КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій

(кафедра)

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «БАКАЛАВР»

на тему: «Розробка онлайн-системи управління проєктами»

КАЗАРЯН АРТУР СТЕПАНОВИЧ

(прізвище, ім’я та по батькові студента повністю)

Київ 2024 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій

(кафедра)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТ
к.т.н., доцент Гончаренко Т.А.

_____
„___” _____2024 року

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «БАКАЛАВР»

на тему: «Розробка онлайн-системи управління проєктами»

Виконав: студент 4-го курсу, групи КН-20-1

Спеціальності: 122 «Комп`ютерні науки

Освітня програма: «Інформаційні управляючі системи і технології»

(шифр і назва напряму підготовки, спеціальності)

Казарян А.С.
(прізвище та ініціали)

Керівник д.т.н., проф. Бородавка Є. В.
(прізвище та ініціали)

Рецензент к.т.н., доц. Шабала Є.Є.

(прізвище та ініціали)

Київ 2024 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: автоматизації і інформаційних технологій
Кафедра: інформаційних технологій
Освітній рівень: «бакалавр» за ОП
Спеціальність: 122 «Комп`ютерні науки»
Освітня програма: Інформаційні управляючі системи і технології

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТ
к.т.н., доцент Гончаренко Т.А.

_____
„___” _____2024 року

З А В Д А Н Н Я
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «БАКАЛАВР»

Казарян Артур Степанович
Тема роботи: Розробка онлайн-системи управління проєктами
затверджена наказом ректора КНУБА № 2650/2 від 18.11.2023
2. Керівник роботи: Бородавка Євгеній Володимирович, д.т.н, професор кафедри інформаційних технологій проєктування та прикладної математики
3. Строк подання студентом роботи до захисту:
4. Зміст пояснювальної записки за розділами:
Р.1.Аналіз предметної області та постановка задачі
Р.2.Архітектура системи
Р.3.Проєктування бази даних
Р.4.Проєктні рішення та тестовий приклад
5. Інформаційні слайди:

С.1.Ключові аспекти онлайн-систем управління проєктами
С.2.Постановка задачі
С.3.Концептуальна схема системи
С.4.Функціональна блок-схема потоку
С.5.Діаграма сутностей-відносин
С.6.Практична реалізація та результат розробки

6. Календарний план виконання атестаційної випускної роботи

Види робіт та їх зміст	Дата виконання
Р. 1. Аналіз предметної області та постановка задачі	Лютий 2024 р.
Р. 2. Архітектура системи	Березень 2024 р.
Р. 3. Проєктування бази даних	Квітень 2024 р.
Р. 4. Проєктні рішення та тестовий приклад	Травень 2024 р.
Остаточне оформлення роботи	Травень 2024 р.
Направлення роботи на рецензування	Червень 2024 р.
Попередній захист роботи на кафедрі	Червень 2024 р.

7. Консультанти розділів атестаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта, представника комісії	дата	підпис
Прийом програмного продукту	к.т.н., доц. Шабала Є.Є.		

8. Дата видачі завдання: 18.11.2023

Завідувач _____ Гончаренко Т.А.
(підпис) (прізвище та ініціали)

Керівник _____ Бородавка Є.В.
(підпис) (прізвище та ініціали)

Бакалавр _____ Казарян А.С.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Казарян А.С. Розробка онлайн-системи управління проєктами.

Кваліфікаційна випускна робота бакалавра за спеціальністю: 122 «Комп'ютерні науки», спеціалізація: «Інформаційні управляючі системи і технології». – Київський національний університет будівництва та архітектури. – Київ, 2024 .

Робота присвячена створенню онлайн-системи для ефективного керування проєктами. Основна мета системи - надати користувачам простий та інтуїтивно зрозумілий спосіб створення, організації та відстеження завдань і проєктів у вигляді "карток" на дошці. Інструментом реалізації розроблюваної системи виступає фреймворк Next.js від компанії Vercel, створений для розробки високоякісних веб-додатків на основі бібліотеки React, та інші бібліотеки мови програмування JavaScript;

Ключові слова: онлайн-система, класифікація, дошки, проєкти, компоненти, React, Next.js, PostgreSQL, СУБД, UML.

SUMMARY

Kazarian A.S. Development of an online project management system.

Bachelor's thesis in the specialty: 122 "Computer Science", specialization: "Information managing systems and technologies". - Kyiv National University of Construction and Architecture. - Kyiv, 2024.

This thesis is to the creation of an online system for effective project management. The main goal of the system is to provide users with a simple and intuitive way to create, organise and track tasks and projects in the form of "cards" on a whiteboard. The tool for implementing the system under development is the Next.js framework from Vercel, created to develop high-quality web applications based on the React library, and other JavaScript libraries.

Keywords: online system, classification, boards, projects, components, React, Next.js, PostgreSQL, DBMS, UML.

ЗМІСТ

Додаток А. Реалізація дії(action) в системі

Додаток Б. Презентація

Вступ

У сучасному світі, ефективне управління проєктами є ключовим фактором успіху в будь-якій галузі. Незважаючи на це, багато організацій стикаються з викликом нестачі часу та ресурсів для впорядкування та виконання завдань. Ця система вирішує цю проблему, надаючи інтуїтивно зрозумілий інструмент для створення, організації та відстеження завдань та проєктів. Ця онлайн-система дозволяє користувачам створювати дошки для проєктів, створювати завдання, створювати етапи для завдань, та співпрацювати над проєктами в реальному часі. Незалежно від галузі чи розміру команди, ця система надає можливість ефективно керувати проєктами та досягати успіху в умовах сучасного швидкоплинного світу.

Для реалізації головної мети буде використовуватись компонентний підхід та серверний рендеринг. Застосування компонентного підходу дозволить зручно організувати та повторно використовувати частини інтерфейсу, що сприятиме швидкості розробки та підтримці проєкту. Використання серверного рендерингу дозволить забезпечити швидке завантаження контенту для користувачів та покращить індексацію проєкту пошуковими системами.

Розробка онлайн-системи відбуватиметься за допомогою фреймворку Next.js від компанії Vercel.

Перелік умовних позначень

УП – управління проєктами

API – Application Programming Interface

ІС – нформаційна система

БД – база даних

SPA – Single-Page Application (Односторінковий додаток)

SSR – Serder-Side Rendering (Серверний рендеринг)

UML – Unified Modeling Language

MVC – Model-View-Controller

FFDB – Functional Flow Data Diagram

СУБД – Система управління базами даних

SQL – Structured Query Language

ERD – Entity-Relation Diagram

IDE – Integrated Development Environment

ORM – Object-Related Mapping

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної області

Предметною областю цієї дипломної роботи є управління проєктами - це структурований підхід, що охоплює процеси ініціювання, планування, виконання, моніторингу та завершення проєктів. Розробка онлайн-системи для управління проєктами полегшує ці процеси, забезпечуючи централізоване середовище для планування, організації та контролю над проєктами будь-якого масштабу та складності.

У сучасному розумінні поняття “проєкт” тлумачать так:

− це діяльність, захід, що передбачає виконання комплексу певних дій для досягнення певних цілей (одержання певних результатів); близькі за змістом терміни — “господарська діяльність”, “робота (комплекс робіт)”;

− це система організаційно-правових і розрахунково-фінансових документів, необхідних для виконання певних дій або таких, що описують ці дії [1].

Проєкт - це сукупність цілеспрямованих дій, які мають чітко визначені цілі та призначення. Він складається з послідовно виконуваних заходів або робіт, які спрямовані на досягнення конкретного результату. Кожен проєкт є унікальним і має свої специфічні вимоги та завдання, що відрізняються від повсякденної діяльності організації. Важливою особливістю проєкту є його орієнтованість у часі, тобто він має визначені початок і завершення, що дозволяє чітко планувати ресурси та оцінювати досягнення проміжних і кінцевих результатів [1].

Ключовою характеристикою проєкту є його одноразовість та комплексність. Це означає, що проєкт створюється для досягнення певної мети, після чого він завершується, і його результати можуть використовуватися в подальшій діяльності організації. Проєкти зазвичай мають обмежені ресурси, такі як фінанси, час, людські ресурси та матеріали, що вимагає ефективного управління для забезпечення успішного завершення проєкту. Комплексність проєкту передбачає залучення різних підрозділів організації та координацію їх зусиль для досягнення спільної мети.

Проєкти також характеризуються нерегулярністю виконання та повторюваністю дій у різних умовах. Це означає, що хоча проєкти можуть мати схожі етапи або завдання, кожен з них є унікальним через специфіку завдань, ресурсів, учасників і умов виконання. Проєкти можуть варіюватися від невеликих і простих до великих і складних, залежно від масштабу, мети та ресурсів, необхідних для їх виконання. Успішне управління проєктами передбачає ретельне планування, моніторинг і контроль над виконанням завдань, а також адаптацію до змінних умов і вимог, що виникають протягом усього життєвого циклу проєкту.

Мета проєкту – доказовий результат і задані умови реалізації загального завдання проєкту. З точки зору теорії систем управління проєкт як об'єкт управління повинен бути контрольованим і керованим, тобто виділяються певні характеристики, за якими можна постійно контролювати хід виконання проєкту (контрольованість) [2].

Слово проєкт дуже часто вживається у нашому житті. Розробка засобів боротьби зі СНІДом, консервація Чорнобильської АЕС, проведення виборчої кампанії, взяття в оренду та ремонт нового офісу, впровадження нової системи стимулювання персоналу або підготовка до пікніка мають низку спільних ознак, що характеризують їх як проєкти [3]. Це, зокрема, такі ознаки:

- Спрямованість на досягнення мети. Проєкти спрямовуються на досягнення певних результатів — іншими словами, на досягнення мети. Саме 11 ця мета є рушійною силою проєкту, і всі зусилля, що докладаються до його планування та реалізації, спрямовані на її досягнення.
- Проєкти мають численні ієрархічні цілі. Основною метою, наприклад, проєкту, пов'язаного з програмним забезпеченням для комп'ютера, може бути розробка складної системи управління базами даних. Проміжною метою може бути тестування системи в процесі розробки для налаштування програм, а метою нижчого рівня — визначення дат, коли працівники, що розробляють проєкт, звітуватимуть про свої результати на оперативній нараді.

- Координоване виконання пов’язаних між собою дій. Сама сутність проєктів визначає складність їхнього втілення в життя. Проєкти потребують виконання численних завдань, жорстко або гнучко взаємопов’язаних: деякі проміжні завдання не можуть реалізовуватися, доки не завершені інші завдання; інші завдання мають виконуватися паралельно і т. п. Якщо порушується синхронізація виконання різних завдань, весь проєкт може опинитися під загрозою невиконання.
- Часові рамки проєкту. Проєкти виконуються протягом певного проміжку часу (хоча інколи керівникам проєктів, що обстоюють виконання початкових графіків, здається, що проєкт не буде завершено ніколи) і мають більш-менш чітко окреслені початок і закінчення. Проєкт вважається завершеним, коли досягнуті його основні цілі. Під час виконання проєкту значні зусилля спрямовані саме на те, щоб його було завершено у намічений термін. У цьому допомагають графіки, де зазначається час початку і закінчення робіт, які передбачаються проєктом.
- Наявність бюджету. Проєктна діяльність, спрямована на отримання певного результату у заданий проміжок часу, не може відбутися без використання певних ресурсів (матеріальних, людських, фінансових). Тому невід’ємною рисою проєкту є наявність бюджету, який виділяється на 12 забезпечення ресурсних потреб фінансування проєкту, що відповідають його масштабам, змісту і термінам виконання.
- Унікальність. Проєкти — це певною мірою неповторні та одноразові заходи. Водночас рівень унікальності може значно коливатися залежно від особливостей проєкту. Скажімо, якщо йдеться про зведення п’ятдесятого будинку у стилі «стандарт» за програмою житлової забудови, то рівень унікальності цього проєкту досить скромний. Базові елементи такого будинку ідентичні елементам тих сорока дев’яти будинків, що їх було зведено раніше. Проте основні елементи унікальності можуть відбиватися у специфіці земельної ділянки, де розташовується будинок, у рішенні налагодити нову систему опалення і вентиляції або у необхідності працювати з новою бригадою фахівців і т. ін.

Загалом, саме ці п’ять ознак, або характеристик, відрізняють проєкти від інших заходів, планів, програм, ініціатив.

Управління проєктами (project management) - це структурований підхід до планування, організації та контролю за виконанням проєктів з метою досягнення конкретних цілей у встановлені строки, при дотриманні визначених бюджетних і ресурсних обмежень. Ця спеціалізована галузь управління включає в себе застосування різноманітних знань, навичок, інструментів і методів, які допомагають командам успішно реалізовувати проєктні завдання. Основні етапи управління проєктами включають ініціювання, планування, виконання, моніторинг та завершення проєкту, причому кожен етап потребує детального аналізу та управлінських рішень [2].

Етап ініціювання передбачає визначення основних цілей проєкту, його масштабів, зацікавлених сторін та загальних вимог. На цьому етапі здійснюється оцінка доцільності проєкту, формулюються основні завдання і створюється проєктний статут, який затверджується керівництвом. Етап планування є одним з найважливіших, оскільки включає розробку детального плану проєкту, що охоплює графік виконання робіт, бюджет, розподіл ресурсів, управління ризиками, а також визначення критеріїв якості та стандартів виконання. Ретельне планування забезпечує основу для ефективного виконання та контролю проєкту.

На етапі виконання відбувається реалізація плану проєкту, включаючи координацію діяльності команди, управління ресурсами та виконання поставлених завдань. Етап моніторингу та контролю включає постійний нагляд за прогресом виконання проєкту, порівняння фактичних результатів з плановими показниками, і вжиття коригувальних заходів у разі відхилень. Завершення проєкту передбачає офіційне прийняття виконаних робіт, оцінку досягнення цілей, документацію підсумків та навчання на основі отриманого досвіду. Успішне управління проєктами допомагає організаціям ефективно використовувати свої ресурси, досягати стратегічних цілей і забезпечувати високу якість кінцевих результатів [4].

Цикл управління проєктом включає кілька основних етапів:

- Ініціювання - визначення мети, масштабів, вимог до проєкту, формування команди та отримання необхідних дозволів.
- Планування - створення деталізованого плану дій, визначення завдань, термінів, бюджету, розподіл ролей та обов'язків, оцінка ризиків.
- Виконання - координація людських та матеріальних ресурсів відповідно до плану, реалізація запланованих робіт.
- Моніторинг та контроль - відстеження стану виконання проєкту, перевірка відповідності плану, аналіз відхилень та вжиття коригувальних заходів.
- Завершення - формалізація прийняття результатів проєкту замовником, розформування команди, підведення підсумків.

Управління проєктами також включає аспекти управління інтеграцією, змістом, якістю, ризиками, зацікавленими сторонами та іншими областями знань.

Завдання управління проєктами:

- визначення цілей проєкту і проведення його обґрунтування; - виявлення структури проєкту;
- визначення необхідних обсягів і джерел фінансування;
- підбір виконавців;
- підготовка і висновок контрактів;
- визначення термінів виконання;
- розроблення графіка реалізації проєкту;
- розрахунок необхідних ресурсів;
- складання кошторису і бюджету проєкту;
- контроль за ходом виконання проєкту;
- моніторинг проєкту.

Еволюція методів управління проєктами:

1. Техніка мережевого управління (широко застосовується із 70-х рр. XX ст.);
2. Організація робіт над проєктом (використовується з 1975 р.);
3. Календарне управління (використовується з 1975 р.);

4. Логістика (використовується з 1975 р.);

5. ППП для ЕОМ (із 80-х рр. XX ст.);

6. Стандартне управління (із 80-х рр. XX ст.);

7. Структурне управління (із 80-х рр. XX ст.);

8. Ресурсне управління (із 80-х рр. XX ст.);

9. Системний підхід (зокрема, до фази закриття проєкту) – з 1985 р.;

10. Планування і розроблення особливо складних проєктів – з 1985 р.;

11. Пофазна організація роботи над проєктом – з 1985 р.;

12. Імітаційне моделювання – з 1990 р.;

13. Системний підхід до проєкту в цілому – із 90-х рр. XX ст.;

14. Філософія керівництва проєктом – з 1995 р.

Причини появи управління проєктами:

1. Підвищуються темпи змін у промисловості, тому управління проєктами — це один із шляхів досягнення успіху у змаганні зі змінами.

2. Умови ринку стають більш вибагливими, проєкти — масштабнішими і такими, що потребують більшого професіоналізму в управлінні.

3. Дуже часто діяльність менеджерів пов’язана з виконанням проєктів, проте управління проєктами відрізняється від іншої управлінської діяльності, вимагаючи спеціальних умінь, інструментів, організаційної структури тощо.

4. Поглиблюються проблеми інтеграції як різних компаній, так і різних видів діяльності у ході виконання проєктів.

Будь-який проєкт, навіть найменший, вимагає застосування методології управління проєктами (УП) та визначення відповідального за проєкт. УП забезпечує структурований підхід до організації роботи над проєктом, що включає планування, виконання, моніторинг та контроль, а також завершення проєкту. Ця методологія допомагає чітко визначити цілі, задачі, ролі та відповідальності, а також оптимізувати використання ресурсів і мінімізувати ризики. Відповідальний за проєкт, або проєктний менеджер, несе основну відповідальність за координацію роботи команди, виконання завдань у строк та відповідність проєкту заданим вимогам [5].

Для малих і середніх монопроєктів, які не потребують великої кількості ресурсів та складної координації, застосування методів УП можливе без використання спеціальних технічних та інформаційно-програмних засобів. Прості інструменти, такі як електронні таблиці або базові програми для управління завданнями, можуть бути достатніми для успішного виконання таких проєктів. Однак, для середніх і великих мультипроєктів, де є багато залежностей між завданнями і ресурсами, окремі технічні та програмні засоби стають необхідними для ефективного управління. Це можуть бути спеціалізовані програмні платформи, які дозволяють централізовано контролювати всі аспекти проєкту.

Зміст роботи з управління проєктами включає визначення об’єктів та процесів, необхідних для створення цих об’єктів. Це означає, що кожен проєкт розглядається як сукупність взаємопов’язаних елементів, де кожен елемент виконує певну функцію. Об’єкти можуть бути як фізичними, так і нематеріальними, такими як документи, плани чи звіти. Процеси включають всі дії, необхідні для створення, перевірки та завершення цих об’єктів. Наприклад, процеси можуть включати проектування, розробку, тестування, впровадження та моніторинг [8].

Предметна область проєкту декомпозується у його структурній моделі за декількома рівнями на часткові об’єкти та процеси. Цей підхід дозволяє більш детально розглянути всі складові проєкту, визначити їх взаємозв’язки та залежності. Декомпозиція проєкту на часткові об’єкти та процеси забезпечує кращу прозорість та керованість, дозволяє точніше планувати ресурси та оцінювати ризики. Відповідно, управління проєктом стає більш ефективним, оскільки керівництво має можливість зосередитися на конкретних аспектах та приймати більш обґрунтовані рішення.

Оскільки цілі проєкту можуть змінюватися в ході його реалізації, а виявлені помилки повинні бути виправлені, необхідне систематичне управління змінами. Управління змінами включає процеси планування змін, контролю за їх проведенням та оцінку їх впливу на строки, витрати та інші характеристики проєкту. Це дозволяє зберігати контроль над проєктом навіть при виникненні непередбачуваних обставин або необхідності коригування початкових планів. Завдяки

систематичному підходу до управління змінами, можна мінімізувати негативні наслідки і забезпечити успішне завершення проєкту з дотриманням всіх встановлених вимог.На сучасному етапі спостерігається зростання популярності гнучких ітераційних методологій управління проєктами, таких як Agile, Scrum, Kanban, які забезпечують більшу адаптивність та залучення команди [7].

Waterfall (водоспадна модель) - це послідовна модель життєвого циклу розробки програмного забезпечення, де процес розбитий на окремі фази (збір вимог, проєктування системи, реалізація, тестування, розгортання та супровід), які виконуються послідовно одна за одною. Перехід на наступну фазу можливий лише після завершення та схвалення результатів попередньої фази. Це забезпечує чіткий і структурований підхід, але ускладнює внесення змін на пізніх етапах і вимагає ретельного визначення всіх вимог на початку проєкту.

Agile - це гнучкий підхід до розробки продуктів та управління проєктами, який фокусується на ітераційній розробці, тісній співпраці в командах та швидкій адаптації до змін. Основні принципи Agile передбачають залучення замовників, постійне тестування, регулярні ітерації та гнучкість у плануванні. Методологія Agile відкидає традиційний каскадний підхід на користь більш адаптивного, ітеративного циклу розробки та поставки.

Scrum - це одна з найпопулярніших реалізацій Agile у сфері управління проєктами розробки програмного забезпечення. Вона передбачає розподіл проєкту на короткі ітерації (спринти), які зазвичай тривають 2-4 тижні. Scrum акцентує увагу на щоденних зустрічах команди, аналізі прогресу та усуненні перешкод. Ключові ролі включають ScrumMaster, Product Owner та Scrum команду.

Kanban - це методологія управління потоком робіт, яка візуалізує процеси на канбан-дошці з різними колонками для поточних завдань. Kanban фокусується на максимізації ефективності та безперервному поліпшенні процесів. Команди візуально відстежують рух завдань через різні стадії, виявляють вузькі місця та обмеження для оптимізації потоку робіт.

Рисунок 1.1. Agile методологія

Спільними рисами цих методологій є:

- Ітераційний підхід з короткими циклами розробки замість великих одноразових поставок;
- Залучення клієнта/замовника та регулярне тестування для врахування змін вимог;
- Самоорганізовані багатофункціональні команди з різними ролями;
- Візуалізація робочих процесів і статусу завдань для кращої прозорості;
- Акцент на постійному поліпшенні процесів замість чіткого дотримання початкового плану;
- Гнучкість та готовність до змін, а не суворе дотримання фіксованих специфікацій;
- Робота над найбільш пріоритетними завданнями для швидкої реалізації ключових цінностей.

Методології управління проєктами, такі як Agile, Scrum і Kanban, здобули широке розповсюдження завдяки своїй здатності реагувати на зміни та адаптуватися до нових вимог. Їх гнучкість дозволяє швидко вносити корективи у

процес розробки, що є особливо важливим у середовищі з високим ступенем невизначеності. Крім того, ці методології сприяють створенню частих і працюючих поставок, що дозволяє командам постійно перевіряти і покращувати продукт на основі зворотного зв’язку від замовників. Цей підхід значно знижує ризики і підвищує ймовірність успішного завершення проєкту [7].

Ефективне управління проєктами також потребує використання спеціалізованих інструментів та програмного забезпечення, які допомагають візуалізувати завдання і контролювати строки їх виконання. Інструменти, такі як Trello, Jira або Asana, дозволяють створювати візуальні представлення робочих процесів, що полегшує розуміння стану проєкту всіма учасниками команди. Вони також забезпечують можливість відстеження прогресу, що дозволяє проєктним менеджерам і командам своєчасно виявляти і вирішувати проблеми, що виникають під час реалізації проєкту. Важливою функцією цих інструментів є можливість планування та контролю за термінами виконання завдань, що допомагає дотримуватися встановлених графіків.

Окрім візуалізації та контролю, сучасні інструменти управління проєктами забезпечують ефективний розподіл ресурсів і співпрацю учасників у режимі реального часу. Це особливо важливо для команд, які працюють віддалено або в різних часових зонах. Програмне забезпечення для управління проєктами дозволяє членам команди ділитися інформацією, документами і коментарями, забезпечуючи прозорість і доступність даних для всіх зацікавлених сторін. Це сприяє кращій координації дій, швидкому прийняттю рішень і забезпеченню того, що всі учасники проєкту мають однакове розуміння завдань і цілей, що, в свою чергу, підвищує ефективність і якість кінцевого продукту [11].

1.2 Аналіз об’єкта дослідження

Перші онлайн-системи для управління проєктами почали з'являтися наприкінці 1990-х - на початку 2000-х років, коли інтернет та веб-технології стали більш доступними та поширеними. Це був період, коли компанії та організації почали усвідомлювати потенціал інтернету для поліпшення співпраці та координації робочих процесів. Одним із ранніх піонерів у цій галузі була система Basecamp, запущена в 2004 році, яка запропонувала простий і ефективний спосіб управління проєктами через веб-інтерфейс. Basecamp швидко здобула популярність завдяки своїй простоті використання та зручному інтерфейсу, що значно полегшило організацію та моніторинг проєктів.

Ці ранні онлайн-системи заклали основу для подальшого розвитку індустрії управління проєктами. Вони впровадили ключові концепції, такі як централізоване зберігання документів, можливість спільної роботи в режимі реального часу, календарі завдань, а також системи для відстеження прогресу проєктів. Важливою перевагою цих систем була можливість доступу до інформації з будь-якого місця та пристрою, що підключений до інтернету, що суттєво підвищувало гнучкість та мобільність команди. Це було особливо важливо для компаній з віддаленими командами або офісами, розташованими у різних географічних точках.

З розвитком технологій та інтернету онлайн-системи управління проєктами ставали дедалі більш потужними та функціональними. Вони почали включати в себе розширені функції, такі як інтеграція з іншими інструментами та сервісами, аналітичні можливості, автоматизація рутинних завдань, а також розширені функції безпеки для захисту даних. Сучасні платформи, такі як Trello, Asana, Jira та інші, стали невід'ємною частиною робочих процесів багатьох компаній по всьому світу. Вони забезпечують комплексний підхід до управління проєктами, дозволяючи командам ефективно планувати, координувати та виконувати завдання, що, в свою чергу, сприяє підвищенню продуктивності та досягненню стратегічних цілей [3].

Одними з перших були:

- Niku (заснована в 1995 році). Niku розробила одну з перших веб-орієнтованих систем управління портфелями проєктів та ресурсами. Її особливостями були централізоване планування проєктів, відстеження витрат та аналітика;
- Primavera Systems (1983). Хоча Primavera розпочала як локальне програмне забезпечення, в 1998 році вона випустила одну з перших онлайн-систем управління проєктами ProSight для спільної роботи над проєктами;
- Bhindustan Construction Company (1999). Один із перших індійських провайдерів онлайн-рішень для спільної роботи над проєктами у сфері будівництва та інфраструктури;
- eProject (1998). eProject була однією з ранніх платформ управління проєктами на основі підписки, що надавала інструменти для спільної роботи, управління завданнями та документами через інтернет;
- eRoom Technology (1996). eRoom розробила одну з перших хмарних систем для спільної роботи над проєктами з функціями управління документами, обміну повідомленнями та календарями.

Ці ранні системи, хоча й мали певні обмеження у функціоналі та продуктивності порівняно з сучасними рішеннями, проклали шлях для переходу від локальних інструментів управління проєктами до онлайн-співпраці. Вони продемонстрували переваги централізованого доступу, спільного використання даних та мобільності для команд, які працюють над складними проєктами.

У 2000-х набули поширення більш повнофункціональні SaaS-рішення, як от Basecamp, одна з перших платформ для онлайн-управління проєктами, запущена у 2004 році.

З часом, завдяки розвитку веб-технологій, хмарних обчислень та методологій розробки програмного забезпечення, онлайн-платформи управління проєктами стали більш функціональними, потужними та інтуїтивними у використанні, як от Trello, Asana, Jira та інші лідери ринку сьогодні.

Сучасні онлайн-системи все більше інтегрують AI, автоматизацію, можливості візуального проєктування, а також адаптуються під мобільні пристрої та гнучкі робочі процеси.

Онлайн-системи управління проєктами (Project Management Systems) відіграють важливу роль у впорядкуванні, організації та виконанні проєктів у різних галузях та типах організацій. Вони надають засоби для планування, відстеження, співпраці та звітності, що спрощує управління завданнями та допомагає досягати поставлених цілей [14].

Ключовими аспектами сучасних онлайн-систем управління проєктами є:

- Інтерфейс користувача: Сучасні платформи зазвичай мають інтуїтивно зрозумілий та дружній інтерфейс, що сприяє простоті використання навіть для новачків;
- Функціональні можливості: Вони забезпечують широкий спектр функціональності, такий як створення проєктів та завдань, призначення відповідальних, налаштування термінів та пріоритетів, відстеження прогресу, спільна робота над завданнями, комунікація, звітність та аналіз;
- Гнучкість: Більшість платформ пропонують можливості налаштування та адаптації до конкретних потреб команди чи проєкту, так що вони можуть використовуватися у різних сферах та з урахуванням різних методологій управління проєктами;
- Хмарне зберігання та доступ: Вони забезпечують можливість зберігання даних у хмарі, що дозволяє користувачам отримувати доступ до своїх проєктів з будь-якого пристрою та місця, де є Інтернет;
- Спільна робота та комунікація: Онлайн-платформи надають засоби для спільної роботи над завданнями та проєктами, обміну документами та ідеями, а також для комунікації у команді через коментарі, чати, електронну пошту тощо;
- Інтеграція з іншими інструментами: Багато платформ забезпечують можливість інтеграції з іншими популярними інструментами та сервісами, такими як електронна пошта, календарі, чати, засоби співпраці тощо;

- Аналітика та звітність: Вони надають інструменти для аналізу даних про прогрес проєктів, витрати ресурсів, ризики, ефективність команди тощо, що дозволяє керувати проєктами більш ефективно та приймати обґрунтовані рішення;
- Безпека: Онлайн-платформи управління проєктами забезпечують високий рівень безпеки для збереження конфіденційності та цілісності даних проєктів.

Цілі онлайн-систем управління проєктами:

- Централізоване середовище для консолідації всієї інформації про проєкти, завдання, ресурси, терміни тощо;
- Покращення співпраці та комунікації в командах, особливо для географічно розподілених працівників;
- Підвищення прозорості проєктів, забезпечення видимості статусу завдань та загального прогресу;
- Оптимізація планування та розподілу ресурсів, завдань і термінів між учасниками проєкту;
- Зменшення ризиків і проблем через ефективніший моніторинг і контроль проєктних процесів;
- Підвищення продуктивності команд шляхом візуалізації робочих потоків та усунення вузьких місць;
- Генерування звітів та аналітики для оцінки ефективності та вдосконалення процесів.

Ключові функції онлайн-систем управління проєктами:

- Управління завданнями та проєктними дошками;
- Призначення завдань, відстеження статусу та пріоритетів;
- Планування термінів, ресурсів, створення графіків проєктів;
- Співпраця в режимі реального часу, обговорення та спільний доступ;
- Відстеження витраченого часу, хронометраж робіт;
- Управління документами та файлами проєкту;
- Генерування звітів про прогрес, ресурси, виконання бюджету тощо;

- Інтеграція з іншими сервісами: календарями, месенджерами, системами документообігу.

Загалом, онлайн-системи управління проєктами еволюціонували від простих інструментів спільної роботи до комплексних хмарних рішень, що охоплюють весь життєвий цикл проєкту та оптимізують процеси планування, виконання, моніторингу та звітності.

1.3 Опис предмету дослідження

Предметом дослідження мого проєкту є розробка веб-застосунків за допомогою новітніх технологій та способів, таких як фреймворк Next.js.

Розробка веб-застосунків (web application development) є комплексним процесом створення програмного забезпечення, яке функціонує через інтернет або локальну мережу у веб-браузері. Веб-застосунки включають все: від простих веб-сторінок до складних платформ, таких як соціальні мережі, інтернет-магазини або онлайн-системи управління проектами. Процес розробки починається з аналізу вимог та проектування архітектури, де визначаються функціональні можливості, цілісність даних, безпека і зручність користувацького інтерфейсу. Після цього розробляється прототип, який потім реалізується у вигляді готового продукту [5].

Процес розробки веб-застосунків складається з кількох ключових етапів, кожен з яких має своє значення і важливість для створення ефективного та надійного продукту. Ці етапи включають аналіз вимог, проектування, розробку, тестування, розгортання та підтримку. Кожен з цих етапів потребує тісної співпраці між замовниками, розробниками та іншими зацікавленими сторонами для забезпечення успіху проекту.

Основними етапами розробки веб-застосунків є:

1. Аналіз вимог. Перший етап розробки веб-застосунків включає збір та аналіз вимог до продукту. Це передбачає визначення цілей проекту, функціональних та нефункціональних вимог, а також виявлення потенційних користувачів і їхніх

потреб. Під час цього етапу розробники співпрацюють з клієнтами та користувачами, щоб зрозуміти їх очікування і розробити технічне завдання, яке визначає, що саме повинно бути реалізовано.

2. Проєктування. На цьому етапі створюється архітектура веб-застосунку. Це включає розробку структурної схеми системи, яка визначає взаємодію між клієнтською та серверною частинами, а також інтеграцію з базами даних та зовнішніми сервісами. Розробляються макети інтерфейсу користувача, які показують, як виглядатиме і функціонуватиме веб-застосунок. Проектування також включає визначення технологій і інструментів, які будуть використовуватися для розробки.

3. Розробка. Цей етап охоплює написання коду для клієнтської та серверної частин веб-застосунку. Розробники створюють функціональність інтерфейсу користувача, логіку додатку та інтеграцію з базами даних і зовнішніми API. Під час розробки важливо дотримуватися принципів чистого коду і використовувати системи контролю версій для управління змінами в коді. Цей етап також включає регулярні перегляди та рефакторинг коду для забезпечення його якості та ефективності.

4. Тестування. Після завершення розробки проводиться тестування веб-застосунку для виявлення і виправлення помилок. Тестування може включати юніт-тестування, інтеграційне тестування, функціональне тестування і тестування безпеки. Автоматизовані тести допомагають швидко виявляти помилки під час розробки, а ручне тестування забезпечує перевірку користувацького досвіду.

5. Розгортання. Після успішного тестування веб-застосунок готується до розгортання на сервері або в хмарному середовищі. Цей етап включає налаштування середовища виконання, встановлення всіх необхідних залежностей і конфігурацію сервера. Після розгортання проводиться фінальне тестування, щоб переконатися, що застосунок працює коректно у виробничому середовищі.

6. Підтримка та оновлення. Після розгортання веб-застосунку починається етап підтримки, який включає моніторинг роботи системи, виявлення і виправлення помилок, а також впровадження нових функцій та оновлень. Регулярні оновлення

забезпечують безпеку і ефективність веб-застосунку, а також задовольняють потреби користувачів у нових функціях та покращеннях.

Клієнтська частина (frontend) веб-застосунку включає в себе все, що користувач бачить і взаємодіє з на екрані. Це включає дизайн, розмітку, стилізацію та функціональність інтерфейсу користувача. Для створення клієнтської частини зазвичай використовуються HTML, CSS і JavaScript, а також різні фреймворки та бібліотеки, такі як React, Angular або Vue.js. Основне завдання клієнтської частини – забезпечити зручний і інтуїтивно зрозумілий інтерфейс, який дозволяє користувачам легко взаємодіяти з веб-застосунком [5].

Серверна частина (backend) веб-застосунку відповідає за обробку даних, управління базами даних, аутентифікацію та авторизацію користувачів, а також взаємодію з клієнтською частиною через API. Серверний код зазвичай пишеться на таких мовах програмування, як Python, Ruby, Java, PHP або JavaScript (Node.js). Крім того, використання фреймворків, таких як Django, Rails, Spring або Express.js, значно спрощує процес розробки. Серверна частина також забезпечує безпеку, масштабованість та ефективність веб-застосунку, гарантуючи, що всі операції виконуються швидко та безпечно. Інтеграція з базами даних, такими як PostgreSQL, MySQL або MongoDB, дозволяє зберігати і управляти великими обсягами даних, забезпечуючи їх доступність і цілісність [5].

Веб-застосунки можуть бути статичними (тільки клієнтська частина) або динамічними (з серверною частиною для обробки даних і логіки). Вони бувають різних типів: інформаційні сайти, електронна комерція, хмарні сервіси, Single Page Applications, соціальні мережі тощо. Процес їх розробки прагне створити функціональний, ефективний, безпечний та зручний для користувачів продукт.

Раніше процес створення веб-застосунків був значно складнішим і менш гнучким, ніж сьогодні. Ось як він виглядав на ранніх етапах розвитку веб-технологій:

1. Статичні HTML сторінки На самому початку веб-сайти складалися з окремих статичних HTML файлів, які безпосередньо надсилалися веб-сервером користувачам. Будь-які зміни потребували редагування HTML файлів вручну і повторного розгортання.

2. Серверні скрипти (CGI, Perl, PHP) Пізніше з'явилися серверні скрипти CGI, Perl, PHP тощо, які дозволяли динамічно генерувати HTML на стороні сервера. Але логіка застосунку і HTML були змішані, що ускладнювало розробку і підтримку.

3. Серверні фреймворки Потім почали використовуватись перші серверні фреймворки, такі як Ruby on Rails, ASP.NET, які уніфікували архітектуру веб-додатків та полегшували розробку за рахунок концепцій шаблонізації, ОРМ, маршрутизації тощо.

4. Клієнтські скрипти (JavaScript) Поступово почала зростати роль JavaScript на клієнтській стороні для додавання інтерактивності та обробки подій. Однак здебільшого клієнтська частина була тісно пов'язана з HTML.

5. AJAX Технологія AJAX (Asynchronous JavaScript and XML) дозволила робити асинхронні запити на сервер без перезавантаження всієї сторінки, відкривши шлях до більш динамічних інтерфейсів.

6. Клієнтські MV* фреймворки Виникли клієнтські архітектурні патерни і фреймворки (Angular, Backbone, Ember), що сприяли розділенню клієнтської логіки від HTML і дозволяли створювати більш структуровані веб-додатки.

7. Односторінкові застосунки (SPA) Підхід SPA (Single Page Applications) став поширеним з появою React, Vue та сучасних фреймворків, що робили акцент на реактивності, віртуальному DOM та розмежуванні проблемної області між клієнтом і сервером.

8. Серверний рендеринг (Next.js, Nuxt.js) Нарешті, фреймворки наступного покоління, такі як Next.js, Nuxt.js, сполучили переваги SPA з покращеною продуктивністю та SEO за допомогою серверного рендерингу та технік, як пре-рендеринг, гідратація, інкрементальна статична регенерація.

Single Page Application (SPA) – це веб-застосунок або веб-сайт, що складається з однієї єдиної веб-сторінки, яка динамічно оновлює свій вміст, не

перезавантажуючи всю сторінку. Це досягається за допомогою JavaScript і сучасних фреймворків, таких як React, Angular або Vue.js. В SPA навігація між різними розділами здійснюється шляхом завантаження нових даних та оновлення частини сторінки, що забезпечує швидку та плавну взаємодію з користувачем [16].

Однією з головних переваг SPA є поліпшення користувацького досвіду завдяки швидшій навігації та меншому навантаженню на сервер. Оскільки більшість необхідних ресурсів завантажуються один раз під час першого завантаження сторінки, подальші запити до сервера надсилаються лише для отримання потрібних даних, а не для повного оновлення сторінки. Це дозволяє значно зменшити час відгуку і зробити взаємодію з веб-застосунком більш швидкою та приємною.

Однак розробка SPA також має свої виклики, такі як проблеми з SEO (оптимізацією для пошукових систем), складність в управлінні станом застосунку та безпекою. Для вирішення проблем з SEO застосовуються техніки серверного рендерингу або гібридні підходи, такі як використання Next.js для React. SPA також потребують добре продуманого підходу до управління станом, для чого використовуються бібліотеки, такі як Redux або Vuex. Незважаючи на ці виклики, SPA залишаються популярними завдяки своєму сучасному підходу до розробки та значним покращенням користувацького досвіду.

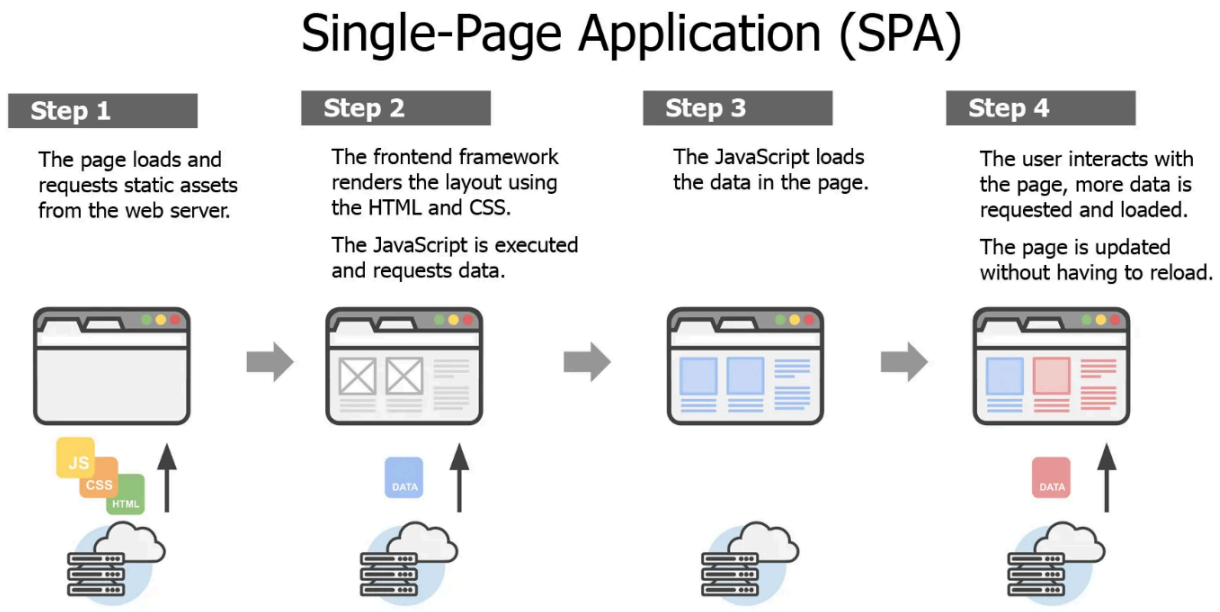


Рисунок 1.2. Single-Page Application (SPA)

Ключові особливості SPA:

- Використовують JavaScript для оновлення частин вебсторінки замість повного перезавантаження;
- Обмінюються даними із сервером через API (Ajax, WebSockets);
- Маршрутизація на клієнті для імітації переходів між "сторінками";
- Високий рівень інтерактивності та відгуку;
- Краща користувацька взаємодія, схожа на мобільні додатки;

Server-Side Rendering (SSR) – це метод рендерингу веб-сторінок, при якому HTML-код генерується на сервері і відправляється на клієнтський браузер для відображення. Це відрізняється від традиційного підходу в Single Page Applications (SPA), де рендеринг здійснюється на клієнтській стороні за допомогою JavaScript. У SSR браузер отримує повністю сформовану сторінку, що дозволяє швидше відображати вміст для користувачів та поліпшує продуктивність, особливо на повільних з'єднаннях. Крім того, це зменшує час до першого значущого малюнка (First Contentful Paint), що покращує користувацький досвід [21].

Однією з основних переваг SSR є покращення SEO (оптимізації для пошукових систем). Оскільки пошукові боти отримують готовий HTML-код, вони можуть легше індексувати вміст сторінок. Це особливо важливо для динамічних веб-застосунків, де вміст постійно змінюється. Крім того, SSR сприяє швидшому часу першого завантаження (First Contentful Paint), оскільки браузеру не потрібно чекати завантаження та виконання JavaScript-коду для рендерингу вмісту. Це дозволяє забезпечити кращий користувацький досвід для користувачів із повільними або ненадійними інтернет-з'єднаннями.

Однак SSR має і свої виклики. Генерація HTML на сервері може збільшити навантаження на серверні ресурси, особливо при обробці великої кількості одночасних запитів. Це може вимагати більш потужного серверного обладнання та ефективної оптимізації серверного коду. Крім того, інтеграція SSR в існуючі

проекти може бути складнішою і вимагати додаткових знань у розробників. Незважаючи на ці виклики, багато сучасних фреймворків, такі як Next.js для React або Nuxt.js для Vue.js, пропонують вбудовану підтримку SSR, що значно спрощує його впровадження. Таким чином, SSR залишається популярним вибором для створення високопродуктивних і SEO-оптимізованих веб-застосунків.

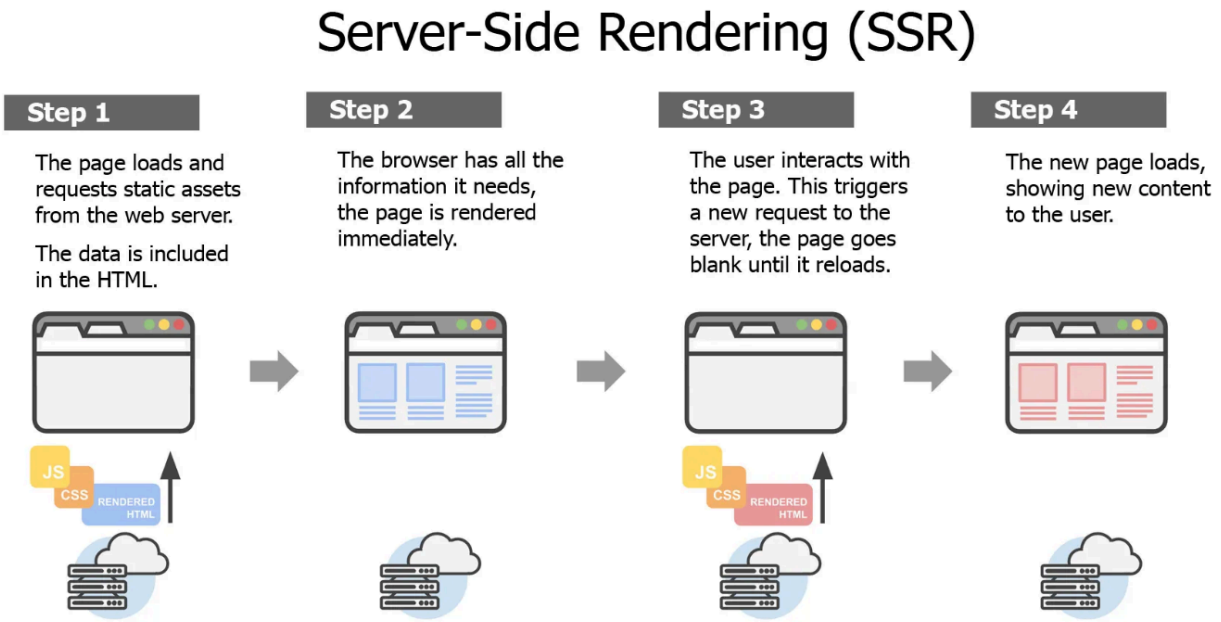


Рисунок 1.3. Server-Side Rendering (SSR)

Переваги SSR:

- Покращена продуктивність первинного завантаження
- Краще індексування та SEO порівняно з чистими SPA
- Менший час до перших змістовних візуалізацій
- Зменшені вимоги до клієнтських ресурсів для початкового рендерингу

Фреймворки, такі як Next.js, дозволяють комбінувати переваги SPA та SSR підходів за допомогою таких технік:

- Пре-рендеринг - статичний рендеринг під час збірки для сторінок без даних;
- Серверний рендеринг - динамічний рендеринг на сервері для сторінок з даними;

- Клієнтський рендеринг - традиційне гідратоване SPA-подібне оновлення після початкового рендерингу;
- Інкрементальна статична регенерація - повторне створення статичних сторінок після оновлення даних.

Таким чином, правильне поєднання SPA та SSR дозволяє досягти найкращої продуктивності, відгуку, SEO та зручності як для користувачів, так і для розробників.

На сучасному етапі веб-розробка значно еволюціонувала, перетворившись з простих, статичних веб-сторінок на складні, інтерактивні системи. Цей розвиток був поступовим і відбувався протягом кількох десятиліть, кожен етап приносив нові інструменти, технології та підходи. Зараз веб-розробка стала більш структурованою завдяки чіткому розмежуванню між серверною та клієнтською частинами. Це розмежування дозволяє розробникам краще керувати кодом, забезпечувати безпеку та масштабованість систем.

Важливою частиною сучасної веб-розробки є компонентний підхід, який дозволяє створювати модульні та багаторазово використовувані частини коду. Це робить процес розробки більш ефективним, дозволяючи розробникам легко підтримувати та оновлювати окремі частини веб-застосунків без потреби переробляти всю систему. Крім того, використання новітніх патернів архітектури, таких як Model-View-Controller (MVC) та Flux, допомагає структурувати код та покращувати взаємодію між різними частинами застосунку [19].

Сучасні інструменти та фреймворки, такі як React, Angular, Vue.js, Next.js та інші, надають розробникам потужні можливості для створення складних та інтерактивних веб-застосунків. Ці інструменти забезпечують високу продуктивність, масштабованість та зручність розробки. Загалом, веб-розробка сьогодні є високотехнологічною та динамічною сферою, яка постійно вдосконалюється і адаптується до нових вимог та викликів цифрового світу.

1.4 Аналіз актуальності

Управління проєктами є критично важливим процесом для забезпечення ефективної діяльності організацій у сучасному динамічному бізнес-середовищі. Здатність швидко адаптуватися до мінливих умов ринку, впроваджувати інновації та оптимізувати використання ресурсів стає ключовим фактором успіху та конкурентоспроможності компаній [14].

Одним із головних викликів для організацій є координація зусиль різних підрозділів, забезпечення прозорості процесів та ефективної комунікації між усіма зацікавленими сторонами проєкту. Традиційні методи управління, такі як електронні таблиці чи паперові документи, часто виявляються неефективними та призводять до втрати часу, дублювання зусиль та неузгодженості даних.

Розробка власної онлайн-системи для управління проєктами дозволить організаціям створити централізоване сховище даних, де вся інформація про проєкти буде зібрана в одному місці та доступна усім учасникам у режимі реального часу. Це забезпечить прозорість процесів, підвищить продуктивність команд та зменшить ризики втрати важливих даних.

Онлайн-система надасть зручні інструменти для планування, відстеження прогресу, управління ресурсами та комунікації між членами команди. Це дозволить ефективно розподіляти завдання, встановлювати пріоритети, відстежувати термінові задачі та оперативно реагувати на зміни в проєктах.

Важливим аспектом є також можливість аналізу та звітності, що дозволить керівництву організації отримувати актуальну інформацію про стан проєктів, виявляти потенційні ризики та приймати обґрунтовані рішення.

Окремо слід відзначити переваги онлайн-системи для віддалених команд та проєктів з розподіленими ресурсами. Завдяки можливості доступу з будь-якої точки світу, де є інтернет-з'єднання, віддалені співробітники зможуть ефективно взаємодіяти та співпрацювати, не відчуваючи географічних бар'єрів.

Для успішного впровадження онлайн-системи для управління проєктами, необхідно врахувати також аспекти безпеки даних та конфіденційності.

Забезпечення захисту інформації про проєкти від несанкціонованого доступу та збереження конфіденційності даних клієнтів і бізнес-партнерів є критично важливим для будь-якої організації. Тому розробка системи повинна включати найсучасніші технології шифрування даних, механізми автентифікації та контролю доступу, а також регулярні аудити безпеки для виявлення та усунення можливих загроз.

Крім того, важливо пам'ятати про потребу в постійній підтримці та оновленні системи. Розвиток технологій та зміни в бізнес-процесах вимагають постійного вдосконалення та адаптації онлайн-платформи для управління проєктами. Через регулярне вдосконалення функціоналу та інтеграцію нових інструментів, організація зможе підтримувати свою конкурентоспроможність і відповідати змінам у вимогах ринку.

Таким чином, розробка онлайн-платформи для управління проєктами є актуальним та необхідним рішенням для забезпечення ефективної діяльності сучасних організацій, підвищення їх конкурентоспроможності та адаптивності до мінливих умов ринку.

1.5 Стан вже існуючих рішень

На ринку вже представлено низку популярних інструментів та платформ для управління проєктами, які активно використовуються організаціями у різних галузях. Одним з лідерів є Trello - веб-застосунок для управління проєктами за допомогою гнучких дошок, на яких можна створювати і переміщати картки, представляючи завдання та їхній стан.

Розробники Trello, компанія Atlassian, зробили акцент на простоті та інтуїтивності інтерфейсу, що забезпечує легкий початок роботи з інструментом. Проте функціональність Trello може бути обмеженою для більш масштабних та комплексних проєктів з великою кількістю учасників та численними завданнями.

Іншим популярним рішенням є Asana - потужна онлайн-платформа для управління проєктами, яка пропонує широкий спектр можливостей: створення проєктів та завдань, встановлення термінів, призначення відповідальних,

відстеження прогресу, інтеграцію з іншими інструментами та багато іншого. Asana має зручний інтерфейс та різноманітні способи візуалізації проєктів, але може бути складною у використанні для невеликих команд.

Одним з найпотужніших і гнучких рішень для управління проєктами вважається Jira від Atlassian. Ця платформа пропонує всеосяжну функціональність для планування, відстеження та звітування про проєкти, але при цьому може бути надмірно складною та дорогою для невеликих організацій або стартапів.

Також на ринку присутні й інші рішення, такі як Basecamp, Wrike, Microsoft Project та інші, кожне з яких має свої переваги та недоліки, а також цільову аудиторію користувачів. Ці рішення вже зарекомендували себе на ринку та активно використовуються компаніями різних розмірів та галузей для керування їх проєктами.

Проте, незважаючи на наявність різноманітних інструментів, багато організацій відчувають потребу у власній спеціалізованій онлайн-платформі для управління проєктами, яка б була повністю адаптована під їхні унікальні бізнес-процеси, вимоги до безпеки та конфіденційності даних, а також інтегрувалася з іншими корпоративними системами. Для невеликих компаній використання відомих онлайн-систем управління проектами може бути фінансово обтяжливим. Щомісячні або щорічні ліцензійні платежі можуть скласти значну частину бюджету. Розробка власного рішення хоч і потребує початкових вкладень, але у довгостроковій перспективі може бути економічно вигіднішою. Власна платформа також може інтегруватися з іншими корпоративними системами, що дозволяє створити єдину інформаційну екосистему, підвищити ефективність процесів і забезпечити безперебійну роботу всієї організації. Це забезпечить додаткові конкурентні переваги та підвищить загальну ефективність управління проектами.

1.6 Визначення цілей дослідження та постановка задачі

Метою дослідження є створення онлайн-системи для ефективного управління проєктами в організаціях різного масштабу та галузей діяльності, що забезпечить

централізоване середовище для планування, організації, відстеження та контролю виконання проєктів.

Реалізація мети здійснюється вирішенням наступних основних завдань:

- Створення зручного та інтуїтивного користувацького інтерфейсу для взаємодії з платформою.
- Реалізація функціоналу для створення проєктів, завдань, встановлення термінів, призначення відповідальних осіб та відстеження прогресу.
- Впровадження механізмів для візуалізації проєктів, таких як гнучкі дошки завдань, діаграми Гантта тощо.
- Забезпечення можливості співпраці та комунікації між учасниками проєкту в режимі реального часу.
- Інтеграція з іншими корпоративними системами та інструментами для консолідації даних та підвищення продуктивності.

Об'єктом дослідження є процес управління проєктами в організаціях.

Предметом дослідження є розробка онлайн-системи для ефективного управління проєктами з використанням сучасних веб-технологій та підходів.

Система є онлайн-система для управління проєктами.

Зовнішнім середовищем є користувачі платформи (менеджери проєктів, команди розробників, керівництво організації) та дані про проєкти, завдання, ресурси тощо.

Розроблена онлайн-система дозволить підвищити ефективність управління проєктами, забезпечити прозорість процесів, оптимізувати використання ресурсів та полегшити співпрацю між учасниками проєктів. Крім того, вона може стати потужним інструментом для навчання та розвитку навичок з управління проєктами в організації.

2. АРХІТЕКТУРА СИСТЕМИ

2.1 Проєктування системи

Правильно спроєктована архітектура системи є критично важливою для успішної розробки та подальшого функціонування складних веб-додатків. Архітектура визначає загальну структуру системи, взаємозв'язки між її компонентами, а також принципи та підходи, які використовуються для досягнення цілей проєкту щодо продуктивності, масштабованості, надійності та безпеки [19].

У сучасному світі, де веб-додатки стають дедалі складнішими, ретельно продумана архітектура відіграє вирішальну роль в забезпеченні ефективного розподілу обов'язків, модульності, підтримки та можливості розширення системи. Добре спроєктована архітектура полегшує розуміння та розробку складних систем, зменшуючи ризики та підвищуючи загальну якість програмного забезпечення.

Під час проєктування архітектури необхідно враховувати численні аспекти, такі як вибір технологій, архітектурних підходів (мікросервіси, безсерверна архітектура тощо), забезпечення масштабованості, стійкості, безпеки, а також інтеграції з існуючими системами та сервісами. Крім того, архітектура повинна відповідати вимогам бізнесу, забезпечуючи відповідну функціональність та продуктивність для задоволення потреб користувачів.

MVC (Model-View-Controller) - це архітектурний шаблон програмного забезпечення, який розділяє додаток на три взаємопов'язані частини: модель, представлення та контролер. Цей шаблон широко використовується в розробці веб-додатків та інших типів додатків, оскільки він забезпечує чітке розділення відповідальностей та сприяє модульності коду.

Для онлайн-системи управління проєктами MVC може бути дуже корисним. Модель може містити логіку доступу до даних, таких як проєкти, задачі, користувачі та інші пов'язані об'єкти. Представлення відповідає за відображення даних користувачеві через інтерфейс користувача, наприклад, сторінки веб-додатку. Контролер обробляє запити користувача, взаємодіє з моделлю для отримання або оновлення даних і вибирає відповідне представлення для відображення результатів.

Використання MVC для онлайн-системи управління проєктами забезпечує кілька переваг. По-перше, розділення на окремі компоненти сприяє модульності та легшому тестуванню коду. По-друге, ізольована природа компонентів дозволяє розробникам працювати паралельно над різними частинами системи. По-третє, MVC спрощує повторне використання коду та підтримку, оскільки зміни в одному компоненті мають мінімальний вплив на інші частини системи.

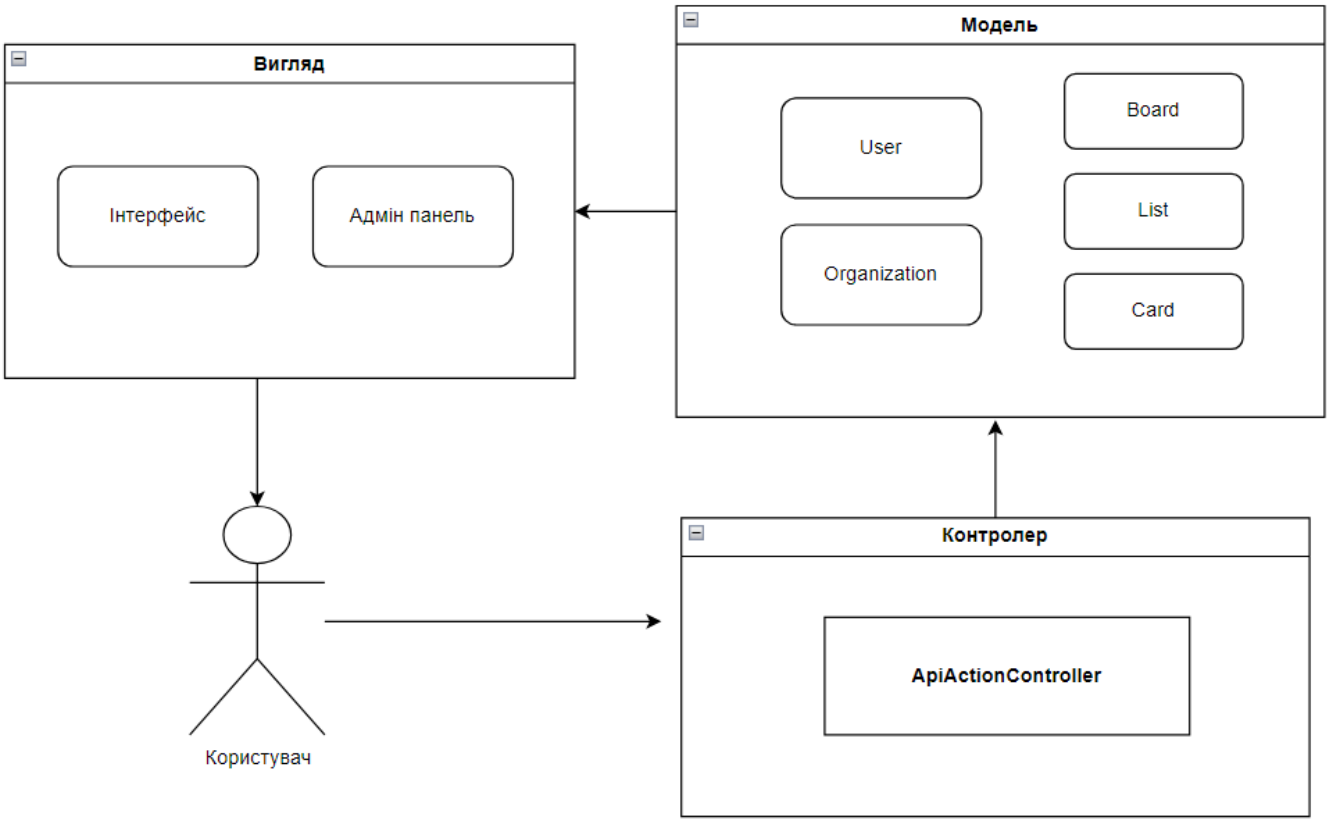


Рисунок 2.1 Шаблон проєктування MVC системи

Модель системи складається з основних класів User, Organization, Board, List, Card. Ці класи інкапсулюють дані та логіку обробки, пов'язані з відповідними сутностями системи. Контролер ApiActionController є центральною точкою системи і відповідальний за обробку усіх дій(actions), які може виконати користувач з усіма об’єктами системи. Всі можливі дії з основними об’єктами системи подані у Таблиці 12.

Таблиця 2.1 Список усіх дій(actions)

Дія (action)	HTTP-метод	Аргумент методу контролера
sign-up	POST	UserSignUpCredentialsType
sign-in	POST	UserSignInCredentialsType
sign-out	POST	-
update-user	PUT	UserInfoType
create-organization	POST	OrganizationInfoType
update-organization	PUT	OrganizationInfoType
invite-users	POST	UserInvitationsType
change-user-role	PATCH	UserRoleType
create-board	POST	BoardInfoType
update-board	PUT	BoardInfoType
delete-board	DELETE	BoardId
create-list	POST	ListInfoType
update-list	PUT	ListInfoType
update-list-order	PATCH	ListInfoType
copy-list	POST	ListInfoType
delete-list	DELETE	ListId
create-card	POST	CardInfoType
update-card	PUT	CardInfoType
update-card-order	PATCH	CardInfoType
copy-card	POST	CardInfoType
delete-card	DELETE	CardId

Переглянути приклад коду дії(action) можна в Додатку А, за цим же принципом виконані і інші дії.

2.2 Концептуальна схема системи

Концептуальна схема є високорівневим уявленням системи, яке візуалізує основні компоненти та їх взаємозв'язки без деталізації реалізації. Вона допомагає зрозуміти загальну структуру системи та взаємодію її частин на концептуальному рівні [27].

Концептуальна схема відіграє важливу роль у процесі проєктування та комунікації в команді розробників. Вона дозволяє представити архітектуру системи в узагальненому вигляді, абстрагуючись від технічних деталей реалізації. Це полегшує обговорення та розуміння високорівневих концепцій, підходів та принципів, які лежать в основі системи. Наприклад, при розробці системи управління проєктами, концептуальна схема може допомогти зрозуміти, як користувачі будуть взаємодіяти з дошками, картками та списками, які є ключовими елементами системи. Вона також дозволяє визначити основні потоки даних та взаємодії між різними компонентами системи.

Крім того, концептуальна схема допомагає виявити потенційні проблеми або недоліки в архітектурі на ранніх стадіях проєктування, коли їх легше вирішити. Це сприяє ефективній комунікації між різними зацікавленими сторонами, такими як архітектори, розробники та бізнес-аналітики, забезпечуючи спільне розуміння загальної картини. Виявлені проблеми можуть включати можливі вузькі місця в продуктивності, непередбачувані взаємозалежності між компонентами або можливі конфлікти в даних. У результаті, концептуальна схема допомагає створити більш надійну і добре спроєктовану систему, що відповідає вимогам бізнесу та користувачів, і є основою для подальшої деталізації та реалізації проєкту.

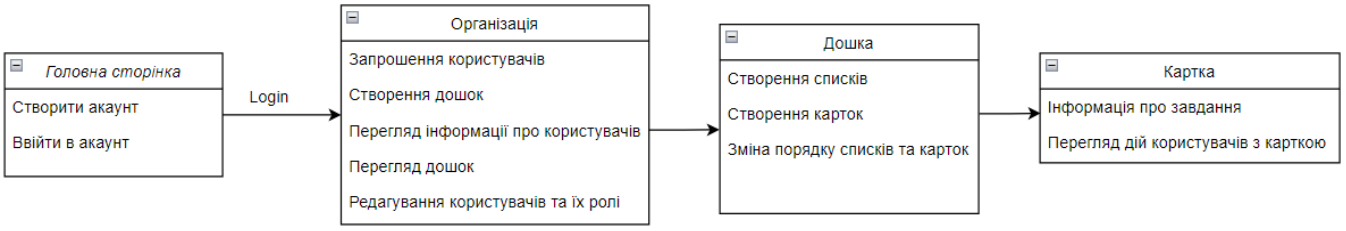


Рисунок 2.2. Концептуальна схема системи

2.3 Функціональна схема системи

Функціональна блок-схема потоку (Functional Flow Block Diagram, FFBD) – це графічне представлення функціональних потоків у системі або процесі. Вона використовується для візуалізації послідовності операцій, завдань або кроків, необхідних для виконання певної функції чи досягнення кінцевої мети. FFBD дозволяє систематично представити кожен етап процесу, показуючи логічні зв'язки між різними функціями та забезпечуючи ясність щодо того, як одна функція впливає на іншу. Це особливо корисно для складних систем, де важливо розуміти взаємозалежність і послідовність дій [27].

Один з основних принципів FFBD полягає в поділі системи або процесу на окремі функції, які розташовані в певному порядку. Кожна функція зображується у вигляді прямокутника, а потоки між ними вказуються стрілками, що відображають напрямок і послідовність виконання. Цей підхід дозволяє визначити всі можливі сценарії, альтернативні шляхи та рішення, які можуть бути прийняті в різних умовах. Завдяки цьому, FFBD допомагає ідентифікувати критичні точки процесу та області, де можуть виникати проблеми або необхідні покращення.

FFBD також корисна для комунікації між різними учасниками проекту. Вона надає зрозумілий і детальний огляд процесу, який може бути легко зрозумілий як технічними фахівцями, так і нетехнічними зацікавленими сторонами. Це сприяє кращій координації та співпраці в команді, оскільки всі учасники мають спільне уявлення про процеси та їх послідовність. Використання FFBD може допомогти уникнути непорозумінь і помилок, що виникають через різні інтерпретації вимог або процесів.

Крім того, FFBD є важливим інструментом для управління ризиками та оптимізації процесів. Вона дозволяє проводити аналіз ідентифікації вузьких місць та можливих затримок у системі, а також оцінювати вплив змін на загальну продуктивність. Аналіз функціональних потоків допомагає знаходити можливості для покращення процесів, зменшення витрат і підвищення ефективності. Таким чином, FFBD є потужним засобом для планування, аналізу та покращення систем і

процесів, забезпечуючи структурований підхід до управління та оптимізації операцій.

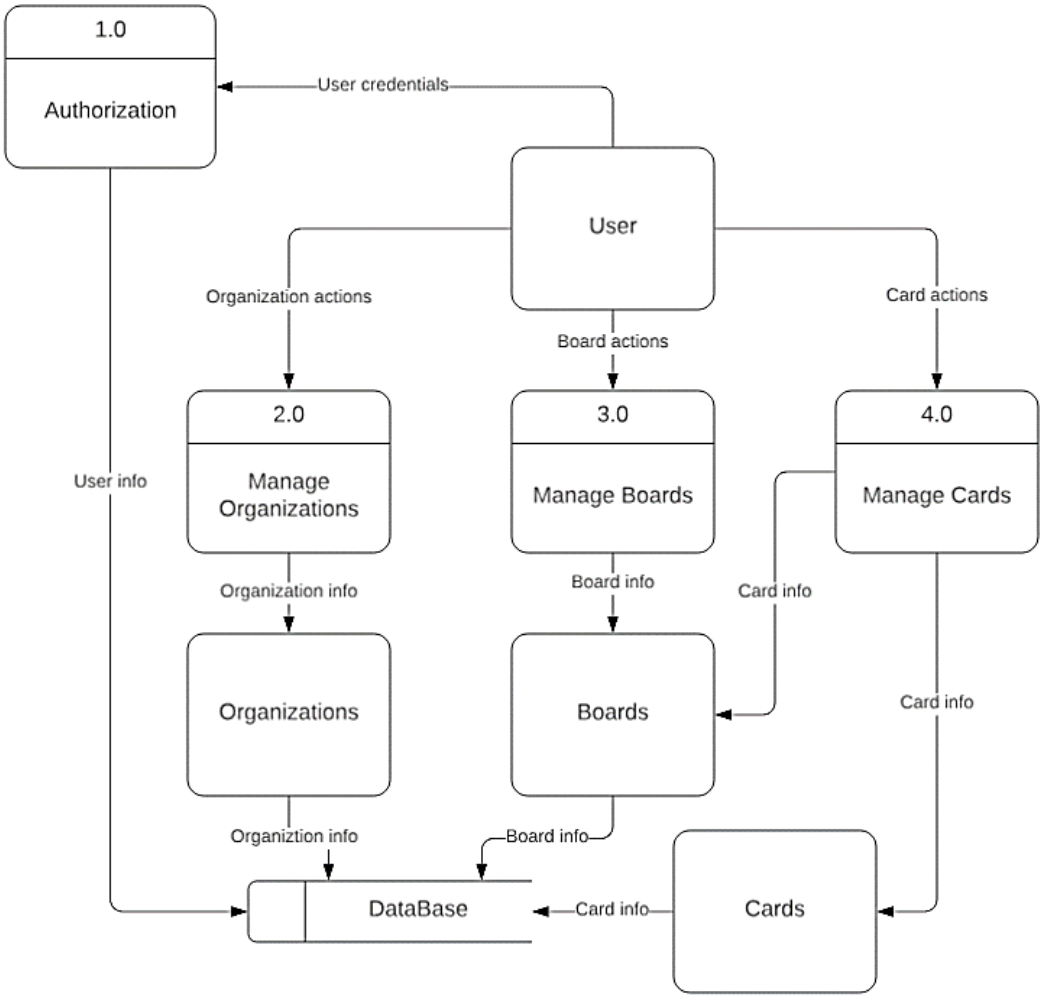


Рисунок 2.3. FFBD системи

3. ПРОЄКТУВАННЯ БАЗИ ДАНИХ.

3.1 Опис вибору СУБД

База даних - це фундаментальний компонент сучасних інформаційних систем, який відіграє ключову роль у зберіганні, організації та управлінні даними. За своєю суттю, база даних - це структурована колекція інформації, що зберігається в електронному форматі, дозволяючи ефективний доступ, маніпулювання та оновлення даних. Від простих списків контактів у смартфонах до складних систем аналізу даних у великих корпораціях, бази даних є невід'ємною частиною нашого цифрового життя [31].

Структурно база даних складається з таблиць, які є її основними будівельними блоками. Кожна таблиця представляє певну сутність або концепцію, наприклад, "Клієнти" або "Замовлення". Таблиці організовані в рядки та стовпці, де кожен рядок (також відомий як запис) представляє окрему сутність, а стовпці (або поля) визначають різні атрибути цієї сутності. Наприклад, у таблиці "Клієнти" рядок може представляти окремого клієнта, а стовпці - його ім'я, адресу, номер телефону та інші особисті дані.

Існує кілька типів баз даних, кожен з яких призначений для задоволення різних потреб. Найпоширенішими є реляційні бази даних, які використовують структуровану мову запитів (SQL) для управління даними. Вони базуються на понятті відносин між таблицями, де дані в одній таблиці можуть бути пов'язані з даними в іншій через спільні поля. Наприклад, таблиця "Замовлення" може бути пов'язана з таблицею "Клієнти" через ідентифікатор клієнта. З іншого боку, нереляційні (NoSQL) бази даних пропонують більшу гнучкість у структурі даних, що робить їх ідеальними для роботи з неструктурованими даними, такими як документи, графи або пари ключ-значення.

Центральним елементом системи бази даних є Система Управління Базами Даних (СУБД). Це потужне програмне забезпечення, яке діє як інтерфейс між базою даних і її користувачами, забезпечуючи інструменти для визначення структури даних, маніпулювання даними та керування доступом. СУБД відповідає

за виконання запитів, підтримку цілісності даних, обробку транзакцій та забезпечення безпеки. Популярні СУБД включають MySQL, PostgreSQL, Oracle для реляційних баз даних, і MongoDB, Cassandra для нереляційних.

Ключовими концепціями в теорії баз даних є поняття ключів і відносин. Первинний ключ - це унікальний ідентифікатор для кожного рядка в таблиці, наприклад, ідентифікаційний номер клієнта. Зовнішні ключі використовуються для встановлення зв'язків між таблицями, дозволяючи створювати складні відносини та забезпечуючи цілісність даних. Нормалізація - це процес організації даних для мінімізації надмірності та залежностей, що робить базу даних більш ефективною та легшою для обслуговування.

Переваги використання баз даних численні і значущі. Вони забезпечують високий рівень цілісності даних, гарантуючи, що інформація залишається точною та несуперечливою. Механізми безпеки в СУБД дають тонкий контроль над тим, хто може отримувати доступ до даних і що вони можуть з ними робити. Ефективність - ще одна ключова перевага, оскільки бази даних оптимізовані для швидкого пошуку, вибірки та оновлення даних, навіть у великих наборах. Більше того, бази даних масштабовані, здатні рости разом із потребами організації, і підтримують одночасний доступ, дозволяючи багатьом користувачам працювати з даними одночасно.

Сфери застосування баз даних охоплюють практично всі галузі. У бізнесі вони лежать в основі систем управління відносинами з клієнтами (CRM) та планування ресурсів підприємства (ERP). Інтернет-магазини та соціальні мережі спираються на бази даних для управління каталогами продуктів, профілями користувачів та взаємодіями. У науці бази даних зберігають величезні обсяги геномних даних або астрономічних спостережень. Медична галузь використовує їх для безпечного зберігання історій хвороб пацієнтів та сприяння клінічним дослідженням. З ростом Інтернету речей (IoT) бази даних стають критично важливими для обробки потоків даних, що надходять від мільярдів підключених пристроїв.

Однак разом із цими можливостями постають і значні виклики. Ера великих даних вимагає від баз даних обробки безпрецедентних обсягів різноманітних даних, що генеруються з високою швидкістю. Питання безпеки стають дедалі складнішими, оскільки кіберзлочинці постійно шукають вразливості. Підтримка високої продуктивності при зростанні навантаження є постійною проблемою, що вимагає ретельної оптимізації. Крім того, у світі, де дані часто розподілені між різними системами, інтеграція різнорідних джерел даних стає дедалі важливішою.

Отже, база даних - це набагато більше, ніж просто сховище інформації. Це потужний, гнучкий і розумний інструмент, який дозволяє організаціям ефективно зберігати, організовувати та отримувати цінну інформацію зі своїх даних. У міру того як світ стає дедалі більш залежним від даних, роль баз даних лише зростатиме, роблячи їх невід'ємною частиною нашого цифрового майбутнього.

Система управління базами даних (СУБД) - це складний програмний комплекс, який займає центральне місце в інфраструктурі сучасних інформаційних систем. За своєю суттю, СУБД є потужним інструментом, що об'єднує програмні та лінгвістичні засоби, призначені для створення, зберігання, управління та використання баз даних. Ці системи виходять далеко за межі простого зберігання інформації; вони забезпечують ефективну організацію даних, гарантують їхню цілісність та безпеку, а також надають користувачам та розробникам зручні та гнучкі інтерфейси для взаємодії з даними [29].

Роль СУБД в екосистемі управління даними неможливо переоцінити. У світі, де інформація стає найціннішим ресурсом, СУБД виступає як зберігач і розпорядник цього багатства. Її основна місія - забезпечити структурований доступ до інформації, що дозволяє швидко знаходити та обробляти потрібні дані серед величезних масивів. Але функції СУБД цим не обмежуються. Вона також відповідає за мінімізацію дублювання даних, що економить місце для зберігання і запобігає неузгодженості. Крім того, СУБД стоїть на сторожі цілісності даних, гарантуючи, що вони залишаються точними та несуперечливими. В епоху, коли кібербезпека стає критичною, СУБД також забезпечує надійний захист даних від несанкціонованого доступу та зловживань.

Історія СУБД - це історія еволюції нашого розуміння того, як найкраще організувати та використовувати інформацію. У 1960-х роках, на зорі цифрової ери, перші СУБД відображали наше тодішнє бачення структури даних. Вони були розроблені для зберігання інформації в ієрархічній або мережевій формі, імітуючи організаційні схеми та соціальні мережі того часу. Однак справжній прорив стався в 1970-х роках з появою реляційних СУБД. Засновані на чіткій математичній теорії, ці системи дозволили представляти дані у вигляді простих таблиць зі зв'язками між ними. Цей підхід виявився настільки потужним і гнучким, що реляційні СУБД досі домінують у галузі.

У ті ж 1970-ті роки відбулася ще одна революція - розробка мови структурованих запитів SQL. Ця мова стала універсальним інструментом для взаємодії з базами даних, дозволяючи користувачам та програмістам формулювати складні запити простою, майже розмовною мовою. У 1980-х роках, з розвитком об'єктно-орієнтованого програмування, з'явилися об'єктно-орієнтовані СУБД. Ці системи краще підходили для роботи зі складними об'єктами, такими як мультимедійні файли або геопросторові дані, які не вкладалися в традиційні табличні структури.

Останнє десятиліття ознаменувалося бурхливим розвитком хмарних технологій, і СУБД не стали винятком. Хмарні СУБД, які надають доступ до баз даних через Інтернет, змінюють правила гри. Вони пропонують безпрецедентну масштабованість і гнучкість, дозволяючи організаціям збільшувати або зменшувати ресурси бази даних на вимогу. Більше того, вони звільняють компанії від тягаря управління серверною інфраструктурою, даючи змогу зосередитися на своїй основній діяльності.

В сучасному світі, де дані лежать в основі майже кожного бізнес-процесу, СУБД виконує функції, що виходять далеко за рамки простого зберігання інформації. Розглянемо, наприклад, великий онлайн-магазин. Тут СУБД не просто зберігає дані про товари, замовлення та клієнтів; вона організовує цю інформацію так, щоб кожен запит - чи то пошук товару, чи оформлення замовлення -

виконувався миттєво. Це досягається за допомогою складних індексів і оптимізації запитів, які є ключовими інструментами СУБД.

Але робота СУБД не обмежується швидким отриманням даних. За допомогою інструментів вилучення та оновлення вона дає змогу динамічно управляти інформацією. Коли клієнт оновлює свою адресу доставки або коли товар переміщується зі складу, ці зміни негайно відображаються в базі даних. Більше того, СУБД гарантує, що такі оновлення не порушать цілісність даних. Наприклад, вона не дозволить видалити запис про клієнта, якщо у нього є активні замовлення. Різноманітність сучасних задач обробки даних призвела до появи широкого спектру СУБД, кожна з яких оптимізована під певні потреби. За моделлю даних СУБД поділяються на реляційні (використовують таблиці), ієрархічні (дані у вигляді дерева) та мережеві (складні взаємозв'язки між даними). Вибір моделі залежить від характеру даних і типу запитів.

Функціональне призначення також визначає тип СУБД. Оперативні СУБД оптимізовані для швидкого виконання транзакцій в режимі реального часу, наприклад, в банківських системах. Аналітичні СУБД, навпаки, призначені для обробки складних запитів над великими обсягами історичних даних, що є типовим для бізнес-аналітики.

За розподілом даних СУБД бувають централізованими, де вся інформація зберігається на одному сервері, і розподіленими, де дані розподілені між кількома серверами. Розподілені СУБД краще справляються з високими навантаженнями і стійкіші до відмов. Ще один критерій - тип використання: SQL-орієнтовані СУБД слідують стандартам мови SQL, тоді як NoSQL пропонують більш гнучкі моделі даних.

У підсумку, Система управління базами даних - це не просто сховище даних, а інтелектуальний інструмент, що забезпечує ефективне збереження, організацію та використання інформації.

Розглянемо основні СУБД за моделлю даних. Розуміння переваг і недоліків кожного типу допомагає вибрати найкращий варіант залежно від конкретних потреб проєкту.

Таблиця 3.1 Порівняння типів СУБД

Тип СУБД	Опис	Переваги	Недоліки
Ієрархічні	Організовують дані у вигляді ієрархії, де кожен запис має одного “батька” і багато “нащадків”.	– Простота в моделюванні зв’язків. – Ефективний доступ до деревоподібних структур даних.	– Обмеження в гнучкості структури даних. – Утруднення у вираженні складних відносин.
Мережеві	Дають змогу встановлювати складні зв’язки між даними, створюючи структуру, схожу на граф.	– Підтримка складних і взаємопов’язаних даних. – Гнучкіша, ніж ієрархічні, щодо зв’язків. – Сприяють поданню складних бізнес-процесів.	– Складність в управлінні структурою і підтримці запитів. – Складнощі в підтримці та обслуговуванні.
Реляційні	Засновані на теорії відносин і таблиць. Кожна таблиця представляє окреме відношення.	– Простота у використанні та моделюванні даних. – Незалежність даних від фізичної структури зберігання.	– Продуктивність може страждати за великих обсягів даних. – Складність у поданні складних ієрархій даних.
Об’єктно-орієнтовані	Працюють з об’єктами, де дані представлено у вигляді об’єктів із методами та властивостями.	– Зручність у моделюванні складних об’єктних структур. – Підтримка успадкування, поліморфізму та інкапсуляції.	– Складність у міграції з традиційних СУБД. – Обмежена підтримка мови SQL.
Об’єктно-реляційні	Комбінують риси реляційних і об’єктно-орієнтованих СУБД.	– Поєднання переваг реляційних і об’єктно-орієнтованих моделей. – Гнучкість у роботі з об’єктами	– Додаткова складність у проєктуванні. – Можливі проблеми з продуктивністю через додаткові шари абстракції.

Розподілені бази даних – це важливий тренд, де дані логічно сприймаються як єдине ціле, навіть якщо вони фізично розкидані по різних комп’ютерах. Основні характеристики цього тренду включають горизонтальне масштабування, глобальну реплікацію і підвищену консистентність. Всі ці аспекти спрямовані на ефективне використання систем управління базами даних і управління великими обсягами даних.

Нові технології також впливають на еволюцію СУБД. У сфері інтернету речей (IoT), системи управління базами даних стикаються з потоком даних від безлічі пристроїв, вимагаючи нових методів обробки даних у реальному часі. Технології блокчейна, своєю чергою, можуть використовуватися для забезпечення безперервності даних і підвищення безпеки в базах даних.

СУБД надають різноманітні заходи безпеки, включно з контролем доступу, шифруванням даних, підтриманням цілісності даних і резервним копіюванням. Контроль доступу обмежує доступ до даних, шифрування робить дані непридатними для несанкціонованих користувачів, а підтримання цілісності та резервне копіювання – забезпечують надійність даних.

Вибір правильної системи управління базами даних відіграє ключову роль в ефективності та стійкості проєкту. Потрібно враховувати тип даних, вимоги проєкту, продуктивність, масштабованість і зручність використання. Оцінітювати швидкість виконання запитів, індексування, масштабованість вертикального і горизонтального типу, сумісність та інтеграцію з інструментами розробки. Також потірбно не забувати про рівень складності адміністрування.

PostgreSQL - це потужна, відкрита об'єктно-реляційна система управління базами даних (СУБД), яка має багату історію розвитку протягом більш ніж 30 років. Вона відома своєю надійністю, масштабованістю та здатністю обробляти великі обсяги даних, що робить її відмінним вибором для багатьох сучасних веб-застосунків, включаючи онлайн-системи управління проєктами [24].

Однією з ключових переваг PostgreSQL є її повна відповідність стандартам ACID (Atomicity, Consistency, Isolation, Durability). Це означає, що наша онлайн-система управління проєктами завжди матиме цілісні та надійні дані, навіть

у випадку збоїв системи або одночасного доступу багатьох користувачів. Коли команди одночасно оновлюють статуси завдань, додають коментарі або змінюють терміни, PostgreSQL гарантує, що всі ці операції будуть виконані точно і без конфліктів.

PostgreSQL також пропонує багатий набір функцій для роботи зі складними запитами та маніпуляціями з даними. Для нашої системи управління проєктами це означає можливість легко створювати складні звіти та аналітику. Наприклад, ми можемо використовувати window functions для аналізу прогресу проєкту в часі, JSON типи даних для зберігання гнучких структур метаданих проєктів, або full-text search для швидкого пошуку в описах завдань і документах.

Масштабованість - ще одна сильна сторона PostgreSQL. З ростом нашої бази користувачів і збільшенням кількості проєктів, вам знадобиться база даних, яка може легко адаптуватися. PostgreSQL чудово справляється з цим завдяконням своїй архітектурі, що дозволяє ефективно використовувати багатоядерні процесори та розподілені системи. Вона також підтримує паралельне виконання запитів, що прискорює обробку складних звітів, які часто потрібні в системах управління проєктами.

Безпека даних є критично важливою для будь-якої онлайн-системи, особливо тієї, що містить конфіденційну інформацію про проєкти. PostgreSQL пропонує розширені функції безпеки, такі як аутентифікація на основі ролей, шифрування на рівні колонок і детальне управління доступом. Це дозволяє нам точно контролювати, хто може переглядати, редагувати або видаляти певні частини даних проєкту, що є ключовим для збереження конфіденційності в корпоративному середовищі.

У світі управління проєктами часто потрібні спеціалізовані типи даних, такі як дати, часові діапазони для планування, географічні дані для розподілених команд, і навіть користувацькі типи для унікальних потреб бізнесу. PostgreSQL відмінно підходить для цього завдання, пропонуючи широкий спектр вбудованих типів даних і можливість створювати власні. Це дозволяє нам моделювати дані проєкту найбільш природним і ефективним способом.

PostgreSQL має велику і активну спільноту, що забезпечує чудову документацію, численні розширення та швидке виправлення будь-яких виявлених проблем. Це означає, що під час розробки системи управління проєктами ми матимемо доступ до багатьох ресурсів і зможемо швидко знайти рішення будь-яких технічних проблем.

У підсумку, PostgreSQL - це надійна, функціонально багата і високопродуктивна база даних, яка ідеально підходить для онлайн-системи управління проєктами. Вона забезпечить цілісність даних, підтримає складні запити для аналітики, масштабуватиметься разом з бізнесом і захистить важливу інформацію про проєкти. З PostgreSQL ми отримаємо міцний фундамент для створення потужної та гнучкої системи, здатної задовольнити всі наші потреби в управлінні проєктами.

Таблиця 3.2 Порівняльний аналіз PostgreSQL з іншими СУБД

Критерій	**PostgreSQL**	**MySQL**	**Oracle**
Тип ліцензії	Open source	Open source	Proprietary
Вартість	Безкоштовно	Безкоштовно	Платна
Відкритість	Відкритий вихідний код	Відкритий вихідний код	Закритий вихідний код
Підтримувані мови програмування	SQL, PL/pgSQL, Python, Perl, Java, C, C++, Ruby	SQL, MySQL, PHP, Python, Perl	SQL, PL/SQL, Java, C, C++, C#, Python, Ruby
Підтримувані операційні системи	Windows, Linux, macOS, FreeBSD	Windows, Linux, macOS	Windows, Linux, macOS, Solaris, HP-UX, AIX, z/OS
Підтримувані архітектури	x86, x86-64, ARM, RISC-V	x86, x86-64, ARM	x86, x86-64, ARM, RISC-V
Продуктивність	Хороша	Хороша	Відмінна
Надійність	Висока	Висока	Висока
Безпека	Висока	Хороша	Відмінна

Продовження таблиці 3.2 Порівняльний аналіз PostgreSQL з іншими СУБД

Функціональність	Широкий набір функцій	Широкий набір функцій	Широкий набір функцій
Масштабованість	Хороша	Хороша	Відмінна
Гнучкість	Висока	Висока	Висока
Підтримка	Активна	Активна	Активна

Ці переваги в сукупності створюють потужну і надійну платформу для управління даними. Те, до якого виду СУБД належить PostgreSQL, плюс те, що відкритий вихідний код надає свободу налаштування та адаптації, підтримка різноманітних функцій – усе це забезпечує гнучкість у реалізації різних сценаріїв використання. Висока продуктивність, надійність і рівень безпеки – забезпечують ефективну і захищену роботу з даними, що робить PostgreSQL привабливим вибором для онлайн-системи управління проєктами.

3.2 Опис всіх таблиць та полів бази даних

Діаграма сутностей-відносин (ERD) - це візуальне представлення основних сутностей у системі реляційної бази даних, а також їхніх атрибутів і зв'язків між ними. Основна мета ERD - забезпечити чіткий і детальний аналіз схеми бази даних, яка складається з таблиць, полів і зв'язків між таблицями. ERD, також часто званий моделлю «сутність-зв'язок», використовується під час проєктування баз даних, системного аналізу та розроблення програмного забезпечення для визначення, візуалізації та документування структури й організації бази даних [28].

ERD були вперше представлені в 1970-х роках доктором Пітером Ченом, піонером у галузі інформатики. Основна мета Чена полягала в тому, щоб надати простий, зрозумілий і стандартизований метод фіксації та ілюстрації складних взаємозв'язків усередині системи бази даних. ERD складаються з декількох компонентів, включно із сутностями, атрибутами та відношеннями:

Сутності - це основні об'єкти, такі як людина, місце або річ, навколо яких

будується база даних. В ERD вони представлені у вигляді прямокутників, і кожен об'єкт має унікальне ім'я, що дає змогу відрізняти його від інших об'єктів у системі.

Атрибути - це властивості або характеристики об'єктів, наприклад ім'я або вік людини. В ERD вони зображуються овалами і з'єднуються з відповідною лінією.

Відносини являють собою зв'язки між сутностями і зазвичай позначаються ромбовидною формою в ERD. Відносини описують, як сутності зв'язуються одна з одною, наприклад, відносини «один до одного», «один до багатьох» або «багато до багатьох».

У контексті розроблення програмного забезпечення ERD відіграють важливу роль на етапі проєктування і планування, надаючи наочний посібник, який дає змогу розробникам, зацікавленим сторонам і користувачам зрозуміти різні відносини між сутностями та їхніми атрибутами. Цей схематичний огляд схеми даних допомагає виявити потенційні проблеми та області для оптимізації, що особливо корисно під час розроблення складних застосунків, які передбачають великі завдання управління даними. Крім того, ERD можуть слугувати документацією для адміністраторів баз даних та інших членів команди, які беруть участь в обслуговуванні та постійному розвитку системи.

Ключовою перевагою ERD у контексті реляційних баз даних є їхня тісна відповідність зі структурою самих баз даних. Реляційні бази даних засновані на таблицях, у яких зберігаються структуровані дані - таблиця для кожного об'єкта, а відносини між об'єктами подано через обмеження первинного і зовнішнього ключів. ERD надають простий та інтуїтивно зрозумілий спосіб представлення цих ключових елементів та їхніх взаємозв'язків, сприяючи плавному переходу між проєктуванням, впровадженням та обслуговуванням бази даних.

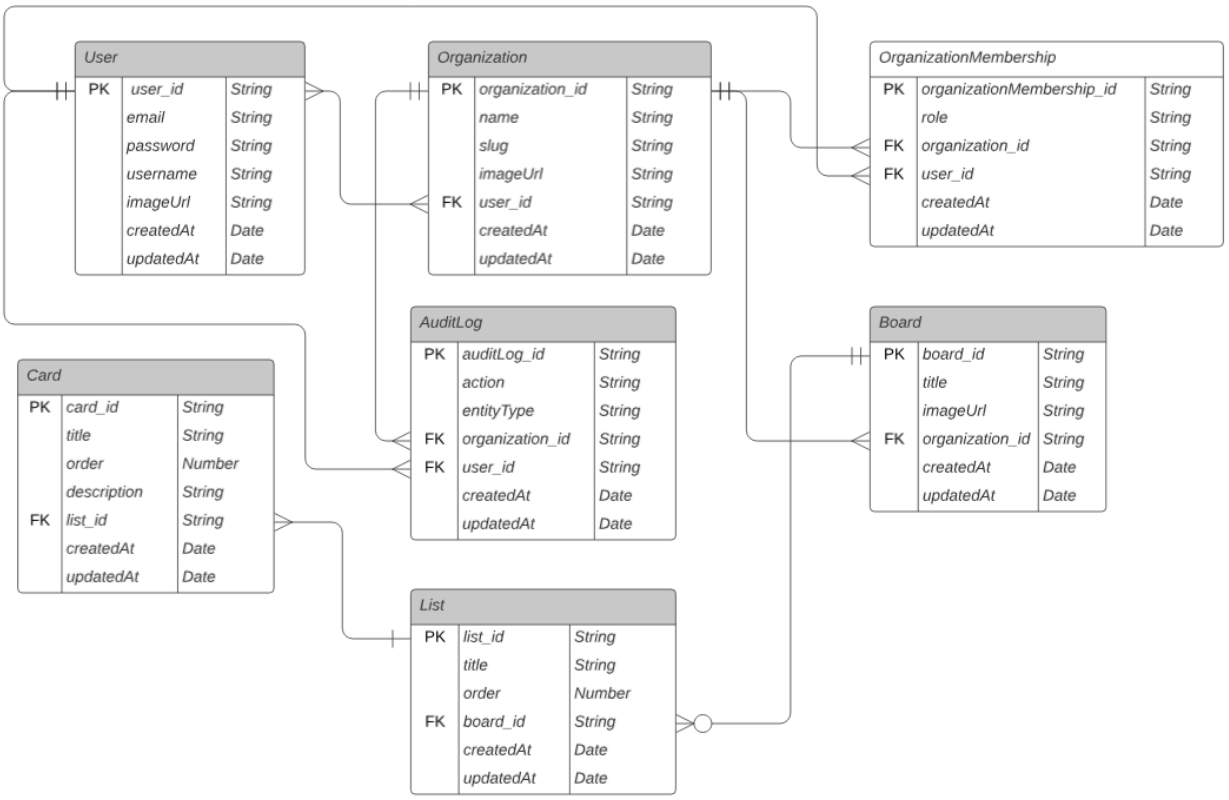


Рисунок 3.1. ERD бази даних

Таблиця 3.3. Опис таблиць та полів БД

Таблиця	Назва поля	Тип	Опис поля
User	user_id	String	PK, для зберігання ідентифікатора
	email	String	Пошта, по якій користувач авторизується в акаунт

Продовження таблиці 3.3 Опис таблиць та полів БД

	password	String	Пароль користувача в акаунт
	username	String	Ім’я користувача
	imageUrl	String	Посилання на аватар користувача
	createdAt	Date	Дата створення

	updatedAt	Date	Дата оновлення
Organization	organization_id	String	PK, для зберігання ідентифікатора
	name	String	Назва організації
	slug	String	Скорочення назви організації
	imageUrl	String	Посилання на зображення організації
	user_id	String	FK, Посилання на користувачів, які є в організації
	createdAt	Date	Дата створення
	updatedAt	Date	Дата оновлення
Organization Membership	organizationMembership_id	String	PK, для зберігання ідентифікатора

Продовження таблиці 3.3 Опис таблиць та полів БД

	role	String	Роль користувача в організації
	organization_id	String	FK, посилання на організацію, до якої приєднанний користувач
	user_id	String	FK, посилання на користувача
	createdAt	Date	Дата створення
	updatedAt	Date	Дата оновлення

Board	board_id	String	PK, для зберігання ідентифікатора
	title	String	Назва дошки
	imageUrl	String	Посилання на зображення, яке буде на задньому фоні дошки
	organization_id	String	FK, посилання на організацію, до якої належить дошка
	createdAt	Date	Дата створення
	updatedAt	Date	Дата оновлення
List	list_id	String	PK, для зберігання ідентифікатора

Продовження таблиці 3.3 Опис таблиць та полів БД

	title	String	Назва списку
	order	Number	Порядок списку в дошці
	board_id	String	FK, посилання на дошку, до якої належить список
	createdAt	Date	Дата створення
	updatedAt	Date	Дата оновлення
Card	card_id	String	PK, для зберігання ідентифікатора
	title	String	Назва картки

	order	String	Порядок картки в списку
	description	String	Опис картки
	list_id	String	FK, посилання на список, до якої належить картка
	createdAt	Date	Дата створення
	updatedAt	Date	Дата оновлення
AuditLog	auditLog_id	String	PK, для зберігання ідентифікатора
	action	String	Опис про дію, яка відбулася
	entityType	String	Тип дії

Продовження таблиці 3.3 Опис таблиць та полів БД

	organization_id	String	FK, посилання на організацію, до якої належить дія
	user_id	String	FK, посилання на користувача, який виконав дію
	createdAt	Date	Дата створення
	updatedAt	Date	Дата оновлення

–Таблиця "User" містить повну інформацію про зареєстрованих користувачів системи, включаючи їхні облікові дані, персональні відомості та налаштування.

–Таблиця "Organization" зберігає дані про організації, які можуть створювати користувачі. Організації призначені для об'єднання учасників для спільної роботи над проєктами, створення дошок та управління завданнями.

–Таблиця "OrganizationMembership" відображає інформацію про членство користувачів в організаціях, включаючи їхні ролі та дозволи.

–Таблиця "Board" містить відомості про дошки проєктів, які створюються в рамках організацій для управління завданнями через списки та картки.

–Таблиця "List" зберігає дані про списки на дошках, які використовуються для відстеження статусу виконання завдань (карток).

–Таблиця "Card" містить повну інформацію про картки (завдання) на дошках проєктів;

–Таблиця "AuditLog" фіксує записи про дії користувачів в організаціях, такі як створення дошок, зміна статусів завдань, створення або редагування завдань та інші важливі події для відстеження активності та аудиту.

4. ПРОЄКТНІ РІШЕННЯ ТА ТЕСТОВИЙ ПРИКЛАД

4.1 Вибір середовища розробки

Середовище розробки (IDE - Integrated Development Environment) - це потужний програмний пакет, який об'єднує в собі різні інструменти та функції, призначені для полегшення процесу написання, тестування, налагодження та розгортання програмного забезпечення. Воно зазвичай включає в себе редактор коду, компілятор чи інтерпретатор, засоби автодоповнення, підсвічування синтаксису, відлагоджувач, інтеграцію з системами контролю версій, інструменти для автоматизації збірки та розгортання, а також багато інших корисних функцій, що допомагають програмістам ефективно розробляти програми.

Середовище розробки допомагає підвищити продуктивність і полегшити роботу програмістів шляхом об'єднання всіх необхідних інструментів в одному місці. Воно забезпечує зручний інтерфейс для написання та редагування коду, дозволяє швидко знаходити та виправляти помилки за допомогою відлагоджувача, автоматизує процеси збірки та розгортання, полегшує співпрацю в команді завдяки інтеграції з системами контролю версій та допомагає дотримуватися кодових стандартів і кращих практик за допомогою різноманітних плагінів та налаштувань. В результаті, використання потужного середовища розробки може значно підвищити продуктивність програмістів та якість написаного коду.

Visual Studio Code (VS Code) - це безкоштовний, крос-платформений редактор коду з відкритим вихідним кодом, розроблений Microsoft. Він є потужним і гнучким інструментом для розробки програмного забезпечення, орієнтованим на продуктивність і зручність використання. VS Code підтримує широкий спектр мов програмування, включаючи JavaScript, TypeScript, Python, Java, C/C++, C#, Go, Ruby та багато інших, і має багату екосистему розширень, створених як офіційною командою розробників, так і спільнотою.

VS Code пропонує інтуїтивний та легко налаштовуваний інтерфейс користувача, інтелектуальне автодоповнення коду, вбудовану підтримку Git, потужний дебагер, інтеграцію з системами збірки та можливості для покращення

продуктивності, такі як розділення терміналу, передперегляд Markdown та багато іншого. Він також відомий своєю високою продуктивністю та здатністю ефективно працювати з великими проєктами. Завдяки безкоштовності, крос-платформеності, широкій підтримці мов програмування, активній спільноті розробників та постійному вдосконаленню, VS Code став одним із найпопулярніших середовищ розробки серед професіоналів та ентузіастів у всьому світі.

Таблиця 4.1. Порівняльний аналіз VS Code з іншими IDE

Функція	**Visual Studio Code**	**Atom**	**WebStorm**
Тип	Вільне, відкрите джерело	Вільне, відкрите джерело	Пропрієтарне (безкоштовна та платна версії)
Платформи	Windows, macOS, Linux	Windows, macOS, Linux	Windows, macOS, Linux
Мови програмування	TypeScript, JavaScript, Python, Java, C/C++, C#, Go, Ruby та багато інших	Підтримуються розширеннями	JavaScript, TypeScript
Інтеграція з Git	Вбудована	Вбудована	Вбудована
Дебагер	Вбудований	Розширення	Вбудований
Автодоповнення коду	Вбудоване	Розширення	Вбудоване

Продовження таблиці 4.1. Порівняльний аналіз VS Code з іншими IDE

Підсвічування синтаксису	Вбудоване	Вбудоване	Вбудоване
Інтеграція з системами збірки	Підтримуються розширеннями	Розширення	Вбудована підтримка
Підтримка плагінів/розширень	Велика кількість розширень	Велика кількість розширень	Велика кількість плагінів
Продуктивність	Висока	Середня	Висока
Зручність використання	Інтуїтивний користувацький інтерфейс	Складний користувацький інтерфейс	Потужний, але складний для початківців
Спільнота	Велика спільнота	Середня спільнота	Велика спільнота

Visual Studio Code є привабливим вибором середовища розробки для розробки онлайн-системи управління проєктами завдяки своїй безкоштовності, відкритому вихідному коду, крос-платформеності, широкій підтримці мов програмування, інтуїтивному та налаштовуваному інтерфейсу користувача, потужній екосистемі розширень, вбудованій підтримці Git, інтелектуальному автодоповненню коду, вбудованому дебагеру, високій продуктивності та активній спільноті розробників, що забезпечує постійний розвиток і підтримку цього універсального та зручного інструменту для різноманітних завдань розробки програмного забезпечення.

4.2 Опис проєктних рішень з програмного забезпечення

React — JavaScript-бібліотека, яка застосовується для розробки UI (призначених для користувача інтерфейсів). Це одна з найпопулярніших технологій для цієї мови, але не зовсім повноцінний фреймворк. React відповідає за інтерфейс програми, то як вона буде виглядати, а її інші компоненти пишуться за допомогою інших бібліотек та фреймворків, наприклад, Next.js. Він додає додатку гібридний статичний та серверний рендеринг, а також набір інших корисних функцій, що спрощують розробку [28].

Однак розробники визнають проблеми такого підходу: неможливість роботи веб застосунку при вимкненому JavaScript в браузері користувача або його відсутності, потенційні проблеми з безпекою, значне збільшення часу початкового завантаження сторінки, оскільки необхідно одразу отримати від сервера повний код застосунку та шкода для пошукової оптимізації сайту. Такі фреймворки, як Next.js, розв'язують ці проблеми, дозволяючи окремим частинам або всьому веб-сайту генеруватися на стороні сервера перед відправленням клієнту.

Next.js — це React фреймворк для створення сучасних веб-додатків, орієнтованих на продуктивність та SEO. Він був розроблений компанією Vercel (раніше Zeit) і базується на React, але додає додаткові функції та оптимізації.

Це один із найпопулярніших додатків до цієї JavaScript-бібліотеки. З його допомогою легко створювати продуктивні та оптимізовані для пошукових систем сайти, які сподобаються не тільки роботам краулер, але й користувачам [25].

Next.js максимально використовує можливості React, підтримує TypeScript та складну маршрутизацію, а також надає додаткові функції з коробки. З цим фреймворком можна створити web-додаток будь-якої складності з нуля або поступово модернізувати старий сайт — щоби він швидше завантажувався та стабільніше працював.

Основні переваги Next.js:

- Server-Side Rendering (SSR): Next.js за замовчуванням використовує серверний рендеринг, що означає, що початкове завантаження сторінки

відбувається на сервері, а не в браузері. Це забезпечує кращу швидкість завантаження, оптимізацію для пошукових систем (SEO) і кращий досвід користувача.

- Static Site Generation (SSG): Крім SSR, Next.js також підтримує статичний рендеринг сторінок під час збирання проєкту. Це дозволяє створювати швидкі, оптимізовані для SEO статичні сайти, які можна розгорнути на CDN.
- Клієнтський рендеринг: Next.js також підтримує традиційний клієнтський рендеринг для більш інтерактивних функцій, таких як одностраничні додатки (SPA).
- Автоматична оптимізація коду: Next.js автоматично оптимізує код, включаючи мініфікацію, розділення коду за модулями та гаряче оновлення модулів для покращення продуктивності.
- Маршрутизація на основі файлової системи: Next.js використовує файлову систему для визначення маршрутів додатка, що робить процес створення нових сторінок дуже простим.
- API Routes: Next.js дозволяє створювати API-маршрути безпосередньо в додатку, уникаючи необхідності налаштовувати окремий сервер.
- Підтримка статичних і динамічних імпортів: Next.js підтримує як статичний імпорт (для завантаження коду під час збирання), так і динамічний імпорт (для завантаження коду на вимогу).
- Вбудована підтримка CSS-in-JS: Next.js має вбудовану підтримку CSS-in-JS бібліотек, таких як styled-components та styled-jsx.
- Інструменти розробки: Next.js надає потужні інструменти розробки, включаючи швидкий перезапуск сервера, гаряче оновлення коду та корисні повідомлення про помилки.
- Інтеграція з React: Next.js тісно інтегрований з React, що дозволяє використовувати всі функції та бібліотеки React.

Next.js широко використовується для створення високопродуктивних веб-додатків, блогів, електронної комерції, портфоліо та багато іншого. Його підхід

до серверного рендерингу та статичної генерації сторінок робить його привабливим вибором для веб-сайтів та додатків, які потребують високої швидкості завантаження та оптимізації для пошукових систем.

Таблиця 4.2 Порівняння Next.js з іншими фреймворками

Функція	Next.js	Remix	Gatsby
Рендеринг	Серверний (SSR), Статичний (SSG), Клієнтський	Серверний (SSR), Статичний (SSG)	Статичний (SSG)
Маршрутизація	Файлова система	Файлова система	Графічний запит
Стилі	CSS-in-JS (styled-jsx, styled-components)	CSS-модулі, Vanilla CSS	CSS-модулі, Vanilla CSS
Кешування	Вбудоване кешування	Вбудоване кешування	Кешування через плагіни
Продуктивність	Висока, завдяки SSG, розділенню коду	Висока, завдяки SSR та вбудованому кешуванню	Висока, SSG та кешуванню
SEO	Висока, завдяки серверному рендерингу	Висока, завдяки серверному рендерингу	Висока, завдяки статичній генерації
Підтримка	Активна, від Vercel	Активна, від Remix	Активна, від Gatsby
API	API Routes	Remix Handlers	Використовує Node.js сервер
Мета	Загальна веб-розробка	Веб-розробка з акцентом на форми та мутації даних	Статичні сайти та проєкти контент-маркетингу

Усі три фреймворки пропонують серверний рендеринг та статичну генерацію для покращеної продуктивності та SEO. Проте, Next.js є більш універсальним рішенням, що охоплює всі ці сценарії, включаючи клієнтський рендеринг для одностраничних додатків (SPA). Він також пропонує більше можливостей для стилізації з CSS-in-JS.

ORM (Object-Relational Mapping) - це техніка, яка дозволяє працювати з реляційними базами даних за допомогою об'єктно-орієнтованих концепцій. Вона забезпечує абстракцію над базою даних, перетворюючи таблиці на класи, рядки - на об'єкти, а стовпці - на атрибути об'єктів. Замість написання SQL-запитів безпосередньо, розробники можуть використовувати об'єктно-орієнтовані конструкції, такі як класи, об'єкти та методи для взаємодії з базою даних [31].

ORM широко використовується в сучасному веб-розробленні та створенні додатків із кількох причин. По-перше, вона полегшує розробку, приховуючи деталі роботи з базами даних за об'єктною абстракцією, що робить код більш читабельним та легким для підтримки. По-друге, ORM забезпечує переносимість коду між різними системами управління базами даних, оскільки розробники взаємодіють з абстрактним рівнем, а не зі специфічними SQL-діалектами. По-третє, ORM часто пропонує додаткові функції, такі як кешування запитів, відстеження змін об'єктів та автоматичне згладжування відмінностей між об'єктно-орієнтованою та реляційною моделями даних. Загалом, ORM підвищує продуктивність розробки та полегшує підтримку додатків, що взаємодіють з базами даних.

Prisma - це сучасний інструмент для роботи з базами даних, який спрощує взаємодію між вашим Node.js або TypeScript додатком і базою даних. На відміну від традиційних ORM (Object-Relational Mapping), Prisma використовує інноваційний підхід, який робить роботу з базами даних більш інтуїтивною та безпечною [26].

Однією з ключових особливостей Prisma є його потужна система типізації. Ви визначаєте свою схему даних у файлі Prisma Schema Language (PSL), який потім компілюється в типізований клієнт Prisma. Це означає, що користувач отримуєте автоматичне доповнення коду та перевірку типів у вашому IDE, що значно зменшує кількість помилок під час розробки. Для системи управління проєктами це буде

особливо корисно, оскільки доведеться працювати зі складними структурами даних, такими як проєкти, завдання, користувачі та їхні взаємозв'язки.

Prisma також відрізняється своєю гнучкістю щодо баз даних. Він підтримує широкий спектр популярних баз даних, включаючи PostgreSQL, MySQL, SQLite та Microsoft SQL Server. Це означає, що ми можемо вибрати базу даних, яка найкраще відповідає нашим потребам - чи то високопродуктивний PostgreSQL для масштабної системи, чи то SQLite для швидкого прототипування. Крім того, Prisma полегшує міграцію між базами даних, якщо наші потреби зміняться в майбутньому.

Ще одна перевага Prisma - це його потужні можливості запитів. Замість написання складних SQL-запитів або використання заплутаного API запитів ORM, Prisma пропонує чистий, інтуїтивно зрозумілий API на основі JavaScript. Ми можемо легко фільтрувати, сортувати, об'єднувати та агрегувати дані за допомогою простих методів виклику.

Prisma також спрощує обробку відносин між даними. У системі управління проєктами у нас будуть відносини - проєкти мають багато завдань, завдання належать до проєктів і призначаються користувачам, користувачі можуть бути членами кількох проєктів. Prisma робить роботу з цими відносинами простою та інтуїтивно зрозумілою, дозволяючи нам легко витягувати пов'язані дані або створювати нові записи з відносинами в одному запиті.

У контексті безпеки Prisma також виділяється. Він використовує параметризовані запити за замовчуванням, що захищає нас від SQL-ін'єкцій. Більше того, Prisma надає детальний контроль доступу на рівні моделей, що дозволяє нам точно визначити, які користувачі можуть читати, створювати, оновлювати або видаляти певні типи даних. Це критично важливо для системи управління проєктами, де різні користувачі (менеджери, розробники, клієнти) повинні мати різні рівні доступу.

Prisma чудово працює з сучасними архітектурами, такими як serverless та мікросервіси. Якщо ми вирішимо побудувати вашу систему управління проєктами як набір невеликих, незалежних сервісів (наприклад, сервіс для управління завданнями, інший для управління користувачами), Prisma легко інтегрується в

кожен з них. Його швидке завантаження та ефективне використання з'єднань з базою даних роблять його відмінним вибором для serverless функцій, які часто запускаються з холодного старту.

Отже, для нашої онлайн-системи управління проєктами Prisma пропонує потужне, безпечне та гнучке рішення для роботи з базами даних. Його сильна типізація, інтуїтивний API запитів та відмінна підтримка відносин ідеально підходять для складних даних, з якими ми працюватимемо. Крім того, його акцент на безпеці та сумісність з сучасними архітектурами гарантують, що наша система буде не лише функціональною, але й безпечною та масштабованою.

Таблиця 4.3 Порівняння Prisma з іншими ORM

Особливість	**Prisma**	**Sequelize**	**TypeORM**
Тип інструменту	ORM нового покоління	Традиційний ORM	Традиційний ORM
Основна мова	TypeScript	JavaScript	TypeScript
Типізація	Сильна, автогенерована	Слабка	Сильна, ручна
Визначення схеми	Prisma Schema Language (PSL)	JavaScript	TypeScript / JavaScript / Декоратори
Підтримувані БД	PostgreSQL, MySQL, SQLite, SQL Server	Багато SQL БД	Багато SQL і NoSQL БД
Міграції	Автоматичні, декларативні	Через CLI, імперативні	Через CLI або декоратори
API запитів	Функціональний, типобезпечний	Chaining API	Chaining API, QueryBuilder

Продовження таблиці 4.3 Порівняння Prisma з іншими ORM

Відносини	Інтуїтивні, легкі в запитах	Потребують налаштування	Потребують декораторів

Продуктивність	Висока	Середня	Висока
Безпека типів	Дуже висока	Низька	Висока
Raw SQL	Так, з типізацією	Так	Так
Швидкість старту	Швидка	Повільна	Середня
Підтримка serverless	Відмінна	Погана	Середня
Контроль доступу	На рівні моделей	Ні	Ні
Основні переваги	Типобезпечність, інтуїтивний API	Простота, зрілість	Гнучкість, TypeScript
Підходить для	Складні схеми, безпека типів	Традиційні додатки	Великі проєкти, різні БД

У процесі вибору технологічного стеку для нашої онлайн-системи управління проєктами було ретельно проаналізували доступні інструменти та фреймворки, враховуючи їхні сильні сторони, спільноту підтримки, продуктивність та зручність у використанні. В результаті було обрано комбінацію таких технологій і їх переваг, як:

1. Visual Studio Code (VS Code):

- Безкоштовність та відкритий код забезпечують доступність для всіх учасників команди.
- Крос-платформеність дозволяє розробникам працювати в звичних операційних системах.
- Інтуїтивний інтерфейс прискорює адаптацію нових членів команди.

- Потужна екосистема розширень надає можливість налаштувати середовище під конкретні потреби проєкту.

2. Next.js:

- Серверний рендеринг (SSR) та статична генерація (SSG) покращують SEO та швидкість завантаження сторінок.
- Файлова система для маршрутизації спрощує структуру проєкту та навігацію.
- Автоматична оптимізація коду підвищує продуктивність застосунку.
- Тісна інтеграція з React дозволяє використовувати всі переваги цієї бібліотеки.

3. Prisma (ORM):

- Типова безпека зменшує кількість помилок під час роботи з даними.
- Автоматична генерація клієнтів на основі схеми бази даних прискорює розробку.
- Візуальні інструменти полегшують моделювання та управління базою даних.
- Ефективні запити оптимізують взаємодію з базою даних.
- Підтримка різних баз даних надає гнучкість у виборі сховища.

Обраний стек технологій ідеально підходить для нашої системи з кількох причин:

1. Продуктивність: Next.js та Prisma оптимізовані для високої продуктивності, що критично важливо для онлайн-системи, де користувачі очікують миттєвої реакції.
2. Масштабованість: Архітектура Next.js дозволяє легко масштабувати застосунок, а Prisma забезпечує ефективну роботу з базою даних навіть при зростанні обсягу даних.

3. Типова безпека: Використання TypeScript у поєднанні з Prisma значно зменшує ризик помилок, пов'язаних з типами даних, що особливо важливо для системи управління проєктами, де точність даних є ключовою.

4. Зручність використання: Інтуїтивний інтерфейс VS Code та потужні візуальні інструменти Prisma роблять процес розробки більш зручним, що дозволяє команді зосередитися на створенні функціональності, а не на боротьбі з інструментами.

5. Швидкий старт: Завдяки простоті налаштування VS Code, готовим шаблонам Next.js та автоматичній генерації коду Prisma, ми можемо швидко розпочати розробку та зберегти гнучкість для майбутніх змін.

6. SEO та доступність: Функції SSR та SSG у Next.js покращують видимість нашого застосунку в пошукових системах та його доступність для користувачів зі слабким інтернет-з'єднанням або пристроями з обмеженими ресурсами.

Цей технологічний стек не лише відповідає поточним вимогам нашої онлайн-системи управління проєктами, але й забезпечує міцну основу для майбутнього зростання та адаптації до нових викликів. Обрані інструменти дозволяють нам створити надійне, швидке та зручне рішення, яке допоможе користувачам ефективно керувати своїми проєктами в динамічному бізнес-середовищі.

4.3 Користувацький інтерфейс програмної системи

У даній системі управління проєктами користувацький інтерфейс створений із використанням бібліотеки React, що забезпечує гнучкий та інтерактивний досвід для кінцевого користувача. Для стилізації компонентів інтерфейсу застосовуються сучасні CSS-бібліотеки ShadcnUI та TailwindCSS, які надають широкий спектр готових стилів та утиліт, полегшуючи процес розробки та забезпечуючи привабливий і послідовний дизайн.

Ці компоненти інтерфейсу приймають вхідні дані, отримані з моделей системи, і відображають їх у зручному та інтуїтивно зрозумілому форматі. Крім

того, вони дозволяють користувачам взаємодіяти з системою, вводячи дані та виконуючи різноманітні дії, такі як створення, оновлення чи видалення організацій, дошок, списків та карток завдань.

Для кращого розуміння можливих дій користувача в системі, на Рисунку 4.5 представлена діаграма прецедентів, яка наочно демонструє весь спектр функціональних можливостей, доступних для різних користувачів у межах даної онлайн-системи управління проєктами.

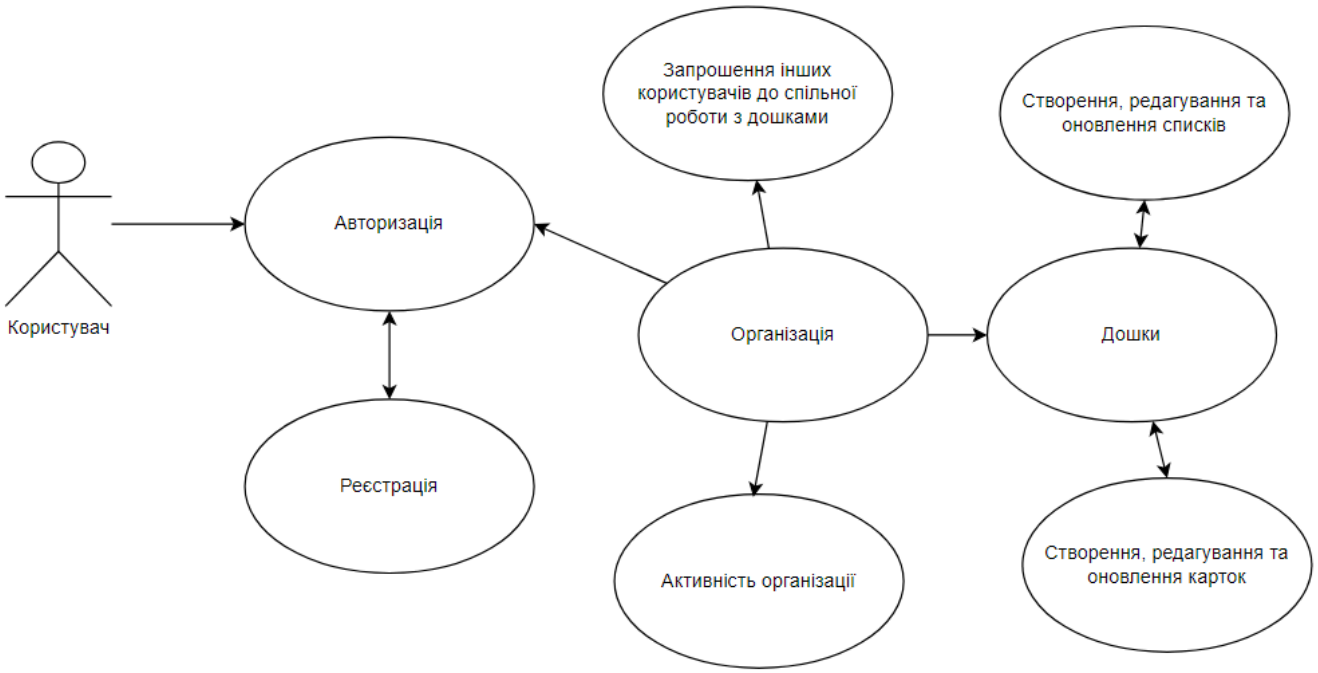


Рисунок 4.1 Діаграма прецедентів

4.4 Сценарій роботи користувача з системою

При переході на веб-сайт, неавторизований користувач одразу бачить головну сторінку, на якій він може перейти на сторінку реєстрації або авторизації.

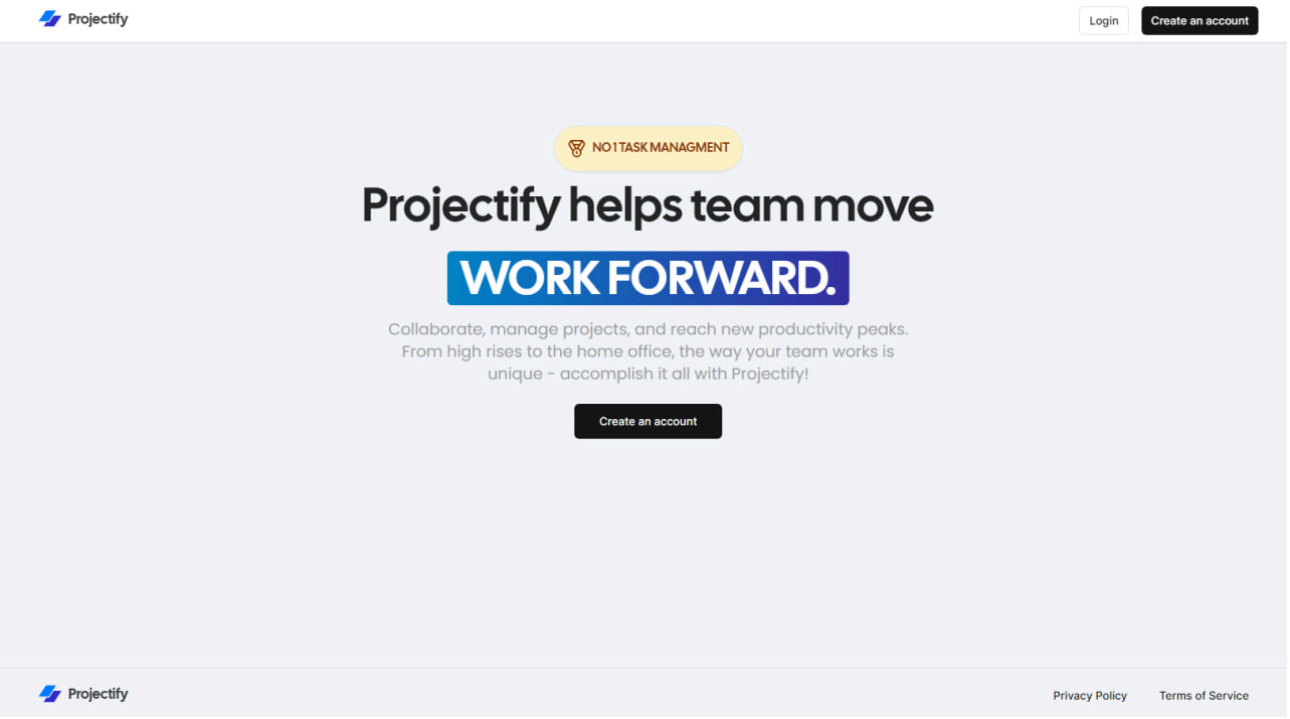


Рисунок 4.2 Головна сторінка неавторизованого користувача

Натискаючи на кнопку “Створити акаунт”, користувач потрапляє на сторінку з модальним вікном, в якому можна створити аккаунт.

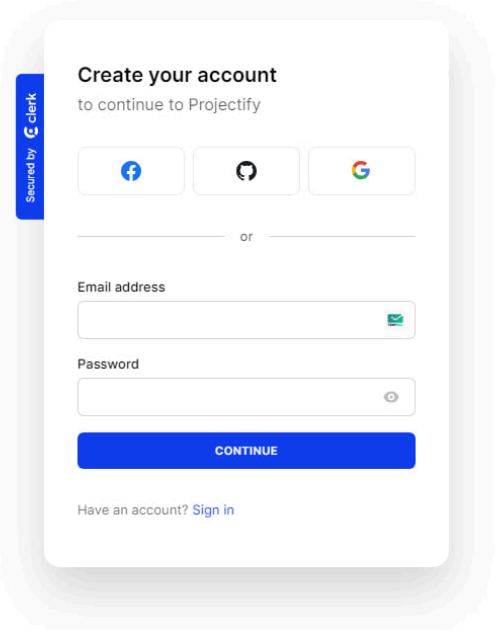


Рисунок 4.3 Сторінка створення акаунту

Після створення акаунту, користувачу потрібно створити організацію(якщо він не був запрошений до будь-якої іншої), запросити інших користувачів для спільної роботи в режимі реального часу.

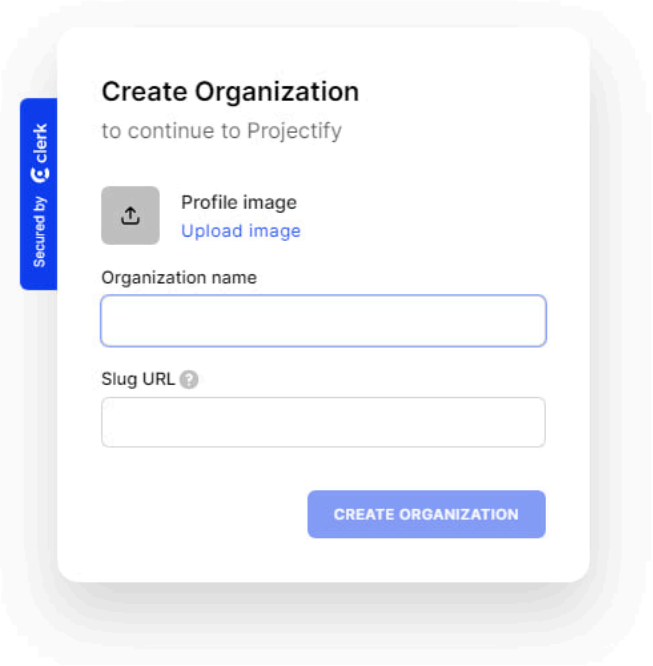


Рисунок 4.4 Модальне вікно з формою створення організації

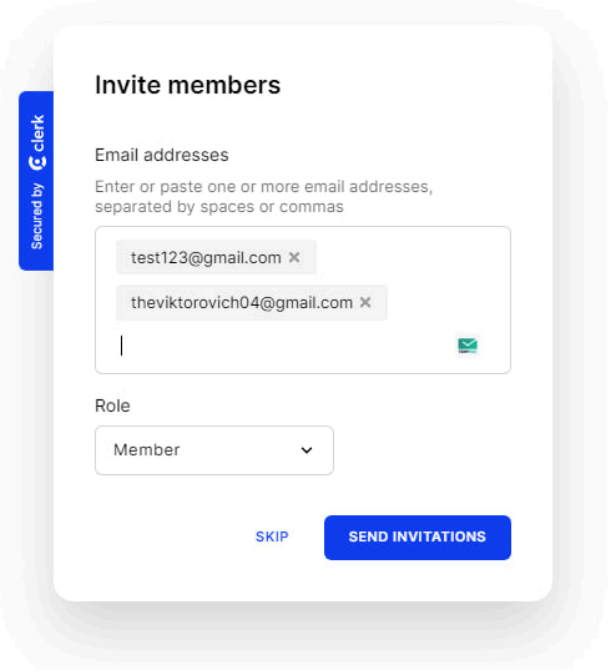


Рисунок 4.5 Модальне вікно з формою запрошення користувачів до організації

Після авторизації та створення організації, користувача перенаправляє на головну сторінку, на якій показує всі організації, до яких приєднаний користувач, та дошки організацій. На цій сторінці користувач може:

1. Переглянути, перейти до наявних дошок, або створити нову;
2. Змінити інформацію про свій акаунт або організацію(якщо є роль “Admin”);
3. Переглянути активність організації.

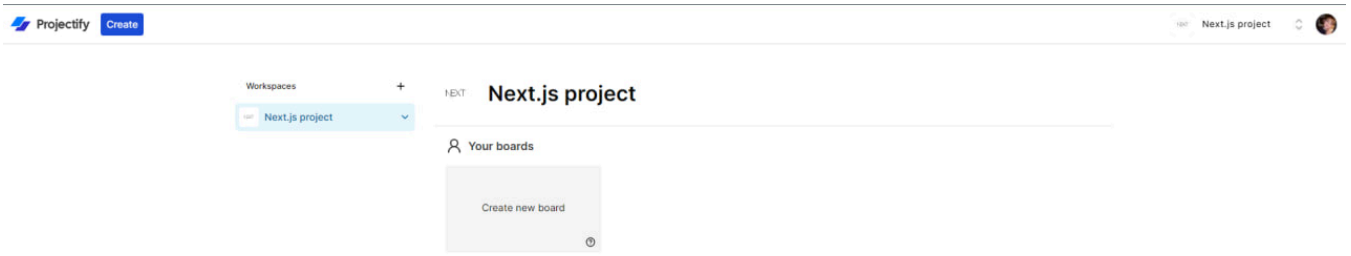


Рисунок 4.6 Головна сторінка авторизованого користувача

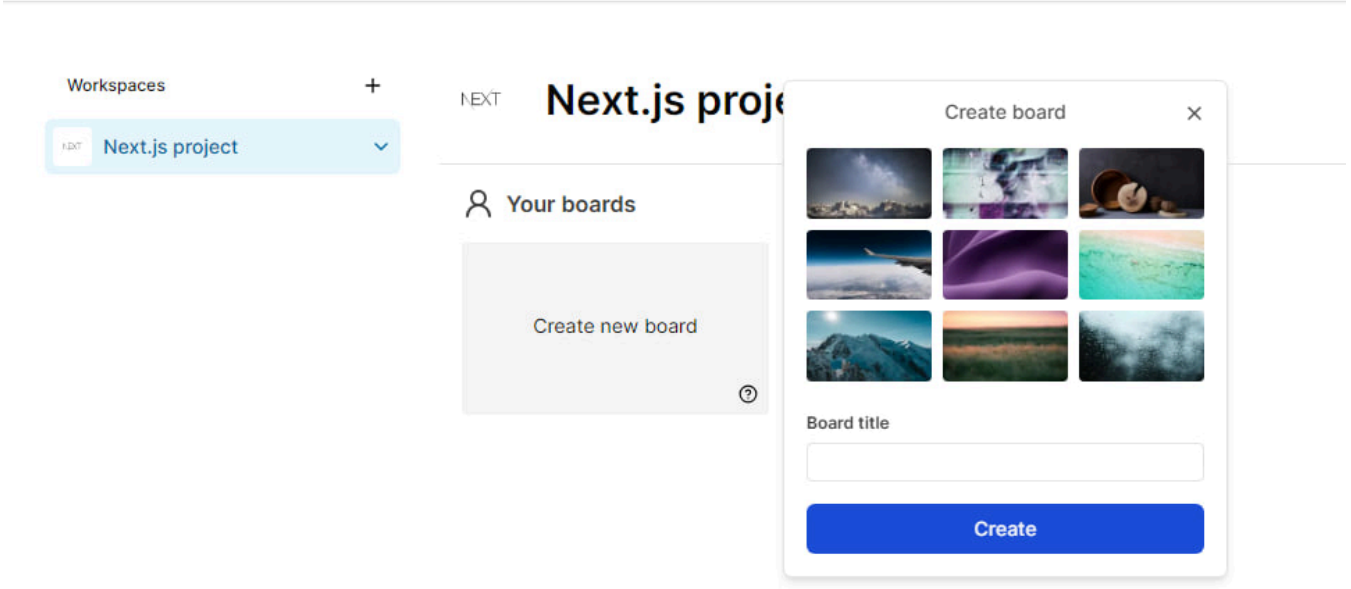


Рисунок 4.7 Модальне вікно з формою створення дошки

Після успішного створення дошки, користувача перенаправляє на сторінку дошки, в якій він може:

1. Редагувати дані про дошку;
2. Створювати списки(етапи) для карток та редагувати їх;
3. Створювати та редагувати картки;
4. Переміщати списки та картки в потрібному йому порядку;

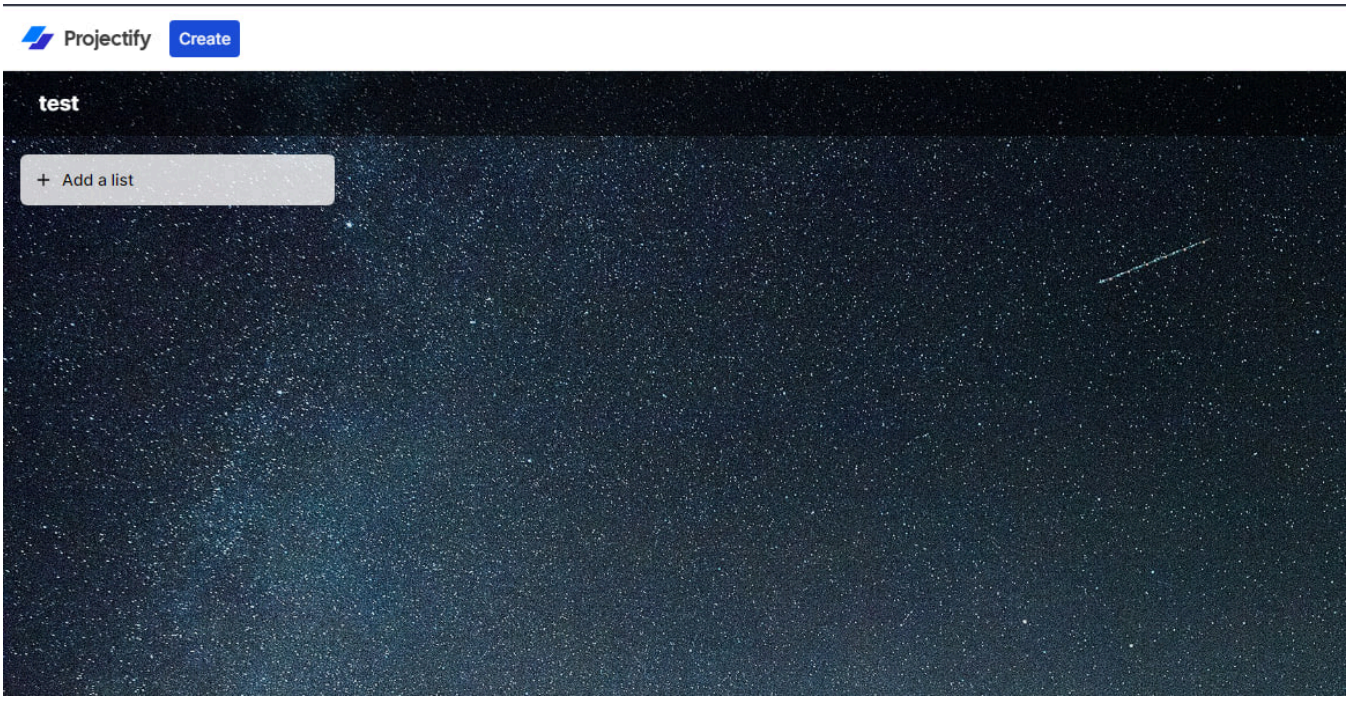


Рисунок 4.8 Сторінка дошки

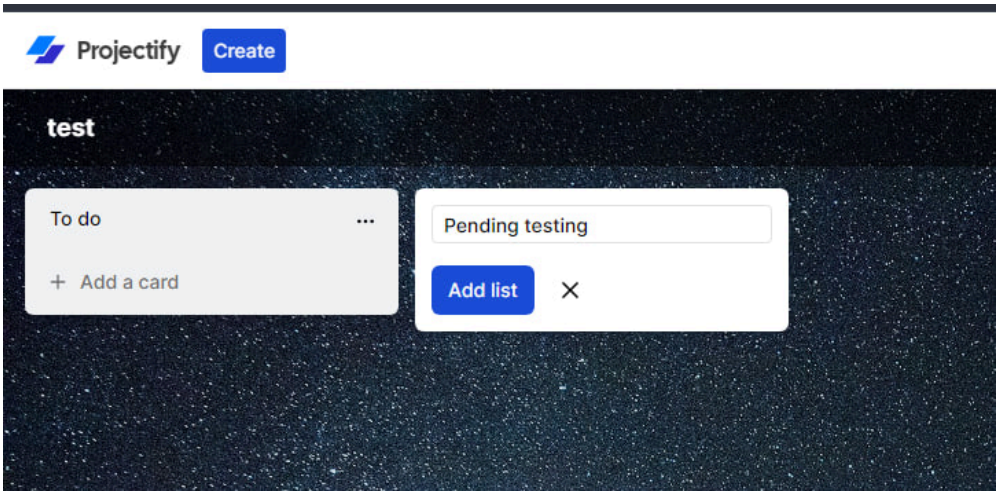


Рисунок 4.9 Створення списків(етапів)

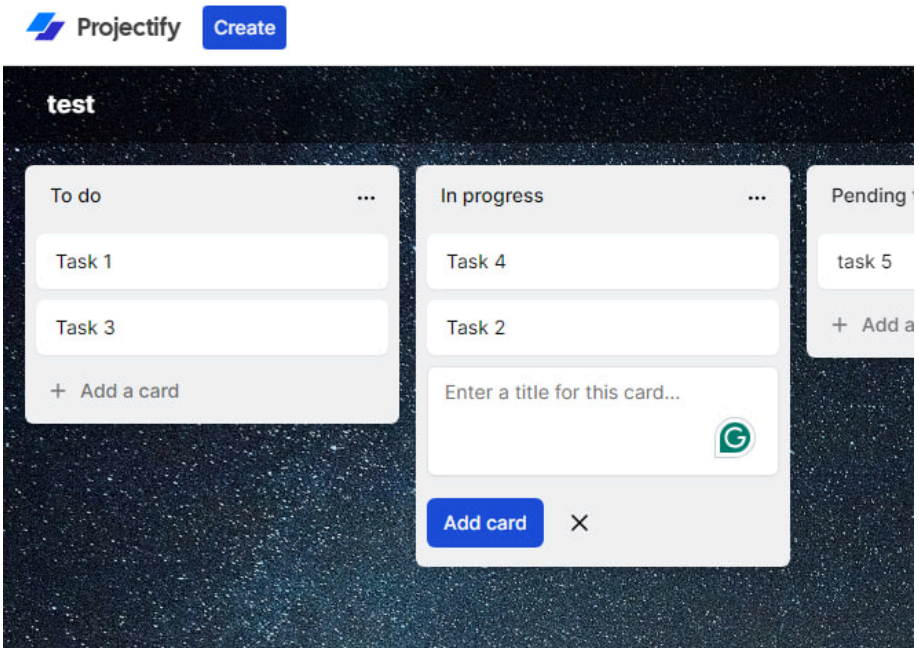


Рисунок 4.10 Створення карток(завдань)

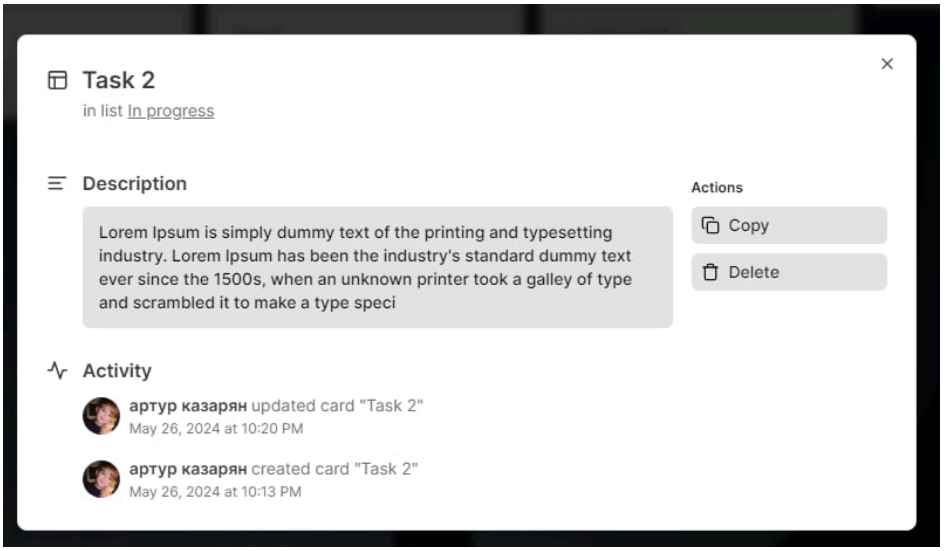


Рисунок 4.11 Модальне вікно з інформацією про картку

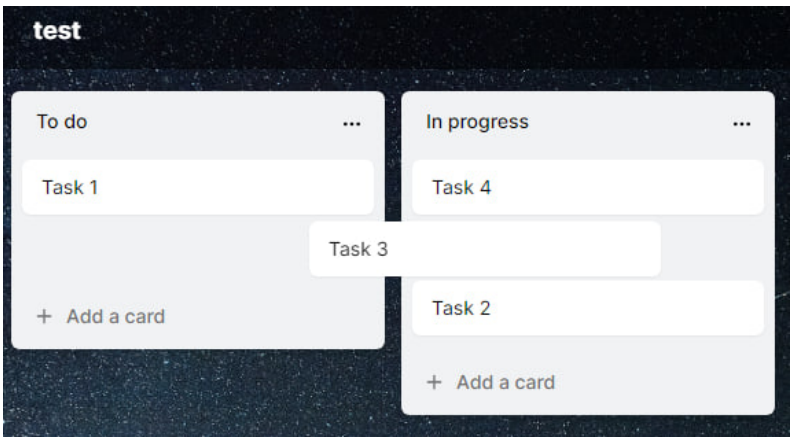


Рисунок 4.12 Процес зміни етапів для картки

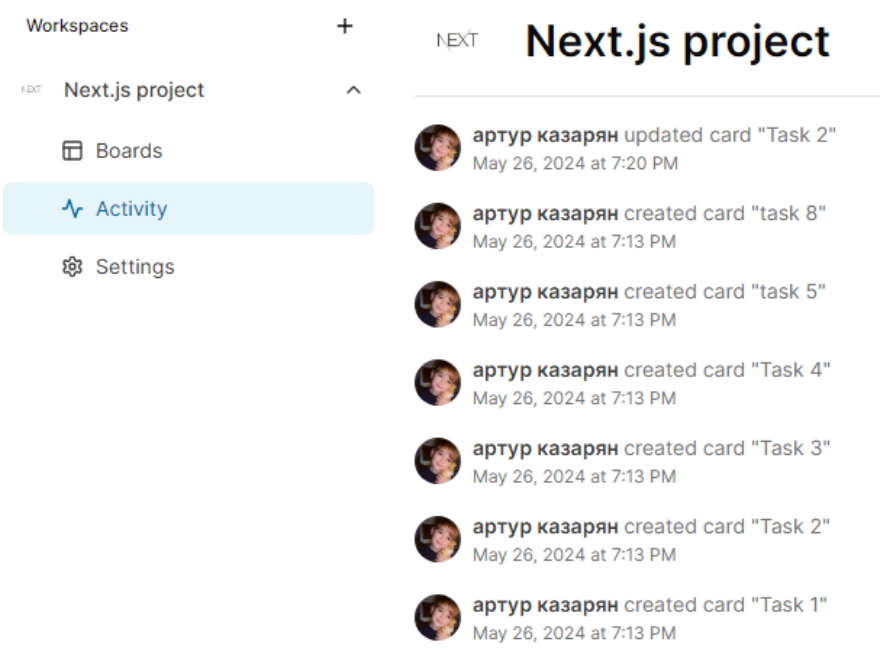


Рисунок 4.13 Список активностей організації

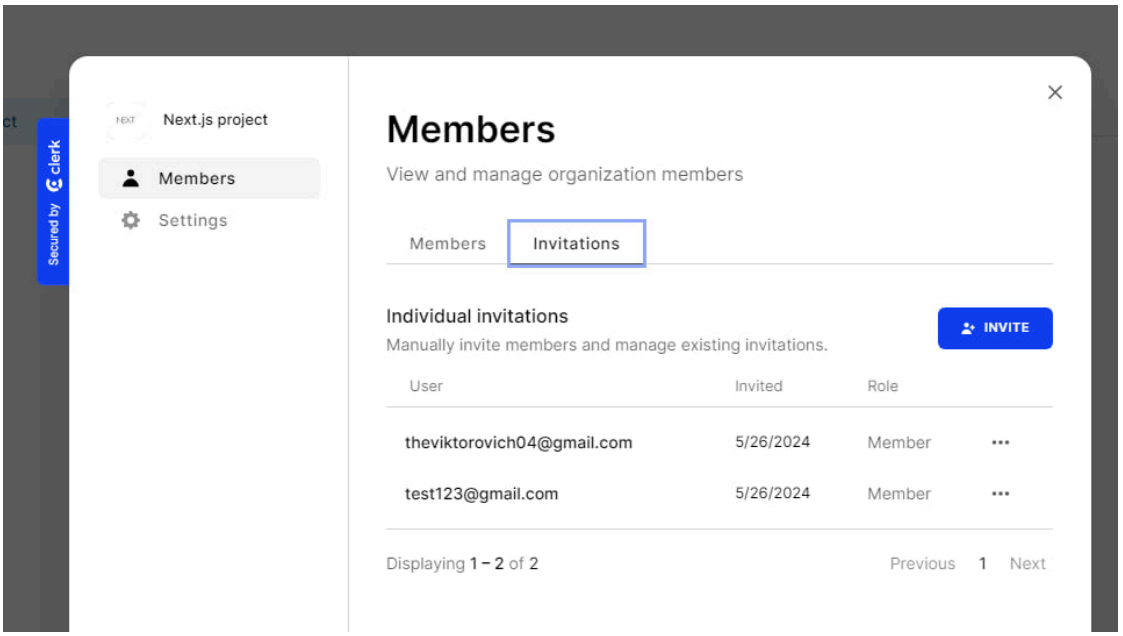


Рисунок 4.14 Інформація про запрошених користувачів

ВИСНОВКИ

У процесі виконання атестаційної роботи було розроблено сучасну онлайн-систему управління проєктами з використанням новітніх технологій веб-розробки, таких як Next.js, React.js, Prisma та TailwindCSS. Дана система покликана вирішити актуальну проблему ефективної організації та відстеження завдань і проєктів, надаючи користувачам інтуїтивно зрозумілий інтерфейс у вигляді дошок з "картками" завдань.

Для досягнення поставленої мети було проведено ґрунтовний аналіз предметної області, що дозволив визначити ключові вимоги та особливості функціонування системи управління проєктами. На основі отриманих результатів було здійснено комплексне проєктування самої системи та її бази даних, забезпечуючи надійність, масштабованість та ефективність роботи.

Отже, в результаті виконання атестаційної випускної роботи було проведено:

- Ретельний аналіз предметної області, вивчення існуючих рішень та виявлення потреб користувачів.
- Комплексне проєктування системи управління проєктами, визначення її архітектури, компонентів та взаємозв'язків між ними.
- Проєктування бази даних, яка забезпечує ефективне зберігання та обробку даних, пов'язаних з проєктами, завданнями, користувачами та іншими сутностями.

На основі розглянутого дослідження зроблено висновок, що задачу управління проєктами можна значно оптимізувати, а також покращити ефективність та зменшити кількість витраченого часу завдяки впровадженню сучасної онлайн-платформи, що дозволяє централізовано керувати проєктами та забезпечує прозорість і оперативність виконання завдань.

Список використаних джерел

1. «Управління проєктами», Л.Є. Довгань [Електронний ресурс] // Режим доступу: https://ela.kpi.ua/server/api/core/bitstreams/cf735d19-339f-44c8-8b5a-42f40468edf4/content

2. Jeff Sutherland, Ken Schwaber. Scrum: The Art of Doing Twice the Work in Half the Time. – Currency, 2014.

3. Eric Ries. The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. – Crown Business, 2011.

4. Gene Kim, Kevin Behr, George Spafford. The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win. – IT Revolution Press, 2013.

5. Ian Sommerville. Software Engineering. 10th Edition. – Pearson, 2015.

6. Martin Fowler. Patterns of Enterprise Application Architecture. – Addison-Wesley, 2002.

7. Robert C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship. – Prentice Hall, 2008.

8. Steve McConnell. Code Complete: A Practical Handbook of Software Construction. 2nd Edition. – Microsoft Press, 2004.

9. Scott W. Ambler, Mark Lines. Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise. – IBM Press, 2012.

10. Kent Beck. Extreme Programming Explained: Embrace Change. 2nd Edition. – Addison-Wesley, 2004.

11. Mike Cohn. Agile Estimating and Planning. – Prentice Hall, 2005.

12. Grady Booch. Object-Oriented Analysis and Design with Applications. 3rd Edition. – Addison-Wesley, 2007.

13. Alistair Cockburn. Writing Effective Use Cases. – Addison-Wesley, 2001.

14. M. Thomas. "Project Management: From Simple to Complex". – Saylor Foundation, 2014. [Електронний ресурс] // Режим доступу: https://saylordotorg.github.io/text_project-management-from-simple-to-complex/

15. What is a project management [Електронний ресурс] // Режим доступу: https://www.pmi.org/about/what-is-project-management

16. What a single page application is [Електронний ресурс] // Режим доступу: https://medium.com/@egoossaert/what-are-single-page-applications-spa-addeaf6717cc

17. What is a project [Електронний ресурс] // Режим доступу: https://kissflow.com/project/what-is-a-project/

18. Project management [Електронний ресурс] // Режим доступу: https://www.managementstudyguide.com/project-management.htm

19. Software architecture patterns [Електронний ресурс] // Режим доступу: https://www.turing.com/blog/software-architecture-patterns-types/

20. Патерни проектування [Електронний ресурс] // Режим доступу: https://refactoring.guru/design-patterns

21. Web development guide [Електронний ресурс] // Режим доступу: https://www.browserstack.com/guide/web-application-development-guide

22. SSR [Електронний ресурс] // Режим доступу: https://solutionshub.epam.com/blog/post/what-is-server-side-rendering

23. Що таке СУБД [Електронний ресурс] // Режим доступу: https://foxminded.ua/systema-upravlinnia-bazamy-danykh

24. Postgres [Електронний ресурс] // Режим доступу: https://data-life-ua.com/db/postgres-is-eating-the-database-world/

25. Документація Next.js [Електронний ресурс] // Режим доступу: https://nextjs.org/docs

26. Документація Prisma [Електронний ресурс] // Режим доступу: https://www.prisma.io/docs

27. Lucidchart [Електронний ресурс] // Режим доступу: https://www.lucidchart.com/pages

28. Документація React.js [Електронний ресурс] // Режим доступу: https://react.dev/

29. DBMS [Електронний ресурс] // Режим доступу: https://www.techtarget.com/searchdatamanagement/definition/database-management-system

30. Документація Postgres [Електронний ресурс] // Режим доступу: https://www.postgresql.org/docs/current/index.html

31. ORM [Електронний ресурс] // Режим доступу: https://www.theserverside.com/definition/object-relational-mapping-ORM

32. Data modeling [Електронний ресурс] // Режим доступу: https://www.ibm.com/topics/data-modeling

33. Database modeling techniques [Електронний ресурс] // Режим доступу: https://vertabelo.com/blog/database-modeling-techniques/

34. Web design rules [Електронний ресурс] // Режим доступу: https://www.wix.com/blog/web-design

ДОДАТКИ

Додаток А. Реалізація дії(action) в системі

Приклад коду А.1. create-card/index.ts

```
const handler = async (data: InputType): Promise<ReturnType> => {
  const { userId, orgId } = auth();

  if (!userId || !orgId) {
    return {
      error: "Unauthorized",
    };
  }

  const { title, boardId, listId } = data;
  let card;

  try {
    const list = await db.list.findUnique({
      where: {
        id: listId,
        board: {
          orgId,
        },
      },
    });

    if (!list) {
      return {
        error: "List not found",
      };
    }

    const lastCard = await db.card.findFirst({
      where: { listId },
      orderBy: { order: "desc" },
      select: { order: true },
    });

    const newOrder = lastCard ? lastCard.order + 1 : 1;
```

```
    card = await db.card.create({
      data: {
        title,
        listId,
        order: newOrder,
      },
    });

    await createAuditLog({
      entityId: card.id,
      entityTitle: card.title,
      entityType: ENTITY_TYPE.CARD,
      action: ACTION.CREATE,
    });
  } catch (error) {
    return {
      error: "Failed to create.",
    };
  }

  revalidatePath(`/board/${boardId}`);
  return { data: card };
};

export const createCard = createSafeAction(CreateCard, handler);
```

Приклад коду А.2. create-card/types.ts

```
import { z } from "zod";
import { Card } from "@prisma/client";

import { ActionState } from "@/lib/create-safe-action";

import { CreateCard } from "./schema";

export type InputType = z.infer<typeof CreateCard>;
export type ReturnType = ActionState<InputType, Card>;
```

Приклад коду А.3. create-card/schema.ts

```
import { z } from "zod";
```

```
export const CreateCard = z.object({
  title: z
    .string({
      required_error: "Title is required",
      invalid_type_error: "Title is required",
    })
    .min(3, {
      message: "Title is too short",
    }),
  boardId: z.string(),
  listId: z.string(),
});
```

Додаток Б. Презентація

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Кваліфікаційна робота
на здобуття освітнього рівня «бакалавр» за темою:
«Розробка онлайн-системи управління проєктами»

Керівник КР: д.т.н., проф. Бородавка Є.В.
Розробив: студент спеціальності
122 «Комп`ютерні науки» ОС «бакалавр»
Казарян А.С.

Київ 2024 р.

- **Актуальність обраної теми** полягає в тому, що ефективне управління проєктами є критично важливим для успіху будь-якої організації в сучасному динамічному бізнес-середовищі. Онлайн-система управління проєктами дозволить командам співпрацювати, відстежувати прогрес, розподіляти ресурси та підвищувати загальну продуктивність проєктної діяльності.
- **Метою дослідження** є розробка онлайн-платформи управління проєктами, яка забезпечить централізоване місце для планування, організації та контролю проєктів. Платформа надасть зручні інструменти для створення організацій, дошок та завдань, допомагаючи командам ефективно керувати своїми проєктами.

- **Проєкт -** це сукупність цілеспрямованих дій, які мають чітко визначені цілі та призначення. Він складається з послідовно виконуваних заходів або робіт, які спрямовані на досягнення конкретного результату.
- **Мета проєкту** – доказовий результат і задані умови реалізації загального завдання проєкту. З точки зору теорії систем управління, проєкт як об'єкт управління повинен бути контрольованим і керованим, тобто виділяються певні характеристики, за якими можна постійно контролювати хід виконання проєкту.

Ознаки та характеристику проєкту:

- Унікальна інноваційна ідея;
- Актуальність для ринку;
- Життєздатність та перспективність;
- Економічна ефективність реалізації;
- Практична корисність продукту;
- Привабливість для інвесторів;
- Можливість масштабування бізнесу;
- Конкурентні переваги рішення;
- Високий потенціал зростання;
- Довгострокова стійкість проєкту.

Саме ці ознаки, або характеристики, відрізняють проєкти від інших заходів, планів, програм, ініціатив.

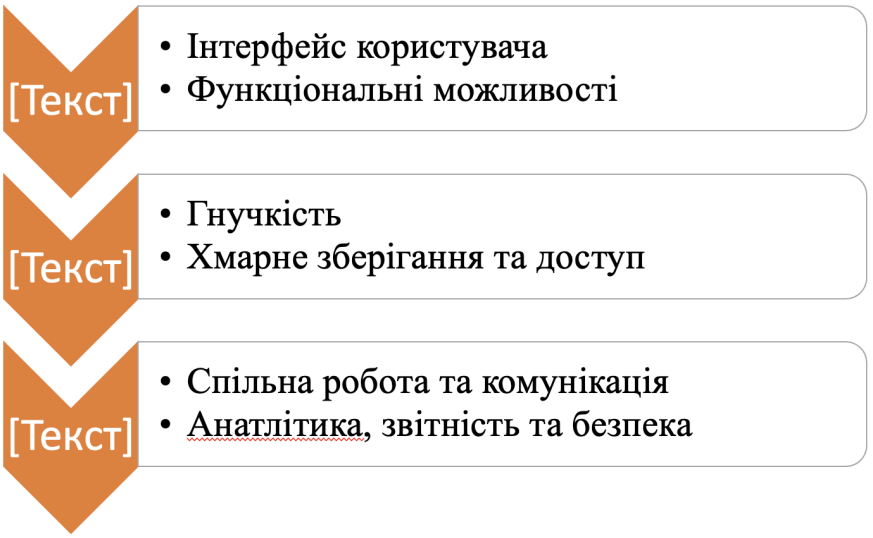
Управління проєктами (project management) - це структурований підхід до планування, організації та контролю за виконанням проєктів з метою досягнення конкретних цілей у встановлені строки, при дотриманні визначених бюджетних і ресурсних обмежень.

Основні етапи управління проєктами включають ініціювання, планування, виконання, моніторинг та завершення проєкту, причому кожен етап потребує детального аналізу та управлінських рішень.

Основні завдання управління проектами:

- Планування, організація, керівництво, контроль;
- Визначення цілей, термінів, ресурсів;
- Розподіл завдань, відповідальностей, обов'язків;
- Управління ризиками, змінами, комунікаціями;
- Забезпечення якості, дотримання бюджету.

Ключові аспекти онлайн-систем управління проєктами:



Розробка веб-застосунків (web application development) є комплексним процесом створення програмного забезпечення, яке функціонує через інтернет або локальну мережу у веб-браузері. Процес розробки починається з аналізу вимог та проектування архітектури, де визначаються функціональні можливості, цілісність даних, безпека і зручність користувацького інтерфейсу. Після цього розробляється прототип, який потім реалізується у вигляді готового продукту

Основними етапами розробки веб-застосунків є:

1. Аналіз вимог;
2. Проєктування;
3. Розробка;
4. Тестування;
5. Розгортання;
6. Підтримка та оновлення.

Постановка задачі

Мета: розробити онлайн-платформу управління проєктами, для прискорення і спрощення роботи команди над проєктом.

Онлайн-платформа повинна мати такі ключові функції:

1. Реєстрацію користувачів для створення персональних облікових записів.
2. Створення організацій та надання ролей із відповідними дозволами для користувачів всередині організації.
3. Запрошення нових учасників для співпраці в рамках організації над спільними проєктами.
4. Формування дошок проєктів, де можна створювати картки (завдання), визначати етапи (стадії виконання) для карток та змінювати їх статус.
5. Створення, редагування та видалення карток завдань на дошці проєкту.
6. Відстеження змін на дошці проєкту в режимі реального часу, а також переглядати історію змін та авторів змін.
7. Перегляд активності користувачів в межах організації для контролю та аналізу їх діяльності.

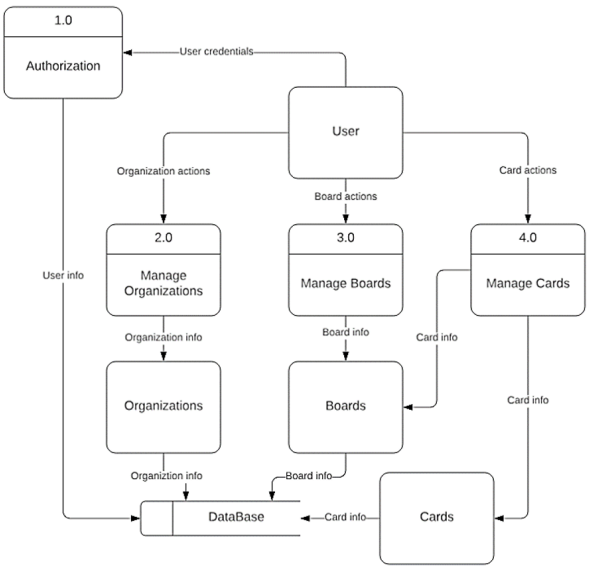
Концептуальна схема системи

Концептуальна схема є високорівневим уявленням системи, яка візуалізує основні компоненти та їх взаємозв'язки без деталізації реалізації. Вона допомагає зрозуміти загальну структуру системи та взаємодію її частин на концептуальному рівні

Функціональна блок-схема потоку

Функціональна блок-схема потоку (Functional Flow Block Diagram, FFBD) – це графічне представлення функціональних потоків у системі або процесі.

FFBD дозволяє систематично представити кожен етап процесу, показуючи логічні зв'язки між різними функціями та забезпечуючи ясність щодо того, як одна функція впливає на іншу.



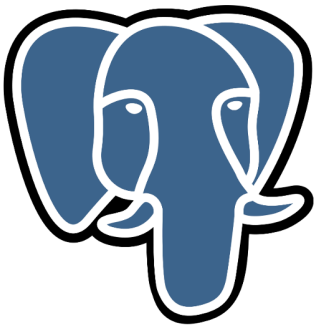
Проєктування бази даних системи

База даних - це фундаментальний компонент сучасних інформаційних систем, який відіграє ключову роль у зберіганні, організації та управлінні даними. За своєю суттю, база даних - це структурована колекція інформації, що зберігається в електронному форматі, дозволяючи ефективний доступ, маніпулювання та оновлення даних.

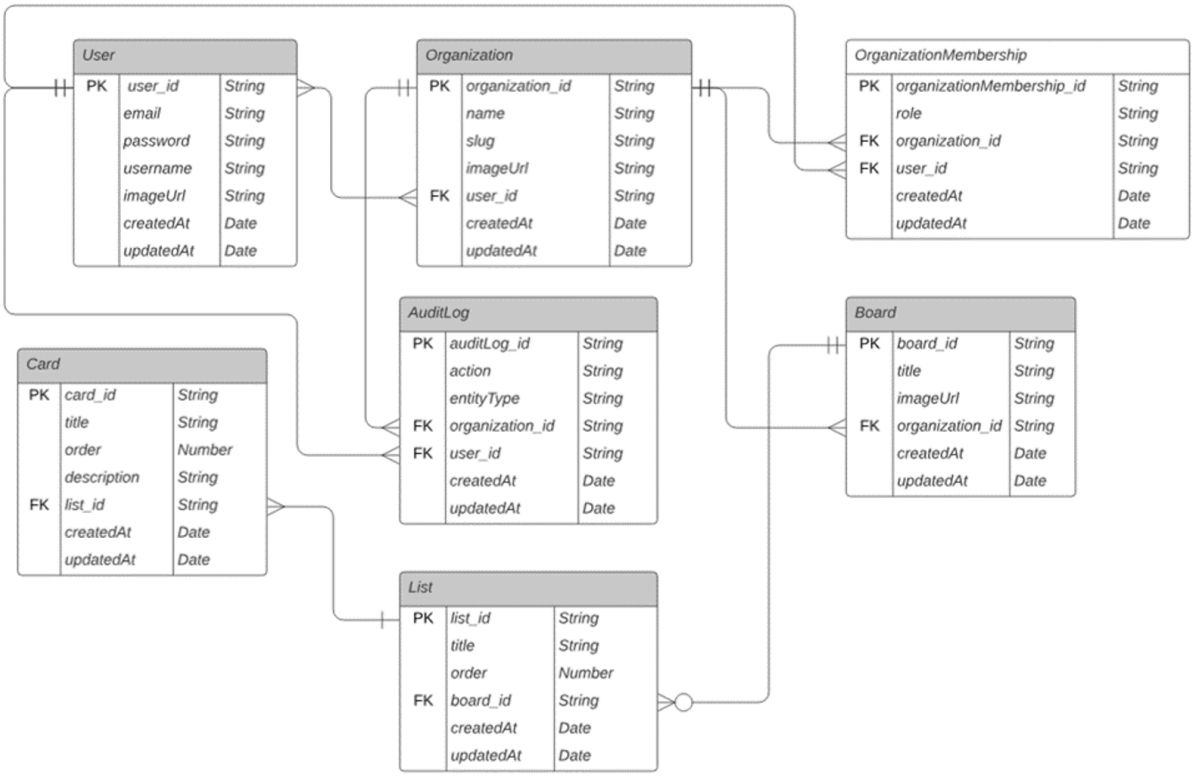
Система управління базами даних (СУБД) - це складний програмний комплекс, який займає центральне місце в інфраструктурі сучасних інформаційних систем. За своєю суттю, СУБД є потужним інструментом, що об'єднує програмні та лінгвістичні засоби, призначені для створення, зберігання, управління та використання баз даних.

Було проаналізовано найпопулярніші СУБД, такі як PostgreSQL, MySQL та Oracle, та серед них було обрано PostgreSQL, оскільки ця система управління базами даних є потужною та надійною. Вона підтримує об'єктно-реляційну модель даних, що робить її гнучкою та масштабованою. Також PostgreSQL є безкоштовною та відкритою системою, що забезпечує високий рівень безпеки та сумісність з різними операційними системами.

PostgreSQL забезпечить цілісність даних, підтримає складні запити для аналітики, масштабуватиметься і захистить важливу інформацію про проєкти. З PostgreSQL ми отримаємо міцний фундамент для створення потужної та гнучкої системи, здатної задовольнити всі наші потреби в управлінні проєктами.



Діаграма сутностей-відносин

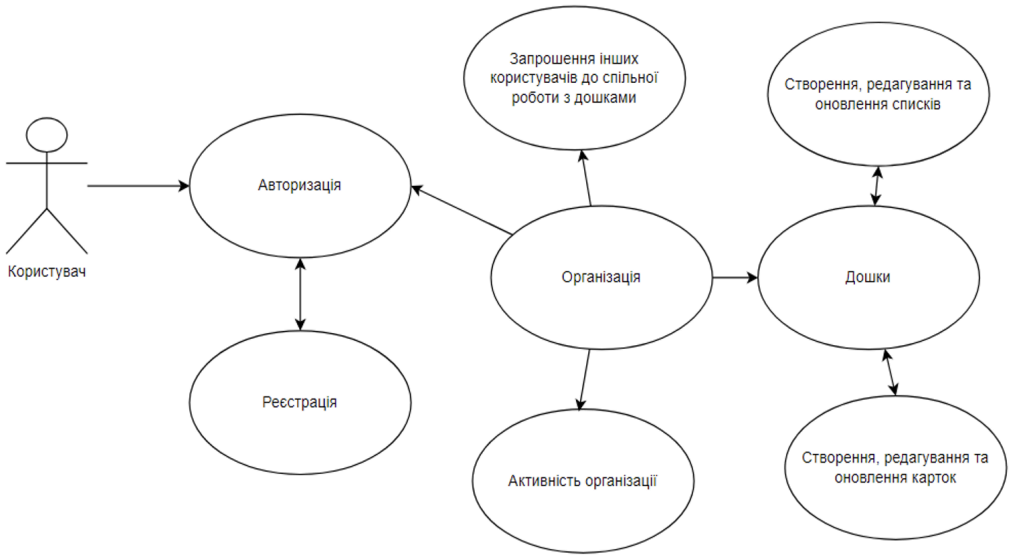


Практична реалізація

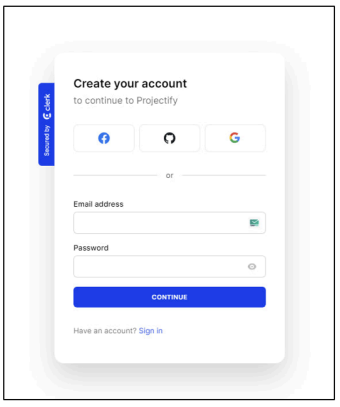
- Visual Studio Code було обрано основним середовищем розробки системи.
- Мовою програмування було обрано TypeScript.
- Основним фреймворком було обрано Next.js.
- TailwindCSS та ShadcnUI було обрано для стилізації.
- Prisma було обрано для роботи з базою даних.

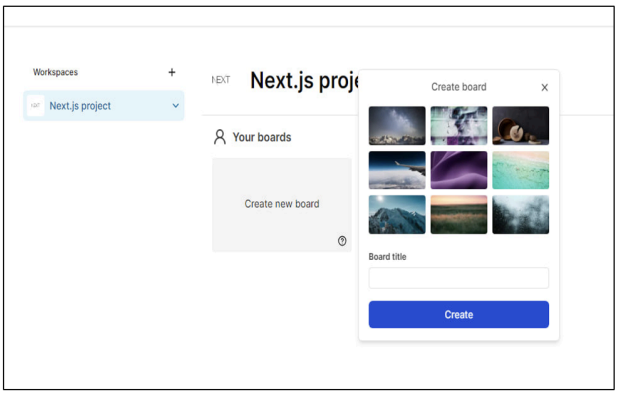
Цей технологічний стек не лише відповідає поточним вимогам нашої онлайн-системи управління проєктами, але й забезпечує міцну основу для майбутнього зростання та адаптації до нових викликів. Обрані інструменти дозволяють нам створити надійне, швидке та зручне рішення, яке допоможе користувачам ефективно керувати своїми проєктами в динамічному бізнес-середовищі.

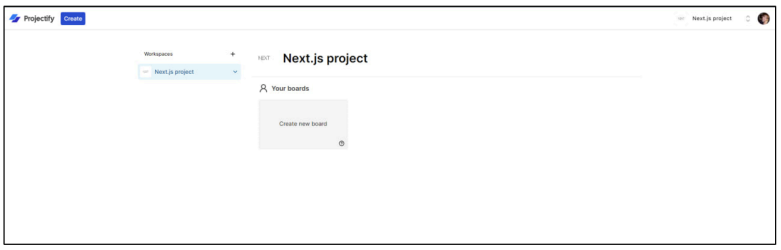
Концептуальна схема

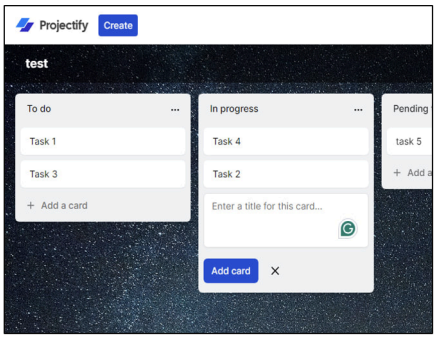


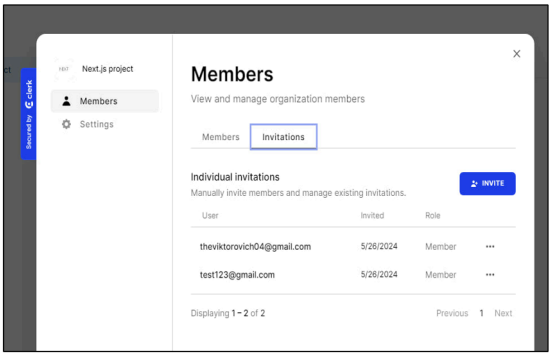
Результат розробки

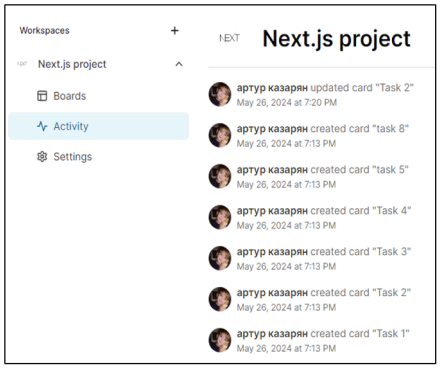


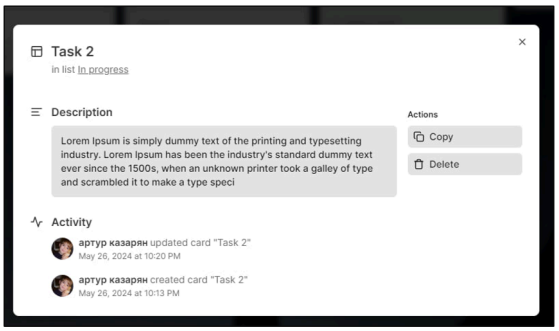












ВИСНОВКИ

У процесі виконання атестаційної роботи було розроблено сучасну онлайн-систему управління проєктами з використанням новітніх технологій веб-розробки, таких як Next.js, Prisma та TailwindCSS. Дана система покликана вирішити актуальну проблему ефективної організації та відстеження завдань і проєктів, надаючи користувачам інтуїтивно зрозумілий інтерфейс у вигляді дошок з "картками" завдань.

Для досягнення поставленої мети було проведено ґрунтовний аналіз предметної області, що дозволив визначити ключові вимоги та особливості функціонування системи управління проєктами. На основі отриманих результатів було здійснено комплексне проєктування самої системи та її бази даних, забезпечуючи надійність, масштабованість та ефективність роботи.

Отже, в результаті виконання атестаційної випускної роботи було проведено:

- Ретельний аналіз предметної області, вивчення існуючих рішень та виявлення потреб користувачів.
- Комплексне проєктування системи управління проєктами, визначення її архітектури, компонентів та взаємозв'язків між ними.
- Проєктування бази даних, яка забезпечує ефективне зберігання та обробку даних, пов'язаних з проєктами, завданнями, користувачами та іншими сутностями.

На основі розглянутого дослідження зроблено висновок, що задачу управління проєктами можна значно оптимізувати, а також покращити ефективність та зменшити кількість витраченого часу завдяки впровадженню сучасної онлайн-платформи, що дозволяє централізовано керувати проєктами та забезпечує прозорість і оперативність виконання завдань.