МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет) Автоматизації і комп'ютерних систем
Кафедра Інформаційних технологій, штучного інтелекту і кібербезпеки

«До захисту в ЕК»	**«До захисту допущено»**
Директор інституту(декан факультету)	Завідувач кафедри
_____ ___Андрій ФОРСЮК__	_____ Сергій ГРИБКОВ__
(підпис) (ім'я та прізвище)	(підпис) (ім'я та прізвище)
«10» __червня_____ 2024р.	«10» ____червня_____ 2024р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

зі спеціальності 122 «Комп'ютерні науки»
(код та назва спеціальності)

освітньо-професійної програми __Комп'ютерні науки

на тему: Розроблення програмного додатку управління проектів ФОП "Горенко Тарас Анатолійович"

Виконав: здобувач _4_ курсу, групи __КН-4-2___

Мельничук Роман Васильович_____ _____
(прізвище, ім'я, по батькові повністю) (підпис)

Керівник Ліманська Наталія Володимирівна_____ _____
(прізвище , ім'я та по батькові повністю) (підпис)

Консультанти_____ _____
(ім'я та прізвище) (підпис)

_____ _____
(ім'я та прізвище) (підпис)

_____ _____
(ім'я та прізвище) (підпис)

Рецензент _____ _____
(ім'я та прізвище) (підпис)

Я як здобувач(ка) Національного університету харчових технологій розумію і підтримую політику університету з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволеної допомоги під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач_____
(підпис)

Київ - 2024р.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет) Автоматизації і комп'ютерних систем
Кафедра Інформаційних технологій, штучного інтелекту і кібербезпеки
Освітній ступінь бакалавр
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інформаційних технологій, штучного інтелекту і кібербезпеки
Сергій ГРИБКОВ

" 15 " квітня 2024 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
Мельничука Романа Васильовича
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення програмного додатку управління проектів ФОП "Горенко Тарас Анатолійович"
керівник роботи Ліманська Наталія Володимирівна, ст. викл.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від 15 квітня 2024 року № 279-кс
2. Строк подання здобувачем роботи 03.06.2024 р.
3. Вихідні дані до роботи
дані про організаційну структуру підприємства ФОП "Горенко Тарас Анатолійович", дані про процес управління проєктами, організаційна структура підприємства, схема функціонування відділів.
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) системний аналіз діяльності підприємства, моделі функціонування підприємства, модель бази даних, розробка інтерфейсу користувача, інструкція користувача, охорона праці та навколишнього середовища
5. Перелік графічного матеріалу
1. Організаційно-функціональна схема підприємства
2. Скріншоти інтерфейсу користувача системи

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Ліманська Н. В., ст. викл.		
2	Ліманська Н. В., ст. викл.		
3	Ліманська Н. В., ст. викл.		
4	Ліманська Н. В., ст. викл.		

7. Дата видачі завдання 15 квітня 2023 року

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів виконання кваліфікаційної роботи	Строк виконання етапів	Примітка
1	Дослідження предметної області та постановка завдання на проектування	29.04.2024	Виконано
2	Проектування бази даних	01.05.2024	Виконано
3	Створення програмного додатку	23.05.2024	Виконано
4	Дослідження питання охорони праці на підприємстві	24.05.2024	Виконано
5	Оформлення пояснювальної записки та створення презентації	26.05.2024	Виконано

Здобувач _____ Мельничук Р.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Ліманська Н.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломний проект: 89 сторінок, 61 рисунка, 15 джерел, 25 таблиць.

Тема диплому: “Розроблення програмного додатку управління проектів ФОП “Горенко Тарас Анатолійович”.

Метою даного дипломного проекту є автоматизація управління задачами та проєктами на підприємстві. Провести аналіз засобів розробки та способів розміщення WEB системи, виконати її моделювання, конструювання та тестування. Мета проекту – обґрунтування актуальності розробки програмного додатку управління проєктами для ІТ-компанії, розглянути питання охорони праці при роботі з комп’ютерною технікою.

У кваліфікаційній роботі було проведено системний аналіз об’єкта дослідження, виявлено задачі автоматизації, досліджено та описано етапи розробки, проаналізовано існуючі аналоги додатку, що розроблявся, обґрунтовано потребу в окремій системі. Також було створено фізичну модель бази даних.

КЛЮЧОВІ СЛОВА: NEXT.JS, REACT.JS, JAVASCRIPT, TAILWIND, VITE, TYPESCRIPT, PRISMA, VISUAL STUDIO CODE, MYSQL, AIVEN, CLERK, WEB, WEB-СИСТЕМА, ДОДАТОК, УПРАВЛІННЯ ЗАДАЧАМИ ТА ПРОЄКТАМИ.

ANNOTATION

Diploma project: 89 pages, 61 figures, 15 sources, 25 tables.

Diploma topic: “Development of a software application for project management of the individual entrepreneur Horenko Taras Anatoliiovych”.

The purpose of this diploma project is to automate the management of tasks and projects at the enterprise. To analyse the development tools and methods of placing the WEB system, to perform its modelling, design and testing. The purpose of the project is to substantiate the relevance of developing a project management software application for an IT company, to consider the issues of labour protection when working with computer equipment.

In the qualification work, a systematic analysis of the research object was carried out, automation tasks were identified, the development stages were investigated and described, existing analogues of the application under development were analysed, and the need for a separate system was substantiated. A physical model of the database was also created.

KEYWORDS: NEXT.JS, REACT.JS, JAVASCRIPT, TAILWIND, VITE, TYPESCRIPT, PRISMA, VISUAL STUDIO CODE, MYSQL, AIVEN, CLERK, WEB, WEB-SYSTEM, APPLICATION, TASKS AND PROJECTS MANAGEMENT.

ЗМІСТ

ВСТУП

У сучасному світі технології розвиваються неймовірними темпами, суттєво впливаючи на всі сфери нашого життя. Особливо помітним є вплив інформаційних технологій на бізнес-середовище, де ефективне управління проектами та задачами стає ключовим фактором успіху. У зв'язку з цим, інтеграція сучасних систем керування проектами в компаніях різних масштабів набуває дедалі більшої актуальності.

Застосування сучасних технологій дозволяє створювати потужні та гнучкі додатки, які значно спрощують процес управління проектами. Вони забезпечують зручний інтерфейс для командної роботи, сприяють підвищенню продуктивності та ефективності робочих процесів, а також дозволяють швидко адаптуватися до змінних умов ринку.

Інтеграція системи керування проектами в сучасні компанії сприяє не лише оптимізації внутрішніх процесів, але й покращенню взаємодії між працівниками та підвищенню загальної ефективності роботи. Це, в свою чергу, дозволяє компаніям досягати кращих результатів та залишатися конкурентоспроможними на ринку.

Об’єкт дослідження – ІТ-компанія зі створення сайтів

Предмет дослідження – робота відділу розробки

Мета – розробити додаток для управління проєктами на підприємстві.

Відповідно до мети були визначені такі завдання:

- провести системний аналіз об’єкта дослідження;
- визначити нинішній стан комп’ютеризації;
- розробити функціональну модель;
- проаналізувати існуючі рішення аналогів системи;
- обґрунтувати доцільності проектування і розробки;
- скласти технічне завдання;
- описати комплекс задач системи.

РОЗДІЛ 1. СИСТЕМНИЙ АНАЛІЗ ОБ'ЄКТА ДОСЛІДЖЕННЯ ТА ВИЯВЛЕННЯ ЗАДАЧ АВТОМАТИЗАЦІЇ

1.1. Загальна характеристика підприємства

Підприємство ФОП «Горенко Тарас Анатолійович» було засноване два роки тому як стартап, що спеціалізується на наданні послуг у сфері веб-розробки. Сьогодні воно об'єднує двох дизайнерів, одного менеджера проєктів, п'ятьох програмістів, двох тестувальників, CEO, SEO-спеціаліста та бухгалтера. Кожен член команди вносить унікальний вклад у спільну справу, що дозволяє підприємству рухатися вперед та досягати нових висот.

Основною метою підприємства, що спеціалізується на розробці веб-сайтів, є надання якісних послуг у сфері веб-розробки, веб-дизайну та просування сайтів в інтернеті. Компанія прагне досягати лідерства на ринку IT-послуг шляхом реалізації інноваційних проєктів, використання передових технологій та надання висококласного сервісу.

Стратегічні цілі включають:

- інновації у сфері веб-розробки: постійне оновлення технічної бази та впровадження новітніх розробок у процес створення веб-сайтів, щоб забезпечити клієнтам унікальні та ефективні рішення;
- збільшення портфоліо клієнтів: розширення бази клієнтів за рахунок приваблення великих корпоративних замовників, а також розвитку партнерських відносин з іншими IT-компаніями та агенціями цифрового маркетингу;
- підвищення якості послуг: ключовим фактором успіху є не лише інноваційність та креативність рішень, але й висока якість кінцевого продукту. Підприємство встановлює суворі стандарти якості на всіх етапах розробки проєкту, від дизайну до запуску сайту, що гарантує клієнтам отримання надійного та ефективного продукту;
- розвиток та навчання персоналу: підприємство приділяє велику увагу професійному та особистісному розвитку своїх співробітників. Регулярно організовуються тренінги, воркшопи та майстер-класи, які

дозволяють команді не тільки вдосконалювати свої навички, але й ефективно працювати над складними проєктами, обмінюватися досвідом та ідеями;

- вихід на міжнародний ринок: з метою забезпечення сталого зростання, підприємство прагне до розширення своєї присутності на міжнародних ринках. Це передбачає не тільки просування послуг за кордоном, але й розробку проєктів, що враховують специфіку та вимоги іноземних замовників, адаптацію до міжнародних стандартів якості та взаємодію з іноземними партнерами.

Ці стратегічні цілі визначають напрямок довгострокового розвитку підприємства та допомагають концентрувати зусилля на досягненні встановлених завдань. Підприємство прагне не лише до збільшення прибутковості, але й до створення позитивного іміджу на ринку IT-послуг, розвитку інноваційної культури та підтримки високих стандартів у сфері веб-розробки.

1.2. Основні напрямки діяльності підприємства

Підприємство у сфері веб-розробки встановило собі за мету не просто виконання замовлень на розробку сайтів, а створення комплексних інтернет-рішень, які допомагають клієнтам реалізувати їхній бізнес-потенціал у повному обсязі. Розробка веб-сайтів "під ключ" стала фундаментом діяльності компанії, охоплюючи весь спектр послуг – від проєктування і дизайну до програмування, тестування та подальшої підтримки.

Зусилля компанії спрямовані на створення унікальних веб-сайтів, які не тільки приваблюють відвідувачів своїм дизайном, але й забезпечують зручність користування, швидкість завантаження та оптимізацію під пошукові системи. Це дозволяє сайтам клієнтів ефективно конвертувати відвідувачів у покупців або абонентів, забезпечуючи високий рівень взаємодії з цільовою аудиторією.

Інтернет-маркетинг є ще одним критично важливим напрямком для підприємства. Розуміння того, що навіть найкращий сайт не принесе очікуваних результатів без ефективного просування, спонукало компанію розробити комплексні стратегії інтернет-маркетингу, які включають SEO-оптимізацію,

контекстну рекламу, роботу в соціальних мережах та електронну комерцію. Ці заходи спрямовані на збільшення видимості сайтів у мережі, залучення цільового трафіку та підвищення конверсії.

Технічна підтримка та супровід сайтів є не менш важливим аспектом діяльності компанії. Постійний моніторинг стану сайтів, оперативне вирішення будь-яких технічних проблем та регулярне оновлення контенту забезпечують безперебійну роботу інтернет-ресурсів клієнтів, зберігаючи їх актуальність та ефективність.

Окрім безпосередньої розробки, підприємство надає консалтингові послуги в області веб-розробки та цифрового маркетингу, допомагаючи клієнтам визначити найефективніші стратегії для розвитку їх онлайн-бізнесу. Аналітика ринку, дослідження поведінки користувачів та вивчення конкурентного середовища дозволяють компанії пропонувати клієнтам обґрунтовані рекомендації, спрямовані на підвищення ефективності їх інтернет-проєктів.

Таким чином, підприємство прагне виконувати комплексну місію - не лише створювати веб-сайти, але й забезпечувати їх успіх у мережі, використовуючи для цього всі доступні інструменти та технології. Стратегія компанії заснована на інноваціях, якості, клієнтоорієнтованості та неперервному розвитку, що дозволяє їй втілювати найсміливіші ідеї та задовольняти найвищі вимоги своїх клієнтів.

1.3. Структура підприємства

Центральне місце в структурі займає виконавчий директор (CEO), який визначає стратегічний напрямок розвитку компанії та відповідає за загальне керівництво.

Під керівництвом CEO працює команда професіоналів, розділена на кілька ключових відділів:

- Керівник проєктів відповідає за координацію проєктних команд і забезпечення виконання всіх проєктів у встановлені терміни із дотриманням технічних вимог та бюджету. Він також є основним зв'язком

між командою розробників та клієнтами, відповідаючи за звітність та комунікації.

- Відділ дизайнерів складається з двох фахівців, котрі займаються розробкою візуального оформлення сайтів. Вони відповідають за створення дизайн-концепцій, прототипів інтерфейсів та забезпечують зручність використання та естетичну привабливість продуктів.
- Розробники - це команда з п'яти програмістів, яка працює над створенням функціональної частини проєктів: від розробки архітектури сайтів до написання коду і впровадження необхідних технічних рішень. Вони працюють з різними мовами програмування та технологіями адаптуючи їх під потреби кожного окремого проєкту.
- Відділ тестувальників (QA) включає двох спеціалістів, що займаються перевіркою якості готових продуктів. Їхнє завдання - забезпечити, щоб усі елементи сайту працювали без збоїв і відповідали технічному завданню, а також виявляти та усувати будь-які помилки перед запуском проєкту.
- SEO-спеціаліст відповідає за оптимізацію сайтів з метою поліпшення їх позицій у пошукових системах, що включає роботу з ключовими словами, змістом сайту, його структурою та зовнішніми факторами, які впливають на ранжування.
- Бухгалтер забезпечує ведення фінансового обліку, оптимізацію податкових витрат, звітність перед державними органами та контроль за бюджетами проєктів.

Ця структура забезпечує ефективне управління ресурсами підприємства та гарантує злагоджену роботу всієї команди. Кожен співробітник має чітко визначені функції та відповідальність, що дозволяє компанії досягати високих результатів та задовольняти вимоги найвимогливіших клієнтів.

На рис. 1.2 зображена організаційно-функціональна схема підприємства ФОП «Горенко Тарас Анатолійович».

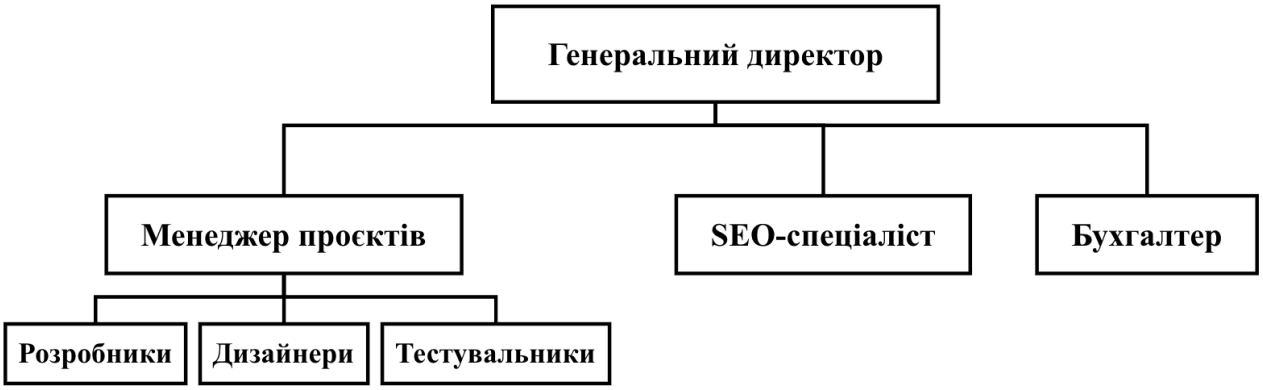

Рисунок. 1.1 – Організаційно-функціональна схема ФОП «Горенко Тарас Анатолійович»

В таблиці 1.1 зображена функціональна схема ФОП «Горенко Тарас Анатолійович».

Таблиця 1.1. Функціональна схема. Взаємодія підрозділів

№	Підрозділ	Одержання	Надання
1	Генеральний директор	- стратегічні звіти від усіх відділів - аналіз ринку та конкурентів	- керівні рішення - стратегічний напрямок розвитку компанії
2	Менеджер проєктів	- звіти про стан проєктів від команд розробників - вимоги клієнтів	- координація проєктних команд - розподіл ресурсів і завдань
3	Дизайнери	- вимоги до дизайну від менеджера проєктів	- макети дизайну
4	Розробники	- технічні завдання від менеджера проєктів - зворотний зв'язок від тестувальників	- розроблений функціонал сайтів - технічна документація

Продовження таблиці 1.1. Функціональна схема. Взаємодія підрозділів

№	Підрозділ	Одержання	Надання
5	Тестувальники	- розроблений код від програмістів - оновлення програмного забезпечення	- звіти про помилки і баги - рекомендації щодо удосконалень
6	SEO-спеціалісти	- аналітика веб-трафіку - зміни алгоритмів пошукових систем	- стратегії оптимізації для пошукових систем - рекомендації щодо контенту
7	Бухгалтер	- фінансові звіти від усіх відділів - банківські виписки	- фінансовий облік та звітність - бюджетування та фінансове планування

1.4. Розробка функціональної моделі за аналіз існуючих бізнес-процесів

Обрано CASE-засіб «bpmn.io», який підтримує моделювання відповідно до стандартів BPMN (Business Process Model and Notation). «bpmn.io» надає зручний інтерфейс та набір функцій для створення функціональних моделей, які можуть бути легко інтегровані з іншими IT-системами підприємства. Розглянемо процес створення веб-сайту.

1. **Пошук клієнта генеральним директором:** генеральний директор використовує свої контакти та ресурси для ідентифікації потенційних клієнтів, які можуть мати інтерес до створення нового веб-сайту або оновлення існуючого, організовує зустрічі з ними.
2. **Оптимізація вимог:** після встановлення першого контакту з клієнтом генеральний директор спільно з менеджером проєктів аналізує та оптимізує вимоги до проекту, забезпечуючи їхню відповідність до можливостей компанії та очікувань клієнта.

3. **Планування проекту:** процес планування включає розробку базового плану проекту, який визначає ключові етапи, ресурси, бюджет та таймлайн. Він також включає розподіл завдань між командами розробників, дизайнерів та інших учасників проекту.
4. **Розробка та дизайн:** на цьому етапі команда дизайнерів та розробників працює над створенням макетів та розробкою сайту. Розробники займаються кодуванням, в той час як дизайнери створюють візуальний вигляд.
5. **Тестування:** тестувальники перевіряють сайт на відповідність технічним завданням та вимогам користувачів. Вони виявляють та усувають баги, переконуючись, що сайт готовий до запуску.
6. **Передача веб-сайту клієнту:** після завершення всіх етапів розробки та успішного тестування, сайт передається до продакшн-оточення, де він стає доступним для користувачів. Перед запуском відбувається остаточна перевірка функціональності та профілювання продуктивності.

Кожен з цих етапів вимагає тісної координації між усіма учасниками проекту, а також чіткого зворотного зв'язку та відстеження прогресу. Успішне виконання цього процесу вимагає не тільки технічних навичок, але й ефективного проектного управління та комунікації всередині компанії.

Розглянемо bpmn-діаграму (рисунок 1.2), щоб виявити прогалини в процесі розробки.

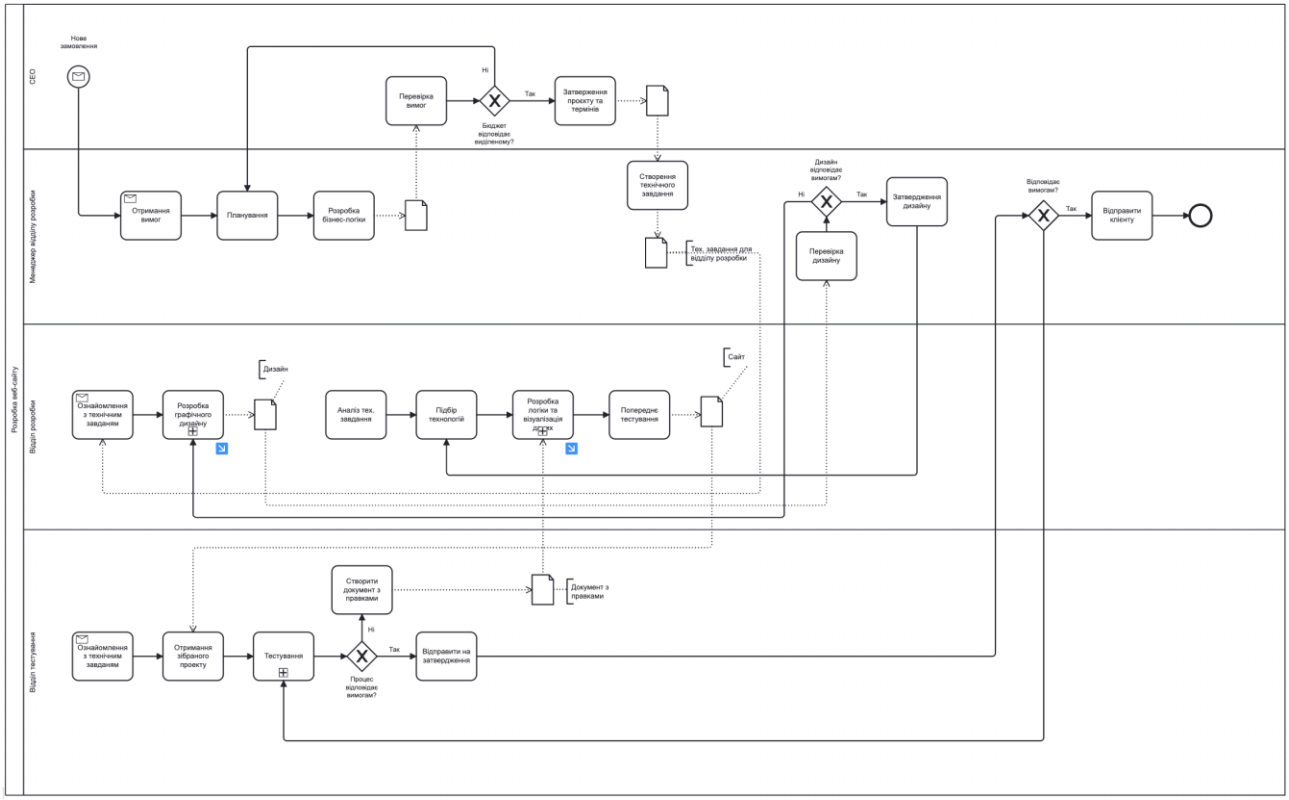

Рисунок 1.2 – BPMN-діаграма процесу створення веб-сайту

1.5. Аналіз нинішнього стану комп’ютеризації та виявлені проблеми

Стан автоматизації на підприємстві відіграє вирішальну роль у забезпеченні ефективності, продуктивності та конкурентоздатності компанії. Автоматизація процесів є одним з ключових факторів, який дозволяє підприємству швидко адаптуватися до змін на ринку та задовольняти зростаючі вимоги клієнтів.

Наразі підприємство має середній рівень автоматизації ключових бізнес-процесів. Головною причиною, яка зумовлює такий рівень – це відсутність системи управління проєктами. Зараз управління проєктами відбувається на примітивному рівні, а саме в таблиці Google Sheets. Як наслідок - відсутнє ефективне планування, моніторинг ходу робіт та ресурсів, відстеження термінів виконання задач та оперативне реагування на будь-які зміни у проєктах. З рештою процесів ситуація краща.

Автоматизовані системи обліку дозволяють вести фінансовий облік з високою точністю та прозорістю, забезпечуючи своєчасне виявлення фінансових результатів та аналіз ефективності бізнесу.

Комунікаційні процеси в компанії також піддані автоматизації. Використання сучасних CRM-систем та інструментів для внутрішнього обміну повідомленнями значно підвищує швидкість та якість обміну інформацією між відділами, а також з клієнтами. Це спрощує процес управління відносинами з клієнтами та збільшує їх задоволеність від взаємодії з підприємством.

Автоматизація процесів тестування та випуску продуктів дозволяє компанії забезпечити високу якість та надійність розроблюваних веб-сайтів. Застосування автоматизованих інструментів тестування та неперервної інтеграції (CI/CD) сприяє ефективному виявленню та усуненню помилок, скороченню часу на розробку та впровадження нових функцій.

У наступному пункті буде надано детальну інформацію про програмне забезпечення та технології, що використовуються на підприємстві для автоматизації процесів, що дозволить глибше зануритися в технічні аспекти роботи компанії.

Кожен ключовий етап розробки, такий як: створення та передача технічного завдання в розробку, затвердження дизайну, тестування і т.д. вимагає тісної співпраці між відділами, а також відслідковування поставлених і виконаних завдань та обмін документами. Зараз це відбувається за допомогою чату Telegram та Google Sheets, що є критично неефективно. Працівники витрачають багато часу на орієнтацію та пошук необхідних даних в Google Sheets, мають великі ризики втрати уваги та сконцентрованості при переході в Telegram.

1.5.1. Задачі автоматизації

Мінімізувати фінансові втрати за рахунок оптимізації. Досягти конкурентної переваги за рахунок скорочення термінів. Забезпечити оперативне отримання повної і достовірної інформації про задачі та проєкти. Реалізувати

збереження інформації в зручному вигляді та виконати захист від несанкціонованого доступу. 1.6.

Огляд наявних рішень для вирішення визначених проблем є одним з ключових етапів у розробці нової системи управління проектами. Цей етап дозволяє дослідити різноманітні програмні варіанти, оцінити їхні переваги та недоліки. Порівняльний аналіз доступних рішень допомагає знайти найбільш оптимальний і ефективний шлях для вирішення завдань управління проектами.

Серед можливих аналогів розроблюваної системи розглядаються вже існуючі рішення, які описані далі.

«Asana»

Asana є потужним інструментом управління проектами, який відрізняється своєю універсальністю, інтуїтивно зрозумілим інтерфейсом та широким спектром функцій. Він підходить для різних типів проектів та команд, від невеликих до великих. Основні особливості Asana включають призначення завдань, терміни виконання, інтегровані календарі та можливість інтеграції з іншими інструментами, такими як Slack, Google Drive та Microsoft Teams, що дозволяє централізувати всю важливу інформацію.

Asana сприяє підвищенню продуктивності, забезпечуючи ефективність управління завданнями та поліпшуючи командну співпрацю. Використання Asana дозволяє командам ефективніше керувати своїм робочим навантаженням, дотримуватися термінів та зменшувати стрес, пов'язаний з складними проектами.

Проте, як і будь-який інструмент, Asana має свої обмеження. Для дуже великих або складних проектів інтерфейс Asana може стати перенасиченим, а деякі інструменти та платформи можуть не інтегруватися належним чином. Також деякі команди можуть вважати преміум-план Asana дещо дорогим.

В Asana існує безкоштовний базовий план, який підходить для багатьох користувачів, але преміум-плани пропонують додаткові функції, які можуть виправдати їх вартість. Вона включає функції, як-от безлімітні проекти, завдання та коментарі, дошки Kanban, календарний перегляд і списки. Основні обмеження

безкоштовного плану полягають у підтримці лише 15 учасників команди та 1 000 завдань.

Asana підходить для використання в рамках як водоспадної, так і Agile методологій, надаючи гнучкість у системі, що орієнтована на призначення завдань. Особливо підходить для Agile підходу, оскільки вкладка «Дошка» на часовій шкалі може служити як Kanban дошка для відстеження прогресу різних завдань.

Таким чином, Asana - це універсальний інструмент управління проектами, який підходить для широкого спектру команд і проектів, пропонуючи баланс функцій і зручності використання. Він особливо корисний для команд, які надають перевагу простоті використання та різноманітним інтегративним функціям.

«ClickUp»

ClickUp — це багатофункціональна система управління проектами, яка вирізняється своєю високою налаштовуваністю та багатством функцій. Вона підходить для організацій, які потребують індивідуально налаштованого інструменту управління проектами. Система має потужні можливості для практик Agile, включаючи інструменти для створення дашбордів, шаблонів та інструментів для спринтів. Крім того, ClickUp пропонує різноманітність інтеграцій і сумісна з більш ніж 1000 інструментів, що дозволяє користувачам підтримувати ефективні робочі процеси.

Основні переваги ClickUp включають безлімітне сховище в платних планах, імпресивний набір функцій у безкоштовному плані, доступні ціни та зручний інтерфейс. Однак, через високу налаштовуваність та численні функції, існує крута крива навчання. Також відзначалися окремі випадки проблем з продуктивністю.

ClickUp пропонує кілька тарифних планів, включаючи безкоштовний план "Free Forever", який надає доступ до необмеженої кількості проектів та користувачів, але з обмеженням сховища до 100 МБ. План "Unlimited" коштує $7 на користувача на місяць і включає всі функції безкоштовного плану, а також

розширену звітність, необмежене сховище та інтеграції. План "Business" за $12 на користувача на місяць пропонує ще більше функцій, включаючи покращені можливості безпеки та управління ресурсами. Найвищий план "Enterprise" пропонується за індивідуальною ціною та включає всі функції бізнес-плану, а також додаткові функції, такі як біле маркування, Enterprise API, SSO та кероване впровадження.

ClickUp ідеально підходить для команд, які шукають високу налаштовуваність та різноманітність, а також для тих, хто використовує Agile методології. Він також зручний для організацій, яким потрібна високорівнева аналітика та звітність. Однак, його складність може бути перевантажуючою для нових користувачів або тих, хто шукає більш простий інтерфейс.

«Wrike»

Система управління проектами Wrike є потужним інструментом, що підходить для різних організацій та команд. Wrike пропонує широкий спектр функцій, які дозволяють керувати ресурсами, оптимізувати процеси, використовувати налаштовувані форми та автоматизувати робочі процеси.

Управління ресурсами: Wrike дозволяє ефективно розподіляти завдання між членами команди, встановлювати залежності та терміни виконання, а також відслідковувати прогрес у реальному часі. Функції управління робочим навантаженням, плануванням ресурсів і відстеженням часу дозволяють краще розподіляти та оптимізувати ресурси.

Оптимізація процесів: Wrike надає можливість налаштування робочого простору, автоматизації робочих процесів, комунікації в реальному часі та створення детальних звітів.

Настроювані форми та автоматизація робочих процесів: Wrike пропонує настроювані форми для заявок та інших потреб, а також автоматизовані робочі процеси, що забезпечують ефективність і знижують час на повторювані завдання.

Аналітика: Wrike надає можливість відстежувати різноманітні аналітичні дані проекту в реальному часі, що дозволяє керівникам проектів завжди бути в курсі поточного стану завдань, продуктивності команди та статусу проектів.

У Wrike є різні пакети та цінові категорії, що включають безкоштовний план, Професійний план, Бізнес-план та інші, зорієнтовані на конкретні потреби, як наприклад пакети для маркетингових команд та професійних послуг. Однак важливо відзначити, що деякі з найбільш передових функцій доступні лише в платних планах, і ціни Wrike можуть бути вищими у порівнянні з іншими альтернативами на ринку. Основні недоліки Wrike включають не надто зручні мобільні додатки та складність у використанні для початківців через велику кількість функцій.

У порівнянні з іншими інструментами управління проектами, Wrike виділяється своїми функціями управлінням ресурсами і можливістю налаштовування робочих процесів. Він також надає розширені можливості аналітики, які допомагають керівникам проектів приймати обґрунтовані рішення.

1.6.1. Зведена характеристика систем оцінювання знань

Отже, детально проаналізувавши три системи, створено зведену таблицю 1.2 для наглядності і простоти розуміння.

Таблиця 1.2. Порівняння систем аналогів

Критерії	**Asana**	**ClickUp**	**Wrike**
Вартість	Безкоштовний план для 15 користувачів; Платні плани від 10.99$/місяць за користувача	Безкоштовний план; Платні плани від 5$/місяць за користувача	Безкоштовний план для 5 користувачів; Платні плани від 9.80$/місяць за користувача

Продовження таблиці 1.2. Порівняння систем аналогів

Критерії	**Asana**	**ClickUp**	**Wrike**
Мова	Підтримує англійську	Підтримує багато мов, включаючи англійську	Підтримує англійську
Зручність використання	Висока, з можливістю доступу через браузер	Висока, з наявністю мобільного додатку	Висока, але зі складнішим інтерфейсом
Основні переваги	Колірні дошки, календарі, інтеграції	Гнучкість, широкий спектр функцій	Розширене управління ресурсами, автоматизація процесів
Основні недоліки	Обмежений функціонал безкоштовного плану	Може бути перевантаженим для нових користувачів	Вища ціна, складний для новачків

1.7. Обґрунтування доцільності проектування й розроблення системи для управління проєктами

Переваги розробки внутрішньої системи управління проєктами включають:

- точне налаштування під бізнес-процеси: можливість адаптувати систему під конкретні потреби бізнесу та робочі процеси;
- ексклюзивні функції: втілення функціоналу, який не доступний у стандартних рішеннях, таких як Trello або Jira, наприклад інтеграція з Telegram-ботом для отримання сповіщень про оновлення в проєктах;
- безпека та конфіденційність: повний контроль над даними та захист від зовнішніх ризиків;

- масштабування: здатність системи зростати та адаптуватися до змінюваних вимог підприємства;
- інтеграція з корпоративною культурою: відображення цінностей та принципів компанії у функціонуванні системи;
- створення єдиного інформаційного простору для спрощення доступу до інформації, необхідної для управління проектами та оптимізації бізнес-процесів.

Як висновок, розробка спеціалізованої системи управління проєктами дозволить підприємству точно налаштувати інструмент під власні процеси, забезпечити безпеку даних та легко інтегруватися з існуючими системами, а також позбутися лімітів, які аналоги такої системи накладають на користувачів. Ще однією важливою опцією може стати інтеграція ШІ для створення звітів на основі активності учасників проєктів. Така система покращить управління ресурсами, ефективність виконання завдань і, як наслідок, підвищить задоволеність як клієнтів, так і учасників команди.

1.8. Концептуальна модель системи

Концептуальна модель нашої інформаційної системи буде орієнтована на ефективне управління проєктами та задачами, аналогічно до функціональної моделі.

В роботу буде впроваджено веб-додаток, який робитиме зручнішим поставлення, контроль, перегляд задач:

- створення задач та проєктів: система дозволяє швидко створювати нові проєкти та задачі, вказуючи необхідні деталі та умови, визначені користувачем;
- зручний моніторинг задач за допомогою системи Kanban;
- перегляд активності учасників, задач та дошок.

Ця концептуальна модель розроблена з урахуванням подальшого вдосконалення та розширення. У майбутньому, можливо, буде реалізовано функції створення push-повідомлень за допомогою Telegram-бота, генерація звітів на основі активності учасників організації за допомогою ШІ.

1.9. Розрахунок економічного ефекту від впровадження ІС

1.9.1 Визначення загальних даних системи

Визначення економічного ефекту від впровадження системи є дуже важливим моментом, адже в його основі лежить техніко-економічне обґрунтування розробки автоматизованої системи.

Визначення розміру оплати праці

Джерелами прибутку від впровадження системи для управління проектами в продуктову компанію можуть бути такі фактори:

- економія часу та ресурсів через оптимізацію робочих процесів;
- зменшення витрат на управління та адміністрування проєктів;
- підвищення продуктивності команди, що дозволяє одночасно вести більше проєктів;
- скорочення часу на виконання завдань завдяки чіткій організації;
- зниження кількості помилок і простоїв у роботі;
- покращення якості продуктів та послуг, що може збільшити лояльність клієнтів.

Визначаємо ознаку – управління проєктами.

Ступінь новизни розроблюваних задач — "В" — використання типових проектних рішень за умови їх змін, розробка проектів, що мають аналогічні рішення.

Група складності алгоритму — 3.

Узагальнені дані вхідної та вихідної інформації для системи управління проектами в продуктовій компанії за видами вхідної та вихідної інформації таблиці 2.1.

Таблиця 1.3. Узагальнені дані для вхідної та вихідної інформації системи для управління проектами в продуктовій компанії

Вид інформації	**Позначення**	**К-сть наборів даних**
Змінна інформація	ЗІ	m=5
Нормативно – довідкова інформація	НДІ	n=5
Банк(база) даних	БД	p=1
Обробка в режимі реального часу	РЧ	Так
Забезпечення телекомунікаційної обробки даних і управління віддаленими об’єктами	ТОУ	Ні

Таблиця 1.4. Визначення витрат часу для системи управління проектами в продуктовій компанії

Вид системи	**Стадія розробки системи**			
	Ескізний проект (ПД), T_1		Технічне завдання, T_2	
	В	Г	В	Г
Управління організацією праці і зарплатою, управління кадрами, норми і нормативи, управління охороною праці				

Визначимо витрати часу на стадіях «технічний проект», «робочий проект» і «впровадження».

Вхідними даними для визначення є:

- кількість форм вхідної інформації 4;

- кількість форм вихідної інформації 2;
- базове значення витрат часу для стадії «Технічний проект» $T_{Б3}$=54;
- базове значення витрат часу для стадії «Робочий проект» $T_{Б4}$=131;
- базове значення витрат часу для стадії «Впровадження» $T_{Б5}$=48.

Базове значення витрат часу ТБ коригується за допомогою поправочних коефіцієнтів для всіх стадій розробки автоматизованої системи.

1.9.2. Визначення витрат часу для стадії «Технічний проект»

$$T_З = T_{БЗ} * k_п * k_о \quad (1)$$

$$k_п = \frac{k_1*m+k_2*n+k_3*p}{m+n+p} \quad (2)$$

Вид використаної інформаії	Ступінь новизни
	В
k_1 (ЗІ)	1.0
k_2 (НДІ)	0.72
k_3 (БД)	2.08

Рисунок 1.3 – Таблиця коефіцієнтів k_1, k_2, k_3 для стадії «Технічний проект»

Стадія розробки системи	Вид обробки	Ступінь новизни
		В
Технічний проект	РЧ	1.26
Робочий проект	РЧ	1.32
Впровадження	РЧ	1.21

Рисунок 1.4 – Коефіцієнти k_1, k_2, k_3 для стадії «Технічний проект»

$$k_п = \frac{(1*4+0.72*2+2.08*1)}{(4+2+1)} = \frac{7.52}{7} = 1.074 \quad (3)$$

$$T_3 = 54 * 1.074 * 1.26 = 73 \quad (4)$$

1.9.3. Визначення витрат часу на стадії «Робочий проект»

$$k_{\text{п}} = \frac{k_1 * m + k_2 * n + k_3 * p}{m + n + p} \quad (5)$$

Таблиця 1.5. Коефіцієнти k_1, k_2, k_3 для стадії «Робочий проект»

Вид використаної інформації	Ступінь новизни
	В
k_1(ЗІ)	1.0
k_2(НДІ)	0.48
k_3(БД)	0.40

$$k_{\text{п}} = \frac{(1.0*4+0.48*2+0.40*1)}{(4+2+1)} = 5.36/7 = 0.766 \quad (6)$$

$$T_4 = T_{\text{Б4}} * k_{\text{п}} * k_{\text{о}} * k_{\text{с}} \quad (7)$$

Для знаходження $k_{\text{с}}$ для формули необхідно ідентифікувати складність контролю вхідної та вихідної інформації.

Тобто $k_{\text{с}} = 1.0$

$$T_4 = 131 * 0.766 * 1.32 * 1.0 = 132.46 \quad (8)$$

1.9.4. Визначення витрат часу на стадії «впровадження»

$$k_{\text{п}} = \frac{k_1 * m + k_2 * n + k_3 * p}{m + n + p} \quad (9)$$

$$T_5 = T_{\text{Б5}} * k_{\text{п}} * k_{\text{о}} * k_{\text{с}} \quad (10)$$

$$T_5 = 48 * 0.766 * 1.21 * 1.0 = 44.5 \quad (11)$$

Отже, загальні витрати людської праці складають:

$$\text{T}_\Sigma = \text{T}_1 + \text{T}_2 + \text{T}_3 + \text{T}_4 + \text{T}_5 \quad (12)$$

$$\text{T}_\Sigma = 67 + 30 + 73 + 132.46 + 44.5 = 346.96 \quad (13)$$

Визначимо чисельність виконавців Ч:

$$Ч = \frac{Т_{\sum}}{Ф} \quad (14)$$

Якщо для виконання роботи припустимо кількість робочих годин складає 530 із 7-годинним робочим днем, тому на розробку проекту виділено Ф, днів:

Ф = 530/7 = 75 днів

Для курсової роботи Ф = 75 днів. Тоді визначаймо кількість місяців із розрахунку 25 робочих днів.

Кількість місяців на розробку, М: ***М = Ф/25 = 75/25 = 3 місяці***

Отже, для виконання такого проекту потрібно така чисельність виконавців Ч, яка обраховується за формулою: ***Ч = 346.96/75 = 5 виконавців***

Приймемо розмір заробітної плати програміста - 25000 грн, тоді загальна сума заробітних плат програмістів складає:

$$\boldsymbol{V_1'} = Ч * М * ЗП = \mathbf{5 * 3 * 25000 = 375000}\ \boldsymbol{грн} \quad (15)$$

1.9.5. Розрахунок річного фонду часу роботи ПК

Дійсний річний фонд часу ПК у годинах дорівнює числу робочих годин у році для оператора, за винятком часу на технічне обслуговування і ремонт ПК (в середньому 5год/міс + 6 роб. днів/рік).

$$Т_{ПК} = 2000-(6*8+5*12) = 1892\ _{год.} \quad (16)$$

Оскільки під час виконання курсової роботи здобувач в середньому витрачає 450 год. машинного часу, то величина фонду часу ПК дорівнює

$$Т'_{пк} = 1892*(450/2000) = 425.7\ _{год.} \quad (17)$$

1.9.6. Поточні витрати на експлуатацію V

Балансована вартість ПК, де Цр - ринкова вартість ПК, орієнтовно складає 40000 грн, $к_{ун}$ – коефіцієнт, що враховує витрати на установку ПК . $к_{ун}$=0,12

$$Ц_{ПК} = Ц_Р * (1 + k_{УН}) = 40000*(1+0,12) = 44800 \text{ грн} \quad (18)$$

Амортизаційні відрахування використання ПК, $З_{АМ,}$ обчислюються за формулою

$$З_{АМ} = \frac{Ц_{ПК}}{H_А} = 44800/5 = 8960 \text{ грн} \quad (19)$$

Витрати на електроенергію ($З_{ЕЛ}$), споживану ПК, обчислюються

$$З_{ЕЛ} = Р_{ПК} * Т_{ПК} * Ц_{ЕЛ} * А \quad (20)$$

де потужність ПК, $Р_{ПК} = 0.5$ кВт; фонд корисного часу роботи ПК, $Т_{ПК} = 435.16$ год, вартість 1 кВт електроенергії для підприємств, $Ц_{ЕЛ}$ = 1,86 грн/кВт, коефіцієнт інтенсивного використання ПК, А = 0.9.

$$З_{ЕЛ} = 0{,}5 * 435.16 * 1.68 * 0.9 = 329 \text{ грн} \quad (21)$$

Витрати на поточний ремонт і технічне обслуговування ПК ($З_Р$)визначаються як 6% від балансової вартості ПК, $Ц_{ПК}$.

$$З_р = Ц_{ПК} * 0.06 \quad (22)$$

$$З_р = 44800 * 0.06 = 2688 \text{ грн} \quad (23)$$

Непрямі витрати, пов’язані з експлуатацією ПК, визначаються як 5% від балансової вартості ПК $Ц_{ПК}$.

$$З_{МАТ} = Ц_{ПК} * 0.05 \quad (24)$$

$$З_{МАТ} = 44800 * 0.05 = 2240 \text{ грн} \quad (25)$$

Поточні витрати на експлуатацію V "

$$V''_1 = З_{ОП} + З_{АМ} + З_{ЕЛ} + З_Р + З_{МАТ} \quad (26)$$

Заробітна плата обслуговуючого персоналу складає в середньому - 10000

Тож, поточні витрати на експлуатацію, V_1'', грн, складають:

$$V_1'' = 10000 + 8960 + 329 + 2688 + 2240 = 24\,217 \text{ грн} \quad (27)$$

А, загальні витрати на розробку програмного забезпечення комп’ютерної системи складуть:

$$V_1 = V_1' + V_1'' = 375\,000 + 24\,217 = 399\,217 \text{ грн} \quad (28)$$

Розрахунок витрат на придбання і установку ПК

$$V_2 = Ц_{ПК} = 50\,000 \text{ грн} \quad (29)$$

1.9.7. Розрахунок витрат на підготовку приміщення і навчання персоналу

Витрати на підготовку приміщення $\mathbf{V_3 = 0}$, так як приміщення є в наявності. Витрати на навчання персоналу V_4. В середньому навчання персоналу триватиме 1 місяць, тому можна вважати, що $V_4 = 4500$ грн;

Загальна вартість розробки і впровадження системи вираховується за формуло:.

$$V_\Sigma = V_1 + V_2 + V_3 + V_4 \quad (30)$$

$$V_\Sigma = 399\,217 + 50\,000 + 0 + 4500 = 453\,717 \text{ грн} \quad (31)$$

Оскільки норма амортизаційних втрат для комп'ютерних систем НА = 5, то для обрахування річного економічного ефекту слід брати до розгляду величину:

$$V_р = \frac{V_\Sigma}{Н_А} \quad (32)$$

$$V_р = \frac{453\,717}{5} = 90\,743.4 \text{ грн} \quad (33)$$

Термін окупності розробки визначається:

$$Т_{ок} = \frac{\mathbf{1}}{К_{ЕФ}}, \quad (34)$$

де коефіцієнт економічної ефективності $К_{ЕФ} = \frac{П_р}{V_р}$, де річний прибуток $П_р$ від впровадження системи буде досягнуто за рахунок збільшення виконаних замовлень, і орієнтовно складатиме 19 000 грн на рік.

$$К_{ЕФ} = \frac{19\,000}{90\,743.4} = 0.209 \quad (35)$$

Отже, термін окупності ІС складатиме:

$$Т_{ок} = \frac{\mathbf{1}}{\mathbf{0.209}} = \mathbf{4.78} \text{ років} \quad (36)$$

1.10. Висновок

Як висновок, розробка спеціалізованої системи управління проєктами дозволить підприємству точно налаштувати інструмент під власні процеси, забезпечити безпеку даних та легко інтегруватися з існуючими системами, а також позбутися лімітів, які аналоги такої системи накладають на користувачів.

Ще однією важливою опцією може стати інтеграція ШІ для створення звітів на основі активності учасників проєктів. Така система покращить управління ресурсами, ефективність виконання завдань і, як наслідок, підвищить задоволеність як клієнтів, так і учасників команди.

Система повинна мати інтуїтивний дизайн, зрозумілий новачкам у сфері розробки, так як підприємство розширяється і набирає в штат багато нових співробітників без комерційного досвіду.

РОЗДІЛ 2. ТЕХНІЧНЕ ЗАВДАННЯ

2.1. Загальні відомості

Найменування системи: «Organizio».

Оформлення результатів робіт: результати робіт щодо створення системи оформлюються відповідно до вимог ДСТУ на кожному етапі розробки. Порядок оформлення та передачі результатів визначається згідно з календарним планом та змістом виконання робіт.

Уточнення та розширення: на наступних етапах створення системи може виникнути потреба в уточненні та розширенні окремих положень.

2.2. Призначення та цілі створення системи

Система управління проєктами «Organizio» – це комплексне програмне рішення, призначене для зручного управління, зберігання та обміну інформацією, включаючи конфіденційні дані, у відповідності з актуальними законодавчими вимогами.

2.2.1. Призначення системи:

Додаток призначений для створення єдиного інформаційного простору для спрощення доступу до інформації, необхідної для управління проектами та оптимізації бізнес-процесів та для зручного створення, керування, моніторингу задачами проєктів. А також забезпечення захисту даних згідно з найвищими стандартами безпеки.

2.2.2. Основні цілі «Organizio»:

1. Мінімізація паперового документообігу, зниження затрат часу та ресурсів на управління проектами.
2. Ефективне управління проектами та задачами через централізовану систему.
3. Підвищення продуктивності роботи шляхом забезпечення оперативного доступу до актуальної інформації.
4. Поліпшення координації між різними підрозділами.

2.3. Характеристика об’єкта автоматизації

Об’єктом інформатизації є діяльність відділу розробки ФОП «Горенко Тарас Анатолійович». Базовий об’єкт впровадження — ФОП «Горенко Тарас Анатолійович».

2.4. Вимоги до системи

2.4.1. Вимоги до системи в цілому.

2.4.1.1. Вимоги до структури і функціонування системи.

2.4.1.1.1. Система має використовувати клієнт-серверну архітектуру і працювати з єдиною базою даних.

2.4.1.1.2. Діагностика роботи системи в мережі відділу розробки передбачає виявлення відхилень в процесі вирішення завдань, проблем у функціонуванні комп’ютерно-технічних засобів і програмних помилок. Користувачам надаються відповідні діагностичні повідомлення.

Взаємодія між підсистемами здійснюється на рівні інформації за допомогою загальної БД та локальних комп'ютерних мереж.

2.4.1.1.3. Розвиток і модернізація системи мають здійснюватися шляхом уточнення, розширення або заміни функціоналу, а також модернізації технічних і програмних засобів з урахуванням впровадження нових поколінь комп'ютерів. Структура і технологія програмного забезпечення системи повинні бути спроектовані таким чином, щоб забезпечити простоту їх модернізації та розвитку, можливість збільшення обсягів оброблюваної інформації та розширення функціональності, а також можливість їх реалізації на нових персональних комп'ютерах.

Система повинна мати програмно-технічні засоби, які включають в себе програми з економіко-математичними та статистичними методами, методами моделювання, а також інструменти для табличного, текстового і графічного відображення даних. Щоб забезпечити сумісність програмного та інформаційного середовищ, слід використовувати загальносистемні протоколи обміну даними та проблемно-орієнтовані пакети прикладних програм для міжмашинних зв'язків, а також уніфіковану систему класифікації і кодування.

2.4.1.1.4. Система має забезпечувати можливість діалогового та мережного (розподіленого) оброблення даних.

2.4.1.2. Вимоги до складу та кваліфікації персоналу.

2.4.1.2.1. Персонал, який користується автоматизованою системою, повинен відповідати наступним вимогам:

- пройти навчання та здобути навички використання комп'ютера;
- суворо дотримуватися технологічних інструкцій при взаємодії з системою в діалоговому режимі;
- суворо дотримуватися умов експлуатації комп'ютера відповідно до інструкцій з експлуатації;
- зберігати інформацію та організовувати резервні копії бази даних відповідно до правил зберігання;
- дотримуватися правил техніки безпеки під час роботи з комп'ютером.

2.4.1.2.2. Користувачі системи можуть включати генерального директора, менеджера проєктів, дизайнерів, розробників та тестувальників. Доступ до системи здійснюється за допомогою входу в особистий акаунт, який відображає рівень привілеїв користувача. Залежно від рівня привілеїв, користувач може мати повний або обмежений доступ до системи.

2.4.1.3. Показники призначення.

2.4.1.3.1. Згідно з пунктом 2.1, показники призначення мають відображати ступінь та якість інформатизації планової, інформаційно-облікової та управлінської діяльності відділу розробки для досягнення його оптимального функціонування. Перелік та допустимі значення показників, за яких система забезпечує цільове призначення, мають бути визначені на етапі технічного проектування.

2.4.1.3.2. Система повинна мати можливість налаштування параметрів об’єкта управління та периферійного обладнання під час її модернізації та розвитку, а також під час зміни процесів та методів організаційного управління.

2.4.1.4. Вимоги до надійності.

2.4.1.4.1. Система є багатофункціональною та призначена для використання протягом робочого дня. Усі функції системи виконуються дискретно. Згідно з ДСТУ 2226-93, оцінка надійності проводиться окремо для кожної функції. Враховуючи особливості функціонування системи, показники її надійності є показниками надійності системи управління базами даних, на яких вона реалізована, та технічних засобів, на яких вона використовується. Основними показниками надійності є:

Li — ймовірність безвідмовного виконання задачі в заданий термін (імовірність того, що і-тий запит буде виконаний);

Kr — коефіцієнт готовності ПТК (програмно-технічного комплексу);

Тв — середній час відновлення ПТК;

Те — мінімальний час між двома відмовами за календарний місяць.

2.4.1.4.2. Комплекс технічних засобів повинен передбачати:

- можливість виконання функціональних завдань з будь-якої робочої станції та перехід до роботи в локальному режимі є обов'язковими функціями для забезпечення надійності програмного та інформаційного забезпечення.

Для забезпечення надійності програмного та інформаційного забезпечення необхідно використовувати такі заходи:

- використання модульного, структурного та об’єктно-орієнтованого програмування;
- використання програмних засобів контролю вхідної інформації, які надають користувачам повідомлення про виявлені помилки;
- використання програмних засобів коригування для виявлення та виправлення помилок у базі даних;
- використання засобів захисту від збоїв, несанкціонованого доступу, помилкових дій персоналу та інших потенційних загроз;
- регулярне створення резервних копій баз даних.

2.4.1.5. Вимоги до безпеки.

Для забезпечення безпеки при експлуатації, налагодженні, монтажі, обслуговуванні та ремонті технічних засобів системи слід дотримуватись вимог стандартів ДСТУ, таких як ДСТУ 2293:2014, ДСТУ EN ISO 7010:2019, ДСТУ 12.0.230:2008, ДСТУ 7237:2011, ДСТУ 7238:2011, ДСТУ 7239:2011. Також слід дотримуватись вимог до доступного рівня освітленості, вібраційних і шумових навантажень, які встановлені відповідними стандартами, наприклад ДСТУ Б А.3.2-15:2011, ДСТУ EN 14253:2018, ДСТУ 2867-94.

2.4.1.6. Вимоги до ергономіки та технічної естетики.

Загальні ергономічні та естетичні вимоги до системи повинні відповідати державним стандартам, таким як ДСТУ 8604:2015, ДСТУ 7298:2013. Освітленість робочого місця має відповідати відповідним стандартам, наприклад ДСТУ EN 12464-1:2016, ДБН В.2.5-28:2018.

Засоби відображення повинні розташовуватися так, щоб кут спостереження екрану складав не більше 45 градусів, а мінімальна відстань спостереження екрану має бути не менше 0,3 м, рекомендована - 0,5 м.

При розробці програмного забезпечення слід створити зручний інтерфейс, який запобігатиме втомленості користувача.

2.4.1.7. Вимоги до експлуатації, технічного обслуговування, ремонту та зберігання компонентів системи:

2.4.1.7.1. Види обслуговування системи повинні відповідати стандартам ДСТУ EN 13306:2019, а загальні вимоги до експлуатації, технічного обслуговування і ремонту мають відповідати ДСТУ 3576-97.

2.4.1.7.2. Для розміщення технічних засобів системи необхідні площі, що визначені в ДБН В.2.2-9:2018. Вимоги до напруги живлення технічних засобів системи - 220/380 В змінного струму, частотою (50±1) Гц. Перерви у живленні не повинні перевищувати 0,001 с.

2.4.1.7.3. Кількість, кваліфікація і режими роботи обслуговуючого персоналу повинні відповідати рекомендаціям, вказаним в технічних умовах і інструкціях з експлуатації окремих технічних засобів.

2.4.1.7.4. Склад, розміщення і умови зберігання компонентів технічних засобів системи повинні відповідати рекомендаціям, зазначеним в експлуатаційній документації на ці елементи.

2.4.1.7.5. Регламент обслуговування повинен відповідати рівню та умовам роботи, щоб у випадку відмови системи забезпечити її роботу в аварійному режимі.

2.4.1.8. Вимоги до захисту інформації від несанкціонованого доступу:

Для надійного зберігання і доступу до інформації потрібно використовувати такі засоби захисту:

- операційні системи серверів;
- веб-сертифікат безпеки SSL;
- локальну мережу та програмне забезпечення захисту в мережі Firewall;
- клієнт-серверну систему управління базами даних, яка включає тригери, представлення, процедури та функції, а також встановлення груп користувачів і ролей використання.

Кожен сеанс роботи системи повинен починатися з введення індивідуального пароля. Система парольного захисту повинна мати власні засоби періодичної зміни паролів або використовувати стандартні засоби середовища розроблення. Для надійного захисту від несанкціонованого доступу кожен працівник повинен мати персональний пароль. Деякі таблиці також слід захистити від можливого редагування, доповнення або вилучення інформації.

2.4.1.9. Вимоги щодо збереження інформації при аваріях.

2.4.1.9.1. Після внесення змін до бази даних, необхідно мати можливість зберігати резервну копію в архіві та відновлювати базу даних із цього архіву в разі її пошкодження.

2.4.1.9.2. Резервний архів та база даних повинні бути збережені на різних носіях або пристроях, щоб забезпечити додатковий рівень захисту від втрати даних.

2.4.1.10. Вимоги щодо захисту від зовнішніх впливів:

2.4.1.10.1. Електричне поле не повинно перевищувати 0,3 В/м² в діапазоні частот від 0,15 до 300 МГц. Для захисту від електромагнітних полів та промислових завад передбачаються різноманітні екрани та фільтри.

2.4.1.10.2. Засоби, які забезпечують захист від шкідливих впливів на функціонування технічних засобів, повинні відповідати вимогам ДБН В.2.2-9-2009, а комп'ютерні системи повинні відповідати стандарту ДСТУ 2506-94 щодо стійкості до зовнішніх впливів.

2.4.1.11. Вимоги до патентної чистоти:

Не проводяться патентні дослідження в рамках створення даної системи.

2.4.1.12. Вимоги до стандартизації та уніфікації:

У системі кодування інформації потрібно дотримуватися світового класифікатора і стандарту.

2.4.2. Вимоги до функцій:

2.4.2.1. Перелік функцій, разом з вхідною та вихідною інформацією, наведений у таблиці 2.1. Функції повинні забезпечити ефективну організацію роботи користувача на основі безперервної технології, включаючи заповнення бази даних, формування звітів та інші функції, зручні для користувача за допомогою підказок та меню на екрані.

Таблиця 2.1. Таблиця функціональної інтеграції системи «Organizio»

№ з/п	**Найменування інтеграції**	**Вхідна інформація**	**Вихідна інформація**
1	Управління завданнями	Запити від користувачів, технічне завдання	Звіти про виконання, оновлені статуси завдань

Продовження таблиці 2.1. Таблиця функціональної інтеграції системи «Organizio»

№ з/п	Найменування інтеграції	Вхідна інформація	Вихідна інформація
2	Контроль доступу	Реєстраційні дані користувачів, правила доступу	Логи доступу, управління рівнями доступу
3	Формування та виведення дошок, задач (карток)	Дані про проєкт, дані про користувача	Інтерфейс дошки проєкту

2.4.3.1. Вимоги до забезпечення:

2.4.3.1.1. Система не потребує спеціалізованого математичного забезпечення для виконання своїх функцій, оскільки вона використовує обрану систему управління базами даних.

2.4.3.2. Вимоги до інформаційного забезпечення:

2.4.3.2.1. Інформаційне забезпечення системи повинно включати достатні дані для виконання всіх функцій. Раціональна організація зберігання та доступу до інформації гарантується, а заповнення бази даних відбувається замовником за визначеними методиками та формами.

2.4.3.2.2. Запобігання втраті даних у разі аварій та порушень у електроживленні передбачає використання резервних копій баз даних.

2.4.3.3. Вимоги до лінгвістичного забезпечення:

2.4.3.3.1. Для програмного забезпечення, що відповідає за функції та обслуговування користувачів, використовуються мови високого рівня для створення програм і мова обраної системи управління базами даних для доступу та маніпулювання даними.

2.4.3.3.2. Взаємодія користувача з системою базується на наборах меню та підказок, а запити користувача надсилаються переважно природною мовою.

2.4.3.4. Вимоги до програмного забезпечення:

2.4.3.4.1. Загальне програмне забезпечення повинно забезпечувати надійне та якісне виконання функцій системи. До нього входять операційна система Windows та система управління базами даних MySQL.

2.4.3.4.2. Основні вимоги до системного програмного забезпечення включають мінімальне використання ресурсів технічних засобів, максимальну швидкодію та повне задоволення функціональних потреб системи.

2.4.3.4.3. Вимоги до операційної системи включають мінімізацію використання ресурсів комп'ютера та максимальну швидкодію при управлінні зовнішніми пристроями.

2.4.3.4.4. Вимоги до СУБД:

- максимальна відповідність функціональним завданням;
- забезпечення надійності;
- ефективне керування необхідним обсягом та структурою;
- швидкість обробки запитів користувачів;
- мінімізація вимог до технічних засобів.

2.4.3.4.5. Програмні засоби введення та виведення даних і ведення діалогу повинні забезпечувати:

- відображення потрібної інформації на екрані у формі відповідних візуальних зображень;
- забезпечення контролю та сигналізації про можливі помилки при введенні даних, а також можливість їх корекції під час введення;
- взаємодія з комп'ютером у процесі введення даних через управління діалогом;
- виведення даних у потрібному форматі (наприклад, у формі документа) відповідно до запиту користувача.

2.4.3.4.6. При розробленні спеціального ПЗ слід виконати наступні вимоги:

Використані програми мають бути сумісні між собою та із загальносистемним ПЗ:

- ПЗ має розроблятися засобами об’єктно-орієнтованого програмування;

- забезпечити відповідність веб-інтерфейсу користувача стандартам браузерів;
- необхідна модульна структура програм;
- повинна бути передбачена можливість розширення складу задач у відповідності з новими функціональними потребами;
- ПЗ не повинно залежати від типу зовнішніх пристроїв (принтерів, дисків, сканерів тощо);
- діалог із користувачем повинен проводитись за допомогою клавіатури або миші з поясненням виконання дій і можливістю отримання підказки.

2.4.3.5. Вимоги до технічного забезпечення.

2.4.3.5.1. Технічні засоби системи (табл. 2.2) повинні забезпечувати виконання функцій, перерахованих в таблиці 2.1.

2.4.3.5.2. Засоби обчислювальної техніки повинні забезпечувати обмін інформації в об’ємах, приведених в п. 2.4.3.2.

Таблиця 2.2. Вимоги до технічного забезпечення системи

№ п/п	**Основні характеристики комп’ютера**	
	Технічне забезпечення для сервера	
1	Процесор	Intel Xeon E5-2670 або аналогічний (2.6GHz, 8 ядер)
2	Оперативна пам'ять	32GB DDR4
3	Жорсткий диск	SSD 1TB
4	Операційна система	Ubuntu Server 20.04 LTS або інша дистрибуція Linux
5	Мережевий інтерфейс	Gigabit Ethernet

Продовження таблиці 2.2. Вимоги до технічного забезпечення системи

№ п/п	**Основні характеристики комп'ютера**	
	Технічне забезпечення для клієнта	
1	Процесор	Intel Core i5-8250U або аналогічний (1.6GHz, 4 ядра)
2	Оперативна пам'ять	8GB DDR4
3	Жорсткий диск	SSD 256GB
4	Операційна система	Windows 10 або MacOS
5	Браузер	Google Chrome, Mozilla Firefox або Microsoft Edge

2.4.3.6. Вимоги до метрологічного забезпечення.

Система не має вимірювальних каналів, вимірювального обладнання і приладів, тому вимоги до даного виду забезпечення не висуваються.

2.4.3.7. Вимоги до організаційного забезпечення.

2.4.3.7.1. Організаційне забезпечення системи розробляється відповідно до державних стандартів щодо автоматизованих систем управління.

2.4.3.7.2. Впровадження системи не передбачає збільшення штату працівників підприємства. Розміщення робочих місць, де буде встановлена система, визначається підприємством.

2.4.3.7.3. При функціонуванні системи встановлені наступні вимоги:

- склад співробітників, яким надається доступ до системи, визначається наказом власника;
- відповідальний за систему забезпечує контроль та прийняття рішень у випадку аварійних ситуацій під час експлуатації.

Загальні вимоги можна переглянути в таблиці 2.2, функціональні – в таблиці 2.3, технічні – в таблиці 2.4.

Таблиця 2.2. Таблиця загальних вимог

№	Вимога	Деталізація
1	Сумісність	Підтримка Windows, macOS, Linux; оптимізовано для мобільних пристроїв
2	Масштабованість	Підтримка від 10 до 1000+ користувачів без зниження продуктивності
3	Інтегрованість	API для інтеграції з зовнішніми системами
4	Універсальна мова	Підтримка англійської мови

Таблиця 2.3. Таблиця функціональних вимог

№	Функціональність	Опис
1	Управління завданнями	Інструменти для планування, відстеження, аналізу завдань і проектів
2	Контроль доступу	Різні рівні доступу для різних користувачів, відповідно до їх ролей у проекті

Таблиця 2.4. Таблиця технічних вимог

№	Технічна характеристика	Вимоги
1	Висока наявність	99,9% часу доступності, мінімізація часу недоступності
2	Резервне копіювання	Автоматичне резервне копіювання даних з можливістю швидкого відновлення
3	Моніторинг	Ведення детального логу дій користувачів, системних подій та аудит безпеки

Ці детальні таблиці дозволяють чітко визначити ключові характеристики і вимоги до системи "Organizio", що забезпечить ефективне управління проектами та бізнес-процесами на підприємстві.

2.5. Склад та зміст робіт щодо створення системи

2.5.1. Визначення етапів:

2.5.1.1. Етап планування та концептуалізації:

- планування проекту та аналіз вимог: 15.12.2023;
- розробка концептуального проекту: 15.01.2024.

2.5.1.2. Етап розробки та тестування:

- розробка системи: 16.01.2024;
- тестування функціональності та безпеки: 15.04.2023.

2.5.1.3. Етап впровадження та дослідної експлуатації:

- впровадження системи в дослідну експлуатацію: 16.04.2023;
- моніторинг та оптимізація: 10.07.2024.

2.5.1.4. Етап повної виробничої експлуатації:

- перехід до повної виробничої експлуатації: 12.07.2024.

2.5.2. Перелік документів, що надаються після завершення відповідних етапів:

2.5.2.1. Концептуальний проект:

- технічні специфікації;
- дизайн-проект інтерфейсу користувача.

2.5.2.2. Інструкції користувачів системи:

- інструкція для кінцевих користувачів;
- інструкція для адміністраторів системи.

2.5.2.3. Інструкція оператора системи:

- посібник з експлуатації системи.

2.5.2.4. Інструкція адміністратора системи:

- інструкція по налаштуванню та управлінню системою.

2.6. Порядок контролю та приймання системи

2.6.1. Випробування системи:

- тестування елементів системи: перевірка кожного модуля на відповідність заданим функціональним та технічним вимогам;
- тестування системи у цілому: інтеграційні тести для перевірки взаємодії між модулями та системами;
- дослідна експлуатація: використання системи в контрольованому середовищі з обмеженим набором користувачів для виявлення помилок та неефективностей;
- додаткові випробування КСЗІ: випробування системи на відповідність стандартам безпеки, включно з тестуванням на проникнення та аудитом безпеки.

2.6.2. Загальні вимоги до приймання робіт за стадіями:

- після закінчення кожного етапу робіт виконується оцінка результатів на відповідність технічному завданню;
- формується звітна документація, що включає результати тестувань і випробувань;
- замовник і виконавець підписують протокол про приймання робіт, що підтверджує завершення етапу та готовність перейти до наступного.

2.7. Вимоги до складу і змісту робіт щодо підготовки об’єкта автоматизації до введення системи у дію

2.7.1. Умови функціонування:

- передбачити необхідні технічні та програмні умови для ефективної роботи системи, включаючи серверне обладнання, мережеві рішення та робочі станції;
- забезпечити належний захист даних, в тому числі шифрування та системи виявлення та запобігання вторгнень (IDS/IPS).

2.7.2. Оргструктура:

- створення команди підтримки, включаючи Service Desk для обробки запитів користувачів та вирішення інцидентів;

- організація навчальних програм для персоналу, який буде використовувати систему, з метою забезпечення ефективного впровадження та експлуатації системи.

2.7.3. Дослідна експлуатація та навчання:

- планування та виконання дослідної експлуатації системи з метою ідентифікації та усунення потенційних проблем перед введенням в повноцінну експлуатацію;
- розробка та реалізація навчальних програм для користувачів та IT-спеціалістів, що включає руководства, відеоуроки та практичні заняття.

2.8. Вимоги до документування

Документація системи "Organizio" повинна бути виконана згідно з національними стандартами та міжнародними нормами, які забезпечують чіткість, точність та зручність у використанні. Вона повинна включати:

2.8.1. Технічну документацію:

- архітектурні схеми системи;
- технічні специфікації та описи модулів;
- вимоги до апаратного та програмного забезпечення.

2.8.2. Експлуатаційну документацію:

- інструкції з інсталяції та налаштування системи;
- керівництва для кінцевих користувачів;
- посібники для системних адміністраторів.

2.8.3. Документацію з обслуговування:

- інструкції з технічного обслуговування;
- процедури виявлення та усунення збоїв;
- плани регулярного технічного обслуговування.

2.8.4. Документацію для розробників:

- API документація та розробницькі інструкції;
- рекомендації щодо розширення функціоналу системи.

2.9. Джерело розробки

Розробка системи "Organizio" повинна базуватися на наукових та практичних знаннях у сфері управління проектами та бізнес-процесами. Це включає:

2.9.1. Попередні розробки:

- аналіз існуючих рішень на ринку;
- використання передового досвіду інших розробників та організацій.

2.9.2. Співпраця з експертами:

- консультації з фахівцями з управління проектами;
- залучення експертів з інформаційної безпеки.

2.9.3. Науковий підхід:

- дослідження найкращих практик та академічних робіт у сфері;
- впровадження інноваційних технологій та методологій.

2.9.4. Підтримка спільноти:

- взаємодія з користувацькими спільнотами для збору зворотного зв'язку;
- бета-тестування з ранніми користувачами для поліпшення продукту.

РОЗДІЛ 3. ОПИС КОМПЛЕКСУ ЗАДАЧ АВТОМАТИЗАЦІЇ

3.1. Інформаційне забезпечення системи

Створення додатку здійснювалось за допомогою таких інструментів та технологій:

1. **Next.js**: Використовувався як основний фреймворк для розробки серверно-клієнтного додатку. Це забезпечує ефективне рендеринг на стороні сервера та клієнта, а також підтримку маршрутизації та API-роутів.
2. **Prisma**: ORM для взаємодії з базою даних, забезпечує простий спосіб роботи з базою даних за допомогою моделей і схем. Використовувався для управління базою даних, міграцій і генерації клієнтського коду для взаємодії з базою даних.
3. **Clerk**: Використовувався для аутентифікації користувачів, надаючи готові рішення для реєстрації, входу та управління користувачами.
4. **React та ReactDOM**: Бібліотеки для створення компонентного інтерфейсу користувача. Вони забезпечують основу для побудови інтерактивних користувацьких інтерфейсів.
5. **Radix UI**: Набір компонентів для створення інтерфейсу користувача, що включає компоненти для діалогів, спливаючих підказок, аватарів та інших елементів інтерфейсу.
6. **React Query**: Бібліотека для управління запитами даних у React-додатках, що забезпечує просте та ефективне завантаження, кешування та синхронізацію даних.
7. **Tailwind CSS**: Утилітарний CSS-фреймворк для швидкої розробки стильових компонентів. Використовувався для створення адаптивних та стильних інтерфейсів користувача.
8. **Zod**: Бібліотека для перевірки та парсингу схем, що використовується для валідації даних у додатку.
9. **Hello Pangea DND**: Бібліотека для реалізації перетягування та скидання елементів у інтерфейсі, що дозволяє користувачам взаємодіяти з елементами шляхом їх переміщення.

10. **Sonner**: Бібліотека для сповіщень у React-додатках, що забезпечує зручний спосіб інформування користувачів про різні події та стани.
11. **Unsplash JS**: Клієнт для доступу до API Unsplash, що дозволяє завантажувати зображення для використання у додатку.
12. **TypeScript**: Мова програмування з статичною типізацією, яка використовується для покращення якості коду та забезпечення його стійкості.

3.1.1. Авторизація.

В проєкті також було використано Clerk для забезпечення функціоналу аутентифікації користувачів.

Clerk надає готові рішення для аутентифікації та авторизації користувачів. Це включає в себе такі функції, як реєстрація, вхід, відновлення пароля, а також підтримку соціальних входів через популярні платформи (наприклад, Google, Facebook).

Clerk дозволяє легко реалізувати контроль доступу до різних частин додатку, базуючись на ролях та дозволах користувачів. Це забезпечує захист чутливих даних та функціоналу від несанкціонованого доступу.

Цей інструмент забезпечує масштабованість системи аутентифікації, дозволяючи обробляти велику кількість користувачів без втрати продуктивності. Це особливо важливо для додатків з високою навантаженістю.

Сервіс легко інтегрується з різними фреймворками та бібліотеками, такими як Next.js, що використовувався в цьому проєкті. Це значно спрощує процес впровадження аутентифікації та економить час розробників.

Clerk надає розширені засоби безпеки, включаючи захист від атак типу brute force, багаторазову аутентифікацію (MFA), а також шифрування даних аутентифікації. Це забезпечує високий рівень безпеки для користувачів додатку.

Таким чином, використання Clerk в цьому проєкті дозволило реалізувати надійний та безпечний функціонал аутентифікації та авторизації, забезпечуючи захист даних користувачів та спрощуючи управління доступом до різних частин додатку.

В проєкті було використано Prisma та Aiven для забезпечення ефективного управління даними та хостингу бази даних.

3.1.2. Використання Prisma.

Prisma було обрано як ORM для управління базою даних. Це дозволяє розробникам працювати з базою даних за допомогою об'єктів, що значно спрощує процес розробки та підтримки коду. Prisma генерує типізовані клієнти на основі схеми бази даних, що допомагає уникати помилок та підвищує продуктивність розробників.

3.1.3. Використання Aiven.

Aiven надає керовані сервіси для різних баз даних, включаючи MySQL, що було використано в цьому проєкті. Використання Aiven для хостингу бази даних забезпечує надійність та масштабованість, а також знімає необхідність в адмініструванні інфраструктури бази даних.

Висока доступність та резервне копіювання: Aiven забезпечує високу доступність баз даних через функції резервного копіювання та відновлення. Це гарантує, що дані залишаються захищеними та доступними навіть у випадку збоїв.

Aiven надає розширені засоби безпеки, включаючи шифрування даних як при передачі, так і в стані спокою. Це забезпечує захист конфіденційних даних та відповідність стандартам безпеки.

Отже, використання Prisma та Aiven в цьому проєкті забезпечує ефективне управління базою даних, високу надійність та безпеку, а також спрощує процес розробки завдяки типізованому ORM та керованим сервісам хостингу.

Таблиці бази даних з назвами полів, їхніми типами та ключами можна переглянути в таблицях 3.1, 3.2, 3.3, 3.4.

Таблиця 3.1. Таблиця «Board»

Поле	**Тип даних**	**Ключі**	**Опис**
id	VARCHAR(36)	PRIMARY KEY, DEFAULT UUID()	Унікальний ідентифікатор дошки
orgId	VARCHAR(255)		Ідентифікатор організації
title	VARCHAR(255)		Назва дошки
imageId	VARCHAR(255)		Ідентифікатор зображення
imageThumbUrl	TEXT		URL мініатюри зображення
imageFullUrl	TEXT		Повний URL зображення
imageUserName	TEXT		Ім'я користувача, який завантажив зображення
imageLinkHTML	TEXT		HTML посилання на зображення
createdAt	DATETIME	DEFAULT CURRENT_TIMESTAMP	Дата та час створення дошки

Продовження таблиці 3.1. Таблиця «Board»

Поле	**Тип даних**	**Ключі**	**Опис**
updatedAt	DATETIME	DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	Дата та час останнього оновлення дошки

Таблиця 3.2. Таблиця «List»

Поле	**Тип даних**	**Ключі**	**Опис**
id	VARCHAR(36)	PRIMARY KEY, DEFAULT UUID()	Унікальний ідентифікатор списку
title	VARCHAR(255)		Назва списку
order	INT		Порядок списку
boardId	VARCHAR(36)	FOREIGN KEY REFERENCES Board(id) ON DELETE CASCADE, INDEX	Ідентифікатор дошки
createdAt	DATETIME	DEFAULT CURRENT_TIMESTAMP	Дата та час створення списку
updatedAt	DATETIME	DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	Дата та час останнього оновлення списку

Таблиця 3.3. Таблиця «Card»

Поле	**Тип даних**	**Ключі**	**Опис**
id	VARCHAR(36)	PRIMARY KEY, DEFAULT UUID()	Унікальний ідентифікатор картки
title	VARCHAR(255)		Назва картки
order	INT		Порядок картки
description	TEXT		Опис картки (необов'язкове поле)
listId	VARCHAR(36)	FOREIGN KEY REFERENCES List(id) ON DELETE CASCADE, INDEX	Ідентифікатор списку
createdAt	DATETIME	DEFAULT CURRENT_TIMESTAMP	Дата та час створення картки
updatedAt	DATETIME	DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	Дата та час останнього оновлення картки

Таблиця 3.4. Таблиця «AuditLog»

Поле	**Тип даних**	**Ключі**	**Опис**
id	VARCHAR(36)	PRIMARY KEY, DEFAULT UUID()	Унікальний ідентифікатор запису

Продовження таблиці 3.4. Таблиця «AuditLog»

Поле	**Тип даних**	**Ключі**	**Опис**
orgId	VARCHAR(255)		Ідентифікатор організації
action	ENUM('CREATE', 'UPDATE', 'DELETE')		Дія (CREATE, UPDATE, DELETE)
entityId	VARCHAR(36)		Ідентифікатор сутності
entityType	ENUM('BOARD', 'LIST', 'CARD')		Тип сутності (BOARD, LIST, CARD)
entityTitle	VARCHAR(255)		Назва сутності
userId	VARCHAR(255)		Ідентифікатор користувача
userImage	TEXT		URL зображення користувача
userName	TEXT		Ім'я користувача
createdAt	DATETIME	DEFAULT CURRENT_TIMESTAMP	Дата та час створення запису
updatedAt	DATETIME	DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	Дата та час останнього оновлення запису

List.**boardId**: Індекс та зовнішній ключ, що посилається на **Board(id)**.

Card.listId: Індекс та зовнішній ключ, що посилається на **List(id)**.

Таблиця **Board** містить інформацію про дошки, які представляють проекти або категорії в системі управління завданнями.

Таблиця **List** зберігає списки задач, що належать до конкретних дошок. Кожен список представляє собою категорію або етап у проекті.

Таблиця **Card** містить задачі або картки, що належать до конкретних списків. Кожна картка представляє собою окреме завдання або елемент роботи.

Таблиця **AuditLog** використовується для зберігання записів про дії користувачів у системі. Вона дозволяє відстежувати зміни та дії, виконані над дошками, списками та картками.

Розглянемо створення бази даних за допомогою Aiven та Prisma.

Aiven пропонує безкоштовний варіант MySQL сервісу, який нам більше, ніж підходить. Його конфігурацію можна розгялнути на рисунку 3.1.

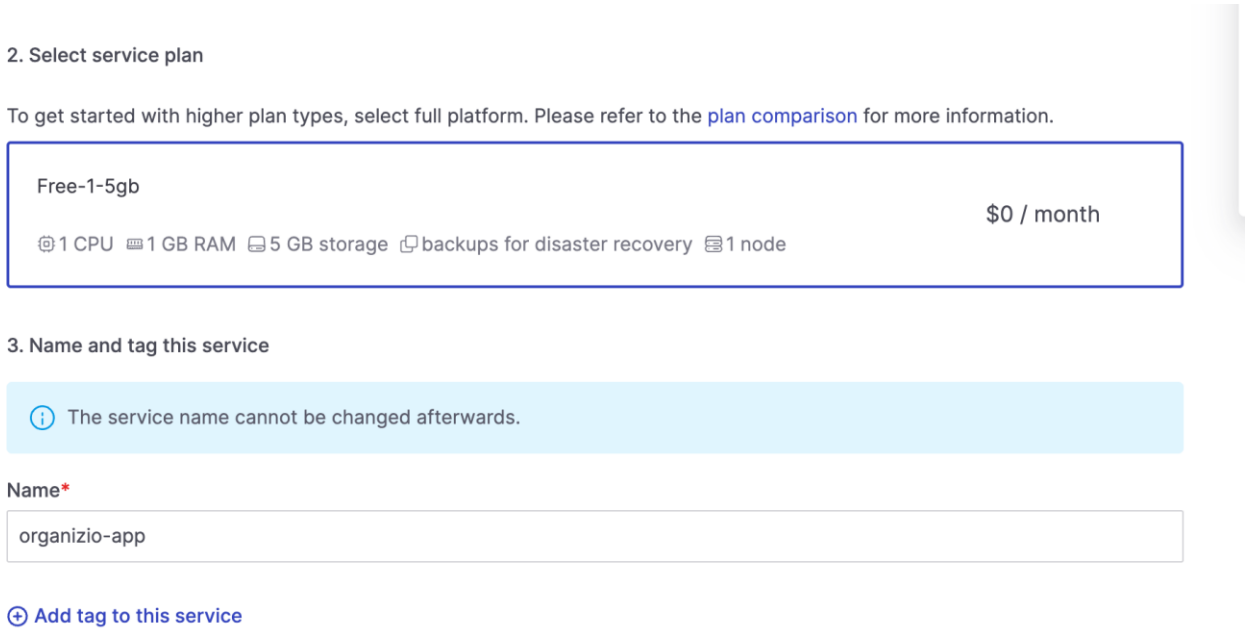

Рисунок 3.1 – Конфігурація безкоштовного сервера в Aiven

Після створення сховища нам дають всі необхідні дані для підключення (рисунок 3.2). Нам потрібен тільки Service URI, який включає в себе всі інші поля, такі як: назва бази даних, хост, порт, пароль, ssl режим. Це поле ми копіюємо та записуємо в файл «.env», де зберігаються всі секретні змінні, які повинні бути обмежені від публічного доступу.

Service URI	mysql://CLICK_TO:REVEAL_PASSWORD@organizio-app-organizio-app.d.aivencloud.com:28826/defaultdb?ssl-mode=REQUIRED
Database name	defaultdb
Host	organizio-app-organizio-app.d.aivencloud.com
Port	28826
User	avnadmin
Password	**********
SSL mode	REQUIRED
CA certificate	Show

Рисунок 3.2 – Згенеровані дані для підключення до бази даних

Модель бази даних та всі залежності описано в файлі «prisma/schema.prisma», код якого можна переглянути в додатку В.10.

Після опису схеми та будь-яких змін в ній, потрібно оновити БД на сервері. Це робиться за допомогою CLI від Prisma. Необхідно написати команди: ***npx prisma db push*** та ***npx prisma generate***.

Для зручної візуалізації бази даних було встановлено додатковий пакет «Prisma Studio». Запустити утиліту можна за допомогою команди ***npx prisma studio***, після чого відкриється нова вкладка в браузері (рис. 3.3), де відображаються всі таблиці, їх поля та записи (рис. 3.4).

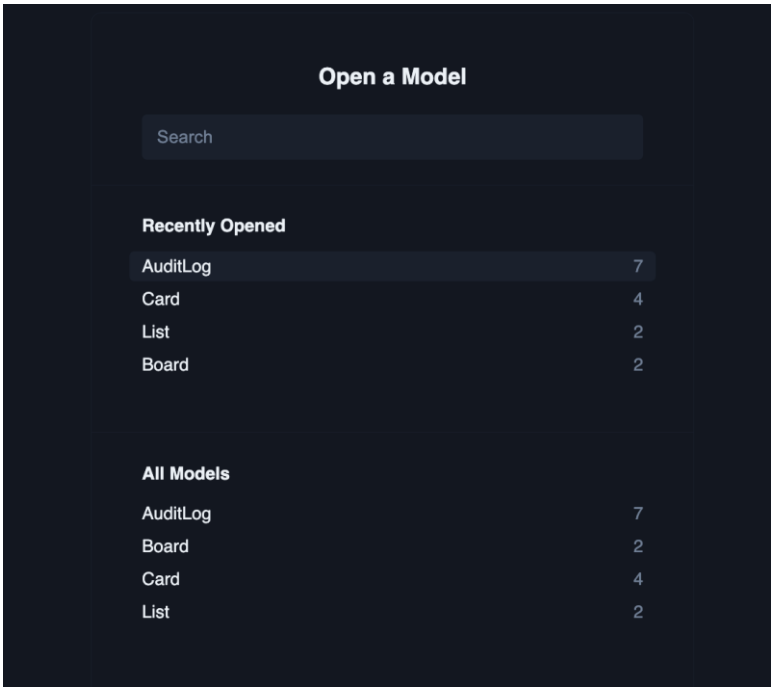

Рисунок 3.3 - Головний екран Prisma Studio

Рисунок 3.4 - Відкрита таблиця Card

Для маршрутизації користувачів було написано middleware, який відбувається між запитами. Цей middleware використовується для управління аутентифікацією та маршрутизацією користувачів у додатку.

Він визначає публічні та приватні маршрути, а також обробляє поведінку користувачів в залежності від їх стану аутентифікації та наявності організації.

Якщо користувач залогінений і намагається потрапити на публічний маршрут (наприклад, головну сторінку), його перенаправляють на вибір організації або безпосередньо до його організації, якщо вона вже існує.

Незалогінені користувачі, які намагаються доступитися до непублічних маршрутів, перенаправляються на сторінку входу. Після успішного входу вони повертаються до сторінки, з якої прийшли. Користувачі, які залогінені, але не мають організації, перенаправляються на сторінку вибору організації, щоб забезпечити наявність організації для подальшої роботи.

Нижче представлено код даного middleware:

```
import { authMiddleware, redirectToSignIn } from '@clerk/nextjs';
import { NextResponse } from 'next/server';
export default authMiddleware({
  publicRoutes: ['/'],
  afterAuth(auth, req) {
    if (auth.userId && auth.isPublicRoute) {
      let path = '/select-org';
      if (auth.orgId) {
        path = `/organization/${auth.orgId}`;
      }

      const orgSelectionUrl = new URL(path, req.url);
```

```
        return NextResponse.redirect(orgSelectionUrl);
      }
      if (!auth.userId && !auth.isPublicRoute) {
        return redirectToSignIn({ returnBackUrl: req.url });
      }
      if (auth.userId && !auth.orgId && req.nextUrl.pathname !== '/select-org') {
        const orgSelectionUrl = new URL('/select-org', req.url);
        return orgSelectionUrl;
      }
    },
});
export const config = {
    matcher: ['/((?!.+\\.[\\w]+$|_next).*)', '/', '/(api|trpc)(.*)'],
};
```

3.2. Алгоритмізація та реалізація комплексу задач автоматизації

Програма логічно розділена на дві частини: презентаційну (лендінг) та платформу (дашборд) (рис. 3.5). Якщо користувач не залогінений в системі, йому в будь-якому випадку відкриватиметься презентаційна сторінка, де відображені кнопки для реєстрації та входу.

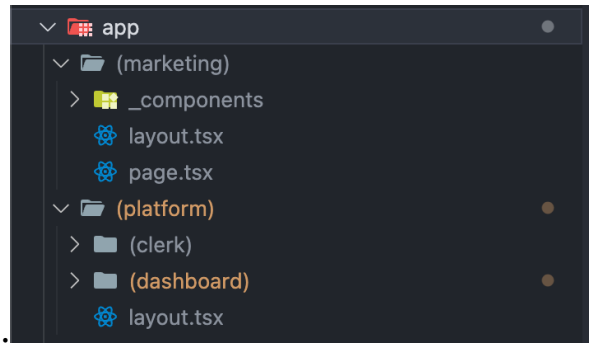

Рисунок 3.5 – Структура папки додатку.

3.2.1. Авторизація за допомогою Clerk

Існують 3 варіанти авторизації:

1. GitHub
2. Google
3. Ручна реєстрація за допомогою логіна та пароля.

Код сторінки входу, файл page.tsx:

```
import { SignIn } from "@clerk/nextjs";
export default function Page() {
  return <SignIn />;
}
```

Та код сторінки реєстрації:

```
import { SignUp } from "@clerk/nextjs";
export default function Page() {
  return <SignUp />;
}
```

Як можна побачити, коду дуже мало. Ми рендеримо готовий компонент, а саме форму, який надає нам Clerk.

Після входу в систему користувача зустрічає форма вибору організації. Можна створити нову або обрати зі списку. В коді сторінки вибору організації можна побачити атрибути «afterSelectOrganizationUrl» та «afterCreateOrganizationUrl». Вони призначені для редіректу користувача після того, як він обере або створить організацію відповідно:

```
import { OrganizationList } from '@clerk/nextjs';
export default function CreateOrganizationPage() {
   return (
      <OrganizationList
         hidePersonal
         afterSelectOrganizationUrl="/organization/:id"
         afterCreateOrganizationUrl="/organization/:id"
      />)}
```

3.2.2. Адміністративна панель організації

Адміністратор може переглянути всі створені організації (див. рис. 3.6) та користувачів (див. рис. 3.7), а також керувати ними в особистому кабінеті Clerk. Також там він може створювати та надавати певні ролі для користувачів (див. рис. 3.8).

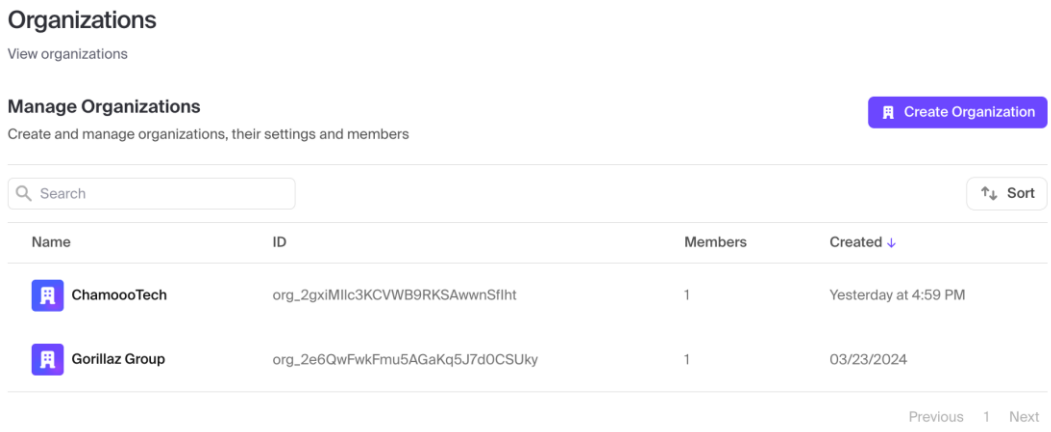

Рисунок 3.6 – Список створених організацій

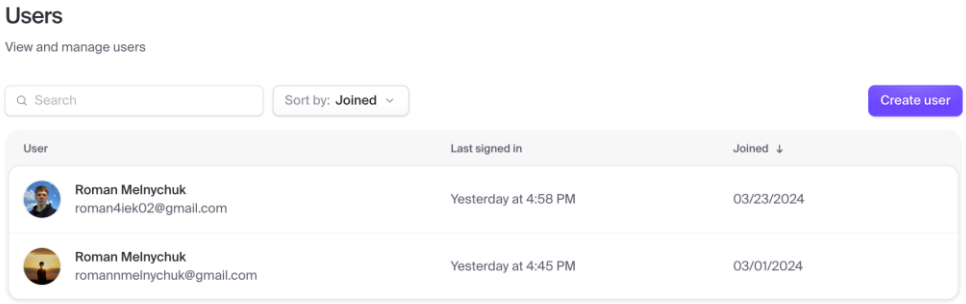

Рисунок 3.7 – Список користувачів

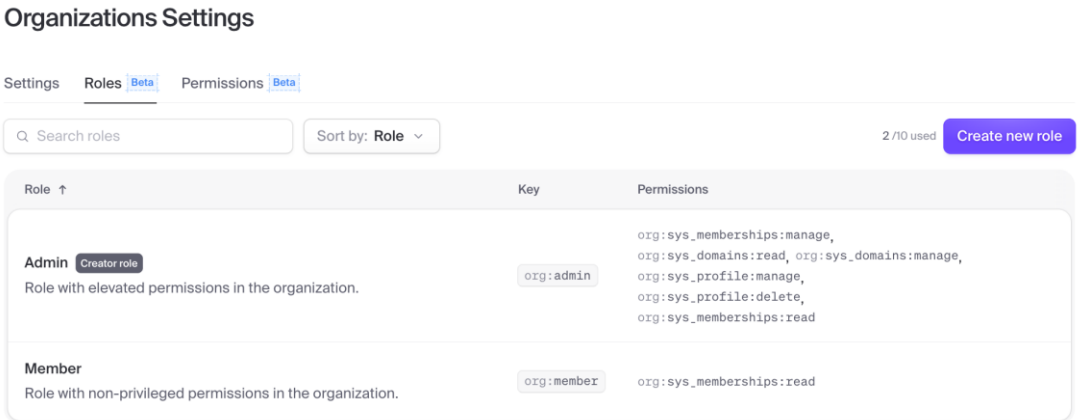

Рисунок 3.8 – Управління ролями користувачів

3.2.3. Дашборд

Розглянемо основний функціонал додатку. Розпочнімо з папки lib, яка містить різноманітні утиліти та допоміжні модулі, які використовуються в різних частинах додатку. Коди файлів надано в додатку В. Перелік файлів та їх призначення:

- **create-audit-log.ts** – містить функцію **createAuditLog**, яка створює запис у журналі аудиту. Функція використовує аутентифікацію користувача через Clerk, отримуючи інформацію про поточного користувача і його організацію. Запис аудиту включає деталі про користувача, тип сутності, її ідентифікатор, назву і дію (створення, оновлення або видалення) (додаток В.1).
- **create-safe-action.ts** – містить функцію **createSafeAction**, яка створює безпечні дії з валідацією вхідних даних за допомогою бібліотеки **zod**. Валідація забезпечує, що дані відповідають визначеній схемі, перш ніж передати їх обробнику. Якщо валідація не проходить, функція повертає помилки полів (додаток В.2).
- **db.ts** – цей файл створює екземпляр клієнта Prisma для взаємодії з базою даних. Екземпляр зберігається в глобальній змінній, щоб уникнути повторного створення клієнта в середовищах розробки, де відбувається часте перезавантаження сервера (додаток В.3).
- **fetcher.ts** – цей файл містить функцію **fetcher**, яка робить запит на вказаний URL і повертає відповідь у форматі JSON. Вона часто використовується для отримання даних з API (додаток В.4).
- **generate-message.ts** – містить функцію **generateMessage**, яка генерує текстове повідомлення на основі запису в журналі аудиту. Повідомлення описує дію, здійснену над певною сутністю (створення, оновлення, видалення), і включає назву сутності (додаток В.5).
- **unsplash.ts** – цей файл містить конфігурацію для API Unsplash. Він створює екземпляр API за допомогою бібліотеки **unsplash-js** і використовує ключ доступу, збережений у змінних середовища. Це

дозволяє взаємодіяти з Unsplash API для отримання зображень (додаток В.6).

- **utils.ts** – цей файл містить утилітарну функцію **cn**, яка об'єднує класи CSS за допомогою **clsx** і **tailwind-merge**. Це корисно для умовного застосування класів Tailwind CSS, дозволяючи легко комбінувати їх та уникати конфліктів (додаток В.7).

3.2.3.1. Серверні події

Для виконання асинхронних дій з обробкою результатів та стану виконання в компоненті React було створено хук **useAction** (додаток В.11). Він забезпечує зручний спосіб керування станом та обробкою результатів асинхронних операцій, таких як запити до сервера або інші обчислювальні задачі.

Хук приймає функцію дії (**action**), яка виконує асинхронну операцію і повертає стан цієї дії (**ActionState**). Він керує станами поля, помилками та даними, використовуючи React state (**useState**). Зокрема, **fieldErrors** зберігає помилки валідації полів, **error** зберігає загальну помилку, якщо вона виникає, **data** зберігає результат успішної операції, а **isLoading** відстежує стан завантаження (виконання операції).

Хук приймає об'єкт опцій (**options**), що дозволяє виконувати додаткові дії у випадку успіху, помилки або завершення операції. Опції включають функції **onSuccess**, яка викликається при успішному завершенні операції, **onError**, яка викликається при виникненні помилки, та **onComplete**, яка викликається після завершення операції незалежно від результату.

Функція **execute** виконує асинхронну операцію і обробляє результат. Вона встановлює **isLoading** в **true** на початку, виконує асинхронну операцію (**action**), оновлює стани **fieldErrors**, **error**, та **data** на основі результату операції, викликає відповідні функції з **options** (при успіху, помилці або завершенні), і встановлює **isLoading** в **false** після завершення операції.

Для управління різними аспектами роботи з дошками, списками та картками в додатку було створено серверні події (рис. 3.9). Ці події охоплюють

створення, оновлення, копіювання та видалення дошок, списків і карток, а також зміну порядку списків і карток.

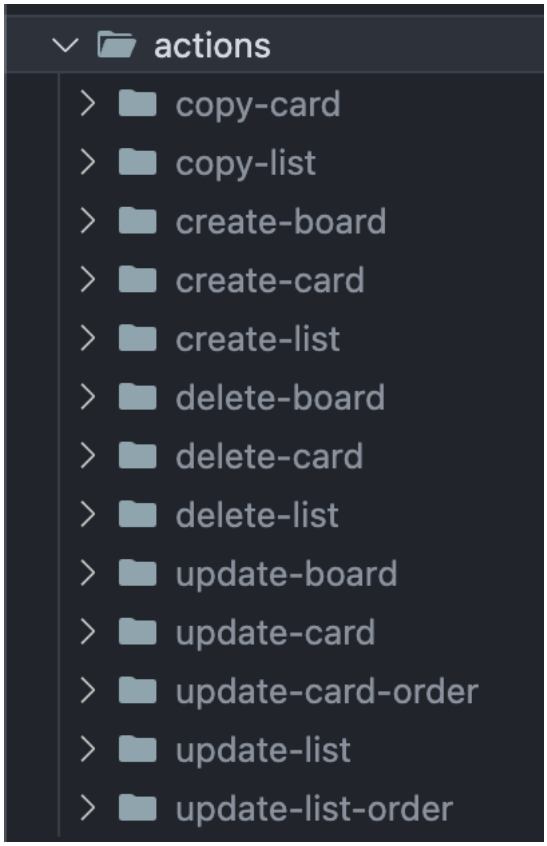

Рисунок 3.9 – Структура директорій серверних подій

Кожна папка має три файли (див. рис. 3.10):

- **index.ts** - головний файл. Він містить основну логіку для обробки дії, наприклад, копіювання картки або списку. У цьому файлі буде реалізовано функцію, яка виконує необхідну операцію, використовуючи дані зі схеми та типів.
- **schema.ts** - файл для визначення схеми валідації даних. Він використовує бібліотеку **zod** для визначення структури та правил валідації даних, що використовуються при копіюванні картки або списку. Наприклад, схема **CopyCard** визначає, що дані повинні містити рядки **id** та **boardId**.
- **types.ts** - Файл для визначення типів даних. Він містить типи, створені на основі схеми валідації з **schema.ts**, а також додаткові типи, що використовуються в обробці дій. Наприклад, тип **InputType** визначається на основі схеми **CopyCard**, а тип **ReturnType** визначає стан дії

(**ActionState**) з вхідними даними **InputType** і вихідними даними типу **Card**.

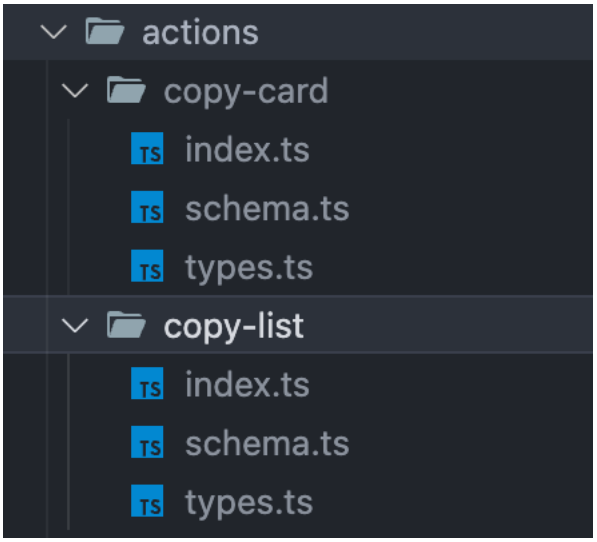

Рисунок 3.10 – Структура файлів серверної події

Для прикладу візьмемо подію для копіювання картки – «CopyCard».

Файл type.ts:

```
import { z } from 'zod';
import { Card } from '@prisma/client';
import { ActionState } from '@/lib/create-safe-action';
import { CopyCard } from './schema';

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

Файл schema.ts, де вказані обов’язкові вхідні поля (id та boardId).

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

export const CopyCard = z.object({
   id: z.string(),
   boardId: z.string(),
});
```

Та головний файл index.ts, де прописана вся логіка копіювання картки:

```
const handler = async (data: InputType): Promise<ReturnType> => {
  const { userId, orgId } = auth();
  if (!userId || !orgId) return { error: 'Unauthorized' };
  const { id, boardId } = data; let card;
  try {
 const cardToCopy = await db.card.findUnique({where:{id,list:{board:{orgId}} }});
    if (!cardToCopy) return { error: 'Card not found.' };
    const lastCard = await db.card.findFirst({
      where: { listId: cardToCopy.listId },
      orderBy: { order: 'desc' },
      select: { order: true },
    }); const newOrder = lastCard ? lastCard.order + 1 : 1;
    card = await db.card.create({
      data: {
        title: `${cardToCopy.title} – Copy`,
        description: cardToCopy.description,
        order: newOrder,
        listId: cardToCopy.listId,
      },
    }) await createAuditLog({
      entityTitle: card.title,
      entityId: card.id,
      entityType: ENTITY_TYPE.CARD,
      action: ACTION.CREATE })
  } catch (error) return { error: 'Failed to copy card.' }
  revalidatePath(`/board/${boardId}`);
  return { data: card };
};
export const copyCard = createSafeAction(CopyCard, handler);
```

У цьому коді реалізується серверна функція для копіювання картки в системі управління завданнями. Спочатку імпортуються необхідні модулі, такі як **auth** з Clerk для аутентифікації, **db** для доступу до бази даних, **createSafeAction** для створення безпечних дій, і **createAuditLog** для створення записів у журналі аудиту. Використовується директива **'use server';**, яка вказує, що цей код виконується на сервері.

Функція **handler** приймає дані типу **InputType** і повертає проміс з результатом типу **ReturnType**. Спочатку виконується аутентифікація користувача за допомогою **auth()**, з об'єкта аутентифікації отримуються **userId** та **orgId**. Якщо користувач не аутентифікований, функція повертає помилку 'Unauthorized'.

Далі із отриманих даних витягуються **id** картки та **boardId** дошки. Виконується пошук картки, яку потрібно скопіювати, з перевіркою, що ця картка належить до організації користувача. Якщо картка не знайдена, функція повертає помилку 'Card not found.'

Після цього визначається новий порядок картки. Знаходиться остання картка у списку і визначається новий порядок для нової картки. Якщо карток у списку немає, новий порядок буде **1.**

Створюється нова картка з тією ж інформацією, що і у вихідної картки, але з новим порядком і доданим суфіксом ' – Copy' до назви. Одночасно створюється запис у журналі аудиту за допомогою **createAuditLog**.

Якщо виникає помилка під час копіювання картки, функція повертає помилку 'Failed to copy card.' Після успішного створення картки викликається **revalidatePath** для оновлення кешу сторінки дошки. В кінці функція повертає успішний результат з даними нової картки.

Функція **copyCard** створюється за допомогою **createSafeAction**, яка застосовує схему валідації **CopyCard** та основний обробник **handler**. Цей код забезпечує безпечне та контрольоване копіювання карток у системі управління завданнями, включаючи валідацію вхідних даних, перевірку аутентифікації, створення журналу аудиту та обробку помилок.

Всі інші події створені за схожим принципом.

4.2.3.2. Багаторазові компоненти

Багаторазові компоненти (reusable components) потрібні для підвищення ефективності розробки, зменшення дублювання коду, забезпечення консистентності інтерфейсу користувача та спрощення підтримки і розширення додатку. Вони дозволяють використовувати одну і ту ж логіку та візуальні елементи в різних частинах додатку, що значно спрощує процес розробки і покращує загальну якість коду.

Опираючись на загальноприйнятий стиль розробки з Next.js, багаторазові компоненти винесені в окрему папку **components** в кореневій директорії проєкту, аби до них був зручний шлях (рис. 3.11).

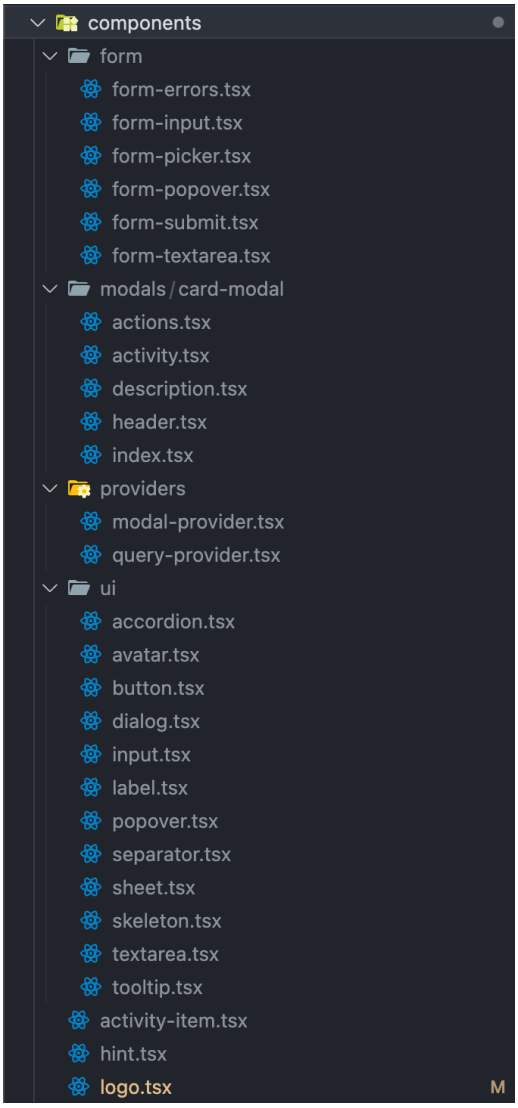

Рисунок 3.11 – Всі багаторазові компоненти

Розглянемо найпоширеніший компонент <Button />, який знаходиться за шляхом **ui/button** (додаток В.12). Він створює кнопку з різними варіаціями.

Компонент **Button** — це багаторазовий компонент кнопки, створений для забезпечення консистентного вигляду кнопок у додатку та спрощення процесу їх стилізації. Він використовується для створення кнопок із заданими стилями та розмірами, що відповідають загальному дизайну додатку. Компонент **Button** створюється за допомогою бібліотек **React**, **@radix-ui/react-slot**, **class-variance-authority** та утиліти **cn** з **@/lib/utils**.

Компонент використовує **cva** для визначення стилів кнопки та їх варіантів. Інтерфейс **ButtonProps** розширює стандартні атрибути HTML-кнопки і додає можливість використання варіантів стилів та розмірів. Компонент **Button** створюється за допомогою **React.forwardRef**, що дозволяє передавати реф (посилання) на елемент кнопки. Він також підтримує можливість використання **Slot** для обгортання дочірніх елементів.

Компонент **Button** використовується в коді React як стандартна кнопка, але з додатковими властивостями для варіантів стилів та розмірів. Наприклад:

```
import { Button } from './path/to/button';
<Button variant="primary" size="lg">Primary Button</Button>
<Button variant="destructive" size="sm">Destructive Button</Button>
<Button variant="outline" size="default">Outline Button</Button>
```

3.2.3.3. API

Для отримання повної інформації про картку зі списку було створено API за допомогою стандартних методів Next.js.

```
import { auth } from '@clerk/nextjs';
import { NextResponse } from 'next/server';
import { db } from '@/lib/db';
export async function GET(req: Request, { params }: { params: { cardId: string } }) {
  try {
    const { userId, orgId } = auth();
```

```
    if (!userId || !orgId) return new NextResponse('Unauthorized', { status: 401 })
    const card = await db.card.findUnique({
      where: {
        id: params.cardId,
        list: {board: {orgId}}},
      include: { list: { select: { title: true } } },
    });
    return NextResponse.json(card);
  } catch (error) {
    return new NextResponse('Internal Server Error', { status: 500 });
  }
}
```

Спочатку імпортуються необхідні модулі: **auth** з Clerk для аутентифікації, **NextResponse** з Next.js для формування відповідей на запити, і **db** з **@/lib/db** для взаємодії з базою даних. Функція **GET** приймає запит **req** і параметри, що включають **cardId** (ідентифікатор картки).

В середині функції виконується аутентифікація користувача за допомогою **auth()**, і отримуються **userId** та **orgId**. Якщо користувач не аутентифікований або не належить до організації, повертається відповідь зі статусом 401 (Unauthorized).

Далі виконується запит до бази даних для отримання картки за ідентифікатором **params.cardId**, з перевіркою, що картка належить до списку, який знаходиться на дошці цієї організації. У запиті до бази даних також включається інформація про список, до якого належить картка, зокрема його назва.

Якщо картка знайдена, вона повертається у форматі JSON за допомогою **NextResponse.json(card)**. Якщо виникає помилка під час виконання запиту або обробки даних, повертається відповідь зі статусом 500 (Internal Server Error).

Також за схожим принципом було створено API для отримання даних про активність учасника організації:

```
import { auth } from '@clerk/nextjs';
import { db } from '@/lib/db';
import { NextResponse } from 'next/server';
import { ENTITY_TYPE } from '@prisma/client';

export async function GET(request: Request, { params }: { params: { cardId: string }
}) {
   try {
      const { userId, orgId } = auth();

      if (!userId || !orgId) {
         return new NextResponse('Unauthorized', { status: 401 });
      }
      const auditLogs = await db.auditLog.findMany({
         where: {
            orgId,
            entityId: params.cardId,
            entityType: ENTITY_TYPE.CARD,
         },
         orderBy: {
            createdAt: 'desc',
         },
         take: 4,
      });

      return NextResponse.json(auditLogs);
   } catch (error) {
      return new NextResponse('Internal Error', { status: 500 });
   }
}
```

3.2.3.4. Дошка проєктів

Для дошок було створено динамічний роут, який забезпечує доступ до різних дошок на основі їхніх ідентифікаторів. Цей роут реалізований через файли **page.tsx** (додаток В.8) та **layout.tsx** (додаток В.9), що дозволяє відображати вміст конкретної дошки та її списків.

У файлі **page.tsx** описано компонент **BoardIdPage**, який приймає параметри з роуту, зокрема **boardId**. Виконується аутентифікація користувача за допомогою **auth()**, і якщо користувач не належить до жодної організації, його перенаправляють на сторінку вибору організації. Потім виконується запит до бази даних для отримання списків та карток, пов'язаних із дошкою. Дані списків передаються в компонент **ListContainer**, який відповідає за їх відображення.

Файл **layout.tsx** містить компонент **BoardIdLayout**, який також виконує аутентифікацію користувача та перевіряє наявність організації. Якщо дошка не знайдена, повертається сторінка **notFound**. В іншому випадку, компонент відображає навігаційну панель **BoardNavbar** з даними дошки та встановлює фонове зображення для дошки. Основний вміст дошки передається через пропс **children**.

3.2.3.5. Списки та drag and drop

Розглянемо файл **list-container.tsx**, в якому відбувається логіка списків. Компонент **ListContainer** відповідає за відображення та управління списками і картками у системі управління завданнями з підтримкою функціоналу drag-and-drop. Компонент використовує бібліотеки **@hello-pangea/dnd** для реалізації drag-and-drop, **Clerk** для аутентифікації, і **Prisma** для взаємодії з базою даних.

3.2.3.5.1. Основні функції та логіка компонента

Компонент **ListContainer** приймає дані списків і ідентифікатор дошки через пропси. Він використовує хук **useState** для зберігання стану впорядкованих списків і хук **useEffect** для оновлення стану при зміні вхідних даних.

Для обробки змін порядку списків та карток використовуються два асинхронні дії (**useAction**), які визначені у відповідних файлах (**updateListOrder** та **updateCardOrder**). Ці дії виконуються за допомогою функцій

executeUpdateListOrder і **executeUpdateCardOrder**, і в разі успіху або помилки відображають відповідні повідомлення за допомогою бібліотеки **sonner**.

3.2.3.5.2. Реалізація drag-and-drop

Функція **onDragEnd** викликається після завершення переміщення елементу. Вона отримує об'єкт результату, що містить інформацію про джерело і місце призначення переміщеного елементу. Якщо елемент не був переміщений або залишився в тому ж місці, обробка завершується.

Для переміщення списків перевіряється тип переміщуваного елементу. Якщо це список (**type === 'list'**), використовується функція **reorder** для зміни порядку списків, після чого оновлені дані зберігаються в стані і виконується асинхронна дія для оновлення порядку списків у базі даних.

Для переміщення карток спочатку визначаються списки джерела і призначення. Якщо картка переміщується в межах одного списку, оновлюється порядок карток у цьому списку. Якщо картка переміщується в інший список, вона видаляється зі списку джерела і додається до списку призначення, після чого оновлюється порядок карток в обох списках. Після цього виконується асинхронна дія для оновлення порядку карток у базі даних.

Компонент **ListContainer** (додаток В.13) використовує компоненти **DragDropContext** і **Droppable** з бібліотеки **@hello-pangea/dnd** для забезпечення контексту drag-and-drop і визначення області, в якій елементи можуть бути переміщені. Усі списки відображаються за допомогою компонента **ListItem**, а форма для додавання нового списку відображається за допомогою компонента **ListForm.**

Таким чином, цей код забезпечує інтерактивне і зручне управління списками та картками за допомогою drag-and-drop, а також підтримує збереження змін у базі даних з асинхронною обробкою дій.

3.3. Інструкція користувача

При відкритті сайту користувача зустрічає презентаційна сторінка, на якій розміщені кнопки для входу та реєстрації.

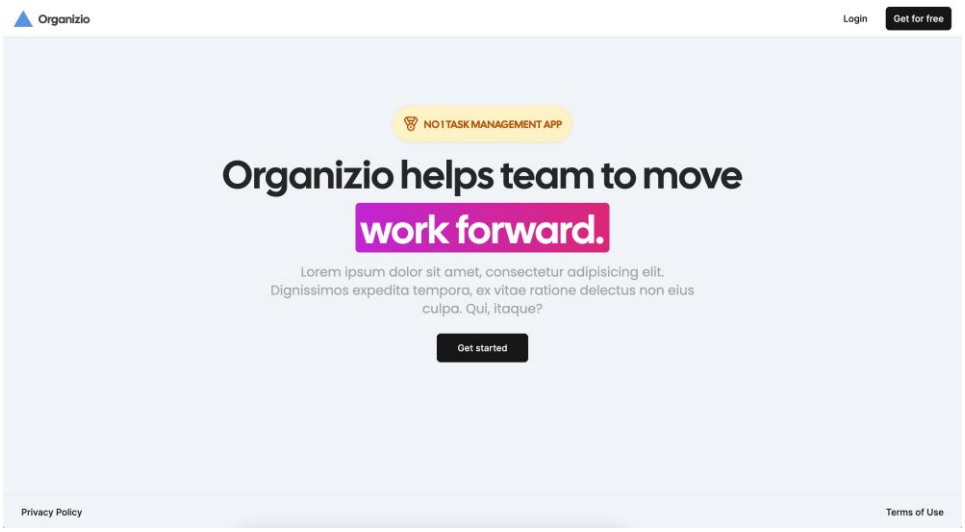

Рисунок 3.3.1 – Презентаційна сторінка

Для входу потрібно натиснути кнопку **Login**, для реєстрації – **Get started** або **Get for free**. Спробуємо зареєструватись. Натискаємо кнопку **Get for free** та потрапляємо на сторінку з формою реєстрації.

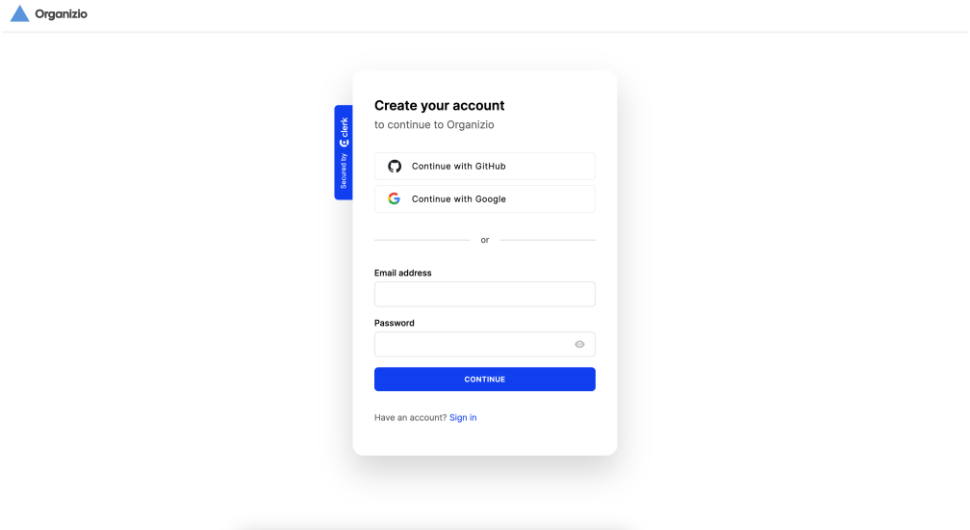

Рисунок 3.3.2 – Сторінка реєстрації

Існує три способи авторизації:

1. через GitHub;
2. через Google;
3. за допомогою email адреси та пароля.

Зареєструємося через Google. Якщо користувач не входить до жодної організації, перед ним появляється вікно, аби створити її. Вказуємо ім’я та, за бажанням, зображення та тиснемо **Create organization**.

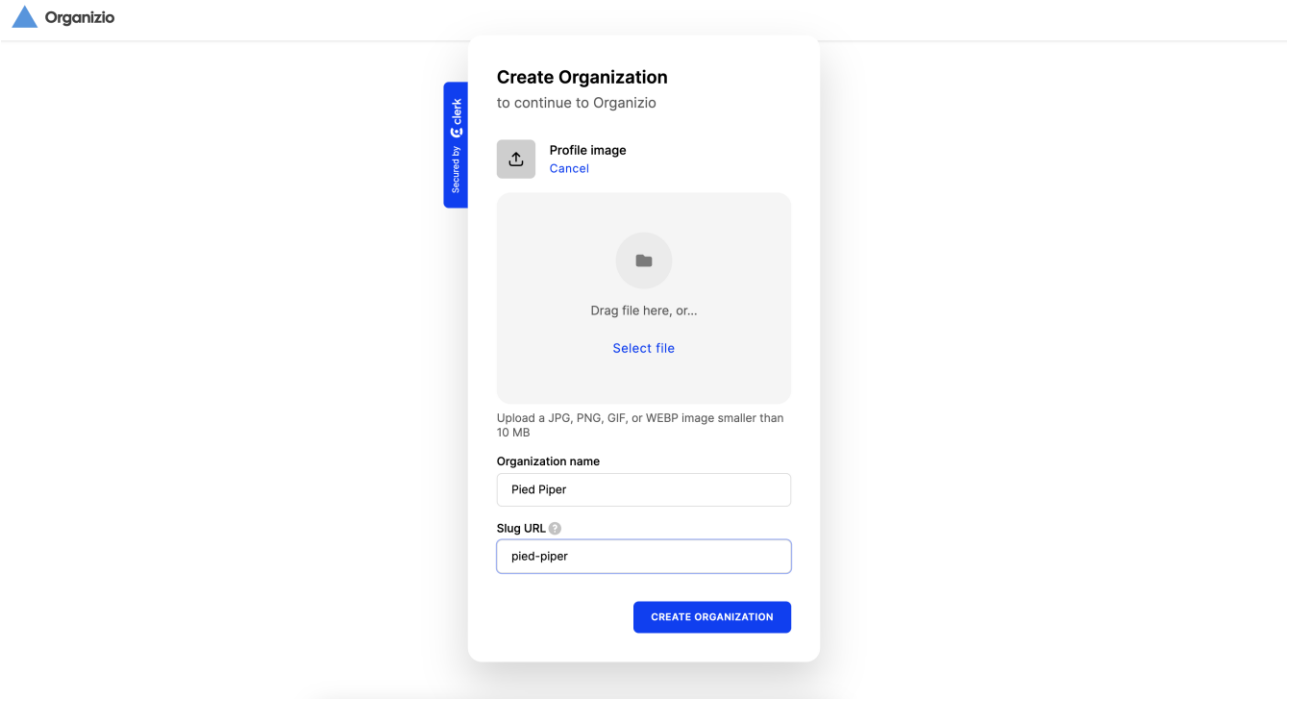

Рисунок 3.3.3 – Сторінка створення організації

Після цього появляється вікно, де ми можемо запросити інших користувачів до організації, вказавши їхні email адреси, та масово встановити для них роль.

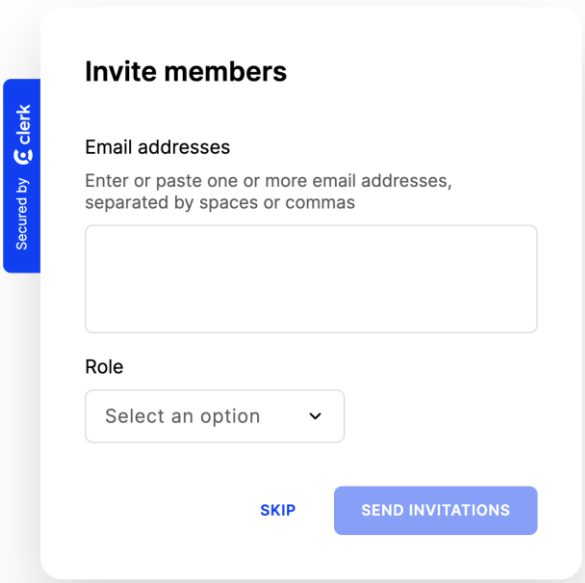

Рисунок 3.3.4 – Форма для запрошення

Після успішної реєстрації та створення організації ми потрапляємо на сторінку дашборду.

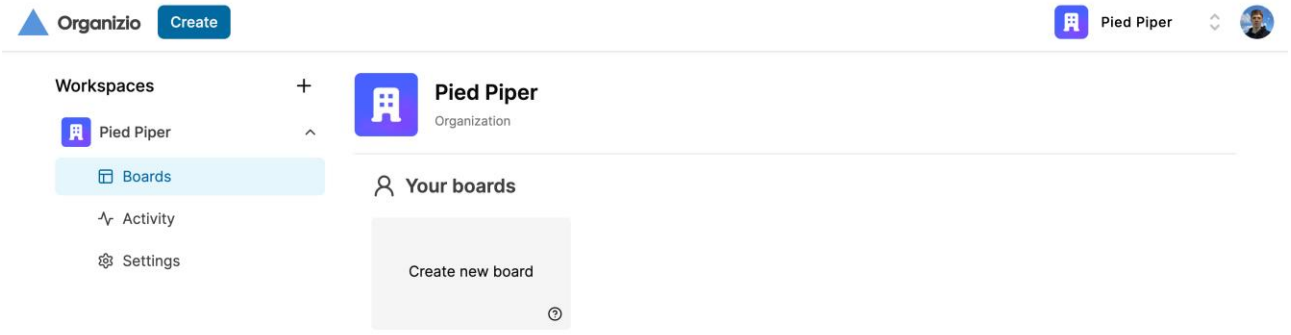

Рисунок 3.3.5 – Головна сторінка дашборду

Зліва відображаються всі робочі простори (технічно – організації), в яких перебуває користувач. Є функція додавання нових просторів, перегляду активності даного простору, налаштування.

Розглянемо налаштування (див. рис. 3.3.6). Тут ми можемо переглянути всіх учасників організації, запросити інших учасників (див. рис. 3.3.7)

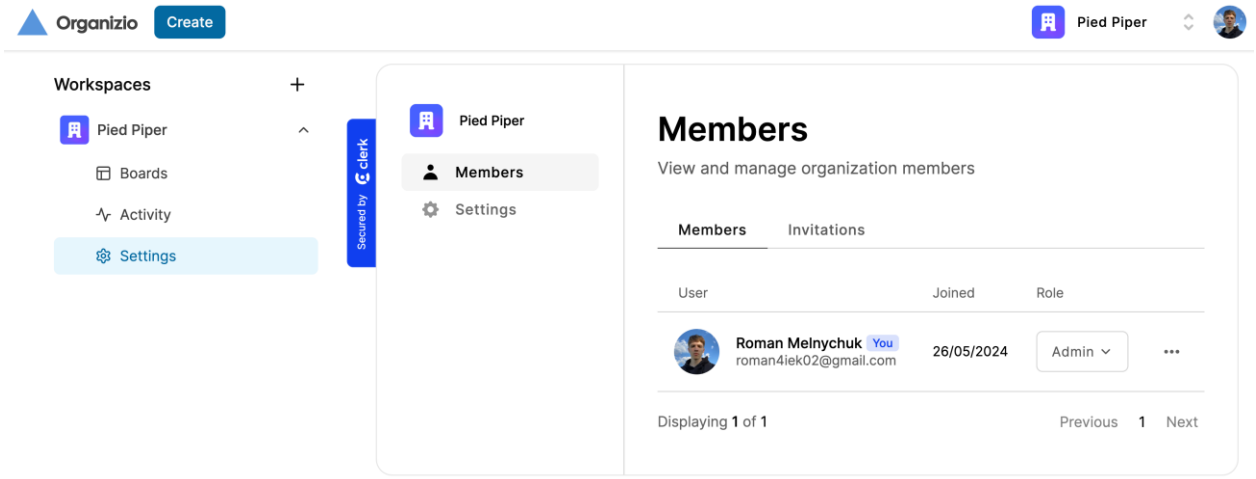

Рисунок 3.3.6 – Сторінка налаштувань організації. Список учасників

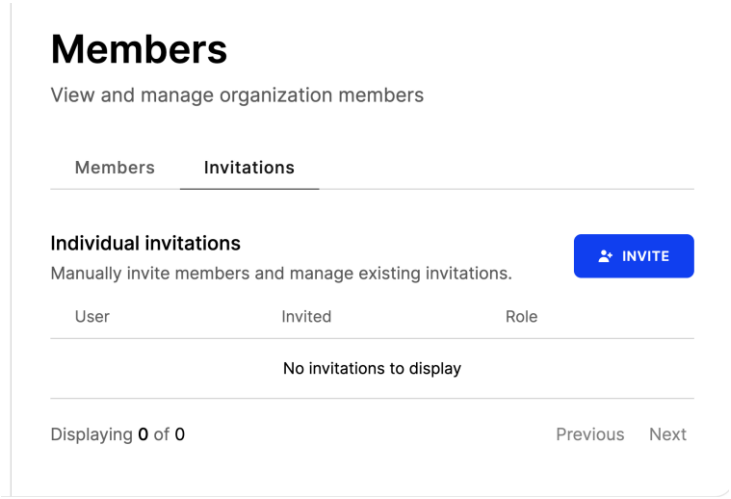

Рисунок 3.3.7 – Сторінка налаштувань організації.

Запрошення в організацію

Також можна покинути або видалити організацію (якщо користувач – це власник).

Повернемося на головну сторінку. Створимо нову дошку проєкту. Для цього натиснемо на кнопку **Create new board**. Після цього появиться модальне вікно, де ми можемо вказати назву для дошки та обрати фонове зображення. Якщо не обрати фонове зображення або не вказати назву, під відповідним полем появиться помилка (рис. 3.3.8). Вкажемо ім’я та зображення та натискаємо **Create** (рис. 3.3.9.).

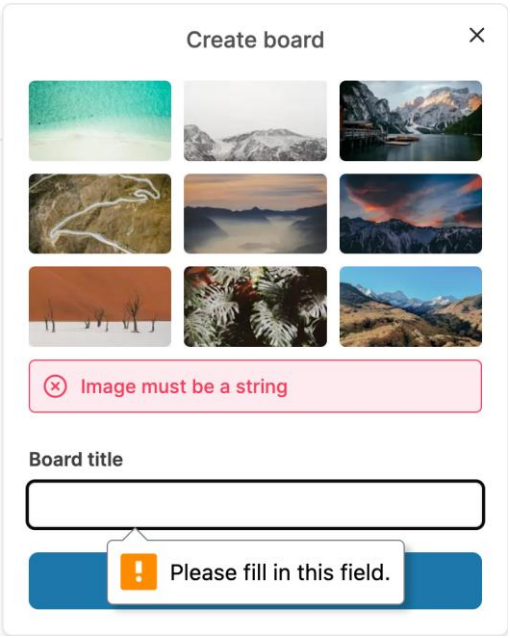

Рисунок 3.3.8 – Створення дошки. Незаповнені поля

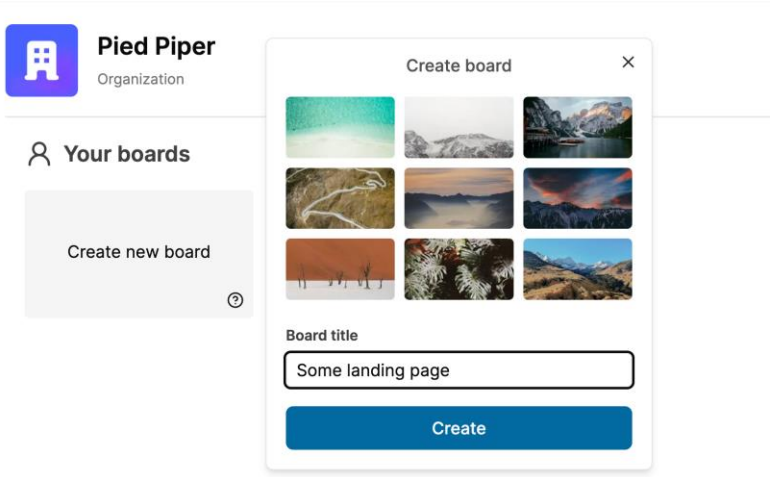

Рисунок 3.3.9 – Створення дошки. Заповнені поля

Після створення дошки ми потрапляємо на сторінку дошки (рис. 3.3.10), а в списку всіх дошок на головній сторінці дашборду з’являється нова дошка (рис. 3.3.11).

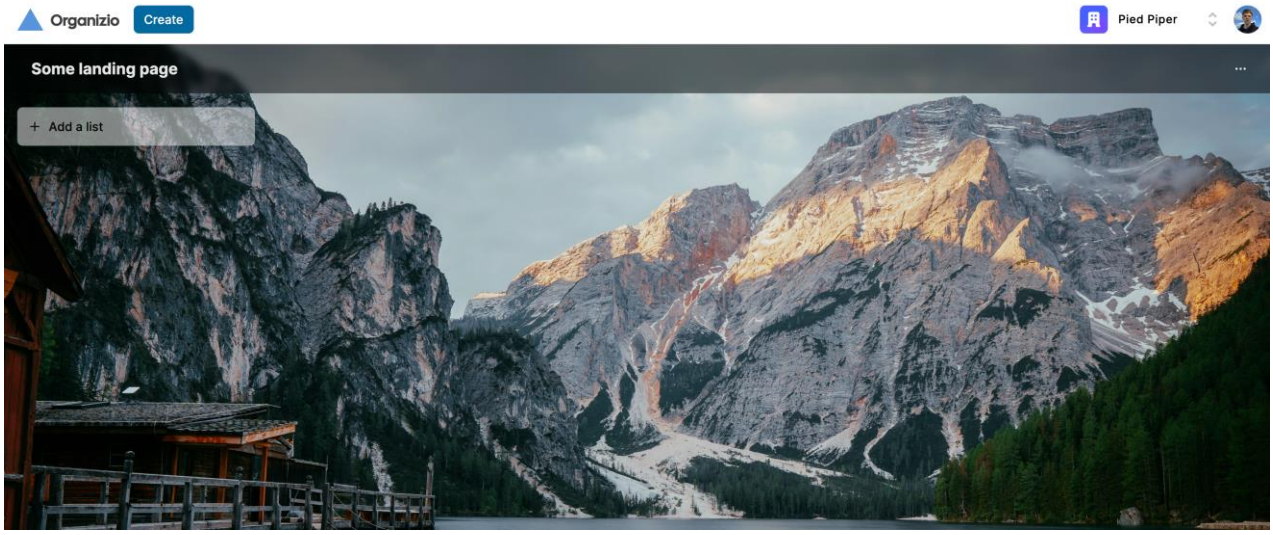

Рисунок 3.3.10 – Сторінка створеної дошки

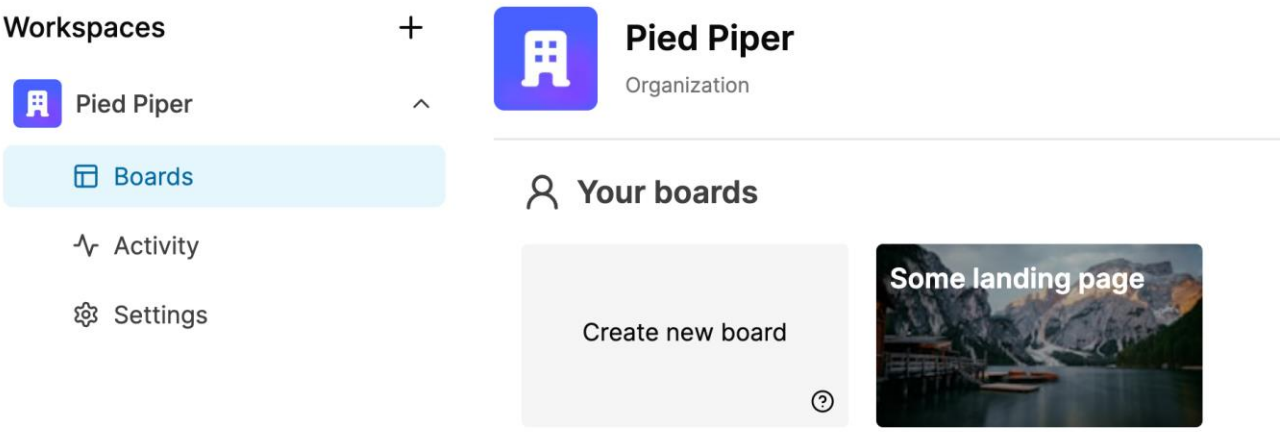

Рисунок 3.3.11 – Створена дошка в списку дошок

В правому верхньому кутку можна побачити три крапочки (рис. 3.3.12), при натисненні на які появиться випадаюче вікно, де є кнопка для видалення дошки, при натисненні на яку дошка видалиться та користувач перенаправиться на головну сторінку дашборду.

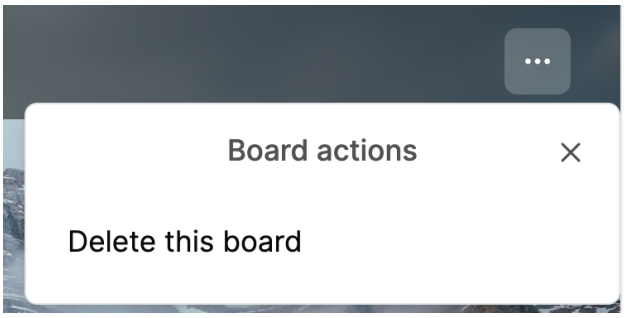

Рисунок 3.3.12 – Опція видалення дошки

Тепер створимо новий список та задамо йому картки (задачі). Для цього натискаємо на кнопку **Add a list**, після чого в тому ж місці з’явиться поле для введення назви списку. Якщо натиснути кнопку **Add list** без заповнення назви, під полем появиться інформаційна помилка (див. рис. 3.3.13). Всі інші обов’язкові поля опрацьовуються таким самим чином.

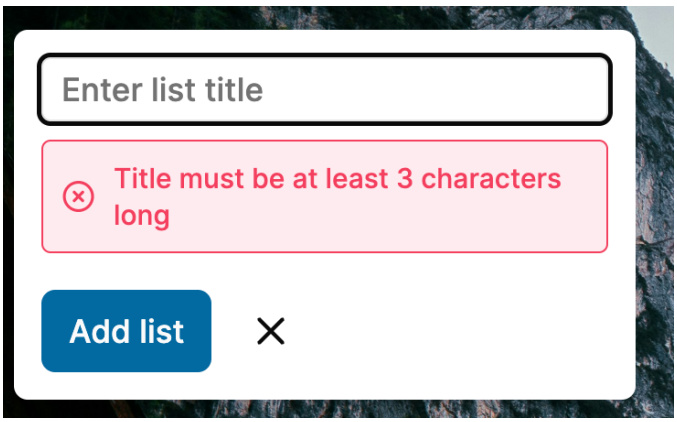

Рисунок 3.3.13 – Незаповнена назва списку

Створимо структуру канбану – тобто три списки: «Backlog», «Потрібно зробити», «В процесі», «Зроблено» (див. рис. 3.3.14).

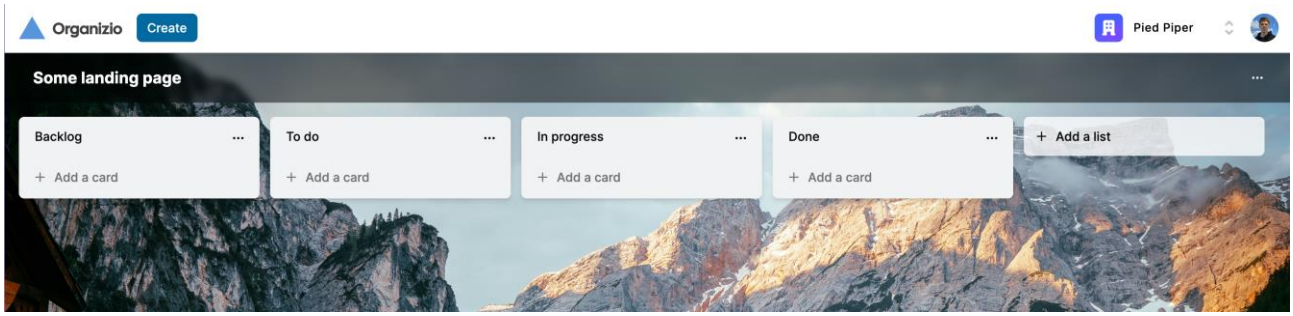

Рисунок 3.3.14 – Створені списки

Можна змінювати порядок списків за допомогою звичайного перетягування ЛКМ.

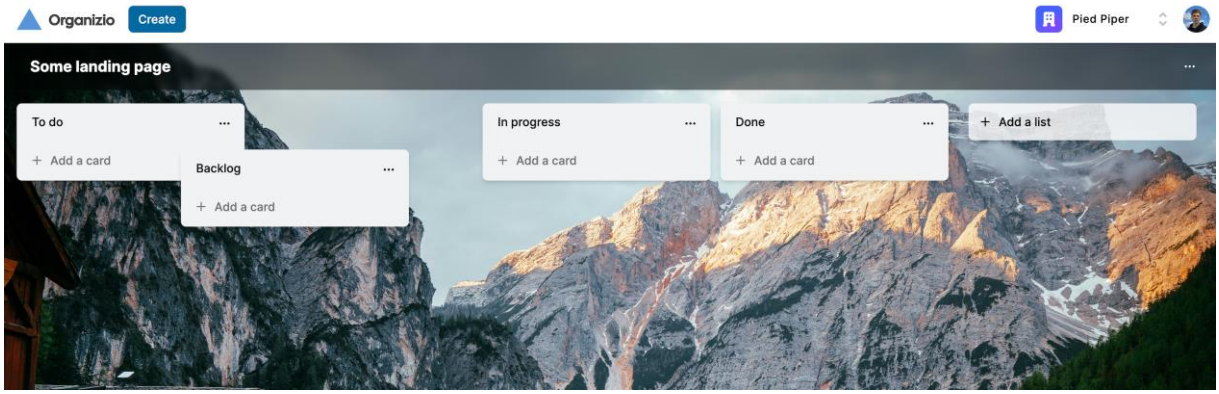

Рисунок 3.3.15 – Перетягування списку

Якщо натиснути на три крапочки зверху справа в списку, появиться вікно, де доступні такі функції: додати картку, скопіювати список, видалити список (див. рис. 3.3.16). Якщо натиснути кнопку «Скопіювати список» список скопіюється зі всіма картками та буде повністю незалежним від оригінального. Для прикладу створимо кілька карток в беклозі та натиснемо кнопку **Copy list**.

Створився новий список з тими самими картками та назвою з приставкою «– Copy» (див. рис. 3.3.17).

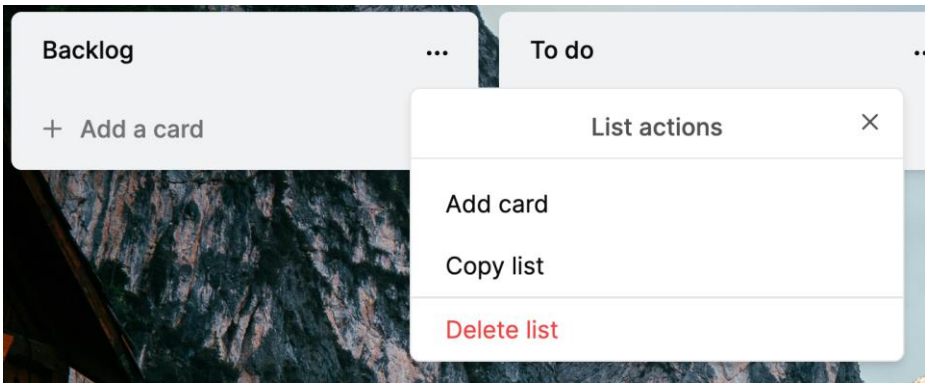

Рисунок 3.3.16 – Опції списку

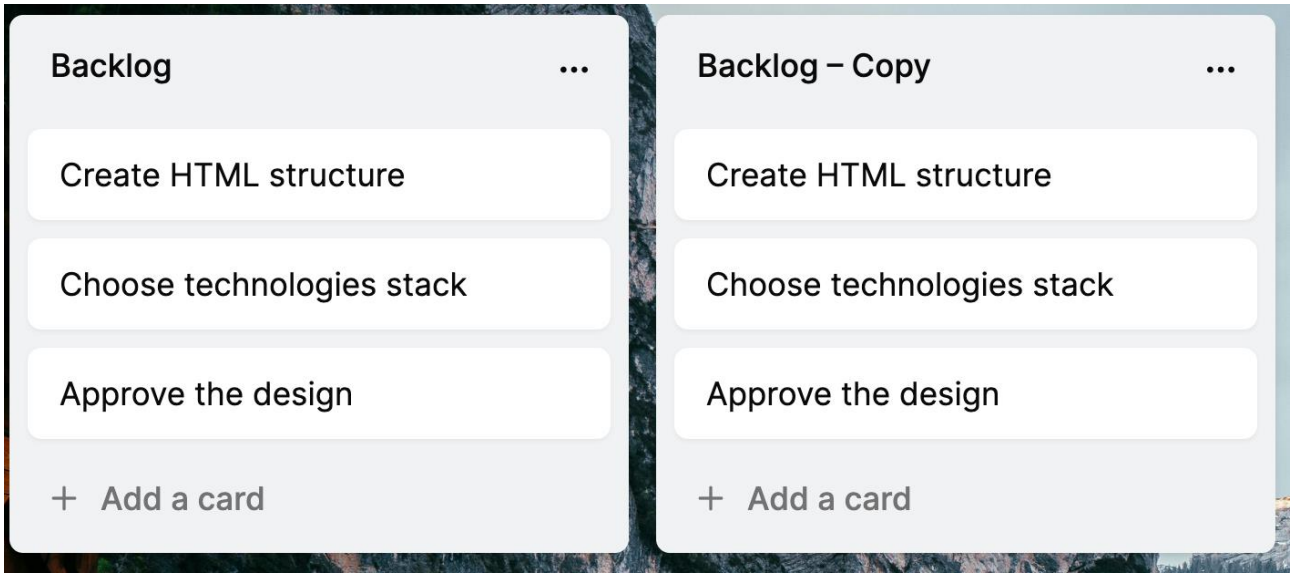

Рисунок 3.3.17 – Оригінальний та скопійований список

Якщо натиснути на окрему картку, появиться модальне вікно, де ми можемо змінити назву картки, задати опис, переглянути активність картки, скопіювати або видалити її. Копіювання відбувається за таким самим принципом, як і копіювання списку.

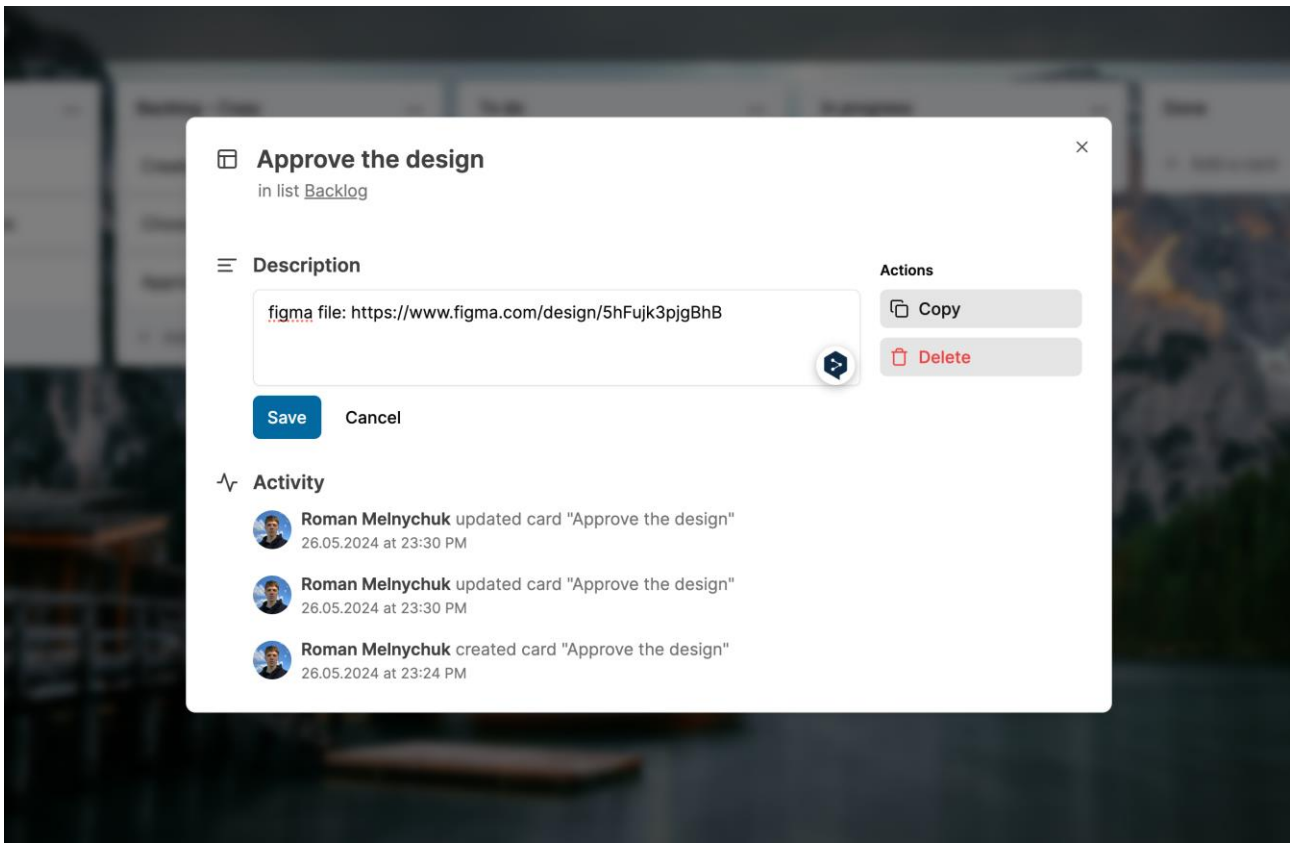

Рисунок 3.3.18 – Модальне вікно картки

Картки можна змінювати місцями, переносити з одного списку в інший за допомогою прийому «drag and drop», як і зі списками.

Кожна подія супроводжується спливаючим повідомленням. Також якщо виникне будь-яка помилка в системі, користувач дізнається про неї таким самим чином.

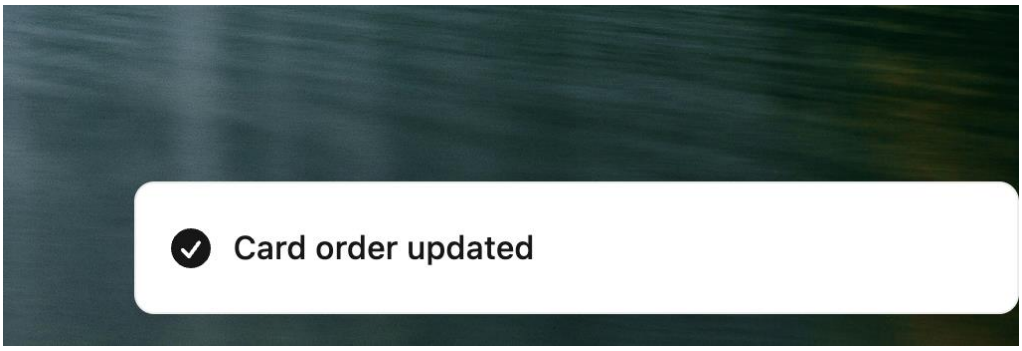

Рисунок 3.3.19 – Спливаюче повідомлення

Якщо після деяких маніпуляцій в дошці перейти на головну сторінку, а потім натиснути на пункт меню **Activity**, то ми зможемо побачити всю активність користувача.

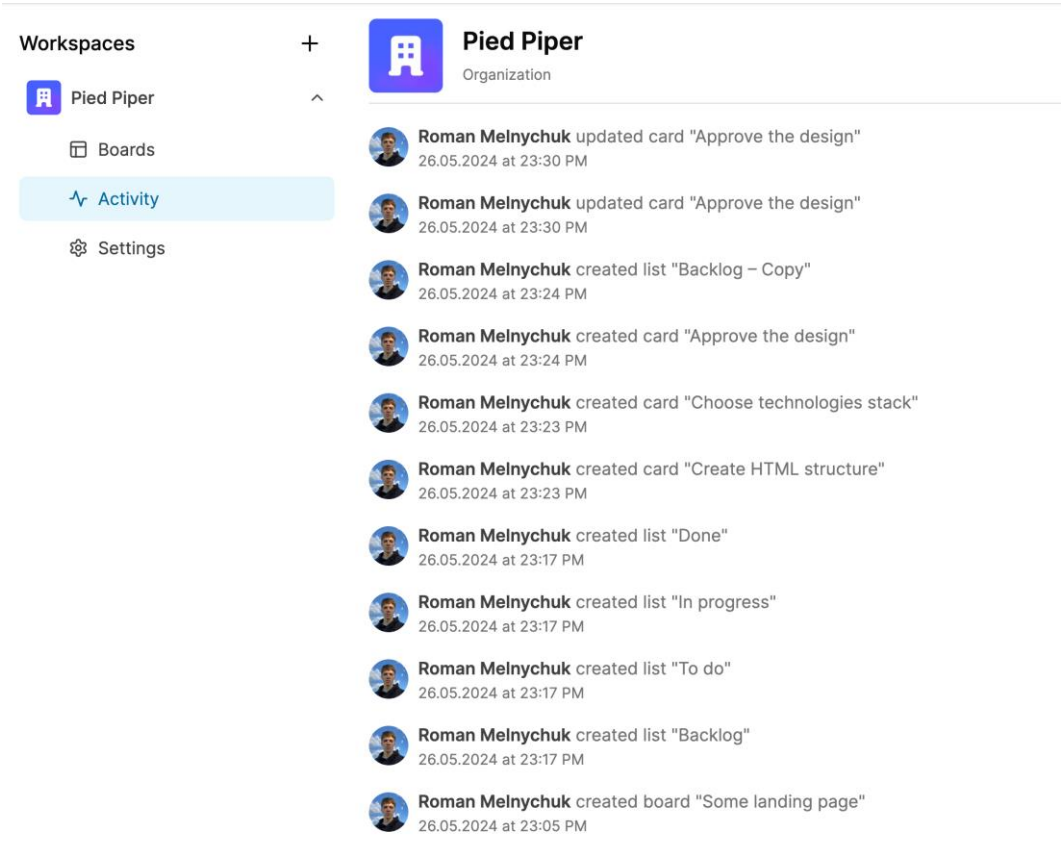

Рисунок 3.3.20 – Активність користувача

Інші учасники організації також можуть переглядати, створювати та змінювати вже наявні дошки, створені іншими учасниками.

Для того, щоб вийти з акаунту, потрібно натиснути на аватар свого акаунта в верхній навігаційній панелі, з’явиться меню, в ньому обрати **Sign out**. Після цього користувач переадресується на презентаційну сторінку.

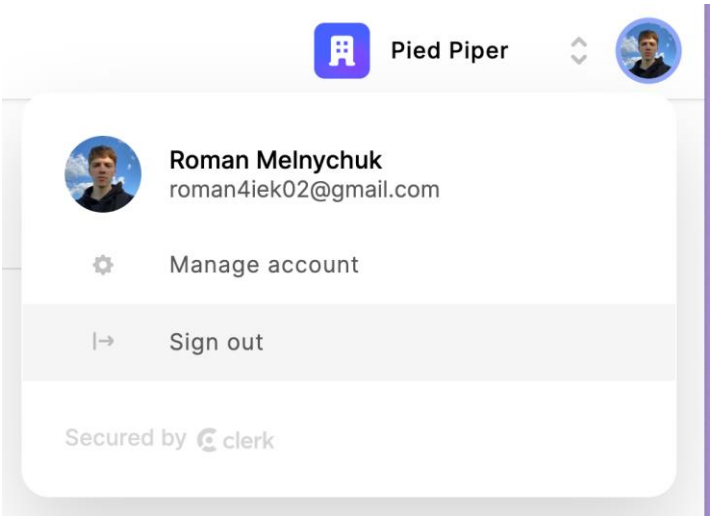

Рисунок 3.3.21 – Вихід з акаунта

РОЗДІЛ 4. ОХОРОНА ПРАЦІ

У ході виконання роботи люди взаємодіють із предметами, знаряддями праці та іншими людьми. На них впливають різні фактори виробничого середовища, зокрема температура, вологість, повітряний потік, шум, вібрації, небезпечні речовини та випромінювання.

Стан здоров'я людини, її працездатність, ставлення до роботи і результати праці значною мірою залежать від умов праці. Робота в умовах високого психічного та фізичного напруження, тривалих статичних навантажень і обмеженої фізичної активності може призвести до неврологічних розладів, психічних порушень, захворювань опорно-рухового апарату та серцевосудинної системи. Зі зростанням складності людино-технологічних систем, економічні, соціальні та інші втрати через невідповідність умов праці можливостям людини стають більшими. Вплив небезпечних та шкідливих виробничих факторів може призвести до нещасних випадків та професійних захворювань.

На сьогодні вплив комп'ютерів, комунікаційного обладнання та інших засобів виробництва на людський організм вивчено недостатньо. У цьому контексті питання охорони праці та техніки безпеки набувають особливої важливості.

Охорона праці – це система заходів, спрямованих на забезпечення безпеки на робочому місці та збереження життя і здоров'я працівників у процесі їхньої трудової діяльності.

Охорона праці є однією з основних систем на підприємствах різних форм власності. Дотримання безпечних методів роботи та правил охорони праці може значно зменшити кількість нещасних випадків на виробництві – до 95%. Залишковий відсоток пов'язаний з людським фактором, якого важко уникнути, тобто травмами, спричиненими недотриманням вимог охорони праці як роботодавцями, так і працівниками.

Охорона праці та здоров'я охоплює практично всі аспекти життя людей, переплітаючись із багатьма технічними та гуманітарними науками і виконуючи найважливішу функцію щодо захисту життя і здоров'я людини у всіх сферах.

Охорона праці включає різноманітні заходи з нормативно-правовою підтримкою, такі як організаційні, технічні, соціально-економічні, гігієнічні, медичні та профілактичні заходи.

Ефективна організація праці вирішує три групи завдань:

- економічні завдання – підвищення продуктивності праці, покращення якості продукції, зниження витрат і економія ресурсів.
- психофізіологічні завдання – створення сприятливих умов праці, зниження частки непродуктивної праці, підтримка здоров'я і працездатності співробітників.
- соціальні завдання – підвищення змістовності та привабливості роботи, стимулювання творчої ініціативи.

Організація праці тісно пов'язана із заходами щодо вдосконалення виробничої системи та системи управління виробництвом.

Основним орієнтиром у сфері охорони праці є Закон України "Про охорону праці". Крім того, діють Конституція України, Кодекс законів про працю, Закон України "Про обов'язкове державне страхування від нещасних випадків на виробництві та професійних захворювань" та інші нормативні акти і міжнародні конвенції.

Інженерна психологія вивчає взаємодію між людиною і технікою, зокрема обчислювальною технікою, для визначення функціональної компетентності людини у виробничій діяльності. Вона фокусується на діяльності операторів, і завдяки її рекомендаціям можна створити такі умови праці, які допомагають зберегти високі психофізіологічні можливості.

Ергономіка займається вивченням системи "людина-машина-середовище" і має велике значення для охорони праці та здоров'я. Вона розробляє рекомендації щодо проєктування і експлуатації технічного устаткування, встановлює вимоги до виробничого середовища для забезпечення комфортних умов праці, підтримки фізичної форми, працездатності і здоров'я.

Технічна естетика вивчає соціокультурні, технічні та естетичні проблеми, що формують гармонійне середовище, яке створюють засоби промислового

виробництва. Вона забезпечує оптимальні умови праці, побуту та відпочинку людей. Це наукова основа дизайну, що аналізує естетичні, економічні, технологічні, гігієнічні та ергономічні фактори, сприяючи оптимізації умов праці людини.

ВИСНОВОК

У даній дипломній роботі було проведено системний аналіз системи управління завданнями для веб-студії. Було виявлено недоліки існуючих процесів та досліджено сферу управління проектами. Розроблено функціональну модель веб-студії та описано загальні характеристики управління проектами. Створено схему організаційної структури студії та описано взаємодію між працівниками. Проведено аналіз поточного стану комп'ютеризації. Розроблено функціональну модель та виконано аналіз існуючих бізнес-процесів. Здійснено огляд існуючих аналогів системи та їх порівняння.

Забезпечено оперативне отримання повної і достовірної інформації щодо управління завданнями. Реалізовано функції додавання, редагування, видалення, перегляду завдань, перегляду активностей учасників.

Розробка веб-системи для управління завданнями може надати численні переваги, включаючи підвищення ефективності, покращення взаємодії з клієнтами, підвищення доступності, зниження витрат та покращення безпеки. Ці переваги роблять розробку веб-системи значущою та вартісною інвестицією для веб-студії.

На різних етапах розробки та проектування було використано такі програмні інструменти: Prisma для управління базою даних, Next.js для розробки інтерфейсу та серверної частини, Clerk для аутентифікації користувачів. Проект реалізовано у середовищі Visual Studio Code з використанням СУБД MySQL.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. HTML Підручник - W3Schools українською [Електронний ресурс] // W3Schools. URL: https://w3schoolsua.github.io/html/index.html#gsc.tab=0 (дата звернення: 19.05.2024).
2. CSS Підручник - W3Schools українською [Електронний ресурс] // W3Schools. URL: https://w3schoolsua.github.io/css/index.html#gsc.tab=0 (дата звернення: 11.05.2024).
3. JavaScript Підручник. Основи вебпрограмування - W3Schools українською [Електронний ресурс] // W3Schools. URL: https://w3schoolsua.github.io/js/index.html#gsc.tab=0 (дата звернення: 20.05.2024).
4. Optimization Models and Educational Teaching Research in Agricultural Logistics System [Електронний ресурс] // IEEE Xplore. URL: https://ieeexplore.ieee.org/document/10201100 (дата звернення: 20.05.2024).
5. Build something great! – Code with Antonio [Електронний ресурс] // Code with Antonio. URL: https://www.codewithantonio.com (дата звернення: 20.05.2024).
6. Управління ІТ проектами [Електронний ресурс] : методичні рекомендації до самостійної роботи для здобувачів освітнього ступеня «Бакалавр» спеціальності 122 «Комп’ютерні науки» освітньо-професійних програм «Комп’ютерні науки» та «Інформаційні системи та штучний інтелект» денної та заочної форм навчання ⁄ укладачі : С. В. Грибков, О. Л. Сєдих ; Національний університет харчових технологій. – Київ : НУХТ, 2022 – 25 с.– № 51.64.
7. Управління ІТ проектами [Електронний ресурс]: методичні рекомендації до виконання курсової роботи для здобувачів освітнього ступеня «Бакалавр» спеціальності 122 «Комп’ютерні науки» освітньо – професійних програм «Комп’ютерні науки» та «Інформаційні системи та штучний інтелект» денної та заочної форм навчання. / Уклад.: С. В. Грибков, О. Л. Сєдих – К.: НУХТ, 2023. – 76 с.

8. Проектування інформаційних систем [Електронний ресурс]: конспект лекцій для студентів освітнього ступеня «Бакалавр» спеціальності 122 «Комп’ютерні науки» денної та заочної форм навчання. Уклад.: О. М. М’якшило, О. В. Харкянен: НУХТ, 2018. – 47 с..
9. Управління ІТ проектами [Електронний ресурс]: лабораторний практикум для студентів напряму підготовки 6.050101 "Комп'ютерні науки" денної та заочної форм навч. / уклад. О. А. Хлобистова, М. В. Гладка. - К. : НУХТ, 2013. – 108 с.. URL: http://library.nuft.edu.ua/ebook/file/51.07A.pdf.
10. Проектування та розробка програмного забезпечення [Електронний ресурс]: метод. рекомендації до викон. курсового проекту для здобувачів освіт. ступ. "Бакалавр" спец. 122 "Комп'ютерні науки" освіт.-проф. програм "Комп'ютерні науки", "Інформаційні системи та штучний інтелект" ден. та заоч. форм навч. / уклад. : О. М. М'якшило, О. В. Харкянен ; Нац. ун-т харч. технол. — Київ : НУХТ, 2023. — 27 с. — каф. інформаційних технологій, штучного інтелекту і кібербезпеки.
11. М’якшило О.М. CASE-технології у проектуванні інформаційних систем: електронний навчальний посібник для студентів вищих навчальних закладів / О.М. М’якшило, Л.Г. Загоровська,– К.: НУХТ, 2017. – 190 с.
12. Проектування інформаційних систем. [Електронний ресурс]: лабораторний практикум для студ. освітнього ступеню "бакалавр" спец. 122 “Комп'ютерні 99 науки ” денної і заочної форм навчання. Частина 1 / Уклад.: О.М. М’якшило, О.В. Харкянен – К.: НУХТ, 2017 – 33 с..
13. Проектування інформаційних систем. [Електронний ресурс]: лабораторний практикум для студ. освітнього ступеню "бакалавр" спец. 122 “Комп'ютерні науки ” денної і заочної форм навчання. Частина 2 "Проектування клієнтського додатку" / Уклад.: О.М. М’якшило, О.В. Харкянен – К.: НУХТ, 2017 – 33 с..
14. Проектування та розробка програмного забезпечення [Електронний ресурс]: лабораторний практикум для здобувачів освіт. ступ. "Бакалавр" спец. 122 "Комп'ютерні науки" освіт.-проф. програм "Комп'ютерні науки",

"Інформаційні системи та штучний інтелект" ден. та заоч. форм навч. / уклад.: О. М. М'якшило, О. В. Харкянен ; Нац. ун-т харч. технол. — Київ : НУХТ, 2022. — 102 с. — каф. інформаційних технологій, штучного інтелекту і кібербезпеки.

15. Проектування інформаційних систем [Електронний ресурс]: методичні рекомендації до виконання курсового проекту для студентів освітнього ступеня «Бакалавр» спеціальності 122 «Комп’ютерні науки» денної та заочної форм навчання./Уклад.: О. М. М’якшило, О. В. Харкянен: НУХТ, 2018. – 24 с.

ДОДАТКИ

Додаток А. Діаграми та схеми

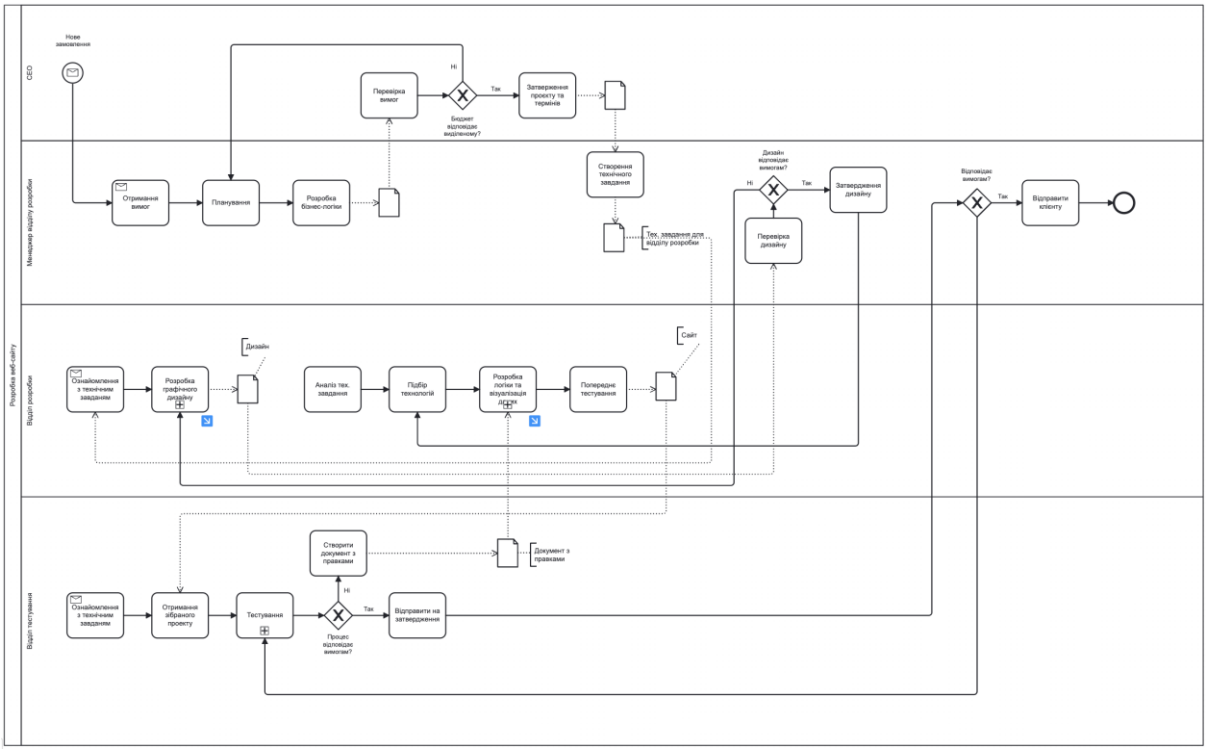

Рисунок А.1 – BPMN-діаграма процесу створення веб-сайту

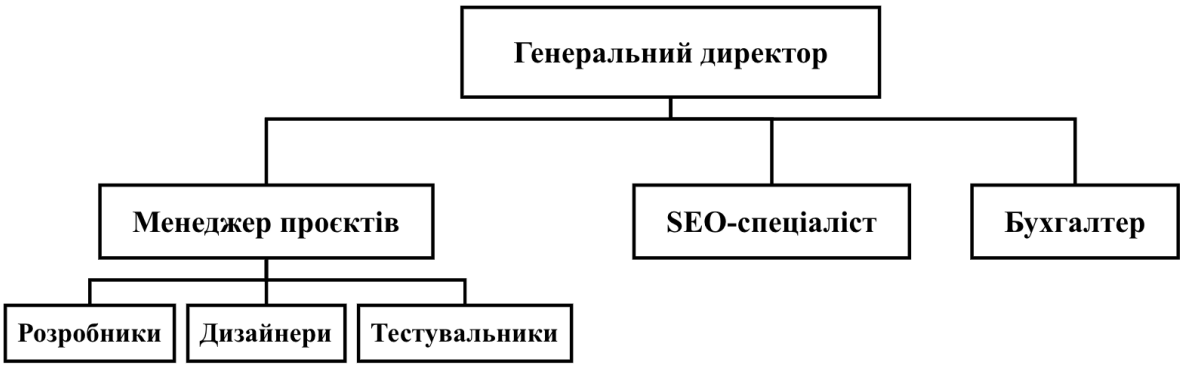

Рисунок А.2 – Організаційно-функціональна схема ФОП «Горенко Тарас Анатолійович»

Додаток Б. Інтерфейс користувача системи

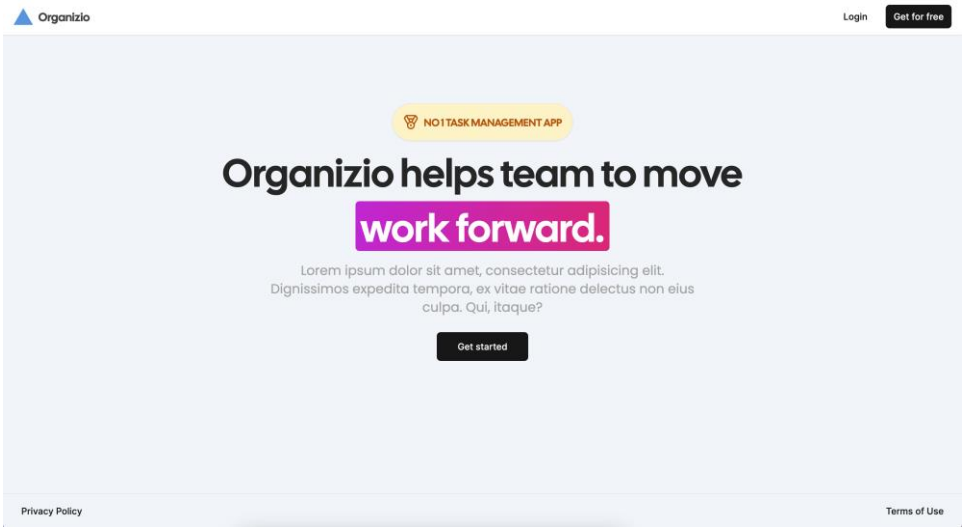

Рисунок Б.1 – Презентаційна сторінка

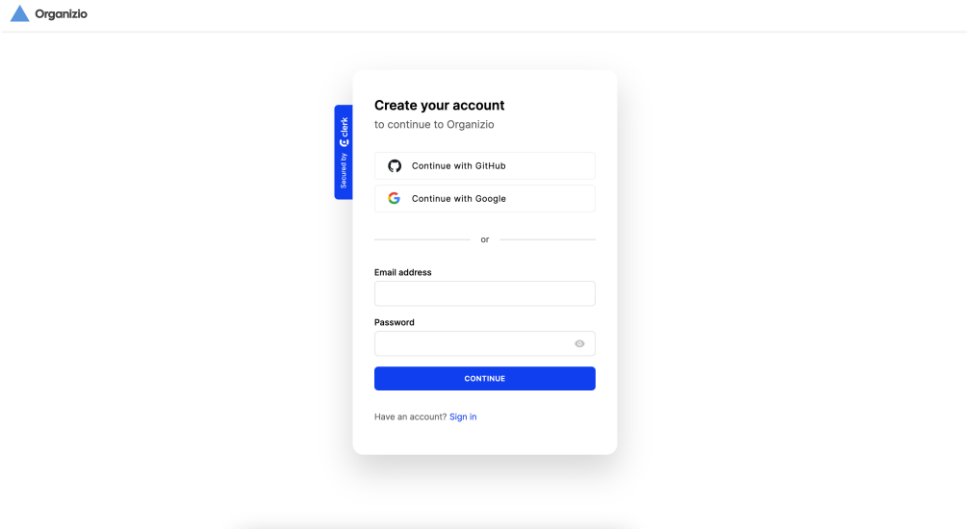

Рисунок Б.2 – Сторінка реєстрації

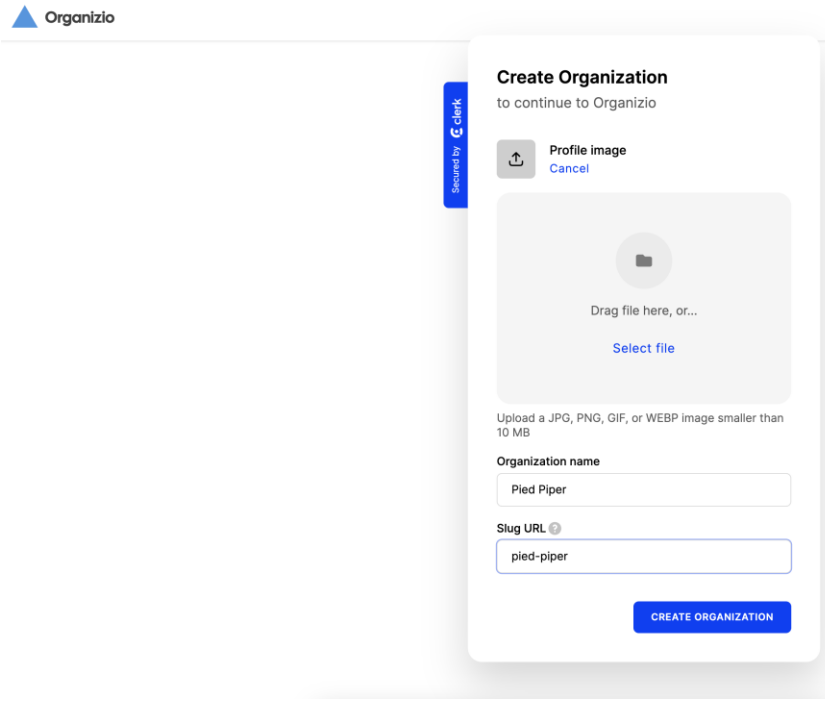

Рисунок Б.3 – Сторінка створення організації

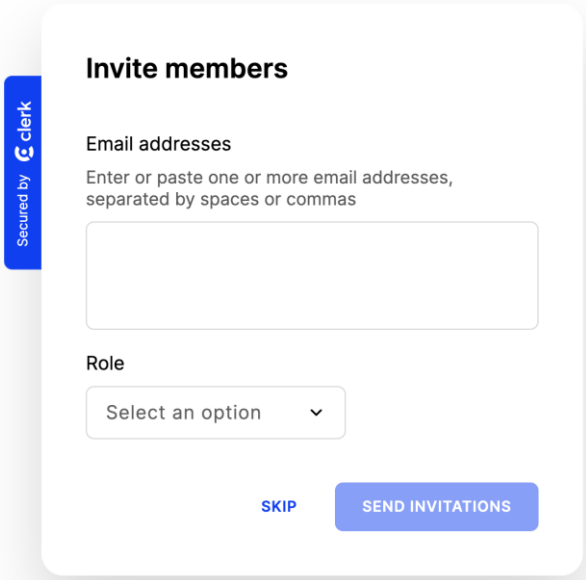

Рисунок Б.4 – Форма для запрошення
інших користувачів до створеної організації

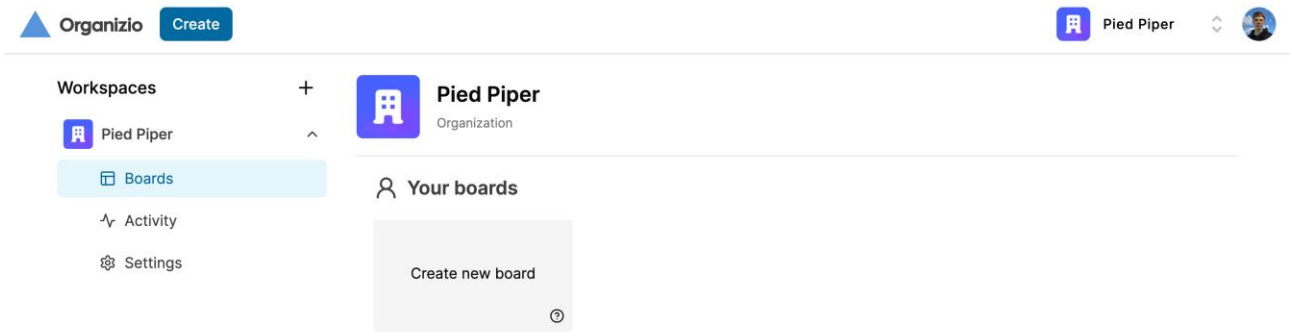

Рисунок Б.5 – Головна сторінка дашборду

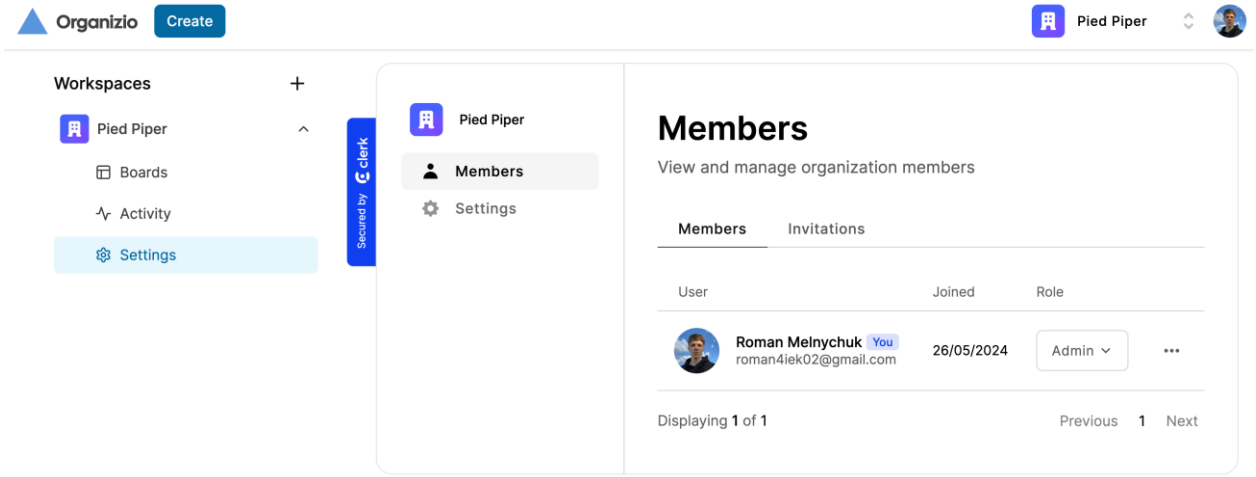

Рисунок Б.6 – Сторінка налаштувань організації.

Список учасників

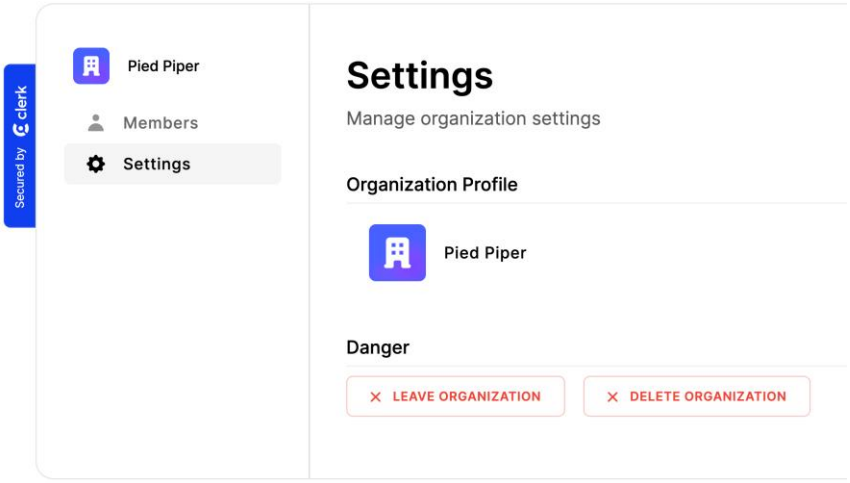

Рисунок Б.7 – Сторінка налаштувань організації. Загальні налаштування

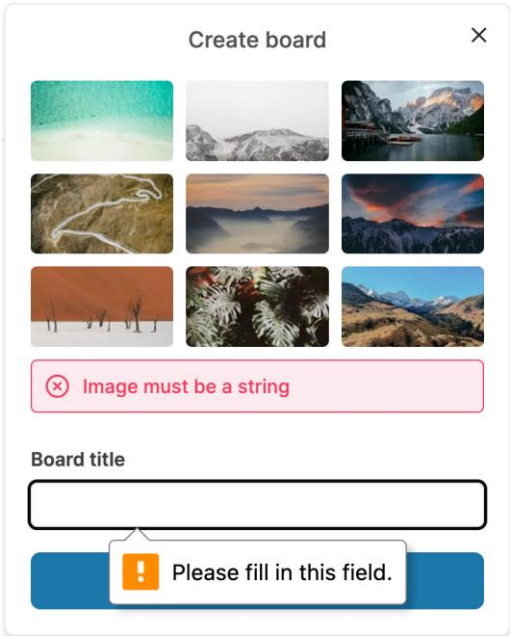

Рисунок Б.8 – Створення дошки. Незаповнені поля

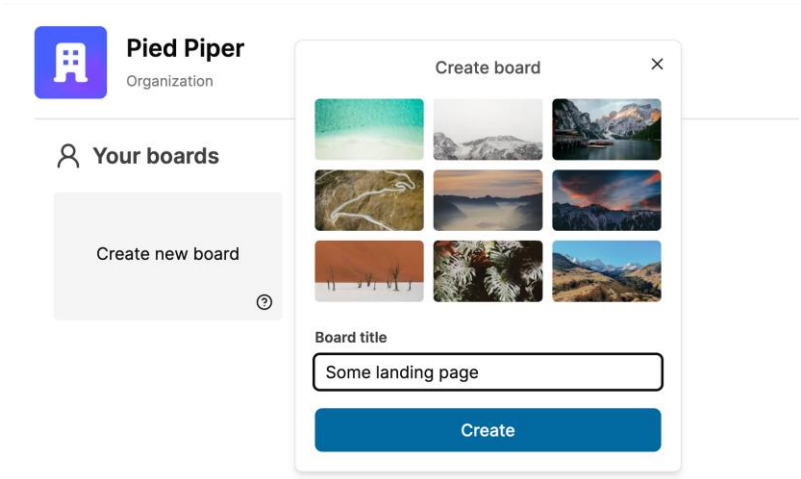

Рисунок Б.9 – Створення дошки. Заповнені поля

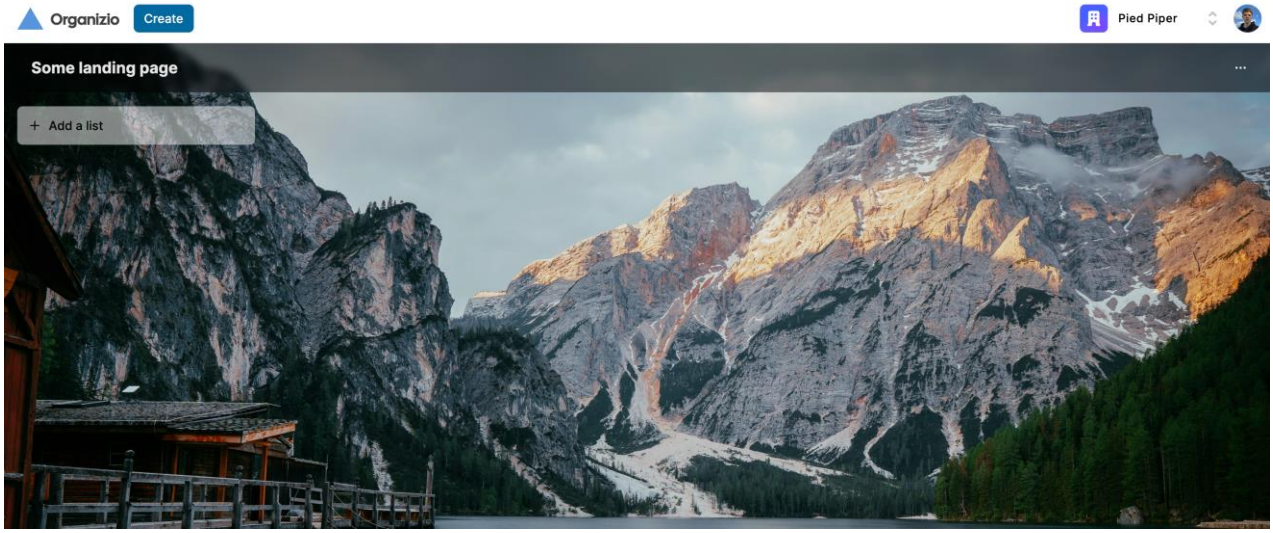

Рисунок Б.10 – Сторінка створеної дошки

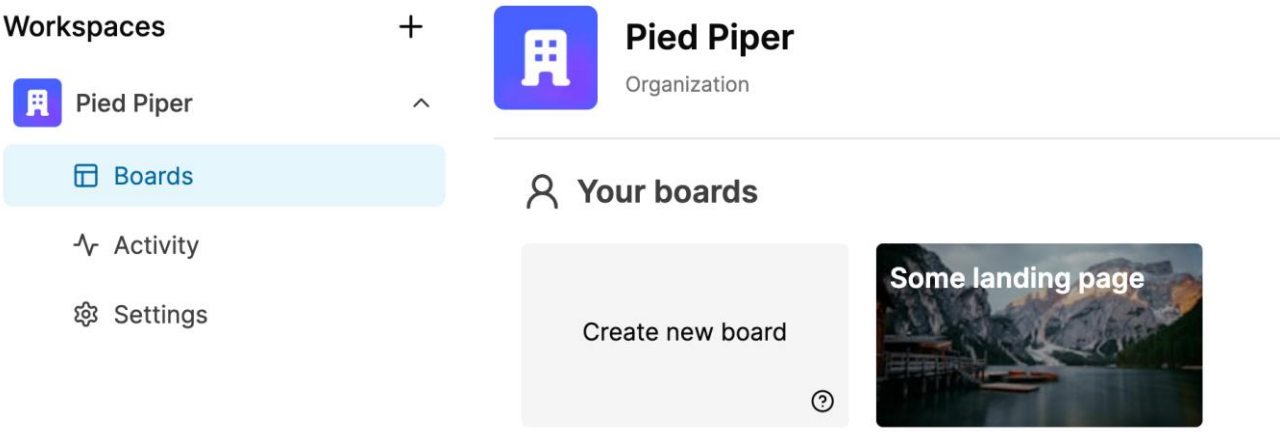

Рисунок Б.11 – Створена дошка в списку дошок

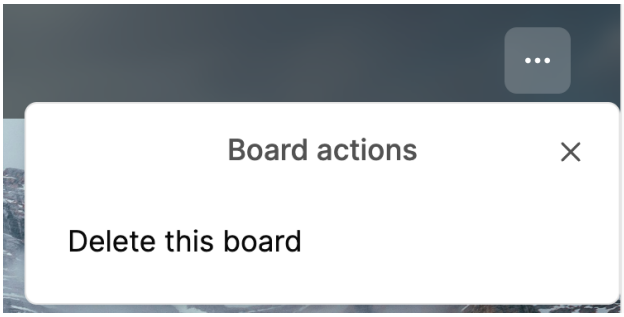

Рисунок Б.12 – Опція видалення дошки

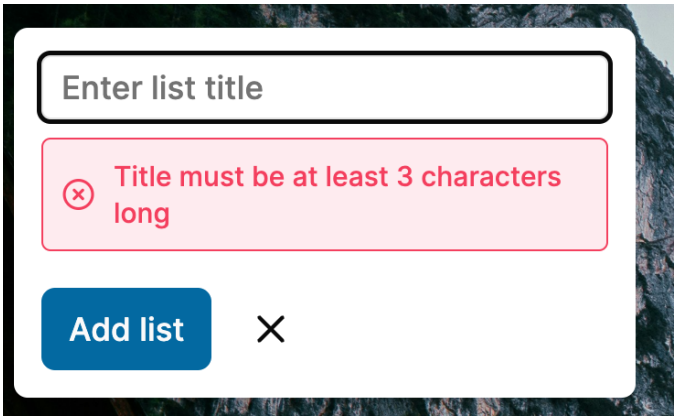

Рисунок Б.13 – Незаповнена назва списку

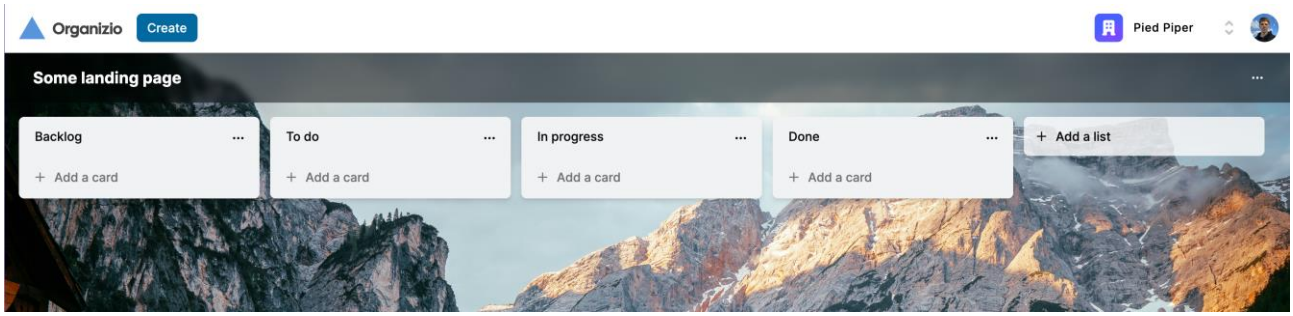

Рисунок Б.14 – Створені списки

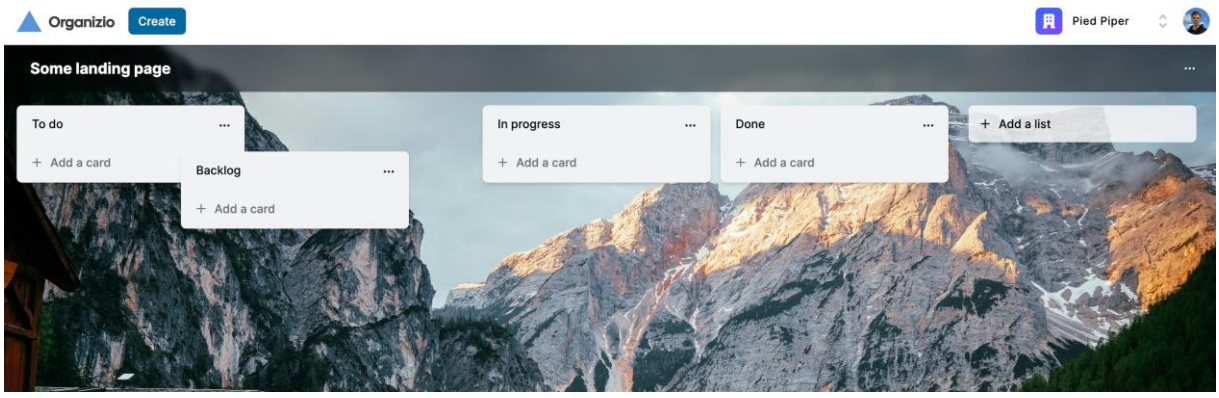

Рисунок Б.15 – Перетягування списку

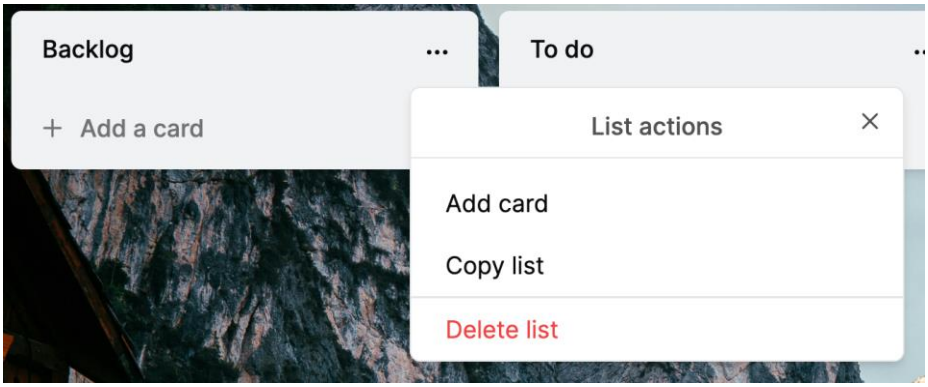

Рисунок Б.16 – Опції списку

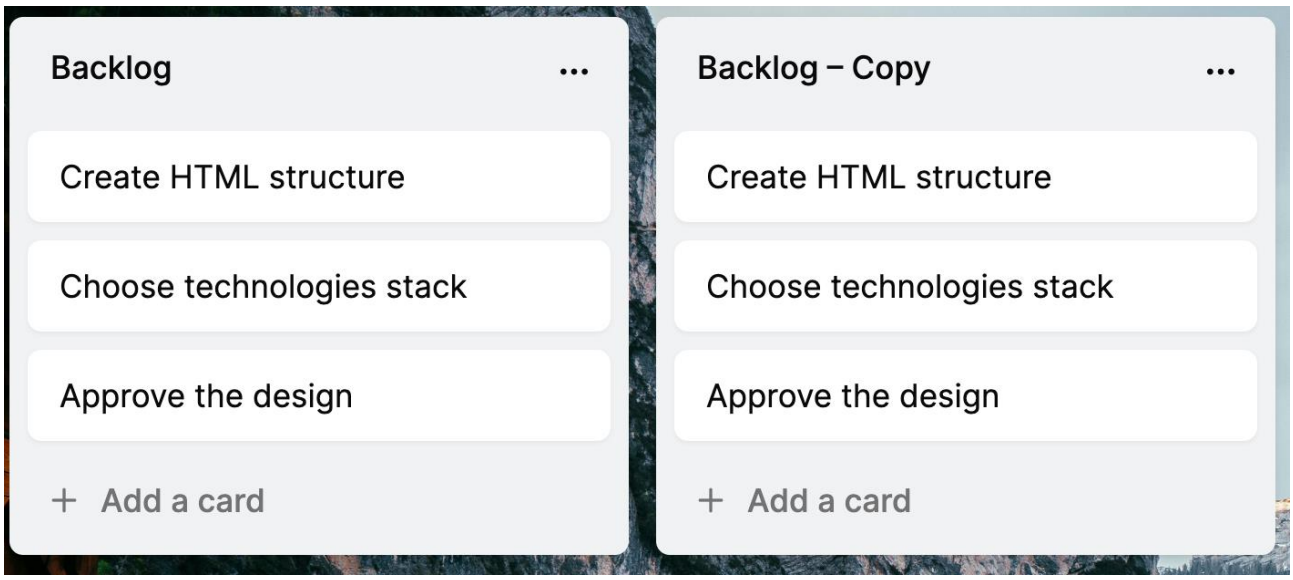

Рисунок Б.17 – Оригінальний та скопійований список

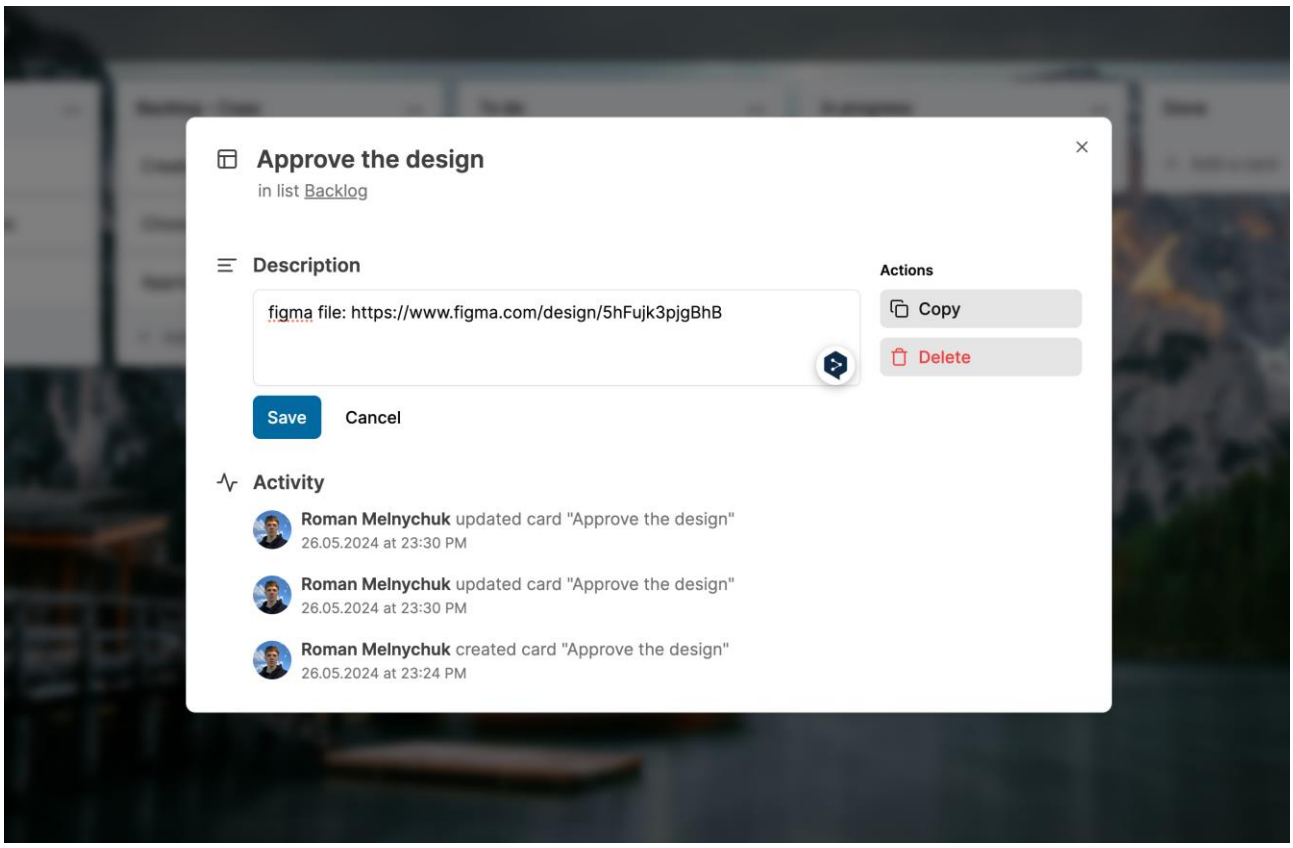

Рисунок Б.18 – Модальне вікно картки

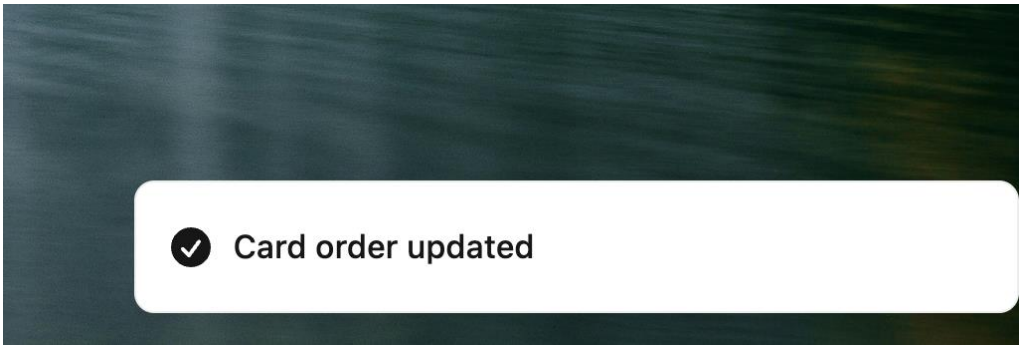

Рисунок Б.19 – Спливаюче повідомлення

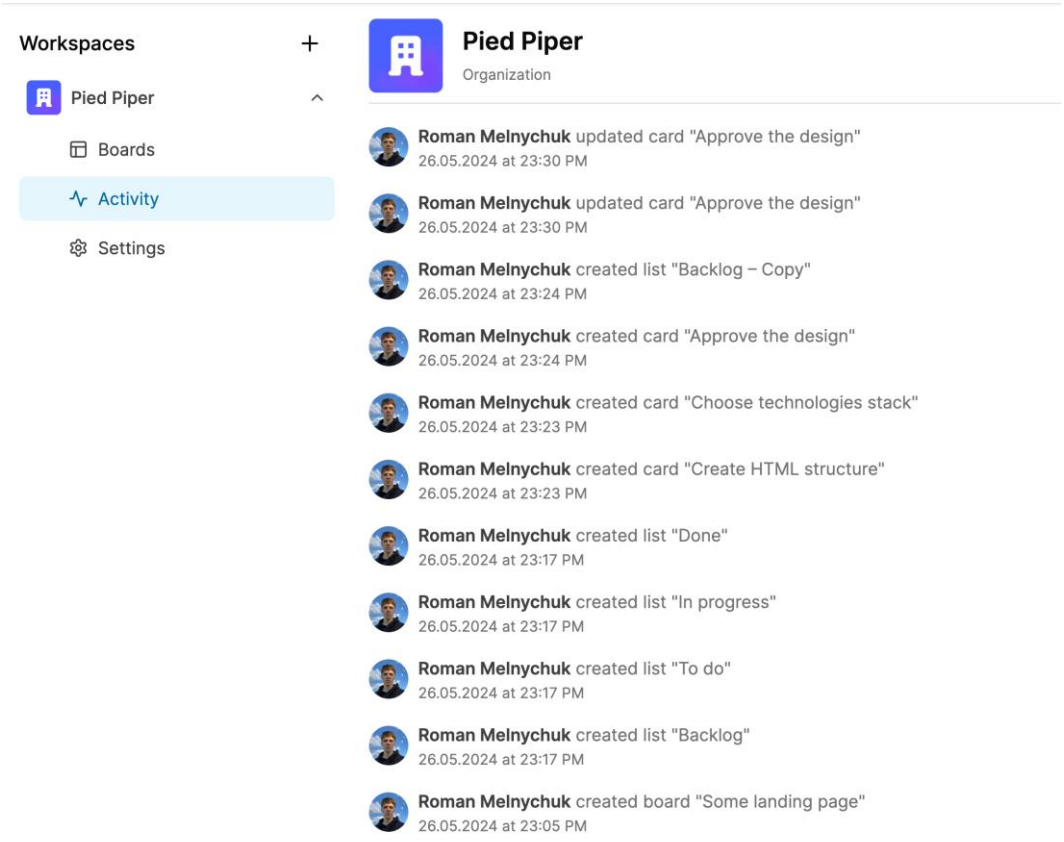

Рисунок Б.20 – Активність користувача

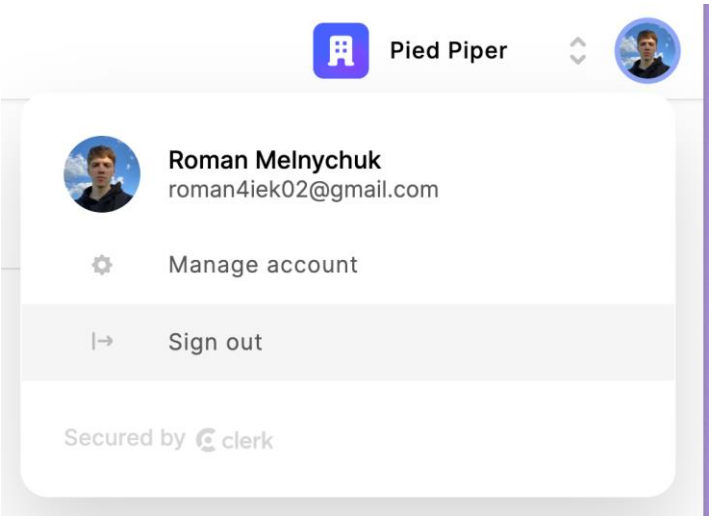

Рисунок Б.21 – Вихід з акаунта

Додаток В. Фрагменти коду програми

Додаток В.1 – Фрагмент код з файла «create-audit-log.ts»

```
import { auth, currentUser } from '@clerk/nextjs';
import { ACTION, ENTITY_TYPE } from '@prisma/client';
import { db } from './db';

interface Props {
    entityId: string;
    entityType: ENTITY_TYPE;
    entityTitle: string;
    action: ACTION;
}

export const createAuditLog = async (props: Props) => {
    try {
        const { orgId } = auth();
        const user = await currentUser();

        if (!user || !orgId) {
            throw new Error('User not found');
        }

        const { entityId, entityType, entityTitle, action } = props;

        await db.auditLog.create({
            data: {
                orgId,
                userId: user?.id,
                userImage: user?.imageUrl,
                userName: user?.firstName + ' ' + user?.lastName,
```

```
            entityId,
            entityType,
            entityTitle,
            action,
          },
        });
    } catch (error) {
      console.error('[AUDIT_LOG_ERROR]', error);
    }
};
```

Додаток В.2 – Фрагмент код з файла «create-safe-action.ts»

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

export type FieldErrors<T> = {
   [K in keyof T]?: string;
};

export type ActionState<TInput, TOutput> = {
   fieldErrors?: FieldErrors<TInput>;
   error?: string | null;
   data?: TOutput;
};

export const createSafeAction = <TInput, TOutput>(
   schema: z.Schema<TInput>,
   handler: (validatedData: TInput) => Promise<ActionState<TInput, TOutput>>,
) => {
   return async (data: TInput): Promise<ActionState<TInput, TOutput>> => {
      const validationResult = schema.safeParse(data);
      if (!validationResult.success) {
         return {
            fieldErrors: validationResult.error.flatten().fieldErrors as
FieldErrors<TInput>,
         };
      }
      return handler(validationResult.data);
   };
};
```

Додаток В.3 – Фрагмент код з файла «db.ts»

```
import {PrismaClient} from '@prisma/client';

declare global {
  var prisma: PrismaClient | undefined;
}

export const db = globalThis.prisma || new PrismaClient();

if(process.env.NODE_ENV !== 'production') {
  globalThis.prisma = db;
}
```

Додаток В.4 – Код файлу «fetcher.ts»

```
export const fetcher = (url: string) => fetch(url).then((res) => res.json());
```

Додаток В.5 – Код файлу «generate-message.ts»

```
import { ACTION, AuditLog } from '@prisma/client';

export const generateMessage = (log: AuditLog) => {
  const { action, entityTitle, entityType } = log;

  switch (action) {
    case ACTION.CREATE:
      return `created ${entityType.toLowerCase()} "${entityTitle}"`;
    case ACTION.UPDATE:
      return `updated ${entityType.toLowerCase()} "${entityTitle}"`;
    case ACTION.DELETE:
      return `deleted ${entityType.toLowerCase()} "${entityTitle}"`;
    default:
```

```
        return `made unknown action with ${entityType.toLowerCase()}
"${entityTitle}"`;
    }
};
```

Додаток В.6 – Фрагмент код з файла «unsplash.ts»

```
import { createApi } from 'unsplash-js';

export const unsplash = createApi({
    accessKey: process.env.NEXT_PUBLIC_UNSPLASH_ACCESS_KEY!,
    fetch: fetch,
});
```

Додаток В.7 – Фрагмент код з файла «utils.ts»

```
import { type ClassValue, clsx } from "clsx"
import { twMerge } from "tailwind-merge"

export function cn(...inputs: ClassValue[]) {
return twMerge(clsx(inputs))
}
```

Додаток В.8 – Фрагмент код з файла «page.tsx»

```
import { auth } from '@clerk/nextjs';
import { db } from '@/lib/db';
import { redirect } from 'next/navigation';
import { ListContainer } from './_components/list-container';
```

```
interface BoardIdPageProps {
    params: {
        boardId: string;
    };
}

const BoardIdPage = async ({ params }: BoardIdPageProps) => {
    const { orgId } = auth();
    if (!orgId) {
        redirect('/select-org');
    }
    const lists = await db.list.findMany({
        where: {
            boardId: params.boardId,
            board: {
                orgId,
            },
        },
        include: {
            cards: {
                orderBy: {
                    order: 'asc',
                },
            },
        },
        orderBy: {
            order: 'asc',
        },
    });
```

```
    return (
        <div className="p-4 h-full overflow-x-auto">
            <ListContainer boardId={params.boardId} data={lists} />
        </div>
    );
};

export default BoardIdPage;
```

Додаток В.9 – Фрагмент коду з файла «layout.tsx»

```
import { db } from '@/lib/db';
import { auth } from '@clerk/nextjs';
import { notFound, redirect } from 'next/navigation';
import React from 'react';
import BoardNavbar from './_components/board-navbar';

export async function generateMetadata({ params }: { params: { boardId: string } }) {
const { orgId } = auth();

if (!orgId) {
return {
title: 'Board',
};
}

const board = await db.board.findUnique({
where: {
id: params.boardId,
orgId,
},
```

```
});

return {
title: board?.title || 'Board',
};
}

const BoardIdLayout = async ({
children,
params,
}: {
children: React.ReactNode;
params: { boardId: string };
}) => {
const { orgId } = auth();

if (!orgId) {
redirect('/select-org');
}

const board = await db.board.findUnique({
where: {
id: params.boardId,
orgId,
},
});

if (!board) {
notFound();
}
```

```
return (
<div
className="relative h-full bg-no-repeat bg-center bg-cover z-10"
style={{ backgroundImage: `url(${board.imageFullUrl})` }}>
<BoardNavbar data={board} />
<div className="absolute inset-0 bg-black/20 -z-10" />
<main className="relative h-full">{children}</main>
</div>
);
};

export default BoardIdLayout;
```

Додаток В.10 – Фрагмент коду з файла «schema.prisma»

```
generator client {
  provider = "prisma-client-js"
}
datasource db {
  provider = "mysql"
  url      = env("DATABASE_URL")
  relationMode = "prisma"
}
model Board {
  id           String @id @default(uuid())
  orgId        String
  title        String
  imageId      String
  imageThumbUrl  String @db.Text
  imageFullUrl   String @db.Text
```

```
  imageUserName   String @db.Text
  imageLinkHTML   String @db.Text
  lists           List[]
  createdAt       DateTime @default(now())
  updatedAt       DateTime @updatedAt
}
model List {
  id              String @id @default(uuid())
  title           String
  order           Int
  boardId         String
  board           Board @relation(fields: [boardId], references: [id], onDelete: Cascade)
  cards           Card[]
  createdAt       DateTime @default(now())
  updatedAt       DateTime @updatedAt
  @@index([boardId])
}
model Card {
  id              String @id @default(uuid())
  title           String
  order           Int
  description     String? @db.Text

  listId          String
  list            List @relation(fields: [listId], references: [id], onDelete: Cascade)
  createdAt       DateTime @default(now())
  updatedAt       DateTime @updatedAt
  @@index([listId])
}
enum ACTION {
```

```
  CREATE
  UPDATE
  DELETE
}
enum ENTITY_TYPE {
  BOARD
  LIST
  CARD
}
model AuditLog {
  id           String @id @default(uuid())
  orgId        String
  action       ACTION
  entityId     String
  entityType     ENTITY_TYPE
  entityTitle    String
  userId       String
  userImage      String @db.Text
  userName       String @db.Text
  createdAt      DateTime @default(now())
  updatedAt      DateTime @updatedAt
}
```

Додаток В.11 – Фрагмент коду з файла «use-action.ts»

```
import { ActionState, FieldErrors } from '@/lib/create-safe-action';
import { useState, useCallback } from 'react';

type Action<TInput, TOutput> = (data: TInput) => Promise<ActionState<TInput, TOutput>>;

interface UseActionOptions<TOutput> {
```

```
    onSuccess?: (data: TOutput) => void;
    onError?: (error: string) => void;
    onComplete?: () => void;
}

export const useAction = <TInput, TOutput>(
    action: Action<TInput, TOutput>,
    options: UseActionOptions<TOutput> = {},
) => {
    const [fieldErrors, setFieldErrors] = useState<FieldErrors<TInput> |
undefined>(undefined);
    const [error, setError] = useState<string | undefined>(undefined);
    const [data, setData] = useState<TOutput | undefined>(undefined);
    const [isLoading, setIsLoading] = useState<boolean>(false);

    const execute = useCallback(
        async (input: TInput) => {
            setIsLoading(true);

            try {
                const result = await action(input);
                if (!result) {
                    return;
                }
                setFieldErrors(result.fieldErrors);
                if (result.error) {
                    setError(result.error);
                    options.onError?.(result.error);
                }
                if (result.data) {
```

```
                setData(result.data);
                options.onSuccess?.(result.data);
            }
        } finally {
            setIsLoading(false);
            options.onComplete?.();
        }
    },
    [action, options],
  );
  return {
    fieldErrors,
    error,
    data,
    isLoading,
    execute,
  };
};
```

Додаток В.12 – Фрагмент коду з файла «button.tsx»

```
import * as React from 'react';
import { Slot } from '@radix-ui/react-slot';
import { cva, type VariantProps } from 'class-variance-authority';
import { cn } from '@/lib/utils';
const buttonVariants = cva(
  'inline-flex items-center justify-center whitespace-nowrap rounded-md text-sm
font-medium ring-offset-background transition-colors focus-visible:outline-none
focus-visible:ring-2 focus-visible:ring-ring focus-visible:ring-offset-2
disabled:pointer-events-none disabled:opacity-50',
  {
    variants: {
```

```
        variant: {
            default: 'bg-primary text-primary-foreground hover:bg-primary/90',
            destructive: 'bg-destructive text-destructive-foreground hover:bg-
destructive/90',
            outline:
              'border border-input bg-background hover:bg-accent hover:text-accent-
foreground',
            secondary: 'bg-secondary text-secondary-foreground hover:bg-
secondary/80',
            ghost: 'hover:bg-accent hover:text-accent-foreground',
            link: 'text-primary underline-offset-4 hover:underline',
            primary: 'bg-sky-700 text-primary-foreground hover:bg-sky-700/90',
            transparent: 'bg-transparent text-white hover:bg-white/20',
            gray: 'bg-neutral-200 text-secondary-foreground hover:bg-neutral-300',
        },
        size: {
            default: 'h-10 px-4 py-2',
            sm: 'h-9 rounded-md px-3',
            lg: 'h-11 rounded-md px-8',
            icon: 'h-10 w-10',
            inline: 'h-auto px-2 py-1.5 text-sm',
        },
    },
    defaultVariants: {
        variant: 'default',
        size: 'default',
    },
  },
);
export interface ButtonProps
```

```
    extends React.ButtonHTMLAttributes<HTMLButtonElement>,
        VariantProps<typeof buttonVariants> {
    asChild?: boolean;
}
const Button = React.forwardRef<HTMLButtonElement, ButtonProps>(
    ({ className, variant, size, asChild = false, ...props }, ref) => {
        const Comp = asChild ? Slot : 'button';
        return (
            <Comp
                className={cn(buttonVariants({ variant, size, className }))}
                ref={ref}
                {...props}
            />
        );
    },
);
Button.displayName = 'Button';
export { Button, buttonVariants };
```

Додаток В.13 – Фрагмент коду з файла «list-container.tsx»

```
'use client';
import { DragDropContext, Droppable } from '@hello-pangea/dnd';
import { ListWithCards } from '@/types';
import { ListForm } from './list-form';
import { useEffect, useState } from 'react';
import { ListItem } from './list-item';
import { useAction } from '@/hooks/use-action';
import { updateListOrder } from '@/actions/update-list-order';
import { updateCardOrder } from '@/actions/update-card-order';
import { toast } from 'sonner';
```

```
interface ListContainerProps {
    data: ListWithCards[];
    boardId: string;
}

function reorder<T>(list: T[], startIndex: number, endIndex: number) {
    const result = Array.from(list);
    const [removed] = result.splice(startIndex, 1);
    result.splice(endIndex, 0, removed);

    return result;
}

export const ListContainer = ({ data, boardId }: ListContainerProps) => {
    const [orderedData, setOrderedData] = useState<ListWithCards[]>(data);

    const { execute: executeUpdateListOrder } = useAction(updateListOrder, {
        onSuccess: () => {
            toast.success('List order updated');
        },
        onError: (error) => {
            toast.error(error);
        },
    });

    const { execute: executeUpdateCardOrder } = useAction(updateCardOrder, {
        onSuccess: () => {
            toast.success('Card order updated');
        },
```

```
    onError: (error) => {
      toast.error(error);
    },
  });

  useEffect(() => {
    setOrderedData(data);
  }, [data]);

  const onDragEnd = (result: any) => {
    const { destination, source, type } = result;

    if (!destination) {
      return;
    }

    // Dropped in the same position
    if (destination.droppableId === source.droppableId && destination.index ===
source.index) {
      return;
    }

    // moving a list
    if (type === 'list') {
      const items = reorder(orderedData, source.index, destination.index).map(
        (item, index) => ({ ...item, order: index }),
      );
      setOrderedData(items);
      executeUpdateListOrder({ items, boardId });
      return;
```

```
        }

        // moving a card
        if (type === 'card') {
            let newOrderedData = [...orderedData];

            // source and destination list
            const sourceList = newOrderedData.find((list) => list.id ===
source.droppableId);
            const destList = newOrderedData.find((list) => list.id ===
destination.droppableId);

            if (!sourceList || !destList) {
                return;
            }

            // check if card exists in the source list
            if (!sourceList.cards) {
                sourceList.cards = [];
            }

            // check if card exists in the destination list
            if (!destList.cards) {
                destList.cards = [];
            }

            // moving the card in the same list
            if (source.droppableId === destination.droppableId) {
                const reorderedCards = reorder(sourceList.cards, source.index,
destination.index);
```

```
    reorderedCards.forEach((card, idx) => {
      card.order = idx;
    });

    sourceList.cards = reorderedCards;

    setOrderedData(newOrderedData);

    executeUpdateCardOrder({
      boardId,
      items: reorderedCards
    });


  } else { // user moves a card to another list

    // remove card from source list
    const [movedCard] = sourceList.cards.splice(source.index, 1);

    // assign the new listId to the moved card
    movedCard.listId = destination.droppableId;

    // add card to destination list
    destList.cards.splice(destination.index, 0, movedCard);

    sourceList.cards.forEach((card, idx) => {
      card.order = idx;
    });
```

```
            // update the order of the cards in the destination list
            destList.cards.forEach((card, idx) => {
              card.order = idx;
            });

            setOrderedData(newOrderedData);

            executeUpdateCardOrder({
              boardId,
              items: destList.cards
            })
          }
        }
      };

      return (
        <DragDropContext onDragEnd={onDragEnd}>
          <Droppable droppableId="lists" type="list" direction="horizontal">
            {(provided) => (
              <ol
                {...provided.droppableProps}
                ref={provided.innerRef}
                className="flex gap-x-3 h-full">
                {orderedData.map((list, index) => {
                  return <ListItem key={list.id} index={index} data={list} />;
                })}
                {provided.placeholder}
                <ListForm />
                <div className="flex-shrink-0 w-1" />
              </ol>
```

```
            )}
          </Droppable>
        </DragDropContext>
    );
};
```