Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка веб-платформи для організації коворкінгу з використанням технологій JavaScript, React, Postgre SQL та Node.js»

Виконала: студентка IV курсу
групи 1ПІ-20б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Назаренко Т.Є.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Катєльніков Д.І.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Богомолов С.В.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри ПЗ _____
«10» червня 2024 р.

ВНТУ – 2024

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»»
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н
12 березня 2024 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Назаренко Тетяні Євгенівні

1. Тема роботи – розробка веб-платформи для організації коворкінгу з використанням технологій JavaScript, React, Postgre SQL та Node.js.

Керівник роботи: Катєльніков Денис Іванович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 11 березня 2024 р. №80.

2. Строк подання студентом роботи 3 червня 2024 р.

3. Вихідні дані до роботи: модель розробки – ітеративна; алгоритми реалізації – управління серверною частиною застосунку з використанням Node.js для обробки запитів; сервер баз даних – PostgreSQL; розробка інтерфейсу користувача – React; вхідні дані – дані про користувачів, список проектів, статус робочих задач; вихідні дані – звіти про прогрес проектів, розподіл завдань між учасниками команди; середовище розробки – Visual Studio Code; мова розробки – JavaScript; операційна система – Windows 10/11; середовище реалізації додатку – браузер.

4. Зміст розрахунково-пояснювальної записки: вступ; обґрунтування вибору методу розробки та постановка задачі дослідження; розробка структури, моделі та алгоритмів веб-платформи; розробка модуля функцій керування завдань; розробка модуля керування адміністраторів; розробка модуля завантаження вкладень; тестування веб-платформи; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; мета; задачі дослідження; актуальність дослідження; об'єкт дослідження; предмет дослідження; практична цінність; наукова новизна; існуючі аналоги; аналіз аналогів; діаграма моделі системи; блок-схема алгоритму роботи головного екрану; блок-схема алгоритму роботи керування дошками; вибір технологій, середовища та СКБД; архітектура бази даних; тестування веб-платформи; апробація та публікація результатів роботи; висновки; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата			
		завдання видав		завдання прийняв	
1-4	Катєльніков Д.І., к.т.н., доцент кафедри ПЗ	13.03.24		03.06.24	

7. Дата видачі завдання 13 березня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз проблеми, обґрунтування актуальності розробки веб-платформи та постановка задачі	14.03.24 – 25.03.24	виконано
2	Розробка структури, моделі та алгоритмів роботи веб-платформи	26.03.24 – 10.04.24	виконано
3	Аналіз і вибір середовища програмування та СКБД	11.04.24 – 18.04.24	виконано
4	Розробка модулів веб-платформи	19.04.24 – 08.05.24	виконано
5	Тестування роботи веб-платформи та розробка інструкції користувача	09.05.24 – 24.05.24	виконано
6	Оформлення матеріалів до захисту БДР	25.05.24 – 30.05.24	виконано

Студентка _____ (підпис) **Назаренко Т.Є.** (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ (підпис) **Катєльніков Д.І.** (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 96 сторінок формату А4, на яких є 34 рисунків, 8 таблиць, список використаних джерел містить 22 найменувань.

У бакалаврській роботі проаналізовано сучасний стан веб-платформ для організації коворкінгу. Визначено об'єкт, предмет, завдання та методи дослідження. Метою дослідження є покращення якості веб-платформ для організації коворкінгу, з метою полегшення організації роботи за проектами користувачів. Для досягнення поставленої мети розроблено структуру програмного додатку, що підтримує процес управління проектами в умовах спільної роботи, алгоритми роботи основних програмних модулів, інтерфейс додатку та програмну реалізацію.

Запропоновано нові алгоритми ефективного управління завданнями та ресурсами в умовах коворкінгу, що сприятиме підвищенню продуктивності роботи команди та покращенню організації проектів.

Веб-платформа була розроблена в середовищі розробки Visual Studio Code з використанням мови програмування JavaScript, React та фреймворку Node.js. В результаті виконання бакалаврської роботи було розроблено веб-платформу та протестовано її на працездатність і коректність.

Отримані результати можуть бути використані для впровадження веб-платформи в робочі процеси коворкінгу та допоможуть покращити організацію та управління проектами.

Ключові слова: веб-платформа; коворкінг; JavaScript; управління проектами.

ABSTRACT

The Bachelor's thesis consists of 96 A4 pages, including 34 figures, 8 tables, and the list of references contains 22 titles.

In the bachelor's thesis analyzes the current state of web platforms for coworking organization. The object, subject, tasks, and research methods are identified. The aim of the study is to improve the quality of web platforms for coworking organization, aiming to facilitate project management for users. To achieve this goal, the structure of the software application supporting project management in collaborative work, algorithms of main software modules operation, application interface, and software implementation were developed.

New algorithms for efficient task and resource management in coworking conditions are proposed, which will enhance team productivity and project organization.

The web platform was developed using Visual Studio Code environment, utilizing JavaScript programming language, React, and Node.js framework. As a result of the Bachelor's thesis, a web platform was developed and tested for functionality and correctness.

The obtained results can be used for the implementation of the web platform into coworking work processes and will help improve project organization and management.

Keywords: web platform; coworking; JavaScript; project management.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Розвиток інформаційних технологій в останні часи вплинув на організацію робочих просторів, спонукаючи до створення нових форм співпраці. Так народився концепт коворкінгу, який став популярним серед різних груп користувачів. Фрілансери, стартапи, незалежні професіонали, представники великих компаній та навіть студенти використовують коворкінг для забезпечення себе робочим простором та можливостями спілкування з колегами.

У першій хвилі свого розвитку офлайн коворкінг в основному привертав фрілансерів, незалежних професіоналів та малі стартапи, які шукали спільне середовище для роботи та обміну ідеями. Проте, з розвитком технологій та змінами у структурі економіки, коворкінг став популярним для більших компаній у онлайн сфері [1], які використовують його для проведення тренінгів, зустрічей та спільних проектів.

Зростаюча популярність коворкінгу викликала потребу у нових інструментах для організації та управління робочими процесами. Веб-платформи для коворкінгу стали важливими інструментами, які дозволяють користувачам знаходити спільні робочі простори, керувати проектами, обмінюватися ідеями та спілкуватися з колегами.

Завдяки розвитку інтернет-технологій, віддалена робота стала все більш поширеною, і тепер онлайн коворкінг дозволяє співпрацювати з будь-якої точки світу [2]. Впровадження віртуального коворкінгу стає досить суттєвим кроком у напрямку полегшення зосередження та підвищення продуктивності роботи. Розробка веб-платформи для організації коворкінгу стає надзвичайно актуальною у сучасному світі, де віддалена робота стає все більш поширеною та необхідною. Зростаюча глобалізація та швидкий розвиток технологій призводять до того, що все більше професійних команд та індивідуальних спеціалістів працюють з різних кутків світу. В таких умовах віртуальний коворкінг стає

важливим інструментом для забезпечення спільної робочої атмосфери, сприяє зближенню та ефективному виконанню завдань.

Актуальність розробки полягає в тому, що вона вирішує ряд проблем, які виникають у віддалених команд та індивідуальних працівників. Віртуальний коворкінг допомагає створити спільний робочий простір, де люди можуть обмінюватися ідеями, спілкуватися та взаємодіяти, незважаючи на фізичну відстань. Крім того, це забезпечує ефективну організацію робочих процесів, включаючи планування, спільне виконання завдань та управління проектами.

Таким чином, розробка веб-платформи для коворкінгу відповідає сучасним вимогам та потребам професійних спільнот, які працюють у віддаленому форматі.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою дослідження є підвищення ефективності веб-платформи для організації коворкінгу, спрямованої на полегшення організації роботи з проектами для користувачів у реальному часі.

Відповідно до поставленої мети в бакалаврській дипломній роботі потрібно вирішити такі **завдання**:

- проаналізувати стан платформ для організації коворкінгу;
- проаналізувати вибір технологій та середовища розробки;
- розробити метод та алгоритм роботи керування доступом до проектів;
- розробити метод та алгоритм роботи керуванням проектів;
- розробити дизайн інтерфейсу користувача;
- провести тестування веб-платформи.

Об'єкт дослідження є процес розробки програмного забезпечення веб-платформи для організації коворкінгу.

Предметом дослідження є алгоритми та методи реалізації платформи коворкінгу.

Методи дослідження. Протягом дослідження було використано: теорія людино-машинної взаємодії, теорія баз даних при організації зберігання інформації, теорія розподілених систем для побудови веб-платформи для організації коворкінгу, тестування для перевірки правильності функціонування платформи та відповідності вимогам.

Новизна отриманих результатів.

Подальшого розвитку отримав метод управління завданнями проекта, у якому на відміну від існуючих методів управління завданнями, застосовується асинхронна обробка запитів за допомогою Node.js, що дозволяє зменшити затримки при оновленні інформації в реальному часі та підвищити продуктивність системи на 20%.

Подальшого розвитку отримав метод адміністрування користувачів, у якому на відміну від існуючих методів адміністрування, застосовується централізована систему керування доступом на базі PostgreSQL, яка дозволяє підвищити безпеку даних та спрощує процес адміністрування за рахунок централізованого контролю доступу.

Практична цінність отриманих результатів. Полягає в створенні програмних засобів платформи для організації коворкінгу, що дозволяє ефективно керувати робочими просторами, завданнями та користувачами при управлінні проектами.

Апробація матеріалів результатів роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на міжнародній науково-практичній конференції «Сучасні тенденції розвитку фінансових та інноваційно-інвестиційних процесів в Україні (2024)».

Публікації. Основні результати дослідження опубліковані в науковій роботі [3] – в тезах доповіді на міжнародній науково-практичній конференції «Сучасні тенденції розвитку фінансових та інноваційно-інвестиційних процесів в Україні (2024)».

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Аналіз стану веб-платформ для організації коворкінгу

В наш час люди стали все більше цінувати комфортні умови для праці. Проте, знаходження таких умов може бути складним завданням, оскільки не завжди можна знайти ідеальне робоче середовище або мати можливість працювати на робочому місці. Такі обмеження часто стають причиною проблеми з пошуком роботи. Онлайн коворкінг вирішує ці проблеми, пропонуючи гнучкість та доступність без потреби фізично перебувати в певному місці. Це особливо актуально в контексті зростання дистанційної роботи та потреби в глобальному співробітництві, де люди можуть працювати разом незалежно від свого місця проживання.

Коворкінг – це метод організації робочого процесу, при якому люди разом працюють над проектами, використовуючи робочі простори та інструменти для роботи. Надання можливостей спілкуватися і обмінюватися ідеями та досвідом з колегами, що впливає на продуктивний і ефективний робочий процес – це все говорить про саму ідею створення коворкінгово простору.

Багатьом людям коворкінг здається оптимальним варіантом для зручної роботи, оскільки вони мають доступ до всіх необхідних робочих зручностей, не потребуючи присутності на робочому місці. Коворкінг може бути зручним способом працювати в комфортному оточенні, забезпечуючи всі необхідні умови для виконання завдань та керування проектами та командою.

Сучасні програми для керування проектами стали необхідним інструментом для будь-якої команди. Вони значно спрощують і прискорюють процес роботи, особливо якщо це великий проект, завдяки своїй універсальності та багатофункціональності. Управління проектами в онлайн коворкінгу передбачає розподіл ресурсів для досягнення максимально ефективних результатів та високої якості, з урахуванням ресурсів, спрямованих на досягнення поставлених цілей.

Здавалося б, коворкінг це ідеальне рішення для сучасних команд і проектів. Свобода робочого графіка, доступ до інфраструктури та можливість обміну ідеями – все це робить його привабливим вибором для творчих і технологічних спеціалістів. Але якщо приглянутися уважніше, стає очевидним, що навіть у такому ідеальному середовищі існують певні труднощі.

Стан платформ для коворкінгу у сфері проектного менеджменту, постійно змінюється відповідно до змін у вимогах та потребах користувачів. Однією з проблем, які можуть виникнути, є нестача універсальних інструментів для роботи над проектами, які б задовольняли потреби різних команд. Хоча багато платформ пропонують широкий спектр функцій, часто вони не ідеально відповідають конкретним вимогам користувачів. Наприклад, одна команда може шукати інструмент для активного спілкування та співпраці над завданнями в реальному часі, тоді як інша команда може потребувати інструмент для управління великими обсягами даних проектів. Така нестача універсальних інструментів може призводити до того, що команди втрачають час на пошук та вибір оптимального рішення для своїх потреб.

Однією з найбільш поширених проблем при використанні веб-платформ для організації коворкінгу є складний та незрозумілий інтерфейс, де у більшості випадків в інтерфейсі завжди присутні багато функцій які зовсім не потрібні для роботи або рідко використовуються. Це особливо ускладнює роботу для новачків, які тільки починають працювати з платформою. Через це люди бояться використувувати платформу бо бояться, що їхні колеги не зможуть розібратися в її управлінні через складність інтерфейсу.

Окрім цього проблема з інтеграціями є не менш важливою, у контексті коворкінгу, де працюють різні команди з різними потребами, інтегровані інструменти є критично важливими для забезпечення зручного та ефективного спілкування та співпраці між учасниками проекту. Однак, інтеграція різних функцій та інструментів може бути складною через їх різнорідність та різні протоколи комунікації. Наприклад, одна команда може використовувати систему управління завданнями типу Kanban, тоді як інша команда може

використовувати методологію Scrum. Інші можуть підтримувати різні інструменти для спільної роботи над документами або плануванням зустрічей. Щоб вирішити цю проблему, веб-платформа для коворкінгу має мати можливості стандартизації та уніфікації інтеграції функцій. Це може включати розробку універсальних API (інтерфейсів програмування додатків), які дозволять різним інструментам взаємодіяти між собою без проблем.

Отже, розробка веб-платформи для організації коворкінгу є доцільною з точки зору удосконалення робочого простору, підвищення ефективності співпраці та досягнення кращих результатів у сфері організації коворкінгу. Розробка веб-платформи відповідає потребам сучасного бізнесу та сприяє створенню продуктивного та комфортного робочого середовища для всіх учасників коворкінгової спільноти.

1.2 Порівняльний аналіз аналогів

Зростання популярності онлайн робочих просторів породжує попит на веб-платформи, що спрощують організацію та управління проектами. Однак, на ринку існує багато існуючих рішень таких веб-платформ, кожен з яких має свої переваги та недоліки.

Для підтвердження актуальності та доцільності розробки веб-платформи для організації коворкінгу, було проведено аналіз існуючих аналогів. Цей аналіз дозволив розглянути широкий спектр інструментів та функціоналу, що вже пропонуються на ринку, та порівняти їх з власною розробкою веб-платформи. Розглянемо деякі приклади рішень:

- Jira;
- Notion;
- Asana;
- Monday.

Перший конкурент – Jira. Це не просто інструмент для управління проектами, але своєрідний робочий простір, де команди можуть спільно працювати, обмінюватися ідеями та взаємодіяти. Часто використовується для

розробки програмного забезпечення, але корисний і для інших типів проектів або завдань. Має продуманий інтерфейс та розширені можливості для роботи з проектами (рисунок 1.1).

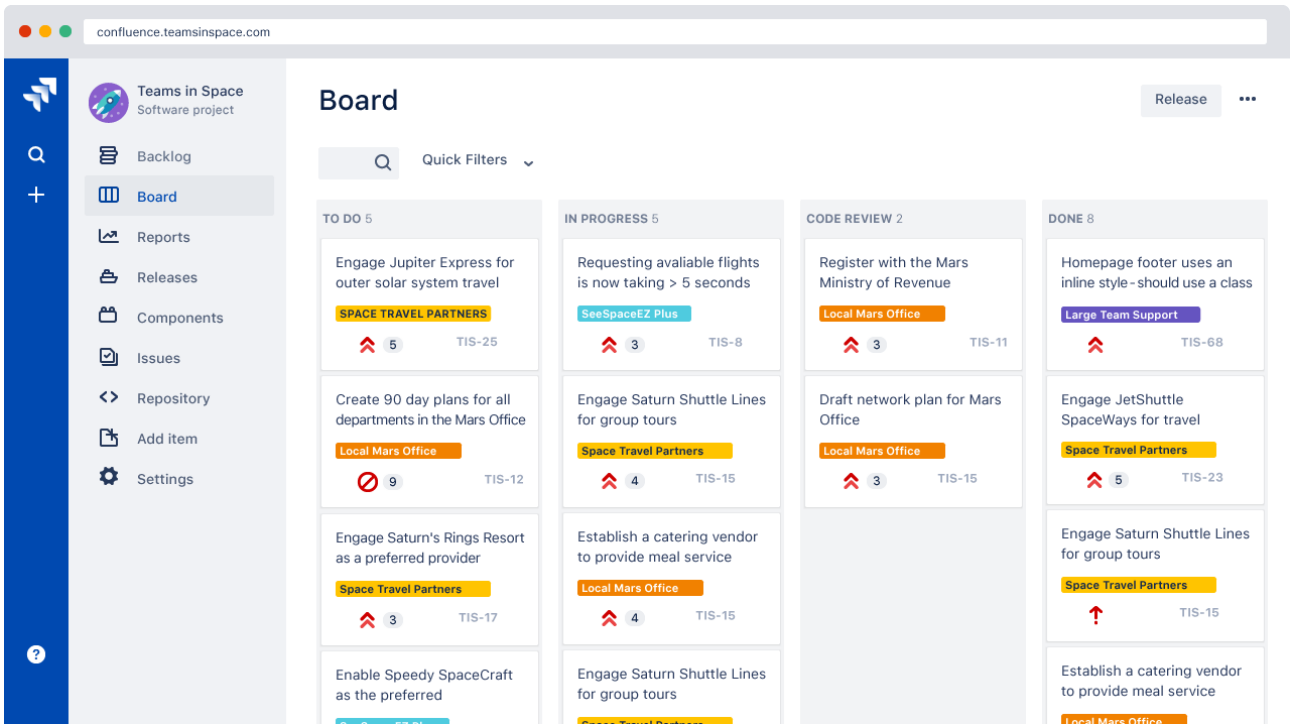

Рисунок 1.1 – Інтерфейс веб-платформи Jira

Jira дозволяє створювати завдання, назначати виконавців, встановлювати пріоритети та відстежувати їх виконання [4]. Увесь життєвий цикл проекту контролюється через Jira, забезпечуючи збереження інформації від початку до кінця та спрощуючи взаємодію завдяки зрозумілому інтерфейсу.

Однією з особливостей Jira є його гнучкість і можливість налаштування під потреби користувачів. За роки розвитку Jira стала потужним інструментом, який поєднує підходи Scrum і Kanban для гнучкого використання в різних проектах. Користувачі можуть легко налаштовувати параметри, створювати правила для бізнес-процесів та інше.

Переваги платформи полягають у широкому функціоналі та гнучкості використання, де можна виконувати багато операцій для керування завданнями та проектами та аналізу результатів проектів. Наявність широкого вибору

шаблонів для проектів та можливість інтеграції дозволяє працювати без обмежень у робочому просторі.

Проте, існують певні недоліки, пов'язані з багатофункціональністю та інтерфейсом Jira, що може зробити її використання складним як для новачків, так і для досвідчених користувачів. Може призвести до перевантаження проектів файлами, сповільнення роботи та завантаження самого робочого простору. Деякі функції доступні лише за підпискою, що може бути недоступним фінансово для деяких користувачів.

Ще однією веб-платформою є відомий Notion [5], призначений для організації робочих просторів і проектів. Notion – це потужний інструмент, що об'єднує можливості створення нотаток, організації завдань та проектів, спільної роботи команди та багато іншого. Інтерфейс Notion вражає своєю простотою та інтуїтивністю (рисунок 1.2). Він має чистий дизайн і легко структурує інформацію.

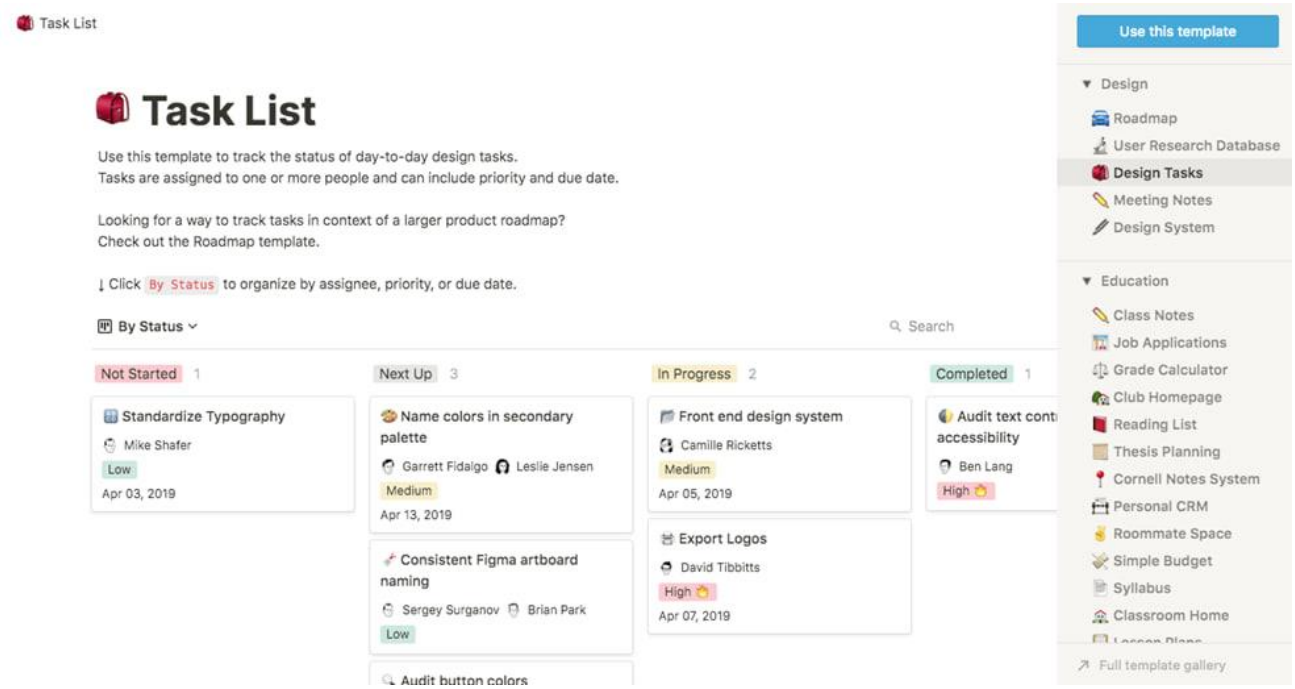

Рисунок 1.2 – Інтерфейс роботи Notion

Користувачі можуть створювати сторінки з різними типами контенту, такими як: тексти, таблиці, списки, фотографії, відео та інше. Однією з ключових переваг є гнучкість налаштування, де можна створювати свої шаблони та

структурувати проекти відповідно до ваших потреб. Крім того, Notion підтримує спільну роботу, де можна ділитися сторінками та взаємодіяти з колегами в реальному часі. Однак для користування можливістю спільної роботи необхідне підключення до швидкісного інтернету, а також освоєння інтерфейсу може зайняти певний час для новачків.

Ще однією веб-платформою вартою уваги є Asana. Її привабливість полягає у стильному та адаптивному дизайні (рисунок 1.3), який робить її ідеальним інструментом для співпраці команд, де кожен може ефективно відстежувати хід роботи та делегувати завдання. Безкоштовна версія з широким спектром функцій робить Asana надійним рішенням для роботи у команді.

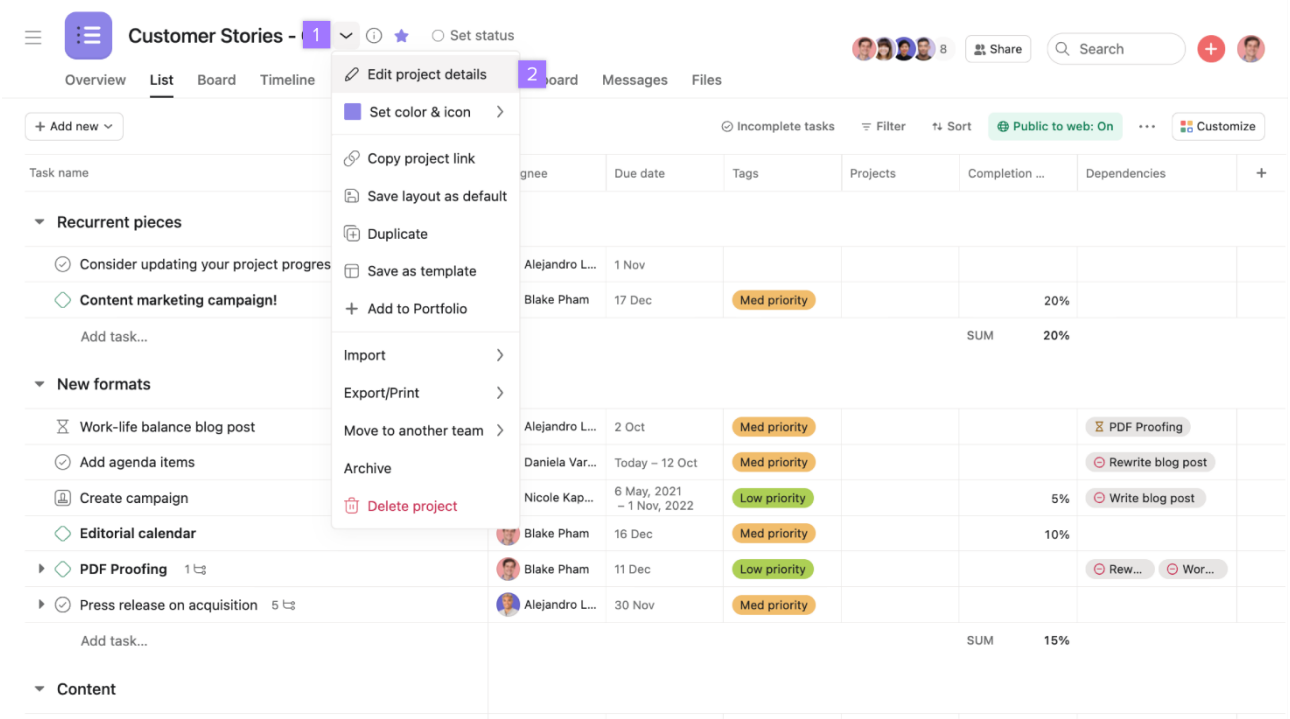

Рисунок 1.3 – Робочий простір Asana

Asana оснащує команди необхідними інструментами для підвищення продуктивності і надає можливості для ефективного моніторингу завдань, робочих процесів і різних типів проектів. Проте є кілька недоліків, таких як можливі обмеження для проектів з великою кількістю графічного матеріалу або мультимедійного вмісту. Крім того, ціни на Asana можуть бути трохи вищими

порівняно з іншими аналогічними платформами, що може вплинути на вибір користувачів.

Monday [6] – це хмарна програма для керування проектами (рисунок 1.4), яку використовують для візуального планування, відстеження та контролю завдань і проектів. Це рішення, яке вражає своєю універсальністю та зручністю у налаштуванні робочих процесів, функціях відстеження часу та ідеальній інтеграції з іншими інструментами.

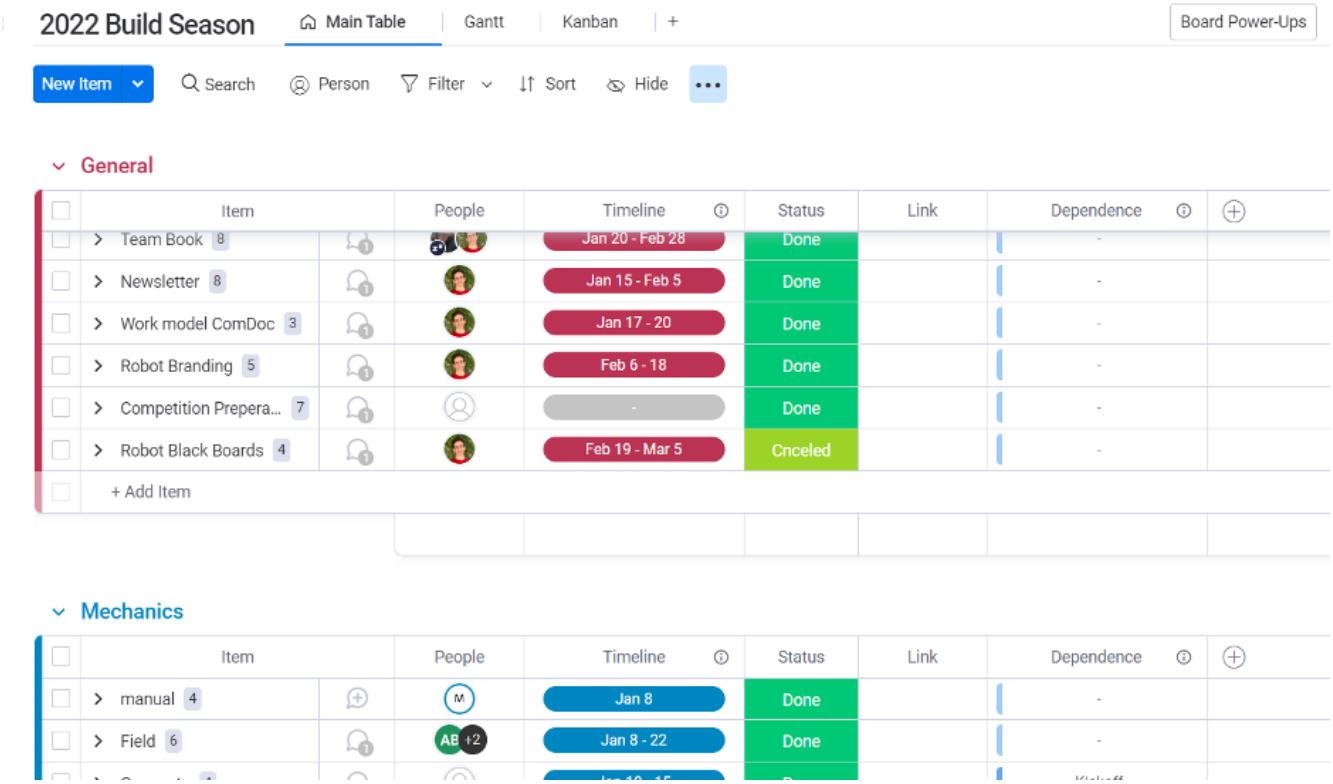

Рисунок 1.4 – Інтерфейс проекту в Monday

Платформа має дружній інтерфейс та інтуїтивно зрозумілі шаблони, якими можуть користуватися навіть користувачі без технічного досвіду. Крім того, вона пропонує бездоганну інтеграцію з іншими інструментами, дозволяючи користувачам оптимізувати свої робочі процеси, об'єднуючи необхідні інструменти в одному місці. Функції автоматизації допомагають ефективно виконувати повторювані завдання, такі як призначення завдань, сповіщення та оновлення статусу, що сприяє збільшенню продуктивності та заощадженню часу.

Позитивні сторони полягають у зручності вирішення завдань та встановлення крайніх термінів за допомогою налаштованого та зручного робочого процесу. Також забезпечується ефективне спілкування між членами команди завдяки інструментам для співпраці, які полегшують обмін файлами, коментарями та оновленнями щодо прогресу проекту.

Щодо недоліків, Monday може вважатися вартістю порівняно з іншими інструментами, особливо для невеликих команд або окремих користувачів. Крім того, платформа не має розширених функцій звітності, що ускладнює аналіз даних та оцінку ефективності проекту.

Результати порівняння аналогів продемонстровано в таблиці 1.1.

Таблиця 1.1 – Порівняльні характеристики веб-платформ.

Критерії/Назва	Jira	Notion	Asana	Monday	CoSpace
Дизайн і інтерфейс	0	0	1	1	1
Розширені функціональні можливості	1	0	1	0	1
Доступність	0	0	1	0	1
Інтеграції	1	1	0	1	1
Кросплатформеність	1	1	1	1	1
Загальна оцінка	**60%**	**40%**	**80%**	**60%**	**100%**

Після аналізу переваг та недоліків кожного з конкурентів у таблиці, виявлено, що розроблюваний додаток «CoSpace» вище за всіма показниками порівняння. Отже, створення власної веб-платформи є актуальним та обґрунтованим рішенням. Розробка «CoSpace» відповідає найвищим стандартам і враховує потреби та очікування користувачів, що робить її перспективним вибором для подальшого розвитку.

1.3 Аналіз методів розв’язання задачі

Ефективне вирішення поставлених завдань є фундаментом успішної розробки програмного продукту. Результати аналізу методів впливають на його конкурентоспроможність, ресурсозатратність та загальну якість. Різноманіття доступних технологій та бібліотек пропонує безліч альтернативних шляхів досягнення мети, кожен з яких володіє своїми перевагами та обмеженнями.

Підхід до розробки веб-платформи на React, Node.js та PostgreSQL, має ряд суттєвих переваг. По-перше, ці технології пропонують широкий спектр інструментів та бібліотек, які значно спрощують процес розробки та підтримки продукту. Крім того, вони підтримують сучасні підходи, такі як компонентна архітектура, що дозволяє створювати високопродуктивні та легко розширювані додатки. Це, у свою чергу, полегшує інтеграцію з іншими сервісами та системами, що може бути надзвичайно корисним для подальшого розвитку продукту.

Важливою перевагою є можливість побудови масштабованої та ефективної архітектури завдяки цій комбінації технологій. Це гарантує стабільну та надійну роботу веб-платформи навіть при значному обсязі даних та навантаженні. Використання PostgreSQL як бази даних забезпечує безпечне та структуроване зберігання даних, а також високу швидкість операцій та стійкість до непередбачуваних ситуацій.

Важливо підкреслити, що кожен з обраних методів має свої переваги та обмеження, які варто врахувати при прийнятті рішення про вибір. Відповідно до цього аналізу, використання JavaScript, React, Node.js та PostgreSQL виявляється оптимальним рішенням для розробки веб-платформи.

1.4 Постановка задачі розробки веб-платформи для організації коворкінгу

Основна ціль розробки веб-платформи для коворкінгу полягає в створенні унікального та інноваційного онлайн-сервісу. Ця платформа має на меті

забезпечити максимальний комфорт та ефективність у вирішенні різноманітних завдань та організації робочого процесу. Зокрема, вона має надавати можливості:

– створювати та редагувати проекти;
– створювати та редагувати дошки;
– створення та редагування картки;
– переміщення карток;
– редагувати і додавати етапи завдань;
– додавати опис для завдання;
– створення міток;
– встановлення секундоміру;
– коментування завдання;
– додавання вкладень у завдання;
– створювати списки на дошці;
– додавати учасників до проектів та списків з розподіленням прав доступу, включаючи адміністратора та звичайних користувачів;
– індикатор прогресу виконання завдань.

Упродовж розробки потрібно виконати:

– проаналізувати стани платформ для організації коворкінгу;
– провести порівняльний аналіз аналогів;
– проаналізувати технології, СКБД та середовища розробки для веб-платформи;
– розробити модель системи;
– розробити загальний алгоритм роботи веб-платформи;
– розробити структуру та дизайн інтерфейсу користувача;
– розробити модулі керуванням завдань, адміністратором та завантаження вкладень;
– розробити інструкцію користувача;
– провести тестування веб-платформи.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У першому розділі роботи було проведено обґрунтування необхідності розробки веб-платформи для організації коворкінгу. Початковий етап включав аналіз існуючих веб-платформ для коворкінгу, оцінку їх функціональності, ефективності та популярності та проведений порівняльний аналіз аналогів з метою виявлення переваг і недоліків наявних рішень. У результаті аналізу було визначено, що розробка власної веб-платформи для коворкінгу є одночасно доцільним рішенням. Поставлена постановка задачі роботи, в якій були визначені основні функції та вимоги до системи з урахуванням потреб користувачів та сучасних технологій.

Отже, перший розділ надав зрозуміле уявлення про стан сучасного ринку онлайн коворкінгів, виявив недоліки існуючих платформ та визначив ключові завдання для подальшої розробки веб-платформи для коворкінгу.

2 РОЗРОБКА СТРУКТУРИ, МОДЕЛІ ТА АЛГОРИТМІВ ВЕБ-ПЛАТФОРМИ

2.1 Розробка моделі системи

Розробка моделі системи є необхідним етапом у створенні інформаційної системи, вона дозволяє отримати чітке уявлення про те, як система буде працювати та які функції вона буде виконувати, враховуючи всі функції та вимоги. У процесі розробки моделі системи веб-платформи для організації коворкінгу, використовувалася UML діаграма діяльності яка продемонстрована на рисунку 2.1. Згідно вказаним даним з діаграми першим кроком є процес авторизації, де користувач вводить свій логін та пароль.

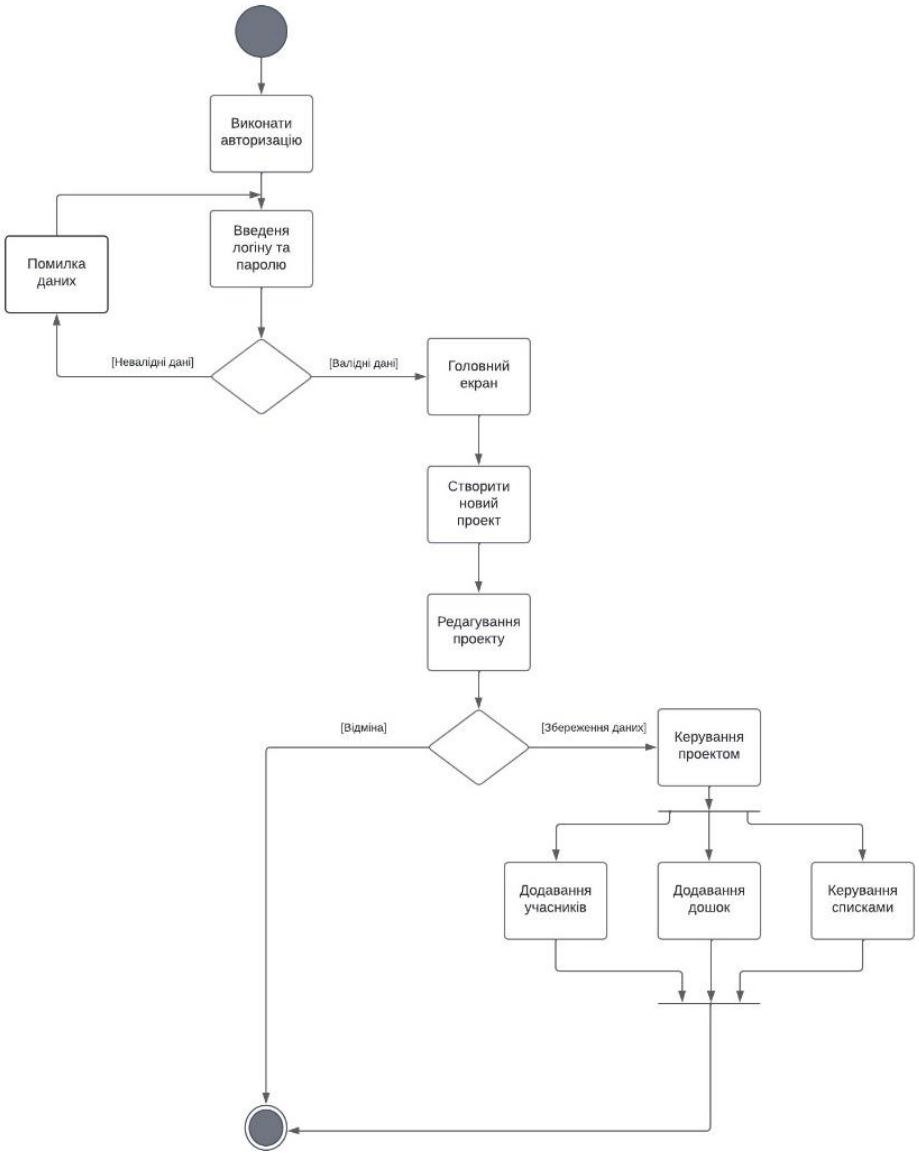

Рисунок 2.1 – Діаграма UML діяльності

Модель системи створює оболонку взаємозв'язків між різними компонентами системи та послідовності дій які виконуються системою або користувачами платформи.

Якщо дані невалідні, система повідомляє про помилку і повертає користувача до початку авторизації. У разі успішної авторизації, користувач попадає на головний екран, де він може створити новий проект. Якщо користувач вирішує не створювати проект, він може відмінити дію, і система не зберігатиме жодних змін. При створенні проекту або редагуванні існуючого, дані зберігаються. Далі, управління проектом поділяється на три функції: додавання учасників, додавання дошок та керування списками. Ці функції дозволяють користувачу організувати та керувати проектом з максимальною ефективністю.

2.2 Архітектура бази даних веб-платформи

Архітектура бази даних грає важливу роль у створенні будь-якої веб-платформи, оскільки вона визначає, як дані будуть зберігатися, організовані та доступні. Вибір відповідної архітектури та технологій бази даних впливає на різні аспекти, такі як продуктивність, масштабованість, безпека та зручність використання платформи. При розробці веб-платформи для організації коворкінгу важливо мати можливість одночасної роботи великої кількості користувачів, які можуть керувати дошками та завданнями проектів. Ця платформа є прикладом розподіленого мережевого застосунку, де сервер відповідає за обробку даних, а клієнтська частина забезпечує інтерфейс для користувачів [7]. Наведена архітектура на рисунку 2.2.

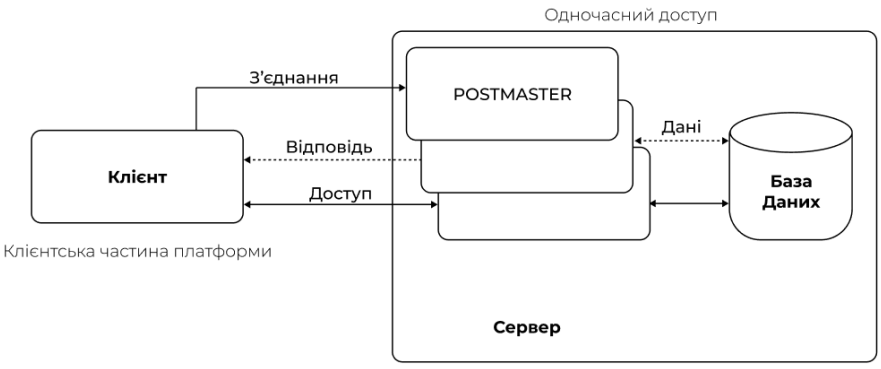

Рисунок 2.2 – Архітектура бази даних клієнт-сервер

У такій архітектурі база даних відіграє ключову роль у зберіганні та управлінні усіма даними платформи. У цій архітектурі, клієнти – це люди, які користуються веб-платформою через свої браузери. Вони надсилають свої запити на сервер, наприклад, для створення нового завдання або оновлення списку проектів. Сервер отримує ці запити і обробляє їх, взаємодіючи з базою даних за потреби, щоб згенерувати відповіді. Ці відповіді потім надсилаються назад користувачам, які їх отримують та бачать на екрані своїх пристроїв.

Така архітектура забезпечує прозорість та зручність для користувачів, оскільки вони можуть легко взаємодіяти з платформою через знайомі інтерфейси. Вона дозволяє розділити роботу між сервером та клієнтами, спрощуючи розробку та підтримку програмного забезпечення. Крім того, ця архітектура забезпечує гнучкість та масштабованість, оскільки клієнтами можуть бути будь-які пристрої, а сервер може бути масштабованим для обробки великого обсягу запитів.

Архітектурні компоненти бази даних визначають структуру та організацію даних у системі. Ці компоненти включають в себе таблиці, відношення між таблицями, індекси, зовнішні ключі та інші елементи, що визначають спосіб зберігання та доступу до даних. Вони є ключовими складовими бази даних і визначають, як дані будуть структуровані та як будуть здійснюватися операції з ними [8]. Архітектурні компоненти наведені в таблиці 2.1.

Таблиця 2.1 – Архітектурні компоненти

Назва	**Опис**
Users	Зберігає інформацію про користувачів платформи.
Projects	Містить дані про проекти, створені користувачами.
Accesses	Зберігає інформацію про права доступу користувачів до проектів.
Tasks	Містить дані про завдання, що входять до складу проектів.

Розглянемо структуру схему бази даних таблиці 2.2, роз'яснюючи призначення кожного поля та вказуючи, як ці дані використовуються у веб-платформі організації коворкінгу. Це дозволить зрозуміти, які конкретно дані

зберігаються в базі даних та як вони впливають на роботу системи та взаємодію з її користувачами.

Таблиця 2.2 – Схема бази даних

Users	**Ім'я поля**	**Коментар**
	Id	Унікальний ідентифікатор користувача
	username	Ім'я користувача
	password_hash	Хеш пароля користувача
	Email	Електронна пошта користувача
	created_at	Дата та час створення запису
Projects	**Ім'я поля**	**Коментар**
	Id	Унікальний ідентифікатор проекту
	Name	Назва проекту
	description	Опис проекту
	created_by	Ідентифікатор користувача, який створив проект
	created_at	Дата та час створення проекту
Accesses	**Ім'я поля**	**Коментар**
	Id	Унікальний ідентифікатор доступу
	user_id	Ідентифікатор користувача
	project_id	Ідентифікатор проекту
	access_level	Рівень доступу користувача до проекту
Tasks	**Ім'я поля**	**Коментар**
	Id	Унікальний ідентифікатор завдання
	project_id	Ідентифікатор проекту, до якого належить завдання
	Title	Назва завдання
	description	Опис завдання
	assigned_to	Ідентифікатор користувача, якому призначено завдання
	Status	Статус завдання
	created_at	Дата та час створення завдання
	updated_at	Дата та час останнього оновлення завдання

Аналізуючи структуру кожної таблиці та її полів, можна з'ясувати, які саме дані зберігаються у базі даних та як вони використовуються для забезпечення роботи платформи. Відомості про користувачів, проекти, доступи та завдання

грамотно організовані у відповідних таблицях, що сприяє зручній та ефективній обробці даних.

2.3 Розробка загального алгоритму веб-платформи

Для розробки веб-платформи важливо спроектувати ефективний та зручний алгоритм роботи (рисунок 2.3), що забезпечить зручність користувачів. При відкритті додатку користувач спочатку потрапляє на головне меню, яке слугує центральною точкою навігації та керування проектами для його команди.

Першим кроком в роботі з додатком є створення проекту. Користувач надає проекту назву, відображаючи його тему або ціль. Після цього користувач створює дошку, яка буде візуальною основою для організації завдань та спільної роботи.

Для початку роботи з проектом необхідно додати учасників. Користувач має можливість запрошувати інших учасників до проекту та надавати їм відповідні ролі доступу для редагування дошки. Це забезпечує контроль над рівнем доступу та забезпечує безпеку даних та конфіденційність інформації.

Важливо пам'ятати, що кожен учасник проекту має свої права доступу, і якщо користувач не має певних прав на редагування, він не зможе виконувати функції, які не доступні йому за його правами доступу. Це забезпечує ефективну спільну роботу та уникнення конфліктів у процесі взаємодії.

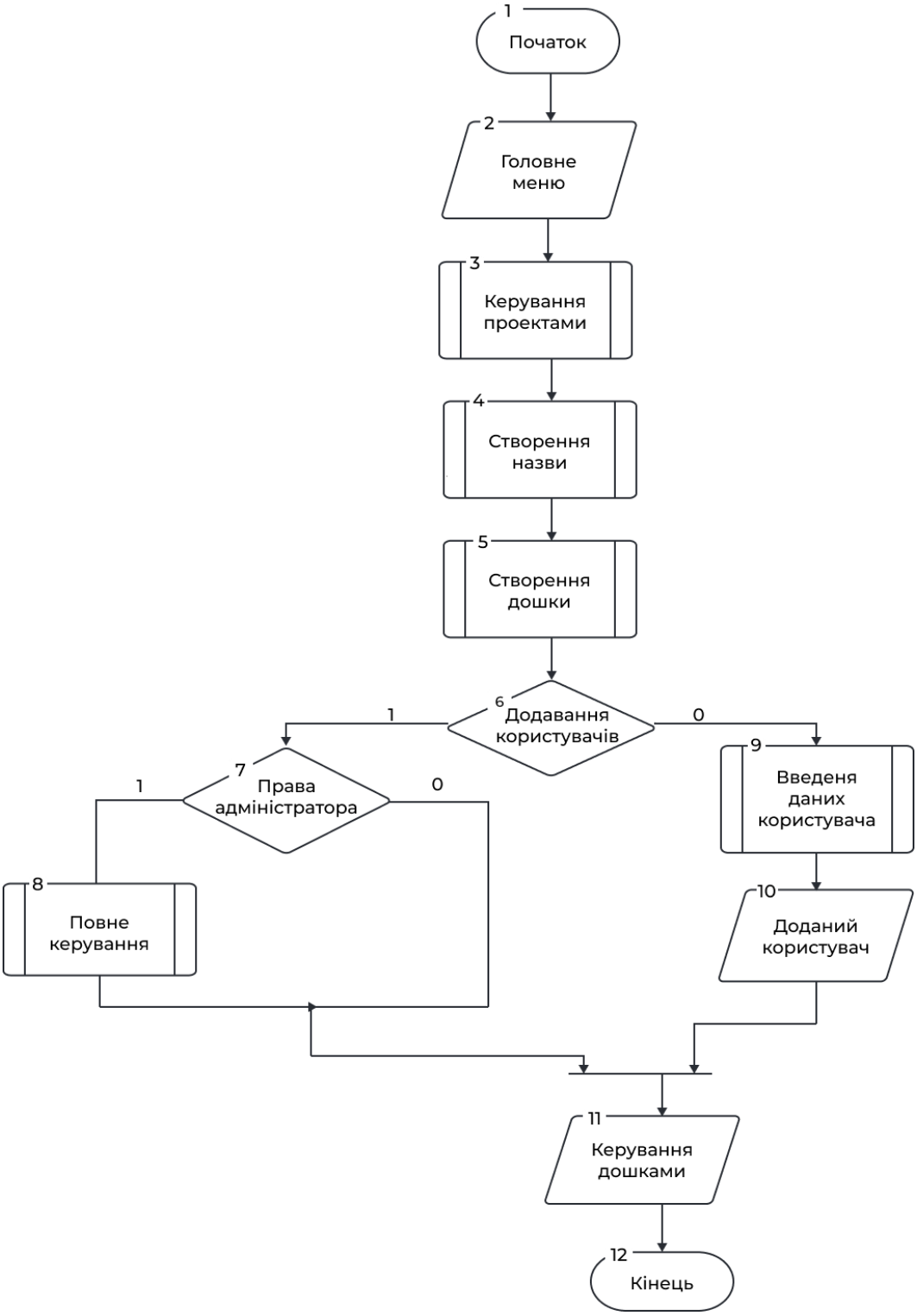

Рисунок 2.3 – Головний екран веб-платформи

Після отримання прав доступу запускається модуль редагування дошки, де відкриваються певні можливості роботи з дошкою та призначення завдань. Для кращого розуміння функціональності керування дошками, було розроблено додатковий алгоритм який продемонстровано на рисунку 2.4.

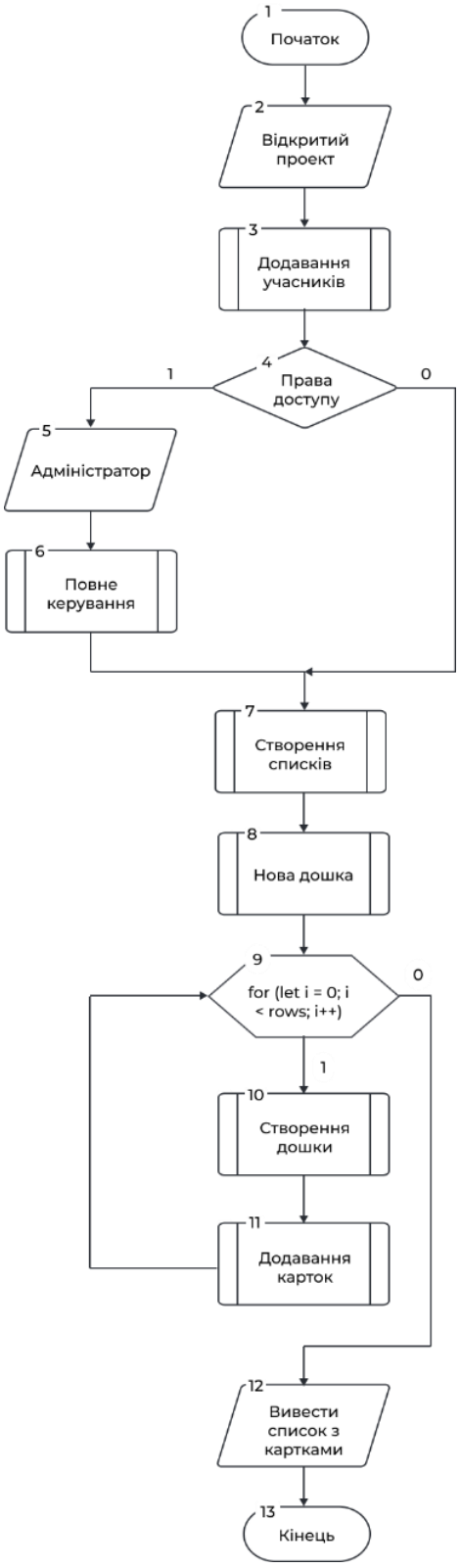

Рисунок 2.4 – Екран керуваннями дошками

Початковим етапом є створення дошки та відповідних списків, а також призначення рівнів доступу для користувачів. Дошок може бути створено довільну кількість для різних проектів без обмежень. Користувачі з правами

адміністратора мають повний контроль над проектом, включаючи можливість видалення проекту та учасників.

Учасники можуть створювати нові списки та додавати до них картки з завданнями. Картки можуть містити інформацію про прогрес виконання завдання, опис завдання, коментарі та прикріплені файли. Кожне завдання позначається відповідальним учасником проекту, і якщо завдання не виконане за пару днів до терміну, система автоматично надсилає термінове повідомлення всім відповідальним за завдання учасникам.

Після аналізу алгоритмів стає зрозумілим, що розроблений додаток надає можливість ефективно керувати проектами. Завдяки цим алгоритмам, система керування проектами стає автоматизованою, що робить розроблений додаток оптимальним вибором для віддалених команд.

2.4 Розробка структури інтерфейсу веб-платформи

Розробка зручного для користувача середовища є основною метою розробки інтерфейсу веб-платформи. Використання технологій бібліотеки React дозволяє побудувати веб-сайти та додатки з привабливими та продуктивними. React допомагає створювати компоненти, які можна повторно використовувати та зручно організовувати. Також React пропонує віртуальну модель DOM, яка дозволяє ефективно оновлювати тільки ті частини веб-сторінки, які змінилися, замість того щоб перезавантажувати всю сторінку. Це робить React надзвичайно потужним інструментом для розробки сучасних веб-додатків.

В більшості випадків користування схожими платформами виникає проблема у великій кількості функцій, що заторможує роботу користувача з системою. Для створення зручної та зрозумілої веб-платформи для користувачів використовується принцип Хіка [9], який стверджує, що час, потрібний для прийняття рішення, збільшується зі зростанням кількості та складності вибору. Тому, було розроблена графічна структура головного екрану яка продемонстрована на рисунку 2.5, де зібрана мінімальна кількість елементів на екрані.

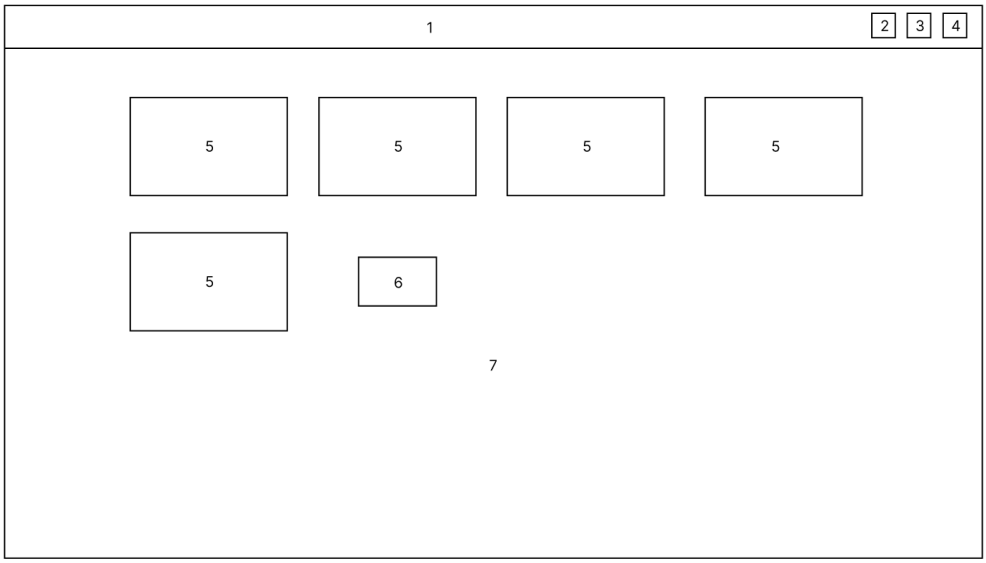

Рисунок 2.5 – Структура головного екрану веб-платформи

На головному екрані користувач може переглядати власні та інших команд проекти, створювати нові. Також з цього екрану можна керувати профілем користувача, сповіщенями та керувати користувачами у системі.

Головний екран повинен містити в собі:

1. Навігаційна панель.
2. Кнопка керування користувачами.
3. Кнопка керування сповіщень.
4. Кнопка керування профілю користувача.
5. Кнопка переходу на проект.
6. Кнопка додавання нового проекту.
7. Робоча область.

Після початку роботи з проектом, користувач попадає на екран керуванням дошки проекта, що зображена на рисунку 2.6. Цей екран створений для зручного управління процесом роботи над проектом та спільної колаборації команди. Цей екран є центральним елементом веб-платформи, де користувач може керувати всіма аспектами свого проекту. Для використання можливостей платформи, перше, що потрібно зробити, створити нову дошку у проекті та надати їй назву.

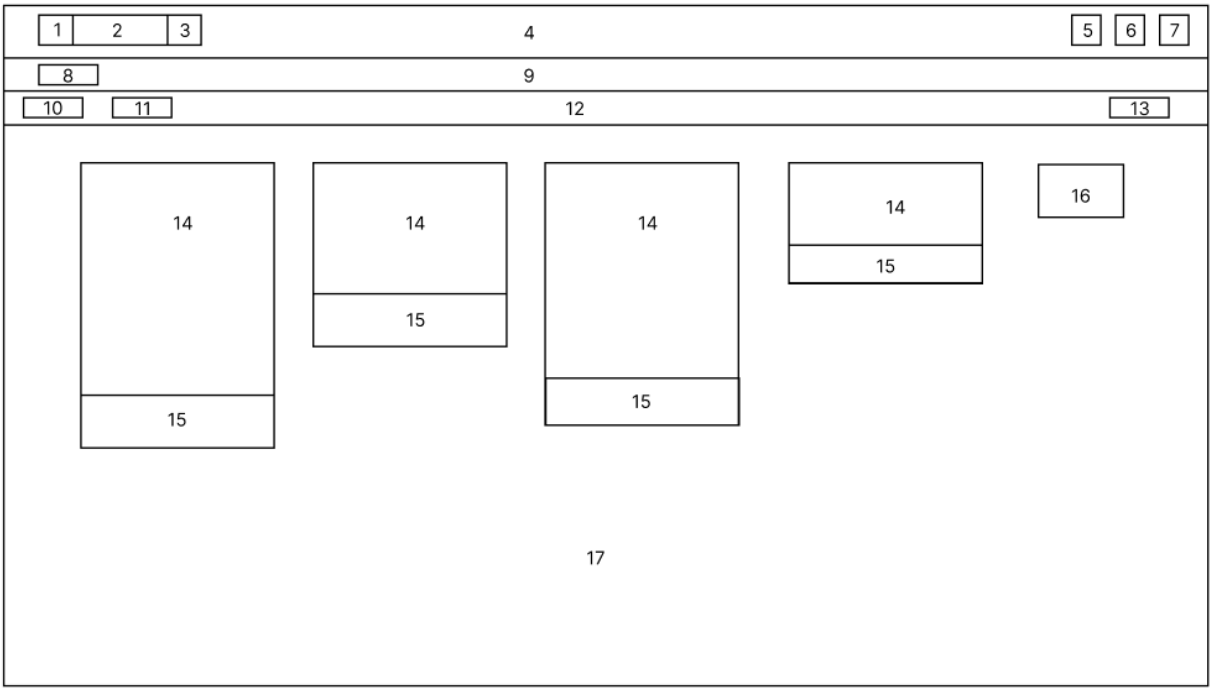

Рисунок 2.6 – Структурна схема керуванням дошки проекту

Після створення дошки, становляться доступні функції для створення списків з дошками, де будуть створюватись картки та призначатись завдання для команди.

Керування дошкою повинно містити в собі:

1. Кнопка назад.
2. Назва.
3. Зміни назви.
4. Навігаційна панель.
5. Кнопка керування користувачами.
6. Кнопка керування сповіщень.
7. Кнопка керування профілю користувача.
8. Дошка.
9. Панель вкладок.
10. Фільтрування учасників.
11. Мітки.
12. Навігаційна панель користувачів.
13. Список учасників.

14. Список карток.

15. Додати нову картку.

16. Додати новий список.

17. Робоча область.

Модуль керування завданнями є важливою складовою функціональності системи управління проектами. Його основна мета – спростити процес створення, оновлення та відстеження завдань для всіх учасників проекту. Після того, як користувач створив нову дошку наступним етапом є створення списків, де будуть розміщуватись картки із завданнями. При створенні нового завдання, відкривається модуль керування завданнями (рисунок 2.7), де користувач може заповнити необхідну інформацію для інших учасників проекту. Модуль керування містить в собі всі необхідні та головні функції для зручності користування та оформлення завдань.

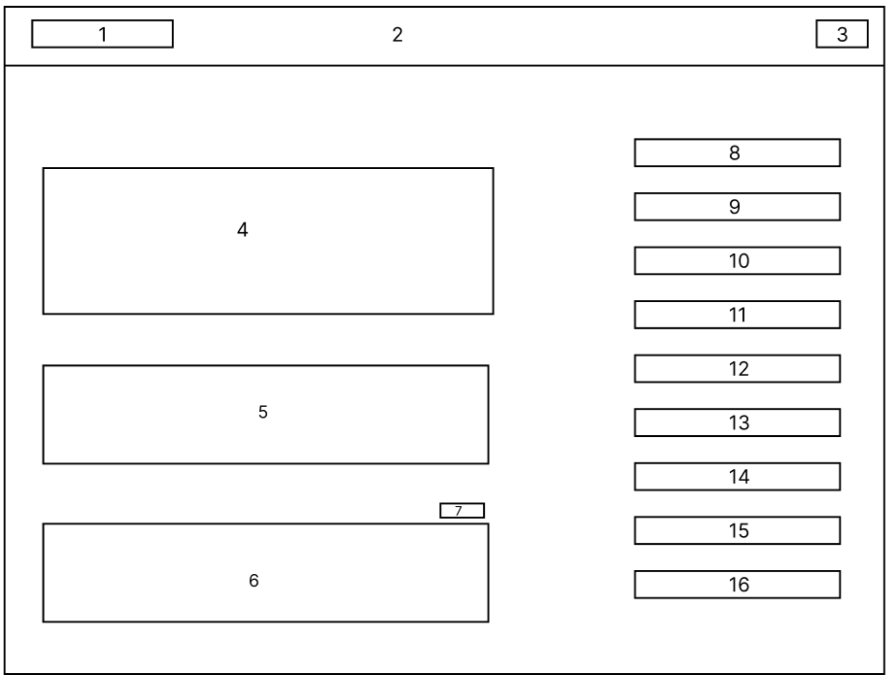

Рисунок 2.7 – Структурна схема керуванням завдань

Модуль керуванням завданням має містити в собі:

1. Назва.

2. Навігаційна панель вікна.

3. Кнопка закрити.

4. Опис.
5. Чек ліст.
6. Останні дії та коментарі.
7. Показати деталі.
8. Учасники.
9. Мітки.
10. Термін виконання завдання.
11. Таймер.
12. Вкладення.
13. Відстеження завдання.
14. Переміщення завдання.
15. Продублювати завдання.
16. Видалити завдання.

2.5 Висновки

В другому розділі було проаналізувано архітектуру бази даних веб-платформи, розроблено алгоритми роботи платформи та модель систем з використанням UML діаграм. Розроблено структурно зручний інтерфейс для користувачів та створено загальний алгоритм функціонування веб-платформи, що дозволяє ефективно керувати проектами та автоматизувати процеси управління ними.

3 РОЗРОБКА МОДУЛІВ ВЕБ-ПЛАТФОРМИ

3.1 Варіантний аналіз і обґрунтування вибору мови програмування

У сучасному світі вибір мов програмування для розробки програмного забезпечення або веб-додатків вирішується на основі різноманітних факторів, включаючи ефективність, продуктивність, екосистему і популярність серед розробників. При розгляді вибору мов для створення веб-додатків часто враховуються можливості розробки як на серверній, так і на клієнтській сторонах, а також зручність у взаємодії з базами даних і іншими сервісами.

Порівняння мов програмування, які часто використовуються для розробки програмного забезпечення, може бути складним завданням, оскільки кожна мова має свої унікальні особливості і призначення. Для цього було проведено аналіз, щоб отримати найкращий варіант для розробки веб-платформи.

JavaScript – полегшена інтерпретована мова програмування з першокласними функціями [10]. JavaScript є основною мовою програмування для створення інтерактивних веб-сайтів та додатків. Він використовується для взаємодії з користувачем на стороні клієнта, а також для створення багатьох серверних додатків. Є однією з найпопулярніших мов програмування, особливо в контексті веб-розробки. Має велику кількість бібліотек і фреймворків, таких як React, Angular і Vue.js.

Плюси:

– можна використовувати як на клієнтській, так і на серверній стороні, завдяки платформі Node.js;

– інтерактивність;

– велика кількість бібліотек та фреймворків.

Мінуси:

– має свої особливості та нюанси, які можуть призвести до виникнення проблем, особливо для початківців;

– різні браузери можуть різнитися у виконанні певних функцій;

– може бути менш ефективним за інші мови, особливо у великих програмах.

PHP, що скорочено від "PHP: Hypertext Preprocessor", є однією з найпопулярніших мов програмування для веб-розробки [11]. Вона була спеціально розроблена для створення динамічних веб-сторінок та розширення можливостей HTML. Одним з головних переваг PHP є його інтеграція з великою кількістю баз даних, таких як MySQL, PostgreSQL, SQLite і багатьма іншими. Це робить PHP популярним вибором для розробки веб-додатків з розширеними функціональними можливостями, такими як інтернет-магазини, блоги, форуми тощо.

Плюси:

– існує велика кількість ресурсів, документації та форумів, де можна знайти допомогу та поради в разі проблем;

– дозволяє швидко створювати веб-додатки.

Мінуси:

– проблеми з безпекою;

– менш ефективний у виконанні деяких завдань;

– не гнучкий у деяких аспектах розробки.

Rust вирізняється своєю високою ефективністю та безпекою [12]. Вона була створена з метою заміни C++ як мови програмування для системного програмування, проте з меншим ризиком виникнення помилок і безпеки. Rust надає розробникам високий рівень контролю над пам'яттю, використовуючи систему власності та інші механізми безпеки. Це робить її популярним вибором для розробки безпечних та ефективних системних програм, таких як операційні системи, вбудовані програми та додатки реального часу.

Плюси:

– має вбудовані механізми безпеки, що дозволяють уникнути багатьох типів помилок, таких як переповнення буфера чи витік пам'яті;

– дозволяє розробникам вибирати рівень контролю над пам'яттю та іншими аспектами програми.

Мінуси:

– складнішим для початківці;

– розробка програм на Rust може займати більше часу.

Python – мова програмування, яка володіє простим синтаксисом та великою кількістю бібліотек [13]. Вона широко використовується у багатьох галузях, включаючи веб-розробку, наукові обчислення, штучний інтелект, машинне навчання, обробку даних та інші. Python є інтерпретованою мовою, що означає, що код виконується рядок за рядком без попередньої компіляції. Це робить його дуже дружнім до початківців і швидким для розробки прототипів.

Плюси:

– має простий і зрозумілий синтаксис;

– має велику кількість бібліотек;

– можуть запускатися на різних операційних системах без змін.

Мінуси:

– менш ефективний у виконанні обчислювально-інтенсивних програмах;

– Python використовує глобальний інтерпретатор, що може призводити до певних проблем з управлінням залежностями та версіями бібліотек;

– можуть потребувати більше пам'яті для виконання.

Розглянемо таблицю 3.1, яка містить порівняння різних мов програмування. Для оцінки найкращого вибору для розробки веб-платформи.

Таблиця 3.1 – Порівняння мов програмування

Критерій	JavaScript	Python	PHP	Rust
Екосистема	5	4	3	4
Швидкодія	3	4	5	3
Безпека	3	4	5	3
Легкість вивчення та використання	5	4	2	5

Продовження таблиці 3.1.

Критерій	JavaScript	Python	PHP	Rust
Розширюваність	5	3	5	4
Оцінка	**86.67%**	**76.67%**	**75%**	**78.33%**

На основі наданих критеріїв і їхніх оцінок, JavaScript справді виглядає як дуже сильний варіант. Він отримав високі бали в таких ключових аспектах, як екосистема, легкість вивчення та використання, і розширюваність. Хоча він може поступатися іншим мовам у швидкодії та безпеці, але в цілому, для багатьох проектів, це компенсується його широким застосуванням і підтримкою.

3.2 Вибір середовища розробки та СКБД

Для початку роботи розглянуто чотири популярних середовища розробки: Visual Studio Code (VSCode), Sublime Text, Atom та WebStorm, де проаналізовано їхні можливості та особливості в контексті створення програмного забезпечення на мові JavaScript.

Sublime Text [14], створений Джоном Спіннером у 2011 році на базі мов програмування C++ та Python, є кросплатформним редактором веб-коду. Із зручним інтерфейсом (рисунок 3.1). Остання версія програми вийшла у 2013 році, але й нині вона залишається популярним інструментом для розробки.

Рисунок 3.1 – Інтерфейс Sublime Text

Переваги Sublime Text включають багатомовність та зручний веб-інтерфейс, низьку вимогливість до ресурсів комп'ютера та можливість розширення функціоналу за допомогою плагінів та тем. Також варто відзначити його функції підсвічування синтаксису для різних мов програмування, автоматичне вставлення коду та підтримку макросів. Sublime Text є комерційним продуктом, доступним за платною ліцензією, і встановлюється на різних операційних системах, включаючи Windows, Linux та MacOS.

Atom – це сучасний текстовий редактор (рисунок 3.2), який відкриває нову еру у розробці програмного забезпечення [15]. Створений на основі Electron і заснований на найкращих традиціях улюблених редакторів, Atom пропонує глибокі можливості настройки, при цьому залишаючись доступним за замовчуванням.

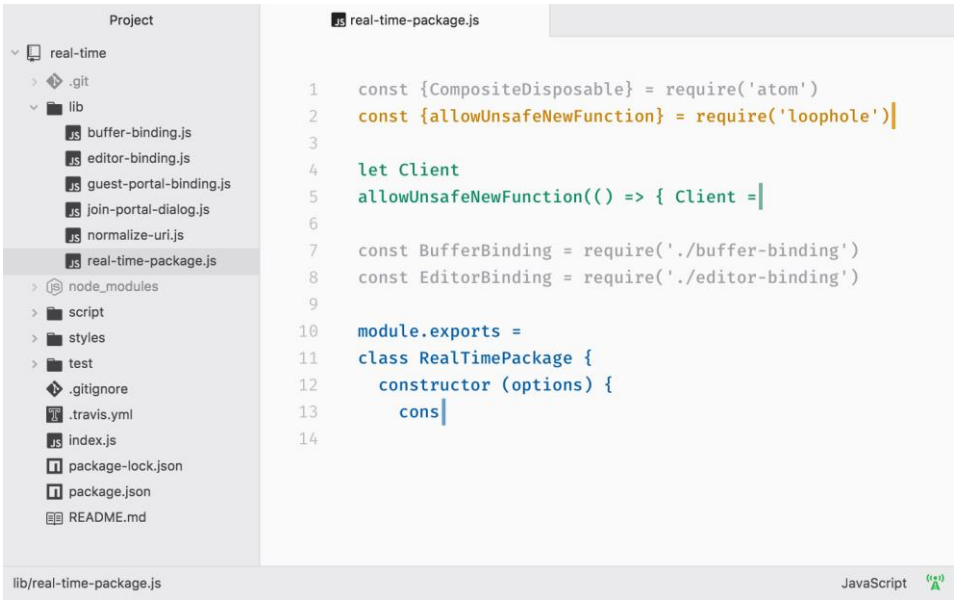

Рисунок 3.2 – Приклад інтерфейсу Atom

Цей редактор відкриває нові можливості для розробки програмного забезпечення, дозволяючи розробникам створювати зручні та потужні інструменти за своїми уподобаннями. Atom підтримує різні операційні системи, включаючи OS X, Windows та Linux, забезпечуючи при цьому однакову ефективність та зручність. Вбудований менеджер пакетів дозволяє знайти та встановити нові пакунки або створити власні прямо з Atom. Розумне

автозаповнення спрощує процес написання коду, а браузер файлової системи дозволяє легко переглядати та відкривати файли та проекти. Atom працює з інтеграціями HTML, JavaScript, CSS і Node.js, використовуючи платформу Electron для створення міжплатформних додатків.

JetBrains WebStorm – це інтегроване середовище розробки (рисунок 3.3), розроблене компанією JetBrains, призначене для роботи з JavaScript, HTML та CSS [16]. Воно базується на платформі IntelliJ IDEA і є спеціалізованою версією PhpStorm, яка включає підмножину його можливостей.

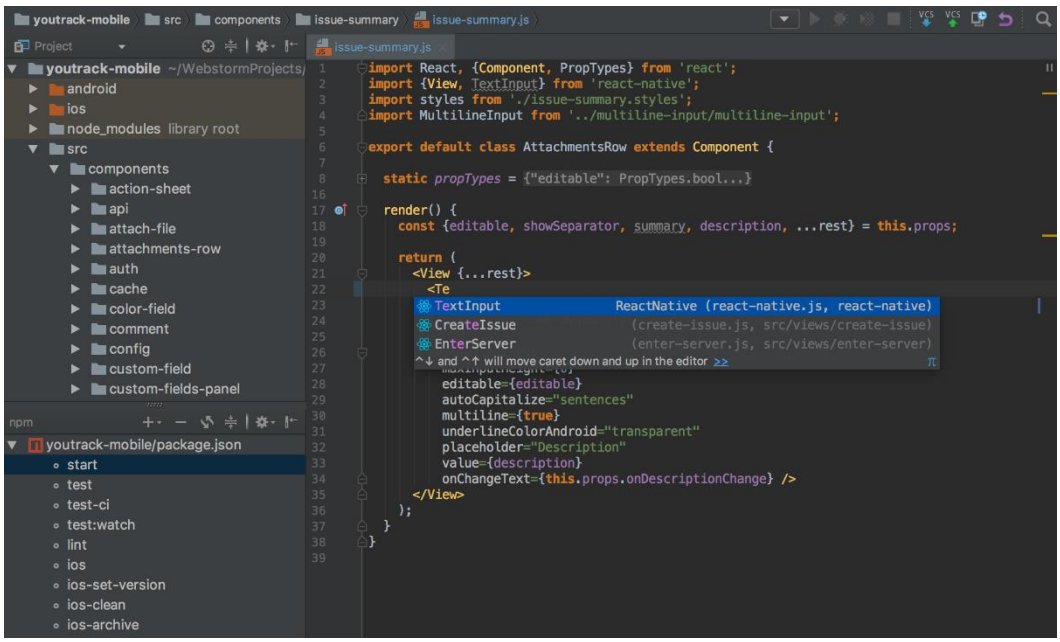

Рисунок 3.3 – WebStorm приклад інтерфейсу

Однією з переваг WebStorm є наявність передвстановлених плагінів для JavaScript, таких як для Node.js, які доступні безкоштовно для користувачів PhpStorm. Також WebStorm має активну спільноту користувачів і підтримку від компанії JetBrains, що забезпечує регулярні оновлення та вдосконалення продукту.

Проте, серед його недоліків слід зазначити вартість, оскільки WebStorm є комерційним продуктом і вимагає платну ліцензію для використання. Це може стати проблемою для окремих користувачів або команд з обмеженим бюджетом. Крім того, через високі вимоги до системних ресурсів, WebStorm може виявитися важким для роботи на менш потужних комп'ютерах.

Visual Studio Code (VS Code) [17], представляє собою безкоштовний редактор (рисунок 3.4) з відкритим вихідним кодом, розроблений компанією Microsoft. Він доступний для операційних систем Windows, Linux та macOS. Незважаючи на свою легкість, VS Code має вражаючі можливості, які зробили його одним з найбільш популярних інструментів для розробки програмного забезпечення останнім часом.

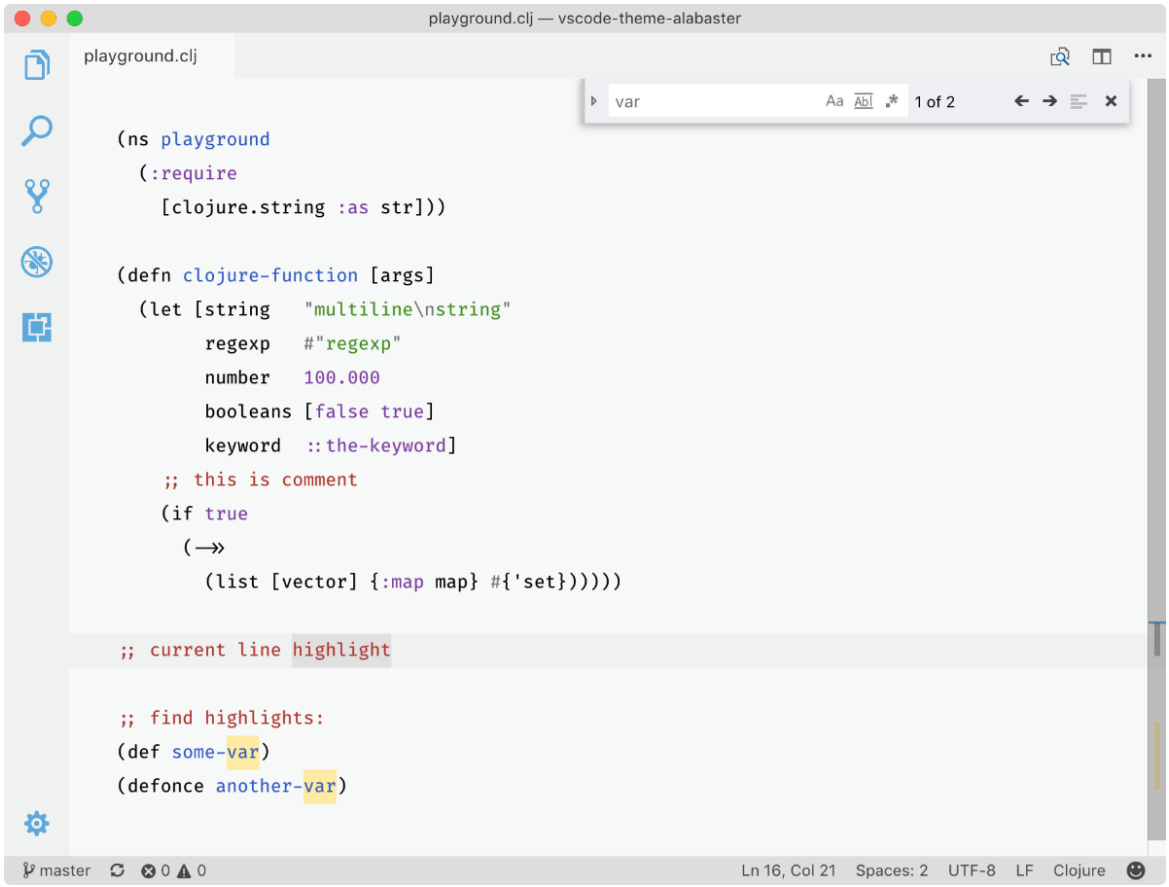

Рисунок 3.4 – Приклад інтерфейсу VSCode

Однією з великих переваг VS Code є його безкоштовність та відкритий код, що робить його доступним для всіх бажаючих без обмежень. Крім того, редактор підтримується на усіх ОС, і це робить його доступним для великого кола користувачів, а серед недоліків можна відзначити великість ресурсів, які можуть бути необхідні для роботи.

У таблиці 3.2 продемонстровано порівняння всіх середовищ розробки таких як: Visual Studio Code, SublimeText, Atom та WebStorm.

Таблиця 3.2 – Порівняння середовищ розробки

Критерій	Sublime Text	Atom	WebStorm	VSCode
Безкоштовність	4	4	3	5
Функціональність	4	4	5	5
Підтримка розширень	4	5	5	5
Кросплатформенність	4	5	5	5
Швидкодія	4	4	4	5
Оцінка	**80%**	**88%**	**88%**	**100%**

Отже, після аналізу різних середовищ розробки з таблиці 3.2, було прийнято рішення працювати з Visual Studio Code.

Для зберігання даних користувачів, файлів у системі потрібно мати свою базу даних, точніше цілу її систему яка буде автоматизована для роботи з даними які завантажуються самими користувачами. База даних (БД) – це система призначена щоб зберігати, оновляти та обробляти значі обсяги взаємопов'язаної інформації. Вони є невід'ємною складовою динамічних веб-сайтів з великою кількістю даних, наприклад таких як: електронні магазини, інформаційні портали та корпоративні ресурси.

Система управління базами даних (СКБД) – це набір програмних засобів призначених для розробки, модифікації та адміністрування баз даних. СКБД забезпечує засоби для формування структури БД, її наповнення даними, внесення змін до вмісту та взаємодії з інформацією. Серед найпопулярніших СКБД у веб-розробці можна виділити MySQL, PostgreSQL, Oracle та MongoDB [18].

Вибір оптимальної СКБД є ключовим етапом у процесі створення будь-якого інтерактивного веб-додатку. Розглянемо три поширені варіанти СКБД, кожен з яких має свої переваги та особливості:

1. PostgreSQL – перевірена часом реляційна СКБД з відкритим вихідним кодом [19] вирізняється архітектурою, що забезпечує високу цілісність даних, розширені можливості масштабування та підтримку найрізноманітніших

функцій, включаючи складні запити, надійні транзакції та зберігання спеціалізованих типів даних, таких як геопросторові. Завдяки активній спільноті розробників PostgreSQL постійно розвивається та вдосконалюється, що робить її ідеальним вибором для проектів, де на перший план виходять висока продуктивність, адаптивність до зростаючих навантажень та можливість тонкого налаштування під специфічні вимоги.

2. MySQL вирізняється інтуїтивним інтерфейсом, оперативністю та відкритістю. Її переваги найкраще розкриваються в невеликих та середніх проектах, особливо тих, де використовується PHP [20]. MySQL надає розробникам широкий арсенал інструментів для ефективного управління та адміністрування даних, що робить її привабливим вибором для багатьох веб-додатків.

3. MongoDB – NoSQL-орієнтована система управління базами даних, що працює з гнучкою, документо-орієнтованою моделлю даних. MongoDB [21] є оптимальним рішенням для проектів, де структура даних може динамічно змінюватися, або для тих, що потребують високої швидкодії запитів та горизонтальної масштабованості. Завдяки своїй гнучкості та продуктивності, MongoDB знайшла широке застосування у сфері веб-розробки, мобільних додатків та аналітичних систем.

Для оцінки всіх розглянутих СКБД та порівняння їх за ключовими параметрами була розроблена таблиця 3.3, що містить оцінки кожної системи за визначеними критеріями.

Таблиця 3.3 – Порівняння вибіру СКБД

Критерій	MySQL	MongoDB	PostgreSQL
Надійність та цілісність даних	4	3	5
Масштабованість та продуктивність	4	4	5
Функціональність та підтримка різних типів даних	3	3	5

Продовження таблиці 3.3.

Критерій	MySQL	MongoDB	PostgreSQL
Простота використання та адміністрування	4	4	5
Оцінка	**88.24%**	**70%**	**100%**

Отже, можна стверджувати, що PostgreSQL демонструє перевагу за більшістю ключових параметрів, що робить її оптимальним рішенням для проектів, де пріоритетними є висока надійність, здатність до масштабування, продуктивність та широкий спектр функціональних можливостей.

3.3 Розробка методу управління завданнями проекта

Головна задача веб-платформи для організації коворкінгу полягає в створенні та ефективному відстеженні завдань у проекті на дошці. Одним із ключових етапів цього процесу є створення нового завдання у карточці. Під час цього процесу (рисунок 3.5), система отримує дані про нове завдання, перевіряє їх коректність. Потім визначає місце завдання на картці, оновлюючи позиції інших завдань, якщо потрібно. Після цього створює новий запис завдання в базі даних та сповіщає всіх користувачів про зміни.

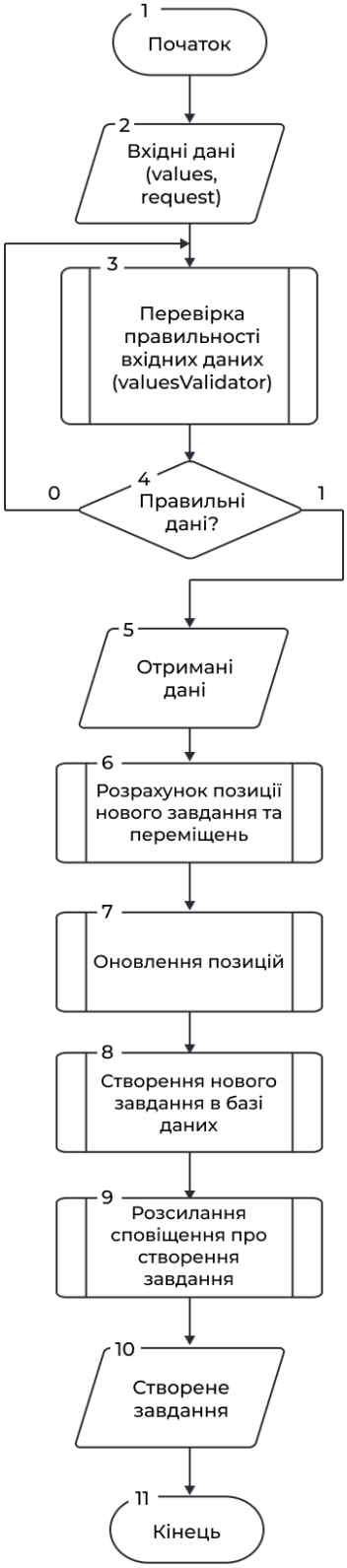

Рисунок 3.5 – CreateTask

Після цього карточка постійно оновлюється, щоб відображати актуальну інформацію про завдання (рисунок 3.6), також оновлює порядок завдань на

дошці, перераховує позиції інших завдань після переміщення одного, зберігає зміни в базі даних та повідомляє користувачів про оновлення.

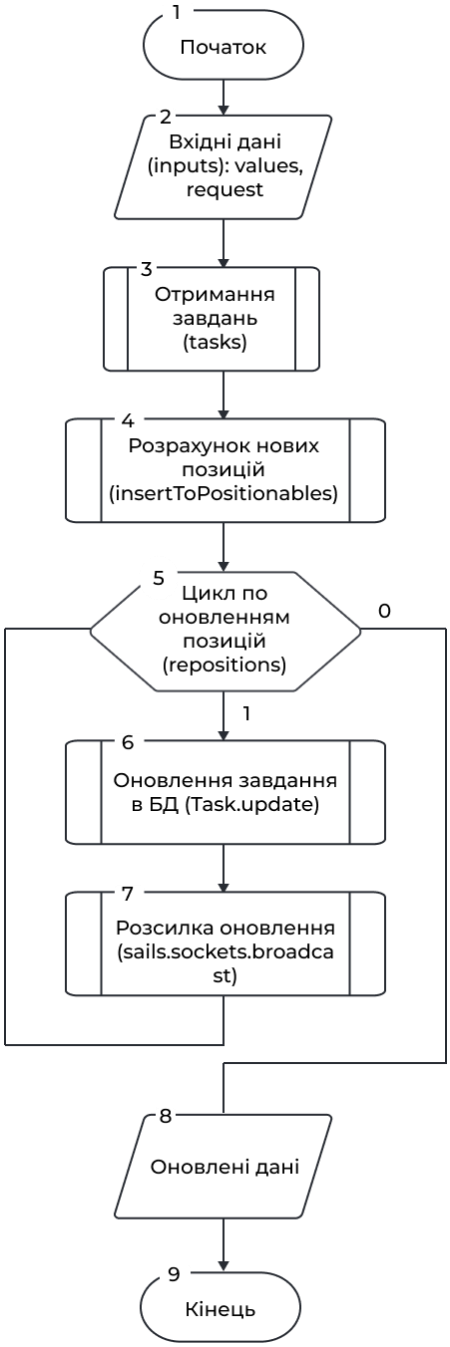

Рисунок 3.6 – UpdateTask

Додатково, для зручності користувача та оптимізації робочого процесу, можливо переміщення завдань у карточці на нову позицію. Це досягається за допомогою методу moveTask, переміщує завдання в списку завдань. Він визначає, в якому списку знаходиться завдання, обчислює нову позицію завдання в цьому списку, а потім оновлює позицію завдання (рисунок 3.7).

У випадку необхідності видалення карточки (рисунок 3.8), система також забезпечує відповідний функціонал. Спочатку завдання видаляється візуально, потім відбувається спроба видалити його з сервера. Якщо видалення успішне, завдання видаляється з сховища даних. Якщо ні, відображається повідомлення про помилку. На завершення, відбувається додаткова обробка видалення завдання.

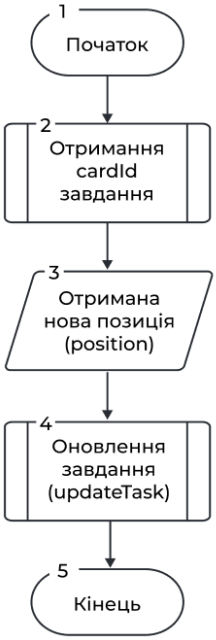

Рисунок 3.7 – MoveTask

Рисунок 3.8 – DeleteTask

3.4 Розробка методу адміністрування користувачів

Основною задачею модуля керування адміністрування є забезпечення доступу до додаткових можливостей взаємодії з учасниками та проектом на дошці веб-платформи для організації коворкінгу. За допомогою цього модуля адміністратори можуть ефективно керувати проектом та його учасниками, забезпечуючи належний рівень організації та співпраці (рисунок 3.9). Цей модуль використовується для перевірки автентифікації користувача перед виконанням певної дії або доступом до певного ресурсу. Основна логіка полягає в перевірці, чи має поточний користувач права адміністратора. Якщо користувач не є адміністратором, функція повертає відповідь зі статусом 404, що сигналізує про відсутність доступу до ресурсу. У випадку, якщо користувач є адміністратором, функція передає управління наступній операції за допомогою функції proceed(), що дозволяє продовжити виконання запиту або доступ до ресурсу. Ця функція реалізує базовий механізм контролю доступу, дозволяючи обмежити доступ до певних функцій або ресурсів лише адміністраторам.

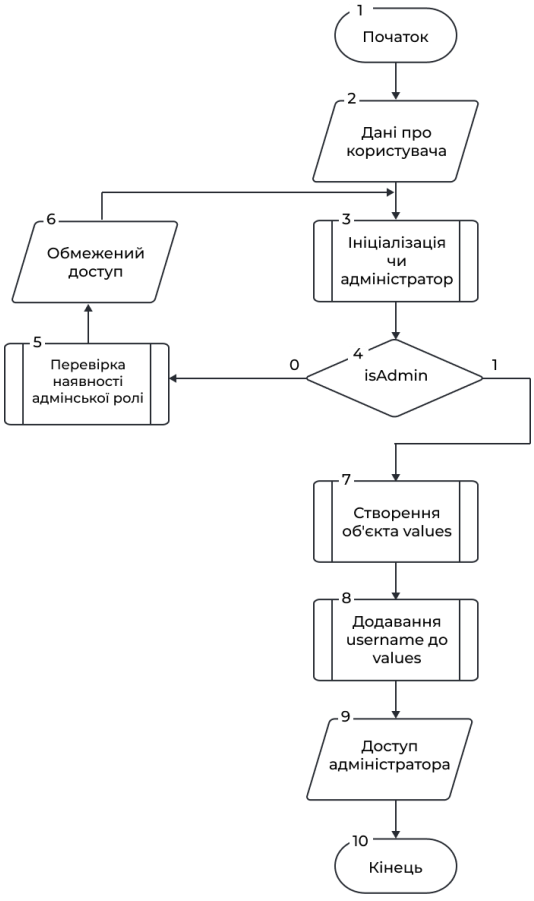

Рисунок 3.9 – isAdmin

Функція createManagerInCurrentProject (рисунок 3.10) дозволяє створювати нових менеджерів проекту для поточного проекту, надаючи їм відповідні повноваження та доступ до необхідних ресурсів. Функція приймає дані нового менеджера та створює спеціальне повідомлення для додавання його до поточного проекту.

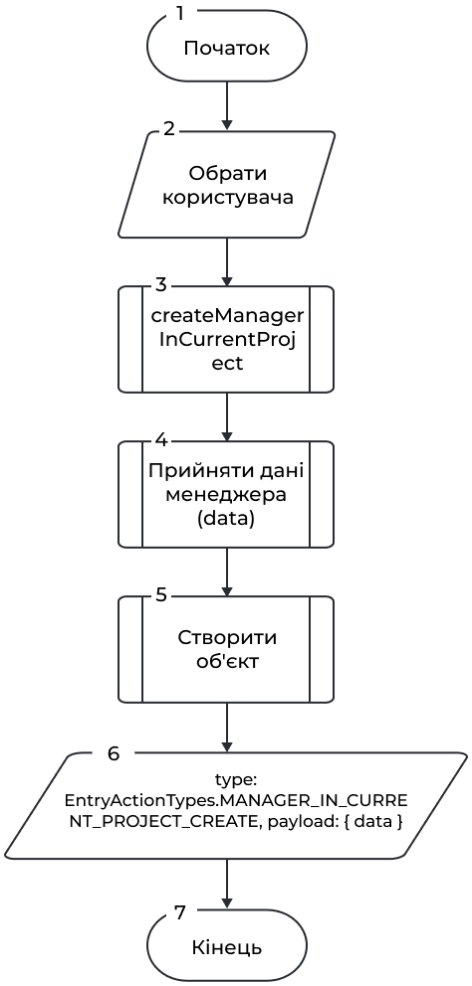

Рисунок 3.10 – CreateManager

3.5 Розробка модуля завантаження вкладень

Цей модуль реалізує важливий функціонал веб-додатка, пов'язаний із завантаженням файлів (рисунок 3.11). Коли користувач намагається завантажити файл, контролер перевіряє його права доступу до відповідної карти в системі, враховуючи його роль у дошці. Якщо права доступу не вистачає або карта не знайдена, генерується відповідна помилка. У разі успішної перевірки, файл завантажується та обробляється. Після обробки він зберігається у системі, а

потім створюється нове вкладення, яке асоціюється з цією карткою. У випадку успішного завершення операції, контролер повертає результат, що містить дані про нове вкладення для подальшої обробки. Якщо ж у процесі виникли проблеми, такі як невдале завантаження або відсутність файлу, генерується відповідна помилка, яка повертається для подальшої обробки.

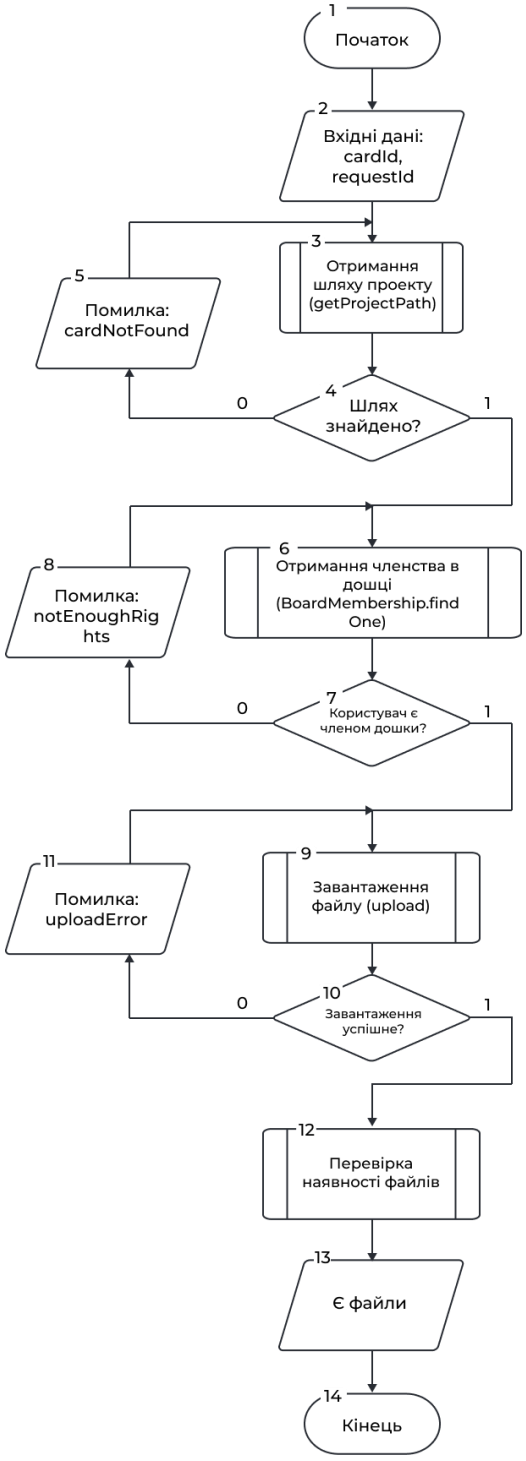

Рисунок 3.11 – Модуль завантаження вкладень

3.6 Висновки

У третьому розділі було проведено аналіз та обґрунтування вибору мови програмування для розробки веб-платформи для організації коворкінгу. Було прийнято рішення використовувати JavaScript як основну мову програмування, що забезпечує динамічність та інтерактивність інтерфейсу. Для підвищення ефективності розробки обрано середовище VS Code, яке надає широкий спектр інструментів та можливостей для роботи з кодом.

У процесі розробки веб-платформи було реалізовано комплекс ключових модулів, спрямованих на забезпечення її функціональності та зручності використання. Модуль керування завданнями дозволяє користувачам ефективно організовувати свою роботу, створюючи, редагуючи, призначаючи та відстежуючи виконання завдань. Модуль керування адміністраторами надає розширені можливості для налаштування платформи та управління користувачами, забезпечуючи контроль над її роботою. Модуль завантаження вкладень сприяє ефективній співпраці, надаючи користувачам зручні інструменти для обміну файлами та спільної роботи над документами.

4 ТЕСТУВАННЯ ВЕБ-ПЛАТФОРМИ

4.1 Тестування системи

Тестування веб-платформи на операційних системах Windows та macOS є важливим етапом для забезпечення перевірки її якості та надійності. Цей процес передбачає оцінку функціональності, безпеки та коректності роботи платформи в широкому діапазоні веб-браузерів.

Перевірка сумісності програми з різними версіями операційної системи є ключовим етапом тестування, оскільки у користувачів можуть бути різні версії ОС Windows, або macOS. Наявність стабільної роботи програми на всіх платформах є важливою. Проведення тестів включає оцінку реакції платформи на різні роздільні здатності екрану, адаптивний інтерфейс і перевірку швидкості завантаження сторінок. Цей процес тестування виконується на тестовій версії програми та на сервері хостингу.

Далі, проводяться тести на безпеку програми, включаючи перевірку на проникнення, збереження конфіденційної інформації та злам програми. Для цього можуть використовуватись спеціальні програмні засоби, які дозволяють моделювати різні атаки на програму.

Після успішного завершення всіх тестів, проводиться оцінка продуктивності програми. Це включає тести на швидкість роботи програми, використання ресурсів та аналіз життєвого циклу програми.

Результати тестування фіксуються у документацію, які аналізуються, щоб вдосконалити програму та гарантувати забезпечення якості її роботи для браузерів Google Chrome, Mozilla Firefox, Safari та Microsoft Edge на платформах Windows і macOS.

Результат тестування продемонстровано на рисунку 4.1.

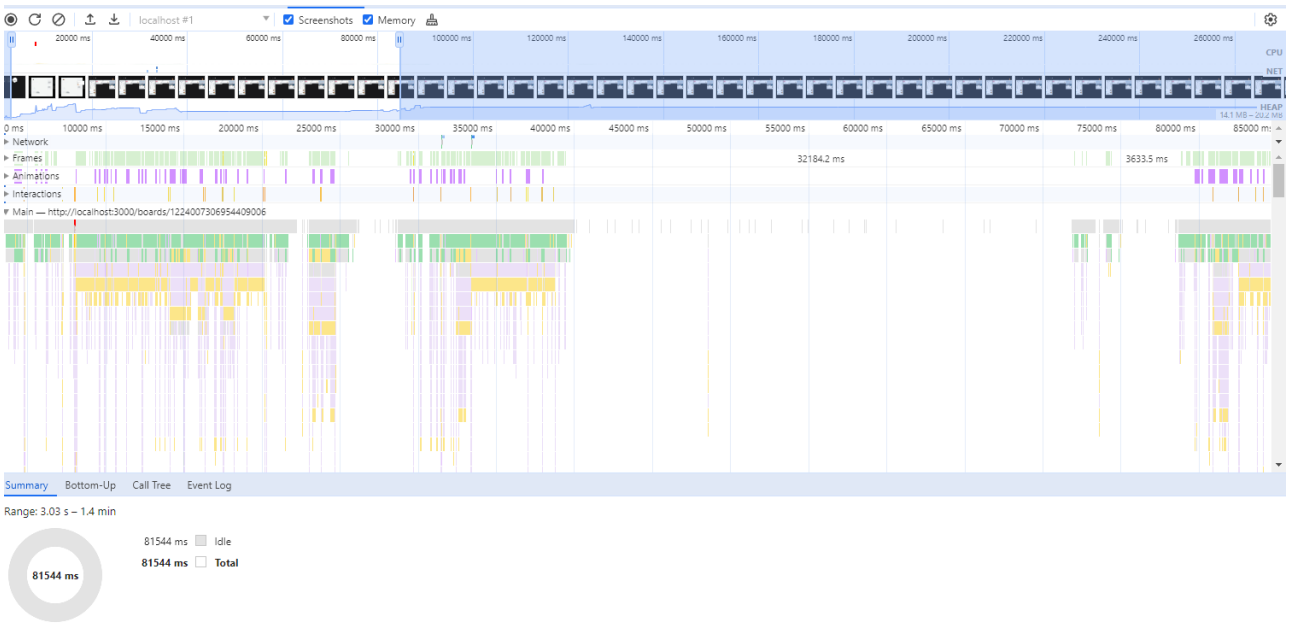

Рисунок 4.1 – Тестування продуктивності системи

Наступним кроком буде тестування екрану входу до системи. Під час цього тестування перевіряється реакція системи на неправильні вхідні дані користувача. У разі спроби реєстрації без інформації про ім'я або пароля, система запропонує ввести дані підсвічуванням, оскільки це є обов'язковою інформацією для подальшої роботи користувача. Результат тестування зображено на рисунку 4.2.

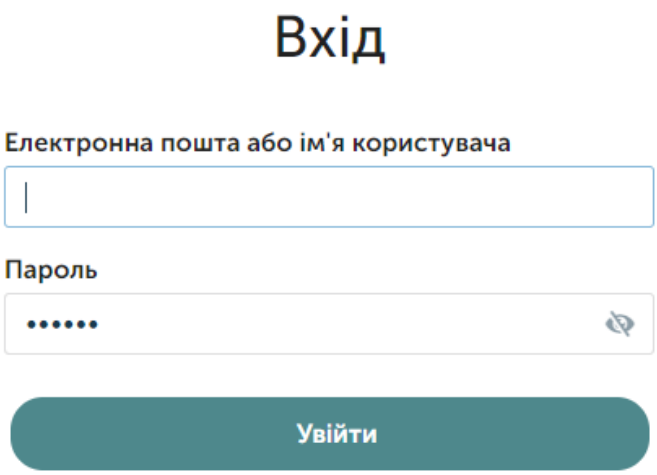

Рисунок 4.2 – Реєстрація користувача без даних

Перевірка на правильність введених даних є однією з контрольних точок. Якщо дані виявляються некоректними, система блокує доступ користувача та видає повідомлення про помилку.

Результати тестування продемонстровано на рисунку 4.3.

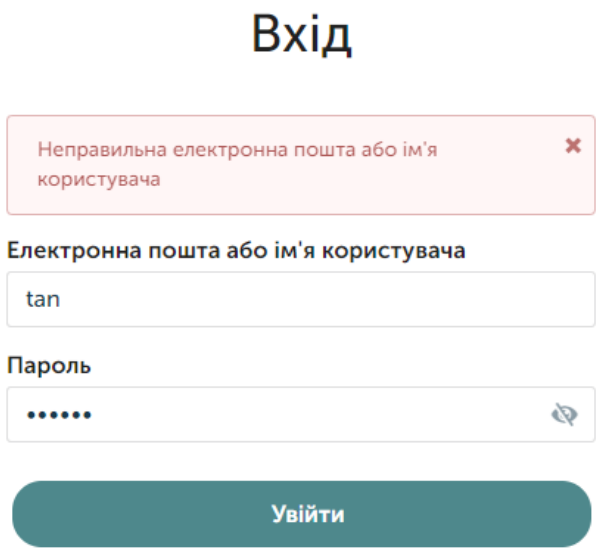

Рисунок 4.3 – Вхід з невірними даними користувача

Після перевірки введення недійсних даних, ми спробуємо правильно заповнити всі поля і перевірити, чи зареєструє система користувача. Після успішної перевірки система перенаправила користувача на сторінку головного меню, що свідчить про успішну реєстрацію користувача. Результати цього тестування показані на рисунку 4.4.

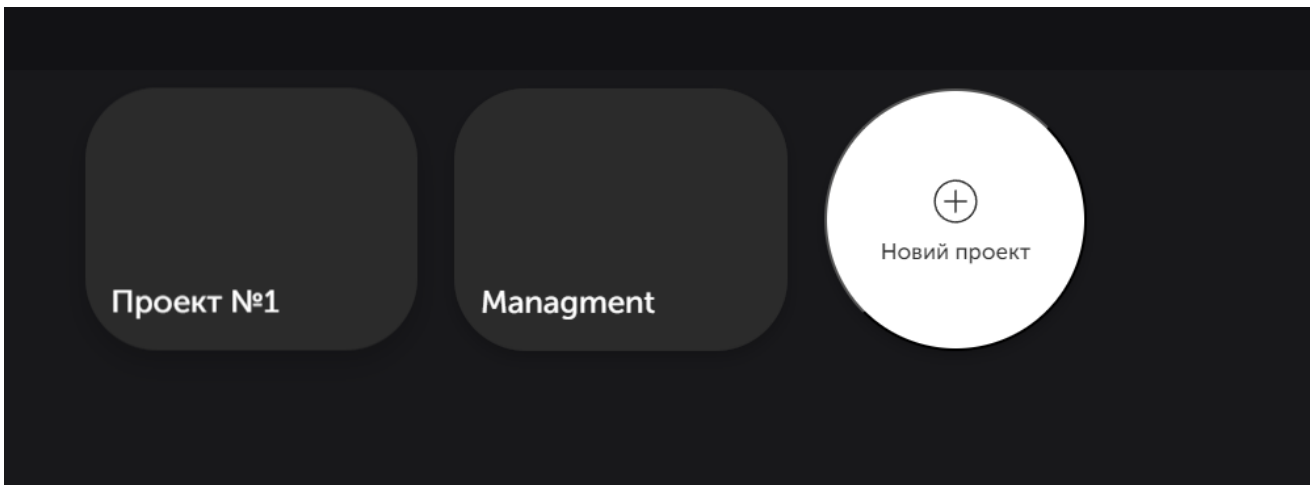

Рисунок 4.4 – Перенаправлення після успішного входу на головний екран

Наступна перевірка це тестування на зміну даних облікового запису, спочатку відбувається перевірка правильності введених даних. Це включає в себе перевірку формату нового електронногої пошти на відповідність стандартам електронної пошти та перевірка чи існує вже така пошта, що пароль відповідає вимогам безпеки. Після цього система автентифікує користувача, перевіряючи правильність введеного поточного пароля. У випадку помилок під час неправильного паролю, користувач повідомляється про помилку, а система відповідно реагує на неї. Якщо користувач успішно пройшов автентифікацію, система змінює старі дані на нові (рисунок 4.5).

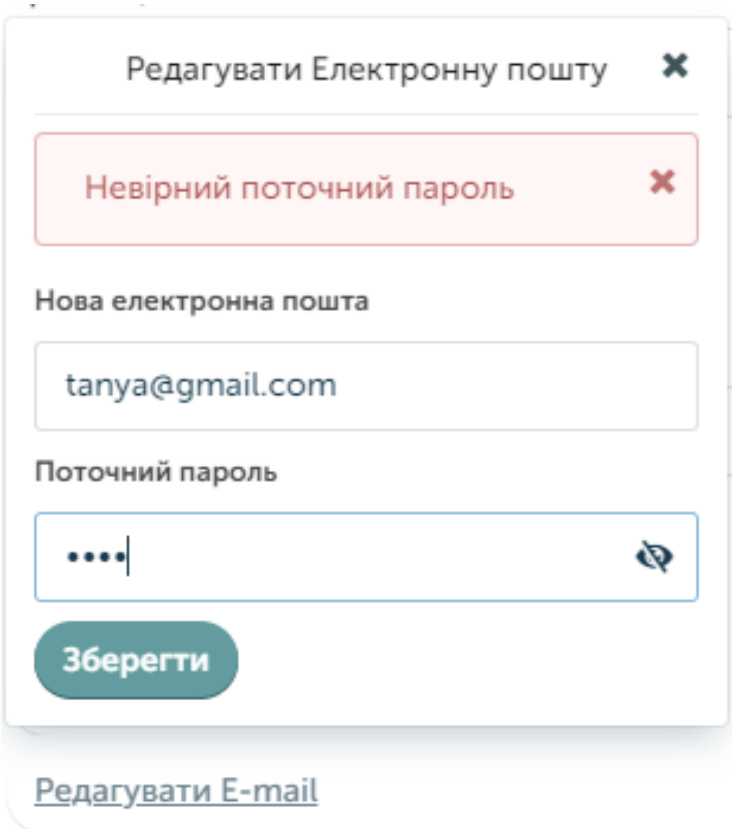

Рисунок 4.5 – Перевірка на поточний пароль користувача

Перевірка на довжину та надійності паролю. Під час перевірки на зміну паролю, спочатку система перевіряє довжину нового пароля (рисунок 4.6). Якщо він складає менше ніж 4 символів, то пароль вважається недостатньо надійним і не дозволяється його змінювати. Користувачу повідомляється про те, що необхідно ввести більш довгий пароль для забезпечення безпеки облікового запису.

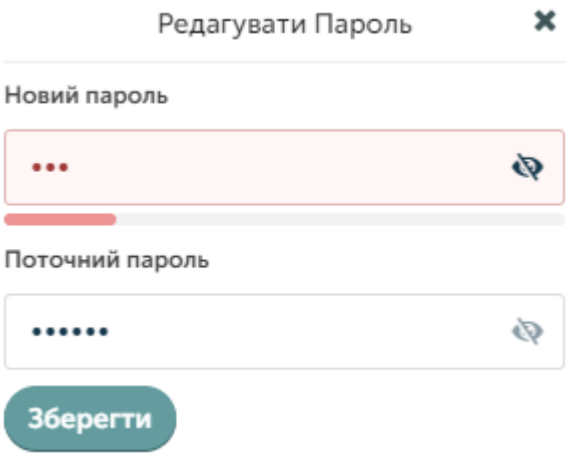

Рисунок 4.6 – Перевірка довжини та надійності пароля

Якщо новий пароль має довжину від 4 до 6 символів, це вважається мінімальним рівнем захисту. У цьому випадку користувачу також надсилається повідомлення, у вигляді індикатору що для підвищення безпеки рекомендується використовувати пароль, що містить більше 7 символів (рисунок 4.7). Коли новий пароль має 7 символів, вважається середнім рівнем захисту. В цьому випадку користувач може змінити свій пароль, але він також повідомляється про можливість підвищити рівень безпеки шляхом використання більш складного пароля.

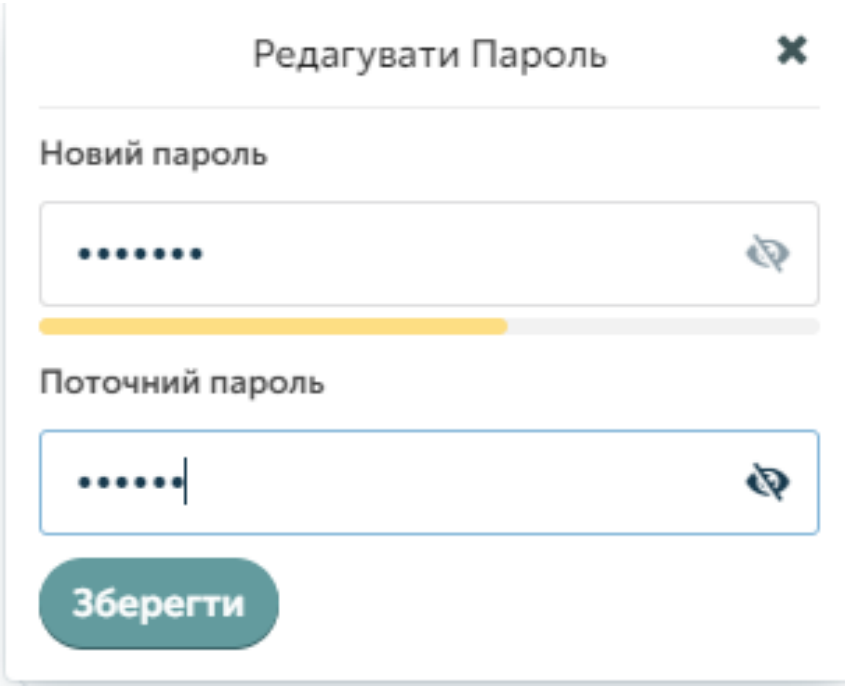

Рисунок 4.7 – Середня довжина паролю

Якщо новий пароль має 9 або більше символів, вважається високим рівнем захисту (рисунок 4.8). У цьому випадку користувач може змінити пароль, і система повідомляє йому про те, що його пароль є дуже безпечним. Така система перевірки дозволяє забезпечити належний рівень безпеки для облікових записів користувачів, одночасно допомагаючи користувачам створювати сильні паролі для захисту своїх даних.

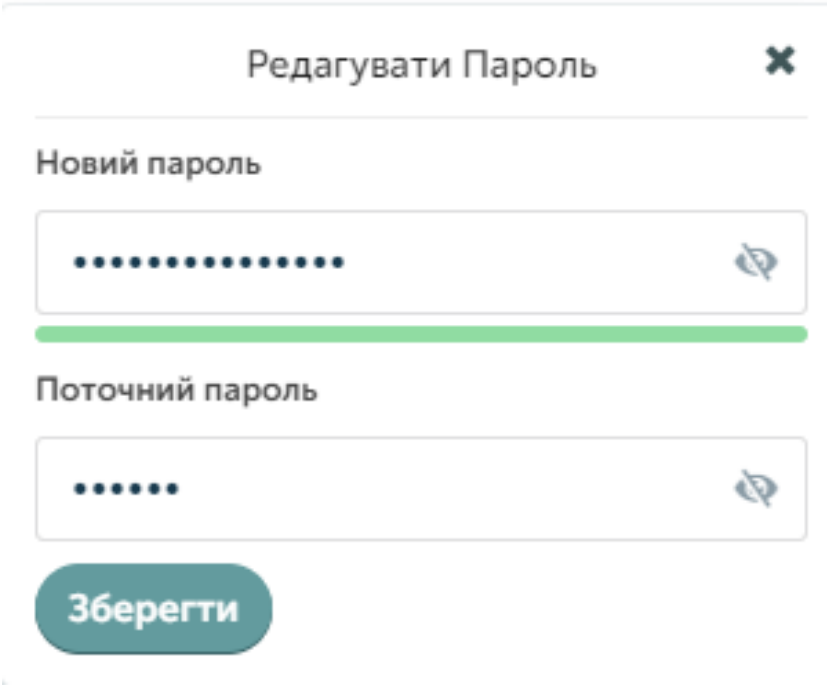

Рисунок 4.8 – Максимальна надійність пароля

Перевірка додавання нового користувача до проекту включає в себе кілька важливих етапів. Після отримання запиту на додавання нового користувача система спочатку перевіряє наявність користувача з вказаною електронною поштою в базі даних проекту. Це важливий крок, оскільки дозволяє уникнути дублювання облікових записів і забезпечити коректну роботу системи. Якщо користувач з таким самим емейлом вже існує, система видасть повідомлення про помилку, що такий користувач вже присутній в системі, і операція додавання буде скасована (рисунок 4.9).

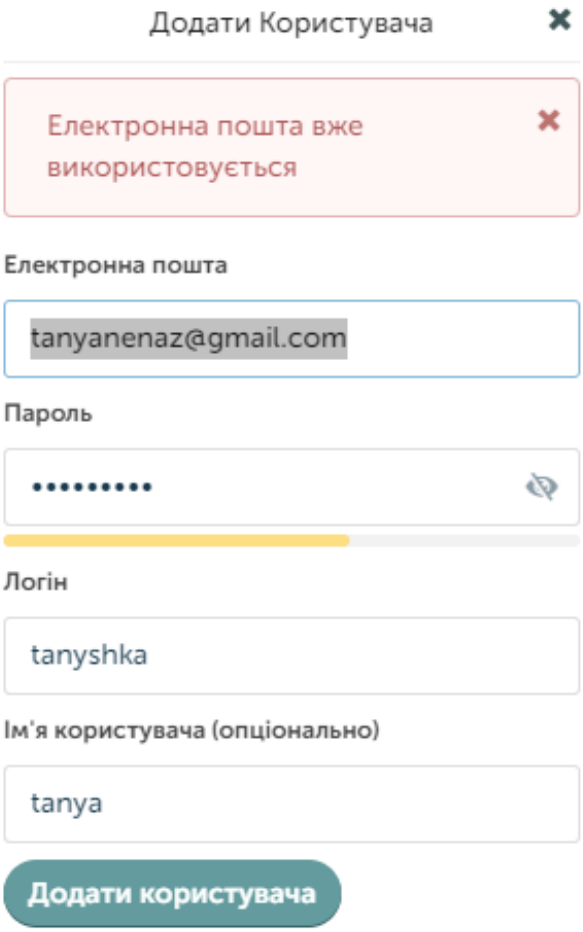

Рисунок 4.9 – Додавання користувача з однаковою поштою

Окрім дублювання пошти, перевіряється наявніть і схожого імені в базі даних, ця перевірка є важливою, оскільки дозволяє уникнути дублювання користувачів з однаковими іменами і забезпечити унікальність облікових записів. Якщо ж вказаного імені користувача немає у системі, то операція додавання буде продовжена, і новий користувач буде успішно доданий до проекту, а якщо є видається помилка введення даних і потрібно внести зміни (рисунок 4.10).

По-друге, ця перевірка сприяє забезпеченню безпеки даних. Шляхом виявлення вже існуючих користувачів з однаковими іменами або електронними адресами система уникне можливості несанкціонованого доступу до облікових записів або зловживання ними. Окрім цього, ефективне управління користувачами включає в себе швидке та безперервне додавання нових учасників до проекту.

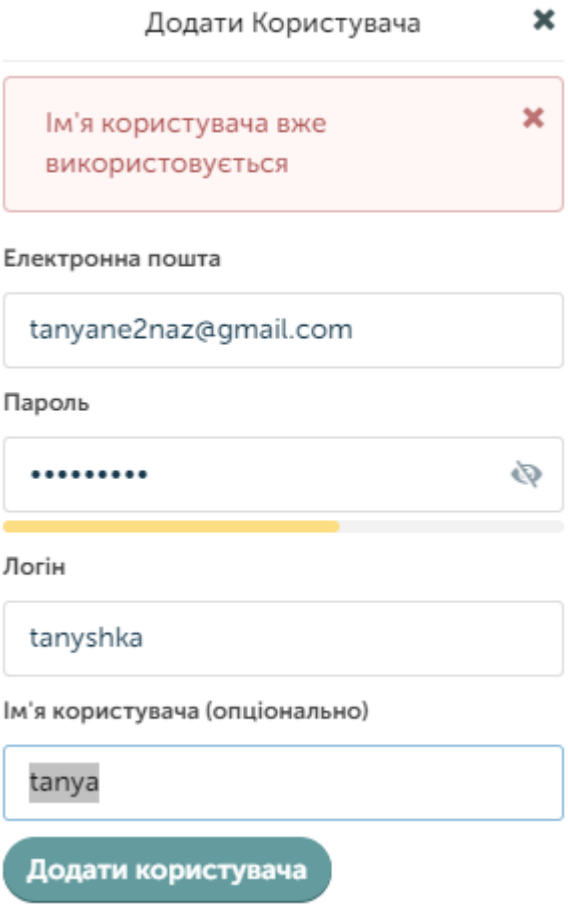

Рисунок 4.10 – Додавання користувача з таким самим іменем

Отже, перевірка цих даних є важливим етапом у забезпеченні безпеки та ефективності управління користувачами у проекті. По-перше, перевірка на унікальність даних перед їх додаванням забезпечує інтегрітет даних у системі. Це означає, що кожен користувач отримує унікальний ідентифікатор, що дозволяє ідентифікувати його в системі безпеки та дозволяє уникнути конфліктів.

4.2 Розробка інструкції користувача

Інструкція користувача включає в себе визначення технічних вимог для запуску програмного забезпечення. Інформація про мінімальні та рекомендовані конфігурації серверного комп’ютера описано в таблицях 4.1 та 4.2.

Таблиця 4.1 – Мінімальні вимоги до конфігурації комп’ютера

Параметр	Характеристика
Операційна система	Windows 10 (32-bit), macOS 10.14 або Ubuntu 18.04 LTS (64-bit)
Тип процесору	Intel Core i5-4460S 2.9 GHz або AMD Athlon X4 870K 3.9 GHz
Об’єм оперативної пам’яті	8 GB
Пропускна здатність мережі	100 Mbps
Вільного місця на диску	80 ГБ

Таблиця 4.2 – Рекомендовані вимоги до конфігурації комп’ютера

Параметр	Характеристика
Операційна система	Windows 10 (64-bit), macOS 10.14 або Ubuntu 18.04 LTS (64-bit)
Тип процесору	Intel Core i7-6700K 4.0 GHz або AMD Ryzen 5 1600 3.2 GHz
Об’єм оперативної пам’яті	16 GB
Пропускна здатність мережі	300 Mbps
Вільного місця на диску	160 ГБ

Інструкція про використання програмного забезпечення передбачає в собі інформацію про функції та можливості веб-платформи.

Користувачі, які вперше працюють з веб-платформою, спочатку мають увійти в систему, використовуючи свої облікові записи. Після входу в систему вони автоматично перенаправляються на головну сторінку, де можуть керувати своїми проектами. Основні функції включають створення нових проектів, перегляд та редагування існуючих проектів, а також можливість видалення власних проектів. Якщо користувач має доступ до проекту в команді, він може переглядати та працювати з проектом, але не матиме можливості видалити його без дозволу адміністратора. Якщо дозвіл адміністратора йому наданий, користувач може без перешкод керувати проектами та його учасниками.

Після переходу до конкретного проекту користувач отримує доступ до модуля управління дошкою, який дозволяє керувати завданнями на дошці. При додаванні нового завдання користувач може вказати дату терміну виконання, додати вкладення, опис завдання, нагадування, мітки та відмітити учасників. Крім того, користувач може відмічати завдання які виконані, додавати нові завдання та видаляти, коментувати їх під час роботи, а також додавати вкладення, такі як зображення та файли. При переході в проект з’являється управління керування дошкою, запускається модуль керуванням задачами (рисунок 4.11) при додаванні нового завдання.

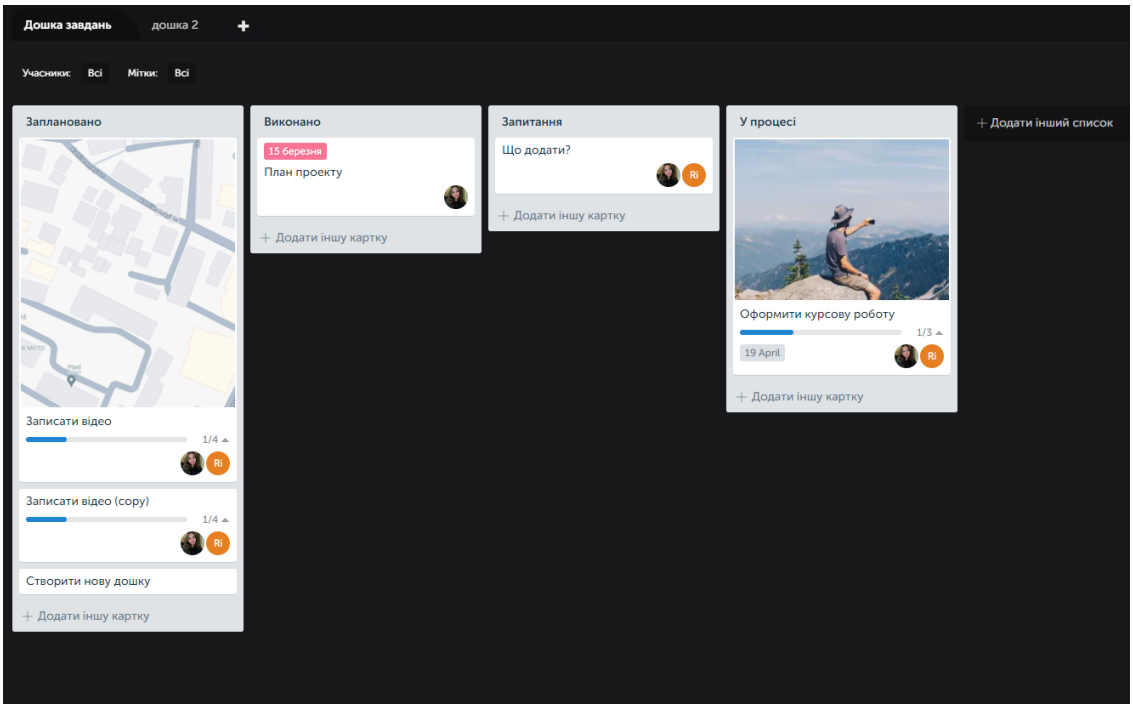

Рисунок 4.11 – Екран управління дошкою

Окрім керування дошкою, користувачі мають можливіть редагувати особисту інформацію та налаштування свого профілю (рисунок 4.12). Один з ключових аспектів – це можливість зміни електронної адреси, яка використовується для входу, а також пароля, щоб забезпечити безпеку облікового запису. Користувачі також можуть змінити свій логін – ім'я, під яким вони входять до системи, для власної зручності. Крім того, вони можуть додати номер телефону до свого облікового запису, щоб отримувати сповіщення або використовувати його як додатковий метод аутентифікації.

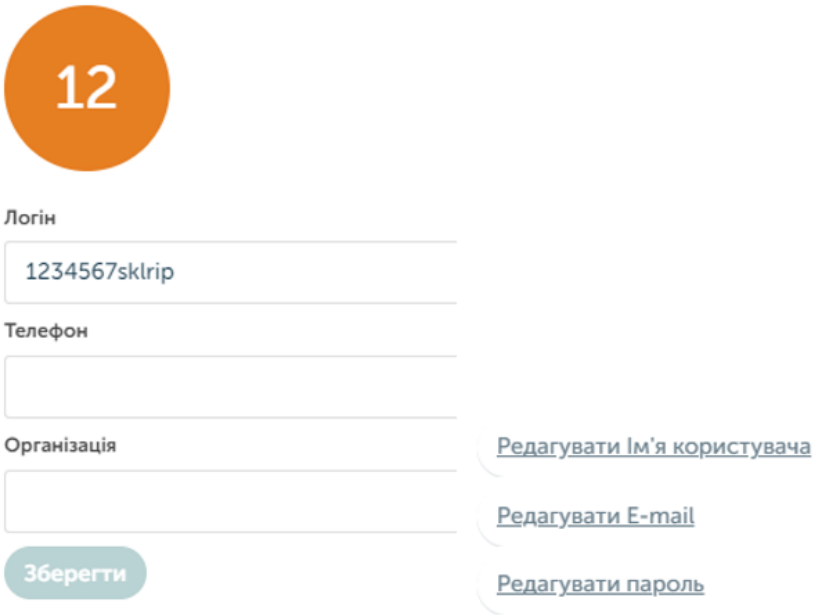

Рисунок 4.12 – Налаштування облікового запису

4.3 Висновки

У четвертому розділі було проведено тестування програмних модулів, для веб-платформи для організації коворкінгу. Особлива увага була приділена валідації полів введення інформації. Під час тестування здійснювалася перевірка коректності обробки даних, введених користувачем, а також виявлення та усунення можливих помилок чи некоректних дій.

Була розроблена інструкція користувача щодо використання програмного забезпечення. Ця інструкція включала в себе докладний опис функціональності

веб-платформи, пояснення основних кроків користування системою та була підготовлена інструкція початкової експлуатації програмного продукту.

У результаті проведення тестування та розробки відповідних інструкцій було досягнуто мети забезпечення захисту даних та зручного користування програмним продуктом.

ВИСНОВКИ

У бакалаврській дипломній роботі розроблено веб-платформу для організації коворкінгу. Для розробки використано середовище програмної розробки Visual Studio Code.

Дипломна робота розроблена згідно методичних вказівок до виконання бакалаврських дипломних робіт [22].

У бакалаврській дипломній роботі виконано наступне:

– проаналізовано стани платформ для організації коворкінгу;

– проведено порівняльний аналіз аналогів;

– поставлено задачу для розробки веб-платформи;

– проаналізовано технології, СКБД та середовища розробки для веб-платформи;

– розроблено модель системи;

– розроблено загальний алгоритм роботи веб-платформи;

– розроблено структуру та дизайн інтерфейсу користувача;

– розроблено модулі керуванням завдань, адміністратором та завантаження вкладень;

– розроблена інструкція користувача;

– проведено тестування веб-платформи.

Під час процесу виконання бакалаврської дипломної роботи було проведено порівняльний аналіз існуючих програмних рішень у сфері організації коворкінгу. Виявлені переваги та недоліки кожного з конкурентів які підтвердили актуальність та доцільність розробки веб-платформи.

У процесі аналізу технологій розробки веб-платформи було обрано JavaScript як основну мову програмування, а для створення інтерфейсу користувача було вирішено використовувати бібліотеку React. Як серверну частину обрано Node.js, а зберігання та управління даними було обрано базу даних PostgreSQL.

Було створено структуру інтерфейсу веб-платформи, що враховує потреби користувачів та забезпечує зручну навігацію та доступ до всіх функцій. Розроблена модель системи відображає її основні компоненти, їх взаємозв'язки та процеси, що відбуваються в рамках платформи.

Розроблено діяльність системи веб-платформи який детально описує послідовність дій та операцій, що виконуються системою при обробці запитів користувачів, управлінні проектами, забезпеченні доступу та інших функцій.

Проведено тестування продуктивності системи, що дозволило оцінити її здатність працювати під навантаженням та виявити потенційні вузькі місця, таакож було проведено тестування на виявлення помилок та забезпечення валідності даних та прав доступу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Who Uses Coworking Spaces in the Modern Workforce? The Tutor Resource: веб-сайт. URL: https://thetutorresource.com/who-uses-coworking-spaces/ (дата звернення: 28.03.2024).

2. Virtual Coworking: The Revolutionary Approach to Deep Work and Focus. Freedom Matters: веб-сайт. URL: https://freedom.to/blog/virtual-co-working/ (дата звернення: 06.04.2024).

3. Назаренко Т.Є. Розробка інноваційних технологій для розвитку сучасних робочих просторів / Т.Є. Назаренко, Катєльніков Д.І. // Матеріали VII Міжнародної науково-практичної конференції Сучасні тенденції розвитку фінансових та інноваційно-інвестиційних процесів в Україні, Вінниця, 2024. URL: https://conferences.vntu.edu.ua/index.php/fiip/fiip2024/paper/view/20162/16686

4. Що таке Atlassian Jira? Softlist: веб-сайт. URL: https://ua.softlist.com.ua/articles/chto-takoe-jira/ (дата звернення: 06.04.2024).

5. Філановський О. Notion, зручна програма для управління завданнями. Notion: веб-сайт. URL: https://icoola.ua/blog/programa-notion-na-iphone/ (дата звернення: 18.04.2024).

6. 30 Top Tools for Online Project Collaboration. Filestage: The world’s best-rated review and approval platform: веб-сайт. URL: https://filestage.io/blog/free-online-project-collaboration/ (дата звернення: 20.04.2024).

7. Java Professional. JavaRush. URL: https://javarush.com/ua/quests/lectures/ua.questservlets.level14.lecture00 (дата звернення: 26.04.2024).

8. Схеми бази даних: компоненти, типи та проєктування. FoxmindEd. URL: https://foxminded.ua/skhemy-bazy-danyh/ (дата звернення: 26.04.2024).

9. Hick’s Law. Laws of UX: веб-сайт. URL: https://lawsofux.com/hicks-law/ (дата звернення: 30.04.2024).

10. JavaScript | MDN. MDN Web Docs: веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 30.04.2024).

11. Конспект лекцій з дисципліни «Технології створення Web-застосувань» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньопрофесійною програмою «Інженерія програмного забезпечення» зі спеціальності 121 – «Інженерія програмного забезпечення» галузі знань 12 – «Інформаційні технології» / Укладач О.О.Шумейко.– Кам'янське: ДДТУ, 2019– 124 с.

12. Rust мова програмування: особливості та переваги в розробці. FoxmindEd. URL: https://foxminded.ua/rust-mova-prohramuvannia/ (дата звернення: 16.05.2024).

13. Що таке мова програмування Python та для чого вона використовується?. Sigma Software University. URL: https://university.sigma.software/what-is-python/ (дата звернення: 16.05.2024).

14. Sublime Text: встановлення, налаштування, плагіни. Laboratories: веб-сайт. URL: https://kr-labs.com.ua/blog/sublime-text-vstanovlennya-nalashtuvannya-plaginy/ (дата звернення: 18.05.2024).

15. A hackable text editor for the 21st Century: веб-сайт. URL: https://atom-editor.cc/ (дата звернення: 18.05.2024).

16. WebStorm: веб-сайт. URL: https://www.wikidata.uk-ua.nina.az/Webstorm.html (дата звернення: 20.05.2024).

17. What is Visual Studio Code: веб-сайт. URL: https://www.educative.io/answers/what-is-visual-studio-code (дата звернення: 20.05.2024).

18. База даних (БД) – Що це таке: веб-сайт. URL: https://hostiq.ua/wiki/ukr/database/ (дата звернення: 20.05.2024).

19. PostGres Structured Query Language (PostgreSQL) | CQR. CQR. URL: https://cqr.company/ua/wiki/protocols/postgres-structured-query-language/ (дата звернення: 20.05.2024).

20. Ростислав Суслов. Система управління базами даних MySQL. Lemon School. URL: https://lemon.school/blog/systema-upravlinnya-bazamy-danyh-mysql (дата звернення: 21.05.2024).

21. MongoDB – Вікіпедія. URL: https://uk.wikipedia.org/wiki/MongoDB (дата звернення: 21.05.2024).

22. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклад. : О. Н. Романюк, Г. О. Черноволик. – Вінниця : ВНТУ, 2022. – 51 с.

ДОДАТКИ

Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
_____ О. Н. Романюк
«12» березня 2024 р.

Технічне завдання
на бакалаврську дипломну роботу
«Розробка веб-платформи для організації коворкінгу з використанням технологій JavaScript, React, Postgre SQL та Node.js»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ к.т.н., доц. Катєльніков Д.І.
«12» березня 2024 р.

Виконала:
_____ студентка гр. 1ПІ-20б, Назаренко Т.Є.
«12» березня 2024 р.

Вінниця – 2024

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка веб-платформи для організації коворкінгу з використанням технологій JavaScript, React, Postgre SQL та Node.js».

Галузь застосування – управління проектами.

2. Підстава для розробки

Підставою для розробки бакалаврської дипломної роботи є рішення засідання кафедри програмного забезпечення – наказ №80 від «11» березня 2024 р. ректора ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки

Метою роботи є підвищення ефективності роботи веб-платформи для організації коворкінгу, спрямованої на полегшення організації роботи з проектами для користувачів у реальному часі.

Призначення роботи – розробка веб-платформи для організації коворкінгу.

4. Технічні вимоги

Вихідні дані до роботи: модель розробки – ітеративна; алгоритми реалізації – управління серверною частиною застосунку з використанням Node.js для обробки запитів; сервер баз даних – PostgreSQL; розробка інтерфейсу користувача – React; вхідні дані – дані про користувачів, список проектів, статус робочих задач; вихідні дані – звіти про прогрес проектів, розподіл завдань між учасниками команди; середовище розробки – Visual Studio Code; мова розробки – JavaScript; операційна система – Windows 10/11; середовище реалізації додатку – браузер.

5. Конструктивні вимоги.

Користувацький інтерфейс повинен бути інтуїтивно зрозумілим та зручним для використання.

Графічна та текстова документація повинна відповідати діючим стандартам України.

6. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до БДР;

– технічне завдання;

– лістинги програми.

7. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

8. Стадії і етапи розробки

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз проблеми, обґрунтування актуальності розробки веб-платформи та постановка задачі	14.03.24 – 25.03.24
2	Розробка структури, моделі та алгоритмів роботи веб-платформи	26.03.24 – 10.04.24
3	Аналіз і вибір середовища програмування та СКБД	11.04.24 – 18.04.24
4	Розробка модулів веб-платформи	19.04.24 – 08.05.24
5	Тестування роботи веб-платформи та розробка інструкції користувача	09.05.24 – 24.05.24
6	Оформлення матеріалів до захисту БДР	25.05.24 – 30.05.24

9. Порядок контролю і приймання

Порядок контролю і приймання роботи регламентується відповідними документами ВНТУ і державними стандартами.

Додаток Б. Перевірка на плагіат

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Розробка веб-платформи для організації коворкінгу з використанням технологій JavaScript, React, Postgre SQL та Node.js»

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: к.т.н., доц. Катєльніков Д.І.

Оригінальність	**96,1%**
Схожість	3,9 %

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, їх велика надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи Назаренко Т.Є.

Керівник роботи Катєльніков Д.І.

Додаток В. Лістинг програми

Main.js

```
//index.html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="Tanya Nazarenko 1PI20b"
      content="Coworking Spacef"
    />
    <link rel="apple-touch-icon" href="logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <title>Coworking Space</title>
  </head>
  <script>window.BASE_URL = "%PUBLIC_URL%";</script>
  <body id="app">
    <div id="root"></div>
  </body>
</html>

//Doshka.jsx
import React, { useCallback, useEffect, useRef, useState } from 'react';
import PropTypes from 'prop-types';
import { useTranslation } from 'react-i18next';
import { DragDropContext, Droppable } from 'react-beautiful-dnd';
import { closePopup } from '../../lib/popup';

import DroppableTypes from '../../constants/DroppableTypes';
import ListContainer from '../../containers/ListContainer';
import CardModalContainer from '../../containers/CardModalContainer';
import ListAdd from './ListAdd';
import { ReactComponent as PlusMathIcon } from '../../assets/images/plus-math-icon.svg';

import styles from './Board.module.scss';

const parseDndId = (dndId) => dndId.split(':')[1];

const Board = React.memo(
  ({ listIds, isCardModalOpened, canEdit, onListCreate, onListMove, onCardMove }) => {
    const [t] = useTranslation();
    const [isListAddOpened, setIsListAddOpened] = useState(false);

    const wrapper = useRef(null);
    const prevPosition = useRef(null);

    const handleAddListClick = useCallback(() => {
```

```
  setIsListAddOpened(true);
}, []);

const handleAddListClose = useCallback(() => {
  setIsListAddOpened(false);
}, []);

const handleDragStart = useCallback(() => {
  closePopup();
}, []);

const handleDragEnd = useCallback(
  ({ draggableId, type, source, destination }) => {
    if (
      !destination ||
      (source.droppableId === destination.droppableId && source.index === destination.index)
    ) {
      return;
    }

    const id = parseDndId(draggableId);

    switch (type) {
      case DroppableTypes.LIST:
        onListMove(id, destination.index);

        break;
      case DroppableTypes.CARD:
        onCardMove(id, parseDndId(destination.droppableId), destination.index);

        break;
      default:
    }
  },
  [onListMove, onCardMove],
);

const handleMouseDown = useCallback(
  (event) => {
    if (event.button) {
      return;
    }

    if (event.target !== wrapper.current && !event.target.dataset.dragScroller) {
      return;
    }

    prevPosition.current = event.clientX;
  },
  [wrapper],
);
```

```
const handleWindowMouseMove = useCallback(
  (event) => {
    if (!prevPosition.current) {
      return;
    }

    event.preventDefault();

    window.scrollBy({
      left: prevPosition.current - event.clientX,
    });

    prevPosition.current = event.clientX;
  },
  [prevPosition],
);

const handleWindowMouseUp = useCallback(() => {
  prevPosition.current = null;
}, [prevPosition]);

useEffect(() => {
  document.body.style.overflowX = 'auto';

  return () => {
    document.body.style.overflowX = null;
  };
}, []);

useEffect(() => {
  if (isListAddOpened) {
    window.scroll(document.body.scrollWidth, 0);
  }
}, [listIds, isListAddOpened]);

useEffect(() => {
  window.addEventListener('mouseup', handleWindowMouseUp);
  window.addEventListener('mousemove', handleWindowMouseMove);

  return () => {
    window.removeEventListener('mouseup', handleWindowMouseUp);
    window.removeEventListener('mousemove', handleWindowMouseMove);
  };
}, [handleWindowMouseUp, handleWindowMouseMove]);

return (
  <>
    <div ref={wrapper} className={styles.wrapper} onMouseDown={handleMouseDown}>
      <div>
        <DragDropContext onDragStart={handleDragStart} onDragEnd={handleDragEnd}>
          <Droppable droppableId="board" type={DroppableTypes.LIST} direction="horizontal">
            {({ innerRef, droppableProps, placeholder }) => (
```

```
              <div
                {...droppableProps}
                data-drag-scroller
                ref={innerRef}
                className={styles.lists}
              >
                {listIds.map((listId, index) => (
                  <ListContainer key={listId} id={listId} index={index} />
                ))}
                {placeholder}
                {canEdit && (
                  <div data-drag-scroller className={styles.list}>
                    {isListAddOpened ? (
                      <ListAdd onCreate={onListCreate} onClose={handleAddListClose} />
                    ) : (
                      <button
                        type="button"
                        className={styles.addListButton}
                        onClick={handleAddListClick}
                      >
                        <PlusMathIcon className={styles.addListButtonIcon} />
                        <span className={styles.addListButtonText}>
                          {listIds.length > 0
                            ? t('action.addAnotherList')
                            : t('action.addList')}
                        </span>
                      </button>
                    )}
                  </div>
                )}
              </div>
            )}
          </Droppable>
        </DragDropContext>
      </div>
    </div>
    {isCardModalOpened && <CardModalContainer />}
  </>
 );
},
);

Board.propTypes = {
 listIds: PropTypes.array.isRequired,
 isCardModalOpened: PropTypes.bool.isRequired,
 canEdit: PropTypes.bool.isRequired,
 onListCreate: PropTypes.func.isRequired,
 onListMove: PropTypes.func.isRequired,
 onCardMove: PropTypes.func.isRequired,
};

export default Board;
```

```
//Lists.jsx
import React, { useCallback, useEffect, useRef } from 'react';
import PropTypes from 'prop-types';
import { useTranslation } from 'react-i18next';
import { Button, Form, Input } from 'semantic-ui-react';
import { useDidUpdate, useToggle } from '../../lib/hooks';

import { useClosableForm, useForm } from '../../hooks';

import styles from './ListAdd.module.scss';

const DEFAULT_DATA = {
  name: '',
};

const ListAdd = React.memo(({ onCreate, onClose }) => {
  const [t] = useTranslation();
  const [data, handleFieldChange, setData] = useForm(DEFAULT_DATA);
  const [focusNameFieldState, focusNameField] = useToggle();

  const nameField = useRef(null);

  const handleFieldKeyDown = useCallback(
    (event) => {
      if (event.key === 'Escape') {
        onClose();
      }
    },
    [onClose],
  );

  const [handleFieldBlur, handleControlMouseOver, handleControlMouseOut] =
useClosableForm(onClose);

  const handleSubmit = useCallback(() => {
    const cleanData = {
      ...data,
      name: data.name.trim(),
    };

    if (!cleanData.name) {
      nameField.current.select();
      return;
    }

    onCreate(cleanData);

    setData(DEFAULT_DATA);
    focusNameField();
  }, [onCreate, data, setData, focusNameField]);

  useEffect(() => {
```

```
    nameField.current.focus();
  }, []);

  useDidUpdate(() => {
    nameField.current.focus();
  }, [focusNameFieldState]);

  return (
    <Form className={styles.wrapper} onSubmit={handleSubmit}>
      <Input
        ref={nameField}
        name="name"
        value={data.name}
        placeholder={t('common.enterListTitle')}
        className={styles.field}
        onKeyDown={handleFieldKeyDown}
        onChange={handleFieldChange}
        onBlur={handleFieldBlur}
      />
      <div className={styles.controls}>
        <Button
          positive
          content={t('action.addList')}
          className={styles.button}
          onMouseOver={handleControlMouseOver}
          onMouseOut={handleControlMouseOut}
        />
      </div>
    </Form>
  );
});

ListAdd.propTypes = {
  onCreate: PropTypes.func.isRequired,
  onClose: PropTypes.func.isRequired,
};

export default ListAdd;

//BG.jsx
import upperFirst from 'lodash/upperFirst';
import camelCase from 'lodash/camelCase';
import React from 'react';
import PropTypes from 'prop-types';
import classNames from 'classnames';

import { ProjectBackgroundTypes } from '../../constants/Enums';

import styles from './Background.module.scss';
import globalStyles from '../../styles.module.scss';

function Background({ type, name, imageUrl }) {
```

```
  return (
    <div
      className={classNames(
        styles.wrapper,
        type === ProjectBackgroundTypes.GRADIENT &&
          globalStyles[`background${upperFirst(camelCase(name))}`],
      )}
      style={{
        background: type === 'image' && `url("${imageUrl}") center / cover`,
      }}
    />
  );
}

Background.propTypes = {
  type: PropTypes.string.isRequired,
  name: PropTypes.string,
  imageUrl: PropTypes.string,
};

Background.defaultProps = {
  name: undefined,
  imageUrl: undefined,
};

export default Background;

//DoshkaActivity.jsx
import React from 'react';
import PropTypes from 'prop-types';

import Filters from './Filters';
import Memberships from '../Memberships';
import BoardMembershipPermissionsSelectStep from '../BoardMembershipPermissionsSelectStep';

import styles from './BoardActions.module.scss';

const BoardActions = React.memo(
  ({
    memberships,
    labels,
    filterUsers,
    filterLabels,
    allUsers,
    canEdit,
    canEditMemberships,
    onMembershipCreate,
    onMembershipUpdate,
    onMembershipDelete,
    onUserToFilterAdd,
    onUserFromFilterRemove,
    onLabelToFilterAdd,
```

```
    onLabelFromFilterRemove,
    onLabelCreate,
    onLabelUpdate,
    onLabelMove,
    onLabelDelete,
  }) => {
    return (
      <div className={styles.wrapper}>
        <div className={styles.actions}>
          <div className={styles.action}>
            <Memberships
              items={memberships}
              allUsers={allUsers}
              permissionsSelectStep={BoardMembershipPermissionsSelectStep}
              canEdit={canEditMemberships}
              onCreate={onMembershipCreate}
              onUpdate={onMembershipUpdate}
              onDelete={onMembershipDelete}
            />
          </div>
          <div className={styles.action}>
            <Filters
              users={filterUsers}
              labels={filterLabels}
              allBoardMemberships={memberships}
              allLabels={labels}
              canEdit={canEdit}
              onUserAdd={onUserToFilterAdd}
              onUserRemove={onUserFromFilterRemove}
              onLabelAdd={onLabelToFilterAdd}
              onLabelRemove={onLabelFromFilterRemove}
              onLabelCreate={onLabelCreate}
              onLabelUpdate={onLabelUpdate}
              onLabelMove={onLabelMove}
              onLabelDelete={onLabelDelete}
            />
          </div>
        </div>
      </div>
    );
  },
);

BoardActions.propTypes = {
  memberships: PropTypes.array.isRequired,
  labels: PropTypes.array.isRequired,
  filterUsers: PropTypes.array.isRequired,
  filterLabels: PropTypes.array.isRequired,
  allUsers: PropTypes.array.isRequired,
  canEdit: PropTypes.bool.isRequired,
  canEditMemberships: PropTypes.bool.isRequired,
  onMembershipCreate: PropTypes.func.isRequired,
```

```
  onMembershipUpdate: PropTypes.func.isRequired,
  onMembershipDelete: PropTypes.func.isRequired,
  onUserToFilterAdd: PropTypes.func.isRequired,
  onUserFromFilterRemove: PropTypes.func.isRequired,
  onLabelToFilterAdd: PropTypes.func.isRequired,
  onLabelFromFilterRemove: PropTypes.func.isRequired,
  onLabelCreate: PropTypes.func.isRequired,
  onLabelUpdate: PropTypes.func.isRequired,
  onLabelMove: PropTypes.func.isRequired,
  onLabelDelete: PropTypes.func.isRequired,
};

export default BoardActions;

//filering.jsx
import React, { useCallback } from 'react';
import PropTypes from 'prop-types';
import { useTranslation } from 'react-i18next';
import { usePopup } from '../../lib/popup';

import User from '../User';
import Label from '../Label';
import BoardMembershipsStep from '../BoardMembershipsStep';
import LabelsStep from '../LabelsStep';

import styles from './Filters.module.scss';

const Filters = React.memo(
  ({
    users,
    labels,
    allBoardMemberships,
    allLabels,
    canEdit,
    onUserAdd,
    onUserRemove,
    onLabelAdd,
    onLabelRemove,
    onLabelCreate,
    onLabelUpdate,
    onLabelMove,
    onLabelDelete,
  }) => {
    const [t] = useTranslation();

    const handleRemoveUserClick = useCallback(
      (id) => {
        onUserRemove(id);
      },
      [onUserRemove],
    );
```

```
const handleRemoveLabelClick = useCallback(
  (id) => {
    onLabelRemove(id);
  },
  [onLabelRemove],
);

const BoardMembershipsPopup = usePopup(BoardMembershipsStep);
const LabelsPopup = usePopup(LabelsStep);

return (
  <>
    <span className={styles.filter}>
      <BoardMembershipsPopup
        items={allBoardMemberships}
        currentUserIds={users.map((user) => user.id)}
        title="common.filterByMembers"
        onUserSelect={onUserAdd}
        onUserDeselect={onUserRemove}
      >
        <button type="button" className={styles.filterButton}>
          <span className={styles.filterTitle}>{`${t('common.members')}:`}</span>
          {users.length === 0 && <span className={styles.filterLabel}>{t('common.all')}</span>}
        </button>
      </BoardMembershipsPopup>
      {users.map((user) => (
        <span key={user.id} className={styles.filterItem}>
          <User
            name={user.name}
            avatarUrl={user.avatarUrl}
            size="tiny"
            onClick={() => handleRemoveUserClick(user.id)}
          />
        </span>
      ))}
    </span>
    <span className={styles.filter}>
      <LabelsPopup
        items={allLabels}
        currentIds={labels.map((label) => label.id)}
        title="common.filterByLabels"
        canEdit={canEdit}
        onSelect={onLabelAdd}
        onDeselect={onLabelRemove}
        onCreate={onLabelCreate}
        onUpdate={onLabelUpdate}
        onMove={onLabelMove}
        onDelete={onLabelDelete}
      >
        <button type="button" className={styles.filterButton}>
          <span className={styles.filterTitle}>{`${t('common.labels')}:`}</span>
```

```
            {labels.length === 0 && <span
className={styles.filterLabel}>{t('common.all')}</span>}
          </button>
        </LabelsPopup>
        {labels.map((label) => (
          <span key={label.id} className={styles.filterItem}>
            <Label
              name={label.name}
              color={label.color}
              size="small"
              onClick={() => handleRemoveLabelClick(label.id)}
            />
          </span>
        ))}
      </span>
    </>
  );
 },
);

Filters.propTypes = {
 users: PropTypes.array.isRequired,
 labels: PropTypes.array.isRequired,
 allBoardMemberships: PropTypes.array.isRequired,
 allLabels: PropTypes.array.isRequired,
 canEdit: PropTypes.bool.isRequired,
 onUserAdd: PropTypes.func.isRequired,
 onUserRemove: PropTypes.func.isRequired,
 onLabelAdd: PropTypes.func.isRequired,
 onLabelRemove: PropTypes.func.isRequired,
 onLabelCreate: PropTypes.func.isRequired,
 onLabelUpdate: PropTypes.func.isRequired,
 onLabelMove: PropTypes.func.isRequired,
 onLabelDelete: PropTypes.func.isRequired,
};

export default Filters;

//attachments.js
const util = require('util');
const { v4: uuid } = require('uuid');

const Errors = {
 NOT_ENOUGH_RIGHTS: {
  notEnoughRights: 'Not enough rights',
 },
 CARD_NOT_FOUND: {
  cardNotFound: 'Card not found',
 },
 NO_FILE_WAS_UPLOADED: {
  noFileWasUploaded: 'No file was uploaded',
 },
```

```
};

module.exports = {
  inputs: {
    cardId: {
      type: 'string',
      regex: /^[0-9]+$/,
      required: true,
    },
    requestId: {
      type: 'string',
      isNotEmptyString: true,
    },
  },

  exits: {
    notEnoughRights: {
      responseType: 'forbidden',
    },
    cardNotFound: {
      responseType: 'notFound',
    },
    noFileWasUploaded: {
      responseType: 'unprocessableEntity',
    },
    uploadError: {
      responseType: 'unprocessableEntity',
    },
  },

  async fn(inputs, exits) {
    const { currentUser } = this.req;

    const { card } = await sails.helpers.cards
      .getProjectPath(inputs.cardId)
      .intercept('pathNotFound', () => Errors.CARD_NOT_FOUND);

    const boardMembership = await BoardMembership.findOne({
      boardId: card.boardId,
      userId: currentUser.id,
    });

    if (!boardMembership) {
      throw Errors.CARD_NOT_FOUND;
    }

    if (boardMembership.role !== BoardMembership.Roles.EDITOR) {
      throw Errors.NOT_ENOUGH_RIGHTS;
    }

    const upload = util.promisify((options, callback) =>
      this.req.file('file').upload(options, (error, files) => callback(error, files)),
```

```
    );

    let files;
    try {
      files = await upload({
        saveAs: uuid(),
        maxBytes: null,
      });
    } catch (error) {
      return exits.uploadError(error.message);
    }

    if (files.length === 0) {
      throw Errors.NO_FILE_WAS_UPLOADED;
    }

    const file = _.last(files);
    const fileData = await sails.helpers.attachments.processUploadedFile(file);

    const attachment = await sails.helpers.attachments.createOne.with({
      values: {
        ...fileData,
        card,
        creatorUser: currentUser,
      },
      requestId: inputs.requestId,
      request: this.req,
    });

    return exits.success({
      item: attachment,
    });
  },
};

//projectcreate.js
module.exports = {
  inputs: {
    name: {
      type: 'string',
      required: true,
    },
  },

  async fn(inputs) {
    const { currentUser } = this.req;

    const values = _.pick(inputs, ['name']);

    const { project, projectManager } = await sails.helpers.projects.createOne.with({
      values,
      user: currentUser,
```

```
      request: this.req,
    });

    return {
      item: project,
      included: {
        projectManagers: [projectManager],
      },
    };
  },
};

module.exports = {
  async fn() {
    const { currentUser } = this.req;

    const managerProjectIds = await sails.helpers.users.getManagerProjectIds(currentUser.id);
    const managerProjects = await sails.helpers.projects.getMany(managerProjectIds);

    let boardMemberships = await sails.helpers.users.getBoardMemberships(currentUser.id);
    const membershipBoardIds = sails.helpers.utils.mapRecords(boardMemberships, 'boardId');

    let membershipBoards = await sails.helpers.boards.getMany({
      id: membershipBoardIds,
      projectId: {
        '!=': managerProjectIds,
      },
    });

    let membershipProjectIds = sails.helpers.utils.mapRecords(membershipBoards, 'projectId', true);
    const membershipProjects = await sails.helpers.projects.getMany(membershipProjectIds);

    membershipProjectIds = sails.helpers.utils.mapRecords(membershipProjects);

    const projectIds = [...managerProjectIds, ...membershipProjectIds];
    const projects = [...managerProjects, ...membershipProjects];

    const projectManagers = await sails.helpers.projects.getProjectManagers(projectIds);

    const userIds = sails.helpers.utils.mapRecords(projectManagers, 'userId', true);
    const users = await sails.helpers.users.getMany(userIds);

    const managerBoards = await sails.helpers.projects.getBoards(managerProjectIds);

    membershipBoards = membershipBoards.filter((membershipBoard) =>
      membershipProjectIds.includes(membershipBoard.projectId),
    );

    const boards = [...managerBoards, ...membershipBoards];
    const boardIds = sails.helpers.utils.mapRecords(boards);

    boardMemberships = boardMemberships.filter((boardMembership) =>
```

```
      boardIds.includes(boardMembership.boardId),
    );

    return {
      items: projects,
      included: {
        users,
        projectManagers,
        boards,
        boardMemberships,
      },
    };
  },
};
const Errors = {
  PROJECT_NOT_FOUND: {
    projectNotFound: 'Project not found',
  },
};

module.exports = {
  inputs: {
    id: {
      type: 'string',
      regex: /^[0-9]+$/,
      required: true,
    },
  },

  exits: {
    projectNotFound: {
      responseType: 'notFound',
    },
  },

  async fn(inputs) {
    const { currentUser } = this.req;

    const project = await Project.findOne(inputs.id);

    if (!project) {
      throw Errors.PROJECT_NOT_FOUND;
    }

    let boards = await sails.helpers.projects.getBoards(project.id);
    let boardIds = sails.helpers.utils.mapRecords(boards);

    const boardMemberships = await sails.helpers.boardMemberships.getMany({
      boardId: boardIds,
      userId: currentUser.id,
    });
```

```
    const isProjectManager = await sails.helpers.users.isProjectManager(currentUser.id, project.id);

    if (!isProjectManager) {
      if (boardMemberships.length === 0) {
        throw Errors.PROJECT_NOT_FOUND; // Forbidden
      }

      boardIds = sails.helpers.utils.mapRecords(boardMemberships, 'boardId');
      boards = boards.filter((board) => boardIds.includes(board.id));
    }

    const projectManagers = await sails.helpers.projects.getProjectManagers(project.id);

    const userIds = sails.helpers.utils.mapRecords(projectManagers, 'userId');
    const users = await sails.helpers.users.getMany(userIds);

    return {
      item: project,
      included: {
        users,
        projectManagers,
        boards,
        boardMemberships,
      },
    };
  },
};
const Errors = {
  PROJECT_NOT_FOUND: {
    projectNotFound: 'Project not found',
  },
};

const backgroundValidator = (value) => {
  if (_.isNull(value)) {
    return true;
  }

  if (!_.isPlainObject(value)) {
    return false;
  }

  if (!Object.values(Project.BackgroundTypes).includes(value.type)) {
    return false;
  }

  if (
    value.type === Project.BackgroundTypes.GRADIENT &&
    _.size(value) === 2 &&
    Project.BACKGROUND_GRADIENTS.includes(value.name)
  ) {
    return true;
```

```
  }

  if (value.type === Project.BackgroundTypes.IMAGE && _.size(value) === 1) {
    return true;
  }

  return false;
};

const backgroundImageValidator = (value) => _.isNull(value);

module.exports = {
  inputs: {
    id: {
      type: 'string',
      regex: /^[0-9]+$/,
      required: true,
    },
    name: {
      type: 'string',
      isNotEmptyString: true,
    },
    background: {
      type: 'json',
      custom: backgroundValidator,
    },
    backgroundImage: {
      type: 'json',
      custom: backgroundImageValidator,
    },
  },

  exits: {
    projectNotFound: {
      responseType: 'notFound',
    },
  },

  async fn(inputs) {
    const { currentUser } = this.req;

    let project = await Project.findOne(inputs.id);

    if (!project) {
      throw Errors.PROJECT_NOT_FOUND;
    }

    const isProjectManager = await sails.helpers.users.isProjectManager(currentUser.id, project.id);

    if (!isProjectManager) {
      throw Errors.PROJECT_NOT_FOUND; // Forbidden
    }
```

```
    const values = _.pick(inputs, ['name', 'background', 'backgroundImage']);

    project = await sails.helpers.projects.updateOne.with({
      values,
      record: project,
      request: this.req,
    });

    if (!project) {
      throw Errors.PROJECT_NOT_FOUND;
    }

    return {
      item: project,
    };
  },
};
```

Додаток Г. Графічна частина

ГРАФІЧНА ЧАСТИНА

РОЗРОБКА ВЕБ-ПЛАТФОРМИ ДЛЯ ОРГАНІЗАЦІЇ КОВОРКІНГУ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ JAVASCRIPT, REACT, POSTGRE SQL ТА NODE.JS

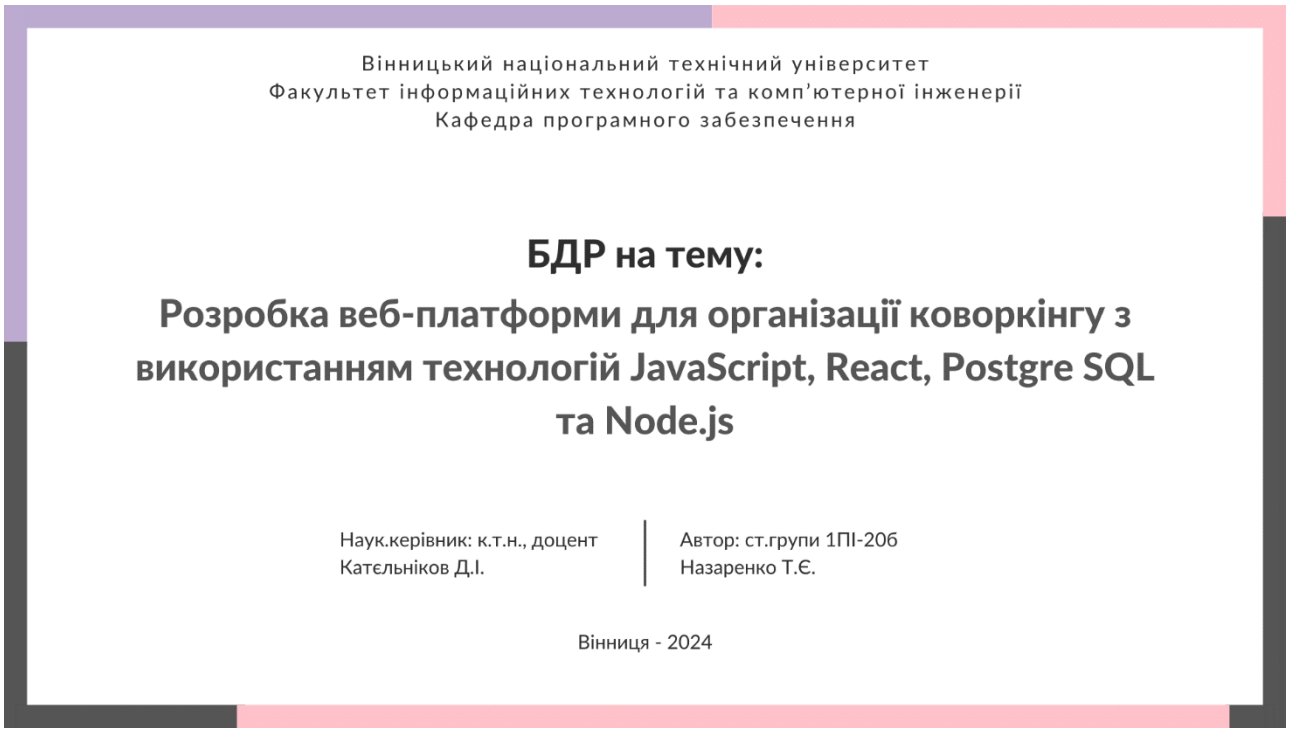

Рисунок Г.1 – Титульний слайд

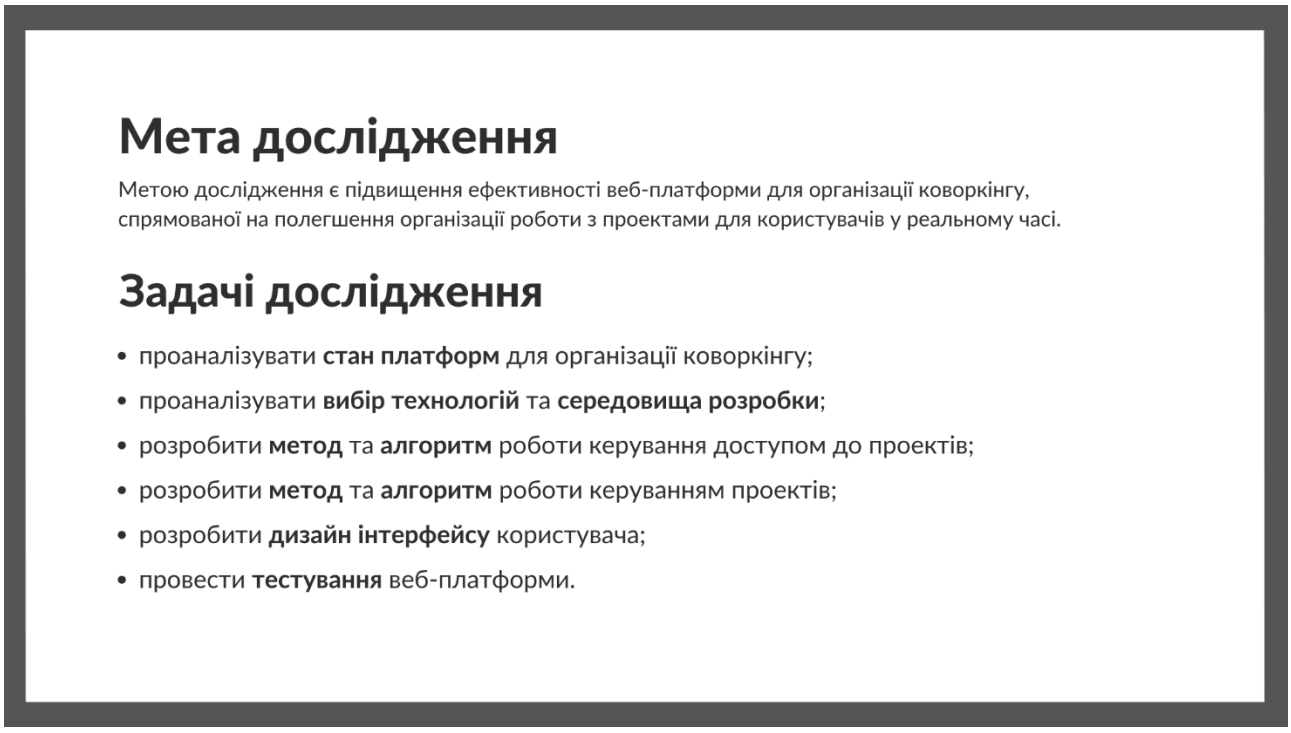

Рисунок Г.2 – Мета та задачі дослідження

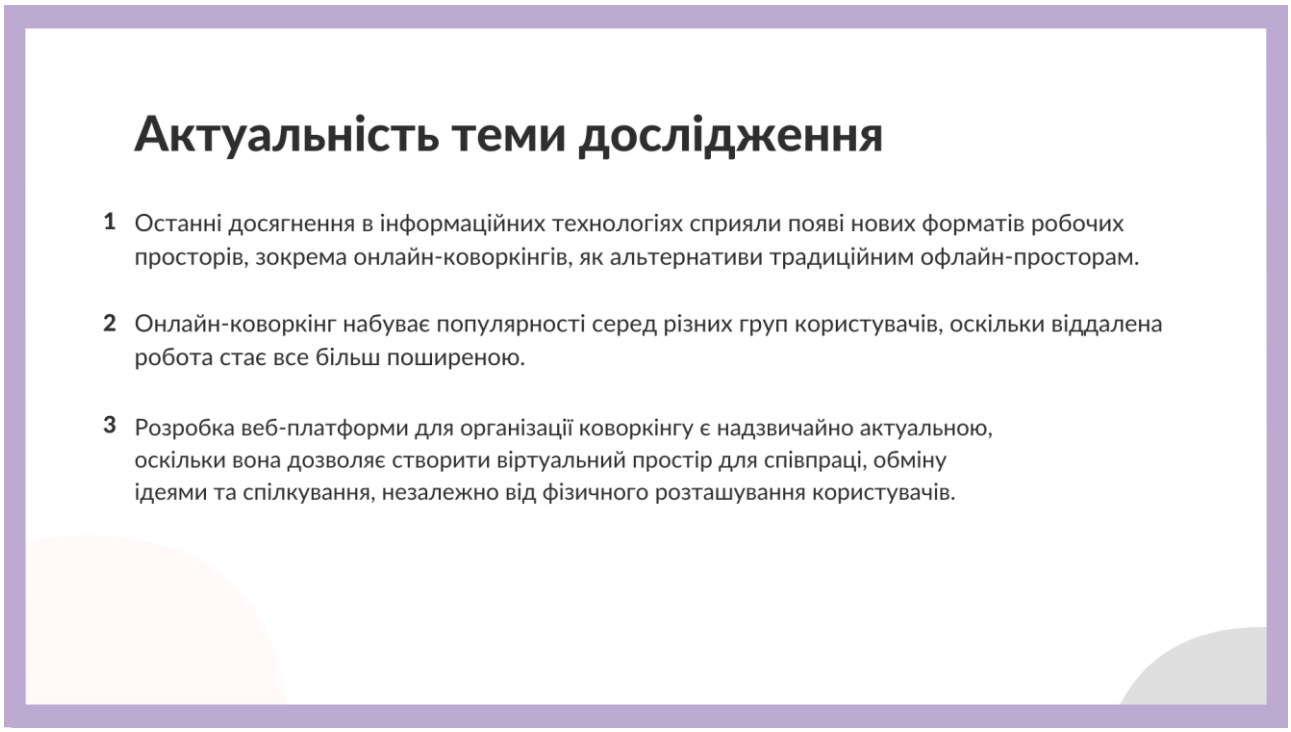

Рисунок Г.3 – Актуальність теми дослідження

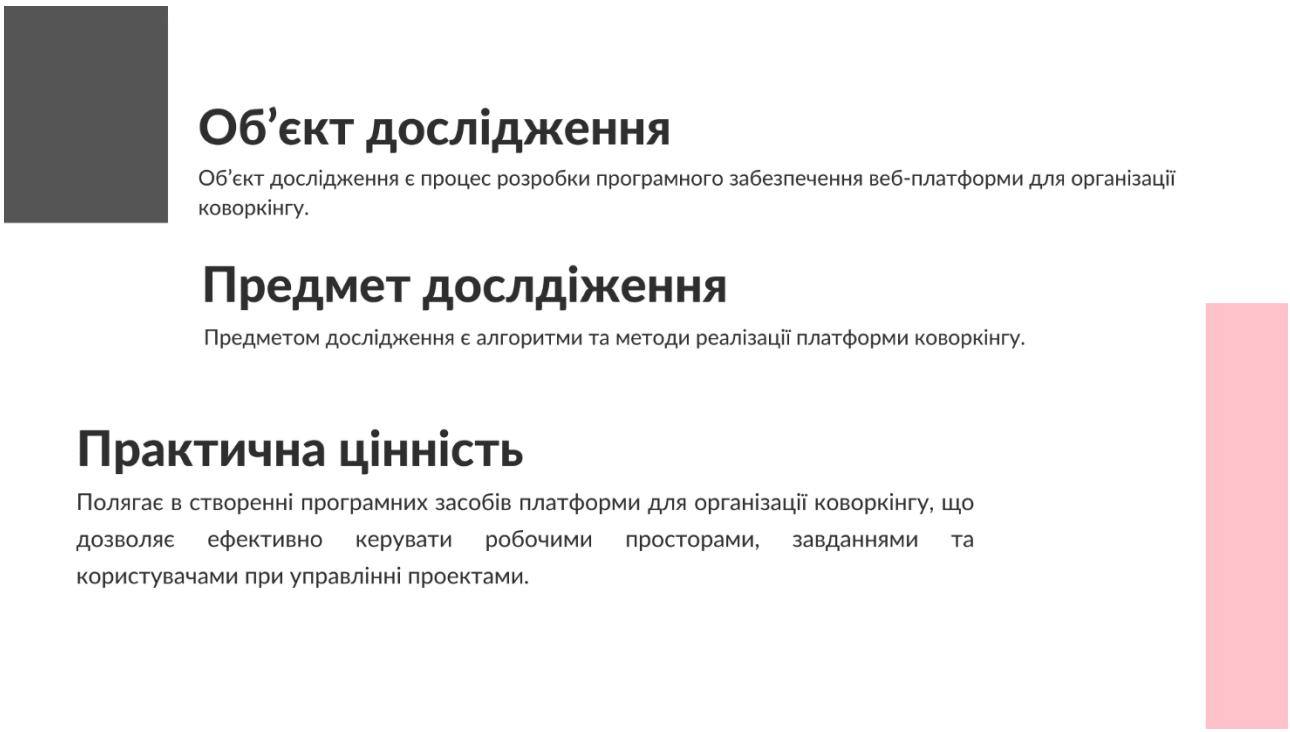

Рисунок Г.4 – Об’єкт, предмет дослідження та практична цінність

Наукова новизна

Подальшого розвитку отримав метод управління завданнями проекта, у якому на відміну від існуючих методів управління завданнями, застосовується асинхронна обробка запитів за допомогою Node.js, що дозволяє зменшити затримки при оновленні інформації в реальному часі та підвищити продуктивність системи на 20%.

Подальшого розвитку отримав метод адміністрування користувачів, у якому на відміну від існуючих методів адміністрування, застосовується централізована систему керування доступом на базі PostgreSQL, яка дозволяє підвищити безпеку даних та спрощує процес адміністрування за рахунок централізованого контролю доступу.

Рисунок Г.5 – Наукова новизна

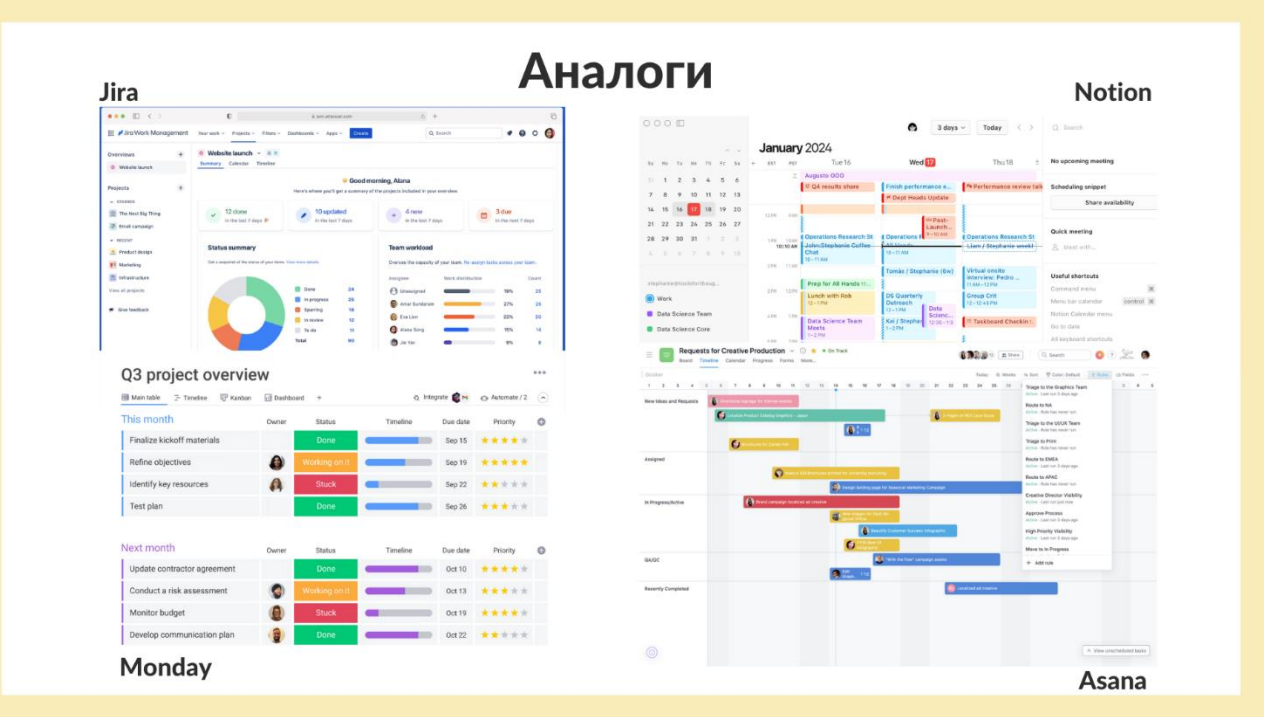

Рисунок Г.6 – Аналоги

Порівняння аналогів

Критерії/Назва	Jira	Notion	Asana	Monday	Власна веб-платформа
Дизайн і інтерфейс	0	0	1	1	1
Розширені функціональні можливості	1	0	1	0	1
Доступність	0	0	1	0	1
Інтеграції	1	1	0	1	1
Кросплатформеність	1	1	1	1	1
Загальна оцінка	60%	40%	80%	60%	100%

Рисунок Г.7 – Порівняння аналогів

Діаграма моделі системи

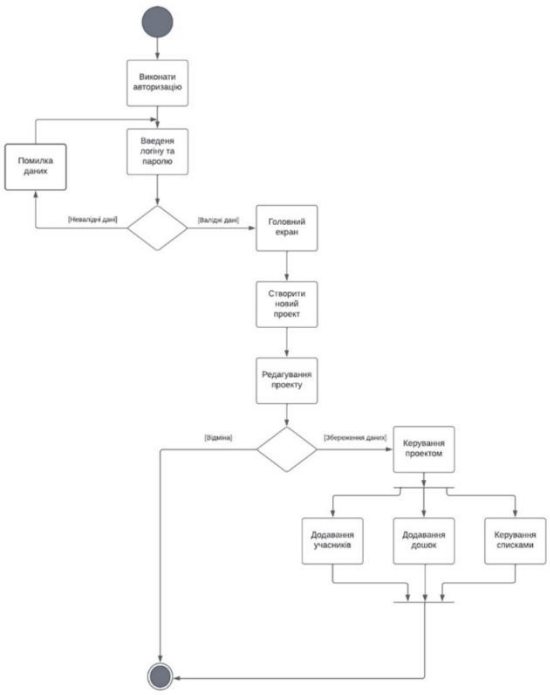

Рисунок Г.8 – Діаграма моделі системи

Алгоритм роботи головного екрану

Рисунок Г.9 – Алгоритм роботи головного екрану

Алгоритм роботи керування дошками

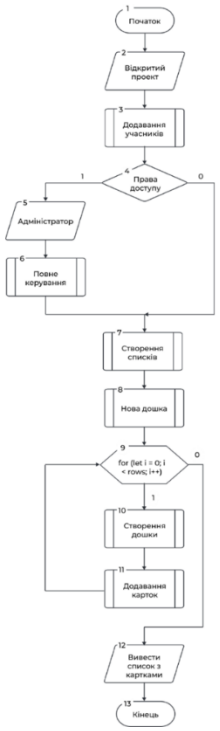

Рисунок Г.10 – Алгоритм роботи керування дошками

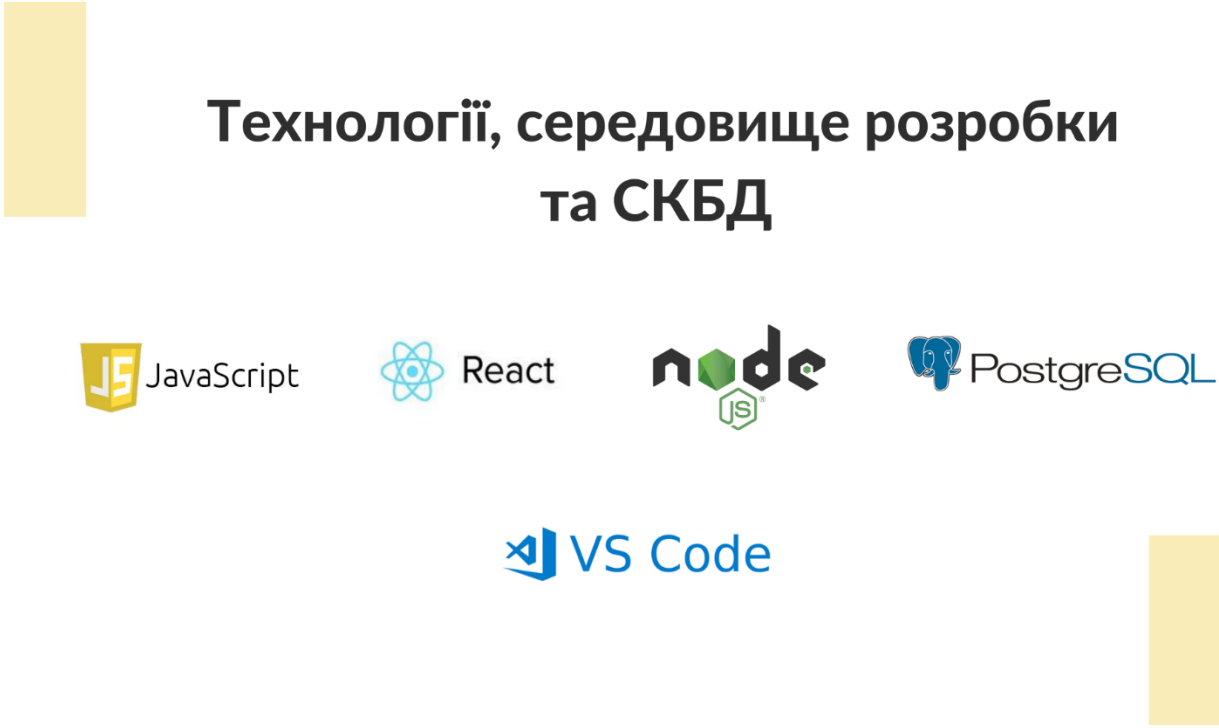

Рисунок Г.11 – Вибір технологій, середовища та СКБД

Рисунок Г.12 – Архітектура бази даних веб-платформи

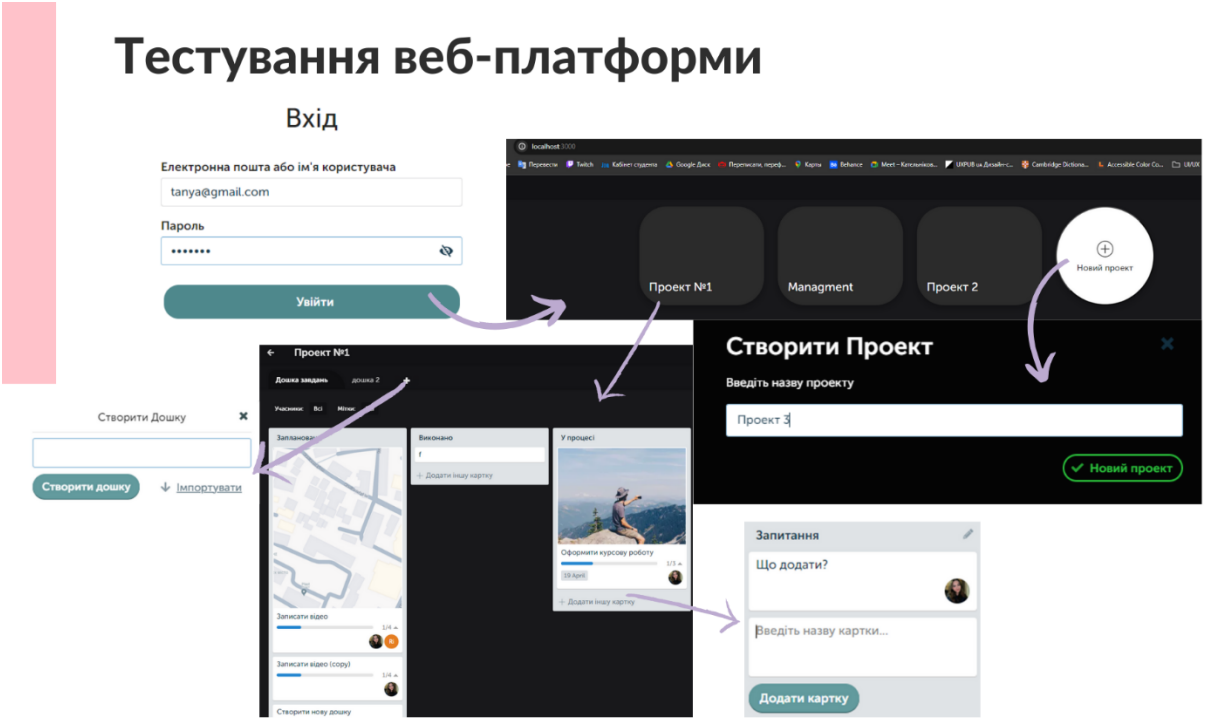

Рисунок Г.13 – Тестування веб-платформи

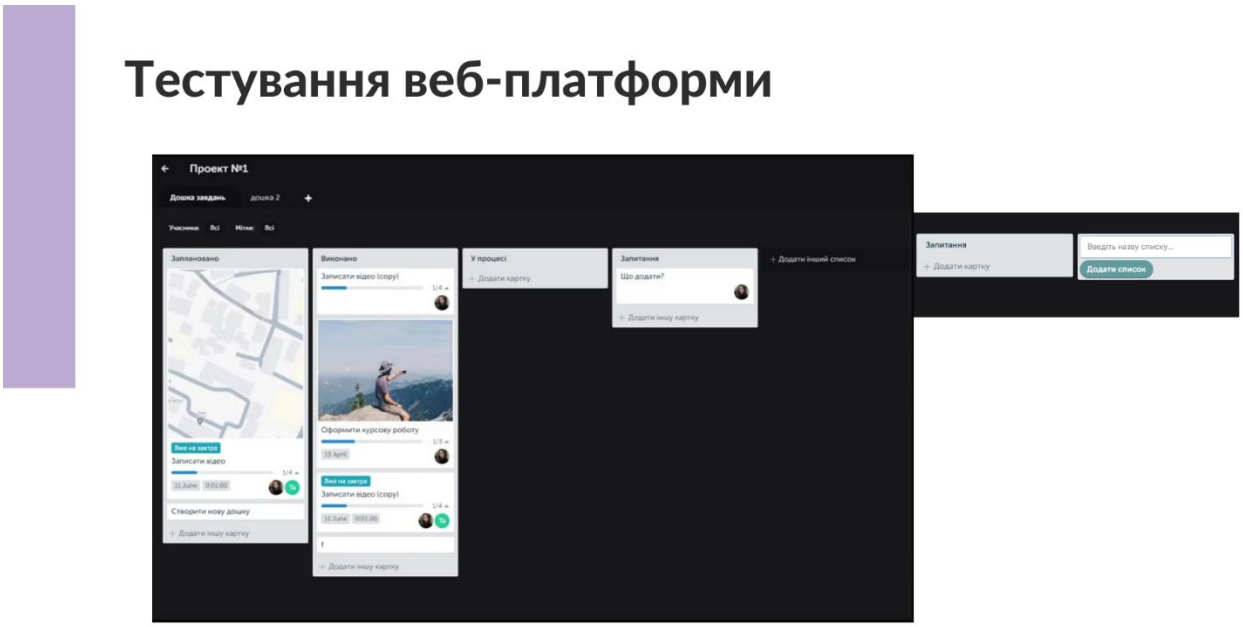

Рисунок Г.14 – Тестування веб-платформи

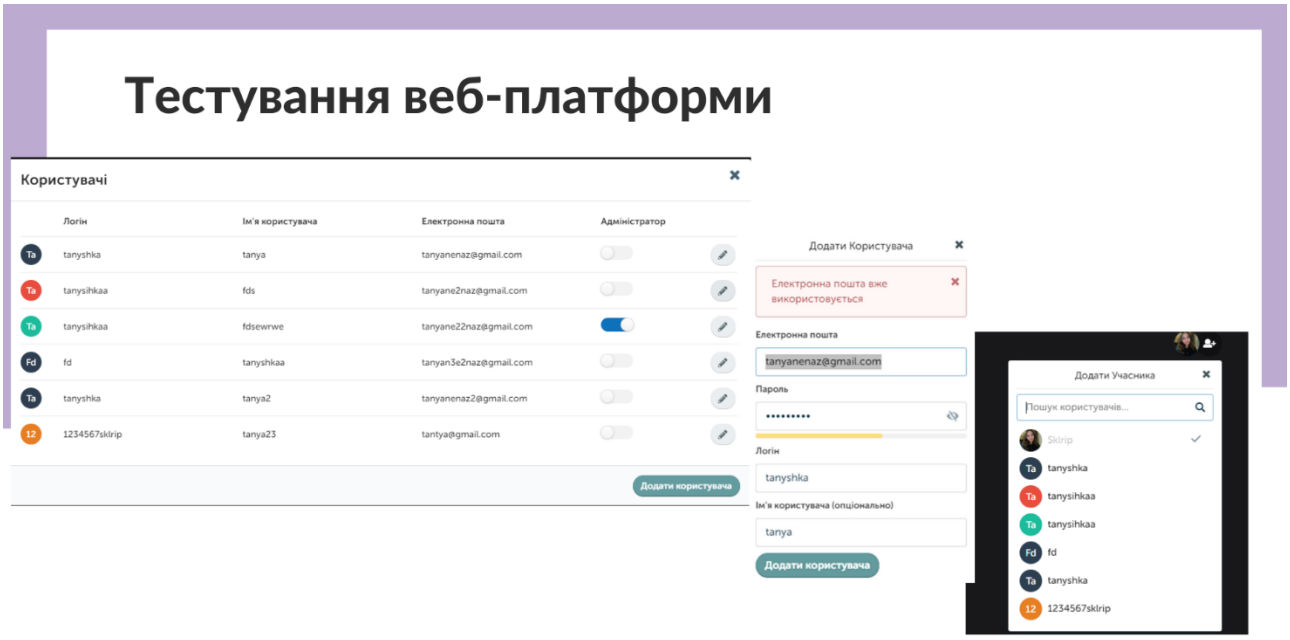

Рисунок Г.15 – Тестування веб-платформи

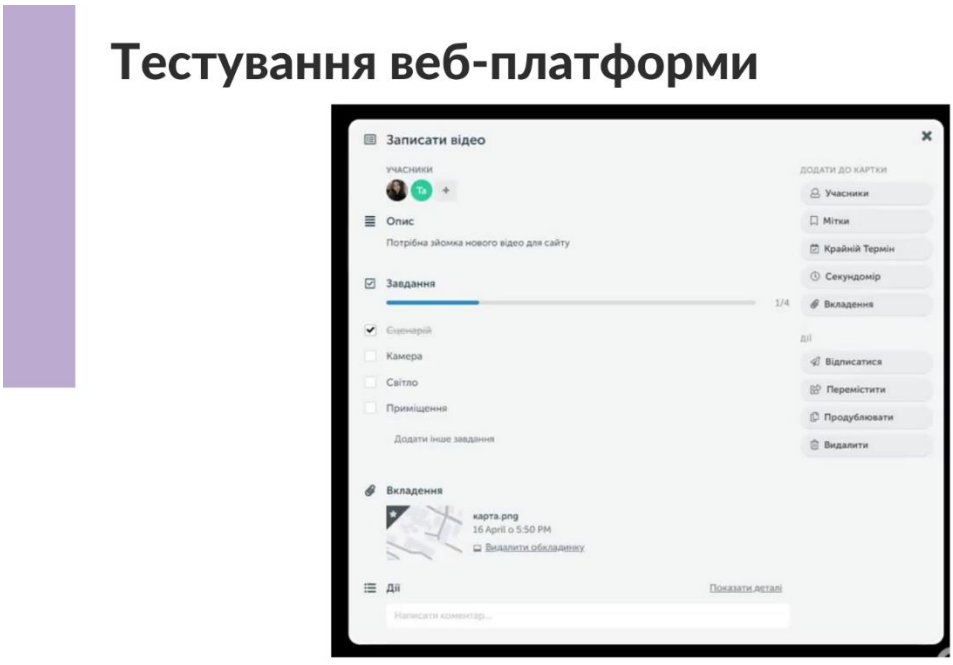

Рисунок Г.16 – Тестування веб-платформи

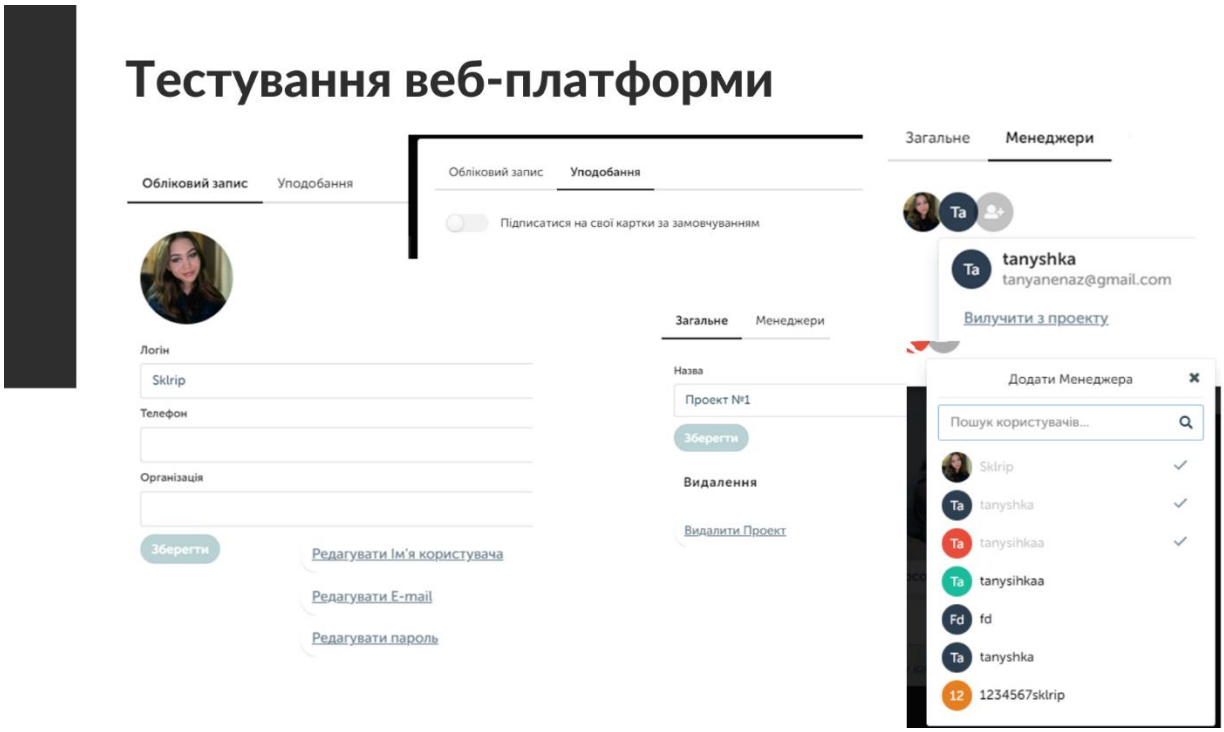

Рисунок Г.17 – Тестування веб-платформи

Апробація та публікація результатів роботи

Результати досліджень доповідалися на:

Міжнародній науково-практичній конференції «Сучасні тенденції розвитку фінансових та інноваційно-інвестиційних процесів в Україні (2024)».

Основні результати досліджень опубліковано в 1 науковій праці у збірниках матеріалів конференції.

Назаренко Т.Є. Розробка інноваційних технологій для розвитку сучасних робочих просторів / Т.Є. Назаренко, Катєльніков Д.І. // Матеріали VII Міжнародної науково-практичної конференції Сучасні тенденції розвитку фінансових та інноваційно-інвестиційних процесів в Україні, Вінниця, 2024. URL: https://conferences.vntu.edu.ua/index.php/fiip/fiip2024/paper/view/20162/16686

Рисунок Г.18 – Апробація та публікація результатів роботи

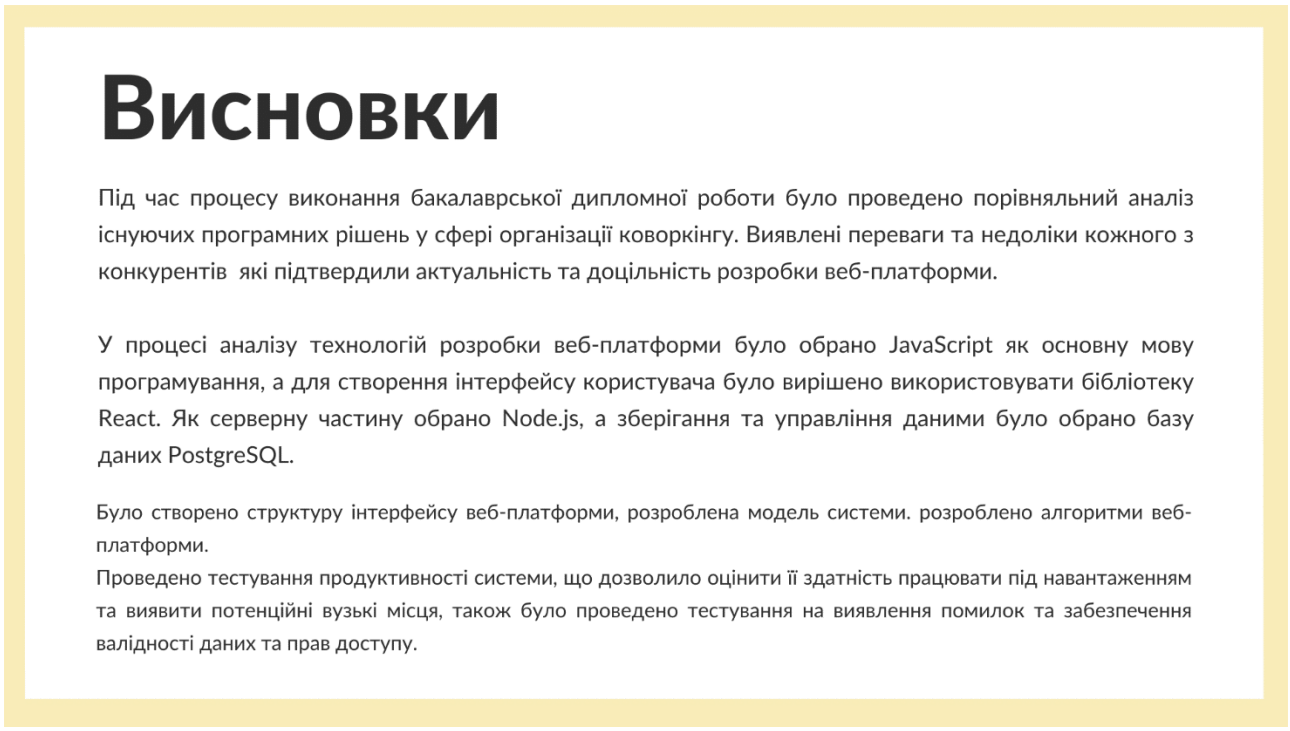

Рисунок Г.19 – Висновки

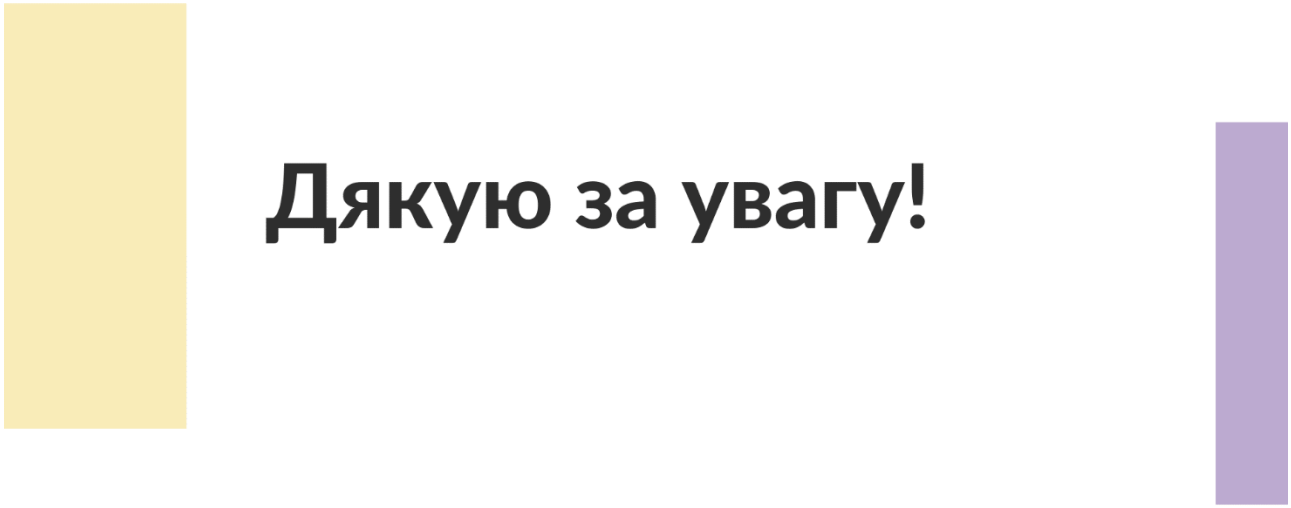

Рисунок Г.20 – Фінальний слайд