Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для управління та контролю якості праці співробітників ІТ-компаній
(тема)

Виконав:
студент 4 курсу, групи ПЗПІ-20-4

Бочаров В. О.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. кафедри ПІ Ворочек О. Г.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Бочарову Владиславу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна системи для управління та контролю якості праці співробітників ІТ-компаній

Затверджена наказом по університету від 20.05. 2024р. № 471 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 14.06.2024

3. Вихідні дані до роботи створення програмної системи управління та контролю якості праці співробітників ІТ-компаній, яка дозволить оптимізувати процеси управління проєктами, підвищити ефективність роботи команди та забезпечити прозорість розподілу ресурсів.

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	08.04.2024	*виконано*
2	Створення специфікації ПЗ	11.04.2024	*виконано*
3	Проектування ПЗ	15.04.2024	*виконано*
4	Розробка ПЗ	15.05.2024	*виконано*
5	Тестування ПЗ	17.05.2024	*виконано*
6	Оформлення пояснювальної записки	20.05.2024	*виконано*
7	Підготовка презентації та доповіді	22.05.2024	*виконано*
8	Попередній захист	11.06.2024	*виконано*
9	Нормоконтроль, рецензування	12.06.2024	*виконано*
10	Здача роботи у електронний архів	13.06.2024	*виконано*
11	Допуск до захисту у зав. кафедри	14.06.2024	*виконано*

Дата видачі завдання 8 квітня 2024р.

Студент (ка)_____ Бочарова В. О.
(підпис)

Керівник роботи _____ доц. кафедри ПІ Ворочек О. Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра, 85 стор., 20 рис., 18 джерел.

УПРАВЛІННЯ, КОНТРОЛЬ, ЯКІСТЬ, KANBAN, ІТ-КОМПАНІЯ, МОВА ПРОГРАМУВАННЯ C#, MICROSOFT SQL SERVER STUDIO, ASP.NET, REACT.

Об’єктом дослідження в даній роботі є розробка програмної системи, призначеної для оптимізації процесів управління та контрою якості праці співробітників ІТ-компаній. Це дослідження присвячене сучасним проблемам, з якими стикають ІТ-компанії, зокрема, необхідність організації робочого процесу та контролю якості праці співробітників.

Метою роботи є розробка системи, яка допоможе користувачам організовувати та управляти проєктами, а ІТ-компаніям допоможе правильно організовувати та управляти проєктами та ресурсами компанії.

Методи розробки базуються на таких технологіях, фреймворки ASP.NET Core та мова програмування C# для серверної частини системи, React та мова програмування Typescript для клієнтської частини, MS SQL Server база даних.

У результаті роботи здійснено програмну реалізацію системи для управління та контролю якості праці співробітників ІТ-компаній. Програмна система складається з серверної частини та клієнтської частини.

CONTROL, IT-COMPANY, C# PROGRAMMING LANGUAGE, MANAGEMENT, QUALITY, ASP.NET, KANBAN, MICROSOFT SQL SERVER STUDIO, REACT

The object of research in this paper is the development of a software system designed to optimize the processes of management and quality control of IT company

employees. This research is devoted to the modern problems faced by IT companies, in particular, the need to organize the workflow and control the quality of employees' work.

The aim of the work is to develop a system that will help users organize, manage projects and help IT companies to properly organize and manage projects and company resources.

The development methods are based on the following technologies: ASP.NET Core framework and C# programming language for the server side of the system, React and Typescript programming language for the client side, MS SQL Server database.

As a result of the work, a software implementation of the system for managing and controlling the quality of work of IT company employees was carried out. The software system consists of a server part and a client part.

Я, Бочаров Владислав Олександрович, студент гр. ПЗПІ-20-4, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система для управління та контролю якості праці співробітників ІТ-компаній», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

Сьогодні IT-компанії відіграють ключову роль у різних галузях, забезпечуючи безперервну роботу інформаційних систем та цифрових технологій. З розвитком технологій та зі зростанням залежності бізнесу від IT-інфраструктури виникає потреба в ефективному управлінні та контролі якості праці IT-компаній.

Актуальність даної роботи зумовлена активним розвитком цифрових технологій та зростанням кількості IT-проєктів. Сучасні компанії стикаються з низкою проблем, пов'язаних з управлінням проєктами, забезпеченням якості розробки та підтримки програмного забезпечення. У зв'язку з цим розробка програмного комплексу для управління та моніторингу якості роботи IT-компаній є важливим завданням.

Метою даної роботи є розробка програмної системи для управління та контролю якості праці співробітників IT-компаній. Основним завданням системи буде забезпечення прозорості та ефективності всіх процесів, пов’язаних із розробкою проєктів. Система забезпечить механізми моніторингу виконання завдань, управління ресурсами, аналізу якості та ефективності проєктів, а також забезпечить можливість швидкого реагування на виникаючі проблеми та зміни в робочому процесі.

Для досягнення цієї мети будуть використані сучасні технології програмування та розробки ПЗ, такі як фреймворки ASP.NET Core та React, а також інструменти для автоматизації тестування та контролю якості. Розроблена система дозволить IT-компаніям підвищити ефективність своєї роботи, забезпечивши більш високу якість програмного забезпечення та задоволеність клієнтів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Сучасний сектор інформаційних технологій є одним із сегментів економіки, що найбільш динамічно розвивається [1]. З кожним роком спостерігається збільшення кількості IT-компаній, розширення спектра пропонованих послуг та постійне вдосконалення технологій. Роль IT у нашому повсякденному житті стає все більш важливою, а сфера інформаційних технологій стає ключовим фактором успішного розвитку багатьох інших галузей.

Однак, поряд зі зростанням активності в IT-сфері, виникають і нові проблеми, з якими стикаються як компанії-розробники, так і їх клієнти. Однією з таких проблем є забезпечення високої якості розробки та функціонування програмного забезпечення. IT-проєкт, який є цілеспрямованим стратегічним заходом, спрямованим на розвиток або модернізацію технологічних і бізнес-процесів, включає створення або вдосконалення програмних і програмно-технічних комплексів, розробку відповідної технічної та організаційної документації, а також прийняття управлінських рішень та заходи щодо їх реалізації [2]. Неполадки у роботі інформаційних систем, невідповідність вимогам клієнтів, а також проблеми з управлінням проєктами в IT-компаніях можуть спричинити серйозні фінансові втрати, втрати довіри з боку замовників і навіть втрати репутації на ринку.

У контексті цієї проблематики стає очевидною необхідність ефективних систем управління та контролю якості роботи IT-компаній. Такі системи повинні надавати комплексний інструментарій для моніторингу процесів розробки, контролю якості програмного забезпечення, що випускається, а також управління проєктами та персоналом.

Одним з основних завдань таких систем є забезпечення прозорості та контролю за кожним етапом життєвого циклу програмного продукту – починаючи від аналізу та проєктування, і закінчуючи тестуванням, впровадженням та підтримкою. Це дозволяє своєчасно виявляти та усувати можливі недоліки,

мінімізуючи ризики виникнення помилок та збоїв у роботі програмного забезпечення.

Водночас важливим аспектом стає аналіз ефективності роботи персоналу, управління ресурсами та розподіл завдань, що дозволяє оптимізувати робочий процес та підвищити загальну продуктивність команди розробників.

У зв’язку з цим розробка програмної системи для управління та контролю якості праці співробітників ІТ-компаній стає актуальним завданням, спрямованим на забезпечення ефективного управління проєктами, контролю якості розробки та підтримки програмного забезпечення, а також оптимізації виробничих процесів в ІТ-компаніях.

Метою даного проєкту є розробка та впровадження програмної системи, призначеної для управління та контролю якості роботи ІТ-компаній. В ході роботи буде проведено аналіз підходів управління проєктами в ІТ-сфері, виявлені основні проблеми та недоліки, а також визначені вимоги до системи, що розробляється. Результатом роботи стане розробка програмної системи, здатної надати зручний інструмент для ефективного управління всіма аспектами проєкту, забезпечувати контроль якості розробки, підтримки програмного забезпечення та управління ресурсами, а також підвищувати продуктивність та ефективність ІТ-компаній.

1.2 Аналіз аналогів

На сучасному ринку програмних рішень для управління проєктами представлено доволі багато різноманітних продуктів, покликаних полегшити роботу як окремих виконавців, так і цілих команд у сфері інформаційних технологій. Одним із найпопулярніших інструментів у цій сфері є Trello [3]. Trello — це програмна система, яка надає користувачам можливість організовувати та керувати проєктами за допомогою системи дошки з картками. На рисунку 1.1 можливо продивитись інтерфейс Trello.

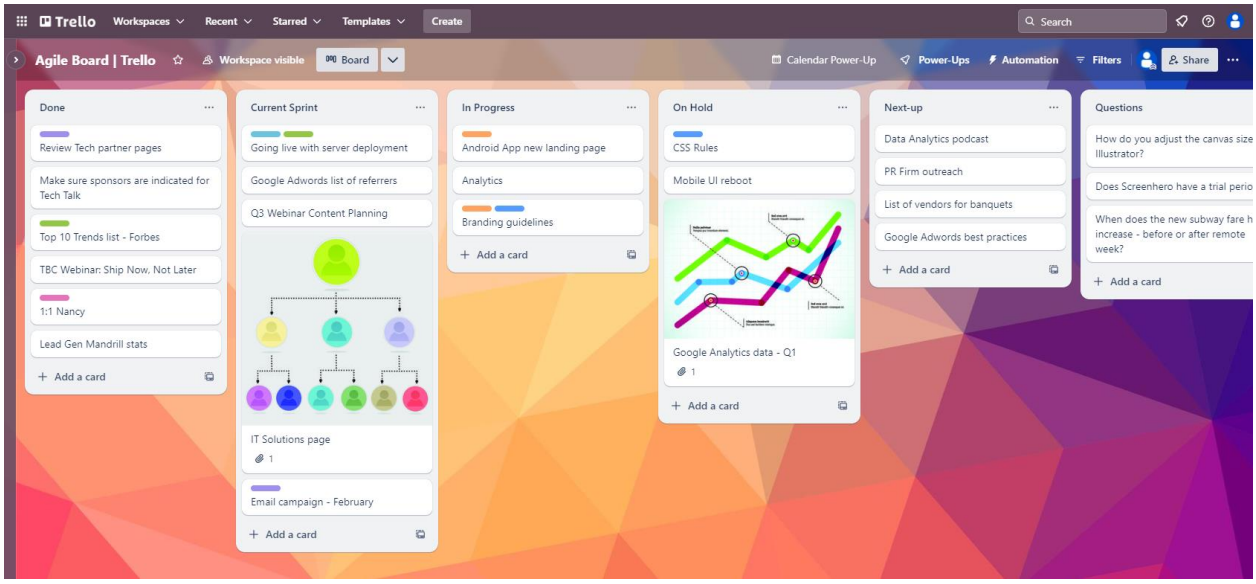


Рисунок 1.1 – Інтерфейс Trello

Trello дозволяє користувачам створювати дошки проєктів та різноманітні списки завдань, які можливо організовувати за розділами або етапами. Кожне завдання представлено у вигляді картки, яка може містити різну інформацію, таку як опис, терміни виконання, вкладені файли та коментарі. Ці картки можливо переміщувати між списками, щоб відстежувати хід виконання завдання. Завдяки своїй гнучкості та простоті Trello можна успішно використовувати для керування проєктами різного розміру та складності.

Однією з найважливіших переваг Trello є інтуїтивно зрозумілий інтерфейс, який легко настроюється, що дозволяє користувачам швидко та легко звикнути до системи. Завдяки своїй простоті та зрозумілості Trello стає улюбленим інструментом для широкого кола користувачів, включаючи як досвідчених IT-фахівців, так і керівників проєктів, для ефективного використання якого не потрібні спеціальні навички чи великий досвід. Інтуїтивно зрозумілий інтерфейс Trello забезпечує швидкий доступ до всіх необхідних функцій, дозволяючи користувачам зосередитися на суті своїх завдань, а не на тому, як правильно налаштувати програму. Це робить Trello привабливим вибором для професіоналів, які цінують зручність і простоту в роботі, але не хочуть витрачати багато часу на навчання та адаптацію до нового інструменту.

Однак, попри свою популярність, Trello також має деякі обмеження. Наприклад, відсутність вбудованих функцій аналітики та звітності може обмежити можливості користувачів відстежувати продуктивність і прогрес проєкту. Trello також не надає можливості управління бюджетом проєкту та ресурсами компанії, що може бути критично важливим для IT-компаній, які прагнуть комплексного управління своєю діяльністю.

Загалом Trello – це зручний інструмент керування проєктами з простим та інтуїтивно зрозумілим інтерфейсом. Він має кілька переваг, таких як гнучкість конфігурації та простота використання, що робить його популярним вибором для багатьох команд та організацій. Однак IT-компаніям, особливо тим, які потребують ширшого та глибшого управління проєктами та ресурсами, може знадобитися більш потужний і функціональний інструмент.

Ще одним з популярних інструментів управління проєктами в сучасному світі є Asana [4]. На рисунку 1.2 можливо продивитись приклад інтерфейсу Asana.

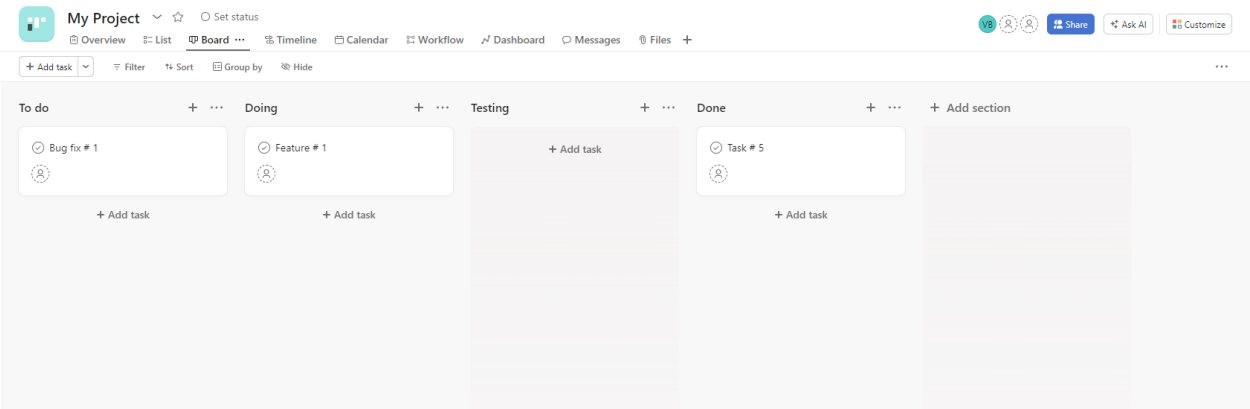


Рисунок 1.2 – Інтерфейс Asana

Подібно до Trello, Asana надає користувачеві зручний інтерфейс і багатий набір функцій для ефективного керування завданнями та проєктами. Платформа Asana позиціюється як потужний інструмент для організації робочих процесів і координації командних зусиль у проєктах будь-якого масштабу.

Asana відрізняється від Trello більш структурованим підходом до управління проєктами. Система Asana заснована на створенні проєктів, завдань та підзадач, що дозволяє користувачам детально розбивати свою роботу на чіткі етапи та кроки. Ключовими елементами Asana є проєкти, які можуть містити необмежену кількість завдань і підзадач, а також команди, які дозволяють групувати учасників проєкту для співпраці та обміну інформацією.

Переваги Asana включають широкий спектр інструментів для планування, керування завданнями та командного спілкування. Користувачі Asana можуть створювати проєкти з різними датами виконання, призначати відповідальних за виконання завдань, встановлювати пріоритети та контролювати прогрес. Багаті можливості Asana також включають гнучкі параметри управління проєктами, такі як календарі, діаграми Ґанта та дошки Kanban, які дозволяють користувачам вибрати відповідний метод організації роботи залежно від специфіки проєкту.

Однією з головних переваг Asana є високий ступінь налаштування. Користувачі можуть створювати власні шаблони проєктів, завдань та звітів, що дозволяє адаптувати платформу до унікальних потреб кожної команди та проєкту. Крім того, Asana надає широкі можливості для автоматизації рутинних завдань, таких як надсилання повідомлень, створення шаблонів завдань і налаштування прав доступу для учасників проєкту.

Окрім цих переваг, варто зазначити, що Asana також має низку недоліків, які можуть обмежувати її використання в певних сценаріях. Істотним недоліком є обмежений функціонал для великих компаній і проєктів. На відміну від деяких інших систем управління проєктами, Asana не пропонує інтегрованих інструментів для управління бюджетом або фінансовими аспектами проєкту. Це може бути складно для компаній, яким потрібно детально відстежувати витрати, аналізувати фінансові показники та керувати бюджетами проєктів.

Ще одним обмеженням є відсутність вбудованих механізмів управління ресурсами та персоналом. Asana зосереджена насамперед на управлінні завданнями та проєктами, чого може бути недостатньо для комплексного управління бізнес-

процесами та ресурсами у великих організаціях. Неможливість розподілити ролі та відповідальність за конкретні завдання в рамках проєкту також може обмежити ефективність команди, особливо в ситуаціях, коли необхідно чітко визначити та контролювати посадові обов'язки.

Варто також зазначити, що Asana має деякі обмеження при роботі з великими обсягами даних і проєктів. Під час роботи над великими проєктами або у великих командах може спостерігатися затримка інтерфейсу, а також зниження продуктивності системи через масштабованість і обмеження продуктивності.

Загалом Asana — це потужний інструмент керування проєктами, який має широкий набір функцій і можливостей, які допоможуть вам ефективно організувати робочий процес. Незважаючи на деякі недоліки, платформа Asana залишається популярним вибором для багатьох компаній і команд, які прагнуть підвищити свою продуктивність і ефективність.

У результаті аналізу аналогів систем управління проєктами можна зробити висновок про те, що на ринку представлена велика різноманітність інструментів і платформ. Кожен із них має свої переваги та недоліки, які важливо враховувати при виборі найбільш прийнятного рішення для конкретних потреб організації. Незважаючи на те, що Trello та Asana надають низку можливостей керування завданнями та проєктами, деякі з їхніх недоліків, як, наприклад відсутність інтегрованих інструментів управління бюджетом і ресурсами, можуть обмежити їх застосування у великих складних організаціях.

У системі, що розробляється, будуть проводитись роботи над тим, щоб усунути ці недоліки, надаючи додатковий функціонал, такий як управління бюджетом компаній і проєктів, детальні статистики для відстеження прогресу проєктів, а також як уміння розподіляти ролі та обов'язки в проєктах. Ці додаткові функції дозволять користувачам нашої системи ефективніше керувати проєктами, покращувати якість роботи та підвищувати загальну продуктивність компанії. Однак слід також зазначити, що деякі недоліки, наявні в наявних аналогах, також

можуть бути враховані в даній системі, та буде проведено робота над їх мінімізацією або усуненням під час розробки та вдосконалення продукту.

1.3 Виявлення та вирішення проблем

У сучасному світі, де IT-компанії відіграють ключову роль у різних галузях, ефективне управління проєктами стає критично важливим для забезпечення успіху та конкурентоспроможності організації. Однак існує низка проблем, з якими організації стикаються під час управління проєктами, зокрема обмежені інструменти для координації роботи, труднощі з відстеженням прогресу та відсутність прозорості у розподілі ресурсів.

Однією з головних проблем є обмежена функціональність доступних інструментів управління проєктами. Незважаючи на широке використання таких платформ, вони часто обмежені базовими можливостями керування завданнями та не забезпечують комплексного підходу до управління проєктами для IT-компаній. Це може призвести до неефективного розподілу ресурсів, відсутності координації між членами команди та, зрештою, до затримок у завершенні проєкту та втрати прибутку.

Іншою поширеною проблемою є відсутність прозорості у відстеженні прогресу проєкту та розподілу ресурсів. Наявні інструменти часто не надають достатньо детальної інформації про статус завдань, час, витрачений на їх виконання, і розподіл ресурсів між різними проєктами. Це ускладнює прийняття обґрунтованих рішень на рівні керівництва та може призвести до недооцінки або перерозподілу ресурсів, що зрештою впливає на ефективність проєкту.

Існує ряд підходів та інструментів вирішення цих проблем. Одна з них – розробка спеціалізованих програмних систем, спрямованих на управління проєктами в IT-компаніях. Ці системи можуть надавати ширший спектр функцій, включаючи управління бюджетом проєкту, детальні статистики та інструменти для розподілу ресурсів відповідно до пріоритетів компанії.

Підсумовуючи, ефективне управління проєктами в IT-компаніях відіграє ключову роль у забезпеченні їх успіху та конкурентоспроможності на ринку. Розуміння проблем, з якими стикаються організації в цій сфері, і використання відповідних рішень, таких як спеціалізовані програмні системи, необхідні для досягнення оптимальних результатів і підвищення продуктивності компанії.

1.4 Постановка задачі

Метою цієї програмної системи є створення інструменту для управління та контролю якості праці IT-компаній, що дозволить оптимізувати процеси управління проєктами, підвищити ефективність роботи команди та забезпечити прозорість розподілу ресурсів. На підставі аналізу переваг та недоліків конкурентів були сформовані основні бізнес-вимоги системи:

– управління проєктами: система повинна дозволяти створювати, редагувати та проводити моніторинг проєктів. Користувачі мають мати змогу призначати завдання та слідкувати за їх виконанням. Візуальні статистики системи та Kanban-дошка допоможуть візуалізувати прогрес. Повинна бути можливість додавати та видаляти учасників проєктів, призначати ролі та відстежувати їхню діяльність. Це дозволить підвищити ефективність управління проєктами, спростити координацію команд та покращити контроль за виконанням завдань;

– статистика продуктивності: система повинна надавати статистику з відстежування продуктивності та якості праці учасників проєктів, що допоможе керівникам ухвалювати обґрунтовані рішення щодо покращення ефективності команди;

– управління компанією: користувачі повинні мати можливість створювати компанії, управляти ресурсами компанії, редагувати дані компанії, відстежувати бюджет, витрати та доходи, а також керувати учасниками компанії та їх ролями. Це дозволить компаніям ефективно контролювати свої фінансові показники, оптимізувати витрати та доходи, а також покращити управління людськими ресурсами;

– управління ресурсами: система повинна забезпечувати інструменти для ефективного розподілу та управління ресурсами компанії. Це сприятиме оптимальному використанню ресурсів, підвищенню продуктивності та зниженню витрат.

1.4.1 Цільова аудиторія

Цільовою аудиторією цієї програмної системи є ІТ-компанії, включаючи як великі організації, так і стартапи з невеликими командами. Окрім ІТ-фахівців, система також може бути корисною для навчання студентів, а також для звичайних користувачів, які можуть використовувати її для керування особистими чи академічними проєктами.

1.4.2 Монетизація

Одним із ключових елементів постановки проблеми є монетизація програмної системи. Наразі система передбачає використання моделі підписки, яка дає користувачам доступ до додаткових функцій та інструментів при умові придбання окремої підписки в системі [5, 6]. Підписка може включати такі опції, як можливість створювати більшу кількість проєктів, додавати більше розділів та завдань, переглядати детальну статистику про проєкти, а також функції, пов’язані з управлінням бюджетами компанії, що дозволить адаптувати програму до потреб різних категорій користувачів.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Основна мета створення цієї програмної системи – це забезпечення користувачів зручним інструментом для керування проєктами та контролю ресурсів менеджерами IT-компаній. Особлива увага має приділятись UI/UX системи задля забезпечення зручності використання системи. Тому за основу програмної системи та як зручний інструмент для управління проєктами було вирішено взяти дошку Kanban [7, 8] для того, щоб легко та швидко управляти процесами проєктів.

Дана програмна система буде складатися з двох частин: серверної частини, де буде розміщена усі основна логіка системи та доступ до даних та клієнтська частин, за допомогою якою користувачі зможуть взаємодіяти з програмною системою.

Далі були сплановані основні функціональні вимоги до програмної системи:

– авторизація та реєстрація користувачів. Необхідно розробити функціонал для авторизації та реєстрації користувачів в системі для їхньої ідентифікації;

– підписки. Цей функціонал необхідний для отримання доходу проєктом. Користувачам будуть доступні декілька видів підписок для забезпечення можливості гнучкого формування набору функцій продукту у відповідності від бізнес-потреб та фінансових можливостей користувача. В першому релізі продукту має бути представлено три види підписок. “Безоплатна підписка” цей формальна підписка, яку не потрібно оформлювати користувачу. Ця підписка буде доступна одразу та буде мати усі стандартні функції та обмеження, тобто при звичайній реєстрації користувача в системі він зможе створювати обмежену кількість проєктів, секцій, завдань та не будуть доступні або також обмежені деякі інші можливості системи. Планується додати “Розширену підпису” для того, щоб надати доступ до додаткових можливостей системи користувачу та будуть зняти деякі обмеження, які будуть накладатися на користувача, які були в нього при звичайній реєстрації. Буде також додана підписка “Enterprise” для того, щоб користувачі могли створювати свої компанії в програмній системі та могли отримати

функціональності пов’язані з управлінням компанії та управлінням ресурсами компанії;

– редагування проєктів. Кожний користувач буде мати змогу створювати проєкти для свої цілей та керувати ними, тобто створювати нові проєкти, редагувати вже створені проєкти та видаляти проєкти;

– редагування секцій в проєктах. Через те, що за основу програмної системи було обрано взяти дошку Kanban, тому в Kanban дошках є секції для того, щоб позначати статуси окремих завдані в них, тому необхідно розробити функціонал для створення, редагування та видалення секцій в проєктах. Необхідно створити тип секцій для того, щоб користувачі могли більш точно, відокремлювати завдання в цих секціях. Ці типи секцій необхідні для того, щоб користувачі розуміли на більш глобальному рівні, що саме робить ця секція. Тому, було вирішено створити три типи секцій: “Зробити”, “Процес роботи” та “Зроблено” або більш зрозумілі аналогічні назви цих типів це: “To do”, “Doing” “Done”;

– редагування завдань в проєкті. Для того, щоб проєкт працював правильно та було зручно їм керувати необхідно зробити функціонал для створення завдань у проєкті для того, щоб учасники проєктів могли розподіляти завдання між собою, змінювати стан завдань незалежно один від одного та управляти цими завданнями. Для цього було необхідно розробити функціонал для того, щоб користувачі могли створювати, редагувати та видаляти завдання;

– управління учасниками проєктів. Зазвичай, велика частина проєктів розробляє не одна людина, а команда. Тому однією з важливих функцій системи є можливість сумісно з командою працювати над одним проєктом. Тому було вирішено додати можливість додавати та видаляти учасників в проєктах;

– управління ролями в проєктах. Цей функціонал буде відповідати за те, щоб було можливо згрупувати учасників проєктів за ролями. Це буде зручно в майбутньому для відстеження статистик та більш зручного розуміння, яка частина людей за яку частину відповідає та до кого можливо звернутися. Цей функціонал, наприклад, буде зручним для ІТ-компаній, бо в деяких компаніях є розподіл за

ролями, такими як, такими як, дизайнери, розробники, тестувальники і т. д. Це допоможе командам більш зручно розподіляти свої ресурси;

– статистики проєктів, де користувачі зможуть продивлятися основні показники, що пов’язані з прогресом проєкту, наприклад: кількість завдань закріплених за кожною секцією, кількість завдань в окремому статусі, кількість завдань закріплених за окремим працівником, та інше;

– управління компанією. Цей функціонал, як і весь функціонал пов’язаний з функціями компанії буде доступний лише після придбання окремого типу підписки, яка згадувалась раніше. Після отримання підписки користувач зможе створити свою компанію та управляти ресурсами компанії, а саме: редагувати дані компанії відслідкувати бюджет компанії, витрати, доходи і т. д., управляти учасниками компанії та управляти ролями. Функції, які доступні компаніям будуть розписані далі;

– управління учасниками компаній. Цей функціонал буде схожий на функціонал управління учасниками проєктів. Цей функціонал буде надавати можливість додавати та видаляти учасників до компанії, переглядати їхні властивості, вартісні показники та відомості щодо якісного рівня роботи даних робітників;

– управління ролями компаній. Цей функціонал буде надавати можливість створити нові ролі для компаній та розподіляти ці ролі серед учасників компаній. Це допоможе в майбутньому відстежувати параметри продуктивності та проводити більш ефективний моніторінг залученості членів команди у роботах проєктів, яка частина людей які ролі має, скільки окремих людей закріплених за окремою роллю в компанії і т. д.;

– статистика компаній. Цей функціонал допоможе відслідковувати набір показників, які пов’язані з ресурсами компанії, від пов’язаних з бюджетом: загальні витрати компанії, доходи, як усієї компанії, так і окремих проєктів і т. д., так і ті, що пов’язані з учасниками компаній;

– повідомлення. Цей функціонал буде відповідати за те, щоб повідомляти користувачів об тих чи інших діях система або дій інших користувачів системи. Повідомлення будуть виконувати дві основні функції. Вони будуть сповіщати користувачів про запрошення до проєкту або до компанії. Вище було зазначено, що програмна система буде мати функціональність управляти учасниками проєктів та компаній. Для того, щоб додати нових учасників до проєктів та до компаній користувачам будуть приходити повідомлення про те, що їх запрошують вступити до проєкту або до компанії. Функціонал повідомлень може розширюватись у майбутньому. До цього функціоналу можуть додаватися такі повідомлення як: закінчення терміну виконання завдання, закріплення користувача за завданням, зміна значень завдання або проєкту за яким закріплений користувач, зміна ролі користувача на проєкті й т. д., але дана роботи не буде містити дані повідомлення, що полегшить розробку проєкту та надасть можливості додавання нових функцій у майбутньому цієї програмної системи.

Необхідно зазначити основні нефункціональні вимоги, які є до цієї системи:

– продуктивність. Система повинна бути здатна обробляти великий обсяг даних та підтримувати роботу з безліччю користувачів. Час відгуку системи має бути мінімальним;

– масштабованість. Система має бути масштабованою, необхідно, щоб система мала можливість масштабуватися, як горизонтально, так і вертикально для покращення продуктивності системи;

– доступність. Система має бути доступна завжди окрім планових або позапланових перерв;

– надійність. Система має бути стабільною та надійною, забезпечуючи високу доступність та мінімальну кількість збоїв;

– зручності використання. Система має бути інтуїтивно зрозумілою та легкою в освоєнні для користувачів різного рівня технічної підготовки. Для цього система повинна мати зрозумілий та простий інтерфейс, зрозумілу структуру меню та навігації;

– безпечність. Система має бути забезпечена багаторівневими заходами захисту, включаючи автентифікацію та авторизацію користувачів, шифрування даних, контроль доступу до функціональних можливостей системи на основі прав користувачів.

Також дана програмна система має наступні обмеження:

– система потребує стабільний інтернет-зв’язок;

– для того, щоб користуватися програмною системою, користувачу необхідно використовувати один з наступних браузерів: Chrome, Mozilla.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування ПЗ

Побудова архітектури та проєктування програмного забезпечення є один з головних етапів розробки програмного забезпечення, оскільки цей етап визначає загальну структуру та функціональні можливості системи. Першим кроком побудови архітектури та проєктування програмного забезпечення є побудова UML діаграм [9]. Однією з перших діаграм, яку потрібно побудувати для розуміння загальної структури та функціональних можливостей системи є UML діаграма прецедентів, яка дозволить ідентифікувати основні функціональні вимоги системи та позначити її взаємодію із зовнішніми користувачами. Діаграма прецедентів є важливим інструментом для аналізу та проєктування користувацьких сценаріїв, що дозволить більш чітко визначити вимоги до системи та її функціональні можливості. На рисунку 3.1 наведена розроблена UML діаграма прецедентів.

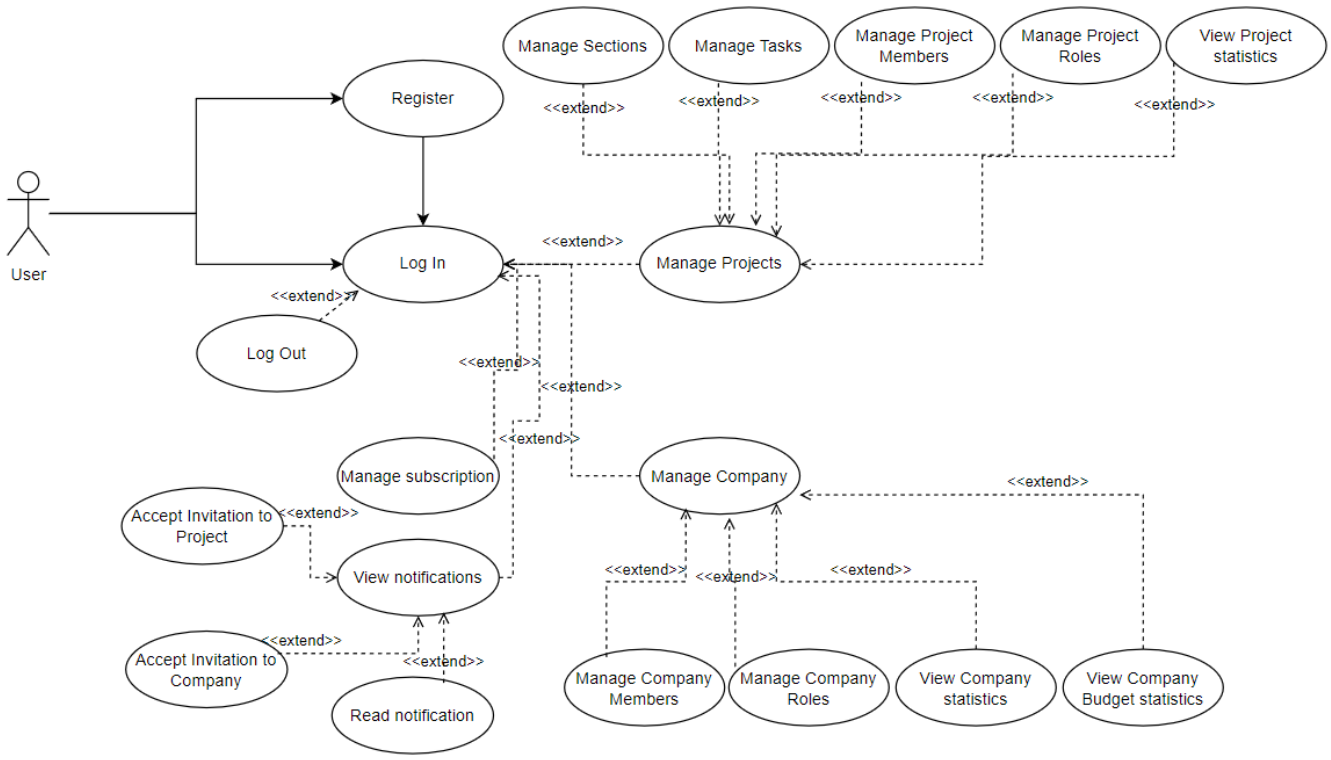


Рисунок 3.1 – UML діаграма прецедентів

Для побудови UML діаграми прецедентів були враховані усі функціональності, які наведені в другому розділі формування вимог до програмної системи.

На цій діаграмі можливо побачити, що з системою буде взаємодіяти лише один тип користувачів звичайні користувачі. Це можуть буди, які фізичні особи, так і компанії, але функціональні можливості не будуть відрізнятись доки ці користувачі не придбають підписку. На даний момент з системою можливо почати взаємодіяти після реєстрації або входу до системи. Після того, як користувач зайде до системи він зможе користуватися функціоналом, який зазначений в другому розділі формування вимог до програмної системи, це: управління проєктами, користувач зможе створювати, редагувати та видаляти проєкти. Під час перегляду проєкту користувач зможе управляти секціями, створювати нові, редагувати вже створені, видаляти, змінювати послідовність секцій, а також користувач зможе додавати завдання в секції, редагувати дані завдань, видаляти їх, змінювати послідовність завдань в секції та переміщати завдання між секціями. В проєкті буде можливість переглядати статистичні дані проєкту для аналізу цих даних надалі користувачем. Також користувач зможе запросити до своїх проєктів інших користувачів для того, щоб вони могли розробляти проєкти разом, а також для цих учасників можливо буде додавати ролі для того, щоб коректна відстежувати діяльність окремих ролей на проєкті. Також користувач зможе переглядати повідомлення системи та приймати запрошення до проєктів та компаній. За бажання користувач зможе придбати підписку, за якою, він зможе отримати додаткову функціональність для створення свої компанії. Після придбання підписки та створення своєї компанії в системі користувач зможе запросити до своєї компанії працівників та роздати їм ролі та управляти даними працівників. До функцій компанії також відносяться такі функції, як: бюджетування, збирання статистичних даних усіх проєктів компанії.

Після побудови UML діаграми прецедентів та з’ясування основних функцій та сценаріїв системи, наступним кроком є створення UML діаграми класів. Ця

діаграма дозволить більш детально визначити структуру програмної системи. На рисунку 3.2 представлена UML діаграма класів.

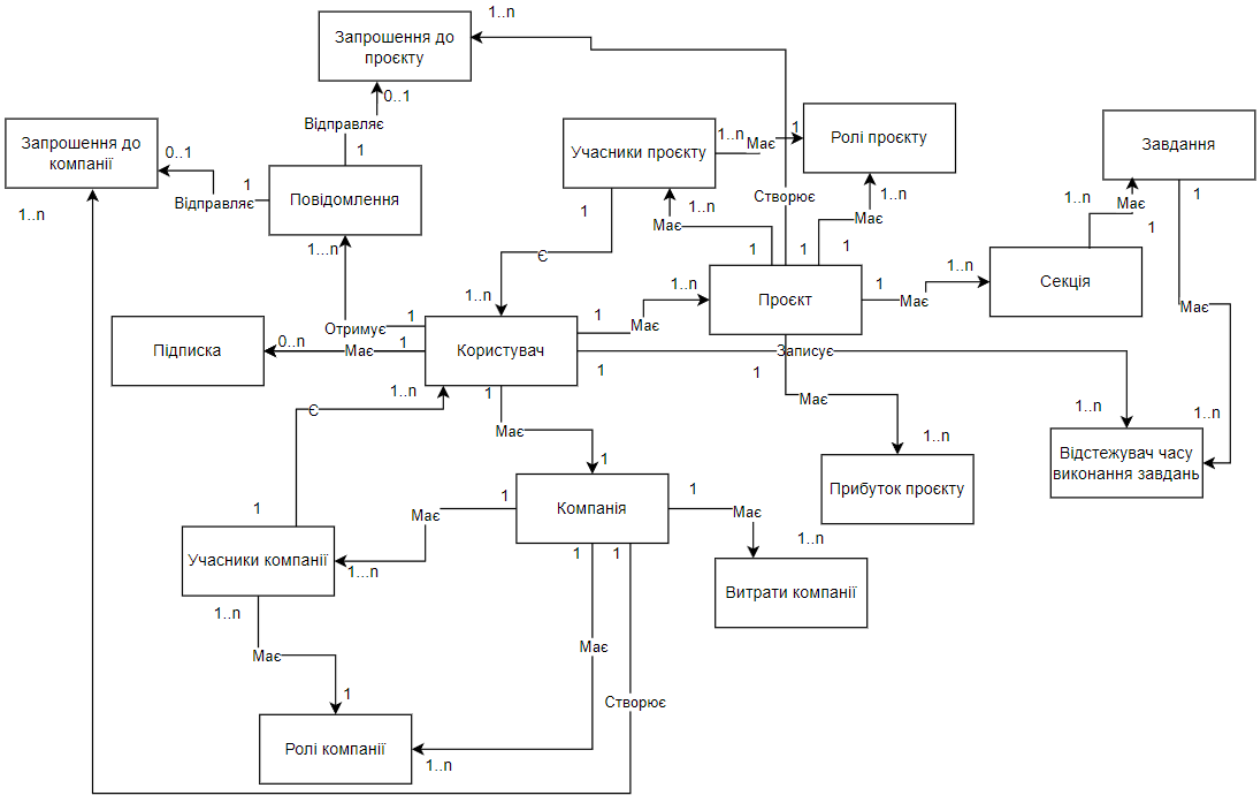


Рисунок 3.2 – UML діаграма класів

На цій діаграмі можливо побачити шістнадцять класів, які з’єднанні між собою зв’язками та разом складають структуру всієї системи.

Основним класом на даній діаграмі є користувач, бо саме користувач та його дії будуть змінювати усі інші класи. Користувач з’єднаний з кількома основними елементами, які своєю чергою з’єднані з іншими важливими елементами. Користувач може мати багато проєктів, проєкт може мати багато секцій, а секції можуть мати багато завдань, також проєкт може мати багато ролей та багато учасників, а там само, проєкт може мати багато прибутків. Користувач має з’єднання з класом, що відстежує час виконання завдань для того, щоб слідкувати скільки часу користувач витратив на завдання. Користувач може мати багато повідомлень та клас повідомлень має зв’язок з усіма іншими повідомленнями. Це

зроблено для того, щоб різні типи повідомлень мали свою інформацію об повідомлені, а батьківський клас повідомлення мав основну інформацію про повідомлення. Батьківський клас повідомлення з'єднаний лише з двома типами повідомлень це запрошення до проєкту та компанії. В майбутньому до цих типів повідомлень можуть додаватися інші типи, тому для роботи з цими повідомленнями була обрана сама така структура класів. Користувач також може мати підписку, оскільки користувач може оформлювати підписку декілька разів, тому зв'язок між класом користувач та класом підписка один до багатьох. А також користувач, після оформлення підписка, зможе мати функціональності пов'язані з компанією, тому створені класи компанії, учасники компанії, ролі компанії та витрати компанії.

3.2 Проєктування архітектури ПЗ

Наступним етапом є проєктування архітектури програмного забезпечення. Проаналізувавши діаграми створенні в розділі UML проєктування ПЗ було обрано клієнт-серверну багаторівневу архітектуру для організації коду на різних рівнях відповідальності. Основними рівнями цієї архітектури будуть: клієнтська частина системи, серверна частина системи та база даних. Таке архітектурне рішення надає високу масштабованість як по горизонталі, так і по вертикалі.

Однією з важливих складових створення архітектури програмного забезпечення є створення UML діаграми розгортання, яка допоможе візуалізувати фізичне розгортання артефактів системи. UML діаграма розгортання, наведена на рисунку 3.3.

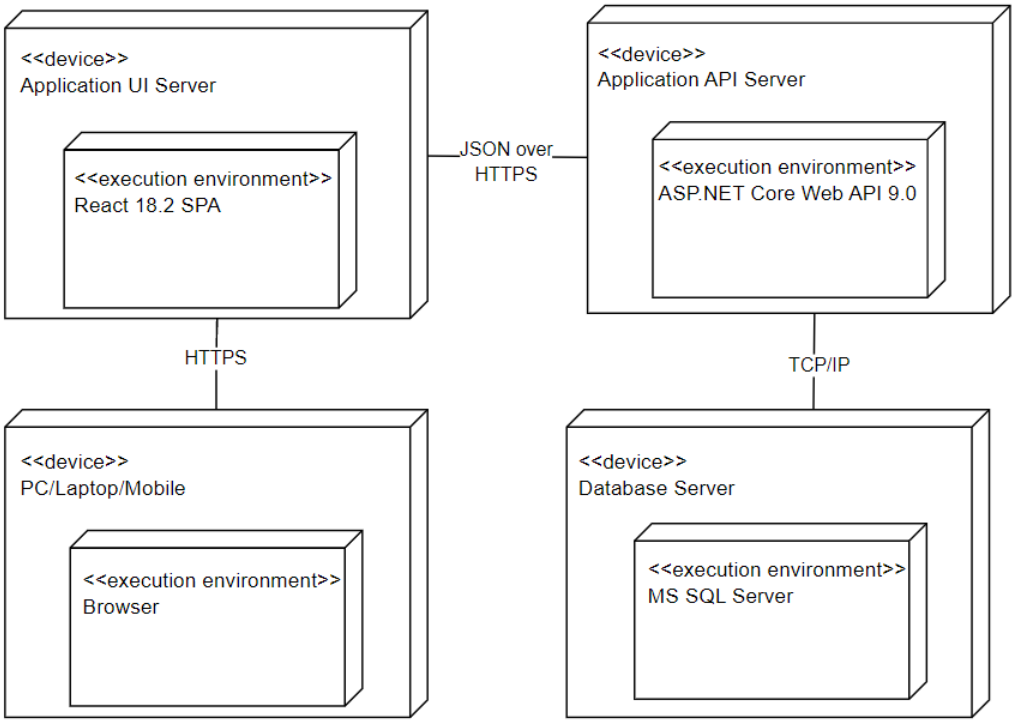


Рисунок 3.3 – UML діаграма розгортання

На цій діаграмі можливо побачити, як будуть розташовуватися пристрої, що вони будуть розміщувати в собі та як будуть взаємодіяти один з одним. За цією діаграмою можливо побачити, що користувач зможе взаємодіяти з системою за допомогою браузеру через свій обраний пристрій, потім буде проходити взаємодія з сервером клієнтської частини за допомогою HTTPS, клієнтська частина, своєю чергою, буде взаємодіяти з серверною частиною системи за допомогою JSON через HTTPS, а серверна частина буде взаємодіяти з базою даних через TCP/IP.

3.3 Проєктування структури зберігання даних

Одним з важливих етапів побудови системи є проєктування структури зберігання даних.

В результаті аналізу предметної області системи та в результаті аналізу архітектури системи була розроблена ER-модель даних для того, щоб структурно відобразити базу даних системи [10]. ER-модель даних, наведена нижче,

відображає основні сутності та зв'язки, які є ключовими для функціонування даної програмної системи. Ця діаграма наведена на рисунку 3.4.

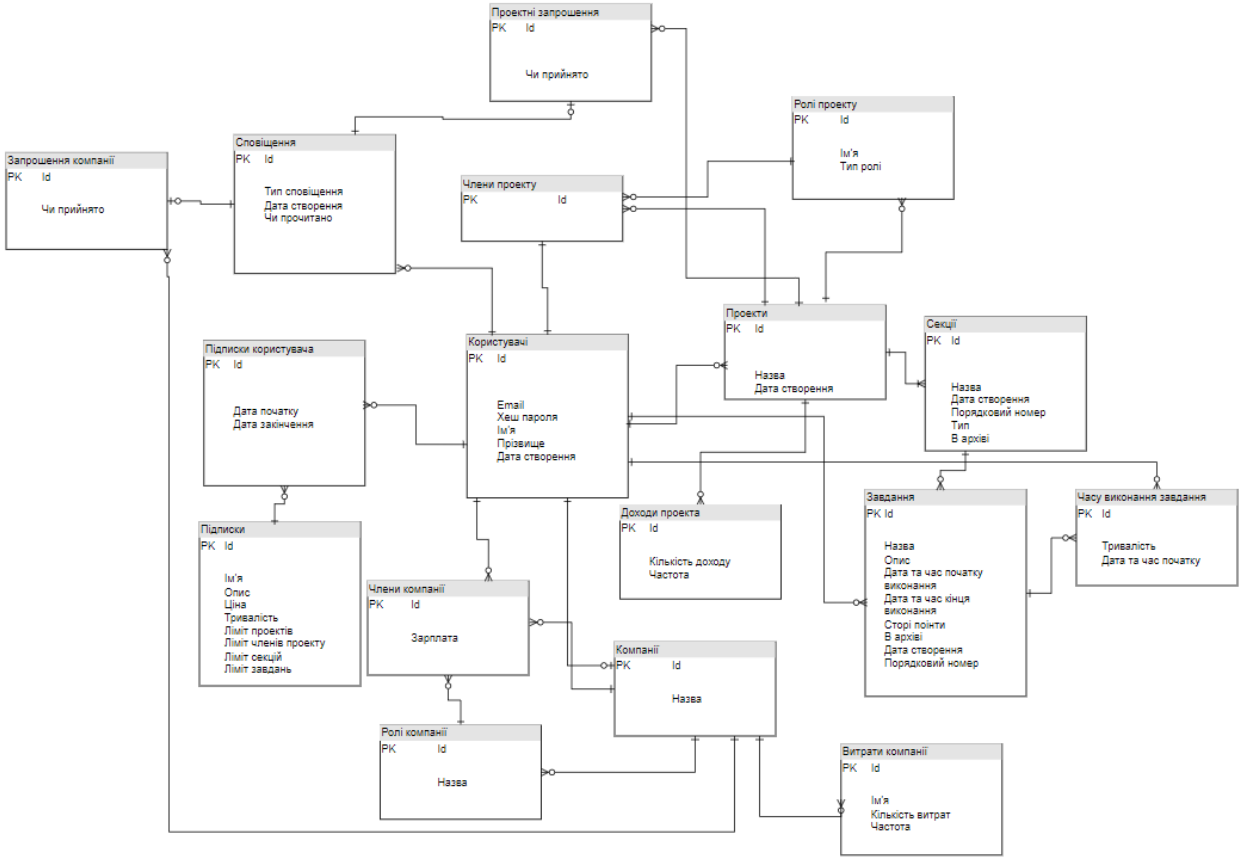


Рисунок 3.4 – ER-модель даних

В цій ER-моделі даних описані основні сутності та зв’язки між цими сутностями, які необхідні для повноцінного функціонування усієї системи. Також ця ER-модель даних відображає основні поля сутностей, які необхідні для повноцінного функціонування системи.

3.4 Приклади найцікавіших алгоритмів та методів

Одним з найцікавіших алгоритмів в даній системі має бути алгоритм роботи Kanban дошки. Алгоритм роботи Kanban дошки поділяється на дві частини для клієнтської частини та для серверної частини. Користувач взаємодіє з дошкою на клієнтській частині системи, змінює дані дошки, після чого відправляється запит

на серверну частину системи для зміни дошки, перевірки правильності зміни даних та збереження відредагованих даних в базі даних.

На клієнтській частині необхідно правильно відображати цю дошку для того, щоб користувачі могли створювати нові секції та завдання, та це все коректно відображати на дошці, а також важливою частиною є реалізація функціонала для зміни порядкових номерів секцій на дошці та завдань в секціях, тобто функціоналу перетягування секції з одного місця на інше, щоб змінювалися порядкові номери секцій на дошці, та теж саме для завдань, щоб у користувача була можливість перетягувати завдання між іншими завданнями в одній секції та між секціями.

Для реалізації функціонала переміщення потрібно, по-перше, при реалізації необхідно отримувати початкові дані для правильної роботи дошки. Необхідно отримати дані ідентифікатора елементу, який буде переміщуватися, який тип цього елементу та дані від куди та куди йде переміщення елемента.

Далі йде алгоритм самого зміщення елементів на дошці. Елементами на дошці позначаються усі елементи, які можуть змінити свій порядковий номер на дошці. В цьому випадку елементами на дошці є секції та завдання.

Спочатку алгоритму потрібна перевірка чи був зміщений елемент, якщо ні, то нічого не відбувається, якщо так, то тоді йде перевірка, який тип елементу був зміщений. Якщо тип елементу, що переміщується є секція, то відповідно потрібно працювати з переміщенням секцій, якщо це завдання, то потрібно переміщувати завдання.

Якщо тип елементу, що переміщується, є секція, тоді береться початковий індекс від куди була переміщена секція та кінцевий індекс куди була переміщена секція, та після чого, змінюється індекс переміщеної секції на кінцевий індекс, потім переглядаються усі секції в проєкті та змінюються порядкові номери секцій, які були пересунуті в результаті зміщення однієї з секцій.

Якщо тип елементу, що переміщується, є завдання, тоді спочатку потрібно перевірити, чи йде це переміщення між однією секцією або між різними секціями. Якщо завдання залишилось в тій же самій секції, але змінило своє місце в секції, то

необхідно змінити індекс цього завдання з початкового індексу на кінцевий індекс та змінити індекси завдань в цій секції, які були пересунуті. Якщо завдання, змінило секцію, в якому воно перебувало з початку пересування, тоді, по-перше, потрібно змінити індекс самого завдання на індекс куди він переноситься, після чого, потрібно змінити порядкові номери завдань в секції з якого завдання переноситься та в секції в яке завдання переноситься.

Таким чином, алгоритм не тільки гарантує, що секції та завдання переміщується плавно та точно через дошку Kanban, але також забезпечує чітку та зрозумілу взаємодію з користувачем. Оцінюючи результати перетягування елементів, алгоритм ретельно контролює порядок і розташування кожного елемента у відповідь на введення користувача.

Коли розділи переміщуються, алгоритм перераховує їхні порядкові номери та змінює їх розташування на дошці, дозволяючи користувачеві контролювати структуру дизайну у спосіб, який найкраще відповідає його потребам. Крім того, під час передачі завдань усередині або між секціями алгоритм гарантує збереження узгодженості даних та послідовності завдань, отже зберігаючи цілісність робочого процесу.

Цей інтелектуальний механізм не тільки покращує взаємодію з користувачем, але й підвищує його ефективність, прискорюючи процеси планування та керування завданнями. Завдяки цьому дошка Kanban стає не тільки інструментом для візуалізації робочого процесу, а й потужним інструментом для організації роботи та досягнення поставлених цілей.

Ще одним цікавим алгоритмом в даній програмній системі є підрахунок продуктивності та якості роботи учасників на проєкті. За основу підрахунку числового показника продуктивності та якості були взяті завдання, за якими закріплюються учасники проєкту, дані завдань, тобто їх кількість, статус в проєкті та час витрачений на виконання цих завдань.

Алгоритм отримання числового показника продуктивності та якості роботи учасника проєкту можна розділити на кілька ключових кроків. По-перше, це збір

даних, спочатку необхідно зібрати всі завдання та дані стосовно цих завдань, які були призначені конкретному учаснику в рамках певного проєкту. Після отримання даних, потрібно перевірити дані та зробити розрахунок базових метрик. Після отримання списку завдань перевіряється, чи є призначені завдання і, чи правильно ідентифікований відповідальний користувач. Якщо завдань немає або користувача не знайдено, система повертає повідомлення про помилку. Якщо завдання знайдено, обчислюються такі базові метрики:

– загальна кількість завдань: кількість усіх завдань, призначених користувачеві;

– кількість виконаних завдань : кількість завдань, статус яких позначено як “виконано”;

– загальний оцінний час: сума всіх оцінних часових витрат на виконання завдань (використовуються так звані “story points”);

– загальний витрачений час: сума фактично витраченого часу на виконання завдань.

Далі, зібрані дані нормалізуються для зручності подальшого аналізу та порівняння:

– коефіцієнт завершення завдань: розраховується як відношення кількості виконаних завдань до загальної кількості завдань, помножене на відсоткову шкалу для подання у вигляді відсотків;

– Ефективність часу: розраховується як мінімальне значення між відсотковою шкалою та відношенням загального оцінного часу до загального витраченого часу, помножене на відсоткову шкалу.

Підсумковий показник продуктивності учасника проєкту розраховується шляхом зважування нормалізованих метрик, щоб визначити загальну ефективність учасника проєкту. Спочатку коефіцієнт завершення завдань і ефективність часу множаться на свої відповідні ваги, які визначають внесок кожної метрики в загальний результат. Потім зважені значення складаються, щоб отримати підсумковий показник продуктивності. У цьому разі використовується рівномірний

розподіл ваг, де кожна метрика має однакову значущість у підсумковому результаті. Таким чином, підсумкова продуктивність учасника являє собою середнє значення двох нормалізованих метрик, надаючи збалансовану оцінку продуктивності учасника.

3.5 Створення UI/UX або іншого дизайну системи

Ще одним важливим етапом побудови системи є створення мокапу системи. Необхідно розробити усі основні сторінки системи для коректної розробки та зручного застосування цих сторінок користувачем.

Спочатку потрібно розробити мокап вступної сторінки на яку буде заходити користувач при першому заході до системи. На рисунку 3.5 зображено мокап початкової сторінки системи.

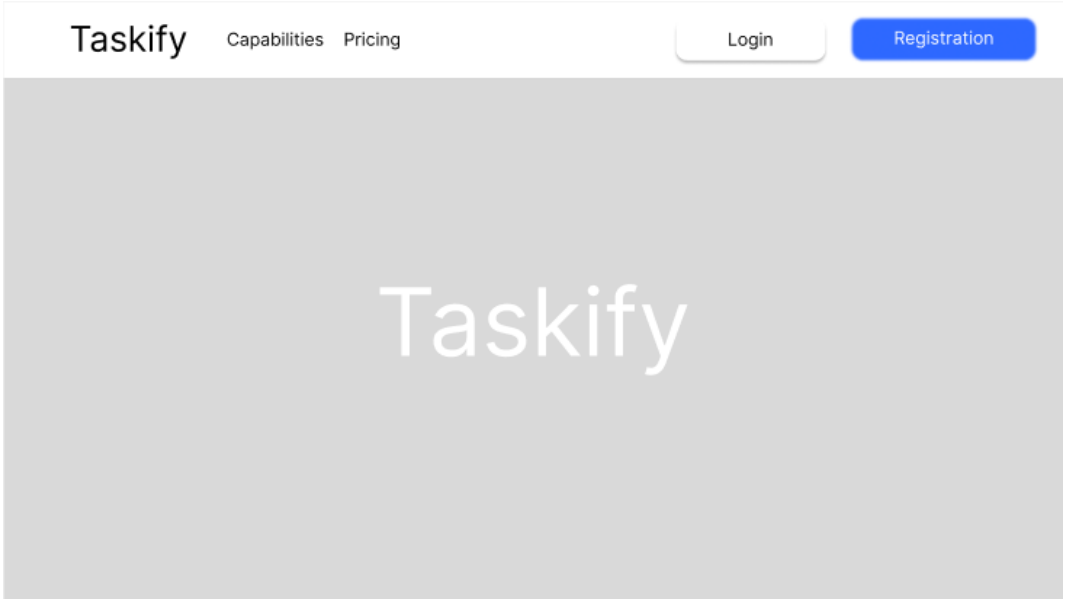


Рисунок 3.5 – Мокап початкової сторінки системи

На даній сторінці можливо побачити меню системи, де користувач може перейти на сторінку з можливостями системи натиснувши кнопку “Capabilities”, де він зможе знайти інформацію про систему, при натисканні на кнопку “Pricing” користувач зможе переглянути ціни на продукт, а також на цій сторінці користувач

зможе перейти до сторінки логіну та реєстрації в системі натиснувши відповідні кнопки “Login” та “Registration”. На самій сторінці, нижче меню, буде розташована рекламні картинки та описовий текст системи для того, щоб зацікавити користувача спробувати дану систему.

Після реєстрації або логіну в системі користувач потрапляє до основної сторінки системи, через яку він зможе взаємодіяти з іншими частинами системи. На рисунку 3.6 представлено мокап основної сторінки користувача.

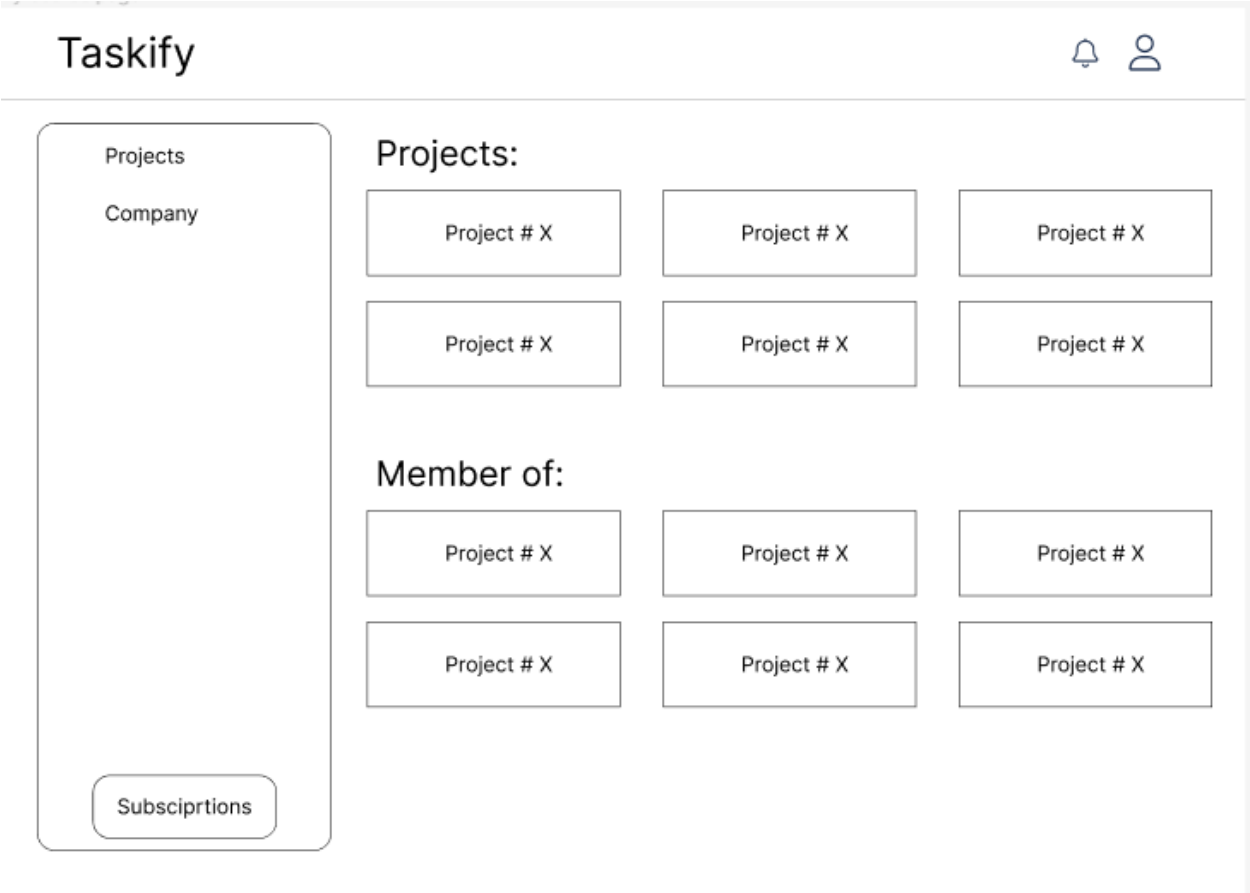


Рисунок 3.6 – Мокап основної сторінки користувача програмної системи

На цій сторінці користувач може переглядати свої проєкти та проєкти до яких він був доданий як учасник проєкту. Також в лівій частині буде представлено меню, за допомогою якого, користувач зможе перейти до таких компонентів як проєкти системи, компанії користувача та перейти до розділу підписок. У верхній частині цієї сторінки буде відображатися основне меню системи, де користувач зможе

швидко перейти до цієї сторінки, переглянути повідомлення системи та перейти до свого профілю.

Після переходу до сторінки окремого проєкту користувач зможе переглянути основну сторінку проєкту, де буде розташована дошка проєкту. На рисунку 3.7 зображено мокап основної сторінки проєкту.

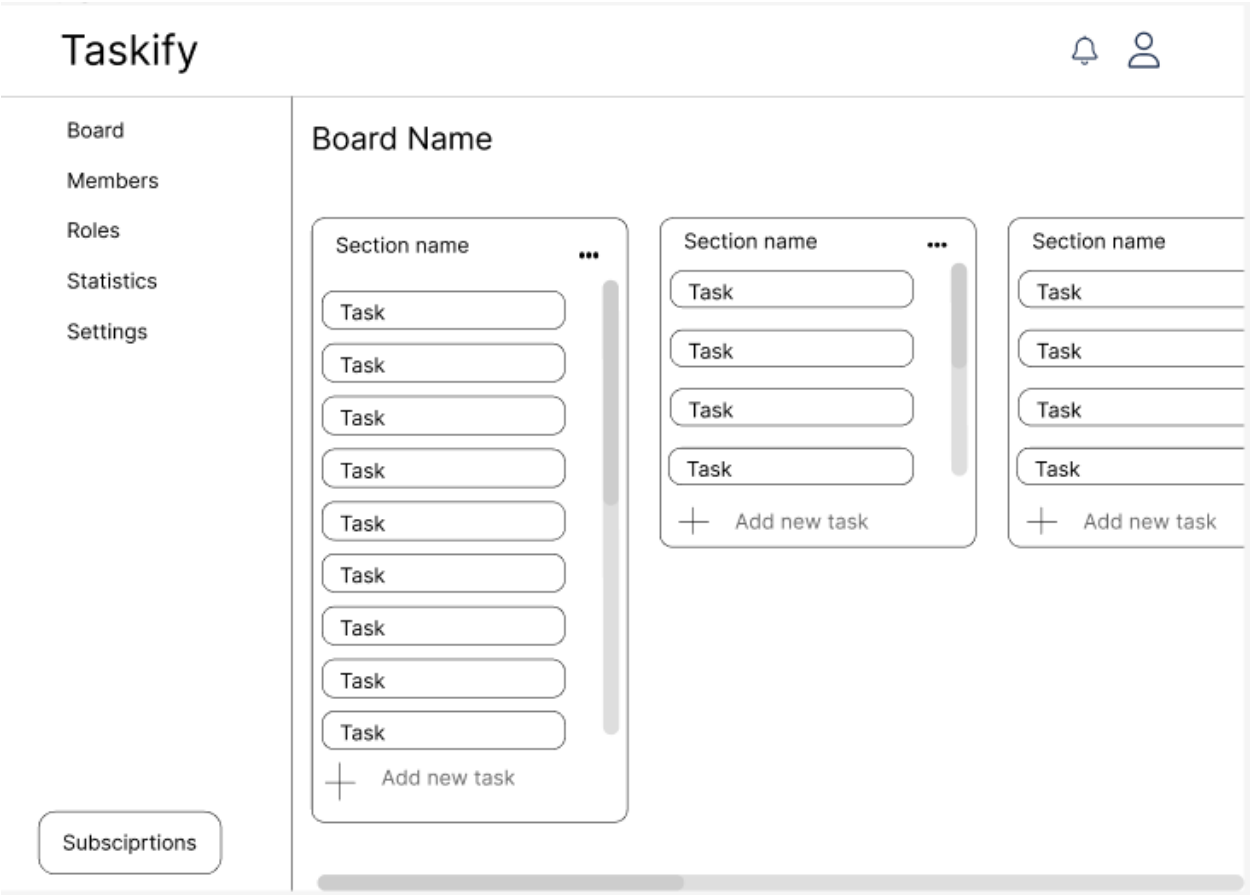


Рисунок 3.7 – Мокап основної сторінки проєкту

На цій сторінці в верхній частині відображається основне меню системи, яке було описано раніше, в лівій частині відображається меню окремого проєкту, за допомогою якого, користувач зможе перейти до основних пунктів проєкту таких як: переглянути дошку, переглянути учасників, переглянути ролі, переглянути статистику, продивитись налаштування та переглянути статус своєї підписки. В правій частині сторінки буде розташована інформація про дошку та дії для її зміни, а також дошка Kanban, яка буде допомагати користувачеві зручно взаємодіяти з

частинами проєкту. На цій дошці користувач зможе створювати, редагувати, видаляти секції та завдання, та переглядати інформацію завдань. Цією дошкою також можливо буде управляти за допомогою методу drag and drop. Метод drag and drop (перетягування та відпускання) – це інтерактивний підхід, за допомогою якого користувач переміщує елементи на екрані за допомогою миші або екранного пристрою. Він зазвичай використовується в онлайн-додатках, таких як дошки Kanban, для полегшення управління та організації інформації. За допомогою цього методу можливо буде переміщувати секції та завдання для покращення UX системи.

Далі будуть наводитись мокапи сторінок, які необхідні для повноцінного функціонування проєкту. Наступна сторінка для якої створювався мокап, це сторінка з інформації про користувачів та функціонал для взаємодії з учасниками проєкту. На рисунку 3.8 відображено мокап сторінки для взаємодії з учасниками проєкту.

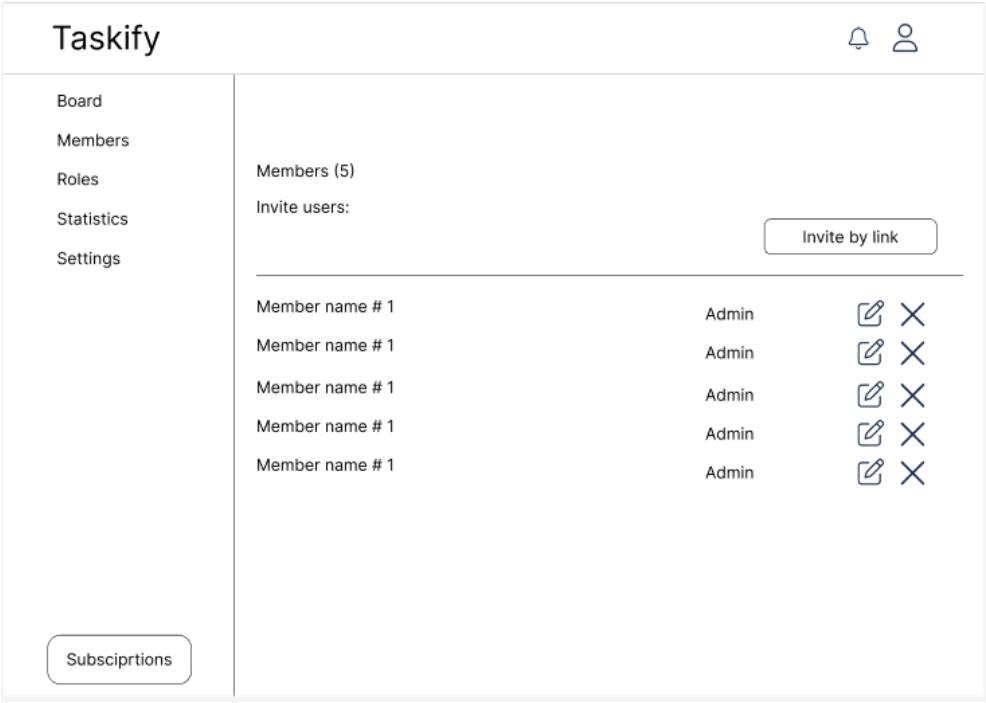


Рисунок 3.8 – Мокап сторінки для взаємодії з учасниками проєкту

На цій сторінці, окрім вже описаних верхнього меню та бокового меню, представлено функціонал для взаємодії з учасниками системи. На цій сторінці користувач, який створив проєкт зможе запросити до свого проєкту інших користувачів та переглянути усіх користувачів системи, а також він зможе відредагувати інформацію про учасників проєкту.

Мокап сторінки з ролями проєкту майже не відрізняється від мокапу сторінки для управління учасниками проєкту, тому було вирішено не розробляти мокап цієї сторінки, а при розробці цієї сторінки переглянути мокап сторінки з учасниками проєкту.

Наступна сторінка для якої необхідно розробити мокап, це сторінка зі статистиками проєкту. На цій сторінці користувач зможе переглянути усі статистики, які представляє системи стосовно окремого проєкту. На рисунку 3.9 представлено мокап сторінки зі статистиками проєкту.

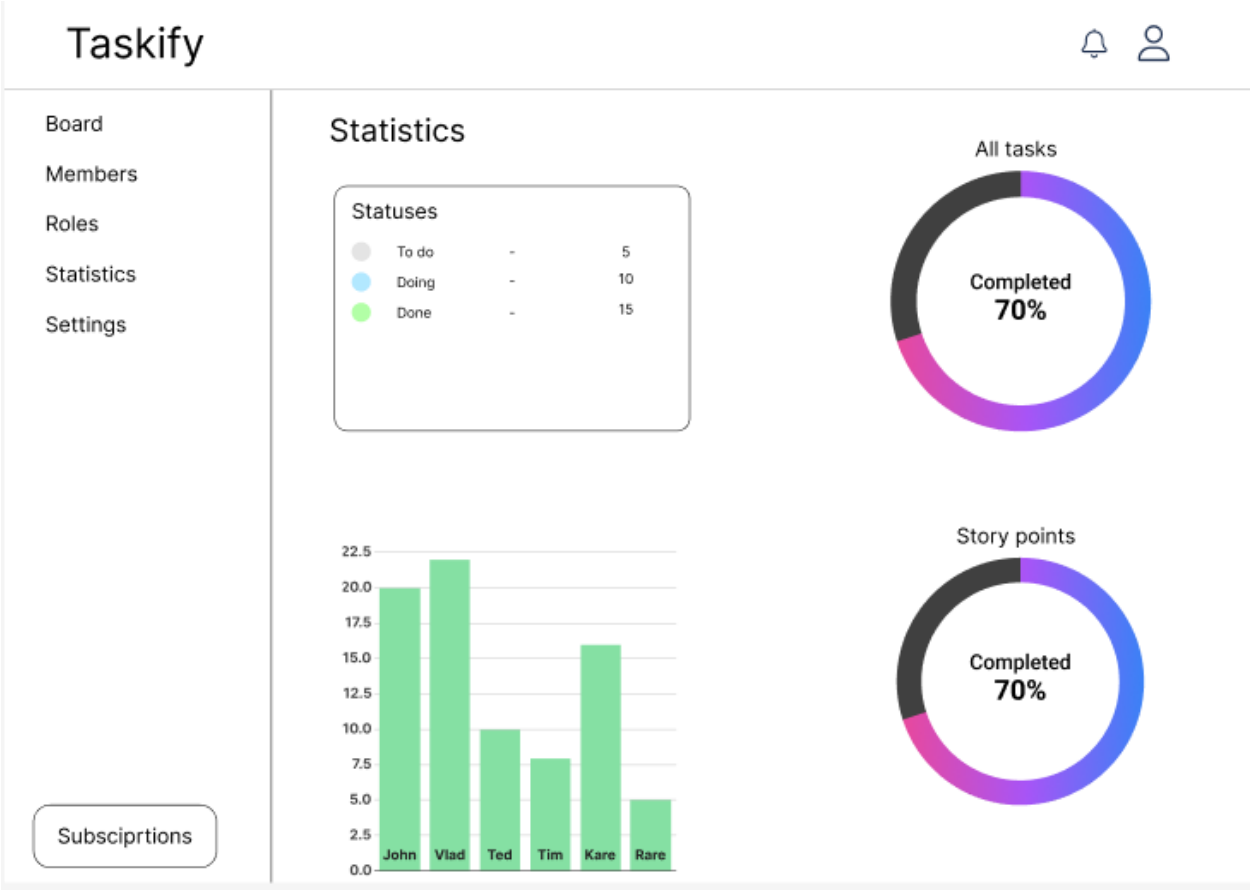


Рисунок 3.9 – Мокап сторінки зі статистиками проєкту

На цій сторінці будуть представлені усі основні статистики проєкту. Ці статистики будуть представлені, як в текстовому представлені, так і у вигляді діаграм.

Наступна сторінка це мокап головної сторінки компанії користувача. На рисунку 3.10 наведено мокап головної сторінки компанії користувача.

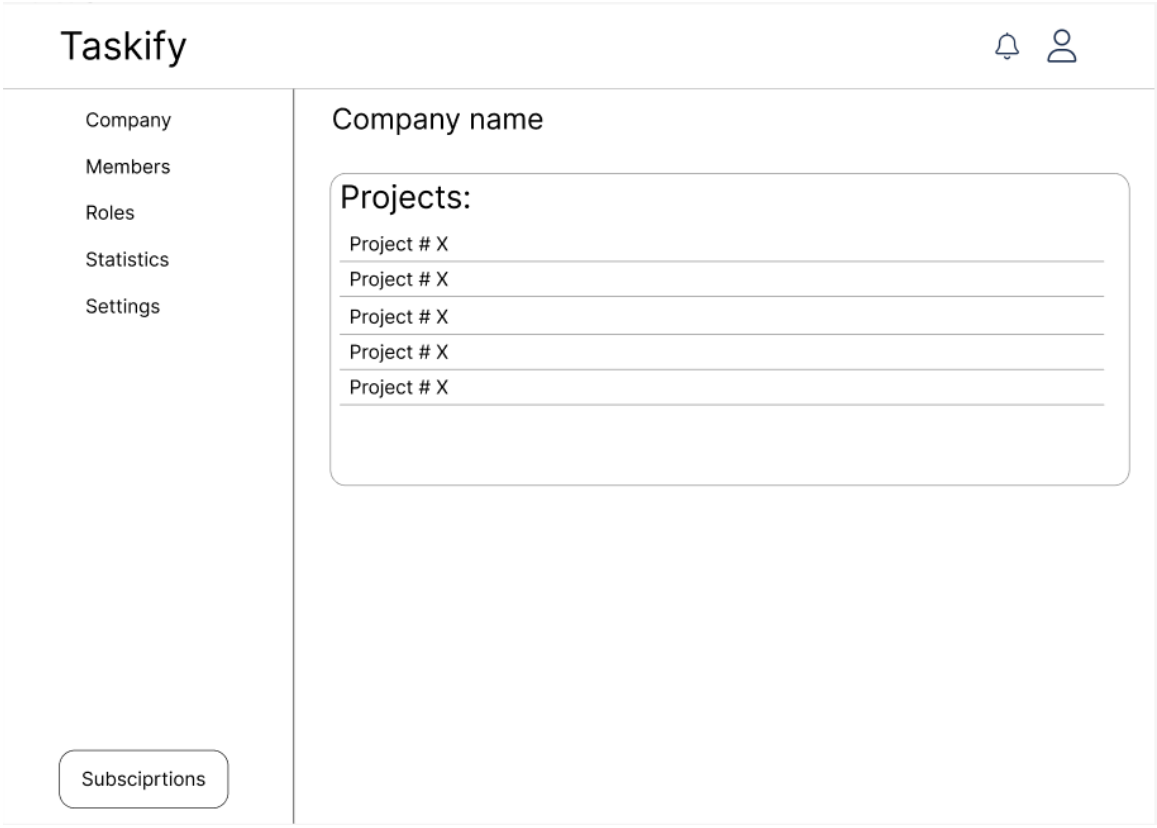


Рисунок 3.10 – Мокап головної сторінки компанії користувача

На цій сторінці можливо побачити та змінити назву компанії та продивитись, які є проєкти у компанії та перейти до редагування даних про ці проєкти.

Мокап сторінки з учасниками компанії, з ролями компанії та статистиками не відрізняється від мокапу відповідних сторінок в проєктах, які були описані раніше. Різниця можливо лише з тим, що учасники та ролі будуть входити до складу компанії, а на сторінці зі статистиками буде приділена увага більше статисткам пов’язаних з бюджетом компаній.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

Одним з важливих етапів побудови програмної системи є вибір прийнятих технологій для розробки програмної системи. Тому важливо описати, які саме технології були обрані для реалізації програмної системи та які переваги цих технологій.

Для розробки серверної частини програмної системи використовувалися: мова програмування C# [11], ASP.NET [12], Entity Framework [13].

C# – це об'єктно-орієнтована мова програмування, розроблена компанією Microsoft. Її було представлено у 2000 році як частину платформи .NET, і відтоді вона стала однією з найпопулярніших мов для розроблення широкого спектра застосунків. C# має наступні переваги:

– простота і читабельність: синтаксис C# нагадує інші відомі мови програмування, такі як C++ та Java, що робить його легким для вивчення та використання;

– об'єктно-орієнтована природа: C# підтримує основні принципи об'єктно-орієнтованого програмування, такі як успадкування, поліморфізм та інкапсуляція, що сприяє створенню гнучких і повторно використовуваних компонентів;

– безпека типів: C# забезпечує сувору типізацію, що допомагає уникнути багатьох типів помилок, таких як помилки приведення типів і некоректні операції з покажчиками;

– підтримка асинхронного програмування: C# має вбудовану підтримку асинхронних операцій, що дає змогу створювати продуктивні та гнучкі додатки, особливо важливі для веброзробки та роботи з великими обсягами даних;

– велика бібліотека класів: платформа .NET надає велику бібліотеку класів, яка містить безліч готових рішень для роботи з файлами, потоками, мережевими операціями, базами даних і багатьма іншими завданнями.

ASP.NET, серверна платформа для розробки вебдодатків. Він дозволяє створювати динамічні вебсторінки, онлайн-додатки та вебслужби. ASP.NET є частиною платформи .NET і пропонує високу продуктивність, надійність і безпеку. ASP.NET має кілька ключових особливостей, серед яких:

– підтримка MVC та Web API: ASP.NET підтримує архітектурні шаблони Model-View-Controller (MVC) і вебінтерфейси, що спрощує створення добре структурованих і підтримуваних додатків;

– швидкість і продуктивність: ASP.NET компілює код на сервері, що забезпечує високу швидкість і швидку обробку запитів;

– безпека: ASP.NET має вбудовані методи захисту від типових загроз, таких як SQL-ін'єкції, що підвищує безпеку онлайн-додатків;

– гнучкість і розширюваність: платформа легко інтегрується з різними бібліотеками та фреймворками, включаючи Angular, React і Vue.js, що робить її ідеальною для розробки сучасних односторінкових додатків (SPA);

– масштабованість: ASP.NET ідеально підходить для створення масштабованих додатків, які можуть обслуговувати велику кількість користувачів і запитів.

Entity Framework (EF) – це технологія ORM (Object-Relational Mapping), для платформи .NET. Він дозволяє розробникам взаємодіяти з базами даних за допомогою об'єктно-орієнтованих моделей, значно спрощуючи доступ до даних та адміністрування. Основні переваги Entity Framework включають:

– легка маніпуляція даними: Entity Framework дозволяє розробникам маніпулювати даними в базі даних за допомогою об'єктів .NET, усуваючи необхідність у складних SQL-запитах;

– автоматичне управління схемою бази даних: EF може генерувати та змінювати схеми баз даних на основі класів моделі даних, що полегшує розробку та оновлення додатків;

– підтримує LINQ (Language Integrated Query), що дозволяє створювати запити до бази даних на C#, використовуючи знайомий синтаксис;

– має потужні функції відстеження змін, які автоматично відстежують і синхронізують зміни, внесені в об'єкти, з базою даних, спрощуючи управління даними;

– міграції баз даних: Entity Framework дозволяє мігрувати бази даних, спрощуючи обробку змін у схемі бази даних протягом життя програми.

При використанні Entity Framework використовувався патерн Repository. Цей патерн забезпечує абстракцію над доступом до даних, відокремлюючи бізнес-логіку від рівня доступу до даних. Це сприяє створенню гнучкої та підтримуваної архітектури, спрощуючи процеси тестування та супроводу додатків.

В рамках цієї програмної системи для реалізації взаємодії з базою даних за допомогою Entity Framework було створено окремий Data Access Layer, який включає репозиторії. Репозиторії надають зручний інтерфейс для виконання CRUD-операцій з сутностями, а також для фільтрації даних. На рисунку 4.1 можливо побачити інтерфейс до загального репозиторію, від якого спадкувалися усі інші репозиторії.

```
public interface IDataRepository<T>
{
    Task<List<T>> GetAllAsync();

    Task<List<T>> GetFilteredItemsAsync(Expression<Func<T, bool>> filter);

    Task<T?> GetByIdAsync(string id);

    Task<T> AddAsync(T item);

    Task UpdateAsync(T item);

    Task DeleteAsync(string id);
}
```

Рисунок 4.1 – Інтерфейс загального репозиторію

Також для фільтрації сутностей та з’єднання різних типів сутностей та фільтрацію цих з’єднаних сутностей було створено абстрактний клас, який

реалізовували інші репозиторії, сутності яких потребували у використанні функціонала фільтрації та поєднання сутностей. На рисунку 4.2 можливо продивитися абстрактний клас для створення функціонала для фільтрації сутностей в програмній системі.

```
public abstract class BaseFilterableRepository<T, TFilterBuilder> : BaseRepository<T>, IFilterableRepository<T, TFilterBuilder>
where T : class
where TFilterBuilder : new()
{
    protected readonly TFilterBuilder _filterBuilder;

    public BaseFilterableRepository(DataContext dbContext) : base(dbContext)
    {
        _filterBuilder = new TFilterBuilder();
    }

    public virtual async Task<List<T>> GetFilteredItemsAsync(Action<TFilterBuilder> buildFilter)
    {
        buildFilter(_filterBuilder);

        var query = _dbContext.Set<T>().AsQueryable();
        query = IncludeEntities(query);

        var filterProperty = _filterBuilder.GetType().GetProperty("Filter");
        var filter = filterProperty.GetValue(_filterBuilder) as Expression<Func<T, bool>>;

        if (filter == null)
        {
            filter = _ => true;
            filterProperty.SetValue(_filterBuilder, filter);
        }

        return await query
            .Where(filter)
            .ToListAsync();
    }

    protected virtual IQueryable<T> IncludeEntities(IQueryable<T> query)
    {
        return query;
    }
}
```

Рисунок 4.2 – Абстрактний клас для фільтрації сутностей

Для зберігання даних у програмній системі було обрано MS SQL Server [14]. Цю реляційну систему управління базами даних (СКБД) створила компанія Microsoft. MS SQL Server – це популярна та надійна система управління базами даних (СУБД), яка часто використовується в бізнес-додатках і хмарних рішеннях. Основні переваги MS SQL Server:

– висока швидкість і масштабованість: MS SQL Server ефективно виконує запити та обробляє дані. Він забезпечує паралельне виконання запитів та оптимізацію операцій з великими обсягами даних. Можливості горизонтального та вертикального масштабування MS SQL Server дозволяють використовувати його як

для невеликих додатків, так і для великих корпоративних систем зі значним навантаженням;

– безпека даних: MS SQL Server містить широкі можливості безпеки, включаючи шифрування даних, контроль доступу на рівні рядків і динамічне маскування даних. Це захищає дані від незаконного доступу та витоку. Підтримка інтеграції з Active Directory та іншими системами управління ідентифікацією та доступом (IAM) дозволяє централізовано адмініструвати безпеку;

– висока доступність та відмовостійкість: MS SQL Server надає рішення високої доступності, такі як Always On Availability Groups, які забезпечують мінімальний час простою і максимальну доступність даних;

– інтеграція з іншими продуктами Microsoft: MS SQL Server легко взаємодіє з Azure, Power BI, Excel та Visual Studio, що полегшує створення складних рішень та аналіз даних. Інтеграція з платформою .NET дозволяє працювати з базою даних з додатків на C# або ASP.NET.

Для створення клієнтської частини програмної були обрані наступні технології: React [15], TypeScript [16], Tailwind CSS [17] та DaisyUI [18]. Ці технології були обрані через їх популярність, продуктивність, гнучкість та простоту використання, що дозволяє створювати сучасні, адаптивні та масштабовані вебдодатки.

React – це популярний фреймворк для створення користувацьких інтерфейсів. Він став однією з найпопулярніших бібліотек для розробки вебдодатків завдяки своїй ефективності та універсальності. Основні переваги React:

– компонентна архітектура: За допомогою React ви можете розділити інтерфейс користувача на окремі багаторазові компоненти, що полегшує розробку та управління програмою;

– використання віртуального DOM для покращення оновлення інтерфейсу, що перевершує стандартні методи оновлення DOM;

– односпрямований потік даних: односпрямований потік даних React полегшує розуміння та управління станом програми, зменшує кількість помилок та спрощує налагодження;

– широка екосистема: React пропонує велику екосистему бібліотек та інструментів, що спрощує створення складних додатків.

TypeScript – це розширення JavaScript, яке забезпечує статичну типізацію. Microsoft створила і підтримує його, і воно стало популярним серед розробників завдяки покращеній підтримці інструментів та безпеці коду. Основні переваги TypeScript:

– статична типізація: TypeScript дозволяє розробникам визначати типи даних, що допомагає виявляти проблеми на етапі збірки, а не під час виконання, а отже, підвищує надійність коду;

– покращена підтримка інструментів: використовуючи статичну типізацію та анотації типів, інструменти розробки, такі як Visual Studio Code, можуть покращити автозавершення, рефакторинг та перевірку типів, підвищуючи продуктивність розробників;

– це розширення є підмножиною JavaScript, тому можливо повторно використовувати наявний код JavaScript, поступово впроваджуючи TypeScript у свій проєкт.

Tailwind CSS – це утиліта CSS-фреймворк, що містить класи для проєктування адаптивних та гнучких користувацьких інтерфейсів. Він дозволяє швидко та ефективно конструювати стилізовані компоненти без необхідності визначати унікальні стилі. Основні переваги Tailwind CSS:

– утилітарний підхід: Tailwind CSS містить кілька готових класів, які можна використовувати для стилізації елементів, мінімізуючи потребу в спеціальних стилях CSS;

– гнучкість і налаштованість, що дозволяє розробникам встановлювати власні стилі та теми, спрощуючи створення виразного та послідовного користувацького досвіду;

– адаптивність і сприйнятливість: Tailwind CSS має інструменти для створення адаптивних та сприйнятливих дизайнів, що спрощує створення інтерфейсів для різних пристроїв і дисплеїв;

– менше CSS: завдяки використанню утилітарних класів і видаленню непотрібних стилів, кінцевий розмір CSS-файлів можна значно зменшити, що призведе до покращення ефективності завантаження сторінки.

На рисунку 4.3 наведено приклад використання Tailwind CSS.

```
return (
    <div className="flex flex-col items-center justify-center mt-5">
        <h2 className="text-3xl font-bold mb-6">Project roles</h2>
        <p className="text-lg text-balance mb-4">Here you can create new project roles for your project</p>
    </div>
);
```

Рисунок 4.3 – Приклад використання Tailwind CSS

DaisyUI – це CSS-плагін для Tailwind, який містить набір готових компонентів інтерфейсу користувача. Він побудований на основі утилітарних класів Tailwind CSS, що дозволяє легко розробляти візуально привабливі та корисні інтерфейси. DaisyUI має наступні ключові переваги:

– надає різні готові компоненти, такі як кнопки, картки, форми та модальні вікна, що допомагає прискорити процес розробки та підвищити узгодженість інтерфейсу;

– інтеграція з Tailwind CSS: оскільки DaisyUI побудований на Tailwind CSS, можливо використовувати всі функції та утиліти Tailwind, змішуючи їх з компонентами DaisyUI для створення власного дизайну;

– простота використання: DaisyUI має простий та зрозумілий синтаксис для використання компонентів, що полегшує їх включення в проєкт.

DaisyUI має доволі багато готових компонентів, які можливо швидко реалізувати у своїй програмній системі. Завдяки DaisyUI було створено декілька компонентів, які використовувалися на клієнтській частині системи. На рисунку 4.4 наведено приклад компонента в якому використовувався DaisyUI.

```
return (
    <>
        <button className={openButtonStyle} onClick={openModal}>
            {openButtonText}
        </button>
        <dialog id={id} className="modal" onClick={e => e.stopPropagation()}>
            <div className={`modal-box ${modalBoxStyle}`}>
                <form method="dialog">
                    <button className="btn btn-sm btn-circle btn-ghost absolute right-2 top-2" onClick={closeModal}>
                        ✕
                    </button>
                </form>
                {React.cloneElement(children as React.ReactElement<any>)}
            </div>
        </dialog>
    </>
);
```

Рисунок 4.4 – Приклад використання DaisyUI

На цьому рисунку можливо побачити, що DaisyUI додає додаткові теги, як це робить Tailwind CSS, але, окрім того, що він додає деяку логіку в ці теги. Наприклад, в цьому випадку, за допомогою тегів “modal”, “modal-box”, “dialog”, можливо створити компонент за допомогою якого можливо відкривати модальне вікно, яке буде відкриватися поверх усіх інших компонентів та має в собі кнопки відкриття та закриття модального вікна. За допомогою лише цих трьох тегів можливо створити доволі багато функціонала та без цього плагіну потрібно було б написати набагато більше коду для того ж самого результату.

Також важливо зазначити для побудови графічного представлення статистик в програній системі використовувалась бібліотека react-chartjs-2. Одним з основних плюсів використання бібліотеки react-chartjs-2 є її гнучкість і різноманіття доступних типів графіків. Вона підтримує різні види діаграм, як-от лінійні графіки, стовпчасті діаграми, кругові діаграми та багато інших, що дає змогу наочно й ефективно візуалізувати дані.

Ще однією значною перевагою react-chartjs-2 є її інтеграція з React. Бібліотека легко впроваджується в компоненти React, що спрощує процес створення інтерактивних і динамічних графіків. Це дає змогу розробникам швидко та зручно оновлювати графічні представлення даних у разі зміни стану застосунку, забезпечуючи тим самим актуальність і точність відображуваної інформації.

Як приклад використання цієї бібліотеки можливо отримати статистику об продуктивності та якості роботи учасників на проєкті. На рисунку 4.5 представлено

статистику, яка показує в числовому представленні продуктивність та якість роботи учасників одного з проєктів в системі.

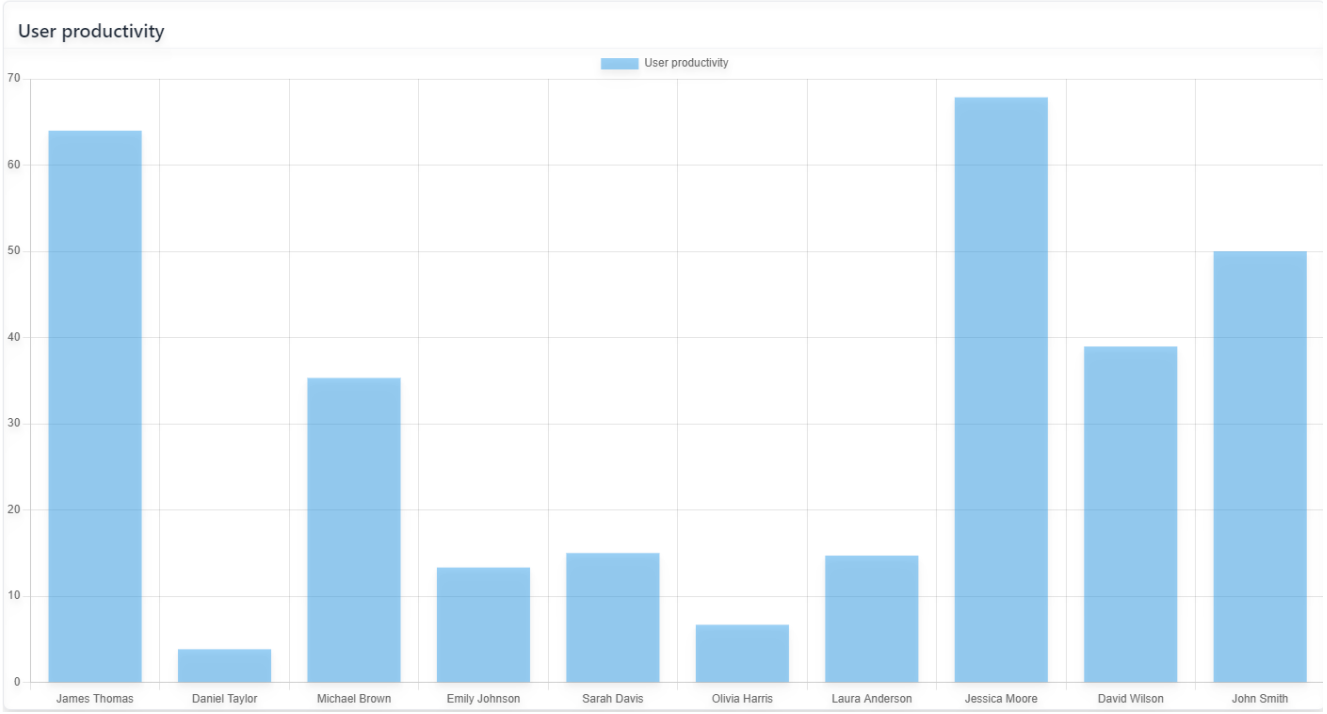


Рисунок 4.5 – Статистика продуктивності та якості роботи учасників проєкту

Висновком до цього розділу можливо зазначити, що технології використані для серверної та клієнтської частин, були обрані з урахуванням продуктивності, масштабованості та зручності використання. Використання C#, ASP.NET та Entity Framework на серверній стороні забезпечило стабільність та надійність системи, а React, TypeScript, Tailwind CSS та DaisyUI на клієнтській стороні дозволили розробити інтуїтивно зрозумілий та привабливий інтерфейс. Продемонстровані приклади коду та описи їх застосування підкреслюють доцільність та ефективність обраних технологій в контексті розробки програмних систем. В цілому, поєднання цих актуальних технологій дозволило створити високоякісний продукт, який відповідає сучасним стандартам розробки програмного забезпечення, а також відповідає очікуванням кінцевих користувачів.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування програмного забезпечення є невіддільною частиною розробки якісних програмних систем. Тестування забезпечує виявлення помилок та дефектів на ранніх стадіях розробки, що дозволяє знизити витрати на їх виправлення надалі. Крім того, тестування допомагає переконатися, що система відповідає заданим вимогам та очікуванням користувачів. Тестування також сприяє покращенню продуктивності, безпеки та зручності використання програмного забезпечення, що в кінцевому підсумку призводить до підвищення загальної задоволеності користувачів та конкурентоспроможності продукту.

Існує два основні види тестування програмного забезпечення: тестування методом чорної скриньки та тестування методом білої скриньки. Тестування методом чорної скриньки фокусується на перевірці функціональності системи без урахування її внутрішньої структури та коду. Цей метод тестування дозволяє оцінити систему з погляду кінцевого користувача, перевіряючи, чи вихідні дані системи відповідають очікуваним результатам на основі вхідних даних.

З іншого боку, тестування методом білої скриньки передбачає детальне вивчення внутрішньої логіки та структури програмного коду. Цей метод тестування вимагає глибокого розуміння архітектури системи та програмного коду, що дозволяє виявляти помилки лише на рівні реалізації.

На відміну від тестування методом чорної скриньки, що фокусується на зовнішніх проявах роботи системи, тестування методом білої скриньки спрямоване на аналіз внутрішньої роботи програми, що дозволяє виявити дефекти, які можуть бути непомітними при поверхневому тестуванні.

Таким чином, тестування методом чорної та білої скриньки взаємодоповнюють один одного, забезпечуючи всебічну перевірку якості програмного забезпечення. У контексті розробки системи для управління та контролю якості роботи співробітників IT-компаній, комбінування цих методів дозволяє не тільки переконатися в коректності та надійності роботи системи з

погляду користувача, а й забезпечити високу якість внутрішньої реалізації, що в кінцевому підсумку сприяє створенню більш надійного та ефективного продукту.

У процесі розробки програмної системи для управління та контролю якості роботи співробітників IT-компаній як методи чорної скриньки було використано функціональне тестування. Процес функціонального тестування включав себе перевірку усіх всіх пунктів інтерфейсу для перевірки коректності їхньої роботи. При виявленні будь-яких проблем вони негайно усувалися, що забезпечувало надійність і правильність роботи інтерфейсу користувача.

Для тестування методом білої скриньки було застосовано модульне тестування (Unit Testing). Основна частина модульного тестування була зосереджена на серверній частині системи, де тестувалися окремі компоненти та модулі для перевірки їхнього коректного функціонування. Unit Testing дозволяв виявляти та усувати помилки на ранньому етапі розробки, забезпечуючи високу якість внутрішньої реалізації системи. На рисунку 5.1 можливо побачити, основні модульні тести програмної системи, їх кількість та їх статус.

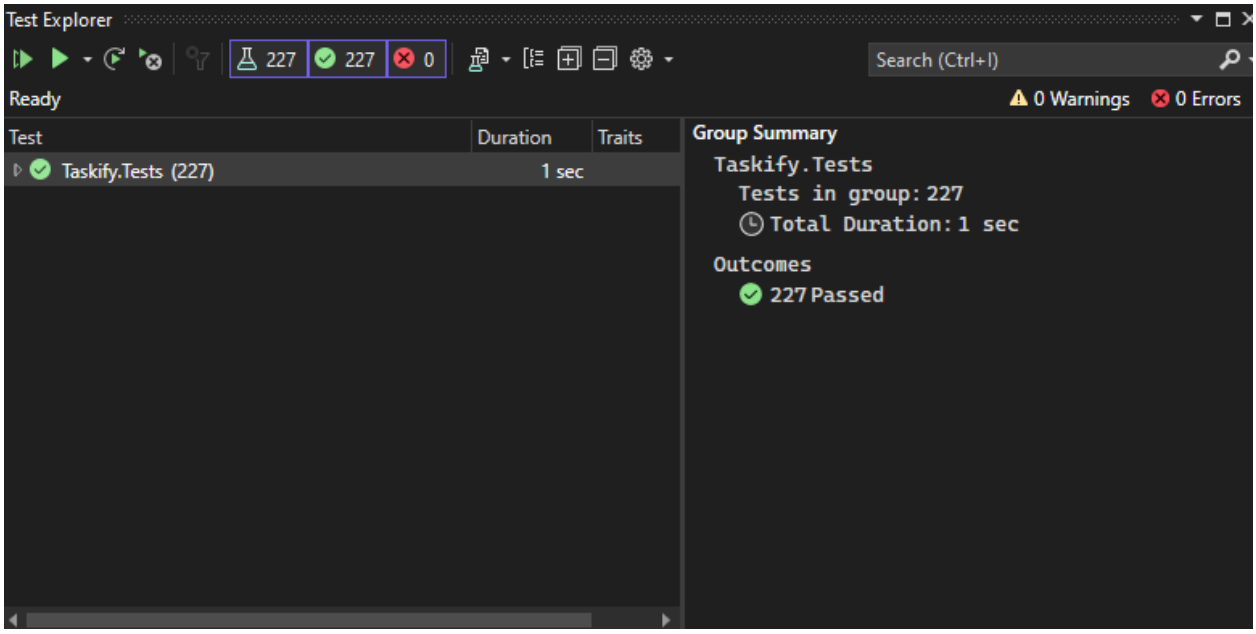


Рисунок 5.1 – Модульні тести програмної системи

Окрім зазначених вище методів, для оцінки навантаження системи було використано Apache JMeter. JMeter - популярний інструмент для тестування продуктивності та навантаження вебдодатків, який імітує велику кількість користувачів та аналізує поведінку системи під великим навантаженням. Під час тестування JMeter використовувався для завантаження системи сценаріями з великою кількістю користувачів. Це тестування класифікується як тестування "чорного ящика", оскільки воно досліджує продуктивність системи з точки зору кінцевого користувача, не впливаючи на внутрішню реалізацію.

Навантажувальне тестування передбачало реплікацію численних паралельних потоків, які представляли активність користувачів. Система тестувалася під навантаженням понад 20 000 запитів на хвилину. Згідно з результатами тестування, середній час реакції системи під високим навантаженням становить близько 250 мілісекунд. На рисунку 5.2 показано графік результатів тестування, на цьому графіку можливо побачити запити, кількість запитів на хвилину, середній час відклику та кількість запитів.

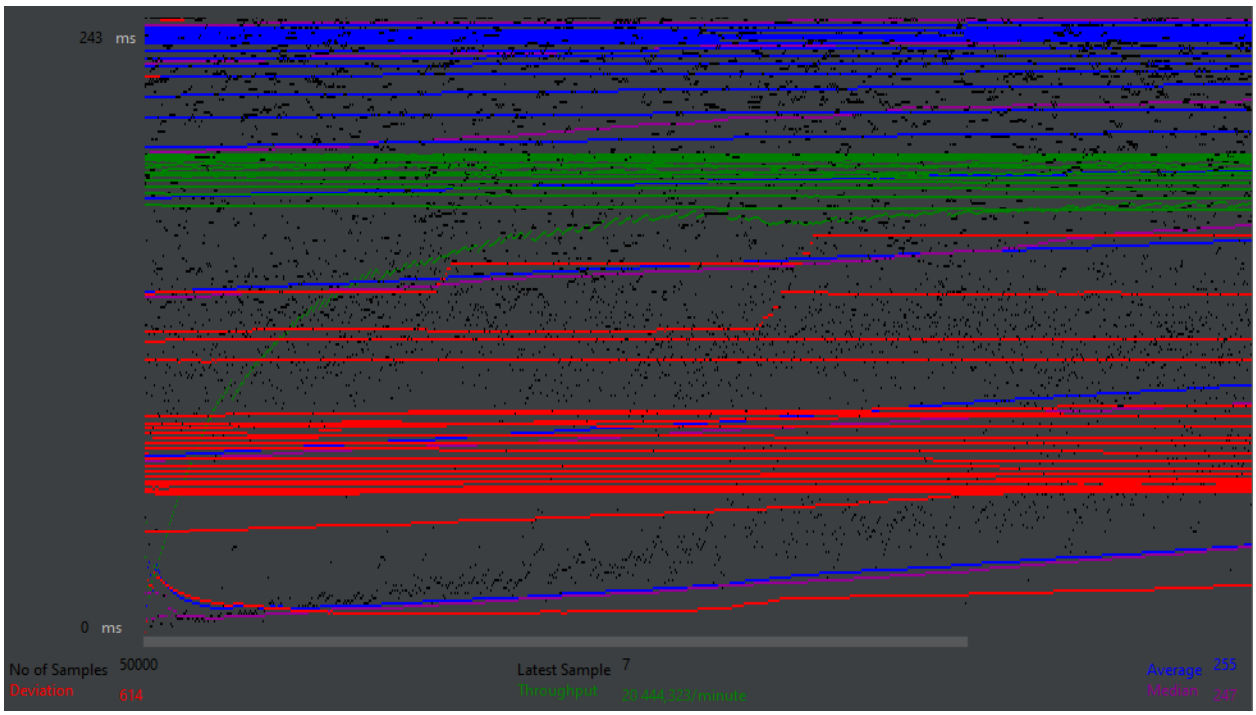


Рисунок 5.2 – Графік тестування навантаження системи за допомогою JMeter

Також на рисунку 5.3 можливо побачити кінцеві результати тестування з численними даними. На даному рисунку зображено загальну кількість запитів, середній час очікування відповіді на запит, мінімальний час очікування відповіді на запит, максимальний час очікування відповіді на запит, відсоток помилок, кількість запитів на секунду та кількість переданих даних.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
HTTP Request	50000	255	0	24436	614,89	0,00%	340,7/sec	499,20	306,66	1500,2
TOTAL	50000	255	0	24436	614,89	0,00%	340,7/sec	499,20	306,66	1500,2

Рисунок 5.3 – Статистичні дані навантаження системи

Таким чином, комплексний підхід до тестування, який містить ряд підходів "чорного ящика" і "білого ящика", забезпечив відмінну якість програмної системи. Використання цих процедур у різних комбінаціях дозволило виявити та усунути найрізноманітніші помилки та недоліки на різних етапах розробки, що значно підвищило надійність та стабільність кінцевого продукту.

Комплексний підхід до тестування, який включав низку методологій і тактик, дозволив нам розробити надійну та ефективну систему, яка задовольняє вимоги кінцевих користувачів, а також забезпечує високу якість продукту та його конкурентоспроможність на ринку.

6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Впровадження програмного забезпечення – це критичний етап процесу розробки, який передбачає встановлення, налаштування та інтеграцію програмної системи. Щоб гарантувати, що система добре функціонує і відповідає вимогам користувачів, ця процедура повинна бути ретельно спланована. Далі будуть детально описано процес розгортання програмного забезпечення, включаючи основні етапи та дії, необхідні для його ефективного виконання.

Аналіз та планування:

- виявлення потреб користувачів;
- збір зворотного зв'язку від користувачів;
- інтерв'ю та опитування цільової аудиторії;
- визначення основних завдань і проблем, які повинна вирішувати система;
- аналіз функціональних і нефункціональних вимог до системи;
- визначення необхідних функцій та можливостей системи;
- оцінити вимоги до продуктивності, безпеки та масштабованості;
- узгодження вимог з ключовими зацікавленими сторонами;
- постановка цілей;
- сформувати конкретні, вимірювальні та досяжні цілі для реалізації;
- визначення критеріїв успіху для кожної цілі;
- визначення часових рамок та ресурсів для досягнення цілей;
- розробка плану впровадження;
- визначення етапів та завдань для кожного етапу впровадження;
- розподіліть відповідальність за виконання завдань;
- визначення часових рамок і дедлайнів для виконання завдань;
- оцінка ризиків та розробка плану управління ризиками;
- визначення потенційних ризиків та проблем;
- оцінити ймовірність та вплив ризиків;

– розробка стратегій для мінімізації та усунення ризиків.

Підготовка інфраструктури:

– оцінка та підготовка апаратного забезпечення;
– оцінка вимог до серверного обладнання;
– придбання та налаштування необхідного обладнання;
– забезпечення резервного копіювання та відновлення даних;
– налаштування програмного забезпечення;
– встановлення необхідних операційних систем та баз даних;
– налаштування серверів додатків та вебсерверів;
– забезпечення безпеки та захисту даних;
– налаштування систем контролю версій;
– забезпечення середовища розробки та тестування;
– автоматизація процесів збірки та розгортання;
– забезпечення безперервного моніторингу та підтримки;
– налаштовуємо системи моніторингу продуктивності.

Навчання та підтримка користувачів:

– створення посібників та інструкцій для користувачів;
– розробка навчальних відео та презентацій;
– підготовка поширених запитань (FAQ) та відповідей на них;
– надання технічної підтримки;
– організація служби підтримки;
– забезпечення зворотного зв'язку та консультацій;
– внесення змін на основі зворотного зв'язку від користувачів;
– забезпечення доступу до актуальної інформації.

Інтеграція та тестування:

– інтеграція з наявними системами;
– аналіз наявних систем та їхньої сумісності;
– розробка інтерфейсів для інтеграції;

– тестування інтеграції та усунення виявлених проблем;
– тестування системи на всіх рівнях;
– проведення модульного тестування;
– виконання інтеграційного та системного тестування;
– проведення навантажувального та стрес-тестування;
– забезпечення цілісності та узгодженості даних.

Розгортання та експлуатація:

– розгортання системи;
– налаштування систем безпеки та доступу;
– налаштування систем моніторингу та оповіщення;
– розробка процесів управління змінами;
– підтримка та обслуговування.

Впровадження програмного забезпечення вимагає комплексного та системного підходу, який включає аналіз потреб користувачів, підготовку інфраструктури, навчання та підтримку користувачів, інтеграцію та тестування, а також розгортання та експлуатацію. Дотримуючись цих етапів і враховуючи всі компоненти процесу, можливо забезпечити ефективне розгортання системи, яка відповідає вимогам і очікуванням користувачів. Цей метод знижує ризики, забезпечує якість та надійність системи, а також підвищує задоволеність та ефективність роботи користувачів.

ВИСНОВКИ

В результаті виконання цієї роботи була створена “Програмна система для управління та контролю якості праці співробітників ІТ-компаній” за допомогою об’єктно-орієнтованої мови програмування C#, фреймворку ASP.NET Core Web API, сервера бази даних MS SQL Server, вебдодаток розроблено з допомогою технології React та мови програмування TypeScript.

Результати роботи були представлені під час XXVIII Міжнародного молодіжного форуму “Радіоелектроніка та молодь у XXI столітті”.

В результаті розробки був створений інструмент для управління та контролю якості праці ІТ-компаній та звичайних користувачів, який дозволить оптимізувати процеси управління проєктами, підвищить ефективність роботи команди та забезпечити прозорість розподілу ресурсів.

Вбудована система програмного забезпечення дозволятиме легко відстежувати робочі процеси проєктів, контролювати виконання робіт і оцінювати продуктивність.

Результати цієї роботи можуть допомогти ІТ-організаціям підвищити ефективність своєї діяльності, оптимізувати розподіл ресурсів та покращити співпрацю персоналу та керівництва. Крім того, підхід сприяє формуванню прозорого та справедливого робочого середовища, що підвищує ентузіазм та продуктивність працівників.

Таким чином, створена програмна система є важливим інструментом для оптимізації управління та контролю якості в ІТ-організаціях, що призводить до покращення бізнес-операцій та підвищення конкурентоспроможності на ринку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Melnyk, M., Korcelli-Olejniczak, E., Chorna, N., Popadynets, N. Development of Regional IT clusters in Ukraine: institutional and investment dimensions. Economic Annals-XXI, 2018. – P. 19-25.

2. Кузьміних В. О., Коваль О. В., Тараненко Р. А. Моделі та засоби управління ІТ- проєктами: навч.-метод. посіб. КПІ ім. Ігоря Сікорського, 2023. – 222 с.

3. Trello documentation. [Електронний ресурс] – URL: https://trello.com/en/guide (дата звернення: 01.05.2024)

4. Asana documentation. [Електронний ресурс] – URL: https://developers.asana.com/ (дата звернення: 01.05.2024)

5. Golmgrein, I. A Comprehensive Overview of Monetization Strategies in Creative Industries. International Journal of Latest Engineering and Management Research (IJLEMR), 2023. – P. 90-100.

6. What is Subscription Marketing: Guide. [Електронний ресурс] – URL: https://sendpulse.com/support/glossary/subscription-marketing (дата звернення: 01.05.2024)

7. Corona, E., Pani, F. E. A review of lean-kanban approaches in the software development. WSEAS transactions on information science and applications, 2013. – P. 1-13.

8. Damij, N., Damij, T. An approach to optimizing Kanban board workflow and shortening the project management plan. IEEE Transactions on Engineering Management, 2021. – P. 1-8.

9. What is UML diagrams? [Електронний ресурс] – URL: https://miro.com/diagramming/what-is-a-uml-diagram/ (дата звернення: 01.05.2024)

10. What is an Entity Relationship Diagram (ERD)? [Електронний ресурс] – URL: https://www.lucidchart.com/pages/er-diagrams (дата звернення: 01.05.2024)

11. C# language documentation [Електронний ресурс] – URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 01.05.2024)

12. ASP.NET documentation [Електронний ресурс] – URL: https://dotnet.microsoft.com/en-us/apps/aspnet (дата звернення: 01.05.2024)

13. Entity Framework documentation hub [Електронний ресурс] – URL: https://learn.microsoft.com/en-us/ef/ (дата звернення: 01.05.2024)

14. SQL Server technical documentation [Електронний ресурс] – URL: https://learn.microsoft.com/en-us/sql/sql-server/ (дата звернення: 01.05.2024)

15. React documentation [Електронний ресурс] – URL: https://react.dev/ (дата звернення: 01.05.2024)

16. Typescript documentation [Електронний ресурс] – URL: https://www.typescriptlang.org/docs/ (дата звернення: 01.05.2024)

17. Tailwind CSS documentation [Електронний ресурс] – URL: https://tailwindcss.com/docs/ (дата звернення: 01.05.2024)

18. DaisyUI documentation [Електронний ресурс] – URL: https://daisyui.com/docs/ (дата звернення: 01.05.2024)

ДОДАТОК А

Лістинг програмного коду

Наданий програмний код виконує базові функції програми, створення, редагування, видалення та отримання запитів щодо проєктів в системі.

Використовується API для зв’язку з вебсайтом.

```
using AutoMapper;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Taskify.BLL.Interfaces;
using Taskify.Core.DbModels;
using Taskify.Core.Dtos;

namespace Taskify.API.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [Authorize]
    public class ProjectsController : ControllerBase
    {
        private readonly IProjectsService _projectsService;
        private readonly IMapper _mapper;
        private readonly ILogger<ProjectsController> _logger;

        public ProjectsController(IProjectsService projectsService, IMapper
mapper, ILogger<ProjectsController> logger)
        {
            _projectsService = projectsService;
            _mapper = mapper;
            _logger = logger;
        }

        [HttpGet("{id}")]
        public async Task<IActionResult> GetProjectById(string id)
        {
            try
            {
                var result = await
_projectsService.GetProjectByIdAsync(id);

                return result.IsSuccess
                    ? Ok(_mapper.Map<ProjectDto>(result.Data))
                    : NotFound();
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "An error occurred in GetProjectById
method.");
                return
StatusCode(StatusCodes.Status500InternalServerError);
```

```
            }
        }

        [HttpPost]
        public async Task<IActionResult> CreateProject([FromBody]
CreateProjectDto projectDto)
        {
            try
            {
                var project = _mapper.Map<Project>(projectDto);
                var result = await
_projectsService.CreateProjectAsync(project);

                return result.IsSuccess
                    ? CreatedAtAction(nameof(GetProjectById), new { id =
result.Data.Id }, _mapper.Map<ProjectDto>(result.Data))
                    : BadRequest(result.Errors);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "An error occurred in CreateProject
method.");
                return
StatusCode(StatusCodes.Status500InternalServerError);
            }
        }

        [HttpPut("{id}")]
        public async Task<IActionResult> UpdateProject(string id,
[FromBody] UpdateProjectDto projectDto)
        {
            try
            {
                // Ensure the id in the path matches the id in the request
body
                if (id != projectDto.Id)
                {
                    return BadRequest("The provided id in the path does not
match the id in the request body.");
                }

                var project = _mapper.Map<Project>(projectDto);
                var result = await
_projectsService.UpdateProjectAsync(project);

                return result.IsSuccess
                    ? Ok(result.Data)
                    : BadRequest(result.Errors);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "An error occurred in UpdateProject
method.");
                return
StatusCode(StatusCodes.Status500InternalServerError);
            }
```

```
        }

        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteProject(string id)
        {
            try
            {
                var result = await _projectsService.DeleteProjectAsync(id);

                return result.IsSuccess
                    ? NoContent()
                    : BadRequest(result.Errors);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "An error occurred in DeleteProject
method.");
                return
StatusCode(StatusCodes.Status500InternalServerError);
            }
        }

        [HttpGet("user/{userId}")]
        public async Task<IActionResult> GetProjectsByUserId(string userId)
        {
            try
            {
                var result = await
_projectsService.GetProjectsByUserIdAsync(userId);

                return result.IsSuccess
                    ? Ok(_mapper.Map<List<ProjectDto>>(result.Data))
                    : BadRequest(result.Errors);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "An error occurred in
GetProjectsByUserId method.");
                return
StatusCode(StatusCodes.Status500InternalServerError);
            }
        }
    }
}
```

ДОДАТОК Б

Слайди презентації

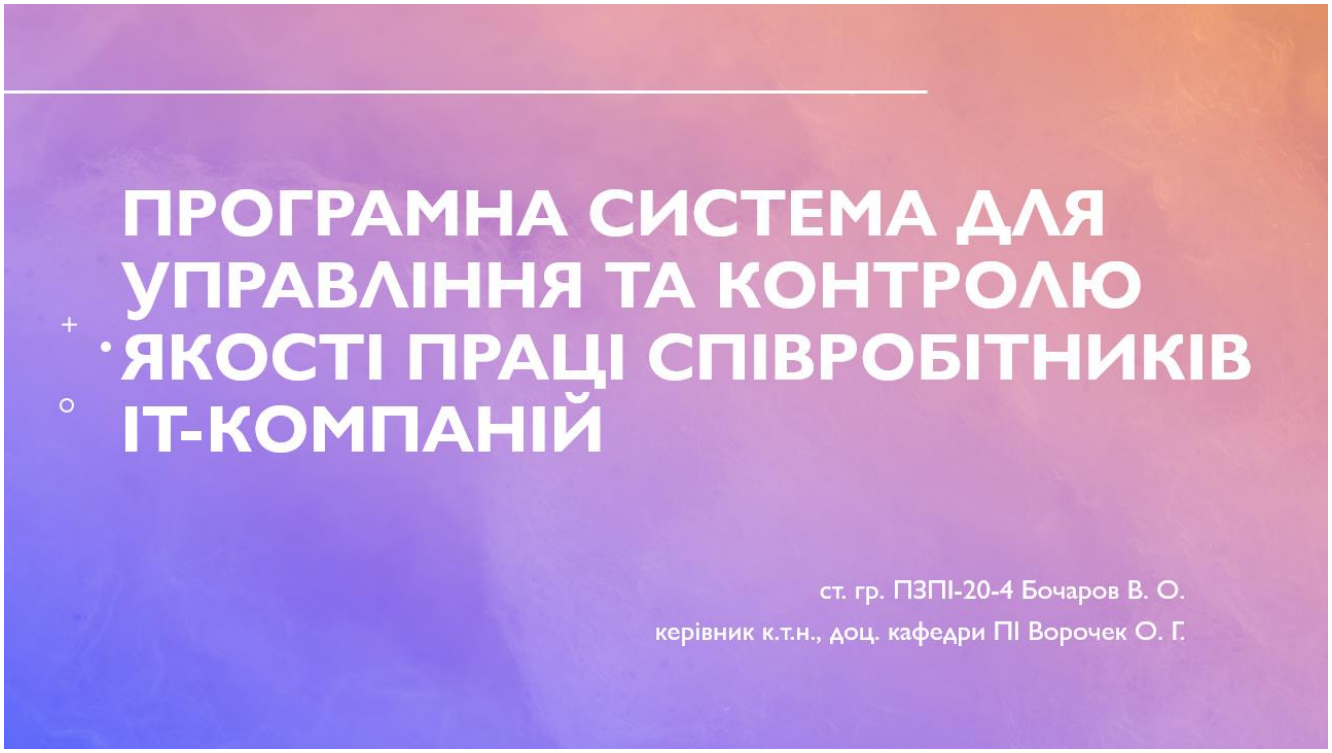


Рисунок Б.1 – Перший слайд презентації

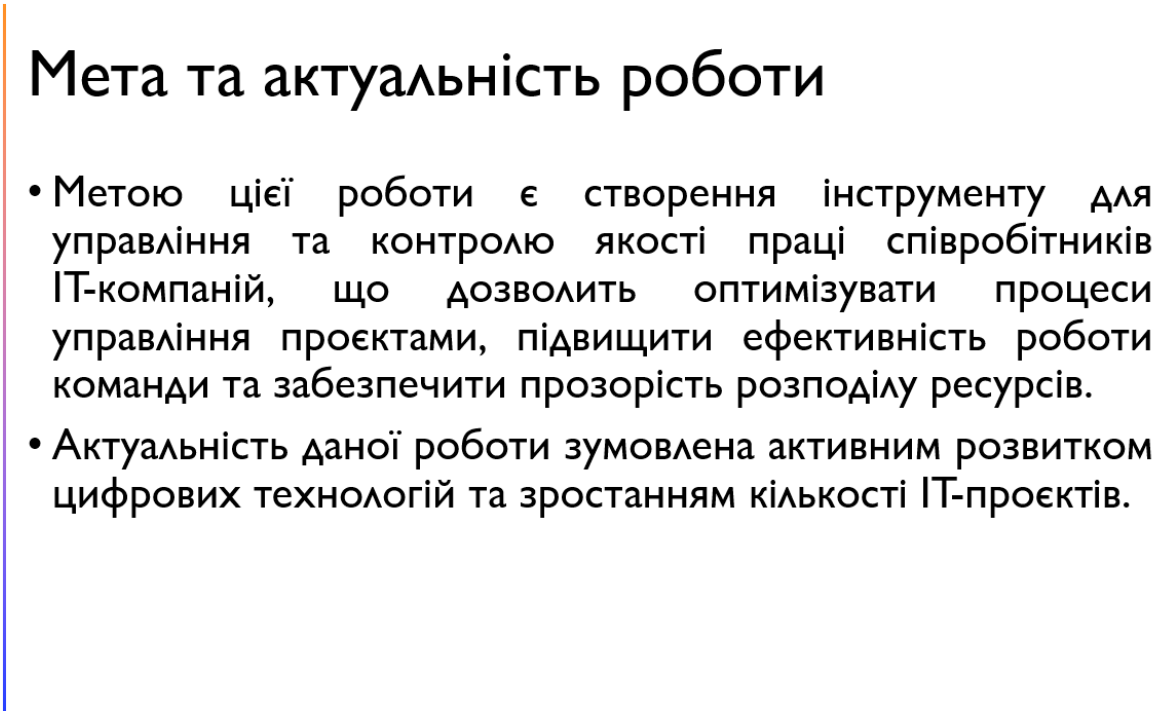


Рисунок Б.2 – Другий слайд презентації

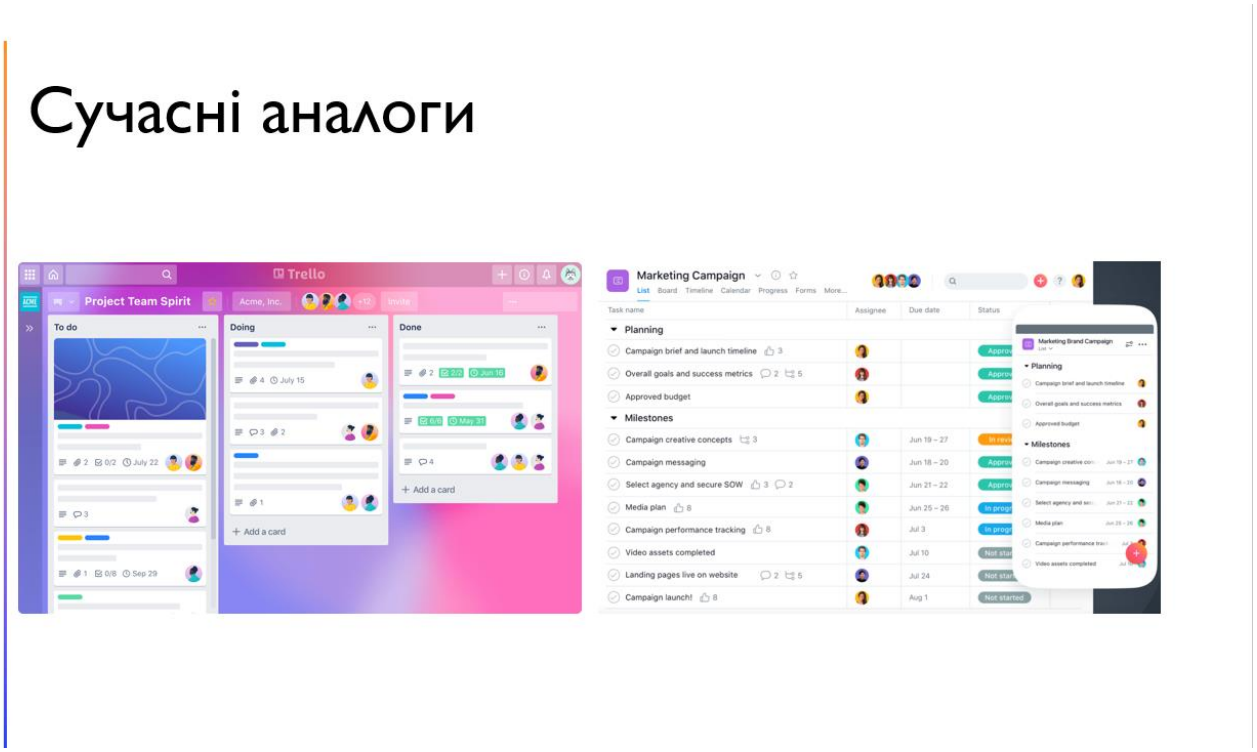


Рисунок Б.3 –Третій слайд презентації

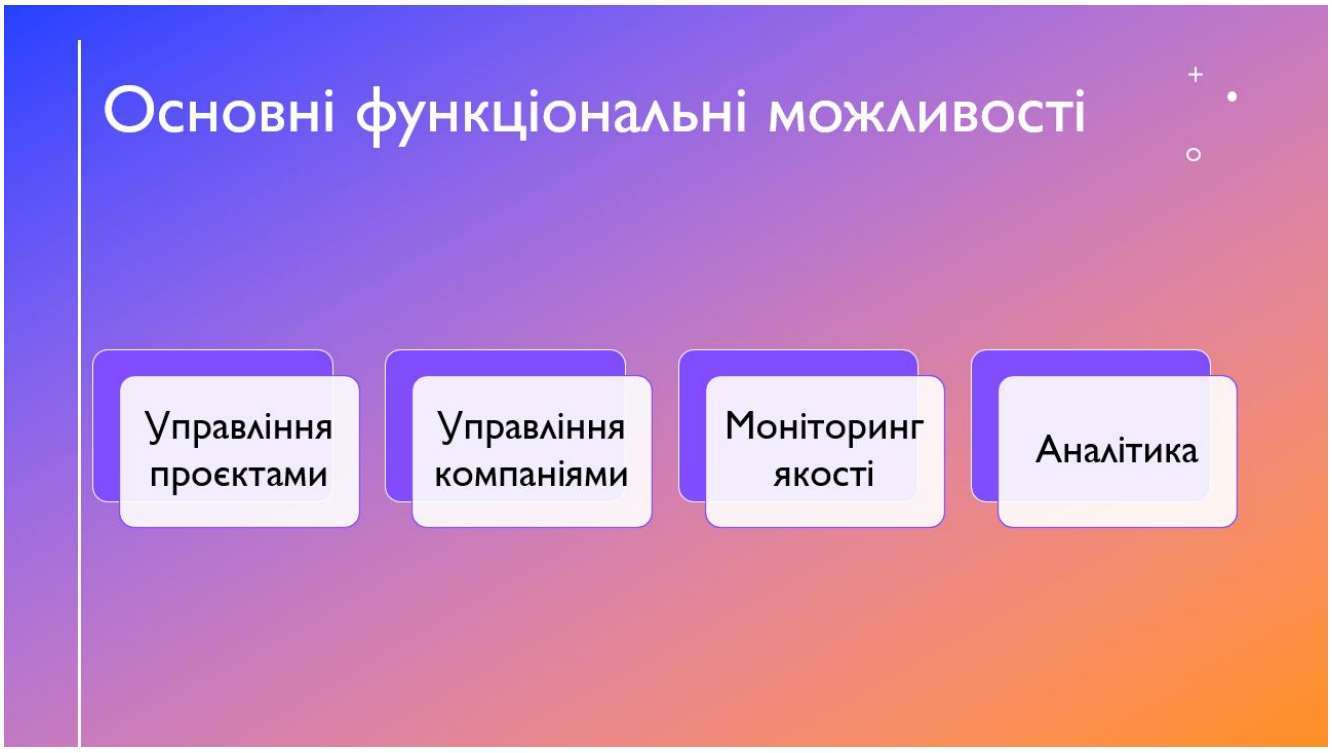


Рисунок Б.4 – Четвертий слайд презентації

UML діаграма прецедентів

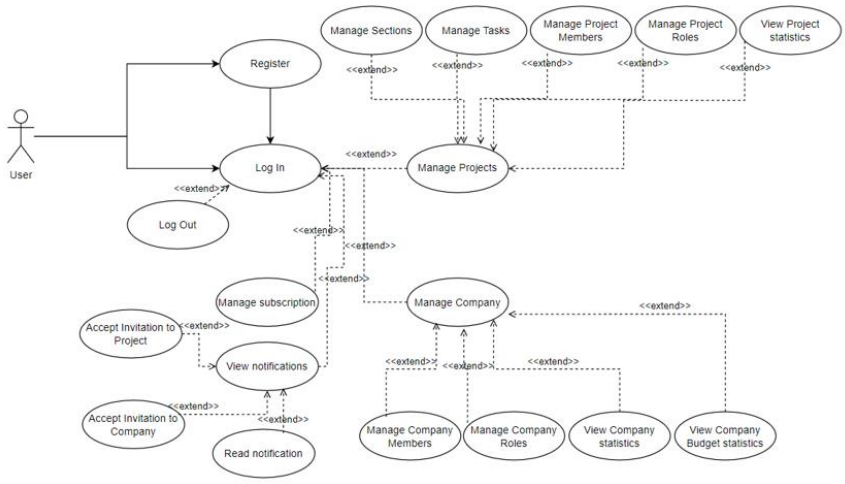


Рисунок Б.5 – П’ятий слайд презентації

UML діаграма класів

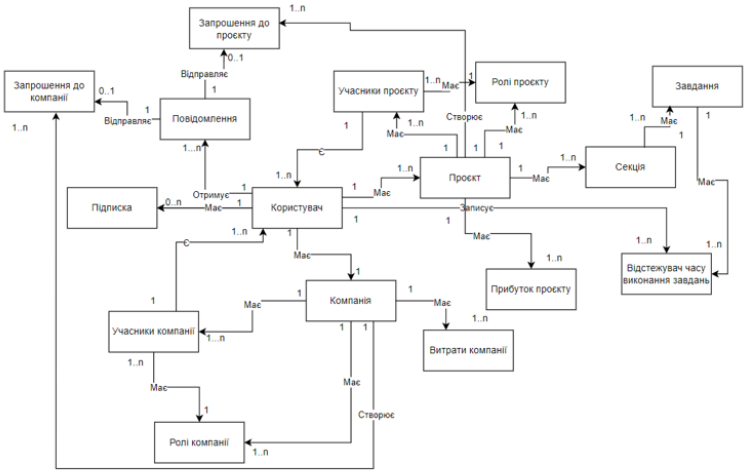


Рисунок Б.6 – Шостий слайд презентації

UML діаграма розгортання

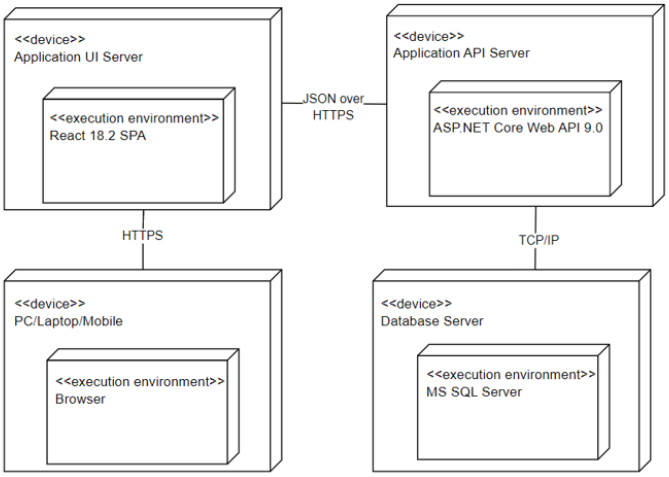


Рисунок Б.7 – Сьомий слайд презентації

Обрані технології

Рисунок Б.8 – Восьмий слайд презентації

Результати

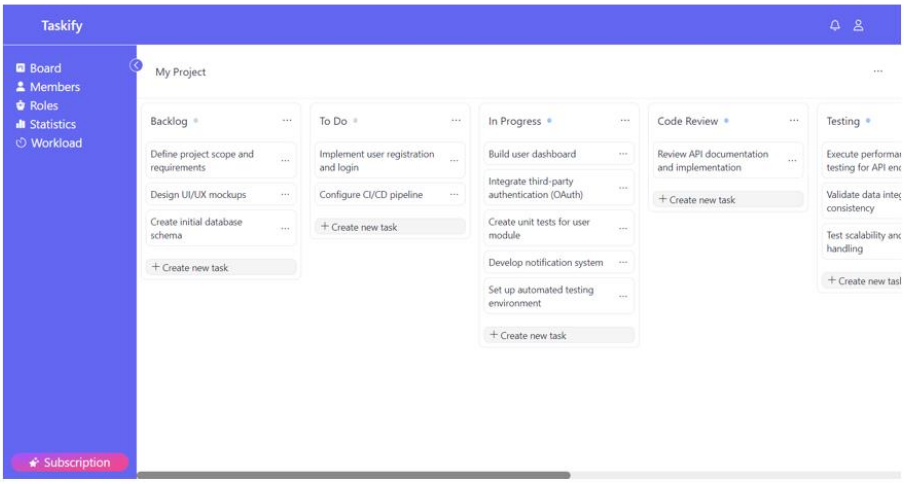


Рисунок Б.9 – Дев’ятий слайд презентації

Результати

Рисунок Б.10 – Десятий слайд презентації

Результати

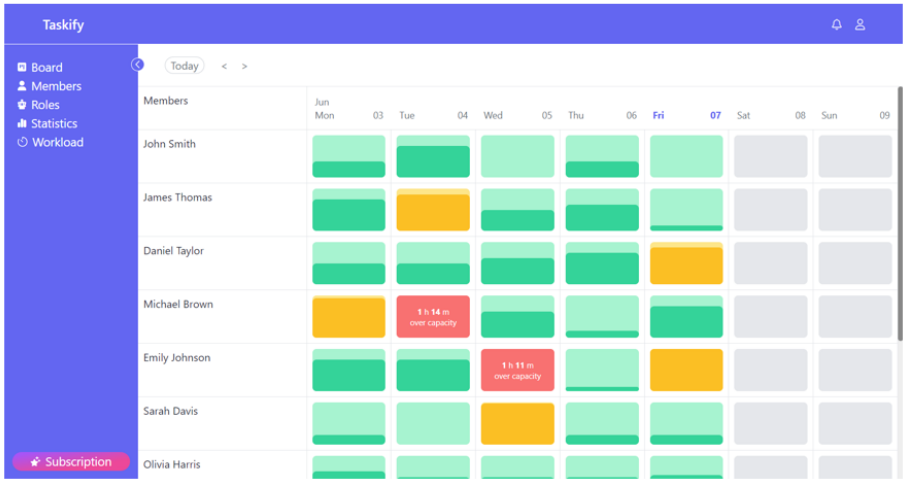


Рисунок Б.11 – Одинадцятий слайд презентації

Тестування. Unit Testing

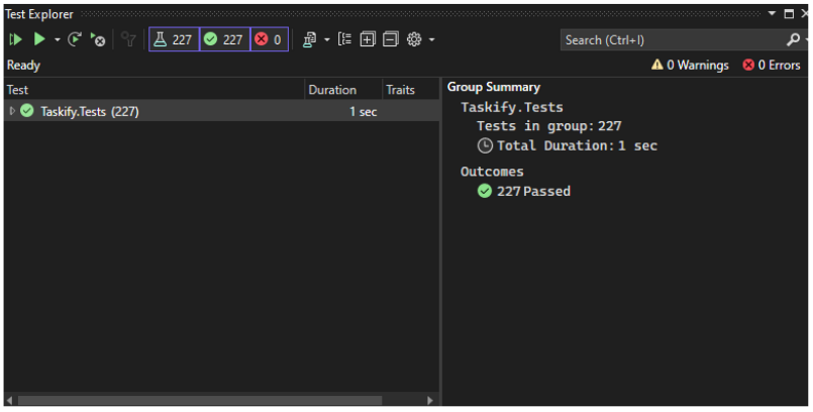


Рисунок Б.12 – Дванадцятий слайд презентації

Навантажувальне тестування

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
HTTP Request	50000	255	0	24436	614,89	0,00%	340,7/sec	499,20	306,66	1500,2
TOTAL	50000	255	0	24436	614,89	0,00%	340,7/sec	499,20	306,66	1500,2

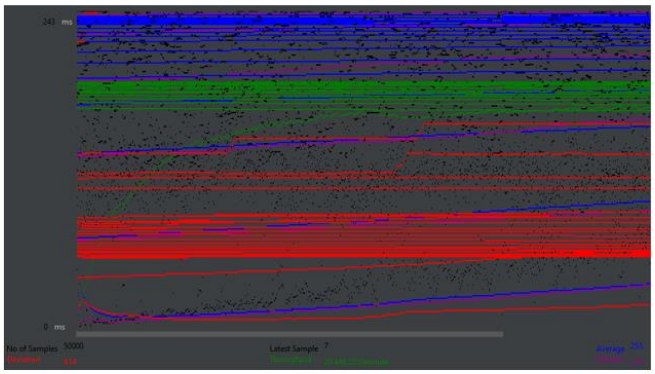


Рисунок Б.13 – Тринадцятий слайд презентації

Висновки

- В результаті виконання цієї роботи була створена “Програмна система управління та контролю якості праці співробітників IT-компаній”
- Результати роботи були представлені під час XXVIII Міжнародного молодіжного форуму “Радіоелектроніка та молодь у XXI столітті”.

Рисунок Б.14 – Чотирнадцятий слайд презентації

ДОДАТОК В

Специфікація

1 Вступ

1.1 Огляд продукту

Програмна система для управління і контролю якості роботи співробітників IT-компаній є інструментом, призначеним для оптимізації процесів управління проєктами та контролю за ресурсами компанії.

1.2 Мета

Метою цієї системи є створення ефективного інструменту для управління проєктами та ресурсами IT-компаній, що забезпечує прозорість і контроль якості виконання завдань.

1.3 Межі

Система вимагає стабільного інтернет-з'єднання та підтримує тільки сучасні браузери, такі як Chrome і Mozilla Firefox.

1.4 Посилання

Додаткові ресурси посилання для повноцінного розуміння системи та джерел системи:

1. Melnyk, M., Korcelli-Olejniczak, E., Chorna, N., Popadynets, N. Development of Regional IT clusters in Ukraine: institutional and investment dimensions. Economic Annals-XXI, 2018. – P. 19-25.

2. Кузьміних В. О., Коваль О. В., Тараненко Р. А. Моделі та засоби управління IT- проєктами: навч.-метод. посіб. КПІ ім. Ігоря Сікорського, 2023. – 222 с.

3. Trello documentation. [Електронний ресурс] – URL: https://trello.com/en/guide (дата звернення: 01.05.2024)

4. Asana documentation. [Електронний ресурс] – URL: https://developers.asana.com/ (дата звернення: 01.05.2024)

5. Golmgrein, I. A Comprehensive Overview of Monetization Strategies in Creative Industries. International Journal of Latest Engineering and Management Research (IJLEMR), 2023. – P. 90-100.

6. What is Subscription Marketing: Guide. [Електронний ресурс] – URL: https://sendpulse.com/support/glossary/subscription-marketing (дата звернення: 01.05.2024)

7. Corona, E., Pani, F. E. A review of lean-kanban approaches in the software development. WSEAS transactions on information science and applications, 2013. – P. 1-13.

8. Damij, N., Damij, T. An approach to optimizing Kanban board workflow and shortening the project management plan. IEEE Transactions on Engineering Management, 2021. – P. 1-8.

9. What is UML diagrams? [Електронний ресурс] – URL: https://miro.com/diagramming/what-is-a-uml-diagram/ (дата звернення: 01.05.2024)

10. What is an Entity Relationship Diagram (ERD)? [Електронний ресурс] – URL: https://www.lucidchart.com/pages/er-diagrams (дата звернення: 01.05.2024)

11. C# language documentation [Електронний ресурс] – URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 01.05.2024)

12. ASP.NET documentation [Електронний ресурс] – URL: https://dotnet.microsoft.com/en-us/apps/aspnet (дата звернення: 01.05.2024)

13. Entity Framework documentation hub [Електронний ресурс] – URL: https://learn.microsoft.com/en-us/ef/ (дата звернення: 01.05.2024)

14. SQL Server technical documentation [Електронний ресурс] – URL: https://learn.microsoft.com/en-us/sql/sql-server/ (дата звернення: 01.05.2024)

15. React documentation [Електронний ресурс] – URL: https://react.dev/ (дата звернення: 01.05.2024)

16. Typescript documentation [Електронний ресурс] – URL: https://www.typescriptlang.org/docs/ (дата звернення: 01.05.2024)

17. Tailwind CSS documentation [Електронний ресурс] – URL: https://tailwindcss.com/docs/ (дата звернення: 01.05.2024)

18. DaisyUI documentation [Електронний ресурс] – URL: https://daisyui.com/docs/ (дата звернення: 01.05.2024)

1.5 Означення та абревіатури

IT: Information Technology (Інформаційні технології)

API: Application Programming Interface (Інтерфейс програмування додатків)

DB: Database (База даних)

UI: User Interface (користувацький інтерфейс)

UX: User Experience (користувацький досвід)

UML: Unified Modeling Language (уніфікована мова моделювання)

2 Загальний опис

2.1 Перспективи продукту

Програмна система для управління і контролю якості праці співробітників IT-компаній є перспективним продуктом, здатним зайняти значне місце на ринку IT. Система розробляється з урахуванням поточних тенденцій і вимог індустрії, забезпечуючи не тільки управління проєктами, а й контроль за ресурсами та бюджетом компанії.

З урахуванням постійного зростання і розвитку IT-ринку, даний продукт надає важливі інструменти для ефективного управління проєктами та ресурсами. Компанії стикаються зі збільшенням обсягу даних і кількістю проєктів, що вимагає ефективних рішень для контролю та оптимізації роботи. Дана система, що об'єднує функції управління проєктами та управління фінансовими аспектами, задовольняє ці потреби, роблячи її потрібним серед IT-компаній.

Крім того, даний продукт має потенціал для використання не тільки в IT-сфері, а й в інших галузях. Компанії з різних секторів можуть застосовувати цю систему для управління своїми проєктами, контролю за бюджетами та ресурсами. Це робить систему універсальним інструментом, корисним для будь-якого бізнесу, який прагне до поліпшення ефективності та прозорості у своїх операціях. Таким чином, перспектива розширення ринку та залучення компаній з різних галузей робить цей продукт ще більш цінним і потрібним.

2.2 Функції продукту

Основні функціональні вимоги до програмної системи:

- авторизація та реєстрація користувачів;
- можливість придбання підписки;
- редагування проєктів;
- редагування секцій в проєктах;
- редагування завдань в проєкті;
- управління учасниками проєктів;
- управління ролями в проєктах;
- формування статистик проєктів;
- управління компанією;
- управління учасниками компаній;
- управління ролями компаній;
- формування статистик компаній;
- формування повідомлень.

2.3 Характеристики користувачів

Основними користувачами системи є менеджери проєктів, фінансові аналітики IT-компаній, звичайні користувачі, якім необхідно створювати проєкти для своїх особистих завдань, люди, що навчаються, співробітники компаній та представники бізнесу. Усі ці користувачі будуть рахуватися, як звичайний

користувач. Вони зможуть відрізнятися від інших користувачів отримавши додаткові функціональності та можливості в системі, при умові придбання додаткових послуг в системі.

2.4 Загальні обмеження

Клієнтський вебдодаток обмежений веббраузером, встановленим на пристрої користувача. Підключення до Інтернету також є обмеженням для програми. Додаток з'єднується з вебсервером через Інтернет, і не існує способу отримати інформацію без зв'язку між клієнтом і вебсервером.

2.5 Припущення та залежності

Основне припущення полягає в тому, що більшість сучасних браузерів працюють однаково і підтримують новітні технології, такі як HTML5. Якщо браузер не підтримує нові технології, інтерпретація коду та виконання інструкцій може призвести до помилок.

3 Конкретні вимоги

3.1 Вимоги до зовнішнього інтерфейсу

3.1.1 Інтерфейс користувача

Описуючи інтерфейс користувача можливо створити моками сторінок для повноцінного розуміння, як буде виглядати інтерфейс користувача.

Спочатку потрібно розробити мокап вступної сторінки на яку буде заходити користувач при першому заході до системи. На рисунку 1 зображено мокап початкової сторінки системи.

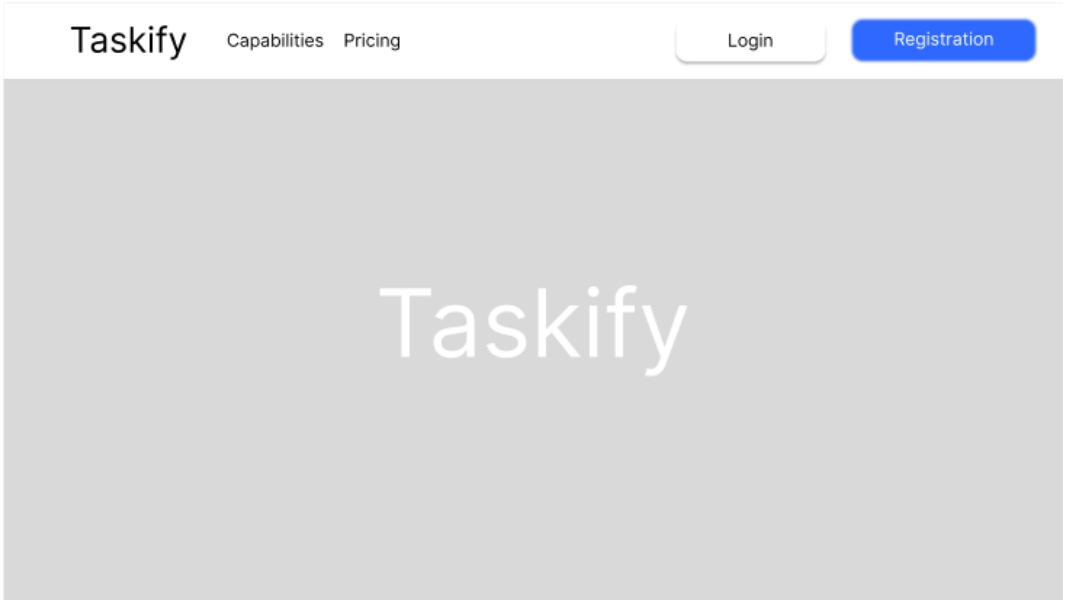


Рисунок В.1 – Мокап початкової сторінки системи

На даній сторінці можливо побачити меню системи, де користувач може перейти на сторінку з можливостями системи натиснувши кнопку “Capabilities”, де він зможе знайти інформацію про систему, при натисканні на кнопку “Pricing” користувач зможе переглянути ціни на продукт, а також на цій сторінці користувач зможе перейти до сторінки логіну та реєстрації в системі натиснувши відповідні кнопки “Login” та “Registration”. На самій сторінці, нижче меню, буде розташована рекламні картинки та описовий текст системи для того, щоб зацікавити користувача спробувати дану систему.

Після реєстрації або логіну в системі користувач потрапляє до основної сторінки системи, через яку він зможе взаємодіяти з іншими частинами системи. На рисунку 2 представлено мокап основної сторінки користувача.

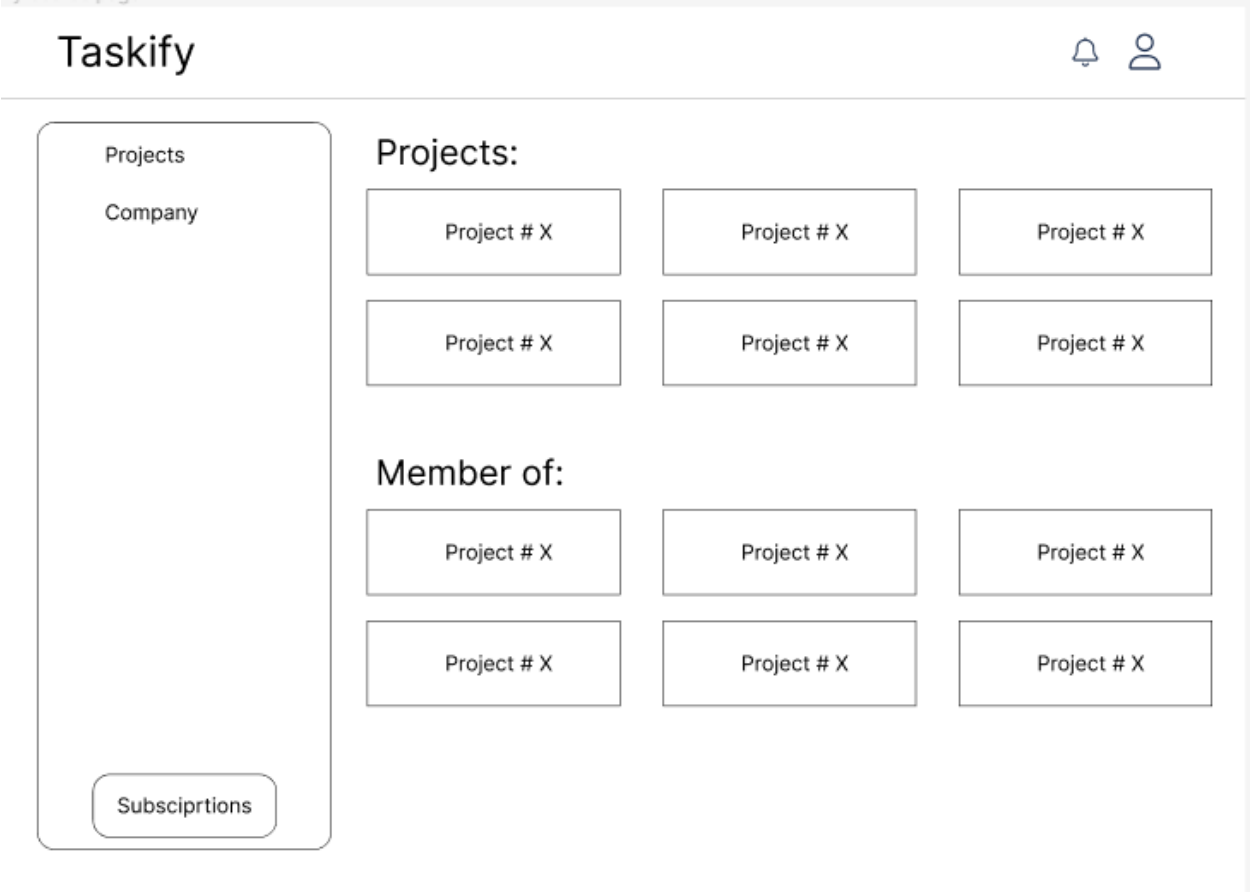


Рисунок В.2 – Мокап основної сторінки користувача програмної системи

На цій сторінці користувач може переглядати свої проєкти та проєкти до яких він був доданий як учасник проєкту. Також в лівій частині буде представлено меню, за допомогою якого, користувач зможе перейти до таких компонентів як проєкти системи, компанії користувача та перейти до розділу підписок. У верхній частині цієї сторінки буде відображатися основне меню системи, де користувач зможе швидко перейти до цієї сторінки, переглянути повідомлення системи та перейти до свого профілю.

Після переходу до сторінки окремого проєкту користувач зможе переглянути основну сторінку проєкту, де буде розташована дошка проєкту. На рисунку 3 зображено мокап основної сторінки проєкту.

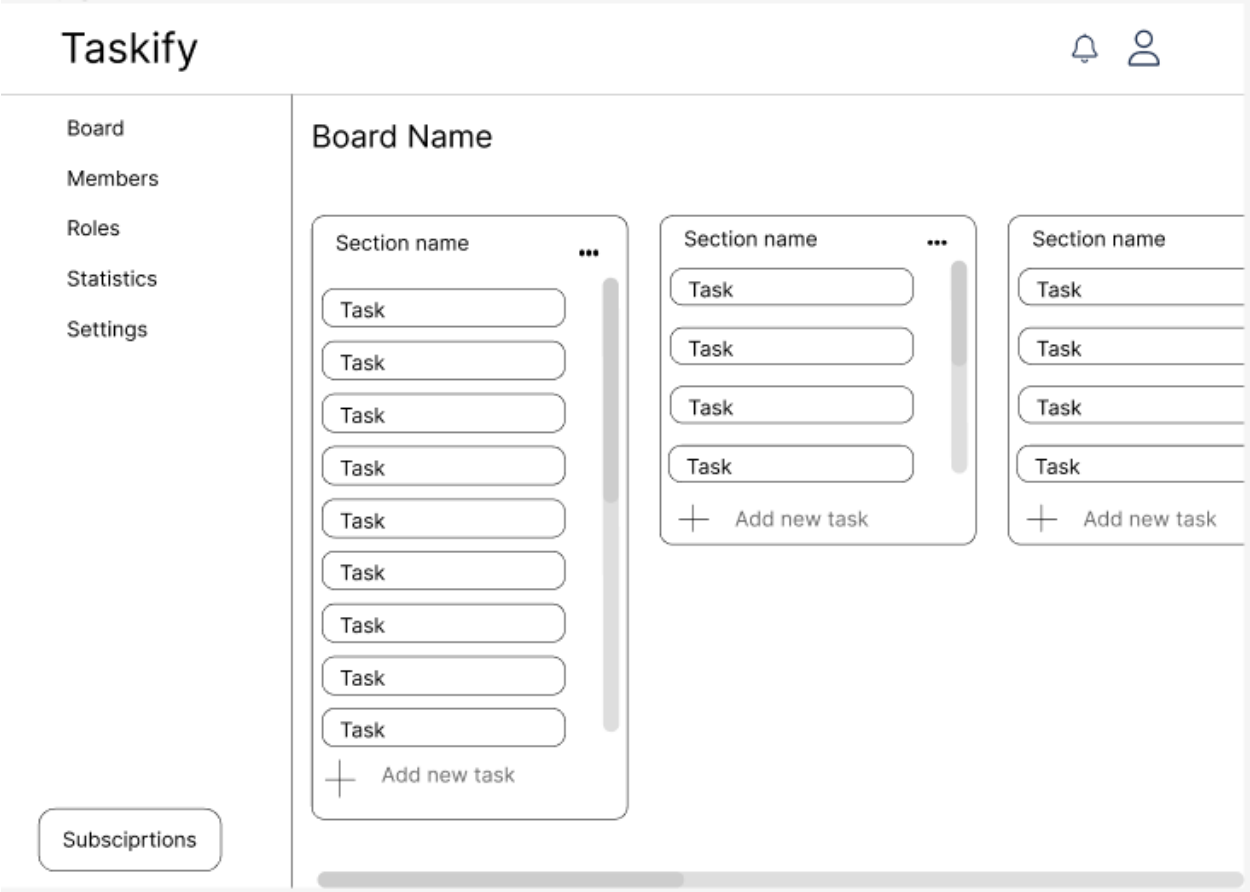


Рисунок В.3 – Мокап основної сторінки проєкту

На цій сторінці в верхній частині відображається основне меню системи, яке було описано раніше, в лівій частині відображається меню окремого проєкту, за допомогою якого, користувач зможе перейти до основних пунктів проєкту таких як: переглянути дошку, переглянути учасників, переглянути ролі, переглянути статистику, продивитись налаштування та переглянути статус своєї підписки. В правій частині сторінки буде розташована інформація про дошку та дії для її зміни, а також дошка Kanban, яка буде допомагати користувачеві зручно взаємодіяти з частинами проєкту. На цій дошці користувач зможе створювати, редагувати, видаляти секції та завдання, та переглядати інформацію завдань. Цією дошкою також можливо буде управляти за допомогою методу drag and drop. Метод drag and drop (перетягування та відпускання) – це інтерактивний підхід, за допомогою якого користувач переміщує елементи на екрані за допомогою миші або екранного пристрою. Він зазвичай використовується в онлайн-додатках, таких як дошки

Kanban, для полегшення управління та організації інформації. За допомогою цього методу можливо буде переміщувати секції та завдання для покращення UX системи.

Далі будуть наводитись мокапи сторінок, які необхідні для повноцінного функціонування проєкту. Наступна сторінка для якої створювався мокап, це сторінка з інформації про користувачів та функціонал для взаємодії з учасниками проєкту. На рисунку 4 відображено мокап сторінки для взаємодії з учасниками проєкту.

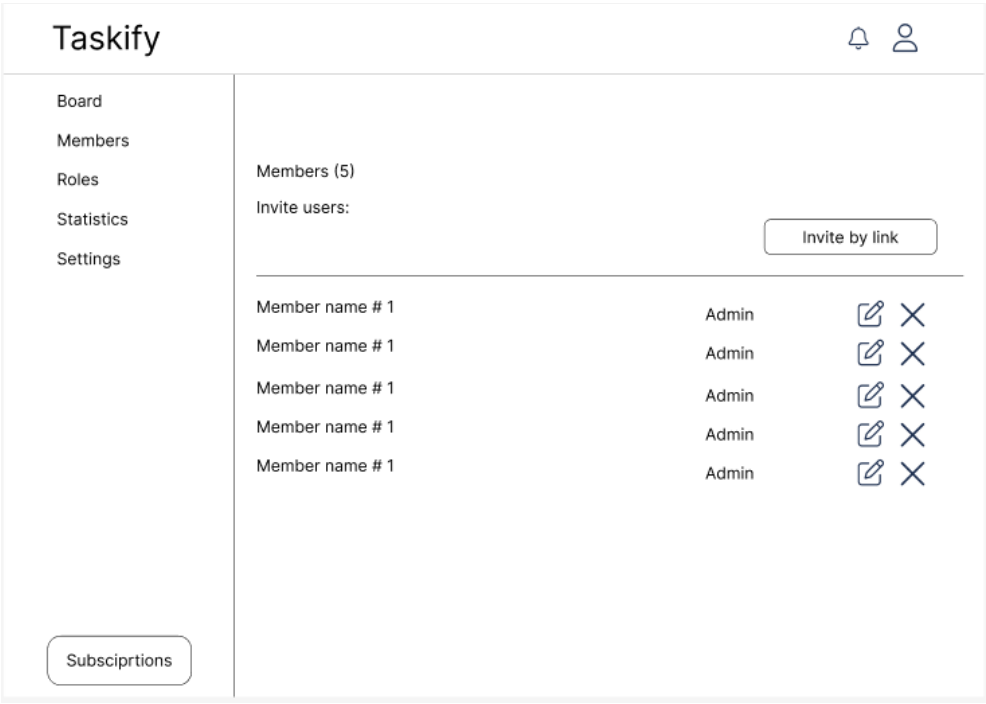


Рисунок В.4 – Мокап сторінки для взаємодії з учасниками проєкту

На цій сторінці, окрім вже описаних верхнього меню та бокового меню, представлено функціонал для взаємодії з учасниками системи. На цій сторінці користувач, який створив проєкт зможе запросити до свого проєкту інших користувачів та переглянути усіх користувачів системи, а також він зможе відредагувати інформацію про учасників проєкту.

Мокап сторінки з ролями проєкту майже не відрізняється від мокапу сторінки для управління учасниками проєкту, тому було вирішено не розробляти мокап цієї

сторінки, а при розробці цієї сторінки переглянути мокап сторінки з учасниками проєкту.

Наступна сторінка для якої необхідно розробити мокап, це сторінка зі статистиками проєкту. На цій сторінці користувач зможе переглянути усі статистики, які представляє системи стосовно окремого проєкту. На рисунку 5 представлено мокап сторінки зі статистиками проєкту.

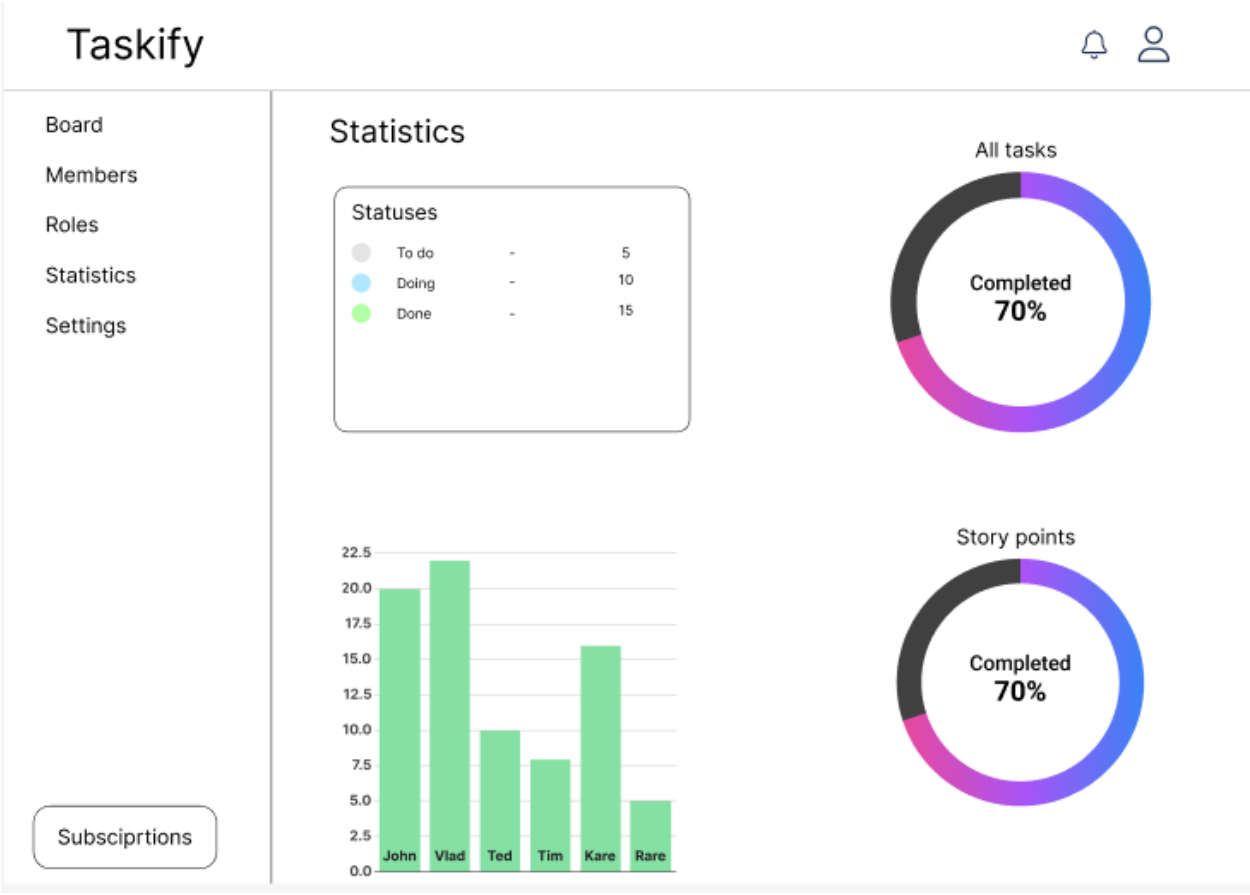


Рисунок В.5 – Мокап сторінки зі статистиками проєкту

На цій сторінці будуть представлені усі основні статистики проєкту. Ці статистики будуть представлені, як в текстовому представлені, так і у вигляді діаграм.

Наступна сторінка це мокап головної сторінки компанії користувача. На рисунку 6 наведено мокап головної сторінки компанії користувача.

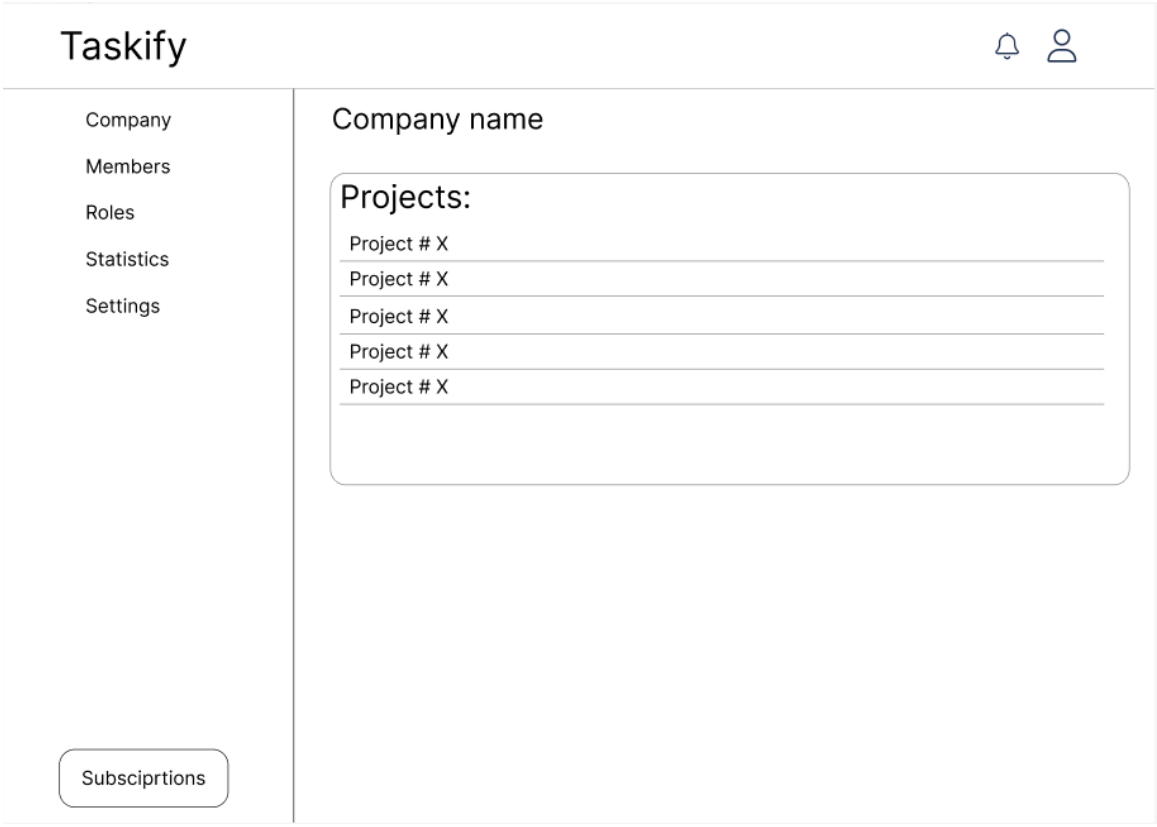


Рисунок В.6 – Мокап головної сторінки компанії користувача

На цій сторінці можливо побачити та змінити назву компанії та продивитись, які є проєкти у компанії та перейти до редагування даних про ці проєкти.

Мокап сторінки з учасниками компанії, з ролями компанії та статистиками не відрізняється від мокапу відповідних сторінок в проєктах, які були описані раніше. Різниця можливо лише з тим, що учасники та ролі будуть входити до складу компанії, а на сторінці зі статистиками буде приділена увага більше статисткам пов’язаних з бюджетом компаній.

3.1.2 Програмний інтерфейс

Для доступу до вебсистеми користувач повинен використовувати веббраузер. Воно повинно підтримувати Javascript, HTML5, CSS3.

3.1.3 Комунікаційний протокол

Для того, щоб користувач мав доступ до вебсистеми, необхідно встановити з'єднання з Інтернетом.

3.1.4 Обмеження пам’яті

Максимальний обсяг даних, оброблюваних системою, обмежений 2 ГБ оперативної пам'яті на кожного користувача.

3.1.5 Операції

Операції в системі включають такі дії:

– створення завдання: для створення завдання необхідно вибрати відповідний проєкт та ввести назву завдання;

– редагування завдань: користувач може змінити параметри вже наявного завдання, як-от назву, опис, строки виконання, відповідальних осіб та інші. Можна також переміщати завданню по секціях всередині проєкту для зміни порядку;

– видалення завдань: для видалення завдання користувач повинен обрати завдання та підтвердити його видалення, щоб уникнути випадкових дій;

– створення проєктів: користувачі можуть створювати нові проєкти обравши назву для проєкту;

– редагування проєктів: користувач може змінити назву та інші параметри проєкту;

– видалення проєктів: для видалення проєкту необхідно підтвердити дію, щоб уникнути втрати даних;

– створення секцій: користувач може створювати нові секції, задаючи їм назви. Секції використовуються для організації завдань за статусами (наприклад, «To do», «Doing», «Done»);

– редагування секцій: користувач може змінювати назву та інші параметри наявних секцій. Можна також переміщати секції всередині Kanban-дошки для зміни порядку;

– видалення секцій: для видалення секції необхідно підтвердити дію. Усі завдання у видаленій секції буде переміщено в іншу секцію;

– створення запрошень у проєкт: користувач може запросити інших користувачів у проєкт, вказавши їхні email. Запрошені користувачі отримають повідомлення і зможуть приєднатися до проєкту після прийняття запрошення;

– створення запрошення в компанію: адміністратор компанії може запросити нових учасників, вказавши їхні email. Запрошені користувачі отримають повідомлення і зможуть приєднатися до компанії після прийняття запрошення;

– створення ролі в компанії: адміністратор компанії може створювати нові ролі, задаючи їхню назву і набір прав доступу. Ролі допомагають розподіляти обов'язки й обмежувати доступ до певних функцій;

– редагування ролі в компанії: адміністратор може змінювати наявні ролі, додаючи або прибираючи права доступу;

– видалення ролі в компанії: адміністратор може видалити роль, якщо вона більше не потрібна. Користувачі з цією роллю отримають іншу роль або буде присвоєна стандартна роль;

– видалення учасників проєкту: адміністратор проєкту може видалити учасника з проєкту, якщо його участь більше не потрібна. Видалений учасник втратить доступ до всіх даних проєкту;

– редагування даних учасників проєкту: адміністратор проєкту може змінювати дані учасників, включно з їхньою роллю в проєкті;

– видалення учасників компанії: адміністратор компанії може видалити учасника, якщо він більше не працює в компанії. Видалений учасник втратить доступ до всіх даних компанії;

– редагування даних учасників компанії: адміністратор може змінювати дані учасників;

– перегляд статистик: користувачі можуть переглядати статистичні дані щодо проєктів і компанії. Це допомагає аналізувати прогрес і ефективність роботи;

– авторизація: для авторизації в системі користувачеві необхідно ввести правильний логін і пароль. Після успішної авторизації користувач отримує доступ до своїх проєктів і даних;

– реєстрація: для реєстрації нового користувача необхідно ввести особисті дані, такі як ім'я, email і пароль.

3.1.6 Функції продукту

Основні функції програмної системи:

– управління проєктами: система повинна дозволяти створювати, редагувати та проводити моніторинг проєктів. Користувачі мають мати змогу призначати завдання та слідкувати за їх виконанням. Візуальні статистики системи та Kanban-дошка допоможуть візуалізувати прогрес. Повинна бути можливість додавати та видаляти учасників проєктів, призначати ролі та відстежувати їхню діяльність. Це дозволить підвищити ефективність управління проєктами, спростити координацію команд та покращити контроль за виконанням завдань;

– статистика продуктивності: система повинна надавати статистику з відстежування продуктивності та якості праці учасників проєктів, що допоможе керівникам ухвалювати обґрунтовані рішення щодо покращення ефективності команди;

– управління компанією: користувачі повинні мати можливість створювати компанії, управляти ресурсами компанії, редагувати дані компанії, відстежувати бюджет, витрати та доходи, а також керувати учасниками компанії та їх ролями. Це дозволить компаніям ефективно контролювати свої фінансові показники, оптимізувати витрати та доходи, а також покращити управління людськими ресурсами;

– управління ресурсами: система повинна забезпечувати інструменти для ефективного розподілу та управління ресурсами компанії. Це сприятиме оптимальному використанню ресурсів, підвищенню продуктивності та зниженню витрат.

3.1.7 Припущення та залежності

Передбачається, що система працюватиме в середовищі з постійним доступом до інтернету і підтримуваними браузерами. Також система залежить від регулярного оновлення даних зовнішніми API.

3.2 Властивості програмного продукту

Програмна система має бути надійною, забезпечуючи стійку роботу без збоїв і високу доступність для користувачів. Система має бути доступною, що передбачає легкий доступ через веббраузери на різних пристроях та інтуїтивно зрозумілий інтерфейс, що спрощує її використання. Важливе значення має безпека системи, що охоплює аутентифікацію та авторизацію, шифрування даних і захист від несанкціонованого доступу. Програмне забезпечення також має бути продуктивним і масштабованим, здатним ефективно обробляти великі обсяги даних і підтримувати одночасну роботу безлічі користувачів, що забезпечить високу швидкість відгуку і стабільність роботи.

3.3 Атрибути програмного продукту

3.3.1 Надійність

Система має бути стабільною та надійною, забезпечуючи високу доступність та мінімальну кількість збоїв.

3.3.2 Доступність

Система має бути доступна завжди окрім планових або позапланових перерв.

3.3.3 Безпека

Система має бути забезпечена багаторівневими заходами захисту, включаючи автентифікацію та авторизацію користувачів, шифрування даних, контроль доступу до функціональних можливостей системи на основі прав користувачів.

3.3.4 Супроводжуваність

Код системи має бути структурований та задокументований, що полегшує її підтримку та модернізацію.

3.3.5 Переносимість

Система повинна мати можливість бути розгорнута на будь-яких серверах, що підтримують технології використані під час розробки системи.

3.3.6 Продуктивність

Система повинна бути здатна обробляти великий обсяг даних та підтримувати роботу з безліччю користувачів. Час відгуку системи має бути мінімальним.

3.4 Вимоги бази даних

Програмна система повинна мати можливість взаємодії з будь-якою реляційною базою даних, забезпечуючи коректну роботу незалежно від того, яка саме база даних використовується. Для досягнення високої продуктивності та надійності, база даних має бути сучасною, добре налаштованою та оптимізованою для обробки великих обсягів даних. Необхідно передбачити захист даних на всіх рівнях, включно з регулярним резервним копіюванням і відновленням. Крім того, база даних повинна підтримувати високий рівень доступності, забезпечуючи безперервний доступ до даних навіть за високого навантаження. Важливим аспектом є забезпечення цілісності даних і запобігання їхній втраті або

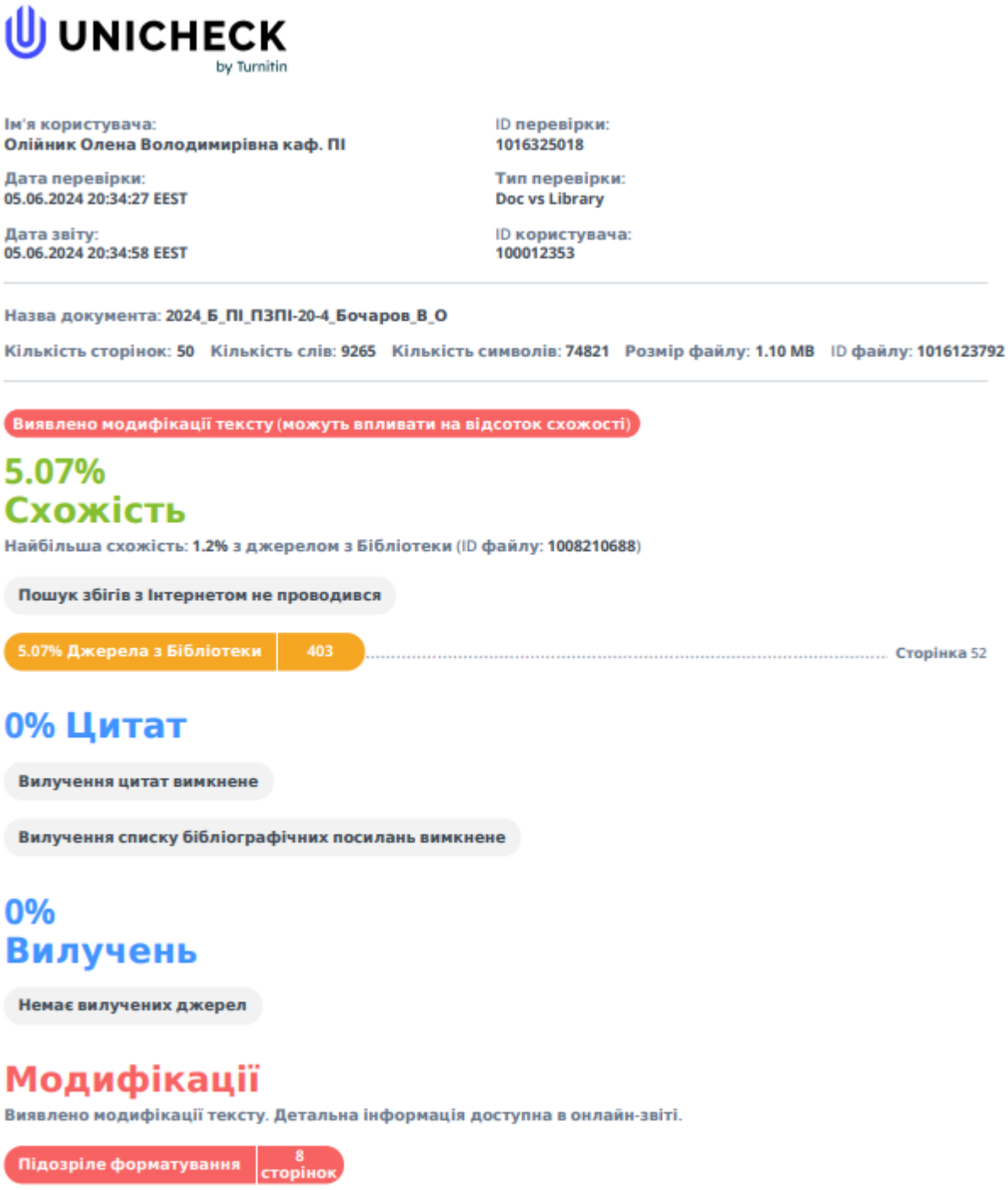
пошкодженню внаслідок збоїв або помилок. Також необхідно враховувати можливість масштабування бази даних для підтримки зростання обсягів даних і кількості користувачів системи.

3.5 Інші вимоги

Програмна система має бути кросплатформной, забезпечуючи роботу на різних операційних системах і пристроях, включно з настільними комп'ютерами, ноутбуками, планшетами та смартфонами. Важливо забезпечити сумісність із різними веббраузерами для зручності користувачів. Необхідно забезпечити високу продуктивність і мінімальний час відгуку, навіть при роботі з великими обсягами даних і високому навантаженні.

ДОДАТОК Г

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



UNICHECK
by Turnitin

Ім'я користувача:
Олійник Олена Володимирівна каф. ПІ

ID перевірки:
1016325018

Дата перевірки:
05.06.2024 20:34:27 EEST

Тип перевірки:
Doc vs Library

Дата звіту:
05.06.2024 20:34:58 EEST

ID користувача:
100012353

Назва документа: **2024_Б_ПІ_ПЗПІ-20-4_Бочаров_В_О**

Кількість сторінок: **50** Кількість слів: **9265** Кількість символів: **74821** Розмір файлу: **1.10 MB** ID файлу: **1016123792**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

5.07%
Схожість

Найбільша схожість: **1.2%** з джерелом з Бібліотеки (ID файлу: **1008210688**)

Пошук збігів з Інтернетом не проводився

5.07% Джерела з Бібліотеки | 403 ..... Сторінка 52

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування | 8 сторінок