, user) {
                                    //If an error occurred, notify the user
                                    if (err) {
                                        console.error(err);
                                        errors.push("Error occurred");
                                        return res.status(500).json(errors);
                                    }

                                    //If everything went well, return a message to
notify the user
                                    messages.push("Successfully left team");
                                    return res.json(messages);
                                });
                            });
                        } else {
                            let errors = [];
                            errors.push(
                                "You cannot leave this team because you are the
Admin. Try deleting it instead"
                            );
                            return res.status(400).json(errors);
                        }
                    });
                }
            },
            //Join Team
            joinTeam: async (req, res) => {
                if (req.body.team) {
                    let messages = [];
                    let errors = [];
```

```
                    //Get the variables entered by the user
                    const {teamId, password} = req.body.team;

                    //Search for the team in database
                    const team = await TeamModel.findById(teamId);

                    //If the team was not found, notify user
                    if (team === null) {
                        errors.push("Team not found");
                        return res.status(404).json(errors);
                    }

                    //Check to see if the password entered by the user matches the
team's password
                    const isMatch = await team.isValidPassword(password);

                    //If the passwords don't match, notify user
                    if (!isMatch) {
                        errors.push("Incorrect password");
                        return res.status(401).json(errors);
                    }

                    let inTeam = req.user.teams.some(
                        (team) => team.toString() === teamId.toString()
                    );

                    if (inTeam) {
                        errors.push("You are already part of this team");
                        return res.status(401).json(errors);
                    }

                    //If everything went ok, add user to team
                    const newUsersInTeam = [...team.users, req.user._id];

                    //Update team with new users
                    TeamModel.findByIdAndUpdate(teamId,                    {users:
newUsersInTeam}).exec(
                        function (err, team) {
                            //If an error occurred while updating team with new
user, notify user
                            if (err) {
                                console.error(err);
                                errors.push("Error occurred");
                                return res.status(500).json(errors);
                            }

                            //Add the new team to the list of teams the user is in
                            const newListOfTeams = [...req.user.teams, team._id];
                            UserModel.findByIdAndUpdate(req.user._id, {
                                teams: newListOfTeams,
                            }).exec(function (err, user) {
                                //If something went wrong when updating user with
new team list, notify user
                                if (err) {
                                    console.error(err);
                                    errors.push("Error occurred when updating user
with new teams");
                                    return res.status(500).json(errors);
                                }
```

```
                                messages.push("Joined team successfully");
                                return res.json(messages);
                            });
                        }
                    );
                } else {
                    let errors = [];
                    errors.push("Opps! Something went wrong");
                    return res.status(400).json(errors);
                }
            },
            //Delete Team
            deleteTeam: (req, res) => {
                //Check to see if the teamId was included in the request params
                if (req.params.teamId) {
                    let messages = [];
                    let errors = [];
                    const teamId = req.params.teamId;
                    TeamModel.findByIdAndDelete(teamId).exec(async  function  (err,
team) {
                        if (err) {
                            console.error(err);
                            errors.push("Error occurred when leaving team");
                            return res.status(500).jsno(errors);
                        }

                        //Get all users that are in this team
                        await UserModel.find({teams: team._id}).exec(function (err,
users) {
                            //If an error occurs, return an error string
                            if (err) {
                                console.error(err);
                                let errors = [];
                                errors.push(
                                    "Error occurred when getting all users that are
in the team"
                                );
                                return res.status(500).json(errors);
                            }
                            //Loop through all the users returns from query
                            users.forEach((user) => {
                                //For each user, filter the team out of the teams
that the user is in
                                var newTeams = user.teams.filter(
                                    (team) => team.toString() !== teamId.toString()
                                );
                                UserModel.findByIdAndUpdate(
                                    user._id,
                                    {teams: newTeams},
                                    (err, user) => {
                                        //If there was an error when updating the
user teams, return an error string
                                        if (err) {
                                            console.error(err);
                                            let errors = [];
                                            errors.project = "Error occurred when
removing team from user";
                                            return res.status(500).json(errors);
```

```
                                    }
                                    //If everything went well, return the user
                                    return user;
                                }
                            );
                        });
                    });

                    //Get all projects that are in this team
                    team.projects.forEach((project) => {
                        ProjectModel.findByIdAndDelete(project).exec(function
(err, project) {
                            //Error occurred, notify user
                            if (err) {
                                console.error(err);
                                let errors = [];
                                errors.push("Error   occurred   when   removing
project from team");
                                return res.status(500).json(errors);
                            }
                        });
                    });

                    //When everything is done, return a message to inform the
user
                    messages.push("Team successfully deleted");
                    return res.json(messages);
                });
            }
        },

        getTeam: (req, res) => {
            if (req.params.teamId) {
                let errors = [];
                const teamId = req.params.teamId;
                TeamModel.findById(teamId)
                    .populate("projects")
                    .populate("users")
                    .exec(function (err, team) {
                        //If something went wrong when fetching team, notify
user
                        if (err) {
                            console.error(err);
                            errors.push("Error occurred");
                            return res.status(500).json(errors);
                        }

                        let belongsToTeam = team.users.findIndex(
                            (user)       =>       user._id.toString()       ===
req.user._id.toString()
                        );

                        if (belongsToTeam === -1) return res.sendStatus(404);

                        //If everything went well, return team
                        return res.json(team);
                    });
            }
        },
```

```
getTeams: (req, res) => {
    let errors = [];
    const userId = req.user._id;

    TeamModel.find({users: {$elemMatch: {$eq: userId}}})
        .sort("name")
        .select("name createdBy _id")
        .exec(function (err, teams) {
            //If error occurred when fetching teams, notify user
            if (err) {
                console.error(err);
                errors.push("Error occurred when fetching teams");
                return res.status(500).json(teams);
            }

            //If everything went well, return teams
            return res.json(teams);
        });
},
getArchivedTeamProjects: (req, res) => {
    if (req.params.teamId) {
        let errors = [];
        const teamId = req.params.teamId;

        TeamModel.findById(teamId)
            .populate("projects")
            .exec(function (err, team) {
                //Error occurred? Notify User
                if (err) {
                    errors.push("Error Occurred");
                    return res.status(500).json(errors);
                }

                //Everything went well? Continue
                const archivedProjects = team.projects.filter(
                    (project) => project.archived === true
                );

                return res.json(archivedProjects);
            });
    } else {
        let errors = [];
        errors.push("Error occurred");
        return res.status(400).json(errors);
    }
},
getTeamWithProjects: (req, res) => {
    if (req.params.teamId) {
        // let errors = [];
        const teamId = req.params.teamId;

        TeamModel.findById(teamId)
            .populate({
                path: "projects",
                select: "name archived team",
            })
            .populate({
                path: "users",
                select: "username image",
```

```
                        })
                        .exec(function (err, team) {
                            if (err) {
                                return res.status(500).json(["Error occurred"]);
                            }

                            let belongsToTeam = team.users.findIndex(
                                (user)       =>       user._id.toString()       ===
req.user._id.toString()
                            );

                            if (belongsToTeam === -1) return res.sendStatus(404);

                            return res.json(team);
                        });
                } else {
                    return res.status(400).json(["Error occurred"]);
                }
            },
            getTeamManagementInfo: (req, res) => {
                if (req.params.teamId) {
                    let errors = [];
                    const teamId = req.params.teamId;

                    TeamModel.findById(teamId)
                        .select("-createdAt -name -projects -password")
                        .populate({
                            path: "users",
                            select: "username image",
                        })
                        .populate({
                            path: "createdBy",
                            select: "username image",
                        })
                        .populate({
                            path: "admins",
                            select: "username image",
                        })
                        .exec(function (err, team) {
                            //If an error occurred, notify the user
                            if (err) {
                                errors.push("Opps, Something went wrong!");
                                return res.status(500).json(errors);
                            }

                            //If everything went well, return the team
                            return res.json(team);
                        });
                } else {
                    return res.status(400).json(["Error occurred"]);
                }
            },
        };
```

projectController.js

```
        const ProjectModel = require("../models/ProjectModel");
        const UserModel = require("../models/UserModel");
        const TeamModel = require("../models/TeamModel");
```

```
module.exports = {
    //Create Project
    createProject: (req, res) => {
        if (req.body.project) {
            let messages = [];
            let errors = [];

            const team = req.body.project.teamId || null;
            const defaultLabels = [
                {
                    name: "Need Help",
                    fontColor: "#fff",
                    backgroundColor: "#32936f",
                },
                {
                    name: "More Info",
                    fontColor: "#fff",
                    backgroundColor: "#2274a5",
                },
                {
                    name: "Bug",
                    fontColor: "#fff",
                    backgroundColor: "#a71d31",
                },
                {
                    name: "Question",
                    fontColor: "#fff",
                    backgroundColor: "#FFAA1D",
                },
            ];

            //Get variables from the user
            const {name, description} = req.body.project;

            //Create an instance of the project model ( a document ) with
the values passed in
            //by the user and other pre-defined values
            const newProject = new ProjectModel({
                name,
                description,
                createdBy: req.user._id,
                labels: defaultLabels,
                createdBy: req.user._id,
                team,
            });

            //Save a new project to the database
            newProject
                .save()
                .then((project) => {
                    //Add the project to the list of projects the user is
in ( adding it by its id )
                    let newProjects = [...req.user.projects, project._id];

                    UserModel.findByIdAndUpdate(req.user._id, {
                        projects: newProjects,
                    }).exec(function (err, user) {
                        //Error occurred? Notify user
```

```
                                if (err) {
                                    console.error(err);
                                    errors.push("Error occurred when adding project
to user");
                                    return res.status(500).json(errors);
                                }

                                if (req.body.project.teamId) {
                                    const teamId = req.body.project.teamId;
                                    TeamModel.findById(teamId).exec(function  (err,
team) {
                                        //Error occurred? Notify user
                                        if (err) {
                                            console.error(err);
                                            errors.push("Error occurred when adding
project to team");
                                            return res.status(500).json(errors);
                                        }

                                        const newProjectsArray = [...team.projects,
project._id];

                                        TeamModel.findByIdAndUpdate(teamId, {
                                            projects: newProjectsArray,
                                        }).exec(function (err, team) {
                                            //Error occurred? Notify user
                                            if (err) {
                                                console.error(err);
                                                errors.push("Error occurred");
                                                return
res.status(500).json(errors);
                                            }

                                            //Return a message if the project was
successfully saved
                                            messages.push("Project    successfully
created");
                                            return res.json(messages);
                                        });
                                    });
                                } else {
                                    //Return   a   message   if   the   project   was
successfully saved
                                    messages.push("Project successfully created");
                                    return res.json(messages);
                                }
                            });
                        })
                        .catch((error) => {
                            errors.push("Error occurred when creating project");
                            return res.status(500).json(error);
                        });
                } else {
                    let errors = [];
                    errors.push("Opps! Something went wrong");
                    return res.status(400).json(errors);
                }
            },
            //Edit Project
```

```
            editProject: (req, res) => {
                if (req.body.project) {
                    if (req.params.projectId) {
                        let messages = [];
                        let errors = [];
                        const projectId = req.params.projectId;

                        //Update project information with the information passed in
by the user
                        ProjectModel.findByIdAndUpdate(projectId, {
                            $set: {
                                name: req.body.project.name,
                                description: req.body.project.description,
                            },
                        }).exec(function (err, project) {
                            //If an error occurred while updating project, notify
user
                            if (err) {
                                console.error(err);
                                errors.push("Error    occurred    when    updating
project");
                                return res.status(500).json(errors);
                            }

                            //If everything went right, notify user
                            messages.push("Project successfully updated");
                            return res.json(messages);
                        });
                    } else {
                        let errors = [];
                        errors.push("Opps! Something went wrong");
                        return res.status(400).json(errors);
                    }
                } else {
                    let errors = [];
                    errors.push("Opps! Something went wrong");
                    return res.status(400).json(errors);
                }
            },
            //Delete Project
            deleteProject: (req, res) => {
                if (req.params.projectId) {
                    let errors = [];
                    let messages = [];
                    const projectId = req.params.projectId;

                    ProjectModel.findById(projectId).exec(function (err, project) {
                        //Error occurred? Notify User
                        if (err) {
                            errors.push("Error occurred");
                            return res.status(500).json(errors);
                        }

                        //Check to see if the project is in a team, if so, then the
team creator has permission to delete the project
                        if (project.team) {
                            TeamModel.findById(project.team).exec(function    (err,
team) {
                                //Error occurred? Notify user
```

```
                                if (err) {
                                    console.error(err);
                                    errors.push("An Error occurred");
                                    return res.status(500).json(errors);
                                }

                                //Check to see if the person deleting the project
is the same person who created the team
                                if          (team.createdBy.toString()          ===
req.user._id.toString()) {

ProjectModel.findByIdAndDelete(projectId).exec(function (
                                        err,
                                        project
                                    ) {
                                        //If  something  went  wrong  when  deleting
project, notify user
                                        if (err) {
                                            console.error(err);
                                            errors.push("Error occurred");
                                            return res.status(500).json(errors);
                                        }


TeamModel.findById(project.team).exec(function (err, team) {
                                            //Error occurred? Notify user
                                            if (err) {
                                                console.error(err);
                                                errors.push("Error occurred");
                                                return
res.status(500).json(errors);
                                            }

                                            const        newProjectsArray        =
team.projects.filter(
                                                (teamProject) =>
                                                    teamProject.toString()      !==
project._id.toString()
                                            );


TeamModel.findByIdAndUpdate(project.team, {
                                                projects: newProjectsArray,
                                            }).exec(function (err, team) {
                                                //Error occurred? Notify user
                                                if (err) {
                                                    console.error(err);
                                                    errors.push("Error occurred");
                                                    return
res.status(500).json(errors);
                                                }

                                                //If everything went right, notify
user
                                                messages.push("Project          was
successfully deleted");
                                                return res.json(messages);
                                            });
                                        });
```

```
                                    });
                                } else {
                                    //If the person trying to delete the project
isn't the same person who created it, notify them
                                    errors.push("You don't have permission to delete
this project");
                                    return res.status(401).json(errors);
                                }
                            });
                        } else {
                            //Check to see if the person deleting the project is the
same person who created the project
                            if           (project.createdBy.toString()          ===
req.user._id.toString()) {

ProjectModel.findByIdAndDelete(projectId).exec(function (
                                    err,
                                    project
                                ) {
                                    //If something went wrong when deleting project,
notify user
                                    if (err) {
                                        console.error(err);
                                        errors.push("Error occurred");
                                        return res.status(500).json(errors);
                                    }

                                    UserModel.findById(req.user._id).exec(function
(err, user) {
                                        if (err) {
                                            console.error(err);
                                            errors.push("Error occurred");
                                            return res.status(500).json(errors);
                                        }

                                        let newProjects = user.projects.filter(
                                            (userProject) =>
                                                userProject.toString()          !==
project._id.toString()
                                        );

                                        UserModel.findByIdAndUpdate(req.user._id, {
                                            projects: newProjects,
                                        }).exec(function (err, user) {
                                            if (err) {
                                                console.error(err);
                                                errors.push("Error occurred");
                                                return
res.status(500).json(errors);
                                            }

                                            //If everything went right, notify user
                                            messages.push("Project was successfully
deleted");

                                            return res.json(messages);
                                        });
                                    });
                                });
                            } else {
```

```
                                //If the person trying to delete the project isn't
the same person who created it, notify them
                                errors.push("You don't have permission to delete
this project");
                                return res.status(401).json(errors);
                            }
                        }
                    });
                } else {
                    let errors = [];
                    errors.push("Opps! Something went wrong");
                    return res.status(400).json(errors);
                }
            },
            getProject: (req, res) => {
                if (req.params.projectId) {
                    let errors = [];

                    const projectId = req.params.projectId;

                    ProjectModel.findById(projectId)
                        .populate("createdBy")
                        .populate({path: "issues", options: {sort: {"status.name":
1}}})
                        .exec(function (err, project) {
                            //If something went wrong when looking for project,
notify user
                            if (err) {
                                console.error(err);
                                errors.push("Error occurred");
                                return res.status(500).json(errors);
                            }

                            if (
                                project.team === null &&
                                req.user._id.toString()                          !==
project.createdBy._id.toString()
                            ) {
                                return res.sendStatus(404);
                            } else if (project.team !== null) {
                                TeamModel.findById(project.team).exec(function
(err, team) {
                                    if (err) {
                                        console.error(err);
                                        errors.push("Error occurred");
                                        return res.status(500).json(errors);
                                    }

                                    let inTeam = team.users.findIndex(
                                        (user)      =>      user.toString()      ===
req.user._id.toString()
                                    );
                                    if (inTeam === -1) return res.sendStatus(404);
                                    else return res.json(project);
                                });
                            } else {
                                //If everything went right, return project
                                return res.json(project);
                            }
```

```
                    });
                }
            },
            getProjects: (req, res) => {
                let errors = [];
                const userId = req.user._id;
                //Find all projects created by a certain user
                ProjectModel.find({
                    $and: [{createdBy: userId}, {team: null}, {archived: false}],
                })
                    .sort("name")
                    .select("name _id createdBy")
                    .exec(function (err, projects) {
                        //If something went wrong when fetching projects, notify
user
                        if (err) {
                            errors.push("Error occurred");
                            console.error(err);
                            return res.status(500).json(errors);
                        }

                        //If everything went well, return projects
                        return res.json(projects);
                    });
            },
            getProjectsForTeam: (req, res) => {
                let errors = [];
                const teamId = req.params.teamId;
                //Find all projects created by a certain user
                ProjectModel.find({
                    $and: [{team: teamId}, {archived: false}],
                })
                    .select(" _id name archived")
                    .exec(function (err, projects) {
                        //If something went wrong when fetching projects, notify
user
                        if (err) {
                            errors.push("Error occurred when retrieving projects");
                            console.error(err);
                            return res.status(500).json(errors);
                        }

                        //If everything went well, return projects
                        return res.json(projects);
                    });
            },
            addLabel: (req, res) => {
                if (req.params.projectId) {
                    if (req.body.label) {
                        let messages = [];
                        let errors = [];
                        const projectId = req.params.projectId;
                        ProjectModel.findById(projectId).exec(function        (err,
project) {
                            //If error occurred when fetching project, notify user
                            if (err) {
                                console.error(err);
                                errors.push("Error occurred");
                                return res.status(500).json(errors);
```

```
                        }

                        //Check to see if the label already exists
                        const hasLabel = project.labels.some(
                            (label)      =>      label.name.toString()      ===
req.body.label.name.toString()
                        );

                        //If it does, notify user
                        if (hasLabel) {
                            errors.push("That label already exists");
                            return res.status(400).json(errors);
                        }

                        //If everything went well continue
                        const newLabels = [...project.labels, req.body.label];

                        //Update Project with new labels
                        ProjectModel.findByIdAndUpdate(projectId,      {labels:
newLabels}).exec(
                            function (err, project) {
                                //If  error  occurred  when  updating  project
labels, notify user
                                if (err) {
                                    console.error(err);
                                    errors.push("Error occurred");
                                    return res.status(500).json(errors);
                                }

                                //If everything went well, notify user,
                                messages.push("Label successfully added");
                                return res.json(messages);
                            }
                        );
                    });
                } else {
                    let errors = [];
                    errors.push("Opps! Something went wrong");
                    return res.status(400).json(errors);
                }
            } else {
                let errors = [];
                errors.push("Opps! Something went wrong");
                return res.status(400).json(errors);
            }
        },
        deleteLabel: (req, res) => {
            if (req.params.projectId) {
                if (req.params.labelId) {
                    let errors = [];
                    let messages = [];
                    const projectId = req.params.projectId;
                    const labelId = req.params.labelId;
                    ProjectModel.findById(projectId).exec(function       (err,
project) {
                        //If error occurred when fetching project, notify user
                        if (err) {
                            console.error(err);
                            errors.push("Error occurred");
```

```
                            return res.status(500).json(errors);
                        }

                        //If everything went well, remove label
                        const newLabels = project.labels.filter(
                            (label)      =>      label._id.toString()      !==
labelId.toString()
                        );

                        ProjectModel.findByIdAndUpdate(projectId,      {labels:
newLabels}).exec(
                            function (err, project) {
                                //If  error  occurred  when  updating  project
labels, notify user
                                if (err) {
                                    console.error(err);
                                    errors.push("Error  occurred  when  updating
project labels");
                                    return res.status(500).json(errors);
                                }

                                //If everything went well, notify user
                                messages.push("Label successfully removed");
                                return res.json(messages);
                            }
                        );
                    });
                } else {
                    let errors = [];
                    errors.push("Opps! Something went wrong");
                    return res.status(400).json(errors);
                }
            } else {
                let errors = [];
                errors.push("Opps! Something went wrong");
                return res.status(400).json(errors);
            }
        },
        editLabel: (req, res) => {
            if (req.params.projectId) {
                if (req.params.labelId) {
                    let errors = [];
                    let messages = [];

                    const projectId = req.params.projectId;
                    const labelId = req.params.labelId;

                    ProjectModel.findById(projectId).exec(function       (err,
project) {
                        //If error occurred when fetching project, notfiy user
                        if (err) {
                            console.error(err);
                            errors.push("Error occurred");
                            return res.status(500).json(errors);
                        }

                        //If everything went well, remove label
                        var label = project.labels.find(
```

```
                                (label)      =>      label._id.toString()      ===
labelId.toString()
                            );

                            var newLabelsArray = project.labels.filter(
                                (label)      =>      label._id.toString()      !==
labelId.toString()
                            );

                            //Check to see if the label already exists
                            const hasLabel = project.labels.find(
                                (label)      =>      label.name.toString()      ===
req.body.label.name.toString()
                            );

                            //If it does, notify user
                            if    (hasLabel    &&    hasLabel._id.toString()    !==
labelId.toString()) {
                                errors.push("That label already exists");
                                return res.status(400).json(errors);
                            }

                            label.name = req.body.label.name;
                            label.backgroundColor = req.body.label.backgroundColor;
                            label.fontColor = req.body.label.fontColor;

                            newLabelsArray = [...newLabelsArray, label];

                            ProjectModel.findByIdAndUpdate(projectId, {
                                labels: newLabelsArray,
                            }).exec(function (err, project) {
                                //If error occurred when updating project labels,
notify user
                                if (err) {
                                    console.error(err);
                                    errors.push("Error occurred");
                                    return res.status(500).json(errors);
                                }

                                //If everything went well, notify user
                                messages.push("Label successfully updated");
                                return res.json(messages);
                            });
                        });
                    } else {
                        let errors = [];
                        errors.push("Opps! Something went wrong");
                        return res.status(400).json(errors);
                    }
                } else {
                    let errors = [];
                    errors.push("Opps! Something went wrong");
                    return res.status(400).json(errors);
                }
            },
            addToArchive: (req, res) => {
                if (req.params.projectId) {
                    let errors = [];
                    let messages = [];
```

```
                const projectId = req.params.projectId;

                //Everything went well, add project to archive
                ProjectModel.findByIdAndUpdate(projectId,           {archived:
true}).exec(
                    function (err, project) {
                        //Error occrrued? Notify User
                        if (err) {
                            errors.push("Error occurred");
                            return res.status(500).json(errors);
                        }

                        //Everything went well, notify user
                        messages.push("Project successfully archived");
                        return res.json(messages);
                    }
                );
            } else {
                let errors = [];
                errors.push("Error occurred");
                return res.status(400).json(errors);
            }
        },
        removeFromArchive: (req, res) => {
            if (req.params.projectId) {
                let errors = [];
                let messages = [];
                const projectId = req.params.projectId;

                //Everything went well, remove project from archive
                ProjectModel.findByIdAndUpdate(projectId,           {archived:
false}).exec(
                    function (err, project) {
                        //Error occrrued? Notify User
                        if (err) {
                            errors.push("Error occurred");
                            return res.status(500).json(errors);
                        }

                        //Everything went well, notify user
                        messages.push("Project   successfully   removed   from
archived");
                        return res.json(messages);
                    }
                );
            } else {
                let errors = [];
                errors.push("Error occurred");
                return res.status(400).json(errors);
            }
        },
        getArchivedProjects: (req, res) => {
            let errors = [];
            const userId = req.user._id;
            //Find all projects created by a certain user
            ProjectModel.find({
                $and: [{createdBy: userId}, {team: null}, {archived: true}],
            })
                .populate("issues")
```

```
                    .exec(function (err, projects) {
                        //If something went wrong when fetching projects, notify
user
                        if (err) {
                            errors.push("Error occurred");
                            console.error(err);
                            return res.status(500).json(errors);
                        }

                        //If everything went well, return projects
                        return res.json(projects);
                    });
            },
        };
```

issueController.js

```
        const IssueModel = require("../models/IssueModel");
        const ProjectModal = require("../models/ProjectModel");
        const TeamModal = require("../models/TeamModel");
        const _ = require("lodash");

        const {NEW_ISSUE} = require("../util/issueStatus");

        const issueUpdateTypes = {
            "STATUS UPDATE": "STATUS UPDATE",
            "DESCRIPTION UPDATE": "DESCRIPTION UPDATE",
            "NAME UPDATE": "NAME UPDATE",
            "LABEL UPDATE": "LABEL UPDATE",
        };

        module.exports = {
            //Create Issue
            createIssue: (req, res) => {
                if (req.body.issue) {
                    let errors = [];
                    let messages = [];

                    //Get variables user entered
                    const {
                        name,
                        description,
                        labels,
                        projectId,
                        assignees,
                    } = req.body.issue;

                    IssueModel.find({name: name, project: projectId}).exec(function
(
                        err,
                        issues
                    ) {
                        if (issues.length === 0) {
                            //Create an instance of the user model ( a document )
with the values
                            //entered by the user and pre-defined values
```

```
                            const newIssue = new IssueModel({
                                name,
                                description,
                                labels,
                                comments: [],
                                assignees,
                                status: NEW_ISSUE,
                                project: projectId,
                                createdBy: req.user._id,
                            });

                            newIssue
                                .save()
                                .then((issue) => {
                                    //If everything went well, notify user

                                    ProjectModal.findById(projectId).exec(function
(err, project) {
                                        //If error occurred when fetching project,
notify user
                                        if (err) {
                                            console.error(err);
                                            errors.push("Error when updating issues
in project");
                                            return res.status(500).json(errors);
                                        }

                                        //If everything went well, update issues in
project
                                        const   newIssues   =   [...project.issues,
issue._id];

                                        //Update Project
                                        ProjectModal.findByIdAndUpdate(projectId, {
                                            issues: newIssues,
                                        }).exec(function (err, project) {
                                            //If   error   occurred   when   updating
project, notify user
                                            if (err) {
                                                console.error(err);
                                                errors.push("Error   when   updating
issues in project");
                                                return
res.status(500).json(errors);
                                            }

                                            //If everything went well, notify user
                                            messages.push("Issue       successfully
added");
                                            return res.json(messages);
                                        });
                                    });
                                })
                                .catch((err) => {
                                    //If error occurred, notify user
                                    errors.push("Error    occurred    when    adding
issue");
                                    return res.status(500).json(errors);
                                });
```

```
                        } else {
                            errors.push("Issue name already exists");
                            return res.status(400).json(errors);
                        }
                    });
                } else {
                    let errors = [];
                    errors.push("Opps! Something went wrong");
                    return res.status(400).json(errors);
                }
            },
            //Edit Issue
            editIssue: (req, res) => {
                if (req.body.issue) {
                    if (req.params.issueId) {
                        let messages = [];
                        let errors = [];
                        const issueId = req.params.issueId;

                        const issueInfo = req.body.issue;

                        IssueModel.findById(issueId).exec(function (err, issue) {
                            //If error occurred, notify user
                            if (err) {
                                console.error(err);
                                error.push("Error occurred when updating issue");
                                return res.status(500).json(errors);
                            }

                            let newUpdates = [...issue.updates];

                            let oldDescription = issue.description;
                            let oldName = issue.name;
                            let oldLabels = issue.labels;

                            if            (oldDescription.toString()            !==
issueInfo.description.toString()) {
                                let newUpdate = {
                                    date: Date.now(),
                                    updateType:       issueUpdateTypes["DESCRIPTION
UPDATE"],
                                    updateInfo: issueInfo.description,
                                    updatedBy: req.body.username,
                                };
                                newUpdates.push(newUpdate);
                            }

                            if (oldName.toString() !== issueInfo.name.toString()) {
                                let newUpdate = {
                                    date: Date.now(),
                                    updateType: issueUpdateTypes["NAME UPDATE"],
                                    updateInfo: issueInfo.name,
                                    updatedBy: req.body.username,
                                };
                                newUpdates.push(newUpdate);
                            }

                            if (!_.isEqual(oldLabels, issueInfo.labels)) {
                                let newUpdate = {
```

```
                                    date: Date.now(),
                                    updateType: issueUpdateTypes["LABEL UPDATE"],
                                    updateInfo: issueInfo.labels.join(" "),
                                    updatedBy: req.body.username,
                                };
                                newUpdates.push(newUpdate);
                            }

                            IssueModel.findByIdAndUpdate(issueId, {
                                $set: {
                                    name: issueInfo.name,
                                    description: issueInfo.description,
                                    labels: issueInfo.labels,
                                    assignees: issueInfo.assignees,
                                    updates: newUpdates,
                                },
                            }).exec(function (err, issue) {
                                //If error occurred, notify user
                                if (err) {
                                    console.error(err);
                                    errors.push("Error   occurred   when   updating
issue");
                                    return res.status(500).json(errors);
                                }

                                //If everything went well, notify user
                                messages.push("Successfully updated issue");
                                return res.json(messages);
                            });
                        });
                    } else {
                        let errors = [];
                        errors.push("Issue Not Found");
                        return res.status(404).json(errors);
                    }
                } else {
                    let errors = [];
                    errors.push("Opps! Something went wrong");
                    return res.status(400).json(errors);
                }
            },
            editIssueStatus: (req, res) => {
                if (req.body.issue) {
                    if (req.params.issueId) {
                        let messages = [];
                        let errors = [];
                        const issueId = req.params.issueId;

                        const newStatus = req.body.issue;
                        IssueModel.findById(issueId).exec(function (err, issue) {
                            //If error occurred, notify user
                            if (err) {
                                console.error(err);
                                error.push("Error occurred when updating issue");
                                return res.status(500).json(errors);
                            }

                            let newCompletedOn;
```

```
                            if (newStatus["name"] === "Completed") newCompletedOn =
Date.now();
                            else newCompletedOn = null;

                            let newUpdates = [...issue.updates];
                            let newUpdate = {
                                date: Date.now(),
                                updateType: issueUpdateTypes["STATUS UPDATE"],
                                updateInfo: newStatus["name"] ? newStatus["name"] :
"undefined",
                                updatedBy: req.body.username,
                            };

                            newUpdates.push(newUpdate);

                            IssueModel.findByIdAndUpdate(issueId, {
                                status: newStatus,
                                completedOn: newCompletedOn,
                                updates: newUpdates,
                            }).exec(function (err, issue) {
                                //If error occurred, notify user
                                if (err) {
                                    console.error(err);
                                    error.push("Error   occurred   when   updating
issue");
                                    return res.status(500).json(errors);
                                }

                                //If everything went well, notify user
                                messages.push("Successfully updated issue");
                                return res.json(messages);
                            });
                        });
                    } else {
                        let errors = [];
                        errors.push("Issue Not Found");
                        return res.status(404).json(errors);
                    }
                } else {
                    let errors = [];
                    errors.push("Opps! Something went wrong");
                    return res.status(400).json(errors);
                }
            },
            //Delete Issue
            deleteIssue: (req, res) => {
                if (req.params.issueId) {
                    let messages = [];
                    let errors = [];
                    const issueId = req.params.issueId;
                    IssueModel.findByIdAndDelete(issueId).exec(function      (err,
issue) {
                        //If error occurred, notify user
                        if (err) {
                            console.error(err);
                            errors.push("Error occurred when deleting issue");
                            return res.status(500).json(errors);
                        }
```

```
                    //If everything went well, notify user
                    messages.push("Successfully deleted issue");
                    return res.json(messages);
                });
            } else {
                let errors = [];
                errors.push("Issue not found");
                return res.status(404).json(errors);
            }
        },
        getIssue: (req, res) => {
            if (req.params.issueId) {
                let errors = [];
                const issueId = req.params.issueId;
                IssueModel.findById(issueId)
                    .populate("createdBy comments project assignees")
                    .exec(function (err, issue) {
                        //If something went wrong when fetching issue, notify
user
                        if (err) {
                            console.error(err);
                            errors.issue = "Error occurred when fetching issue";
                            return res.status(500).json(errors);
                        }

                        //Checks to see if the project the issue belongs too is
part of a team
                        if (issue.project.team) {
                            //If the project is part of a team, get the team

TeamModal.findById(issue.project.team).exec(function (err, team) {
                                if (err) {
                                    console.error(err);
                                    errors.issue = "Error occurred when fetching
issue";
                                    return res.status(500).json(errors);
                                }

                                //Check to see if the user is part of the team
                                let belongsToTeam = team.users.findIndex(
                                    (user)      =>      user.toString()      ===
req.user._id.toString()
                                );

                                //If the user isn't part of the team, send a 404
response
                                if     (belongsToTeam     ===     -1)     return
res.sendStatus(404);

                                //If everything went right, return issue
                                return res.json(issue);
                            });
                        } else {
                            //If it isn't, check to see if the user trying to
view the issue is the owner of the issue
                            if      (issue.project.createdBy.toString()      !==
req.user._id.toString())
                                return res.sendStatus(404);
                            //If everything went right, return issue
```

```
                            return res.json(issue);
                        }
                    });
            }
        },
    };
```

commentController.js

```
    const CommentModel = require("../models/CommentModel");
    const IssueModel = require("../models/IssueModel");

    module.exports = {
        createComment: (req, res) => {
            if (req.body.issueId && req.body.comment) {
                let messages = {};
                let errors = {};
                const issueId = req.body.issueId;

                const {comment} = req.body.comment;
                const username = req.body.username;

                const newComment = new CommentModel({
                    comment,
                    createdBy: username,
                    issue: issueId,
                });

                newComment
                    .save()
                    .then((comment) => {
                        //Add comment to issue document
                        IssueModel.findById(issueId).exec(function (err, issue)
{
                            //If something went wrong when fecthing issue,
notify user,
                            if (err) {
                                console.error(err);
                                errors.comment = "Error occurred";
                                return res.this.status(500).json(error);
                            }

                            const newCommentArray = [...issue.comments,
comment._id];

                            //Update issue with new comments

                            IssueModel.findByIdAndUpdate(issueId, {
                                comments: newCommentArray,
                            }).exec(function (err, issue) {
                                //If something went wrong when updating issue,
notify user,
                                if (err) {
                                    console.error(err);
                                    errors.comment = "Error occurred";
                                    return res.this.status(500).json(error);
                                }

                                //If everything went well, notify user
```

```
                                    messages.comment = "Comment successfully
added";
                                    return res.json(messages);
                                });
                            });
                        })
                        .catch((error) => {
                            console.error(error);
                            errors.comment = "Opps! Something went wrong";
                            return res.status(500).json(errors);
                        });
                } else {
                    let errors = {};
                    errors.general = "Opps! Something went wrong";
                    return res.status(400).json(errors);
                }
            },
            deleteComment: (req, res) => {
                if (req.params.commentId) {
                    let errors = {};
                    let messages = {};
                    const commentId = req.params.commentId;
                    CommentModel.findByIdAndDelete(commentId).exec(function (err,
comment) {
                        //If something went wrong when deleting comment, notify user
                        if (err) {
                            console.error(err);
                            errors.comment = "Error occurred";
                            return res.status(500).json(errors);
                        }

                        //If comment was deleted, remove it from issue
                        IssueModel.findById(comment.issue).exec(function (err,
issue) {
                            //If something went wrong when fetching issue, notify
user
                            if (err) {
                                console.error(err);
                                errors.comment = "Error occurred";
                                return res.status(500).json(errors);
                            }

                            //If issue was fetched, remove comment
                            const newCommentArray = issue.comments.filter(
                                (comment) => comment.toString() !==
commentId.toString()
                            );

                            //Update issue with new comments

                            IssueModel.findByIdAndUpdate(issue._id, {
                                comments: newCommentArray,
                            }).exec(function (err, issue) {
                                //If something went wrong when updating issue
comments, notify user
                                if (err) {
                                    console.error(err);
                                    errors.comment = "Error occurred";
                                    return res.status(500).json(errors);
```

```
                                }

                                //If everything went well, notify user
                                messages.comment = "Comment successfully removed";
                                return res.json(messages);
                            });
                        });
                    });
                } else {
                    let errors = {};
                    errors.general = "Opps! Something went wrong";
                    return res.status(400).json(errors);
                }
            },
            editComment: (req, res) => {
                if (req.params.commentId) {
                    let messages = {};
                    let errors = {};
                    const commentId = req.params.commentId;

                    const {description} = req.body.comment;

                    CommentModel.findByIdAndUpdate(commentId, {
                        description: description,
                    }).exec(function (err, comment) {
                        //If error occurred when updating comment with edit, notify
user
                        if (err) {
                            console.error(err);
                            errors.comment = "Error occurred";
                            return res.status(500).json(errors);
                        }

                        //If everything went well, notify user
                        messages.comment = "Comment successfully updated";
                        return res.json(messages);
                    });
                } else {
                    let errors = {};
                    errors.general = "Opps! Something went wrong";
                    return res.status(400).json(errors);
                }
            },
        };
```