СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки

Кафедра інформаційних технологій та програмування

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної випускної роботи

освітній ступінь бакалавр

спеціальність 121 „Інженерія програмного забезпечення"

спеціалізація „Інженерія програмного забезпечення"

на тему „Система «Task management»"

Виконав: студент групи ІПЗ-19д ________ О.О. Малахов

(підпис) (ініціали і прізвище)

Керівник ________ В.О. Лифар

(підпис) (ініціали і прізвище)

Завідувач кафедри ІТП ________ В.О. Лифар

(підпис) (ініціали і прізвище)

Рецензент________________

Київ - 2023

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки
Кафедра інформаційних технологій та програмування

Освітній ступінь бакалавр

спеціальність 121 „Інженерія програмного забезпечення"
(шифр і назва спеціальності)
спеціалізація „Інженерія програмного забезпечення"
(назва спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТП
_________Лифар В.О.
"____" __________2023 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ ВИПУСКНУ РОБОТУ СТУДЕНТУ
________________Малахову Олександру Олександровичу___________________
(прізвище, ім'я, по батькові)

1. Тема роботи Система «Task management»_________________________________
____
Керівник роботи ____Лифар Володимир Олексійович , д.т.н. _______________
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджений наказом університету від "26"__04___2023 року №240/15.15-ОД_

2. Строк подання студентом роботи______05.06. 2023 р. ______________
3. Вихідні дані до роботи___Об'єктом даної розробки є процес створення системи Task Management.____
____
4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)_Розробка системи Task Management, архітектура та основні класи, створення та реалізація проекту.____
____
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників)
____
____
____

6. Консультанти розділів роботи

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

7. Дата видачі завдання_24.03.2023____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання кваліфікаційної випускноїроботи	Строк виконання етапів	Примітка
1	Одержання завдання на виконання роботи	24.03.2023	
2	Укладання і погодження з керівником плануі етапів виконання роботи	02.04.2023	
3	Узагальнення даних літературних джерел,укладання розділу «Аналіз предметної галузі»	08.04.2023	
4	Аналіз шляхів виконання завдання. Вибір іпогодження з керівником оптимального шляху	09.04.2023	
5	Розробка та тестування програмного продукту	21.05.2023	
6	Укладання, оформлення та погодженняпояснювальної записки з керівником	24.05.2023	
7	Здача готової пояснювальної записки накафедру	27.05.2023	
8	Укладання доповіді і презентації	30.05.2023	

Студент ________________ О.О. Малахов______

(підпис) (ініціали і прізвище)

Керівник роботи _____________ ___В.О. Лифар_____

(підпис) (ініціали і прізвище)

ЛИСТ ПОГОДЖЕННЯ І ОЦІНЮВАННЯ

дипломної роботи студента гр. ІПЗ-19д Малахова О.О.

Науковий керівник

Доцент, д.т.н. ____________ Лифар В.О.

Оцінка наукового керівника: _________________

Рецензент ____

ПІБ, місто роботи, посада

Оцінка рецензента: _________________________________

Кінцева оцінка за результатами захисту:

____

Голова ЕК

_________________ ЛифарВ.О.

підпис

РЕФЕРАТ

Робота містить: 52 сторінки основного тексту, 20 рисунків, 18 використаних джерел.

Метою випускної кваліфікаційної роботи є вивчення особливостей розробки системи Task management.

В ході розробки системи Task management були розглянуті особливості організації роботи даних систем у середовищі інтернет, розглянуто поведінку користувачів в системі управління завданнями, проаналізовані існуючі системи управління завданнями, була створена система управління завданнями, проведено її тестування. Вироблено опис процесу розробки і тестування системи. Реалізовано, а також описаний користувальницький інтерфейс, зроблені знімки екранних форм програмного засобу. Продемонстровано результат виконаної роботи.

Система задовольняє всім вимогам, пред'явленим в технічному завданні.

ЗМІСТ

ВСТУП 5

РОЗДІЛ 1. АНАЛІЗ МЕТОДІВ І ЗАСОБІВ СИСТЕМИ "TASK MANAGEMENT" 7

1.1. Особливості організації роботи систем "task management" у середовищі інтернет 7

1.2. Поведінка користувача в системі управління завданнями 9

1.3. Аналіз існуючих систем "Task management" 11

Висновок до розділу 1 23

РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ 25

2.1. Вибір і опис методології програмування 25

2.2. Аналіз вимог 27

2.3. Проектування 30

2.4 Створення дизайну 31

2.5. Детальне проектування 32

Висновок до розділу 2 35

РОЗДІЛ 3. РОЗРОБКА СИСТЕМИ 37

3.1. Вибір технологій 37

3.2. Розробка серверної частини 40

3.3. Розробка клієнтської частини 46

3.4. Тестування 53

Висновок до розділу 3 55

ВИСНОВКИ 56

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 57

ВСТУП

У сучасному світі, що стрімко розвивається, важливість ефективного управління проектами та завданнями стає все більш очевидною для успішної реалізації будь-якого бізнесу чи організації. Це призводить до зростаючої потреби в системах управління завданнями, які сприяють підвищенню продуктивності, оптимізації ресурсів та полегшенню співпраці між учасниками проектів. Тема цієї дипломної роботи актуальною, бо присвячена розробці системи "Task Management", яка має на меті вирішити ці проблеми та покращити процес управління завданнями.

Метою даної роботи є розробка та впровадження інтегрованої системи управління завданнями, яка забезпечить максимальну ефективність процесу координації робіт, контролю та моніторингу різноманітних завдань в різних організаціях. Основні задачі, що стоять перед дипломною роботою, включають аналіз існуючих систем управління завданнями, розробку концепту нової системи, проектування архітектури та структури бази даних, розробку програмного забезпечення, тестування та впровадження готового продукту.

Виконання даної роботи буде базуватися на таких основних підходах і методах: аналіз існуючих систем управління завданнями та визначення їх сильних та слабких сторін; дослідження сучасних технологій та методів управління проектами; проектування системи на основі моделі клієнт-сервер; використання гнучкої методології розробки програмного забезпечення (Agile) для постійного удосконалення продукту.

Очікується, що розроблена система "Task Management" буде характеризуватися високим рівнем адаптивності та індивідуалізації для кожного користувача, забезпечуючи зручний інтерфейс та широкі можливості налаштувань. Окрім того, система повинна підтримувати різні рівні доступу, що дозволить надавати необхідні повноваження учасникам проектів залежно від їх ролей та обов'язків. Це сприятиме кращій координації робіт та забезпеченню високого рівня безпеки інформації.

Для забезпечення гнучкості та масштабованості розробленої системи, важливо використовувати сучасні технології та платформи, такі як хмарні обчислення, веб-сервіси та API. Такий підхід дозволить інтегрувати систему "Task Management" з іншими інструментами та сервісами, які вже використовуються в організаціях, тим самим полегшуючи процес впровадження та навчання користувачів.

У процесі розробки системи буде здійснено порівняльний аналіз існуючих рішень на ринку, що допоможе виявити їх переваги та недоліки та уникнути потенційних помилок у розробці власної системи. Основними критеріями оцінки існуючих систем будуть: функціональність, зручність використання, гнучкість, безпека, масштабованість та інтеграційні можливості.

Результатом даної дипломної роботи стане розроблена система "Task Management", яка допоможе організаціям підвищити ефективність управління проектами та завданнями, оптимізувати використання ресурсів.

РОЗДІЛ 1. АНАЛІЗ МЕТОДІВ І ЗАСОБІВ СИСТЕМИ "TASK MANAGEMENT"

1.1. Особливості організації роботи систем "task management" у середовищі інтернет

Управління проектами та завданнями є важливою частиною роботи будь-якої організації або бізнесу. Від ефективності цього процесу залежать результати роботи та успіх компанії. У цьому розділі ми проаналізуємо методи та засоби, які використовуються у сучасних системах управління завданнями, і спробуємо визначити ті, які можуть бути застосовані в розробці системи "Task Management".

Методики управління проектами:

– водоспадна методологія (Waterfall) - послідовний підхід до управління проектами, який передбачає розбиття проекту на етапи, які повинні виконуватися один за одним. Мінусом цього підходу є негнучкість та відсутність можливості адаптації до змін;

– гнучкі методології (Agile) - ряд підходів, які передбачають гнучкість, адаптивність та співпрацю між учасниками проектів. До них відносяться Scrum, Kanban та інші. Ці методології дозволяють краще реагувати на зміни та прискорити процес розробки [1].

"Task Management" відноситься до процесу організації, планування, відстеження та контролю завдань, які виконуються в межах проекту або робочого процесу. Це включає створення, призначення, розподіл ресурсів, встановлення термінів та пріоритетів, а також моніторинг статусу завдань та їх виконання. Системи "Task Management" допомагають командам та індивідуальним користувачам керувати своїми завданнями та проектами ефективніше, забезпечуючи відповідність термінам та якості роботи.

Організація роботи систем "Task Management" у середовищі Інтернет має свої особливості, які забезпечують ефективність, доступність та зручність управління проектами та завданнями. Основні особливості включають:

– веб-доступність: інтернет-орієнтовані системи управління завданнями доступні з будь-якого комп'ютера або мобільного пристрою з підключенням до Інтернету. Це дозволяє користувачам працювати з системою з будь-якої точки світу та сприяє мобільності та гнучкості роботи;

– колаборація в реальному часі: середовище Інтернет дозволяє організувати співпрацю між учасниками проекту в реальному часі. Користувачі можуть спілкуватися, обмінюватися файлами, коментувати та оновлювати завдання негайно, що підвищує продуктивність та координацію;

– автоматизація та сповіщення: інтернет-орієнтовані системи "Task Management" дозволяють автоматизувати рутинні задачі та сповіщати про їх виконання;

– інтеграція з іншими сервісами: системи "Task Management" у середовищі Інтернет можуть інтегруватися з різноманітними сервісами, такими як електронна пошта, календарі, системи обліку часу, CRM тощо. Це полегшує обмін даними та співпрацю між різними інструментами, які використовуються в організації;

– хмарні рішення: багато систем "Task Management" розміщуються на хмарних серверах, що забезпечує високу доступність, масштабованість та надійність. Хмарні рішення також полегшують процес оновлення та розширення системи [2].

Тепер розглянемо підходи таск-менеджменту. Agile — це філософія та набір принципів розробки програмного забезпечення, відповідно до яких вимоги та рішення розвиваються завдяки спільним зусиллям міжфункціональних команд. Він виступає за адаптивне планування, еволюційний розвиток, ранню поставку, постійне вдосконалення та гнучкість у відповідь на зміни. Термін «Agile» був популяризований Agile Manifesto в 2001

році. Загальні методології під егідою Agile включають Scrum, Kanban, Lean та Extreme Programming (XP) [3].

Kanban — це гнучкий метод управління проектами, розроблений для візуалізації роботи, обмеження незавершеної роботи (WIP) і максимізації ефективності (або потоку). Спочатку він був розроблений компанією Toyota для підвищення ефективності виробництва. Kanban дозволяє командам візуально керувати своєю роботою, розбиваючи її та візуалізуючи етапи робочого процесу на дошці Kanban, яка зазвичай складається з вертикальних стовпців, що представляють етапи роботи. Кожна частина роботи, представлена карткою, рухається зліва направо під час проходження цих етапів [4].

Scrum — це гнучка структура, яка використовується в основному для розробки продуктів і програмного забезпечення, хоча її також застосовують і в інших сферах. Структура заохочує спільну роботу команди для вирішення складних проблем, наголошуючи на гнучкості, креативності та продуктивності. Принципи та цінності, якими керується Scrum, викладені в Agile Manifesto та Scrum Guide [5].

1.2. Поведінка користувача в системі управління завданнями

Поведінка користувача в системі управління завданнями може бути різноманітною в залежності від ролі користувача, обов'язків та типу завдань. Основні дії та поведінка користувачів незалежно від ролі користувача будь то тестувальник, розробник, замовник або project manager включають:

- створення завдань: користувачі можуть створювати нові завдання, вказуючи необхідні деталі, такі як назва, опис, відповідальні особи, терміни та пріоритети;
- редагування завдань: користувачі можуть змінювати існуючі завдання, оновлюючи інформацію або додавання додаткових деталей;

– призначення завдань: користувачі можуть призначати завдання іншим учасникам команди або себе, встановлюючи відповідальність за виконання завдань;

– відстеження статусу завдань: користувачі переглядають поточний статус завдань, таких як "не розпочато", "в роботі", "завершено" або "відкладено". Це допомагає контролювати прогрес та вчасно ідентифікувати проблеми чи затримки;

– відмітка завдань як виконаних: коли завдання виконано, користувачі можуть позначити його як "завершено", що оновлює статус завдання та допомагає відстежувати прогрес проекту;

– комунікація та співпраця: користувачі можуть обмінюватися повідомленнями, коментарями та документами з іншими учасниками команди. Це сприяє координації та забезпечує ефективну комунікацію між учасниками проекту;

– використання фільтрів та сортування: користувачі можуть використовувати різні фільтри та параметри сортування для кращого відображенення та організації своїх завдань. Це може включати сортування за пріоритетами, датами, статусами або відповідальними особами;

– налаштування сповіщень: користувачі можуть налаштувати сповіщення для отримання повідомлень про зміни в завданнях, наближення термінів або інші важливі події, що відбуваються в системі управління завданнями;

– робота з календарем: користувачі можуть використовувати календар для планування та відстеження термінів виконання завдань, а також для координації роботи з іншими учасниками команди;

– інтеграція з іншими інструментами: користувачі можуть інтегрувати системи управління завданнями з іншими програмами та сервісами, такими як електронна пошта, системи обліку часу, CRM тощо, для підвищення продуктивності та зручності роботи;

– генерація звітів: користувачі можуть створювати звіти про прогрес проекту, завдань та роботи команди, що допомагає контролювати продуктивність та ідентифікувати можливі проблеми або можливості для поліпшення [1].

Поведінка користувачів в системах управління завданнями в значній мірі залежить від вимог та особливостей конкретного проекту або робочого процесу. Однак, незалежно від ситуації, головною метою користувачів є підвищення продуктивно.

1.3. Аналіз існуючих систем "Task management"

Існує велика кількість систем "Task Management" на ринку, і кожна з них має свої особливості, функціональні можливості та характеристики. Далі представлено аналіз деяких популярних систем управління завданнями.

Trello

Trello – це візуальна система управління завданнями, яка використовує підхід Kanban. Користувачі можуть створювати дошки, списки та картки для організації та відстеження завдань. Trello забезпечує простоту використання, гнучкість та інтеграцію з різними інструментами.

Основні компоненти та функції Trello:

– дошки: дошка Trello — це основна організаційна одиниця системи Trello, що представляє проект або робочий процес. Кожна дошка — це окремий простір, де ви можете стежити за проектом або організувати будь-що, над чим працюєте;

– списки: усередині кожної дошки ви створюєте списки, які використовуються для поділу дошки на різні етапи робочого процесу, категорії чи будь-що інше, що має сенс для вашого проекту. Наприклад, у проекті розробки програмного забезпечення списки можуть бути «Виконати», «Виконується» та «Готово»;

– картки: картки є основними одиницями дошки, подібними до наліпок, які ви можете розмістити на фізичній дошці. Вони представляють завдання або елементи. До карток можна додавати деталі, як-от описи, коментарі, вкладення, терміни виконання, мітки та контрольні списки;

– співпраця: Trello розроблено для підтримки командної співпраці. Ви можете додати скільки завгодно людей до дошки, і всі вони зможуть створювати, редагувати, переміщувати та коментувати картки. Trello оновлюється в режимі реального часу, тому всі співавтори можуть бачити зміни, коли вони відбуваються;

– сповіщення: Trello надає систему сповіщень, яка дає вам знати, коли відбувається щось важливе. Ви отримаєте сповіщення, коли хтось згадає вас у коментарі, призначить вам картку, оновить картку, яку ви присвоїли, тощо;

– power Ups: Power Ups – це додаткові функції, які розширюють функціональні можливості дощок Trello. Є бонуси для перегляду календаря, голосування, інтеграції Google Drive, інтеграції Slack і багато іншого. Деякі бонуси розроблено Trello/Atlassian, а інші – сторонніми розробниками;

– інтеграція: Trello забезпечує інтеграцію з багатьма популярними інструментами, такими як Slack, Google Calendar, GitHub, Jira та іншими. Ці інтеграції допомагають Trello вписатися у ваш існуючий набір інструментів і робочі процеси;

– мобільність: Trello розроблено для використання на будь-якому пристрої. Він надає програми для iOS, Android і Інтернету, які синхронізуються в реальному часі [6].

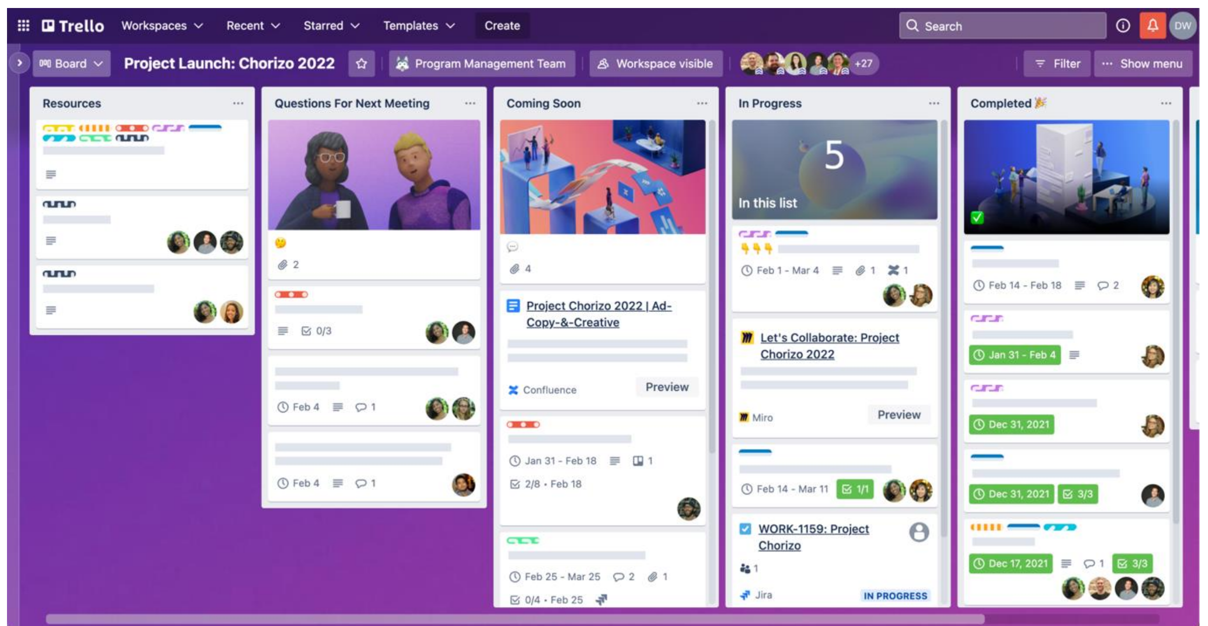

Рис 1.1. Дошка Trello [6].

ClickUp

ClickUp – це високофункціональний інструмент для управління завданнями та проектами, який допомагає командам організовувати роботу і підвищувати продуктивність. Він поєднує в собі різні аспекти управління проектами, співпраці, комунікації та автоматизації в одній платформі. Має декілька відмінностей від інших платформ таск-менеджменту, а саме:

– гнучкість відображення завдань: ClickUp пропонує кілька варіантів відображення завдань, включаючи список, дошку (на кшталт Kanban), таймлайн, календар, та Gantt-діаграму. Користувачі можуть вибирати відображення, що найкраще відповідає їх потребам;

– ієрархічна структура: ClickUp пропонує глибоку ієрархію для організації роботи, яка включає Спейси (Spaces), Проекти (Folders), Списки (Lists), та завдання (Tasks). Кожний рівень можна налаштувати для конкретних потреб команди;

– звітність та аналітика: ClickUp пропонує детальні звіти про прогрес проекту, продуктивність команди, та використання часу;

– кастомізація: ClickUp дозволяє користувачам налаштувати робочий простір відповідно до їхніх потреб, включаючи кастомізацію статусів завдань, використання шаблонів та автоматизацію рутинних процесів [7].

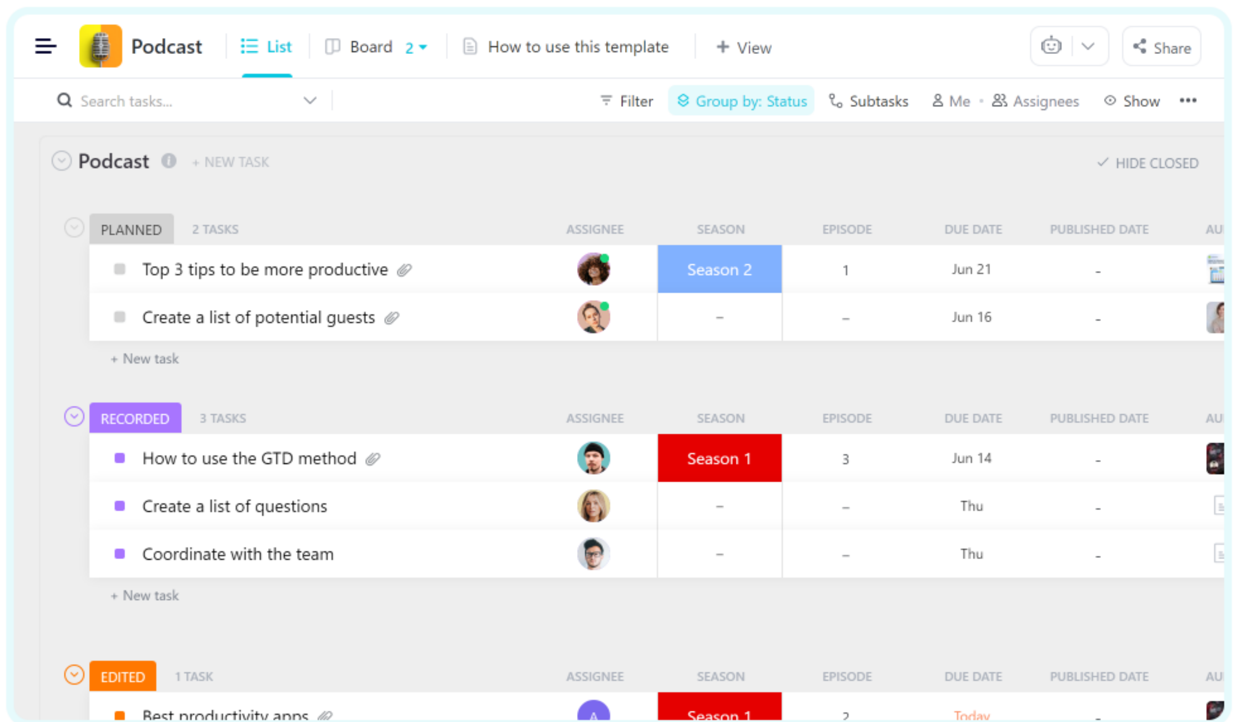

Рис. 1.2. Робочий простір ClickUp [7].

Jira

Jira – це потужна система управління проектами, розроблена Atlassian, яка початково була створена для потреб розробників програмного забезпечення, але з часом розширилася і тепер використовується в різних областях. Основні особливості та функції Jira:

– управління завданнями та проектами: Jira дозволяє створювати, відстежувати, та управляти завданнями та проектами, використовуючи гнучкі налаштування процесів та ієрархії завдань;

– проблеми: Основні одиниці роботи в Jira називаються проблемами, які можуть представляти будь-що: від програмних помилок, завдань, історій, епічних творів або навіть спеціальних типів залежно від потреби;

– робочі процеси. Однією з найпотужніших функцій Jira є настроювані робочі процеси. Робочий процес — це шлях, який проходить проблема від створення до завершення. Робочі процеси можна налаштувати за допомогою стільки статусів і переходів, скільки потрібно для вашого проекту, що дозволить вам точно відобразити процес вашої команди в Jira;

– дошки: Jira надає дошки Scrum і Kanban для візуалізації прогресу роботи та сприяння гнучким методологіям. Дошка відображає проблеми з одного або кількох проектів у стовпцях, які представляють робочий процес команди;

– звітування: Jira пропонує широкі інструменти звітності, включаючи діаграми вигоряння та вигоряння для команд Scrum, кумулятивні діаграми потоків для команд Kanban, діаграми швидкості тощо. Ці звіти допомагають командам відстежувати прогрес і отримувати статистичні дані;

– користувальницькі поля: проблеми Jira можна налаштувати за допомогою спеціальних полів для відстеження додаткової інформації. Наприклад, можна створити поле для «Рівня пріоритету» або «Впливу на клієнта»;

– розширений пошук (JQL): мова запитів Jira (JQL) є потужною мовою пошуку, яка дозволяє користувачам знаходити проблеми на основі майже будь-яких критеріїв. Ця функція особливо корисна для створення спеціальних фільтрів або звітів;

– інтеграція: Jira добре інтегрується з іншими інструментами розробки. Він має вбудовану інтеграцію з Bitbucket і Confluence, які також належать Atlassian, і може бути інтегрований із широким спектром інших інструментів, таких як GitHub, Slack, Jenkins тощо;

– контроль доступу: Jira надає надійні функції контролю доступу, що дозволяє адміністраторам контролювати, хто до чого має доступ [8].

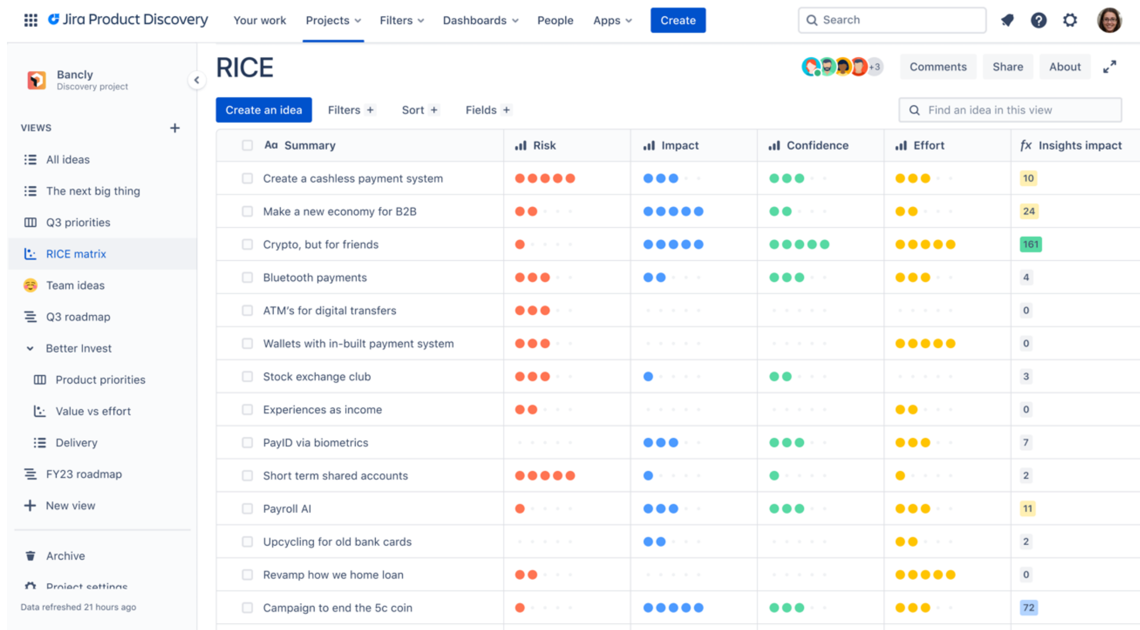

Рис. 1.3. Дошка Jira [8].

Wrike

Wrike – це веб-базована платформа управління проектами, яка забезпечує повний набір інструментів для планування, стеження та звітності про проекти. Зробимо огляд особливостей цієї системи. Система Wrike має наступні особливості:

– централізоване керування проектами: Wrike дає можливість зберігати всі завдання, файли, обговорення і деталі проекту в одному місці;

– робочий потік: Wrike має функції автоматизації робочого потоку, які дозволяють автоматично оновлювати статуси завдань та виконувати інші рутинні дії;

– колаборація: Wrike дозволяє командам співпрацювати в реальному часі, з можливістю додавання коментарів до завдань, ділитися файлами, і вести обговорення в контексті проекту [9].

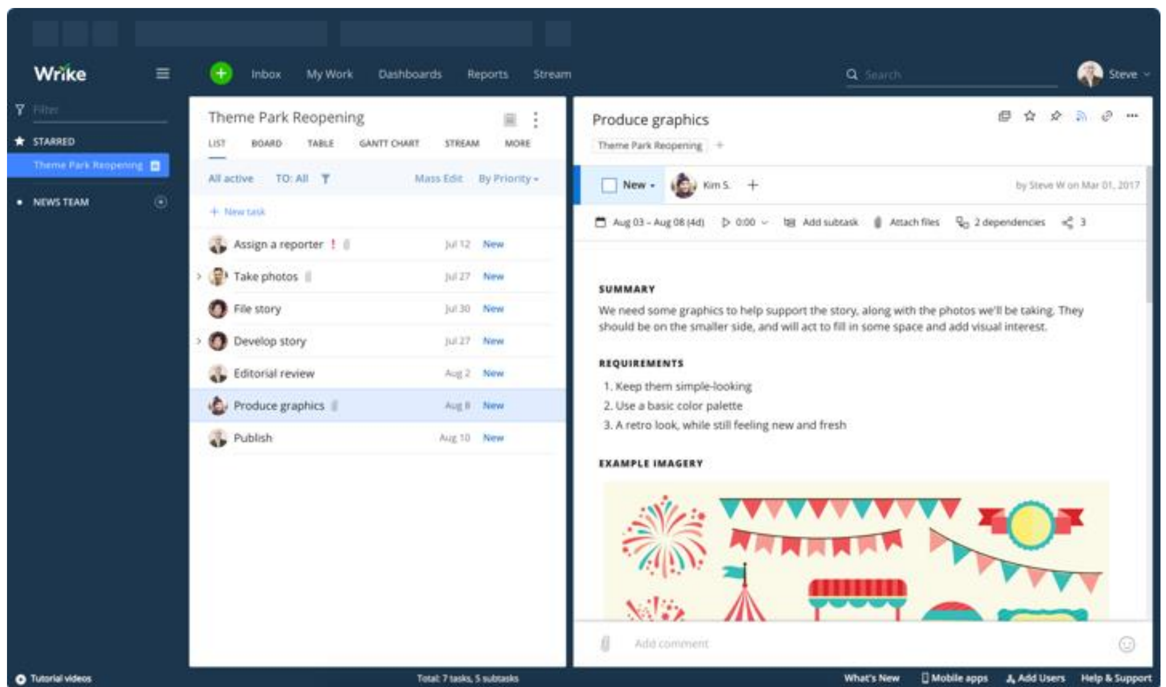

Рис. 1.4. Робочий простір Wrike [9].

Todoist

Досить зручний таск-трекер, що дозволяє розподілити завдання на день, тиждень і т. д. Для завдань є теги, дедлайн, пріоритети, можливість прикріпити файли з Google-диску або Dropbox. Командну роботу покращують канбан-дошки, фільтри, можливість створення завдань через Gmail та Outlook, аналітика активності співробітників. Далі розглянемо основні компоненти та функції Todoist:

– завдання: завдання є основними одиницями роботи в Todoist. Кожне завдання може мати назву, термін виконання, рівень пріоритету та пов’язані мітки та проекти. Завдання також можуть мати нагадування, коментарі та вкладення, якщо ви використовуєте платний план;

– підзавдання: Todoist дозволяє створювати ієрархічні структури завдань, додаючи до завдань підзавдання. Це може бути корисним для розбиття великих завдань на легші частини;

– проекти: у Todoist можна групувати пов’язані завдання в проекти. Проекти можуть бути позначені кольором для легшого візуального розпізнавання та можуть містити стільки завдань і підзадач, скільки потрібно;

– мітки та фільтри: мітки — це спосіб класифікації завдань у Todoist, тоді як фільтри дозволяють створювати власні перегляди ваших завдань на основі таких критеріїв, як термін виконання, рівень пріоритету, мітка тощо;

– повторювані завдання: Todoist підтримує повторювані завдання. Ви можете встановити завдання, яке повторюватиметься щодня, щотижня, щомісяця чи навіть щороку, або використовувати більш складні повторювані шаблони;

– співпраця: Todoist дозволяє вам ділитися проектами з іншими для спільної роботи. Учасники спільного проекту можуть додавати, виконувати та коментувати завдання;

– бали та смуги карми: Todoist включає функцію гейміфікації під назвою Karma. Ви заробляєте очки карми, виконуючи завдання, використовуючи розширені функції та зберігаючи продуктивність. Це може бути цікавим способом залишатися мотивованим;

– інтеграція: Todoist може інтегруватися з великою кількістю інших інструментів, таких як Google Calendar, Amazon Alexa, Dropbox та багато інших;

– шаблони: Todoist надає різноманітні шаблони для поширених випадків використання, таких як керування списком покупок, планування відпустки або відстеження фітнесу. Ці шаблони можуть забезпечити швидкий початок ефективного використання Todoist [10].

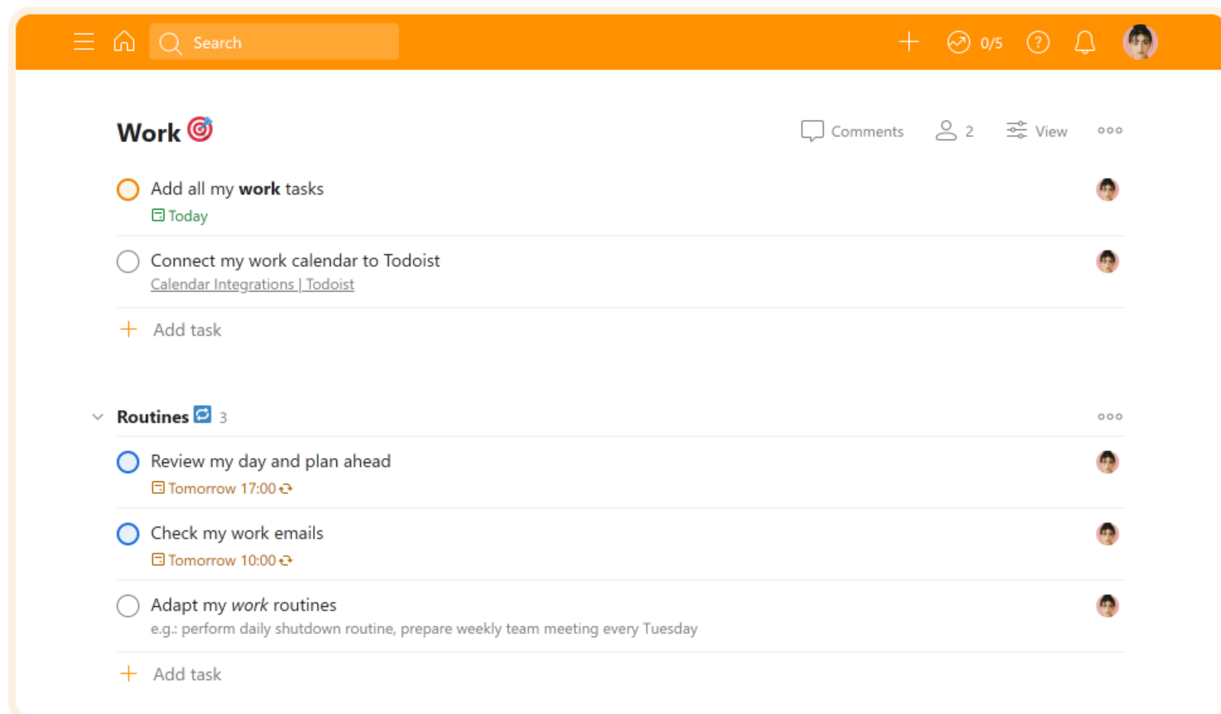

Рис. 1.5. Робочий простір Todoist [10].

Hive

Зручний таск-трекер, що підвищує ефективність команди для рахунку адаптації під потреби користувачів. Наприклад, членам команди доступні індивідуальні прилади, в яких видні призначені кожному з них завдання, розділені на категорії: в роботі, виконані. Можна прикріпити файли, позначити теги, практично одним клацанням миші перемістити завдання між проектами або позначити нового виконавця. Також є аналітика особистої та командної продукції. Основні функції Hive:

– проекти та дії: у Hive проект — це набір дій (еквівалентних завданням на інших платформах). Кожна дія може мати назву, правонаступника, термін виконання, мітку та опис. Дії можна розбити на менші підрозділи.

– різні перегляди: Hive дозволяє перемикатися між різними видами проекту відповідно до вашого робочого процесу чи вподобань, включаючи діаграму Ганта, дошку Kanban, календар, таблицю та портфоліо;

– Hive Forms: ця функція дозволяє створювати форми для стандартизації створення завдань. Надсилання форми можна автоматично перетворювати на дії;

– Hive Mail і Hive Chat: Hive включає електронну пошту та чат у свою платформу, що дозволяє користувачам керувати своєю папкою "Вхідні" та спілкуватися зі своєю командою, не виходячи з програми;

– Hive Notes: ця функція призначена для спільного створення нотаток із можливістю призначати завдання безпосередньо з нотатки;

– відстеження часу: Hive надає функції відстеження часу, дозволяючи користувачам записувати час, витрачений на кожну дію;

– ресурси та керування робочим навантаженням: за допомогою функції керування ресурсами Hive ви можете забезпечити рівномірний розподіл роботи між членами вашої команди;

– аналітика та звітність: інформаційна панель аналітики Hive надає інформацію про продуктивність команди, стан проекту тощо;

– автоматизація: Hive надає можливості автоматизації, які допомагають зменшити ручну повторювану роботу. Наприклад, ви можете автоматизувати оновлення статусу, сповіщення про призначення тощо;

– інтеграція: Hive інтегрується з понад тисячею програм через Zapier і має вбудовану інтеграцію з такими інструментами, як Salesforce, Slack і Zoom;

– обмін файлами та співпраця: Hive інтегрується з платформами для обміну файлами, такими як Google Drive, OneDrive та Dropbox. Ви можете прикріплювати файли до дій і навіть спільно працювати над ними в Hive [11].

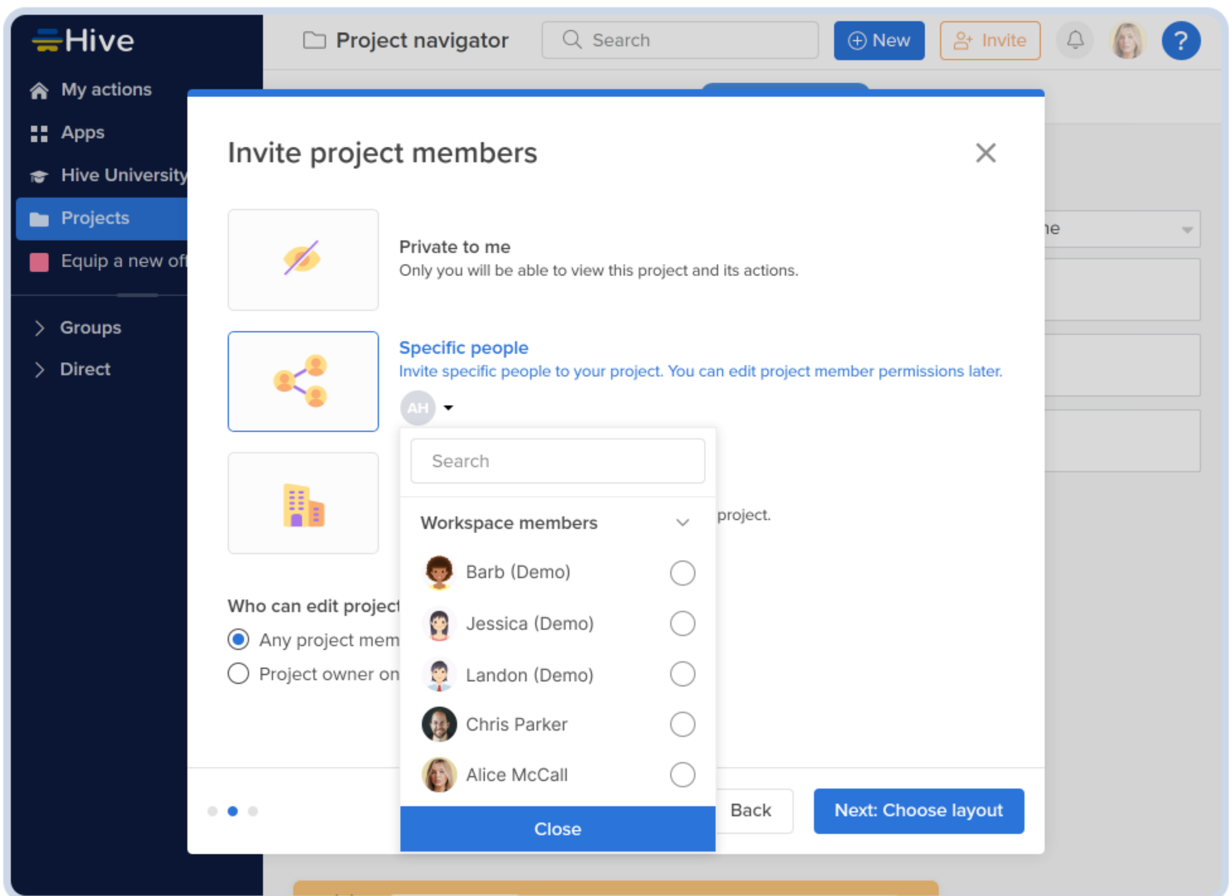

Рис. 1.6. Робочий простір Hive [11].

Asana

Хмарний продукт, що дозволяє створювати списки завдань та призначати їм виконавців, а також вказувати дату завершення. Крім того, завданням можна змінювати рівень важливості, коротко описувати їхню сутність, прикріплювати файли, залишати коментарі. Далі детальніше розглянемо деякі ключові функції Asana:

– завдання та підзавдання: основною робочою одиницею Asana є завдання, яке можна призначити членам команди, вказати дату виконання та розбити на підзавдання. Кожне завдання може містити опис, вкладення та коментарі для співпраці;

– проекти: Asana дозволяє групувати завдання в проекти, які можна візуалізувати кількома способами – у вигляді списку, дошки у стилі Канбан, шкали часу або календаря;

– настроювані поля: Asana підтримує настроювані поля, що дозволяє відстежувати додаткову інформацію про завдання, що відповідає потребам вашого робочого процесу чи проекту;

– розділи та стовпці: для кращої організації проектів Asana пропонує розділи (у режимі перегляду списку) і стовпці (у режимі перегляду дошки), щоб згрупувати пов’язані завдання;

– хронологія та залежності: у режимі перегляду шкали часу Asana забезпечує візуалізацію розкладу вашого проекту, подібну до Ганта. Ви можете намітити свій план, створити залежності між завданнями та налаштувати дати за потреби;

– цілі: функція Asana Goals дозволяє організаціям встановлювати та відстежувати стратегічні цілі, пов’язані з роботою команди;

– автоматизація (правила): Asana дозволяє вам автоматизувати рутинні завдання за допомогою правил. Наприклад, можна створити правило автоматичного переміщення завдання в певний розділ при зміні його статусу;

– коментування та співпраця: Asana полегшує командну співпрацю за допомогою коментарів до завдань, обговорень проекту та командних сторінок для ширших обговорень;

– звітність: Asana надає функції звітності, такі як оновлення статусу проекту та інформаційні панелі для відстеження ключових показників ефективності;

– інтеграція: Asana інтегрується з великою кількістю інших інструментів, таких як Google Drive, Slack, Microsoft Teams тощо, які можуть оптимізувати робочий процес і централізувати вашу роботу;

– портфоліо: функція «Портфоліо» дозволяє менеджерам миттєво контролювати статус кількох проектів. Ця функція доступна в планах Business і Enterprise [12].

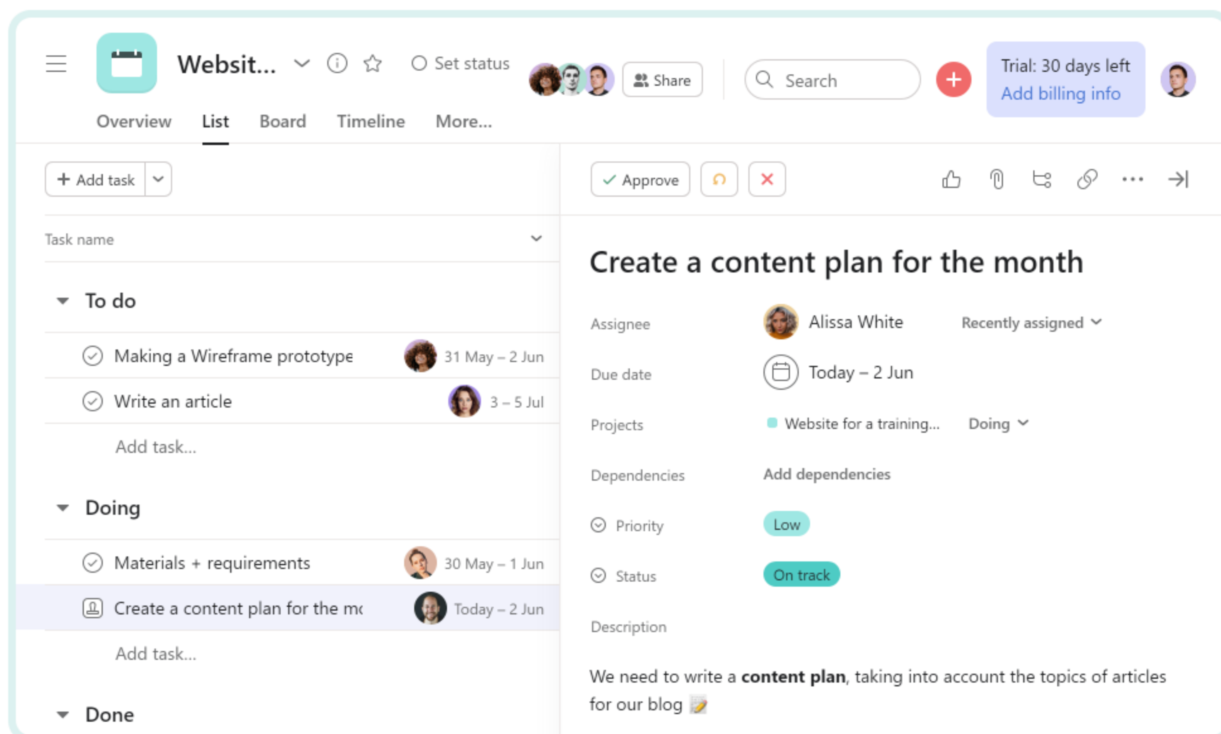

Рис. 1.7. Робочий простір Asana [12].

Висновок до розділу 1

Підсумовуючи, у цьому розділі розглянуто особливості організації систем керування завданнями в Інтернет-середовищі, поведінку користувачів у таких системах та проведено поглиблений аналіз існуючих систем керування завданнями.

Системи керування завданнями є невід’ємною частиною цифрового ландшафту, сприяючи продуктивності та спільним зусиллям як у професійній, так і в особистій сферах. Інтернет-середовище з його великою взаємопов’язаністю та доступом до ресурсів є ідеальною платформою для цих систем. Однак дизайн і організація цих платформ повинні враховувати такі фактори, як зручність використання, доступність і безпека. Технологічна архітектура, дизайн UI/UX та можливості інтеграції цих систем значно впливають на їхню ефективність і задоволеність користувачів.

Що стосується поведінки користувачів у системах керування завданнями, було виявлено, що шаблони значно відрізняються залежно від складності системи, інтуїтивно зрозумілого інтерфейсу та характеру завдань. У той час як деякі користувачі віддають перевагу простим, лінійним методам керування завданнями, інші схиляються до більш складних систем, які дозволяють створювати ієрархії та залежності завдань. Користувачі також тяжіють до систем, які забезпечують візуальну чіткість і пропонують значущі параметри налаштування. Розуміння такої поведінки може керувати розробкою більш зручних систем керування завданнями в майбутньому.

Нарешті, комплексний аналіз існуючих систем керування завданнями продемонстрував широкий спектр рішень, кожне з яких має унікальні сильні та слабкі сторони. Від простих додатків зі списками справ, таких як Google Tasks, до більш складних інструментів управління проектами, таких як Asana та Trello, ринок різноманітний. Очевидно, що успіх цих систем залежить не лише від їхніх функцій, але й від їх адаптованості до мінливих потреб користувачів і технологічного прогресу.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Вибір і опис методології програмування

Спершу треба визначити яку модель життєвого циклу, який потрібно використовувати при створенні веб системи.

Модель життєвого циклу програмного забезпечення (Software Development Life Cycle, SDLC) – це фреймворк або методологія, що визначає етапи та послідовність процесу розробки програмного забезпечення, від початкового аналізу до впровадження та підтримки. Модель життєвого циклу надає структуру та організацію для ефективного керування процесом розробки програмного забезпечення.

Деякі з найпоширеніших моделей життєвого циклу програмного забезпечення включають:

– водоспадну модель: лінійна модель, де кожна фаза виконується послідовно, від аналізу та проектування до реалізації, тестування та впровадження;

– ітеративну модель: модель, яка передбачає повторні ітерації розробки, де кожна ітерація включає аналіз, проектування, реалізацію та тестування;

– спіральну модель: модель, що поєднує елементи водоспадної моделі та ітеративного підходу, з фокусом на управлінні ризиками.

– Agile (Гнучку) модель: колекція методологій, таких як Scrum, Kanban та XP, які підкреслюють ітеративність, співпрацю замовника та гнучкість внесення змін;

– V-модель: модель, що передбачає паралельний процес тестування, який відбувається на кожному етапі розробки;

– RAD (Rapid Application Development): модель, що спрямована на швидку розробку та прототипування програмного забезпечення.

Кожна модель має свої переваги, недоліки та набір фаз, які можуть бути модифіковані або адаптовані відповідно до конкретного проекту або команди розробки [13].

Для розробки даної системи було обрано водоспадну модель. Водоспадна модель програмного циклу є лінійною послідовністю фаз, де кожна фаза виконується послідовно після завершення попередньої фази.

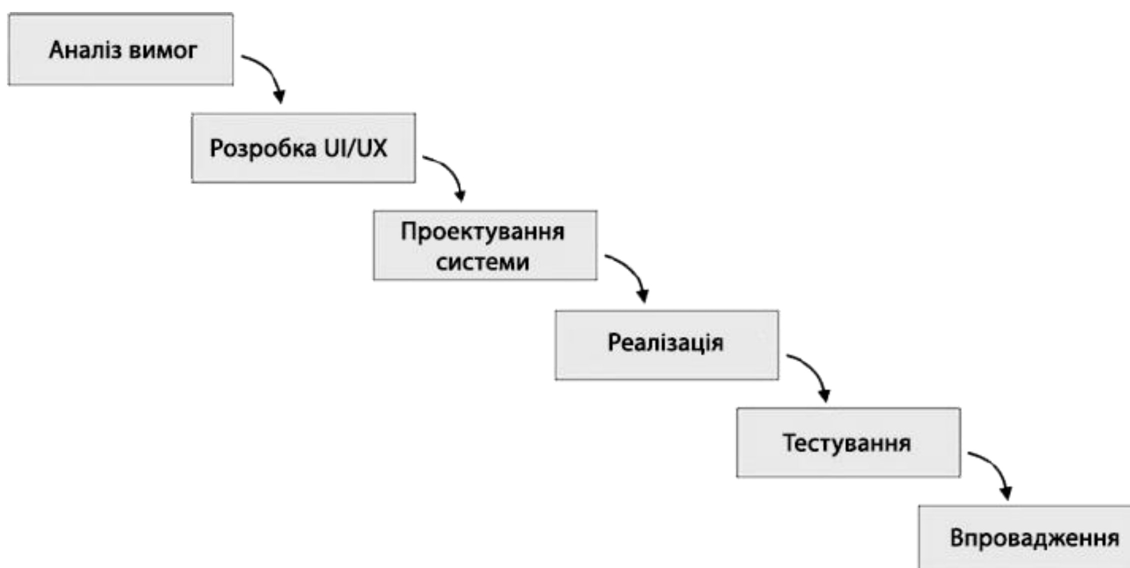

Рис. 2.1. Водоспадна модель життєвого циклу програмного забезпечення [13].

Розглянемо водоспадну модель на прикладі розробки системи управління задачами (Task Management System) на прикладі ClickUp використовуючи модель яка продемонстрована на рисунку 2.1:

– аналіз вимог: у цій фазі проводиться детальний аналіз вимог до системи Task Management. Розробники спілкуються зі замовником та потенційними користувачами, визначають функціональні та нефункціональні вимоги, обговорюють функціональні можливості та обмеження системи;

– проектування: на цій фазі розробляється архітектура системи Task Management. Визначаються компоненти системи, їх взаємозв'язки та функціональні можливості. Розробляється детальний план бази даних, дизайн інтерфейсу користувача та інші аспекти системи;

– реалізація: у цій фазі програмісти розпочинають реалізацію системи на основі розробленого проекту. Фронтенд на ReactJS та бекенд на ExpressJS розробляються окремо. Функціональні модулі, компоненти та інтерфейси користувача реалізуються відповідно до вимог та дизайну;

– тестування: на цій фазі проводиться тестування системи Task Management. Виконуються різні види тестування, такі як модульне тестування, функціональне тестування, інтеграційне тестування та валідація системи. Тестувальники перевіряють роботу системи, виявляють та виправляють помилки;

– впровадження: після успішного проходження тестування система Task Management готова до впровадження. Вона розгортається на серверах, налаштовується на відповідні конфігурації та починає використовуватись реальними користувачами. Усі потрібні документи та посібники для користувачів готуються та демонструються, щоб користувачі могли легко освоїти нову систему;

– підтримка: після впровадження, система потребує підтримки і управління для забезпечення безперебійного функціонування. Команда забезпечує вирішення виниклих проблем, оновлює систему згідно з вимогами безпеки та новими потребами користувачів. При необхідності вносяться зміни в архітектуру та функціонал системи [14].

Отже, водоспадна модель дозволяє систематично та послідовно розробити систему управління задачами на основі вимог та специфікацій ClickUp. Кожен етап допомагає забезпечити якість та ефективність системи, що в свою чергу сприяє задоволенню потреб користувачів.

2.2. Аналіз вимог

Розглянемо такі вимоги:

1. дизайн та інтерфейс користувача: система повинна мати зручний інтерфейс, що легко зрозуміти та навігувати для користувачів. Дизайн повинен

бути естетичним, а також забезпечувати ефективну взаємодію та зручне розміщення елементів на екрані. Вимоги до дизайну можуть включати використання сучасних технологій UI/UX, адаптивного дизайну для різних пристроїв, легку навігацію, візуалізацію даних, тощо;

2. масштабованість та продуктивність: система повинна бути спроектована з урахуванням масштабованості та продуктивності. Це може включати оптимізацію швидкодії системи, використання кешування даних, розподілення завдань на серверах, масштабованість обсягу даних, відсутність перебоїв у роботі системи при великому навантаженні, тощо;

3. реєстрація користувачів: система повинна мати можливість реєстрації нових користувачів, яка включає створення облікового запису з унікальним ідентифікатором і паролем. Можуть також бути вимоги до можливості входу через соціальні мережі або інші системи авторизації;

4. створення та керування задачами: користувачі повинні мати можливість створювати нові задачі, присвоювати їм заголовки, опис, терміни виконання, пріоритети, статуси та інші атрибути. Система повинна надавати зручний інтерфейс для перегляду, редагування та видалення задач;

5. управління проектами та проектними групами: система може мати можливості для створення та керування проектами, а також надання доступу до задач та інформації про проекти різним групам користувачів. Це може включати можливість створення команд, назначення ролей та повноважень, спільного доступу до завдань та спілкування в рамках проекту;

6. спільна робота та комунікація: система може мати можливості для комунікації та спільної роботи користувачів. Це може включати коментування завдань, обговорення проектів, спільний доступ до файлів та документації, повідомлення та сповіщення;

7. безпека та доступ до даних: вимоги до безпеки можуть включати захист конфіденційної інформації, шифрування даних, контроль доступу до завдань та проектів, резервне копіювання даних тощо;

8. звіти та аналітика: система може мати можливості для генерації звітів та аналітики, які надають інформацію про прогрес, продуктивність, терміни виконання, завдання за пріоритетами тощо;

9. сумісність та інтеграція: система повинна бути сумісною з різними операційними системами, браузерами та пристроями. Також можуть бути вимоги до можливості інтеграції з іншими інструментами або сервісами, наприклад, зберіганням даних у хмарі, електронною поштою, календарями, чат-системами тощо.

На веб-сайті велике значення має якість його виконання і відповідність певним стандартам. Недоліки у відображенні або функціональності можуть призвести до втрати відвідувачів. Для досягнення успішного розвитку та підтримки веб-сайту, важливо враховувати наступні вимоги:

1. швидкодія: веб-сайт повинен завантажуватися швидко, оскільки користувачі не будуть чекати довго на відображення сторінки. Короткий час завантаження сприяє покращенню взаємодії з веб-сайтом та збільшенню задоволеності відвідувачів;

2. сумісність з різними браузерами: веб-сайт повинен відображатися коректно на різних версіях браузерів, щоб забезпечити оптимальний досвід для всіх користувачів. Підтримка старших і новіших версій браузерів є важливою для того, щоб уникнути проблем з відображенням та забезпечити зручну навігацію;

3. розділення структури, стилів та скриптів: розділення HTML-коду, CSS-стилів та JavaScript-скриптів на окремі файли полегшує підтримку та зміну веб-сайту. Це також дозволяє кешувати стилі та скрипти, що сприяє зменшенню часу завантаження сторінок.

Відповідність цим принципам допомагає забезпечити доступність, сумісність, швидкість завантаження та відображення сайту, а також спрощує підтримку та модифікацію в майбутньому, що є дуже важливою частиною у випадку, якщо фінальна система буде привабливою для користувачів та зможе принести комерційний прибуток: або зацікавити інвесторів.

2.3. Проектування

Проектування – це процес формування моделі системи, виявлення того методу її розробки, який відповідає критеріям функціональності, при цьому враховуючи встановлені обмеження та доступні технології.

Клієнтська частина:

1. створення інтерфейсу за допомогою передових інструментів, мов розмітки та програмування;
2. адаптування для різних пристроїв. Гарантоване правильне відтворення інтерфейсу на всіх пристроях. В даному випадку системі необхідне коректне відображення для пристроїв з розміром екрану від 900 пікселів в ширину, розробка адаптивності цієї системи для мобільних пристроїв не е пріоритетним завданням без отримання певної аудиторії;
3. ефективний код та медіа-файли для швидкого завантаження і роботи інтерфейсу. Перш за все йдеться про відсутність рекурсій, затримок в отриманні даних з серверу та повторного використання деяких компонентів і їх можливе розширення завдяки наслідуванню. Також слід пам’ятати про оптимізацію медіа файлів, головна сторінка не має містити великих за обсягом фото/відео матеріалів.

Серверна частина:

1. розробка серверної частини за допомогою найновіших технологій і інструментів, а також стабільних версій мов програмування та веб-серверних модулів;
2. застосуванні сучасних фрейморків та бібліотек зарекомендували себе часом та мають постійну підтримку з боку розробників. Це допоможе уникнути проблем з підтримкою системи та її розширенням;
3. застосування документ-орієнтованої бази даних або NoSQL;
4. використання хостингу з характеристиками, які є необхідними для надійного функціонування додатку.

2.4 Створення дизайну

Цей підрозділ присвячений обговоренню загального вигляду системи. Зокрема, викладається важливість мінімалістичного дизайну, що зосереджений на завданнях, і розглядаються основні елементи інтерфейсу, такі як головна панель, сайдбар для навігації, модуль завдань та верхнє меню. Обговорюється використання колірного коду для представлення статусу завдань і різних приоритетів. Розглянемо основні компоненти та елементи дизайну

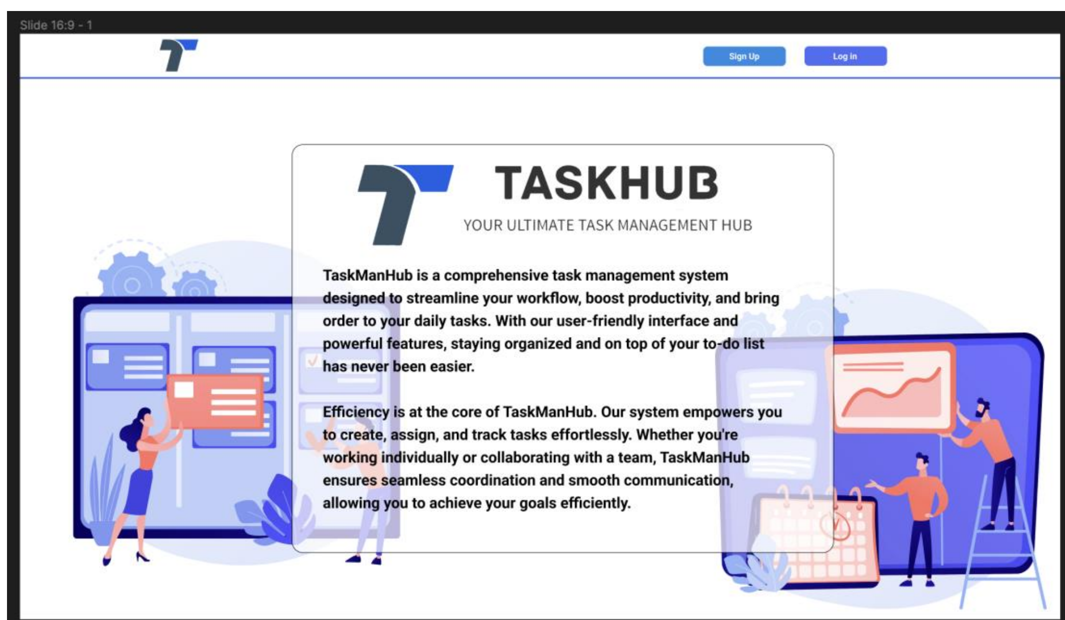

Рис. 2.2. Головна сторінка сайту.

Рис. 2.3. Сторінка авторизації.

Рис. 2.4. Основні елементи дизайну.

2.5. Детальне проектування

Зважаючи на вимоги технічного завдання, було обрано MongoDB як систему керування базами даних (СКБД). MongoDB є нереляційною (документною) базою даних з відкритим кодом. Вона пропонує багато переваг і відмінностей порівняно з традиційними реляційними базами даних, такими як MySQL.

MongoDB зберігає дані у вигляді гнучких документів, які можуть містити будь-яку структуру та відображати реальні об'єкти або сутності. Вона не вимагає фіксованої схеми бази даних, що дозволяє зручно працювати з змінюючимися або неструктурованими даними. Також, MongoDB підтримує вбудовані документи і масиви, що спрощує представлення та роботу зі складними структурами даних.

Однією з ключових особливостей MongoDB є горизонтальне масштабування. Вона може розподіляти навантаження на кластері серверів, що дозволяє обробляти великі обсяги даних та забезпечувати високу доступність. Також, MongoDB підтримує гнучкість при додаванні нових серверів до кластера або зміні його розмірів, що спрощує масштабування системи зростанням обсягу даних.

MongoDB також пропонує багатий набір функцій для роботи з даними, включаючи запити, індексацію, агрегацію, реплікацію та шарування даних. Вона підтримує мову запитів MongoDB Query Language (MQL), яка є простою у використанні і надає можливості для пошуку, фільтрації та аналізу даних.

MongoDB є популярною у веб-розробці, особливо в сфері розробки сучасних додатків, які потребують гнучкості та швидкодії. Вона інтегрується з багатьма популярними мовами програмування, такими як JavaScript, Python, Ruby та іншими, що дозволяє зручно розробляти програми з використанням MongoDB.

Загалом, MongoDB є потужною та гнучкою СКБД, яка надає широкі можливості для зберігання, керування та опрацювання даних. Вона відповідає вимогам сучасних додатків і може бути ефективним вибором для проектів, що вимагають гнучкості схеми даних, масштабованості та продуктивності [15].

Інструменти управління MongoDB

MongoDB надає кілька інструментів для управління базами даних та взаємодії з ними. Ось кілька ключових інструментів, які можна використовувати з MongoDB:

– **MongoDB Shell**: Це інтерактивна командна оболонка, яка надає можливість виконувати команди та запити до бази даних MongoDB. Вона дозволяє виконувати різноманітні операції, такі як створення, зміна та видалення документів, налаштування індексів, агрегація даних та багато іншого. MongoDB Shell використовує мову запитів MQL (MongoDB Query Language) для взаємодії з базою даних.

– **MongoDB Compass**: Це графічний інтерфейс користувача (GUI), який надає візуальну зручність для управління та взаємодії з базою даних MongoDB. MongoDB Compass дозволяє переглядати та редагувати дані, створювати та виконувати запити, налаштовувати індекси, візуалізувати структуру даних та виконувати аналітику. Він є потужним інструментом для розробників та адміністраторів баз даних MongoDB.

– **MongoDB Atlas**: Це хмарний сервіс управління базами даних MongoDB, який надає можливість розгортання та керування базами даних в хмарі. MongoDB Atlas автоматизує процеси налаштування, масштабування, резервного копіювання та керування базою даних. Він також надає інструменти для моніторингу та аналізу продуктивності бази даних.

– **MongoDB Ops Manager**: Це інструмент для керування та моніторингу MongoDB, який надає розширені можливості управління розгорнутими інстансами MongoDB. MongoDB Ops Manager дозволяє налаштовувати систему моніторингу, автоматичне масштабування, резервне копіювання та відновлення даних, а також забезпечує зручний інтерфейс для адміністрування та налагодження бази даних.

– **MongoDB BI Connector**: Це компонент, який дозволяє підключати засоби бізнес-аналітики, такі як BI-інструменти, до бази даних MongoDB. Він надає зручний доступ до даних MongoDB для аналізу та створення звітів.

Ці інструменти надають різні способи управління та взаємодії з базами даних MongoDB, забезпечуючи зручність та ефективність в розробці, адмініструванні та аналізі даних [15].

Для початку нам достатньо 5 колекцій для написання повноцінної БД для нашої системи.

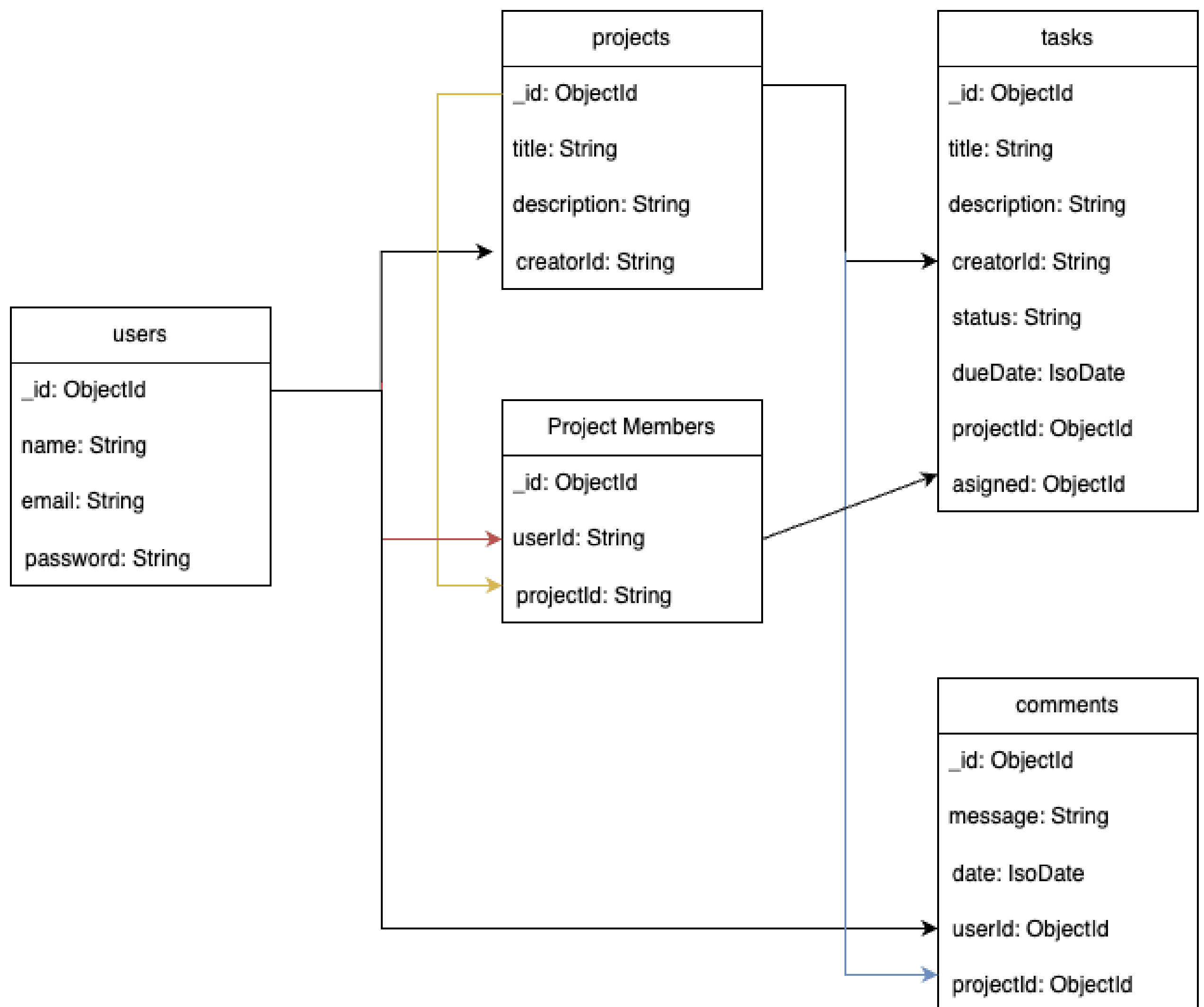

Рис. 2.5. Структура таблиць бази даних.

Висновок до розділу 2

Проектування системи "Task Management" використовує стек технологій MERN (MongoDB, ExpressJs, ReactJs, Node.js), який визнаний як прогресивний та гнучкий для розробки інформаційних систем.

Було вибрано водоспадну модель життєвого циклу програмного забезпечення (ПЗ), що передбачає послідовне виконання стадій розробки. Цей вибір зумовлений чітким визначенням вимог до проекту на початковому етапі та незмінністю їх протягом всього циклу розробки.

Здійснюючи аналіз вимог, було виявлено ключові потреби користувачів, що включають можливість створювати, редагувати, призначати та відслідковувати завдання, а також організовувати спільну роботу над ними.

В процесі проектування та детального проектування було вирішено використовувати MongoDB як базу даних через її високу гнучкість та масштабованість. ExpressJs було обрано як основу для розробки серверної частини заради його простоти та надійності, а ReactJs використовується для розробки клієнтської частини через його гнучкість, продуктивність та популярність в середовищі розробників.

Загалом, процес проектування цієї інформаційної системи було виконано згідно з вибраною водоспадною методологією, враховуючи всі вимоги користувачів.

РОЗДІЛ 3. РОЗРОБКА СИСТЕМИ

3.1. Вибір технологій

Для реалізації бекенду для системи управління задачами (task management) з використанням Express.js та MongoDB, рекомендовано використовувати такі додаткові технології та бібліотеки:

– **Express.js** – це легкий та гнучкий фреймворк для створення веб-додатків на базі Node.js;

– **bcrypt.js**: ця бібліотека дозволяє здійснювати хешування та перевірку паролів у безпечний спосіб. Вона надає простий інтерфейс для генерації солі та хешування паролів, що допомагає забезпечити безпеку користувачів;

– **Mongoose**: Mongoose – це об'єктно-документова бібліотека моделювання для Node.js, яка дозволяє зручно працювати з MongoDB. Вона надає зручний спосіб визначення схем даних, створення моделей та взаємодії з базою даних;

– **body-parser**: Ця middleware-бібліотека для Express.js дозволяє легко отримувати дані з тіла HTTP-запиту, такі як параметри форми або JSON-дані. Вона спрощує обробку даних, що передаються серверу;

– **Joi**: Joi – це бібліотека валідації даних для JavaScript, яка дозволяє валідувати та перевіряти правильність вхідних даних. Використовуючи Joi разом з Express.js, ви можете забезпечити правильну валідацію та обробку даних, які передаються до сервера;

– **JSON Web Tokens (JWT)**: JWT є стандартом для створення токенів аутентифікації та авторизації. Використовуючи JWT, ви можете створювати токени, які містять інформацію про користувача та його права доступу, що дозволяє здійснювати безпечну автентифікацію та авторизацію в вашій системі;

– **CORS**: CORS (Cross-Origin Resource Sharing) – це механізм, який дозволяє контролювати доступ до ресурсів на сервері з іншого джерела (домену). Використовуючи middleware CORS для Express.js, ви можете налаштувати політику доступу до вашого API з інших джерел, що допомагає уникнути проблем з безпекою та доступом до даних;

– **Socket.IO**: Socket.IO – це бібліотека для реалізації веб-сокетів (WebSocket) в Node.js. Використовуючи Socket.IO, ви можете реалізувати багатокористувацькі, реально-часні функціональності, такі як сповіщення про зміни стану задач, спілкування в реальному часі між користувачами тощо.

Ці технології та бібліотеки доповнюють стек Express.js та MongoDB, дозволяючи забезпечити безпеку, зручну роботу з базою даних, обробку даних з запиту та валідацію вхідних даних. Використання цих інструментів сприятиме ефективній та безпечній розробці системи управління задачами.

Для реалізації клієнтської частини системи управління задачами, було обрано наступні технології:

– **ReactJS**: ReactJS є популярною бібліотекою JavaScript для побудови користувацьких інтерфейсів. Вона забезпечує ефективний рендеринг та керування станом компонентів. ReactJS дозволяє створювати компоненти, які можна повторно використовувати, що спрощує розробку і підтримку клієнтської частини додатку;

– **React Query**: React Query – це бібліотека для керування запитами та кешування даних в клієнтському додатку. Вона спрощує взаємодію з сервером шляхом автоматичного керування статусами запитів, кешування даних та оновлення інтерфейсу при отриманні нових даних;

– **Mantine**: Mantine – це сучасна бібліотека компонентів інтерфейсу користувача, побудована на основі ReactJS. Вона надає готові компоненти, які можна використовувати для створення зручного і привабливого інтерфейсу користувача в системі управління задачами;

– **Redux Toolkit**: Redux Toolkit – це офіційний набір інструментів для роботи з Redux, який спрощує керування станом додатку. Він надає зручні

функції для організації стану, редукторів та дій. Redux Toolkit допомагає знизити кількість необхідного коду та покращує продуктивність розробки;

– **React Router**: React Router – це бібліотека для навігації в односторінкових додатках на базі ReactJS. Вона дозволяє визначати шляхи та маршрутизувати користувача до відповідних компонентів в залежності від URL. React Router допомагає створити чітку та зручну навігацію в системі управління задачами;

– **Axios**: Axios – це бібліотека для виконання HTTP-запитів з клієнтської частини додатка. Вона дозволяє зручно взаємодіяти з сервером, отримувати та відправляти дані. Axios має простий та зрозумілий API для роботи з HTTP-запитами;

– **Styled Components**: Styled Components – це бібліотека для стилізації компонентів з використанням CSS-in-JS підходу. Вона дозволяє описувати стилі компонентів безпосередньо в коді, що полегшує підтримку та повторне використання стилів. За допомогою Styled Components можна створити стильний та привабливий інтерфейс користувача;

– **React Toastify**: React Toastify – це бібліотека для відображення повідомлень (тостів) в React-додатках. Вона дозволяє створювати сповіщення та повідомлення, які відображаються короткий час зверху або знизу екрану. За допомогою React Toastify можна легко і зручно повідомляти користувачів про різні події та стан системи;

– **React Hook Form**: React Hook Form – це бібліотека для роботи з формами в React-додатках. Вона надає простий та потужний спосіб валідації та керування станом форм. За допомогою React Hook Form можна легко створювати, валідувати та відправляти дані з форм на сервер;

– **Yup**: Yup – це бібліотека для валідації даних в React-додатках. Вона надає простий та потужний спосіб визначення правил валідації для форм і об'єктів даних. Yup дозволяє перевіряти різні типи даних, включаючи рядки, числа, дати, масиви тощо, та надає зручні методи для визначення обов'язковості, мінімальних та максимальних значень, формату тощо.

Використання Yup спрощує процес валідації та допомагає забезпечити правильність введених користувачем даних.

3.2. Розробка серверної частини

Маршрутизація:

Маршрутизація на бекенді відноситься до процесу визначення шляхів (URL-адрес) та способу обробки запитів веб-додатком. У бекенді, зазвичай за допомогою фреймворків чи бібліотек, таких як Express.js, Node.js, Django, Ruby on Rails і т.д., маршрутизація використовується для встановлення зв'язку між вхідним URL та обробкою, яку необхідно виконати для цього URL.

Маршрутизація допомагає веб-додатку розподілити різні запити на відповідні функції або контролери, які будуть обробляти ці запити і повертати результат. Кожний маршрут визначається за допомогою URL-шаблону та HTTP-методу, який цей маршрут очікує (GET, POST, PUT, DELETE тощо).

Наприклад, в маршрутизації можуть бути визначені такі шляхи:

– **GET /users** – отримання списку користувачів;

– **POST /users** – створення нового користувача;

– **GET /users/:id** – отримання конкретного користувача за його ідентифікатором;

– **PUT /users/:id** – оновлення інформації про користувача;

– **DELETE /users/:id** – видалення користувача за його ідентифікатором.

Коли клієнт (наприклад, веб-браузер) надсилає HTTP-запит до сервера, сервер використовує маршрутизацію, щоб знайти відповідний маршрут для цього запиту. Коли знайдено відповідний маршрут, відповідний обробник (контролер або функція) запускається для обробки запиту. Результат потім повертається клієнту у відповідь на запит.

Маршрутизація дозволяє організовувати та структурувати вашу програму на бекенді, розділяти логіку обробки запитів на більш маневренні шматки та спрощувати розробку та підтримку веб-додатків.

Наведемо приклад маршрутизації для серверної частини Express.js із MongoDB для системи керування завданнями:

```
1  const express = require('express');
2  const router = express.Router();
3
4  // Імпорт необхідних контролерів та проміжного ПЗ (middleware)
5  const authController = require('./controllers/authController');
6  const taskController = require('./controllers/taskController');
7  const authMiddleware = require('./middlewares/authMiddleware');
8
9  // Маршрутизація для автентифікації
10 router.post('/register', authController.register);
11 router.post('/login', authController.login);
12
13 // Захищені маршрути (вимагають аутентифікації)
14 router.get('/tasks', authMiddleware.authenticate, taskController.getAllTasks);
15 router.get('/tasks/:id', authMiddleware.authenticate, taskController.getTaskById);
16 router.post('/tasks', authMiddleware.authenticate, taskController.createTask);
17 router.put('/tasks/:id', authMiddleware.authenticate, taskController.updateTask);
18 router.delete('/tasks/:id', authMiddleware.authenticate, taskController.deleteTask);
19
20 module.exports = router;ы
```

Рис. 3.1. Приклад маршрутизації сервера на Express.js.

У цьому прикладі ми визначаємо маршрутизатор за допомогою **express.Router()** та імпортуємо необхідні контролери та проміжне програмне забезпечення.

Маршрути автентифікації (**/register** і **/login**) обробляються методами authController register і login відповідно.

Захищені маршрути (**/tasks**, **/tasks/:id**, **/tasks**, **/tasks/:id**) вимагають автентифікації та обробляються методами **taskController getAllTasks**, **getTaskById**, **createTask**, **updateTask** і **deleteTask**. Проміжне програмне забезпечення **authMiddleware.authenticate** використовується для забезпечення автентифікації для цих маршрутів.

Контролери:

У Express.js контролер – це модуль або клас, який обробляє логіку та функціональні можливості для певного маршруту або групи маршрутів. Він

відповідає за обробку вхідних запитів, взаємодіє з базою даних або іншими джерелами даних і надсилає відповідні відповіді.

Основна мета контролера – відокремити обробку HTTP-запитів і бізнес-логіку від рівня маршрутизації та проміжного програмного забезпечення. Це допомагає зберегти вашу кодову базу організованою, модульною та легшою для обслуговування.

Визначення: контролер зазвичай є модулем або класом, який експортує методи, що відповідають різним HTTP-дієсловам (GET, POST, PUT, DELETE) або іншим діям, які можна виконати на певному ресурсі чи маршруті.

Запит і відповідь: кожен метод у контролері приймає два параметри: об'єкти **req** (запит) і **res** (відповідь). Об'єкт req містить інформацію про вхідний запит, таку як заголовки запиту, параметри запиту, параметри запиту та тіло запиту. Об'єкт res використовується для надсилання відповіді клієнту.

Обробка відповіді: після виконання необхідних операцій контролер створює відповідну відповідь для повернення клієнту. Це може включати встановлення коду статусу відповіді, додавання заголовків, форматування тіла відповіді та надсилання відповіді за допомогою методів об'єкта res, таких як res.json(), res.send() або res.status().

Поділ проблем: використання контролерів допомагає розділити проблеми маршрутизації, проміжного програмного забезпечення та бізнес-логіки. Маршрути відповідають за зіставлення URL-адрес з методами контролера, проміжне програмне забезпечення може обробляти додаткову обробку або перевірку перед досягненням контролера, а контролер зосереджується на обробці конкретної логіки, пов'язаної з певним маршрутом або ресурсом.

Багаторазове використання: контролери розроблені для багаторазового використання. Їх можна використовувати в кількох маршрутах або навіть у різних проектах, просто імпортувавши модуль або клас контролера та викликаючи його методи. Це сприяє повторному використанню коду та зменшує дублювання коду.

Бізнес-логіка: у методах контролера можна реалізувати необхідну бізнес-логіку для обробки запиту. Це може включати перевірку вхідних даних, взаємодію з базою даних або іншими зовнішніми службами, виконання обчислень або перетворень даних і підготовку відповіді.

Розглянемо контролер на прикладі авторизації користувача.

```
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const User = require('../models/user');

class AuthController {
    static async register(req, res) {
        try {
            const { name, email, password } = req.body;
            // Перевірка, чи користувач з вказаною електронною поштою вже існує
            const existingUser = await User.findOne({ email });
            if (existingUser) {
                return res.status(400).json({ error: 'User with this email already exists' });
            }
            // Хешування пароля
            const hashedPassword = await bcrypt.hash(password, 10);
            // Створення нового користувача
            const newUser = new User({ name, email, password: hashedPassword });
            await newUser.save();
            // Генерація токена доступу
            const token = jwt.sign({ userId: newUser._id }, process.env.JWT_SECRET, { expiresIn: '1h' });
            res.json({ token });
        } catch (error) {
            res.status(500).json({ error: 'Server error' });
        }
    }

    static async login(req, res) {
        try {
            const { email, password } = req.body;
            // Перевірка, чи існує користувач з вказаною електронною поштою
            const user = await User.findOne({ email });
            if (!user) {
                return res.status(401).json({ error: 'Invalid email or password' });
            }
            // Порівняння паролів
            const isPasswordValid = await bcrypt.compare(password, user.password);
            if (!isPasswordValid) {
                return res.status(401).json({ error: 'Invalid email or password' });
            }
            // Генерація токена доступу
            const token = jwt.sign({ userId: user._id }, process.env.JWT_SECRET, { expiresIn: '1h' });

            res.json({ token });
        } catch (error) {
            res.status(500).json({ error: 'Server error' });
        }
    }
```

Рис. 3.2. Клас AuthController з методами авторизації і реєстрації.

У цьому прикладі клас **AuthController** містить статичні методи для реєстрації користувача **register**, входу користувача **login** і отримання даних користувача **getUserData**. В цьому прикладі використовуються бібліотеки bcrypt для хешування паролів і бібліотеку **jsonwebtoken** для створення токенів доступу. Модель користувача представляє дані користувача в базі даних **MongoDB**.

Це є лише спрощеним прикладом, який було модифіковано під час написання програмного забезпечення, цей приклад демонструє лише структуру яку було використано під час написання api для користувачів

Middleware:

У Node.js і Express.js функції проміжного програмного забезпечення (Middleware) відіграють вирішальну роль у обробці вхідних HTTP-запитів і забезпечують додаткову обробку та функціональність перед досягненням остаточного обробника маршруту. Функції проміжного програмного забезпечення — це по суті функції, які мають доступ до об'єктів запиту (req) і відповіді (res), а також до наступної функції в циклі запит-відповідь програми.

Розглянемо чому саме необхідно використовувати middleware:

– **обробка запитів**: функції проміжного програмного забезпечення можуть виконувати різні операції над вхідним запитом, такі як журналювання, автентифікація, перевірка, розбір тіл запиту, обробка файлів cookie, маніпулювання заголовками тощо. Це дозволяє додавати функціональні можливості до вашої програми, не змінюючи кожен окремий обробник маршруту;

– **обробка помилок**: функції проміжного програмного забезпечення можуть обробляти помилки, визначивши функцію проміжного програмного забезпечення обробки помилок із чотирма параметрами (err, req, res, next). Це проміжне програмне забезпечення обробки помилок викликається щоразу, коли виникає помилка, або передається до next() [16].

Розглянемо приклад middleware.

```javascript
class AuthMiddleware {
  static authenticate(req, res, next) {
    // Виконати логіку автентифікації тут
    // Наприклад, перевірити, чи користувач має дійсний токен або сесію
    if (req.isAuthenticated()) {
      // Користувач автентифікований, продовжити до наступного middleware або обробника маршруту
      return next();
    }
    // Користувач не автентифікований, повернути відповідь з помилкою авторизації
    return res.status(401).json({ message: 'Неавторизований доступ' });
  }

  static authorize(role) {
    return (req, res, next) => {
      // Виконати логіку авторизації тут
      // Наприклад, перевірити, чи користувач має відповідну роль
      if (req.user.role === role) {
        // Користувач має необхідну роль, продовжити до наступного middleware або обробника маршруту
        return next();
      }
      // Користувач не має необхідної ролі, повернути відповідь з помилкою заборони доступу
      return res.status(403).json({ message: 'Заборонено доступ' });
    };
  }
}

module.exports = AuthMiddleware;
```

Рис. 3.3. Реалізація проміжного програмного забезпечення на прикладі класу authMidlleware.

У цьому прикладі клас AuthMiddleware визначає два статичних методи: authenticate і authorize. Ці методи можна використовувати як middleware у вашому маршруті для виконання перевірок автентифікації та авторизації.

Метод authenticate перевіряє, чи користувач автентифікований за допомогою функції **req.isAuthenticated()**. Якщо користувач автентифікований, він викликає функцію **next()** для продовження до наступного middleware або обробника маршруту. Якщо користувач не автентифікований, він повертає відповідь з помилкою авторизації.

Метод **authorize** перевіряє, чи користувач має певну роль. Він отримує роль як аргумент і повертає **middleware**, яке виконує перевірку ролі користувача. У тілі middleware виконується логіка перевірки ролі користувача на основі **req.user.role** (припускається, що дані користувача зберігаються у полі **user** об'єкта **req**). Якщо користувач має відповідну роль, middleware викликає функцію **next()** для продовження до наступного middleware або обробника маршруту. Якщо користувач не має необхідної ролі, middleware повертає відповідь з помилкою заборони доступу. Це є необхідною перевіркою для того

щоб користувачі мали змогу робити робочі простори для проекту, або інших цілей.

3.3. Розробка клієнтської частини

Вибір архітектури

Для реалізації клієнтської частини була обрана архітектура featured-based. Архітектура feature-based (або функціональна архітектура) є одним з підходів до організації структури проекту. В цій архітектурі програмне забезпечення розбивається на набір функціональних фіч (або функцій), і кожна фіча має свої власні модулі, компоненти та ресурси.

Основна ідея архітектури featured-based полягає у виокремленні функціональності в окремі модулі (фічі) замість того, щоб мати шари або компоненти, які відповідають за різні аспекти додатку (наприклад, шари MVC). Кожна фіча включає всі необхідні елементи (моделі, контролери, представлення тощо), необхідні для реалізації цієї функціональності. Кожна фіча може мати свої власні шаблони, стилі, маршрутизацію і залежності.

Основні переваги feature-based архітектури:

– **модульність**: функціональність розбивається на незалежні модулі (фічі), що полегшує розробку, тестування і підтримку. Кожна фіча може бути розроблена, тестована і розгорнута окремо, що сприяє розподіленій роботі команди;

– **повторне використання**: функціональні модулі можуть бути повторно використані в інших проектах або функціях. Це забезпечує ефективне використання коду і полегшує масштабування додатку;

– **розширюваність**: додавання нових функцій до додатку здійснюється шляхом додавання нових фіч, що спрощує розширення функціональності без впливу на існуючі функції;

– **командна робота**: розбиття програми на функціональні модулі дозволяє розподіляти роботу між різними командами або розробниками. Кожна

команда може працювати над своїми фічами, не втручаючись в роботу інших команд;

– **тестованість**: функціональні модулі можуть бути легко тестовані незалежно від інших частин додатку. Це спрощує написання автоматичних тестів і забезпечує високу якість програмного забезпечення [17].

Ця архітектура зазвичай є добрим вибором для середньо- і великомасштабних проектів, які мають розширену функціональність і команду розробників. Вона полегшує організацію коду, підтримку та розширення додатку, забезпечує високу модульність та масштабованість.

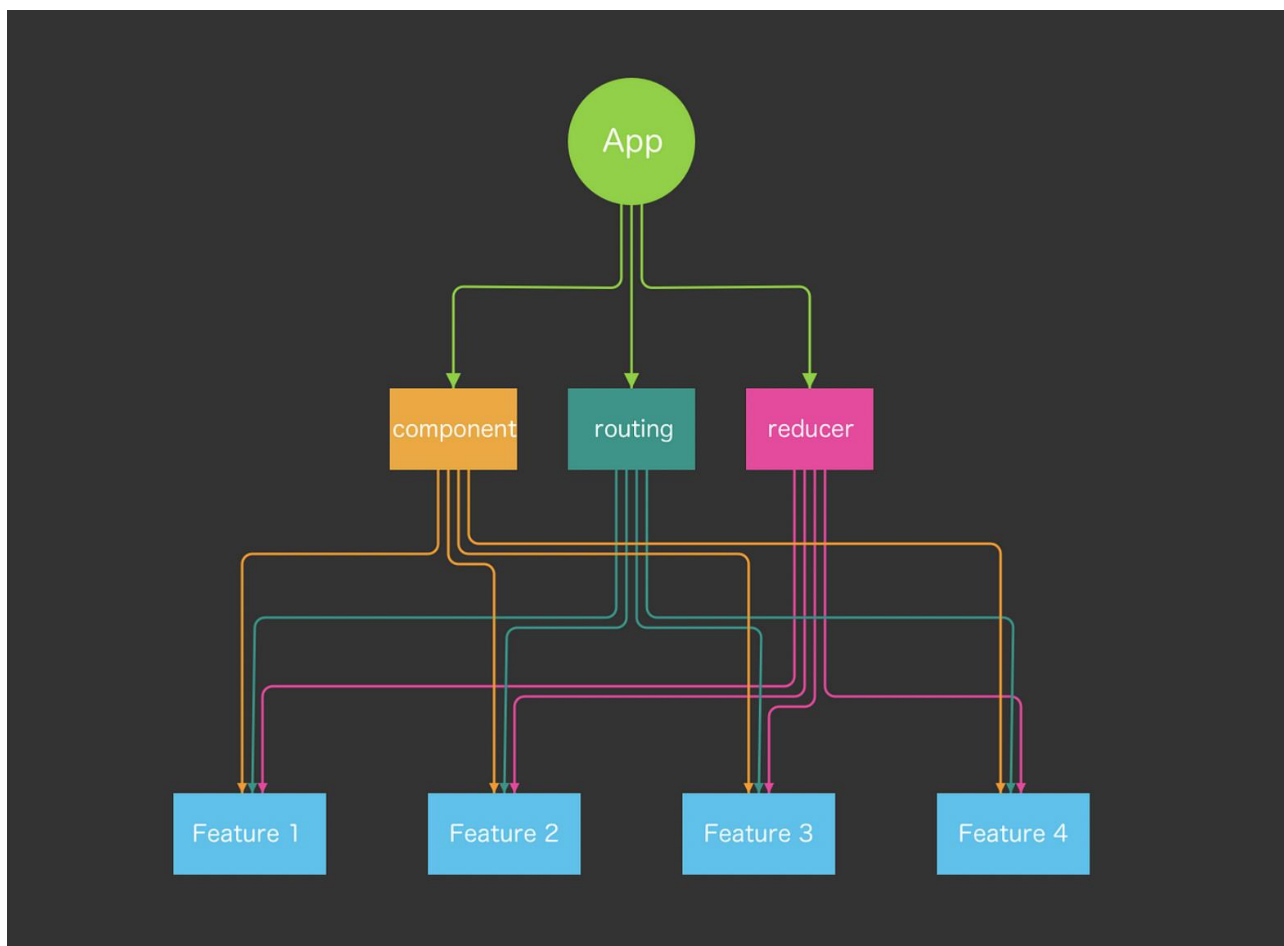

Рис. 3.4. Діаграма архітектури feature-based.

Реалізація HTTP-запитів до сервера

Для реалізації HTTP-запитів було обрано бібліотеку axios. Розглянемо приклад класу авторизації. Вибір бібліотеки Axios для реалізації запитів на сервер в JavaScript/Node.js має декілька обґрунтувань:

– простота використання: Axios має дуже простий та легкий у використанні API. Він надає методи для роботи з HTTP-запитами, такими як GET, POST, PUT, DELETE, із зручним синтаксисом, що дозволяє легко виконувати запити та отримувати відповіді;

– універсальність: Axios може використовуватись як у браузері, так і в середовищі Node.js. Це дозволяє використовувати одну бібліотеку для здійснення запитів на сервер незалежно від того, чи ви працюєте з клієнтською або серверною стороною;

– підтримка властивостей промісів: Axios повертає Promise для кожного запиту, що дозволяє використовувати сучасний підхід до обробки асинхронних операцій за допомогою async/await або методів then/catch. Це спрощує управління асинхронним кодом і обробку результатів запитів [18].

```
import { AxiosInstance } from "axios";
import { ApiResponse } from "./types/api";
import { LoginForm, LoginResponse } from "./types/Login";
import { RegisterForm, RegisterResponse } from "./types/Register";

export class Base {
  request: AxiosInstance;
  baseUrl: string;

  constructor(request: AxiosInstance, baseUrl: string) {
    this.baseUrl = baseUrl;
    this.request = request;
  }
}

export default class Auth extends Base {
  async login(payload: LoginForm) {
    try {
      const response = await this.request.post<ApiResponse<LoginResponse>>(
        `${this.baseUrl}/register`,
        payload
      );
      return response.data;
    } catch (error: any) {
      const parsedError = error.response.message
      console.error(parsedError);
      throw new Error(parsedError);
    }
  }

  async register(payload: RegisterForm) {
    try {
      const response = await this.request.post<ApiResponse<RegisterResponse>>(
        `${this.baseUrl}/login`,
        payload
      );
      return response.data.data;
    } catch (error: any) {
      const parsedError = error.response.message
      console.error(parsedError);
      throw new Error(parsedError);
    }
  }
}
```

Рис. 3.5. Приклад класу API Auth.

```
import axios from "axios";
import Auth from "./Auth";
import TokenStorage from "../helpers/tokenStorage";

export default class Api {
  baseUrl = "/api";
  auth: Auth;

  constructor() {
    this.auth = new Auth(this.request, this.baseUrl);
  }

  // інстанс запитів через який виконаються всі запити на бекенд
  get request() {
    const instance = axios.create();

    instance.interceptors.request.use((config) => {
      const token = TokenStorage.getToken();

      if (token) {
        (config.headers as any).common["Authorization"] = `Bearer ${token}`;
      }

      return config;
    });

    return instance;
  }
}
```

Рис. 3.6. Приклад класу API.

У цьому прикладі використовується бібліотека продемонстровано здійснення HTTP-запитів за допомогою Axios. Ми виконуємо різні типи запитів (GET, POST, PUT, DELETE) до сервера з використанням методів axios.get(), axios.post(), axios.put() і axios.delete(). У кожному випадку, ми передаємо URL запиту та опціонально дані, які треба надіслати на сервер.

Після виконання запиту, ми використовуємо методи .then() та .catch() для обробки успішної відповіді або помилки відповідно. В методі .then() ми можемо отримати доступ до даних відповіді за допомогою response.data, і в методі .catch() ми можемо обробити помилку та вивести її повідомлення.

Реалізація JSX компонентів

Для написання JSX компонентів було використано мову програмування Typescript, React, react-hook-form, mantine (бібліотека інтерфейсу користувача).

```
LoginPage.tsx U ×
src > LoginPage.tsx > Container
1  import React from 'react';
2  import { useForm } from 'react-hook-form';
3  import { Button, TextInput } from '@mantine/core';
4  import styled from 'styled-components';
5  import { useAuth } from './features/Auth/hooks/useAuth';
6  import { LoginForm } from './api/types/Login';
7
8
9  const AuthPage: React.FC = () => {
10   const {handleLogin} = useAuth()
11   const {
12     register,
13     handleSubmit,
14     formState: { errors },
15   } = useForm<LoginForm>();
16
17   return (
18     <Container>
19       <FormContainer>
20         <h2>Login</h2>
21         <Form onSubmit={handleSubmit(handleLogin)}>
22           <InputContainer>
23             <TextInput
24               label="Email"
25               type="email"
26               {...register('email', { required: true })}
27               error={errors.email && 'Required field'}
28             />
29           </InputContainer>
30
31           <InputContainer>
32             <TextInput
33               label="Password"
34               type="password"
35               {...register('password', { required: true })}
36               error={errors.password && 'Required field'}
37             />
38           </InputContainer>
39
40           <Button type="submit">Login</Button>
41         </Form>
42       </FormContainer>
43     </Container>
44   );
45 };
```

```
LoginPage.tsx U ×
src > LoginPage.tsx > Container
47 //styles
48 const Container = styled.div`
49   display: flex;
50   justify-content: center;
51   align-items: center;
52   height: 100vh;
53 `;
54
55 const FormContainer = styled.div`
56   width: 300px;
57   padding: 20px;
58   background-color: #f6f6f6;
59   border-radius: 4px;
60   box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
61 `;
62
63 const InputContainer = styled.div`
64   margin-bottom: 16px;
65 `;
66
67 const Form = styled.form`
68 `
69
70 export default AuthPage;
71
```

Рис. 3.7. Приклад реалізації компоненту LoginPage (сторінка авторизації).

У цьому файлі ми маємо компонент AuthPage, який представляє сторінку авторизації. Основні дії, які відбуваються на цій сторінці, включають використання хука useForm з пакету react-hook-form для обробки форми і валідації полів, виклик функції handleLogin з хука useAuth для авторизації користувача і рендеринг відповідних компонентів з mantine для відображення полів вводу та кнопки.

В компоненті **AuthPage** ми спочатку імпортуємо необхідні залежності, включаючи **React**, **useForm**, **Button** та **TextInput** з **mantine**, а також **styled-components** для стилізації компонентів. Ми також імпортуємо хук useAuth з файлу **useAuth**, який відповідає за авторизацію користувача.

Далі, ми використовуємо хук **useForm** з пакету **react-hook-form**, щоб отримати необхідні методи та стан для обробки форми. Ми деструктуруємо метод **handleSubmit** для обробки подання форми та стан errors, який містить помилки валідації.

У рендері ми використовуємо компоненти **Container**, **FormContainer**, **InputContainer** та **Form** для стилізації розміщення та вигляду форми. Усередині **Form** ми рендеримо компоненти **TextInput** для полів вводу **Email** та **Password**. Ми передаємо властивості **label**, **type**, **register** для підключення до форми з валідацією, а також error, якщо поле має помилку валідації.

Для кнопки **Login** ми використовуємо компонент **Button** з mantine, який має тип submit, щоб спрацювала функція **handleSubmit** при натисканні кнопки.

На останок, ми експортуємо компонент **AuthPage** за допомогою **export default AuthPage**, щоб його можна було використовувати в інших частинах програми.

Хуки (hooks) – це новий механізм, що був представлений у React 16.8. Вони дозволяють використовувати стан і інші функціональності React в компонентах на основі функцій (функціональних компонентах), а не на основі класів (класових компонентах). Хуки роблять код більш зрозумілим, повторно використовуваним і простим у тестуванні.

Розглянемо реалізацію хуків на прикладі useAuth та useWhoAmI хуків.

```
useAuth.ts M ×
src > features > Auth > hooks > useAuth.ts > useAuth > handleLogin > useCallback() callback
1  import { useCallback } from "react";
2  import { LoginForm } from "../../../api/types/Login";
3  import { api } from "../../..";
4  import { useQuery } from "react-query";
5  import { useNavigate } from "react-router-dom";
6  import { toast } from 'react-toastify';
7
8  export const useWhoAmI = () => {
9    const { data, refetch, isLoading, isError, remove } = useQuery(["ACCOUNT"], () => api.auth.whoAmI(), {retry: 0});
10
11   return {
12     data,
13     refetch,
14     isLoading,
15     isError,
16     remove,
17   };
18 };
19
20
21 export const useAuth = () => {
22   const {refetch} = useWhoAmI()
23   const navigate = useNavigate()
24
25   const handleLogin = useCallback(async(payload: LoginForm) => {
26     try {
27       await api.auth.login(payload)
28       await refetch()
29       navigate('/dashboard')
30     } catch (error:any) {
31       const parsedError = error.response.data
32       toast.error(parsedError);
33     }
34   },[refetch, navigate])
35
36   return {
37     handleLogin
38   }
39 };
40
```

Рис. 3.8. Реалізація хука для авторизації та отримання даних користувача.

Цей файл містить два хуки – **useWhoAmI** і **useAuth**, які використовуються для отримання і авторизації користувача.

useWhoAmI хук використовує бібліотеку react-query для виконання запиту до сервера, щоб отримати дані про користувача. Він повертає об'єкт з наступними властивостями:

– **data**: дані про користувача, які отримані з сервера;

– **refetch**: функція для повторного виконання запиту і оновлення даних;

– **isLoading**: прапор, який вказує, чи триває процес завантаження даних;

– **isError**: прапор, який вказує, чи сталась помилка при завантаженні даних;

– **remove**: функція для видалення кешованих даних;

– **useAuth**: Цей хук використовує **useWhoAmI** для отримання даних користувача та **useNavigate** з **react-router-dom** для навігації на іншу сторінку після успішної авторизації. Він також використовує **useCallback** для створення мемоізованої функції **handleLogin**, яка виконує авторизацію користувача. У разі виникнення помилки при авторизації, використовується **toast.error** з бібліотеки **react-toastify** для відображення повідомлення про помилку.

Функція **handleLogin** асинхронно викликає **api.auth.login** для виконання запиту на сервер для авторизації з переданими даними в payload. Після успішної авторизації, вона викликає **refetch** для оновлення даних про користувача, а потім викликає **navigate('/dashboard')** для переходу на сторінку "dashboard". У разі помилки при авторизації можна обробити відповідні помилки в блоку catch.

Ці хуки можуть використовуються при авторизації та отримані даних про користувача.

3.4. Тестування

Для тестування системи було використано мануальний метод тестування.

Мануальне тестування – це процес перевірки функціональності, коректності та взаємодії програмного забезпечення шляхом ручного виконання тестових сценаріїв та перевірки очікуваних результатів. Основний принцип мануального тестування полягає в тому, що тести виконуються вручну людиною, яка відтворює реальне використання програми та перевіряє його роботу.

Тестування Backend

Тестування Backend включає перевірку функціональності, безпеки та продуктивності серверної частини системи.

Перевірка правильності обробки запитів та відповідей сервером. Було відправлено тестові запити до API та перевірено, чи повертаються очікувані дані та статуси відповідей.

Перевірка валідації та обробки даних. Було перевірено, як система обробляє невалідні або некоректні дані, чи повертає відповідні помилки та повідомлення.

Перевірка автентифікації та авторизації. Механізми автентифікації та авторизації працюють належним чином, перевіряючи доступ до захищених ресурсів та перевіряючи рівні дозволів користувачів.

Тестування взаємодії з базою даних. Було перевірено, чи відбувається збереження, зміна та видалення даних у базі даних безпомилково та ефективно.

Тестування Frontend

Тестування Frontend зосереджується на перевірці коректності роботи інтерфейсу користувача, взаємодії з елементами UI та загальної функціональності клієнтської частини системи. Деякі аспекти, які можуть бути включені до розділу "Тестування Frontend", включають:

Перевірка коректності рендерингу та відображення інтерфейсу. Перевірено, чи правильно рендеряться компоненти, чи відображаються дані згідно з очікуваннями.

Перевірка взаємодії з користувачем. Елементи інтерфейсу реагують на дії користувача та виконують відповідні дії, такі як натискання кнопок, введення даних в поля тощо.

Перевірка переходів між сторінками та маршрутизації. Було перевірено, що перехід між сторінками відбувається коректно, маршрутизація працює належним чином та потрібні дані відображаються.

Тестування форм та валідації. Перевірено, чи відбувається правильна валідація введених даних в формах, чи відображаються відповідні помилки, а також чи коректно відбувається відправка та обробка форм.

Висновок до розділу 3

В цьому розділі була розглянута розробка клієнтської частини системи управління задачами з використанням React, TypeScript та різних корисних бібліотек, таких як Mantine, styled-components, react-hook-form та react-query.

Ми розпочали зі створення проекту та його налаштування, встановили необхідні залежності і структурували код. Потім ми перейшли до розробки різних компонентів, таких як сторінка авторизації, форма входу, реєстрації, тощо. Ми використовували Mantine для стилізації компонентів та react-hook-form для управління формами.

Для взаємодії з сервером ми використовували axios для здійснення HTTP-запитів та react-query для кешування та керування станом даних на клієнті. Ми також використовували власні хуки, такі як useAuth та useWhoAmI, для керування авторизацією та отримання даних користувача.

Результатом цього розділу є клієнтська частина системи управління задачами, яка включає сторінку авторизації, форму входу, реєстрації та використання різних бібліотек для полегшення розробки і покращення користувацького досвіду. Приклади в цьому розділі були досить простими їх завдання полягало в тому, щоб пояснити реалізацію компонентів клієнтської частини системи Task management, реальні розміри проекту набагато більші та складніші. В цьому розділі не були продемонстровані компоненти (створення проекту, створення, редагування та видалення завдань, коментарі до завдань тощо)

Всі ці елементи разом утворюють потужну та функціональну клієнтську частину системи управління задачами.

ВИСНОВКИ

У рамках дипломної роботи була розроблена система управління задачами, яка надає зручні та ефективні інструменти для організації та відстеження робочих завдань. Система була розроблена з використанням таких технологій, як MongoDb, ExpressJs, Mantine, React, Axios, Styled-components та react-query.

В ході роботи було проведено аналіз вимог до системи та розроблено архітектуру, що відповідає потребам користувачів. Було враховано функціональні та нефункціональні вимоги до системи, а також забезпечено зручний та інтуїтивно зрозумілий інтерфейс користувача.

Також було проведене мануальне тестування для Frontend-частини системи, де вручну перевірялися функціональність, коректність рендерингу та взаємодії з елементами UI. Це допомогло впевнитися в правильній роботі інтерфейсу користувача та забезпечити відповідність очікуванням користувачів.

У підсумку, розроблена система Task Management є ефективним та зручним інструментом для організації та відстеження робочих завдань. Вона відповідає вимогам користувачів та дозволяє забезпечити ефективне управління задачами. Процес тестування, включаючи модульне та мануальне тестування, був виконаний з високою увагою до деталей, що дозволило забезпечити якість та надійність системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Старченко Г. В. Управління проектами: теорія та практика : навч. посіб. / Чернігів : видавець Брагинець О. В., 2018. 306 с.
2. Сучасні методи управління проектами. URL: https://sgv.in.ua/off-lifaq/25-suchasni-metodi-upravlinnya-proektami (дата звернення: 21.04.2023).
3. Manifesto for Agile Software Development. URL: http://agilemanifesto.org/ (Last accessed: 22.04.2023).
4. Anderson, David J. Carmichael, Andy. Essential Kanban Condensed. Seattle, WA: Lean Kanban University Press. 2016.
5. Schwaber, K., Sutherland, J. The Scrum Guide. 2020. URL: https://www.scrum.org/ (Last accessed: 24.04.2023).
6. Trello. URL: https://trello.com/ (Last accessed: 25.04.2023).
7. ClickUp. URL: https://clickup.com/ (Last accessed: 25.04.2023).
8. Jira. URL: https://www.atlassian.com/software/jira (Last accessed: 25.04.2023).
9. Wrike. URL: https://www.wrike.com/ (Last accessed: 27.04.2023).
10. Todoist. URL: https://todoist.com/ (Last accessed: 27.04.2023).
11. Hive. URL: https://hive.com/ (Last accessed: 27.04.2023).
12. Asana. URL: https://asana.com/ (Last accessed: 27.04.2023).
13. Популярні життєві цикли розробки ПЗ. URL: https://training.qatestlab.com/blog/technical-articles/popular-software-development-life-cycles/ (дата звернення: 30.04.2023).
14. Моделі життєвого циклу, принципи і методології розробки програмного забезпечення (ПЗ). URL: https://evergreens-com-ua.translate.goog/ua/articles/software-development-metodologies.html?_x_tr_sl=uk&_x_tr_tl=ru&_x_tr_hl=ru&_x_tr_pto=sc (дата звернення: 30.04.2023).
15. MongoDB Documentation. URL: https://www.mongodb.com/docs/ (Last accessed: 02.05.2023).

16. Using middleware. URL: https://expressjs.com/ru/guide/using-middleware.html (Last accessed: 03.05.2023).

17. Feature oriented architecture for web applications. URL: https://medium.com/react-weekly/feature-oriented-architecture-for-web-applications-2b48e358afb0 (Last accessed: 03.05.2023).

18. Axios Documentation. URL: https://axios-http.com/docs/intro (Last accessed: 05.05.2023).