НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інформаційних технологій

БАКАЛАВРСЬКА РОБОТА

на тему:

«ІНФОРМАЦІЙНА СИСТЕМА УПРАВЛІННЯ ЗАДАЧАМИ»

Виконав: студент 4 курсу
Рівень бакалавр
Галузь знань 12 «Інформаційні технології»,
спеціальність 122 «Комп’ютерні науки»
Вдовін Олександр Сергійович
Керівник: к.т.н., доцент
Манаков Сергій Юрійович
Рецензент: к.пед.н., доцент
Задерейко Олександр Владиславович

Одеса – 2024

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»

Факультет **кібербезпеки та інформаційних технологій**

Кафедра **інформаційних технологій**

Галузь знань: **12 Інформаційні технології**

Спеціальність: **122 Комп'ютерні науки**

Назва освітньої програми: **Комп'ютерні науки**

ЗАТВЕРДЖУЮ

Завідувач кафедри інформаційних технологій

доцент Н.І. Логінова

_____ «____ »_____ 20__ року

З А В Д А Н Н Я

на кваліфікаційну (бакалаврську) роботу

здобувачу Вдовіну Олександру Сергійовичу

1. Тема роботи: «Інформаційна система управління задачами»

Керівник роботи: к.т.н., доцент Манаков Сергій Юрійович

затверджені наказом Ректора НУ «ОЮА» від «02» квітня 2024 року № 809-06.

2. Строк подання здобувачем роботи «20» травня 2024 року

3. Дата видачі завдання «15» вересня 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі. Аналіз існуючих підходів до вирішення поставлених завдань.	01.12.23	
2	Опис розроблених алгоритмів та програмного забезпечення	26.01.24	
3	Результати експериментів та аналіз отриманих даних	15.03.24	
4	Оформлення пояснювальної записки	17.05.24	

Здобувач _____

(підпис) (прізвище та ініціали)

Керівник роботи _____

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Вдовін О.С. Інформаційна система управління задачами. – Кваліфікаційна робота бакалавра за спеціальністю 122 «Комп'ютерні науки».

Кваліфікаційна робота викладена на 80 сторінках, містить 3 розділи, 70 ілюстрацій, 9 таблиць та 10 використаних джерел.

Об'єктом роботи є інформаційна система для управління задачами.

Предмет роботи – процес розробки інформаційної системи управління задачами.

Метою роботи є розробка інформаційної системи управління задачами.

Метод дослідження – прикладний.

У першому розділі сформульовано цілі та задачі роботи. У другому розділі здійснено опис розроблених алгоритмів та програмного забезпечення інформаційної системи. У третьому розділі приводиться результати експериментів та аналіз отриманих даних.

За результатами роботи зроблено висновки про отримані результати та потенціал її подальшого розвитку.

Ключові слова: Управління задачами, JavaScript, Node.js, VueJs, база даних.

ABSTRACT

Vdovin O.S. Task management information system. – Bachelor's qualifying work in specialty 122 "Computer Science".

The qualification work is laid out on 80 pages, contains 3 chapters, 70 illustrations, 9 tables and 10 used sources.

The object of the work is an information system for task management.

The subject of the work is the process of developing the task management information system.

The purpose of the work is to develop an information system for task management.

The research method is applied.

In the first chapter, the goals and tasks of the work are formulated. The second chapter describes the developed algorithms and information system software. The third section presents the results of the experiments and the analysis of the obtained data.

Based on the results of the work, conclusions were made about the obtained results and the potential for its further development.

Keywords: Task management, JavaScript, Node.js, VueJs, database.

ЗМІСТ

ВСТУП

Інструмент управління задачами — це просте рішення, яке допомагає відстежувати невеликі, великі завдання та різні проекти. Ви можете використовувати його як окремо, так і як систему управління роботою команди.
Існують різні типи інструментів управління завданнями, найпопулярніші серед яких:

- «Паперовий список» із завданнями у вигляді календаря або простий, персоналізований список справ із термінами;
- Застосунок для управління завданнями з повністю настроюваними, вичерпними списками справ із додатковими функціями;
- Інтернет-менеджер завдань з основними функціями та простими списками справ;
- Інструмент управління проектами з функціями, які дозволяють контролювати та контролювати весь життєвий цикл проекту та окремі завдання;
- Дошки Kanban, які допомагають візуалізувати структуру проектів та організувати робочий процес.

Використання таких додатків має багато переваг. Це допомагає організовувати та планувати роботу, постійно інформувати про завдання чи хід проекту, оптимізувати процеси та автоматизувати роботу.
Які саме переваги дає використання таск-менеджера у команді проекту:

1. Систематизація процесів: розбити проект на завдання, відзначати проходження завдання по етапах.
2. Планування роботи: в календарі, канбан-дошках або діаграмі Ганта.
3. Відстежування результатів: врахування робочого часу, визначення ресурсів, витрачених на проект.
4. Зберігання файлів і збирання інформації.
5. Пов'язування проектного менеджера та команду з клієнтом .

6. Об'єднання каналів зв'язку, щоб все спілкування і вирішення по роботі записувалися в одну систему.

В даній кваліфікаційній роботі буде розроблене рішення для спільної роботи невеликої команди над проектами. В основу системи полягає парадигма для керування проектами «канбан»

Об’єктом роботи є інформаційна система для управління задачами.

Предметом роботи є процес розробки інформаційної системи управління задачами.

Метою роботи є розробка інформаційної системи управління задачами.

Методика дослідження – прикладна.

Практична значущість результатів дослідження: Можливість практичного використання за призначенням.

Розділ 1. ПОСТАНОВКА ЗАДАЧІ

1.1 Формулювання цілей та завдань дослідження

Розповсюдження інформаційних технологій призвели до того, що кількість задач середньостатистичної людини значно виросла відносно тої, що була ще на початку десятиліття. Нерідко від нестачі часу деякі задачі можуть загубитися, що може потягнути за собою неприємні наслідки.

Призначення даної інформаційної системи надати можливість контролювати будь які задачі за власними потребами, структуруючи їх за логічними правилами та ділитися задачами із третіми особами для організації спільної роботи.

За даними Google Trends можна побачити, що динаміка популярності за запитом «Task Management» за останні п’ять років утримується на високому рівні, що може свідчити про високий інтерес користувачів до цієї тематики та високого попиту на програмне забезпечення у цій області.

Метою даної роботи є розробка такої інформаційної системи для керування задачами, що вирішить проблеми цільової аудиторії, пов’язані із плануванням задач та організацією спільного доступу до них.

В результаті аналізу конкурентних рішень, представлених на ринку, можна сформувати вимоги до інформаційної системи, ураховуючи їх переваги та недоліки.

Для того, щоб спроектувати ефективну інформаційну систему, потрібно вирішити наступні задачі:

- Провести аналіз існуючих конкурентних програмних рішень у даній предметній області;

– Сформувати вимоги до інформаційної системи керування задачами, базуючись на результатах порівняльного аналізу та дослідження цільової авдиторії;
– Спроектувати базу даних, яка задовільнить вимогам застосунку;
– Обгрунтувати вибір технологічних рішень для розробки;
– Розробити архітектуру та бізнес-логіку інформаційної системи;
– Розробити програмне забезпечення;
– Провести тестування програмного забезпечення.

1.2. Аналіз існуючих підходів до вирішення поставлених завдань

У результаті аналізу інформаційних систем, наявних на ринку було виявлено високу конкуренцію у даній предметній області. Під час аналізу особлива увага була звернута на спосіб відображення завдань, зручність користування та функціональні можливості конкурентних сервісів.
Першим розглянуто web-застосунок Asana. Основний вигляд інтерфейсу застосунку наведено на рисунку 1.1

.

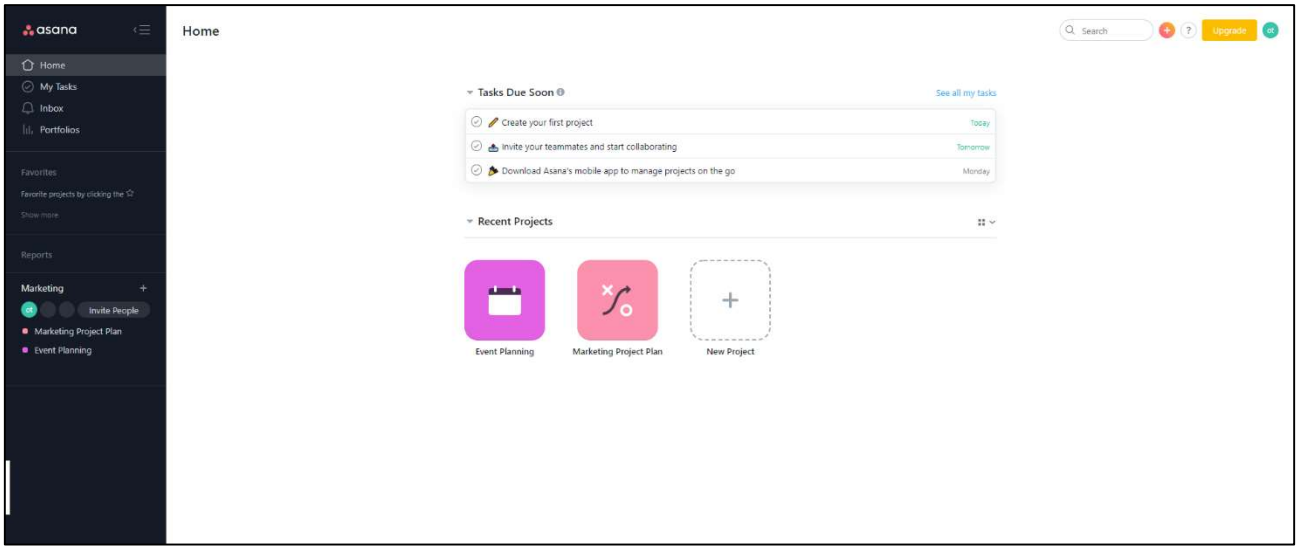

Рисунок 1.1 – Інтерфейс вебзастосунку Asana

Цей застосунок для управління проектами було розроблено колишніми членами команди Facebook і позиціонується він як ефективний інструмент для малих або середніх компаній. У той же час його можливості були оцінені багатьма великими організаціями.

Головною перевагою Asana є її великий набір інструментів. Більше того, функціонал дуже різноманітний, і ним можуть користуватися як звичайні виконавці, так і менеджери проектів, а також керівництво вищого рівня.

Однак у цієї програми є свої недоліки для управління проектами. Перш за все, це необхідність постійно вносити зміни: встановлювати нові статуси завдань, коригувати пріоритет їх виконання.

Недоліком є представлення завдань в одному дереві. З одного боку, така структура максимально чітка, вона дозволяє побачити всю кількість майбутніх робіт. У той же час, менші завдання було б зручніше відображати у вигляді ToDo-аркуша.

Кожна команда має можливість створити для себе робочий простір. Кожен робочий простір може включати в себе проекти, а кожен проект, в свою чергу, задачі.

Робочий простір зображено на рисунку 1.2.

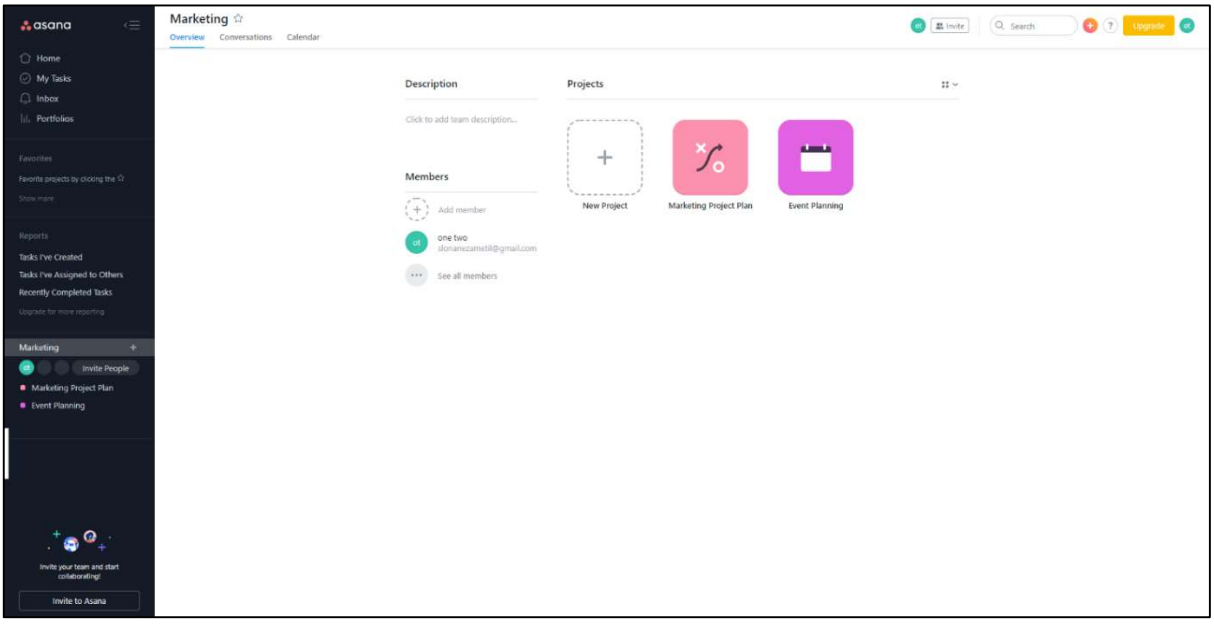

Рисунок 1.2 – Робочий простір команди у Asana

Користувачі мають можливість додавати нові задачі, додавати коментарі до існуючих задач та прикріпляти файли.

Кожна задача відображається у вигляді картки, яка містить всю необхідну інформацію, коментарі та функціонал для керування задачею. Кожен користувач має можливість залишити коментар під задачею.

Вигляд картки у web-застосунку Asana зображено на рисунку 1.3.

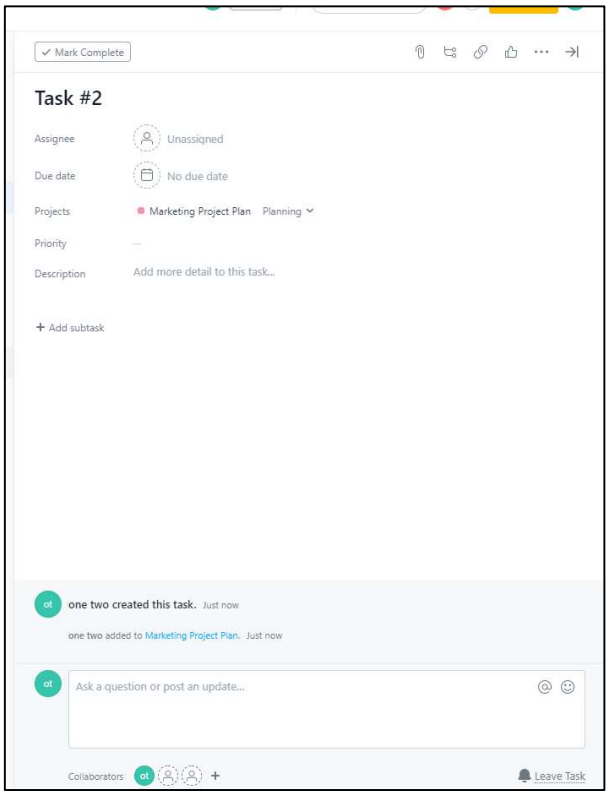

Рисунок 1.3 – Перегляд задачі у Asana

Також можна переглянути задачі у вигляді календаря. Це може бути зручно для точнішого відстеження дедлайнів.

Відображення задач у вигляді календаря продемонстровано на рисунку 1.4.

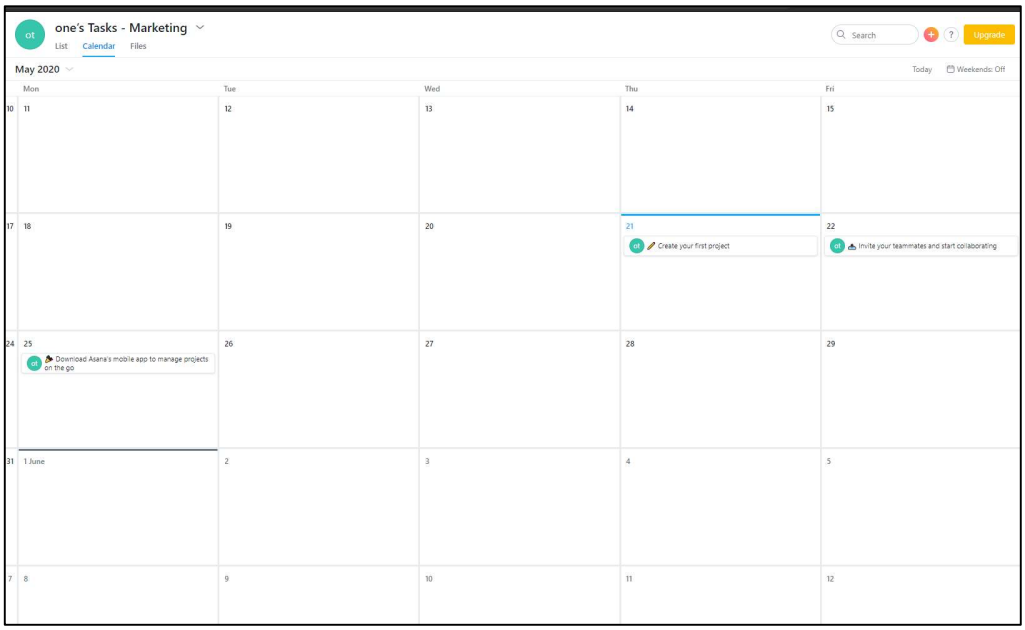

Рисунок 1.4 – Перегляд задач у виді календаря у Asana

Наступним застосунком який розглянуто, є Trello. Ця програма – це електронна дошка із завданнями, де вони розподіляються залежно від стану по різним стовпцям, наприклад, «Треба виконати», «Виконується», «Завершено». У процесі виконання завдання картка переходить від першого стовпця до останнього. Ця методологія називається Kanban, вперше вона була застосована на заводах Toyota, там вона дозволяла контролювати поставку компонентів. Згодом принципи Канбана були універсалізовані та адаптовані для застосування в різних областях. Вони також підходять для управління проектами, про що свідчить служба Trello.

Однак, за всієї простоти використання, інструменти Kanban також мають свої недоліки:

– неможливо оцінити кількість витрачених завдань та час, який залишився для їх виконання;
– важкий у використанні для відносно великих команд;
– неможливо здійснити довгострокове планування.

Звичайно, у Trello є і свої переваги, які визначають популярність цієї послуги для управління проектами. До таких переваг можна віднести можливість

групувати завдання за допомогою кольорових ярликів, додавати коментарі та примітки до кожного завдання, додавати файли та посилання на них. Крім того, існує дуже зручна система спільної роботи та призначення відповідальних за виконання завдання.

Інтерфейс Trello зображено на рисунку 1.5.

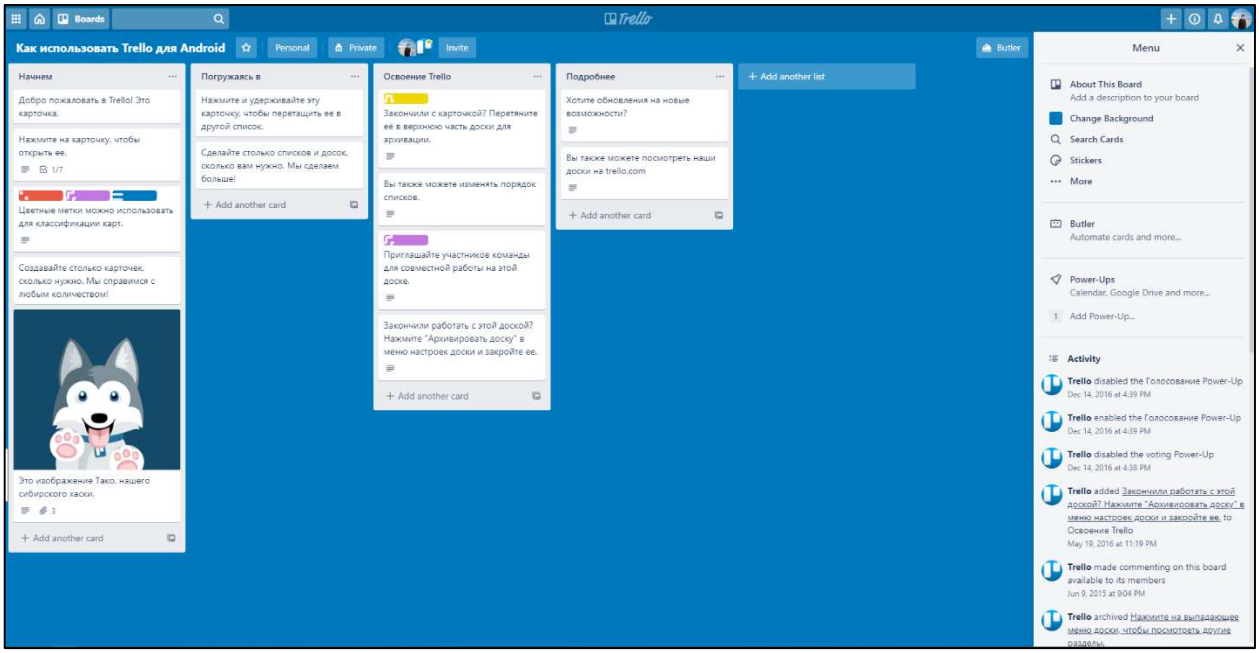

Рисунок 1.5 – Інтерфейс Trello

По аналогії із Asana у Trello можна створювати проекти. Тут вони мають назву «Дошки». Кожна дошка має в собі списки, які мають в собі картки. Кожна картка – це і є окрема задача. Вигляд картки Trello зображено на рисунку 1.6.

Функціонал Trello може бути розширений за допомогою «Покращень», в яких представлено безліч плагінів та інтеграційних модулів для покращення роботи із системою. Усі покращення представлені у вбудованому маркетплейсі.

Вигляд маркетплейсу продемонстровано на рисунку 1.7.

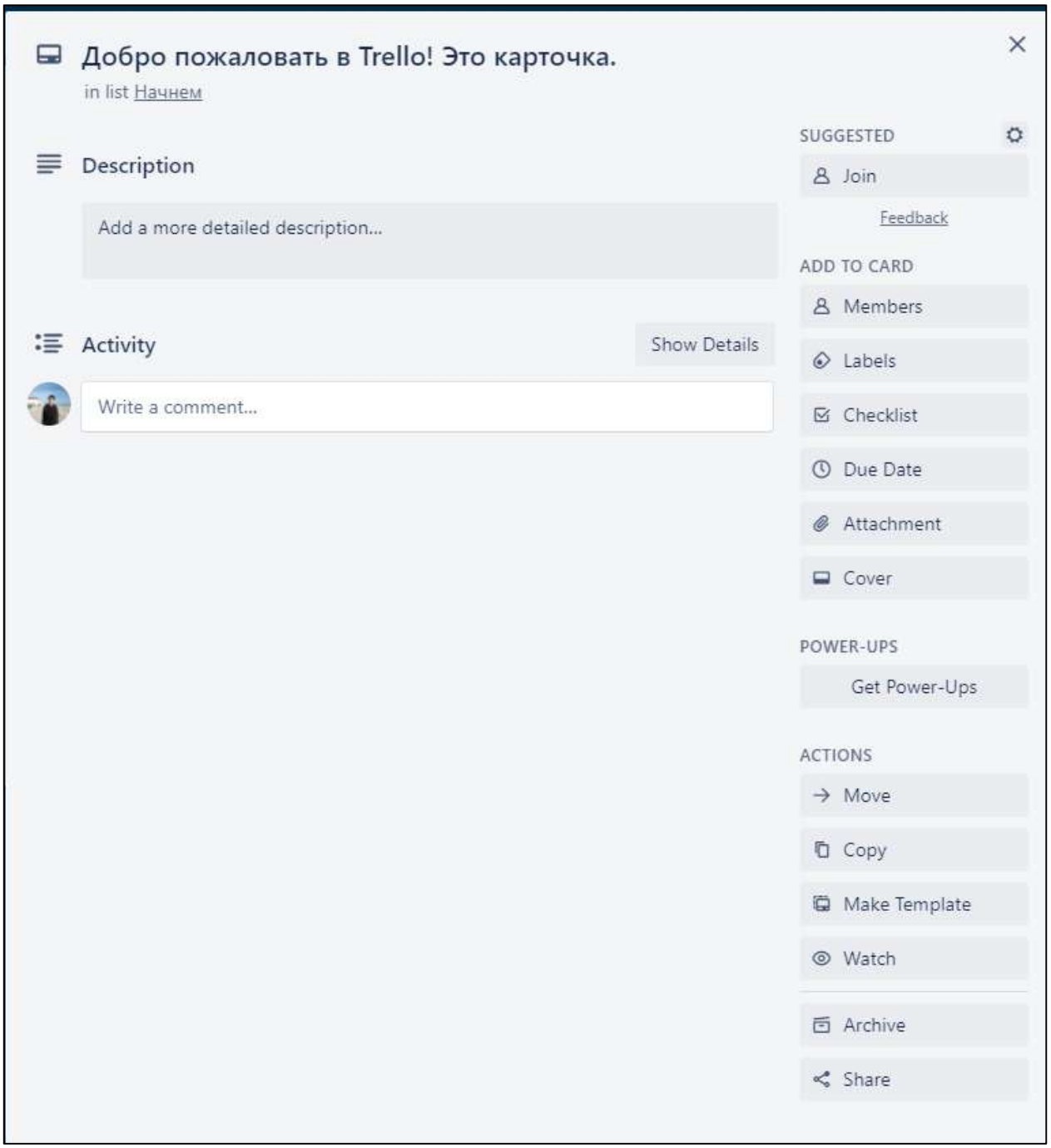

Рисунок 1.6 – Перегляд картки в Trello

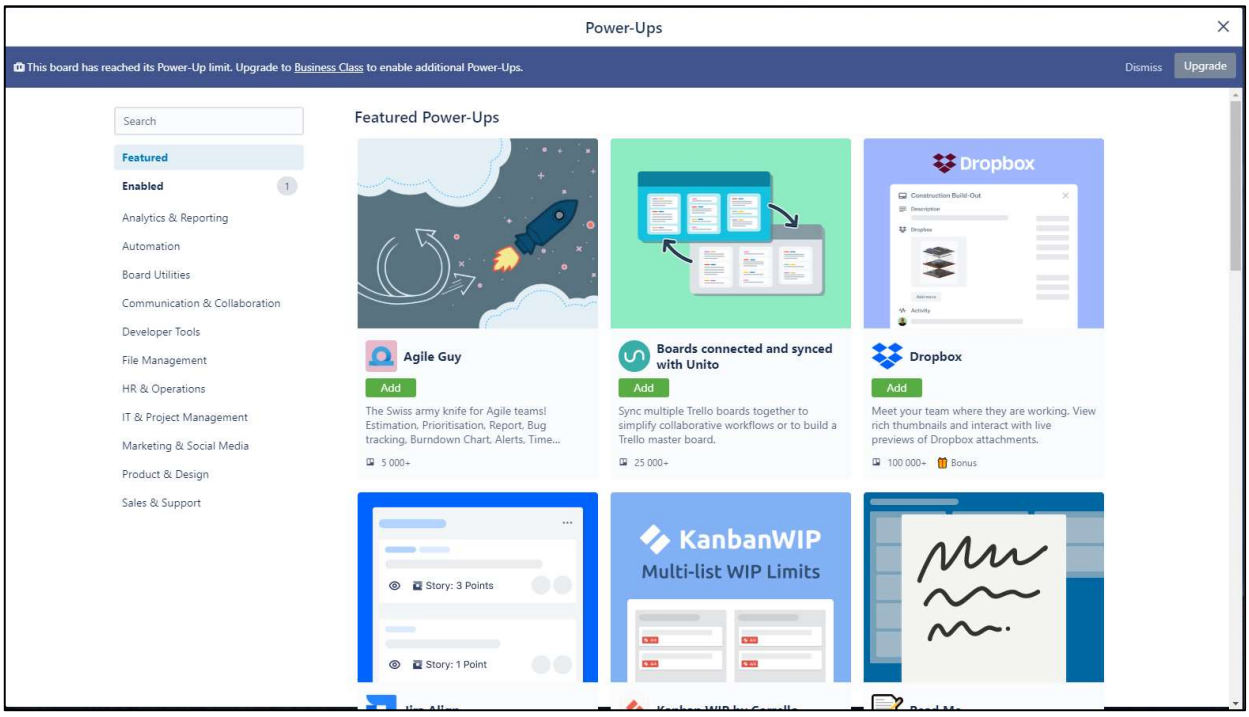

Рисунок 1.7 – Маркетплейс покращень Trello

Третій застосунок, який буде розглянуто, це Microsoft To Do. Центральним елементом є інструмент «My day». Він дозволяє додавати задачі із загального списку до власного розкладу дня.

Зовнішній вигляд застосунку зображено на рисунку 1.8.

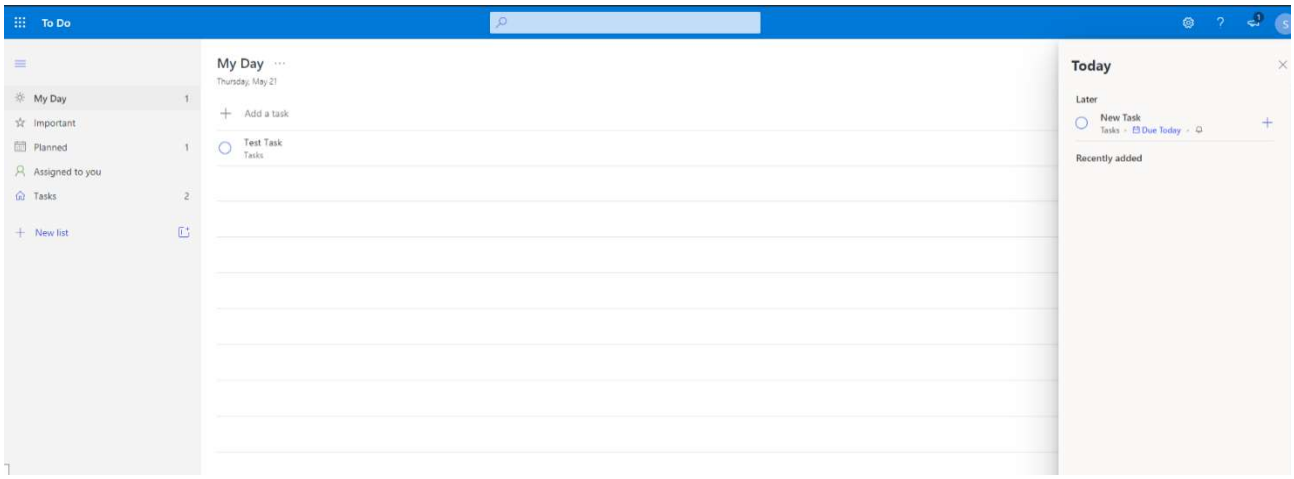

Рисунок 1.8 – Зовнішній вигляд Microsoft To-Do

До кожної задачі можна додавати чек-ліст, кінцевий термін, встановлювати нагадування та обрати одну або декілька категорій, до якої належить дана задача. Меню управління задачею продемонстровано на рисунку 1.9.

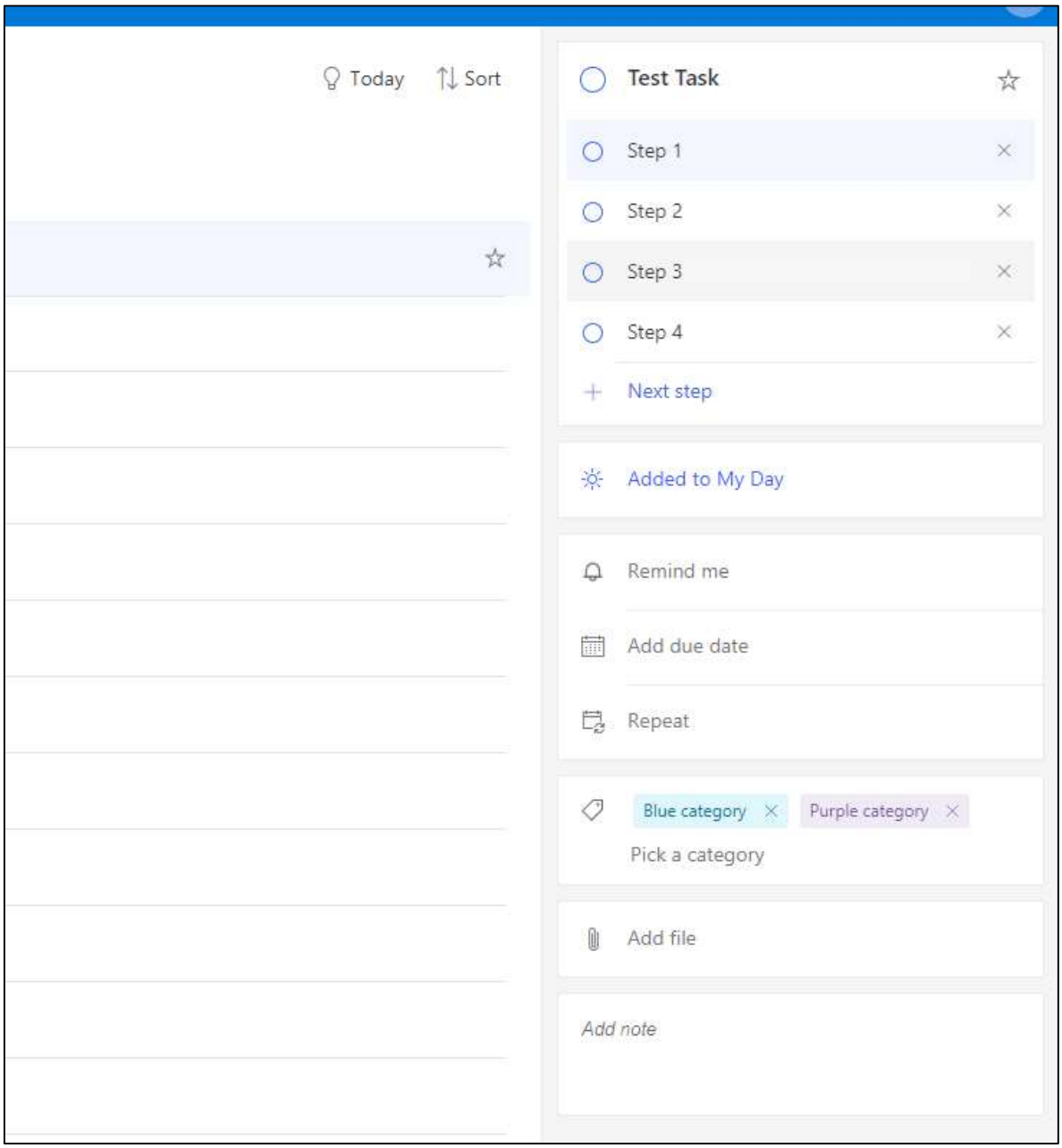

Рисунок 1.9 – Налаштування задачі

Виконаємо порівняльний аналіз функціональних можливостей оглянутих програмних рішень. Такий аналіз дозволить краще побачити слабкі місця кожного програмного застосунку та врахувати їх під час розробки вимог до програмного забезпечення.

В таблиці 1.1 наведено результати порівняльного аналізу. У стовбцях таблиці розташовані назви розглянутих інформаційних систем, а у строках таблиці – обрані функціональні можливості. Знаком «плюс» відзначено наявність зазначеного функціоналу у системі, а знаком «мінус» його відсутність.

Таблиця 1.1 – Порівняльний аналіз обраних інформаційних систем

Функціональна можливість	**Asana**	**Trello**	**Microsoft To-Do**
Керування задачами	+	+	+
Спільне використання	+	+	-
Можливість працювати над декількома проектами одночасно	+	+	+
Створення To-Do листів	+	+	+
Створення вкладених задач	+	+	-
Перегляд завдань у вигляді власного розкладу	+	-	+
Представлення задач у вигляді канбану	+	+	-

Аналіз, наведений у табл. 1.1, дає змогу стверджувати, що всі розглянуті системи мають достатнє функціональне наповнення, але не можуть повністю задовольнити потреби користувача.

1.3 Опис очікуваних результатів

Для розробки вимог до інформаційної системи скористаємось методом «Модель Кано». Дани метод був створений доктором Норіакі Кано, професором управління якістю в Токійському університеті наук у 1984 році. Доктор Норіакі розробив цю модель, досліджуючи фактори, що сприяли задоволенню та лояльності клієнтів.

Модель визначає п’ять категорій потенційних реакцій клієнтів на нову особливість, починаючи від невдоволення до захоплення клієнтів.

Використовуючи модель Кано, команди проектів складають список можливих функцій, що змагаються за ресурси розвитку та простір у дорожній карті. Потім команда зважить ці функції відповідно до двох конкуруючих критеріїв:

– Їх потенціал задовольнити клієнтів;

– Інвестиції, необхідні для їх реалізації.

Модель Kano визначає п’ять категорій функцій, які команди бажають розвивати:

– Must-be Quality (Обов’язкові)

– One-Dimensional Quality (Одновимірні)

– Attractive Quality (Привабливі)

– Indifferent Quality (Неважливі)

– Reverse Quality (Небажані)

Перед початком розробки застосунку було проведення опитування цільової аудиторії проекту. Перелік питань та відповідей на них наведено у Додатку А. Функціонал, що належить до категорій Must-be або Attractive буде реалізований у першу чергу.

Результати опитування представленні у таблиці 1.2.

За результатами опитування серед цільової аудиторії очікуваний функціонал застосунку був розділений на три категорії, що наведені у таблиці 1.3.

Таблиця 1.2 – Модель Кано

Атрибут	M	O	A	I	R	Категорія
Створення декількох проектів	14	2	11	3	0	Must-be
Створення списків для розподілу задач	14	1	11	4	0	Must-be
Керування задачами	25	0	3	2	0	Must-be
Встановлення дедлайну заадачі	14	3	10	3	0	Must-be
Спільне використання	17	0	5	8	0	Must-be
Керування користувачами	15	1	8	6	0	Must-be
Створення To-Do листів	7	2	9	12	0	Indifferent
Створення вкладених задач	8	4	12	6	0	Attractive
Перегляд завдань у вигляді власного розкладу	8	2	11	9	0	Attractive
Представлення задач у вигляді канбану	15	0	10	5	0	Must-be

Таблиця 1.3 — Очікуваний функціонал розподілений на три категорії

Мають бути	Привабливі	Неважливі
Створення декількох проектів	Створення вкладених задач	Створення To-Do листів
Створення списків для розподілу задач	Перегляд завдань у вигляді власного розкладу	
Керування задачами		
Встановлення дедлайну заадачі		
Спільне використання		
Керування користувачами		
Представлення задач у вигляді канбану		

Розділ 2. ОПИС РОЗРОБЛЕНИХ АЛГОРИТМІВ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Огляд використовуваних технологій та інструментів

Для проектування Backend частини інформаційної системи було обрано мову програмування JavaScript. Спочатку JavaScript був створений для того, щоб «Оживити» веб-сторінки. Сьогодні він може виконуватися не тільки в браузері, а й на сервері або на будь-якому іншому пристрої, який має спеціальну програму, що називається «движком» JavaScript.

Node.js – це середовище виконання JavaScript на стороні сервера, що побудовано на JavaScript-рушію V8, розроблений Google та використовується для побудови швидких, масштабованих мережних додатків. В основі Node.js полягає асинхронна подієво-орієнтована модель з неблокуючими операціями введення та виведення, що робить її легкою та ефективною.

JavaScript-рушій – це програма, або, іншими словами, інтерпретатор, що виконує код, написаний на JavaScript. Движок може бути реалізований з використанням різних підходів: у вигляді звичайного інтерпретатора, у вигляді динамічного компілятора (або JIT-компілятора), який, перед виконанням програми, перетворює вихідний код на JS в байт-код якогось формату.

Рушій з відкритим кодом V8 був створений компанією Google, він написаний на C ++. При проектуванні V8 розробники поставили собі за мету поліпшити продуктивність JavaScript. Для того, щоб домогтися високої швидкості виконання програм, V8 транслює JS-код в більш ефективний машинний код, не використовуючи інтерпретатор. Движок компілює JavaScript-код в машинні інструкції в ході виконання програми, реалізуючи механізм динамічної компіляції, як і багато сучасних JavaScript-рушіїв. Основна відмінність полягає в тому, що V8 не використовує при виконанні JS-програм байт-код або будь-який проміжний код.

Іншими словами, Node.js дозволяє писати дуже продуктивний серверний код з використанням JavaScript.

Основні переваги використання Node.js:

– Зручний для побудови швидких додатків, оскільки Node здатний обробляти величезну кількість одночасних з'єднань з високою пропускною здатністю.

– Ефективність. У веб-застосунку найбільше часу потрібно, щоб виконати всі запити до бази даних. За допомогою Node.js ми можемо виконати всі їх одразу, зменшуючи час відгуку для повільних запитів;

– Мова розробки JavaScript. Ми маємо можливість розділити код між frontend і backend.

– Швидкість виконання. Двигун V8 постійно розширює межі і є одним з найшвидших інтерпретаторів динамічних мов на планеті. Крім того, засоби введення / виведення Node мають дійсно легку вагу, в результаті чого досягається настільки повне використання потенціалу системи введення / виводу, наскільки це можливо.

Для розробки frontend частини також був обрана мова JavaScript. JavaScript - мультіпарадігменна мова програмування, що підтримує об'єктно-орієнтований, імперативний і функціональний стилі. Є реалізацією стандарту ECMAScript.

Також було обрано фреймворк для роботи із JavaScript — VueJs. Vue — це прогресивний фреймворк для створення користувацьких інтерфейсів. На відміну від фреймворків-монолітів, Vue створений придатним для поступового впровадження. Його ядро в першу чергу вирішує завдання рівня представлення (view), що спрощує інтеграцію з іншими бібліотеками та існуючими проектами. З іншого боку, Vue повністю підходить і для створення складних односторінкових додатків (SPA, Single-Page Applications), якщо використовувати його спільно з сучасними інструментами та додатковими бібліотеками.

Для проектування системи було обрано патерн MVC. Дана концепція була описана ще у 1979 році Трюгве Реенскаугом, що працював над язиком

програмування Smalltalk в Xerox PARC. Розшифровується MVC як «Model-View-Controller», що можна перекласти як «Модель-Представлення-Контролер». В оригінальній концепції була описана ідея та роль кожного елементу моделі, представлення та контролеру. Але зв'язки між ними були описані не конкретизовано. Також відрізняли два типи моделі:

– Активна модель – сповіщує про зміни, а представлення підписуються на ці сповіщення. Це дозволяє зберегти незалежність моделі від представлення та від контролера.

– Пасивна модель – не має можливості взаємодії із представленнями та контролерами, а використовується ними як джерело даних.

Основна мета застосування даної моделі лежить у тому, щоб розділити бізнес-логіку від її візуалізації. Завдяки такому розділенню підвищується можливість її повторного використання. Це ефективно у випадку, якщо користувачу потрібно отримати доступ до одних і тих самих даних у різних місцях або контекстах.

Архітектура розроблюваної інформаційної системи представлена на рисунку 2.1.

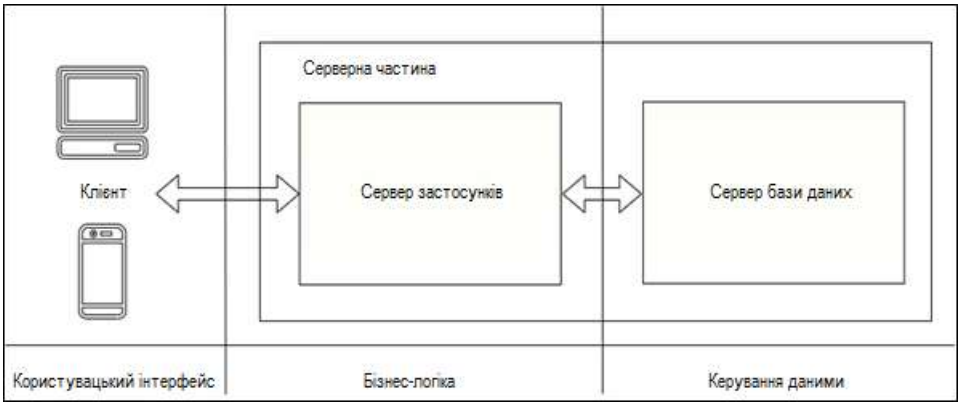

Рисунок 2.1 – Архітектура розроблюваної інформаційної системи

Взаємодія між Frontend та Backend частинами відповідає принципам REST API. REST або RESTful – це загальні принципи організації взаємодії застосунка з сервером за допомогою протоколу HTTP.

HTTP – це протокол, що дозволяє отримувати різні ресурси, наприклад HTML-документи. Протокол HTTP лежить в основі обміну даними в Інтернеті. HTTP є протоколом клієнт-серверної взаємодії, що означає ініціювання запитів до сервера самим одержувачем. Отриманий підсумковий документ буде складатися з різних піддокументів, що є частиною підсумкового документа: наприклад, з окремо отриманого тексту, опису структури документа, зображень, відео-файлів, скриптів і багато чого іншого.

Особливість REST полягає у тому, що сервер не запам'ятовує дані про користувача між запитами. В кожному запиті передається інформація про клієнта, наприклад, ідентифікатор сессії. Взаємодія із сервером відбувається за допомогою чотирьох операцій:

- Отримання даних;
- Додавання даних;
- Модифікація даних;
- Видалення даних.

У якості системи управління базами даних було обрано PostgreSQL. PostgreSql – вільна об'єктно-реляційна система управління базами даних, що базується на мові SQL. Однією з сильних сторін PostgreSQL є її архітектура. Як і багато комерційних СУБД, PostgreSQL може застосовуватися в клієнт-серверному середовищі, що дає масу переваг як користувачам, так і розробникам. Основою PostgreSQL є серверний процес бази даних. Він виконується на одному сервері.

Доступ з додатків до даних бази здійснюється за допомогою процесу бази даних. Клієнтські програми не можуть отримати доступ до даних самостійно, навіть якщо вони працюють на тому ж комп'ютері, на якому виконується серверний процес. Такий поділ клієнтів і сервера дозволяє побудувати розподілену систему.

Завдяки тому, що маніпулювання даними зосереджено на сервері, СУБД не доводиться контролювати численних клієнтів, які отримують доступ в спільно

використовуваний каталог сервера, і PostgreSQL може підтримувати цілісність даних навіть при одночасному доступі великої кількості користувачів.

2.2 Розробка логічної структури бази даних

Логічна структура була розроблена на основі попередньо проведеного аналізу акторів системи. Логічну структуру зображено на рисунку 2.2.

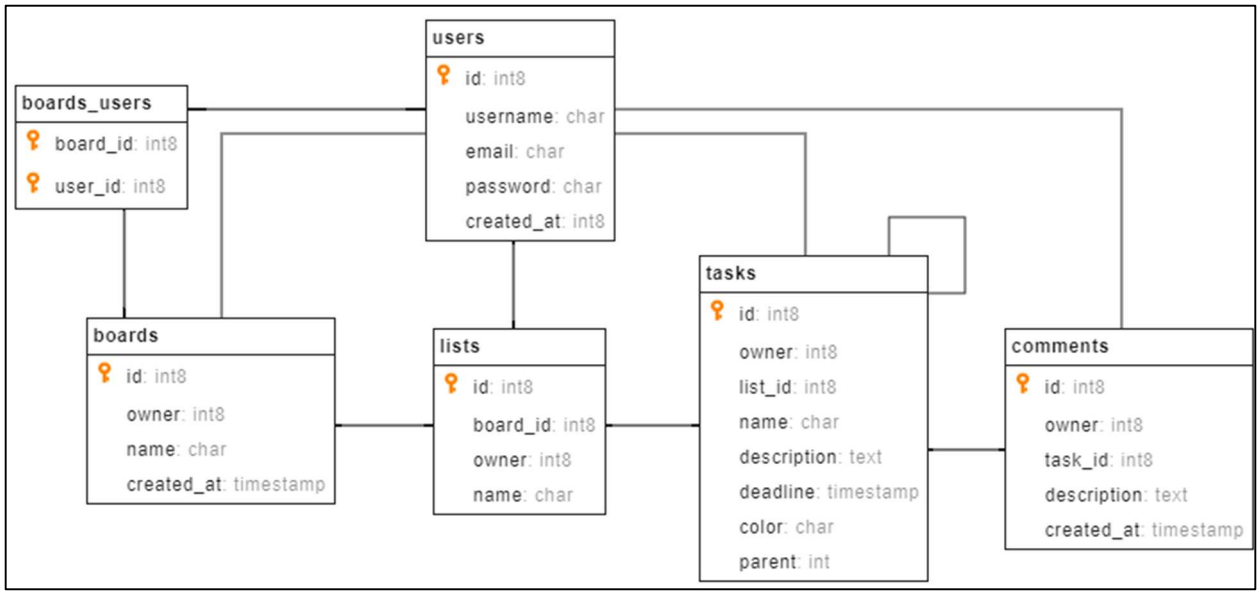

Рисунок 2.2 – Логічна структура бази даних

Перед початком проектування були визначені сутності бази даних інформаційної системи:

- Користувачі (Users)
- дошки (Boards);
- списки (Lists);
- задачі (Tasks);
- коментарі (Comments).

Сутність «Користувачі» зберігає в собі інформацію про кожного зареєстрованого користувача системи. Атрибути «username» та «email» даної сутності повинні

бути унікальними. Поля «email» та «password» використовуються для авторизації користувача у системі. Перелік атрибутів даної сутності наведений у таблиці 3.1.

Таблиця 2.1 – Опис атрибутів сутності «Користувачі»

Атрибут	Опис	Обмеження
id	Унікальний ідентифікатор користувача	PK
username	Юзернейм користувача	N,U
email	Поштова адреса користувача (логін)	N,U
password	Пароль користувача	N
created_at	Дата створення користувача	

Наступна сутність, яку ми розглянемо, це сутність «Дошки». Дана сутність зберігає інформацію про усі дошки у системі. Серед інших атрибутів окремо виділимо атрибут «owner». Даний атрибут являє собою зовнішній ключ та створений для того, щоб поєднати сутність «Дошки» із сутністю «Користувачі». Завдяки цьому система зберігає інформацію про власника дошки та розподіляє права доступу для користувачів.

Повний перелік атрибутів даної сутності представлений у таблиці 2.2.

Таблиця 2.2 – Опис атрибутів сутності «Дошки»

Атрибут	Опис	Обмеження
id	Унікальний ідентифікатор дошки	PK
name	Назва дошки	N
owner	Унікальний ідентифікатор власника дошки	N, FK
created_at	Дата створення дошки	

Сутність «Списки», по аналогії із попередньо розглянутими сутностями, зберігає в собі повну інформацію про списки наявні у системі: їх назву та розположення у системі. Атрибут «board_id» являє собою зовнішній ключ, що поєднує дану сутність із сутністю «Дошки». Таким чином система зберігає інформацію про те, якій дошці належить той чи інший список.

Перелік атрибутів та їх обмеження для розглянутої сутності приведено у таблиці 2.3

Таблиця 2.3 – Перелік атрибутів та їх обмеження для сутності «Дошки»

Атрибут	Опис	Обмеження
id	Унікальний ідентифікатор списку	PK
name	Назва списку	N
board_id	Унікальний ідентифікатор дошки	N, FK

Наступна сутність зберігає інформацію про усі задачі наявні у системі. Коротко розглянемо деякі із її атрибутів. Атрибут «id» – унікальний ідентифікатор, первинний ключ даної сутності. Атрибут «name» – ім'я задачі. Атрибут «color» зберігає інформацію про колір, яким виділено задачу. Колір зберігається у форматі HEX. «list_id» – це унікальний ідентифікатор списку, до якого належить задача, відповідно даний атрибут є зовнішнім ключем, який з’єднує між собою «Задачі» та «Списки», що дозволяє отримувати інформацію про те, до якого списку належить задача. Атрибут «owner» також є зовнішнім ключем та поєднує «Задачі» із сутністю «Користувачі» та відповідно до інших сутностей, зберігає інформацію про власника даної задачі, таким чином система може розподілити права доступу користувачів до даної задачі.

Повний перелік усіх атрибутів сутності «Задачі» з їх обмеженнями приведений у таблиці 2.4.

Таблиця 2.4 – Опис атрибутів сутності «Задачі»

Атрибут	**Опис**	**Обмеження**
id	Унікальний ідентифікатор задачі	PK
name	Назва задачі	N
description	Опис задачі	
created	Дата створення задачі	
deadline	Кінцевий срок виконання задачі	
color	Колір задачі	
list_id	Унікальний ідентифікатор списку	N, FK
owner	Унікальний ідентифікатор власника задачі	N, FK

Сутність «Коментарі» зберігає інформацію про коментарі у системі. Дана сутність має два атрибути, що являють собою зовнішні ключі (таблиця 2.5):

- Атрибут «task_id». Даний ключ поєднує сутність «Коментарі» із сутністю «Задачі» та зберігає інформацію про задачу, до якої належать коментарі.
- Атрибут «owner». Даний ключ поєднує сутність «Коментарі» із сутністю «Користувачі» та зберігає інформацію про власників коментарів.

Таблиця 2.5 – Опис атрибутів сутності «Коментарі»

Атрибут	**Опис**	**Обмеження**
id	Унікальний ідентифікатор коментарю	PK
description	Текст коментарю	N
created	Дата створення коментарю	
task_id	Унікальний ідентифікатор задачі	N, FK
owner	Унікальний ідентифікатор власника коментарю	N, FK

2.3 Задачі користувачів системи

На основі аналізу предметної області та опитування цільової аудиторії на предмет бажаного функціоналу інформаційної системи, було сформовано перелік задач користувачів системи та виділено два типи акторів:

- адміністратор дошки;
- робітник.

Усі користувачі системи мають однакові можливості. Кожен користувач одразу після реєстрації та входу до системи має можливість створювати власні дошки. Після створення дошки користувач отримує рівень доступу адміністратора системи та усі його функції. Усі користувачі, яким наданий загальний доступ до дошки, будуть мати звичайний рівень доступу робітника. Власник дошки має можливість у будь який момент часу передати права на керування дошкою будь якому зареєстрованому користувачу системи, використовуючи для цього лише його Email-адресу.

Після передання права на керування дошкою, попередній власник втрачає права адміністратора дошки та набуває прав робітника.

Адміністратор системи на відміну від робітника має можливість видаляти або редагувати будь яку задачу, список або саму дошку. Робітник у свою чергу має можливість створювати задачі, підзадачі та коментарі. А видаляти та редагувати може лише ті задачі, підзадачі та коментарі, власником яких він являється. Власником задачі користувач може стати не тільки безпосередньо створивши задачу, але й отримавши такі права від поточного власника.

Задачі користувачів детально описані в таблиці 2.6.

Таблиця 2.6 – Задачі користувачів системи

Доступ	**Задача**	**Опис**
Всі	Реєстрація	Будь який користувач інформаційної системи має можливість самостійно створити аккаунт, використовуючи Email та пароль.
Всі	Вхід в систему	Зареєстрований користувач має можливість увійти до системи, використовуючи Email та пароль, вказані при реєстрації у системі.
Всі	Створення дошки	Для створення нової дошки потрібно ввести її ім’я. Ім’я дошки може складатися із будь яких символів. Довжина імені обмежена 60 символами. При створенні дошки користувач отримую права адміністратора цієї дошки.
Адміністратор	Редагування дошки	Редагуючи дошку, можна змінити її назву та змінити залучених співробітників.
Адміністратор	Видалення дошки	Під час видалення дошки видаляються також усі списки та задачі, що присутні на цій дошці.
Адміністратор	Створення списку	Для створення списку потрібно ввести його ім’я. Також можна обрати колір списка.
Адміністратор	Редагування списку	При редагуванні списку можна змінити його ім’я або колір.
Адміністратор	Видалення списку	Під час видалення списку видаляються також усі задачі цього списку.

Продовження таблиці 2.6

Доступ	Задача	Опис
Адміністратор дошки	Видалення списку	Під час видалення списку видаляються також усі задачі цього списку.
Всі	Створення задачі	Для створення задачі потрібно вказати її назву. Також можна вказати дату дедлайну, задати колір картки, додати опис та обрати користувачів, які будуть виконавцями задачі.
Всі	Редагування задачі	Редагуючи задачу, можна змінити її назву, опис, колір, дату дедлайну, виконавців та власника задачі. При зміні власника, новий власник набуде прав адміністрування задачою, в той час як попередній власник втратить таку можливість та буде залучений як виконавець.
Всі	Видалення задачі	При видалені задачі також будут видалені усі підзадачі та коментарі, зв’язані із цією задачею.
Всі	Долучення співробітників до задачі	Щоб долучити співробітника до задачі потрібно обрати його username із випадаючого списку.
Всі	Створення підзадач	До кожної задачі можна додати підзадачі. Для створення підзадачі потрібно перейти у материнську директорію та натиснути кнопку “Add subtask”. Після чого вказати назву підзадачі. Далі з підзадачею можна працювати як зі звичайною задачею.
Всі	Створення комментарів	Для створення коментарю необхідно перейти на сторінку задачі та додати текст коментаря у відповідне поле.
Всі	Видалення коментарів	
Всі	Редагування коментарів	При редагуванні коментарю можна змінити його зміст.

2.4 Use case діаграми користувачів

Функціонал застосунку був поділений на 6 модулів відповідно до їх сутностей.

Першим модулем є модуль доступу. В цьому модулі розділення прав доступу немає, тому актор тут один — це гість. Цей модуль включає в себе дві функції:

- Реєстрація у системі;
- вхід до системи.

Для реєстрації у системі користувачу необхідно ввести Email, пароль та повторити пароль для перевірки. Для того, щоб реєстрація пройшла успішно, мають бути дотримані наступні умови:

- Корректно введений Email. Email повинен бути у форматі «user@mail.com»;
- Email не використовується у системі іншим користувачем;
- Введені паролі співпадають.

В іншому випадку користувач отримає відповідне повідомлення про похибку.

Use-case діаграма модулю даного модулю представлена на рисунку 2.3.

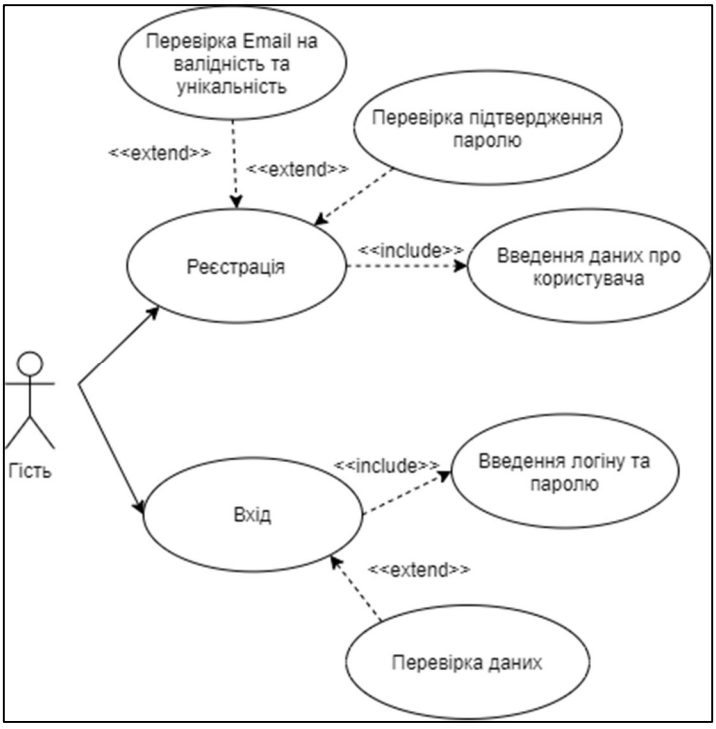

Рисунок 2.3 – Опис модулю «Доступ».

Наступний розглянутий модуль — це модуль «Дошки». На цьому етапі відтворюється розподіл прав користувачів. Будь який користувач може створити дошку. Одразу після створення він набуде прав адміністратора дошки та отримає доступ до таких функцій як редагування та видалення дошки. Редагування включає в себе три функції:

- Зміна назви дошки;
- зміна адміністратора дошки;
- поширення доступу.

Під зміною адміністратору мається на увазі передача прав адміністрування дошкою іншому користувачу системи. Користувач, що отримує права на адміністрування дошки, повинен бути попередньо зареєстрований у системі. Після передачі прав адміністрування дошкою, поточний адміністратор втрачає можливість вносити будь які зміни до дошки.

Під поширенням доступу мається на увазі залучення інших користувачів системи до спільної роботи над дошкою. Для залучення потрібно користувача потрібно натиснути відповідну кнопку у меню. Та ввести його Email у відповідному полі, після чого натиснути кнопку «Add User». Добавлений користувач автоматично набуде прав робітника. Робітник має доступ тільки на перегляд дошки.

Use case діаграма модулю «Дошки» для адміністратора зображена на рисунку 2.4.

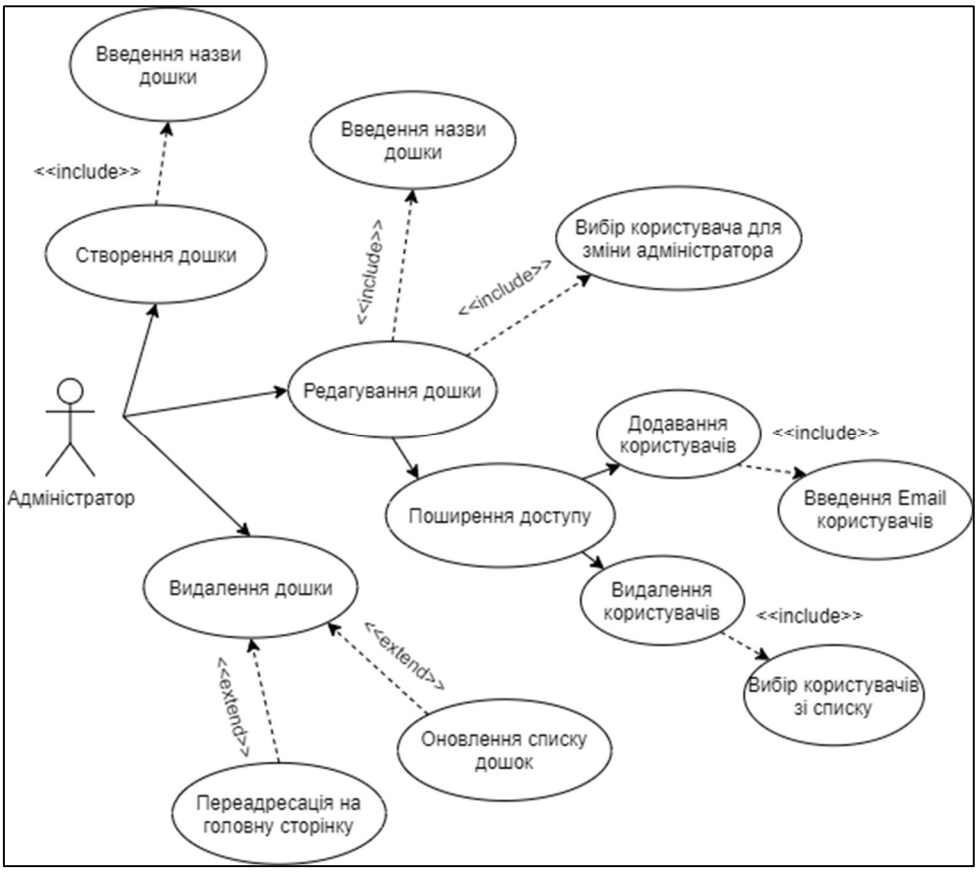

Рисунок 2.4 – Опис модулю «Дошки» для актора «Адміністратор»

Наступним буде розглянуто модуль «Списки». Призначення даного модуля — розподілення задач за логічною структурою. Як саме виконувати розподілення вирішує адміністратор.

Для того, щоб створити новий список, необхідно перейти до бажаної дошки та натиснути кнопку «Add List» та ввести ім'я майбутнього списку. Зробити це може лише адміністратор дошки. При видалені списку виконується каскадне видалення усіх задач у цьому списку. Робітник має доступ тільки на перегляд списків.

Use case діаграма для модулю «Списки» представлена на рисунку 2.5.

Рисунок 2.5 – Опис модулю «Списки» для актора «Адміністратор»

Наступний модуль — «Задачі». Цей модуль найбагатший на функціонал. Створити задачу може будь який користувач системи. Щоб це зробити, необхідно перейти на сторінку дошки та натиснути кнопку «Add Task» у тому списку, до якого плануємо додати задачу, ввести назву задачі та натиснути клавішу «Enter». Задача буде створена та додана до обраного списку.

Якщо перейти на сторінку задачі, до неї можна буде її відредагувати або видалити.

Процес редагування задачі включає в себе наступні функції:

- Зміну назви задачі;
- зміну дати дедлайну;
- зміну кольору задачі;
- завершення задачі;
- зміну списку задачі;
- зміну власника задачі;

– створення підзадачі;
– додання користувачів до задачі.

Щоб перемістити задачу до іншого списку, потрібно перейти на сторінку дошки та методом «Drag and Drop» перетягнути задачу до цільового списку. Одразу після цього інформація буде збережена в БД.

При видаленні задачі автоматично будуть видалені коментарі, пов’язані із нею.

Також користувач має можливість створити підзадачу. Для цього необхідно натиснути кнопку «Add Subtask» та натиснути «Enter». Одразу після цього користувач буде переадресований до щойно створеної підзадачі. Актор «Робітник» має аналогічний функціонал у даному модулі.

Use-case діаграма для модулю «Задачі» представлена на рисунку 2.6.

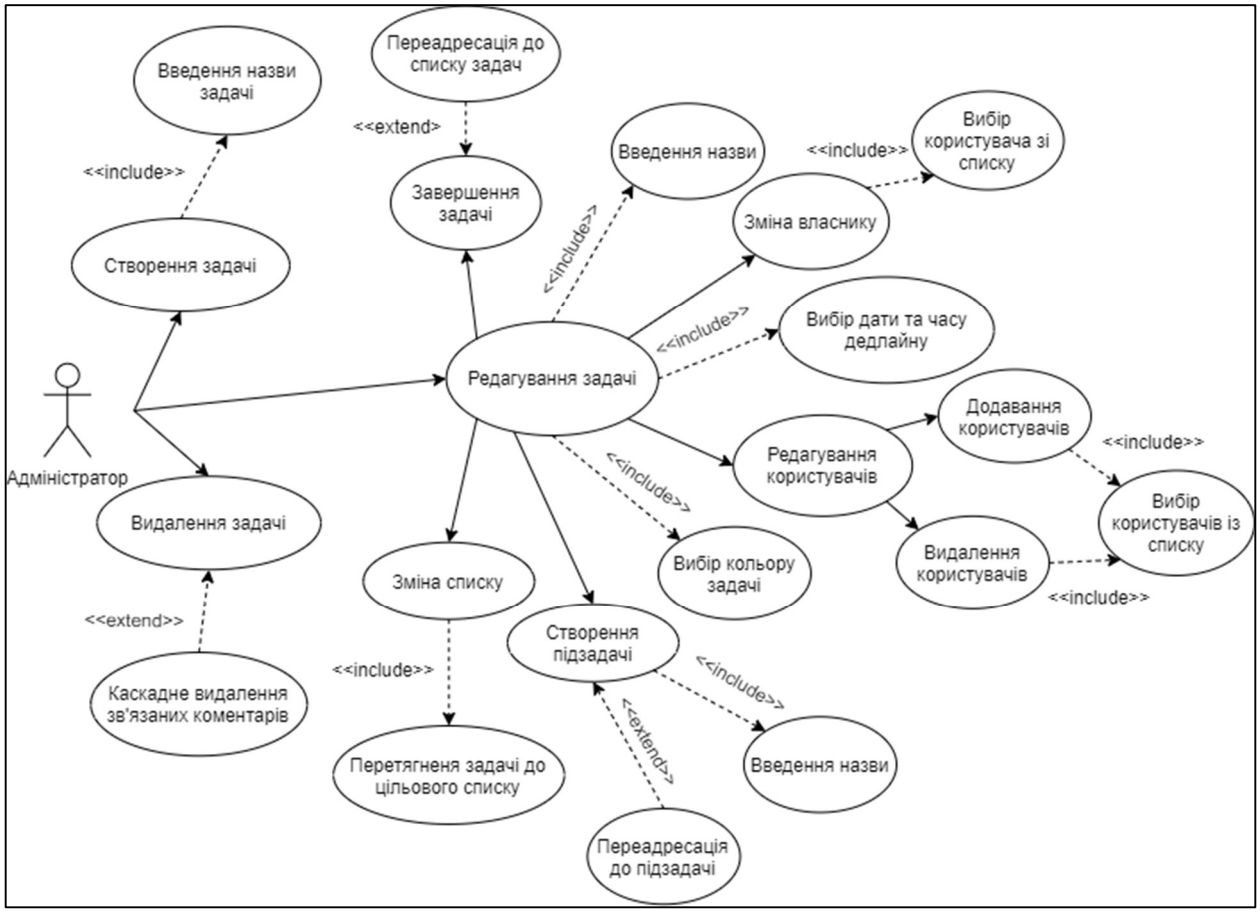

Рисунок 2.6 – Опис модулю «Задачі» для актора «Адміністратор»

Останнім розглянутим модулем є модуль «Коментарі». По аналогії із модулем «Задачі» робітник може створювати коментарі у будь якій задачі, але редагувати або видаляти може лише власні коментарі. Адміністратор може редагувати та видаляти будь які коментарі.

Щоб додати коментар до задачі, необхідно перейти до цільової задачі, натиснути кнопку «Add Comment», ввести текст коментарю та натиснути клавішу «Enter». Коментар буде збережено та додано до задачі. Також є можливість відредагувати зміст коментарю. це зробити, необхідно натиснути на клавішу «Edit» та відредагувати його зміст. Щоб зберегти зміни, необхідно натиснути кнопку «Save». Для видалення коментарю потрібно натиснути відповідно кнопку «Delete». Актор «Робітник» має аналогічний функціонал у даному модулі.

Use-case діаграма для модулю «Коментарі» представлена на рисунку 2.7.

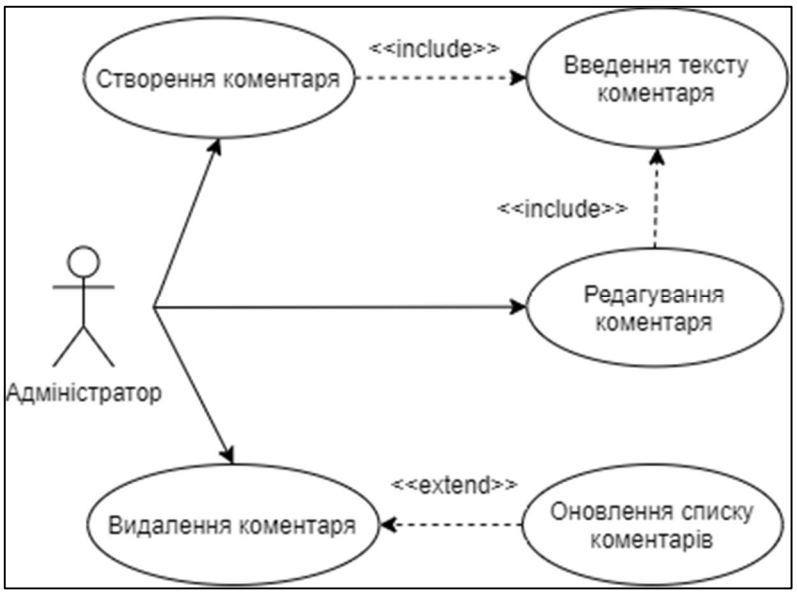

Рисунок 2.7 – Опис модулю «Коментарі» для актора «Адміністратор»

2.5 Розробка діаграми класів

Для візуалізації загальної структури ієрархії класів системи була побудована діаграма класів. Діаграма класів визначає типи класів системи і різного роду статичні зв'язки, які існують між ними. На діаграмах класів

зображуються також атрибути класів, операції класів та обмеження, які накладаються на зв'язки між класами.

Основними елементами є класи і зв'язки між ними. Класи характеризуються за допомогою атрибутів і операцій.

Атрибути описують властивості об'єктів класу. Більшість об'єктів в класі отримують свою індивідуальність через відмінності в їх атрибутах і взаємозв'язку з іншими об'єктами. Однак, можливі об'єкти з ідентичними значеннями атрибутів і взаємозв'язків. Тобто індивідуальність об'єктів визначається самим фактом їх існування, а не відмінностями в їх властивості.

Клас «Router» визначає, як застосунок відповідає на клієнтський запит до конкретної адреси. Для обробки запиту можна вказати кілька функцій зворотного виклику. Контролери реалізують всю основну логіку backend частини застосунку та повертають сформовану інформацію залежно від запиту клієнта. Модель надає дані і методи роботи з ними: запити до бази даних, перевірка на коректність. Модель не залежить від представлення і контролера, просто надаючи доступ до даних і управління ними.

Розроблена діаграма класів представлена на рисунку 2.8.

Клас «Router» — маршрутизатор, що працює із запитами від клієнта. Даний клас містить такі методи як: get, post, put, delete. Ці методи є похідними від відповідних методів HTTP. Маршрутизація визначає, як застосунок відповідає на клієнтський запит до конкретної адреси, яким є URI, і певного методу запиту HTTP (GET, POST і т.д.). Кожен маршрут може мати одну або кілька функцій обробки, які виконуються при зіставленні маршруту.

Опис методів класу:

- get() — використовується для отримання даних, наприклад, запит інформації щодо задачі;
- post() — використовується для створення даних, наприклад, створення нової задачі;

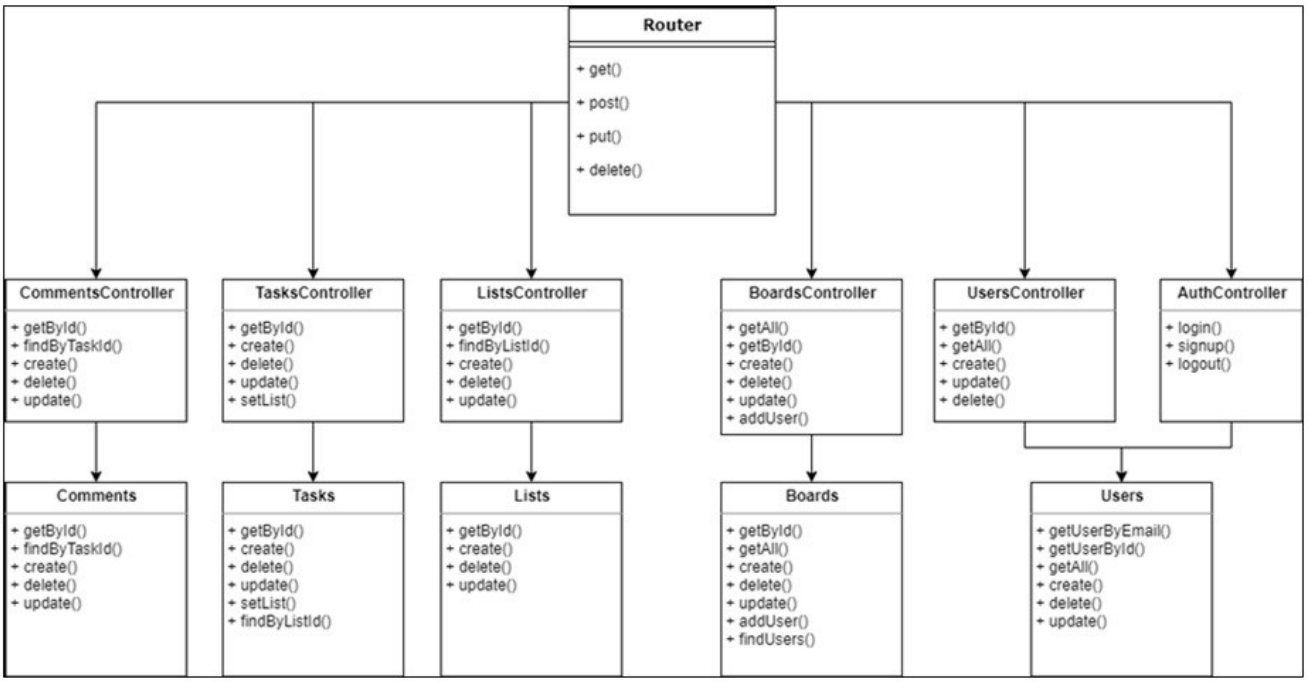

Рисунок 2.8 – Діаграма класів

– put() — використовується для оновлення даних, наприклад, зміни назви задачі;
– delete() — використовується для видалення даних, наприклад, видалення задачі.

Клас « AuthController» є нащадком класу «Router». Він відповідає за логіку модуля «Доступ». А точніше за реєстрацію та вхід користувача до системи. Методи даного класу:

– login() — відповідає за процес входу користувача до системи.
– logout() — відповідає за процес виходу користувача із системи.
– signup() — відповідає за процес реєстрації користувача.

Клас «UsersController» — відповідає за роботу із сутністю «Users». Методи класу:

– getById() — пошук інформації про користувача за його унікальним ідентифікатором (id).
– getAll() — надання інформації про всіх користувачів.
– create() — створення нового користувача.

- update() — оновлення інформації про користувача.
- delete() — видалення інформації про користувача.

Клас «BoardsController» — відповідає за роботу із сутністю «Дошки». Методи даного класу:

- getAll() — надання інформації про всі дошки користувача.
- getById() — пошук інформації про дошку за її унікальним ідентифікатором (id).
- create() — створення нової дошки.
- update() — оновлення інформації про дошку.
- delete() — видалення дошки.
- addUser() — додання нового користувача до дошки (операція поширення доступу).

Клас «ListsController» — відповідальний за роботу із сутністю «Списки». Методи даного класу:

- getById() — надання інформації про список за його унікальним ідентифікатором (id).
- create() — створення нового списку.
- update() — оновлення інформації про список.
- delete() — видалення списку.

Клас «TasksController» — відповідальний за роботу із сутністю «Задачі». Методи даного класу:

- getById() — надання інформації про задачу за її унікальним ідентифікатором (id).
- setList() — переміщення задачі до відповідного списку.
- create() — створення нової задачі.
- update() — оновлення інформації про задачу.
- delete() — видалення задачі.
- findByListId() — пошук задач за ідентифікатором списку.

Клас «CommentsController» — відповідальний за роботу із сутністю «Коментарі». Методи даного класу:

- getById() — надання інформації про коментар за його унікальним ідентифікатором (id).
- findByTaskId() — пошук коментарів за ідентифікатором задачі.
- create() — створення нового коментарю.
- update() — оновлення коментарю.
- delete() — видалення коментарю.

Клас «Users» — працює із сутністю «Користувачі» та відповідає за операції із даними із БД. Методи даного класу:

- getUserByEmail() — пошук запису у таблиці за атрибутом «email».
- getUserById() — пошук даних про у таблиці за атрибутом «id».
- getAll() — вилучення усіх записів із таблиці.
- create() — створення нового запису у таблиці.
- update() — оновлення запису у таблиці.
- delete() — видалення запису із таблиці.

Клас «Boards» — працює із таблицею «Дошки» та «Дошки – Користувачі». Відповідає за операції із даними із БД. Методи даного класу:

- getById() — пошук запису у таблиці за атрибутом «id».
- getAll() — вилучення усіх записів із таблиці.
- create() — створення нового запису у таблиці.
- update() — оновлення запису у таблиці.
- delete() — видалення запису із таблиці.
- addUser() — створення нового запису у таблиці «Дошки – Користувачі» із ідентифікатором користувача та дошки.
- findUsers() — вилучення записів із таблиці «Дошки – Користувачі» за атрибутом «board_id».

Клас «Lists» — працює із таблицею «Списки» та відповідає за операції із даними із БД. Методи даного класу:

- getById() — пошук запису у таблиці за атрибутом «id».
- create() — створення нового запису у таблиці.
- update() — оновлення запису у таблиці.
- delete() — видалення запису із таблиці.

Клас «Tasks» — працює із таблицею «Задачі» та відповідає за операції із даними із БД. Методи даного класу:

- getById() — пошук запису у таблиці за атрибутом «id».
- create() — створення нового запису у таблиці.
- update() — оновлення запису у таблиці.
- delete() — видалення запису із таблиці.
- setList() — пошук запису за атрибутом «id» та оновлення атрибуту «list_id».

findByListId() — пошук записів за атрибутом «list_id».

2.6 Алгоритми взаємодії користувача із системою

Будемо описувати алгоритми взаємодії користувача із системою за допомогою UML-діаграм послідовностей.

Перша взаємодія користувача із системою відбувається у модулі «Користувачі». А саме реєстрація. Спочатку email користувача перевіряється на валідність. Далі дані відсилаються на сервер та email звіряється із вже існуючими користувачами. Якщо email не зайнято, то створюється новий запис до БД.
Діаграму активності реєстрації користувача зображено на рисунку 2.9.

Рисунок 2.9 – Процес реєстрації користувача

Якщо користувач був зареєстрований раніше, то для початку роботи йому потрібно увійти у систему. Для цього потрібно ввести логін і пароль, що використовувалися під час реєстрації. Логін перевіряється на валідність та відправляється запит до БД, де відбувається пошук запису із відповідним логіном. Якщо запис існує, то введений користувачем пароль порівнюється із паролем у БД. Якщо паролі співпадають, то користувач авторизується у системі та перейде до головної сторінки.

Діаграму активності реєстрації користувача зображено на рисунку 2.10.

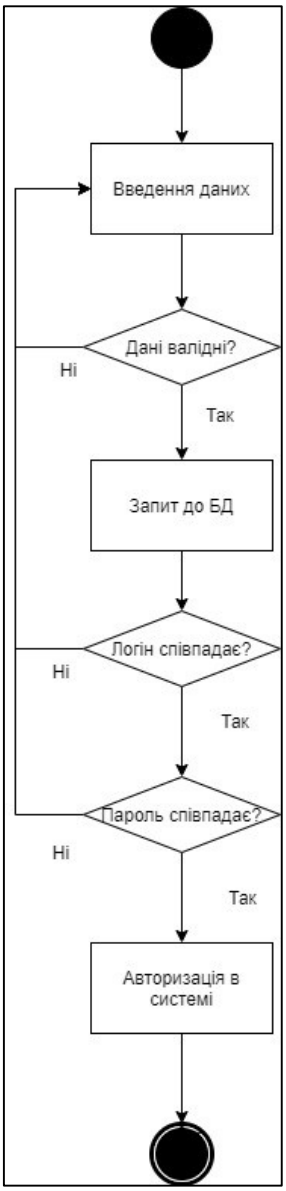

Рисунок 2.10 – Процес авторизації користувача в системі

Наступний модуль дозволяє користувачу взаємодіяти із дошками, де він може створити нову дошку, відредагувати або видалити існуючу.

Розглянемо процес створення нової дошки. В першу чергу перевіряється статус авторизації користувача. Якщо він не був авторизований у системі, то користувач автоматично буде переадресований до сторінки входу до системи. Після проходження авторизації користувач зможе створити нову дошку. Далі під час кожного запиту буде проводитися перевірка статусу авторизації і у випадку негативного статусу, користувач знову буде переадресований на сторінку входу. Процес створення дошки зображено на рисунку 2.11.

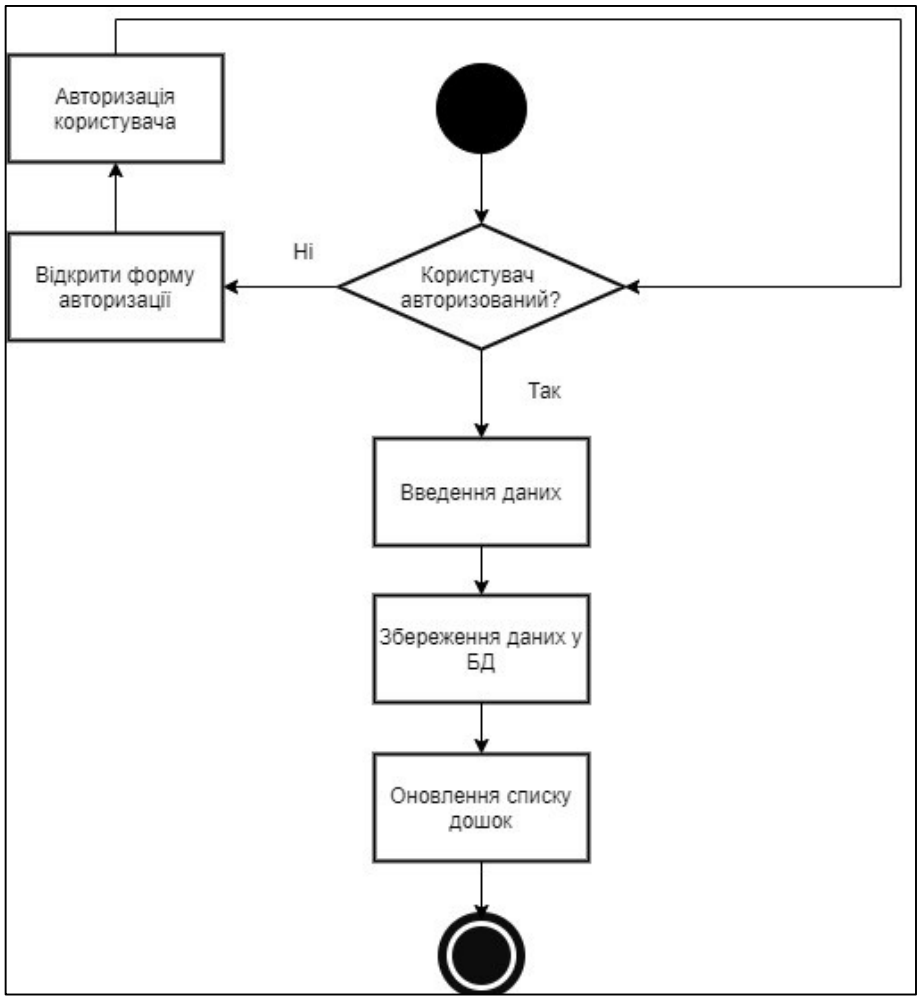

Рисунок 2.11 – Процес створення дошки

Наступним розглянемо процес редагування дошки. Користувач має можливість редагувати тільки ті дошки, яких власником він є. В першу чергу система перевіряє статус авторизації користувача та залежно від результату перевірки переходить на наступний етап або очікує авторизації від користувача. На наступному етапі проводиться ще одна перевірка на власника дошки. Якщо користувач не є власником дошки, то кнопка «Edit» для нього буде прихована, а на відповідний запит сервер відповість помилкою. Алгоритмічна схема процесу редагування дошки зображена на рисунку 2.12.

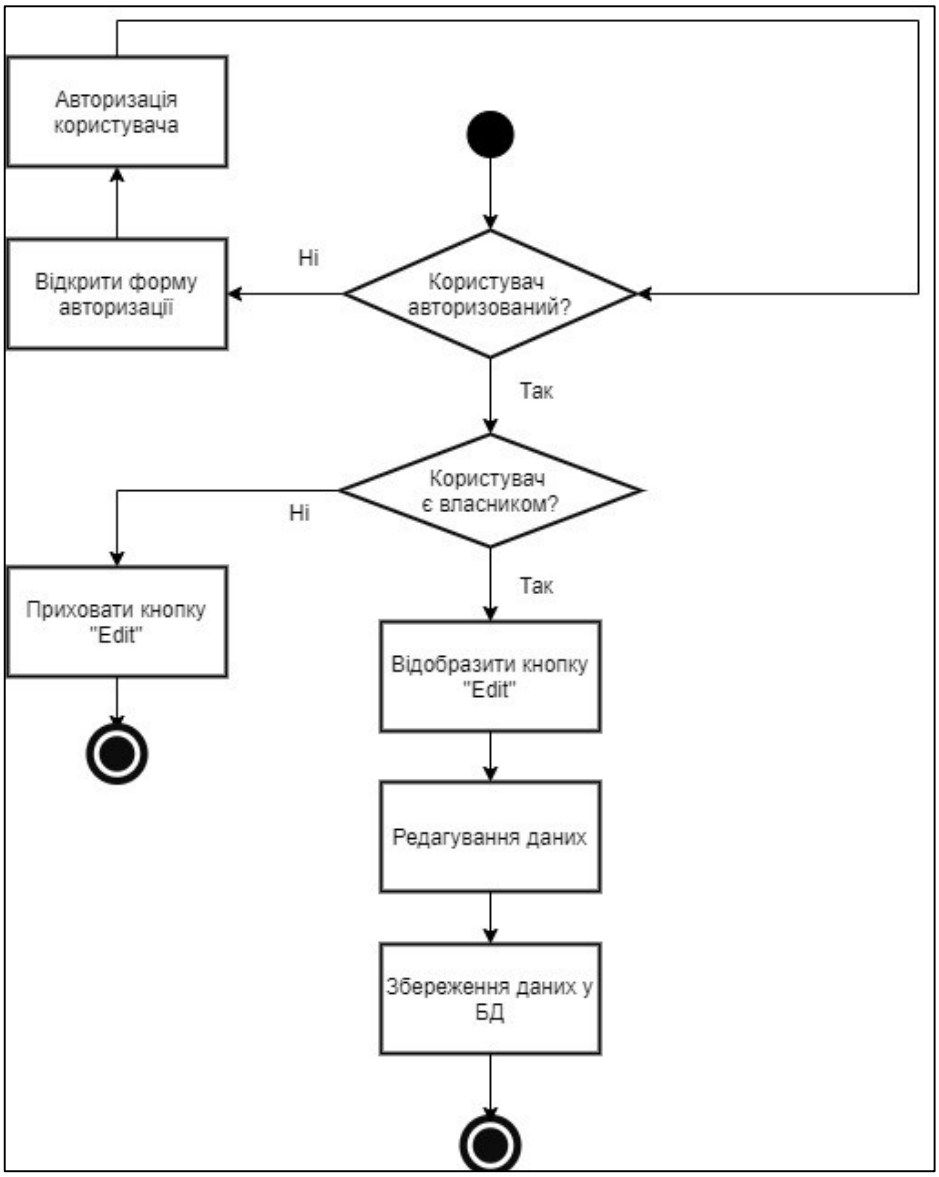

Рисунок 2.12 – Процес редагування дошки

Наступним буде розглянуто процес видалення дошки. По аналогії із процесом редагування дошки, спочатку виконується перевірка статусу авторизації користувача, потім перевірка на власника дошки. І у випадку, якщо обидві перевірки повернуть позитивний результат, система відобразить кнопку «Delete» у інтерфейсі користувача. На рисунку 2.13 продемонстровано процес видалення дошки.

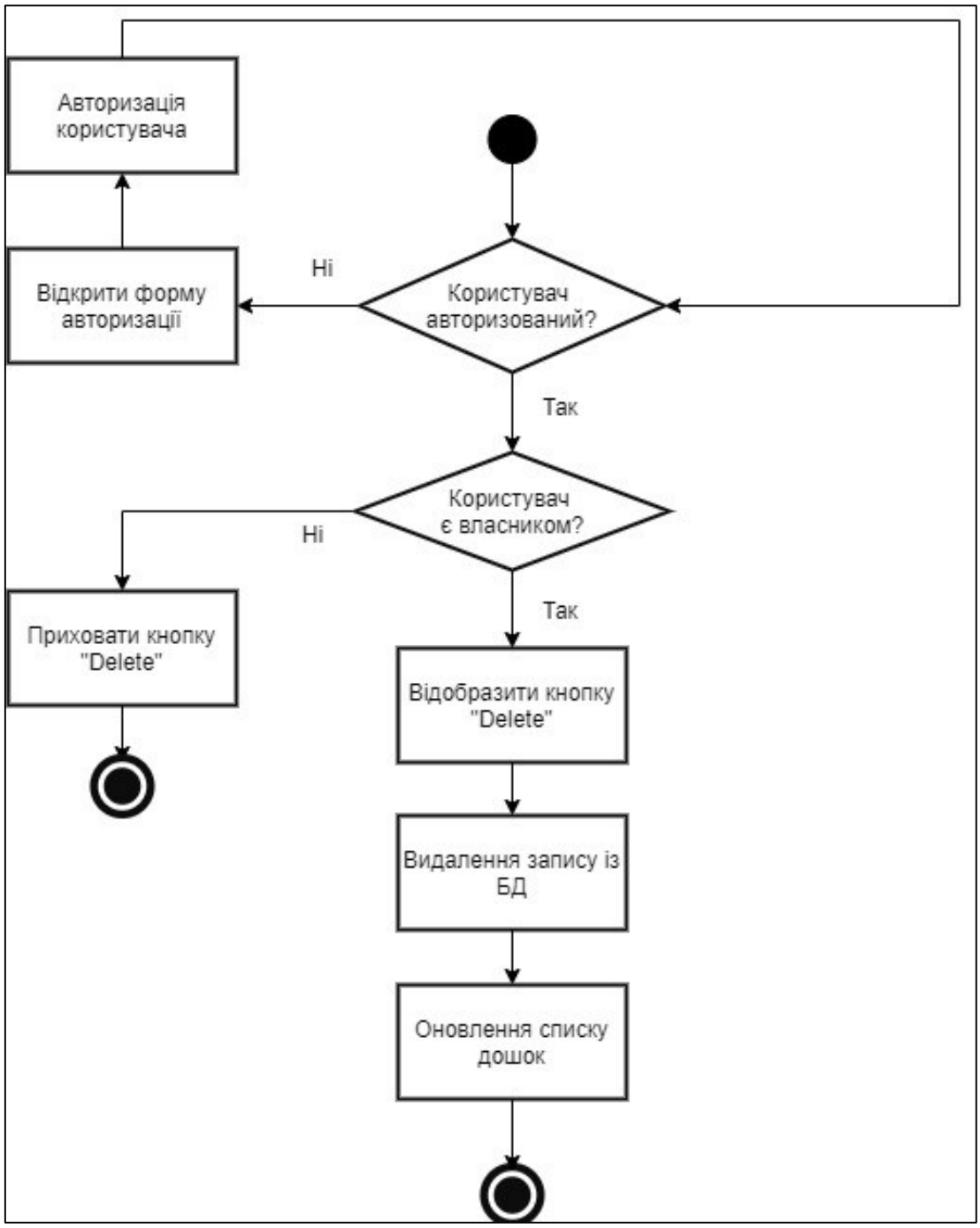

Рисунок 2.13 – Процес видалення дошки

І останній процес, який доступний адміністратору дошки — поширення доступу. Щоб отримати доступ до цієї функції, треба також пройти перевірку статусу авторизації та на власника дошки. Якщо обидві перевірки пройдені, то зв’явиться кнопка «Add member». Натиснутивши на дану кнопку, адміністратору буде запропоновано ввести email-адресу користувача, якому він хоче поширити доступ. Після цього відбудеться пошук користувача у базі даних за email-адресою

та у випадку відсутності користувача, адміністратор отримає повідомлення про помилку та пропозицію ввести інший email.

На рисунку 2.14 представлено процес поширення доступу до дошки

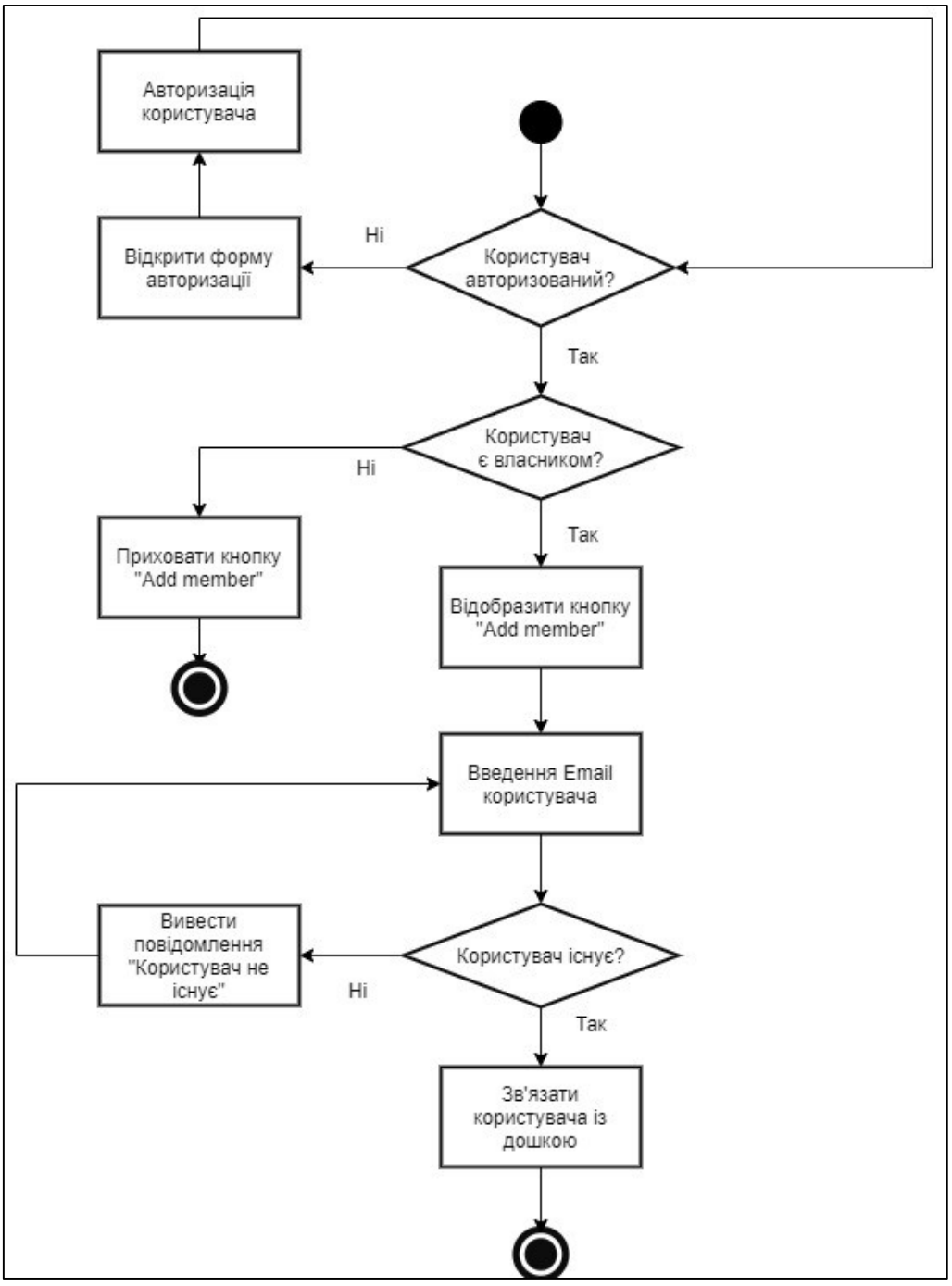

Рисунок 2.14 – Процес додавання співробітника

Після того як користувач створив дошку він автоматично набуде прав її адміністратора. Таким чином він зможе проводити операції із списками, що належать цій дошці.

Першим процесом, пов’язаним із списками, є процес його створення. Створювати та редагувати списки може тільки власник дошки, тому під час запиту на дану операцію відбудеться перевірка на власника дошки.

На рисунку 2.15 представлено процес створення списку.

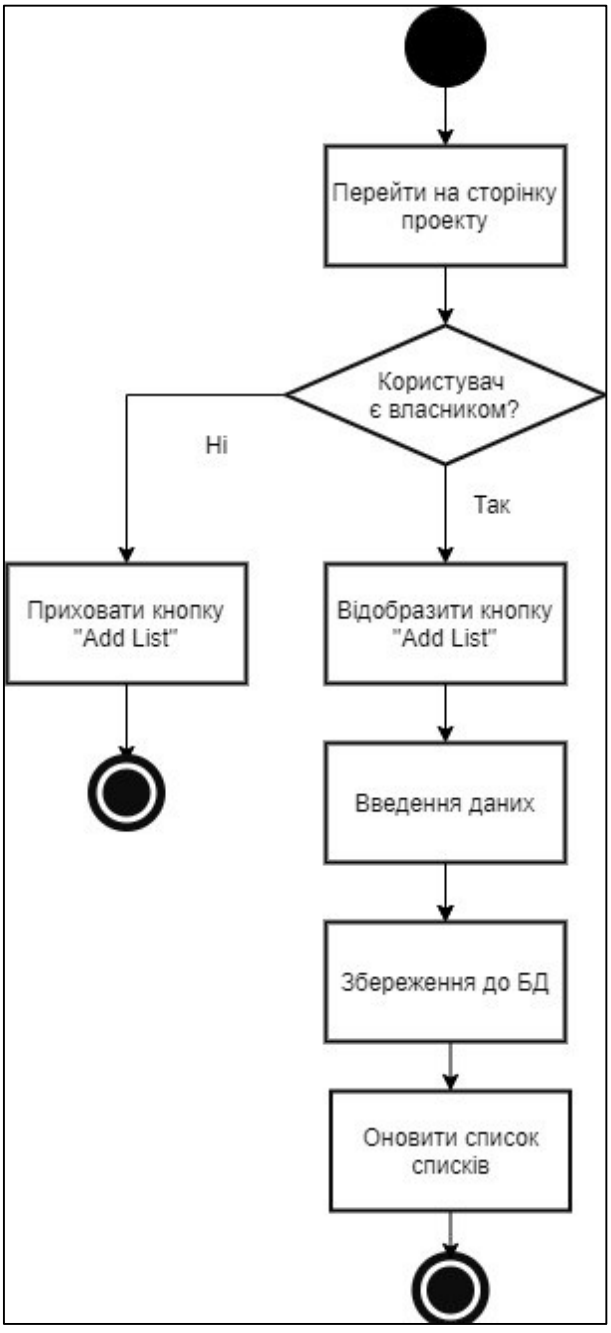

Рисунок 2.15 – Процес створення списку

Аналогічно до процесу створення списку, редагувати списки може тільки адміністратор дошки. Тому при запиті на цю операцію також відбувається перевірка на власника дошки.

На рисунку 2.16 зображено процес редагування списку.

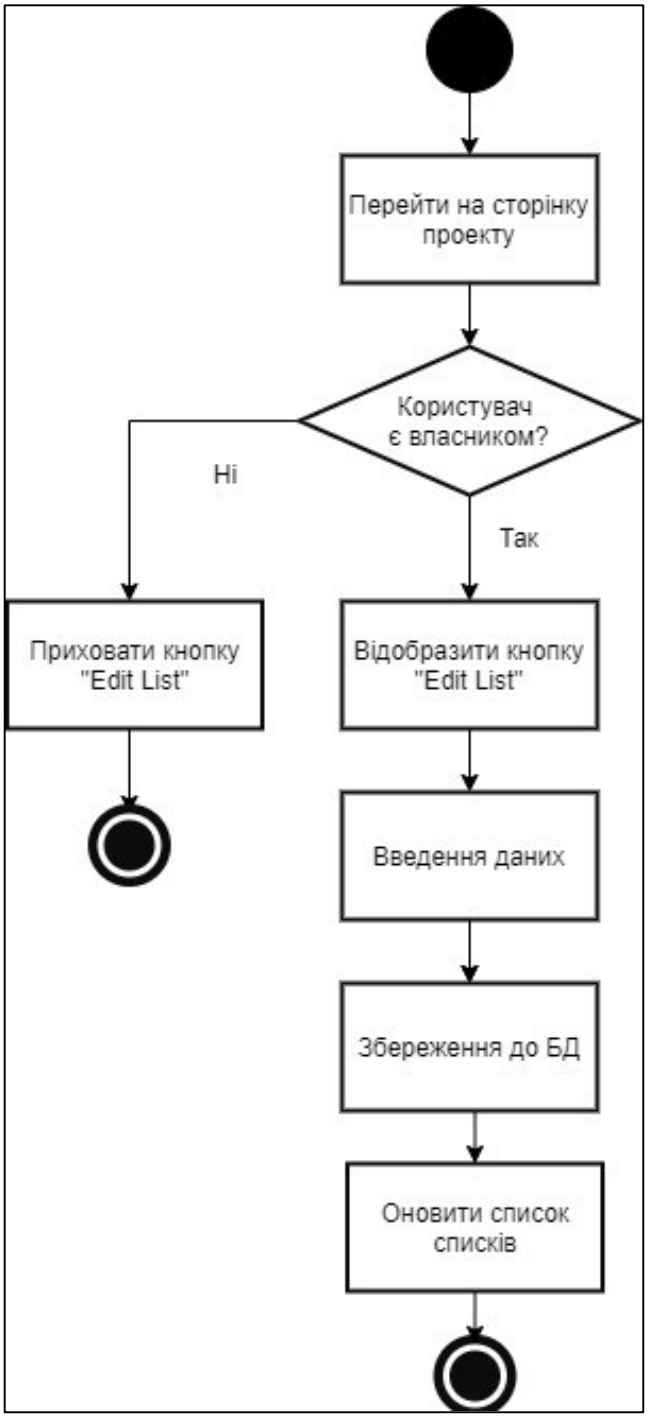

Рисунок 2.16 – Процес редагування існуючого списку

Далі розглянемо процес видалення списку. Доступ до цього процесу є тільки у адмністратора дошки. Тому відбувається перевірку на власника дошки. Під час видалення списку видаляються також усі задачі у даному списку.

Процес видалення існуючого списку зображено на рисунку 2.17.

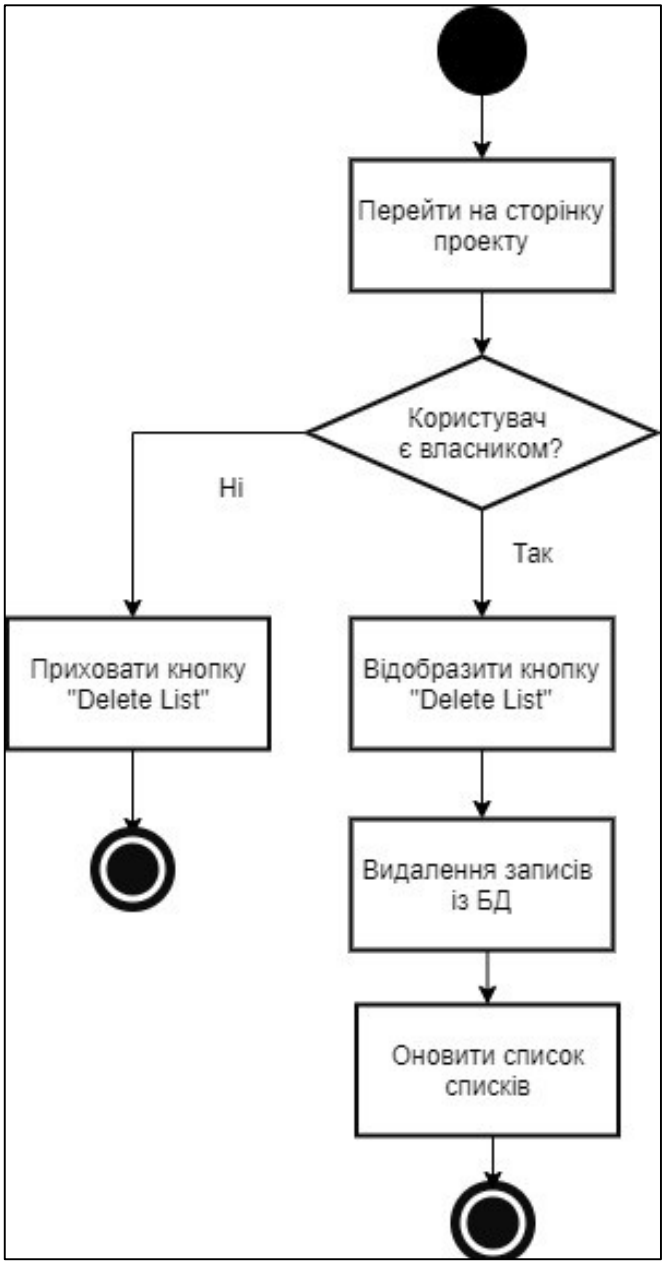

Рисунок 2.17 – Процес видалення списку з дошки

На відміну від списків, створювати задачі може будь який користувач, який має доступ до дошки. Наступним процесом розглянемо створення нової задачі. Для того, щоб ініціювати даний процес, користувачеві потрібно натиснути кнопку «Add Task» у цільовому списку та ввести назву задачі, після чого відбудеться перевірка валідності введених даних. Якщо дані валідні, нова задача буде створена, а список задач відновлений.

На рисунку 2.18 зображено процес створення нової задачі.

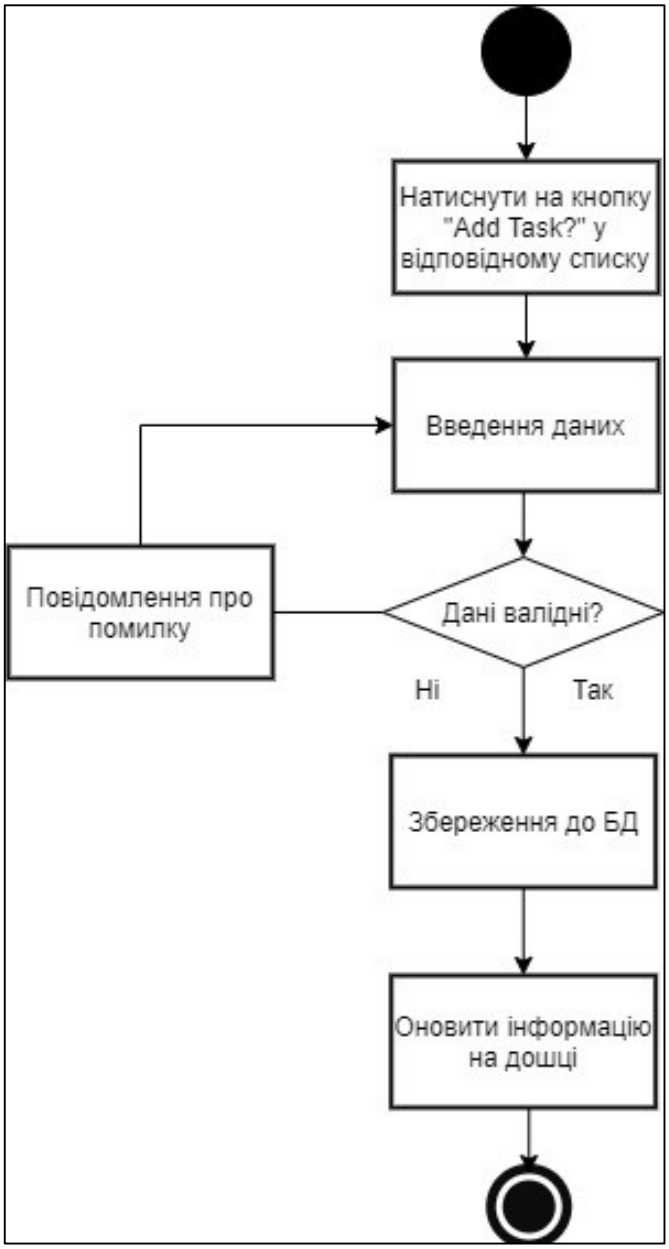

Рисунок 2.18 – Процес створення нової задачі

Процес редагування задачі є дещо складнішим. Зробити це можуть лише власник задачі або адміністратор дошки. Тому першим кроком після ініціації даного процесу відбувається перевірка на те, чи є користувач власником задачі. Якщо результат негативний, то відбудеться перевірка на ти, чи є користувач адміністратором дошки. Якщо результат також негативний, то користувач отримує повідомлення про помилку. Якщо якась з цих перевірок повернула

позитивний результат, то система відобразить користувачеві інтерфейс редагування задачі. Після введення даних відбудеться перевірка на валідність і при успішному завершені перевірки, оновлені дані задачі будуть збережені до бази даних.

На рисунку 2.19 зображено процес редагування задачі.

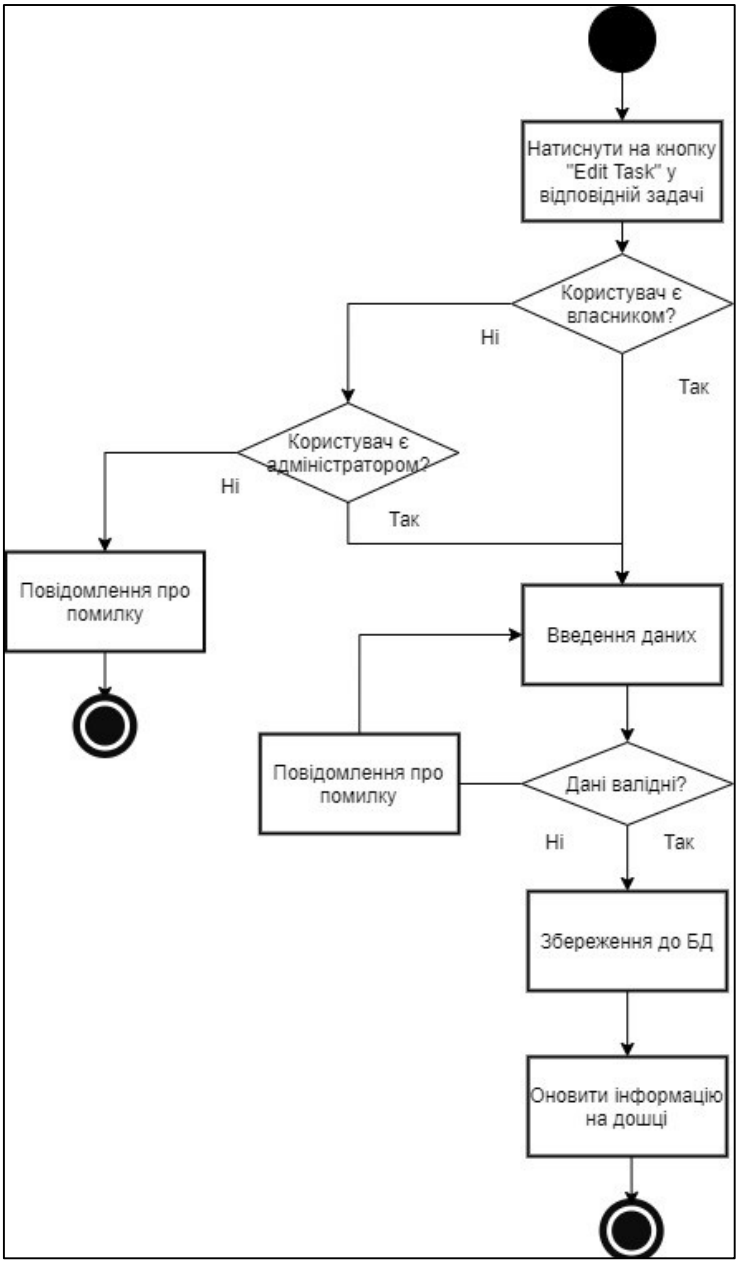

Рисунок 2.19 – Процес редагування задачі

Наступний процес — видалення задачі. Аналогічно до попереднього процесу, доступ до даного процесу мають власник задачі та адміністратор дошки. Тому після ініціації процесу відбудеться перевірка на власника задачі та слідом на

власника задачі. Якщо результат хоча б одної перевірка поверне позитивний результат, задача буде видалена із бази даних. Разом із задачею із бази даних також видаляються усі пов’язані з нею коментарі незалежно від того чи є ініціатор видалення власником коментарю. На рисунку 2.20 представлено процес видалення задачі.

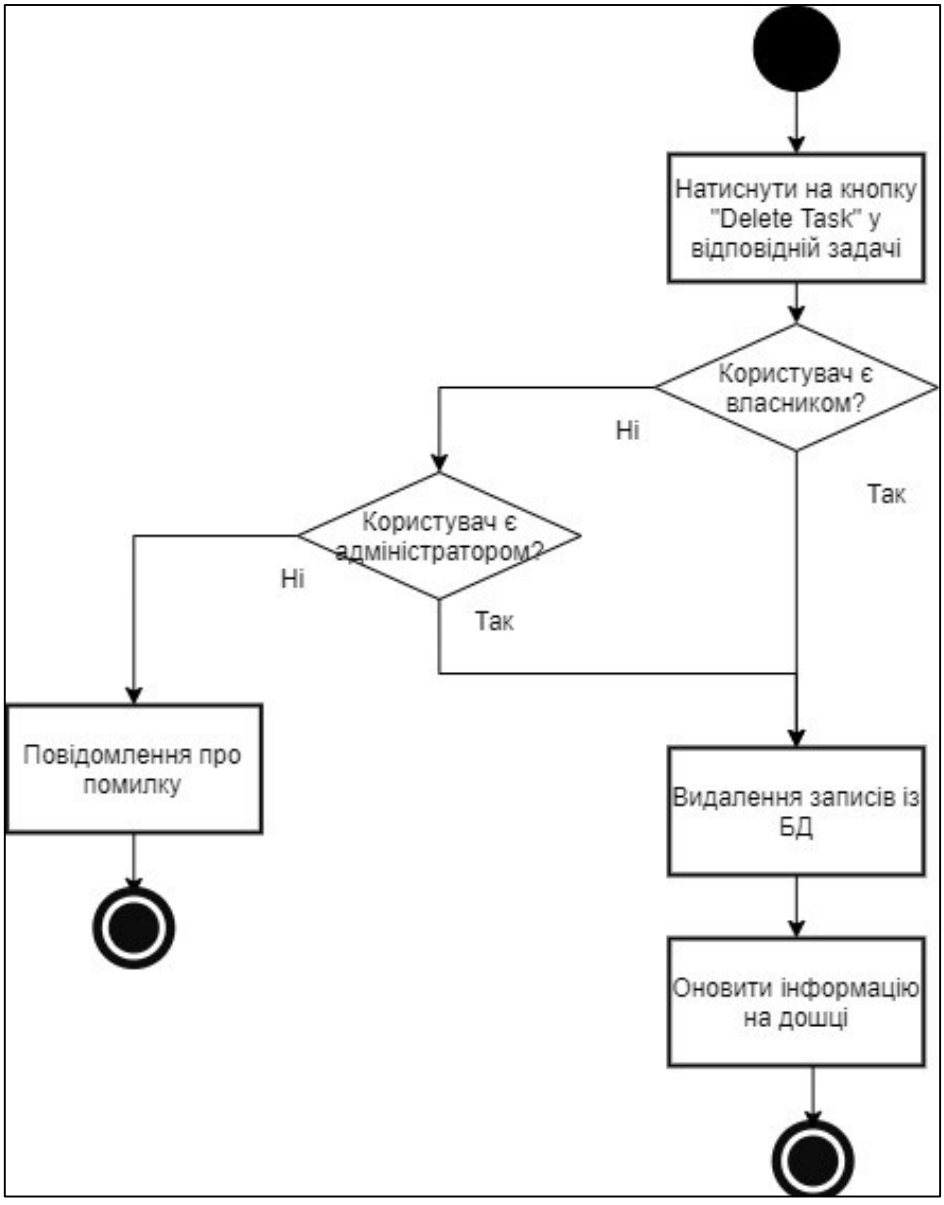

Рисунок 2.20 – Видалення задачі

Далі розглянемо процес залучення робітників до роботи над задачею. Залучити можна будь яку кількість співробітників. Зробити це може як власник картки, так і адміністратор. Для успішного виконання даного процесу потрібно

користувачеві потрібно перейти перевірку на власника задачі або адміністратора дошки та вибрати бажаного робітника із випадаючого списку. Для того, щоб додати робітника до задачі, потрібно попередньо поширити йому доступ до дошки.

Процес залучення співробітників до задачі наведено на рисунку 2.21.

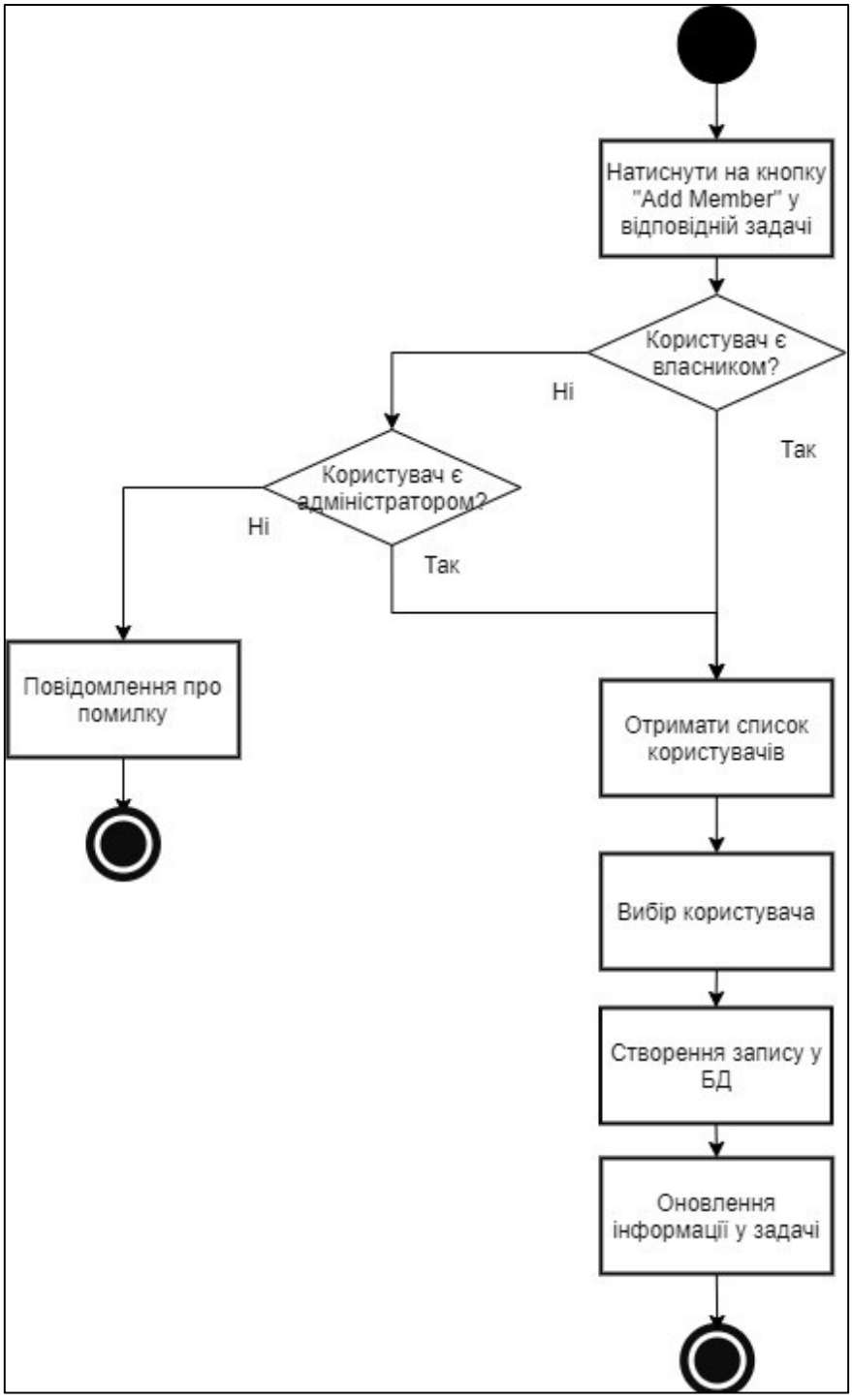

Рисунок 2.21 – Процес залучення співробітників до задачі

До кожної задачі є можливість додати підзадачу. Зробити це може будь який співробітник, що залучений до задачі. Після створення підзадачі ініціатор створення автоматично стане її власником, а доступ до дочірньої задачі отримають усі користувачі материнської задачі.

Процес створення підзадачі зображено на рисунку 2.22.

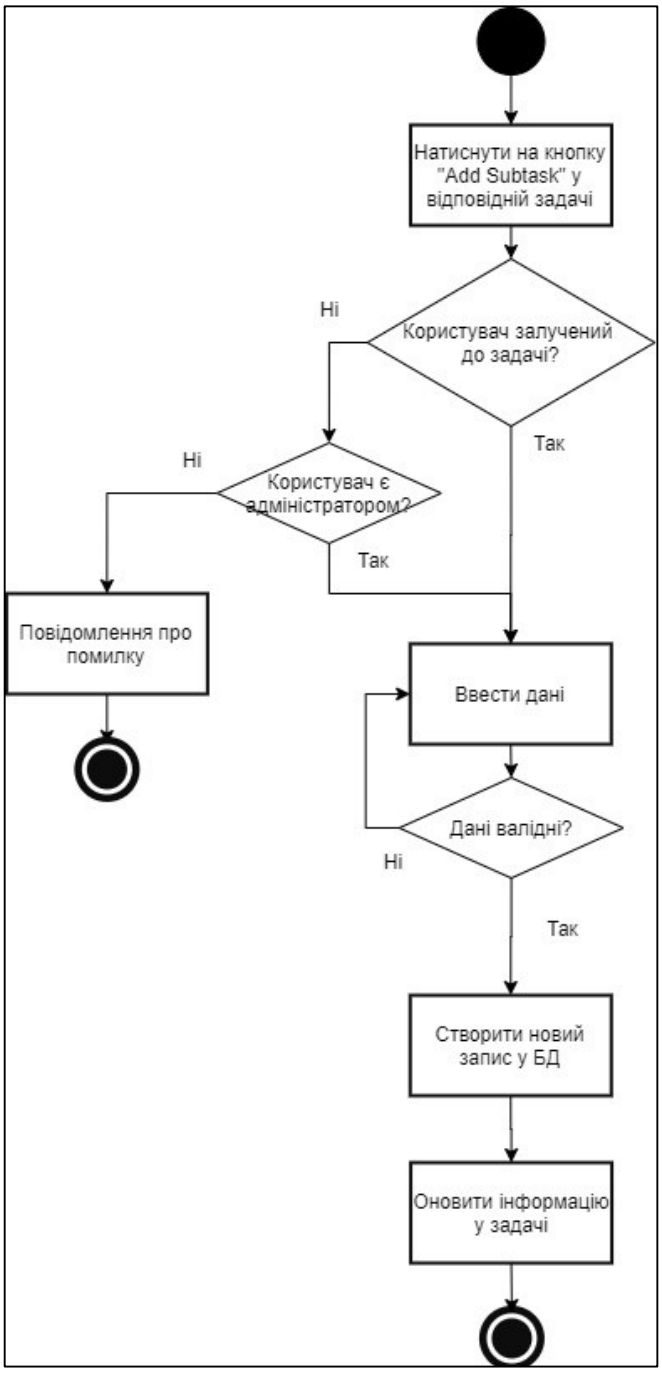

Рисунок 2.22 – Процес створення підзадачі

Переходимо до модулю «Коментарі». Коментарі потрібні для того щоб зручно фіксувати необхідну інформацію прямо у задачах. Залишити коментар може будь який користувач, незалежно від того чи залучений він до задачі.

Для того, щоб додати новий коментар, потрібно перейти на сторінки бажаної задачі та натиснути кнопку «Add comment». Після цього буде відображено поле для ввода тексту коментарю. Після введеня тексту коментарю відбудеться перевірка на валідність даних. При успішному завершені перевірки, коментар буде додано до задачі.

На рисунку 2.23 зображено процес створення коментарю.

Рисунок 2.23 – Процес створення коментарю

Відредагувати коментар може виключно його власник. Навіть адміністратор не може відредагувати чужий коментар. Відповідно алгоритм редагування коментарю включає в себе перевірку на те чи є ініціатор власник коментарю. При негативному результаті перевірки ініціатор отримує повідомлення про помилку. При позитивному — відобразиться поле для редагування тексту коментарю. Після збереження змін відбудеться перевірка на валідність введених даних.

На рисунку 2.24 зображено процес редагування коментаря.

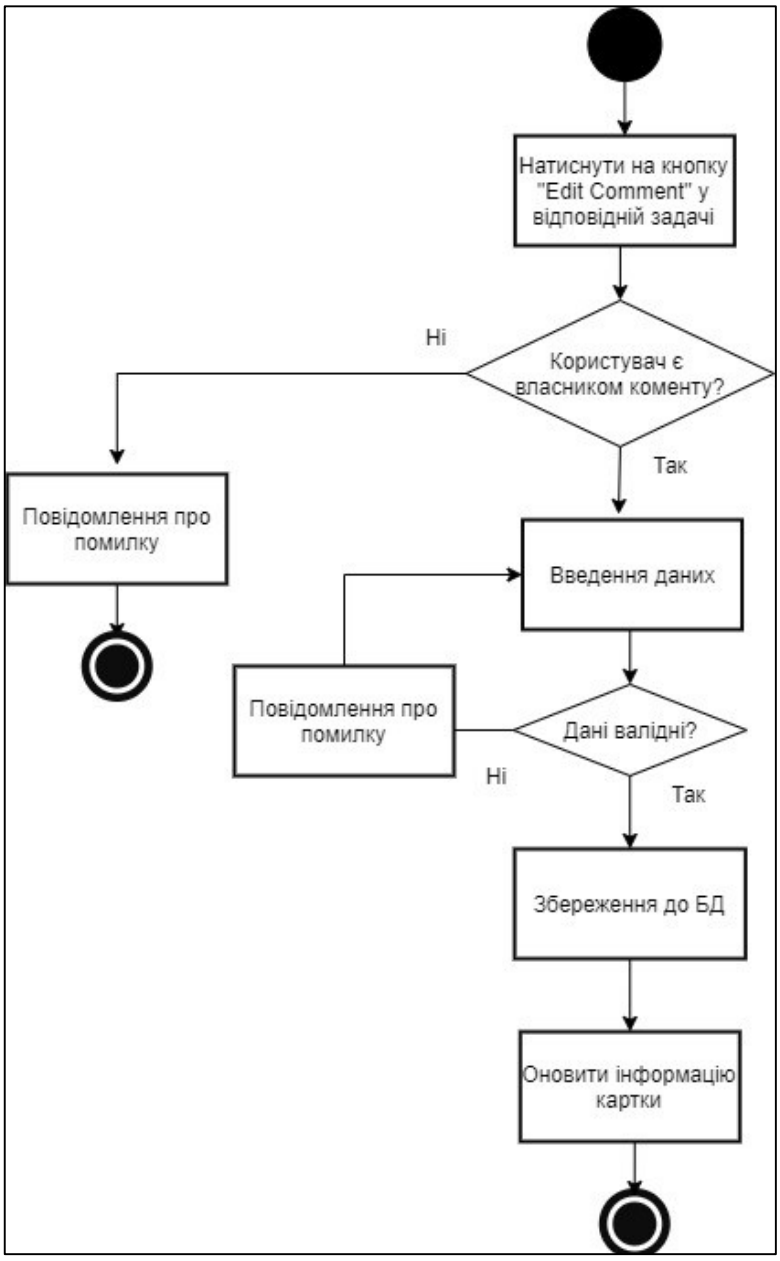

Рисунок 2.24 – Процес створення тест-плану

Видалити коментар можуть: власник коментаря, власник задачі або адміністратор дошки. Тому алгоритм процесу видалення коментарю включає в себе триетапну перевірку. Якщо кожна з цих перевірок поверне негативний результат, ініціатор отримає повідомлення про помилку.

Процес редагування тест-плану зображено на рисунку 2.25.

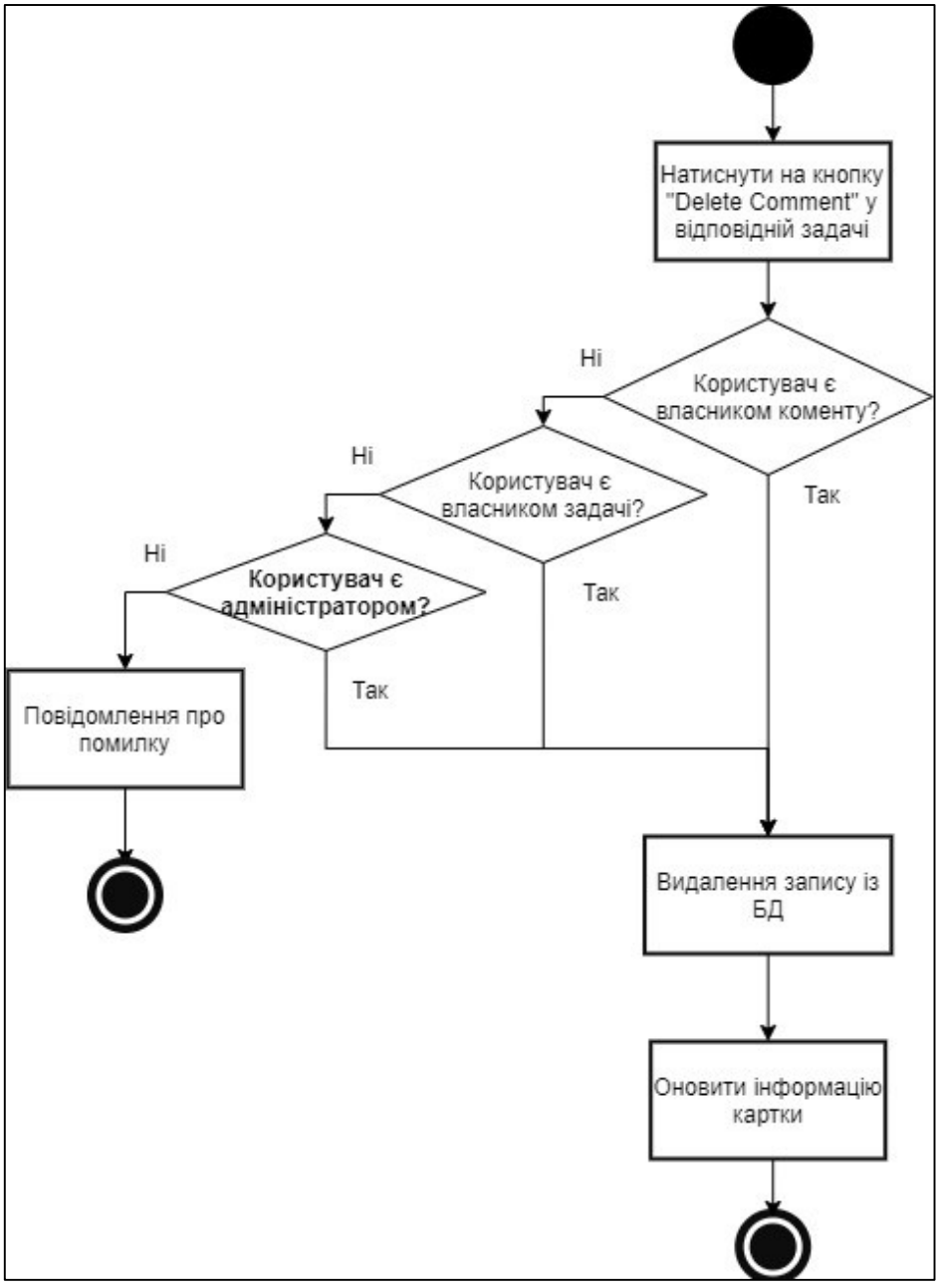

Рисунок 2.25 – Процес видалення коментаря

Розділ 3. РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТІВ ТА АНАЛІЗ ОТРИМАНИХ ДАНИХ

3.1 Інтерфейс користувача

Для введення даних до інформаційної системи розроблено інтерфейс користувача.

Розробку інтерфейсу користувача було реалізовано за допомогою VueJS. VueJS — це прогресивний JavaScript фреймворк з відкритим вихідним кодом, призначений для розробки користувальницького інтерфейсу. Він є одним з найпопулярніших фреймворків для спрощення веб-розробки. VueJS працює в основному з рівнем уявлення.

Одним з ключових моментів в роботі Vue.js є віртуальний DOM. Структура веб-сторінки, як правило, описується за допомогою DOM (Document Object Model), яка представляє організацію елементів html на сторінці. Для взаємодії з DOM (додавання, зміни, видалення html-елементів) застосовується JavaScript. Але коли ми намагаємося маніпулювати html-елементами за допомогою JavaScript, то ми можемо зіткнутися зі зниженням продуктивності, особливо при зміні великої кількості елементів. А операції над елементами можуть зайняти деякий час, що неминуче позначиться на призначеному для користувача досвід. Однак якби ми працювали з коду js з об'єктами JavaScript, то операції проводилися б швидше.

Для цього Vue.js використовує віртуальний DOM. Віртуальний DOM представляє легковажну копію звичайного DOM. Якщо застосунку потрібно дізнатися інформацію про стан елементів, то відбувається звернення до віртуального DOM. Якщо дані, які використовуються в застосунку Vue.js, змінюються, то зміни спочатку вносяться в віртуальний DOM. Потім Vue вибирає мінімальний набір компонентів, для яких треба виконати зміни на веб-сторінці, щоб реальний DOM відповідав віртуальному. Завдяки віртуальному DOM підвищується продуктивність програми.

При проектуванні інтерфейсу були дотримані принципи веб-компонентів. Компоненти — це екземпляри Vue, що використовуються повторно. Компонентний підхід дозволяє уникнути мішанини коду і чітко вибудовувати архітектуру програми. Будь-яку складну сторінку завжди можна розбити на менші складові. Кожну з таких частин при виділенні в компонент простіше підтримувати, а при необхідності повторювати розбиття всередині компонента на ще менші частини.

Ієрархія компонентів застосунку представлена на рисунку 3.1.

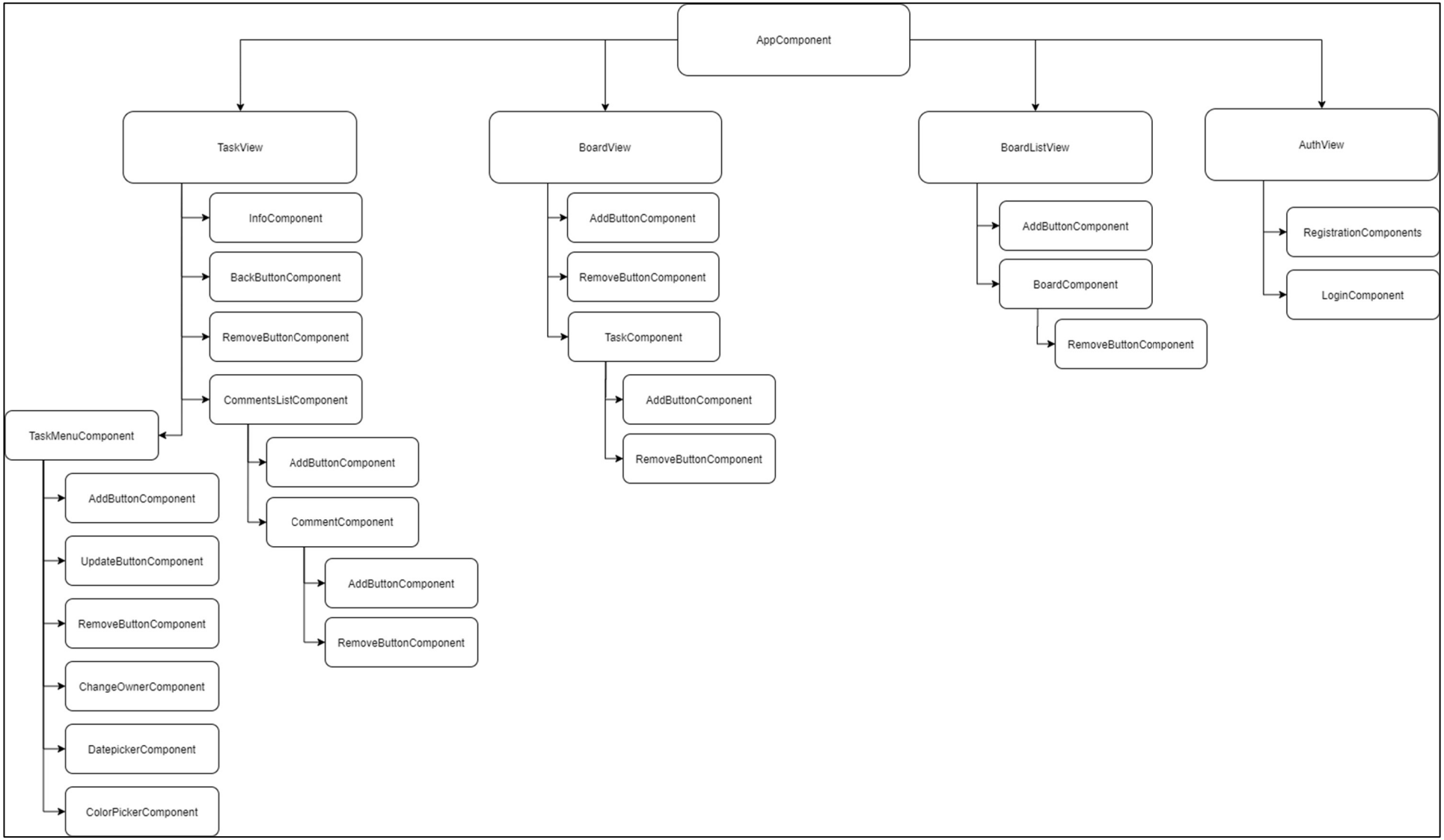

Рисунок 3.1 – Ієрархія компонентів інтерфейсу користувача

Такі компоненти, як AuthView, BoardListVIew, BoardView, TaskView, використовуються як представлення для маршрутизації.

У застосунку присутні наступні компоненти:

- AppComponent — головний компонент. Vue відстежує усе, що відбувається всередені цього компонента.
- AuthView — даний компонент містить в собі компоненти для реєстрації та входу до системи.
- BoardListView — список дошок, доступних користувачеві. Цей компонент використовується як головна сторінка.
- BoardView — представлення дошки. Даний компонент містить всю інформацію щодо конкретної дошки.
- TaskView — представлення задачі. Даний компонент містить всю інформацію щодо конкретної задачі.
- RegisterComponent — форма реєстрації користувача.
- LoginComponent — форма входу до системи.
- AddButtonComponent — універсальна кнопка «Додати». Використовується для створення нових дошок, списків, задач, коментарів.
- RemoveButtonComponent — універсальна кнопка «Видалити». Використовується для видалення дошок, списків, задач, коментарів.
- TaskComponent — картка задачі у списку задач.
- BoardComponent — картка дошки у списку дошок.
- InfoComponent — інформація про задачу.
- BackButtonComponent — універсальна кнопка «Назад». Використовується для переходу на попередню сторінку.
- CommentListComponent — список коментарів.
- CommentComponent — коментар.
- TaskMenuComponent — меню дій для задачі.
- ChangeOwnerComponent — універсальний компонент для зміни власника.
- DatepickerComponent — календар для вибору дати.

– ColorpickerComponent — палітра для вибору кольору.

Розглянемо роботу компонентів на прикладі компоненту TaskView. Зазвичай застосунок будується як дерево вкладених компонентів. Перше, що відбувається при створені елементу, це запит до інформації про задачу. Для цього у хуці життєвого циклу created викликається action-метод getTaskById глобального сховища Vuex (рис 3.2), у даному методі виконується запит до бази даних та його результат повертається і зберігається у змінну (3.3) .

Хуки життєвого циклу Vue дають нам можливість дізнатися коли компонент був створений, доданий в DOM, оновлений або знищений. Хук created спрацьовує коли екземпляр компонента вже був створений, але DOM ще не змонтований та сторінку не сформовано.

Vuex —централізоване сховище даних для всіх компонентів програми з правилами, що гарантують, що стан може бути змінено тільки передбачуваним чином.

```
created() {
  this.$store.dispatch('getTaskById', this.$route.params.id)
},
computed: {
  getTask() {
    return this.$store.getters.getTask
  }
}
```

Рисунок 3.2 – Виклик методу «getTaskById» та повернення інформації

```
getTaskById (state, id) {
  axios
  .get(state.route + '/tasks/' + id, {withCredentials: true})
  .then (response => {
    state.task = response.data
    getList(response.data.list_id)
  });

  function getList(id) {
    axios
    .get(state.route + '/lists/', {
      withCredentials: true,
      params: {
        id: id
      }
    })
    .then (response => {
        getUsers(response.data.board_id)
    });
  }

  function getUsers(id) {
    axios
    .get(state.route + '/boards/' + id + '/users')
    .then(response => {
      for(var i = 0; i < response.data.length; i++) {
        state.users.push(response.data[i])
      }
    })
    .catch(error => {
      console.log('err')
      console.log(error)
    })

  }
},
```

Рисунок 3.3 – Реалізація методу «getTaskById»

Для того, щоб була можливість використовувати компонент, його потрібно імпортувати та зареєструвати. На рисунку 3.4 наведений приклад імпорту компонентів із сторонніх файлів та їх реєстрація.

```
import axios from 'axios'
import BackButton from '@/components/BackButton.vue'
import TaskInfo from '@/components/TaskInfo.vue'
import RemoveButton from '@/components/removeButton.vue'
import Comments from '@/components/Comments.vue'
import TaskMenu from '@/components/Task-menu.vue'
export default {
  name: 'Task',
  components: {
    TaskInfo,
    BackButton,
    RemoveButton,
    Comments,
    TaskMenu,
  },
```

Рисунок 3.4 – Реєстрація компонентів

Зареєстровані компоненти вже можна використовувати, але від буде мало користі, якщо не буде можливості передавати в них дані. Звісно, така можливість є. На рисунку 3.5 демонструється передача інформації про задачу у дочірні компоненти «TaskInfo» та «TaskMenu».

```
<template>
  <div class="task-page">
    <div class="left-bar">
      <BackButton />
    </div>
    <div class="content">
      <TaskInfo :task="getTask"></TaskInfo>
      <div class="comments">
        <Comments :id="this.id"/>
      </div>
    </div>
    <div class="left-bar">
      <TaskMenu :task="getTask"></TaskMenu>
    </div>
  </div>
</template>
```

Рисунок 3.5 – Передача об’єкту задачі до дочірніх компонентів

Передача здійснюється через атрибут компонентів «props». Тепер в дочірньому компоненті ми маємо можливість працювати із надісланим об’єктом. Для цього потрібно звернутися до об’єкту task. На рисунку 3.6 продемонстровано приклад використання інформації із даного об’єкту.

```
<template>
  <div v-if="owner" class="task-menu">
    <RemoveButton type="tasks" :id="this.task.id">Remove</RemoveButton>
    <UpdateButton type="tasks" :id="this.task.id">Update</UpdateButton>
    <div @click="dueDate" class="due-date off">Due Date</div>
    <div v-if="dateUpdate" class="datepicker">
      <Datepicker v-model="task.deadline" class="due-date on" />
      <input @click="dueDateSubmit"  type="button" class="submit" value="Submit">
    </div>
    <ColorPick :object="this.task">Choose Color</ColorPick>
    <ChangeOwner :route="this.$store.getters.getRoute + '/tasks'" :object="this.task"></ChangeOwner>
    <AddButton type="tasks" :id="this.getTask.list_id" :parent="this.getTask.id">Add subtask</AddButton>
  </div>
</template>
```

Рисунок 3.6 – Html-шаблон компоненту «Task-menu»

3.2 Оцінка результатів розробки інформаційної системи

Оцінку результатів розробки інформаційної системи буде здійснено шляхом її тестування. Тестування в області розробки програмного забезпечення – це процес оцінки того, що всі частини програми поводяться так, як очікувалося.

Тести виконують перевірку програмного забезпечення, перевіряючи, що отриманий результат відповідає специфікаціям, враховуючи різні вхідні дані. Кожна з цих перевірок називається тестовим сценарієм (test case).

У тестових сценаріях перевіряються вимоги і характеристики конкретної функціональної можливості (функціоналу). Вони можуть надавати певні деталі або кроки, щоб їх можна було відтворити.

Говорячи про те, навіщо потрібно тестування, можна згадати такі його переваги:

– Економить гроші: без належного тестування кількість часу і ресурсів, необхідних для підтримки продукту, набагато більше, ніж інвестиції в тестування.

- Забезпечує безпеку коду при командній роботі. Наявність тестів роблять цей процес більш безпечним, оскільки ніхто не зламає щось, що не дізнавшись про це. Це також відноситься і до майбутнього, тести забезпечує безпеку коду, коли ви повернетеся через рік або два для внесення змін.
- Покращує якість коду: ваш продукт менш схильний до збоїв в роботі оскільки тести допомагає написати більш надійний і хороший код, який менш схильний до помилок.
- Робить рефакторинг простим і безпечним: створення програмного забезпечення - це ітеративний процес. Вимоги змінюються з плином часу, отже, змінюється і функціональність. Наявність хорошого тестового покриття дозволяє вам модифікувати певний код, перевіряючи, що тести все ще успішно проходять. Якщо це не так, ви робите правки в код таким чином, щоб тести пройшли.

Методологія тестування «Black box». Нам невідома внутрішня структура коду і не маємо доступу до бази даних.

Процес дослідження ПО на помилки здійснюється шляхом генерації тестових випадків виходячи з аналізу функціональної специфікації (документ, який описує бажані характеристики системи і підсумковий результат) або певних елементів системи.

Плюси «Black box»:

- Можливість визначення помилок, які не покриває метод "White box". Наприклад, при розробці було упущено якусь функціональність. Код працює добре, а ось відсутній елемент в системі, таким чином, щоб вважається вагомим багом.
- Аналіз ПЗ на наявність дефектів здійснюється як би на призначеному для користувача рівні (з позиції користувача) без поглиблення у внутрішню структуру системи.
- Потрібно значно менше часу, завдяки тому, що тест-кейси можна складати відразу після складання специфікації.

Мінуси:

- Пропуск моментів, які не прописані в специфіці, але присутні в функціональності коду.
- Невелике охоплення - тестування піддаються лише кілька вступних значень.
- При нечіткої специфікації можлива труднощі в складанні тест-кейсів.

Проведемо тестування модулів системи за наступними критеріями:

пріоритет (priority) – властивість тестового артефакту, що впливає на черговість виконання завдання або усунення дефекту. Є характеристикою, яка визначається з точки зору бізнесу.

- серйозність (severity) – властивість тестового артефакту, що характеризує вплив артефакту на працездатність програми. Є характеристикою, яка визначається з точки функціональності.;
- кроки відтворення (step to Reproduce);
- очікуваний результат (expected Result);
- фактичний результат, опис та рисунок.

Тестування функції реєстрації. Пріоритет — високий. Серйозність — критична. Кроки відтворення:

- введення в поле «Email» поштову адресу користувача у некоректному форматі.

  Очікуваний результат: відобразиться повідомлення про помилку.

  Фактичний результат: повідомлення відобразилося.

  Результат показано на рисунку 3.7.

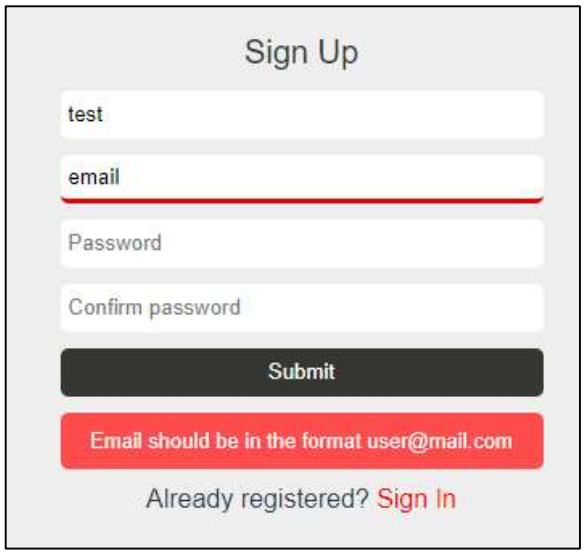

Рисунок 3.7 – Повідомлення про невалідну Email-адресу

Функція перевірки паролей. Кроки відтворення:

– введення у поля «password» та «confirm password» даних, що відрізняються між собою.

Очікуваний результат: відобразиться повідомлення про помилку.

Фактичний результат: повідомлення відобразилося.

Результат показано на рисунку 3.8.

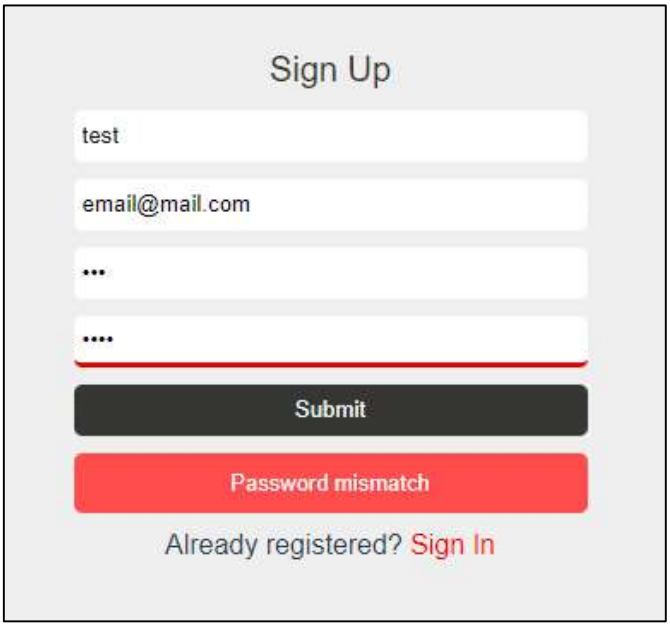

Рисунок 3.8 – Повідомлення про неспівпадаючі паролі

Перевірка імені користувача. Дане поле не може бути пустим. Кроки відтворення:

– Залишити поле «username» пустим та заповнити інші поля.
– Натиснути кнопку «Submit».

Очікуваний результат: відобразиться повідомлення про помилку.

Фактичний результат: повідомлення відобразилося.

Результат показано на рисунку 3.9.

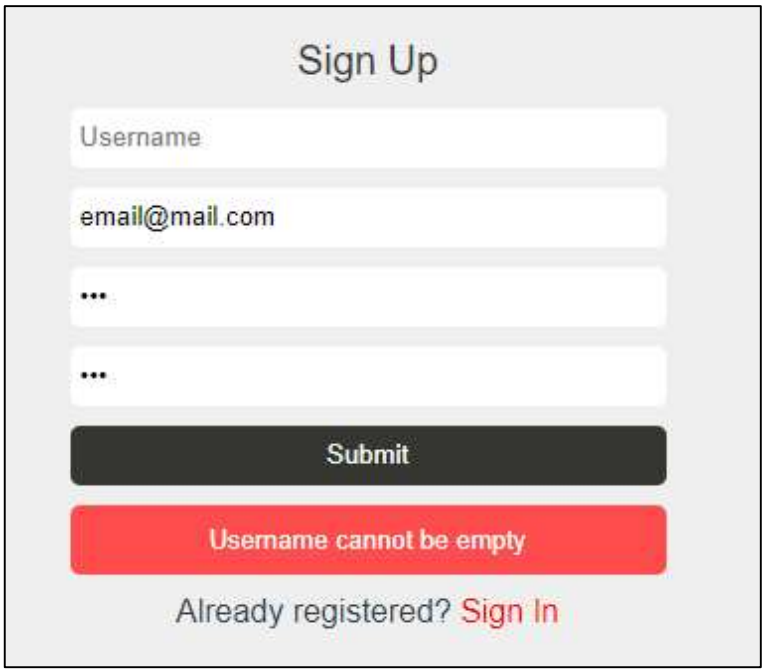

Рисунок 3.9 – Повідомлення про пусте ім'я користувача

Введення коректних даних в усі поля. Кроки відтворення:

– Заповнити усі поля відповідно до вимог.
– Натиснути кнопку «Submit».

Очікуваний результат: реєстрація закінчиться успішно та користувач буде переадресований на сторінку входу.

Фактичний результат: реєстрація відбулася, користувач був переадресований.

Результат показано на рисунку 3.10.

Тестування функції входу до системи. Пріоритет — високий. Серйозність – критична. Кроки відтворення:

– Ввести в поле «Email» неіснуючу адресу.
– Ввести в поле «Password» неправильний пароль.

Очікуваний результат: відобразиться повідомлення про помилку.

Фактичний результат: повідомлення відобразилося.

Результат показано на рисунку 3.11.

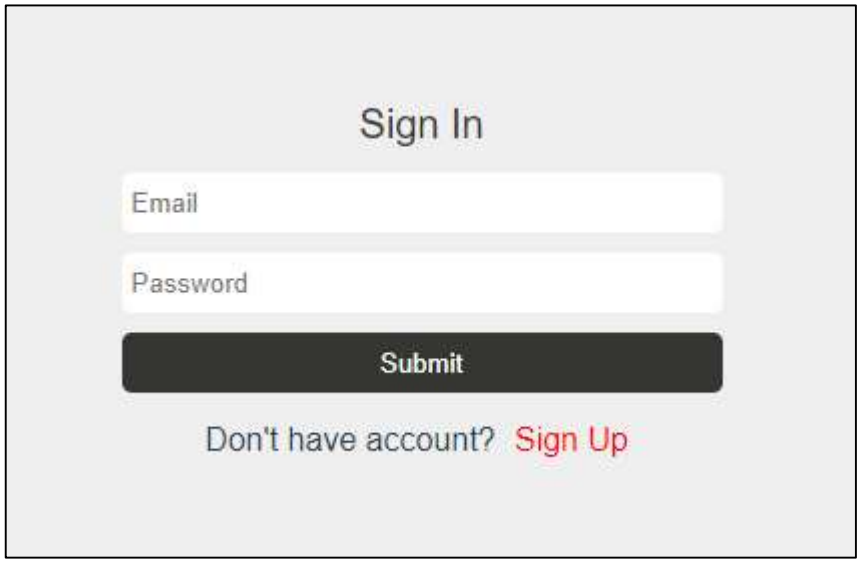

Рисунок 3.11 – Сторінка входу

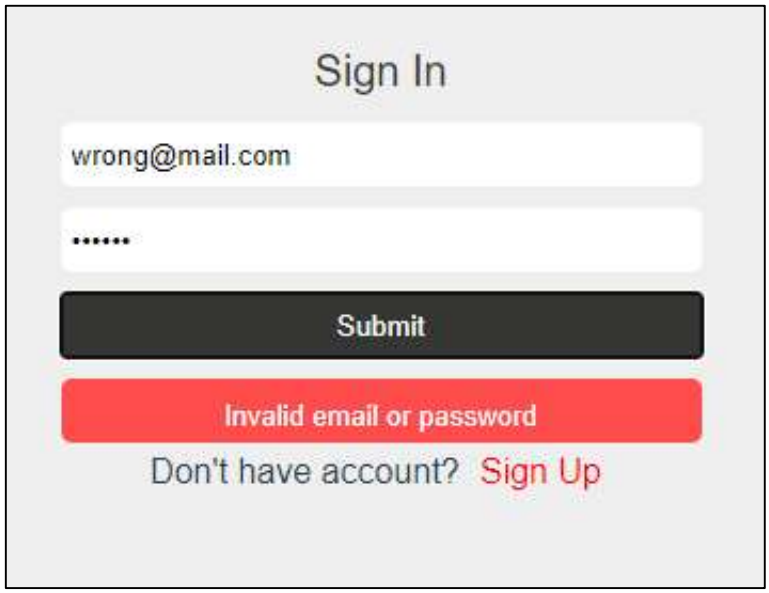

Рисунок 3.12 – Повідомлення про некоректний Email або пароль

Введення коректних даних в усі поля. Кроки відтворення:

– Заповнити усі поля відповідно до вимог.
– Натиснути кнопку «Submit».

Очікуваний результат: відбудеться вхід до системи та користувач буде переадресований на головну сторінку.

Фактичний результат: вхід відбувся та користувач був переадресований на головну сторінку. Результат показано на рисунку 3.13.

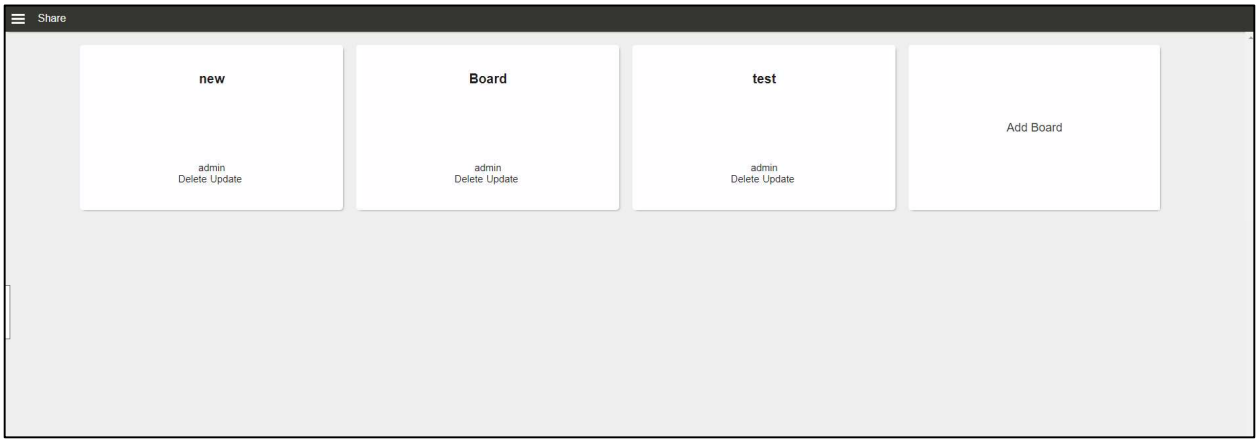

Рисунок 3.13 – Головна сторінка

Тестування функції створення дошки. Пріоритет — високий. Серйозність — критична. Кроки відтворення:

— Натиснути кнопку «Add Board»
— Ввести назву дошки «New Board».
— Натиснути клавішу «Enter» на клавіатурі.

Очікуваний результат: дошка буде створена та добавлена до списку.

Фактичний результат: дошка була створена та добавлена до списку.

Результат продемонстровано на рисунку 3.14.

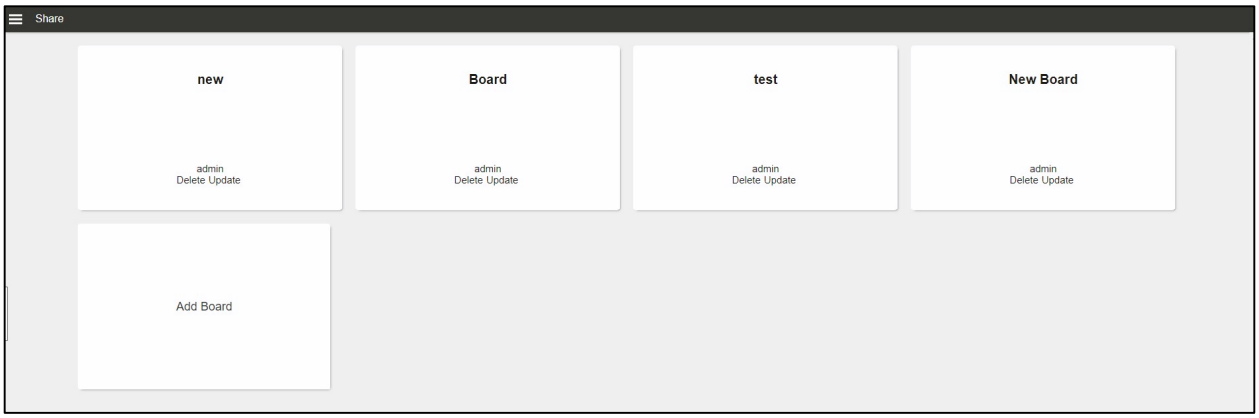

Рисунок 3.14 – Нова дошка була створена та добавлена до списку

Тестування функції оновлення дошки. Пріоритет — середній. Серйозність — середня. Кроки відтворення:

— Натиснути кнопку «Update» на дошці.

— Ввести нову назву дошки «New Board Updated».

— Натиснути кнопку «Save».

Очікуваний результат: ім’я дошки буде змінено.

Фактичний результат: ім’я дошки було змінено.

Результат продемонстровано на рисунку 3.15.

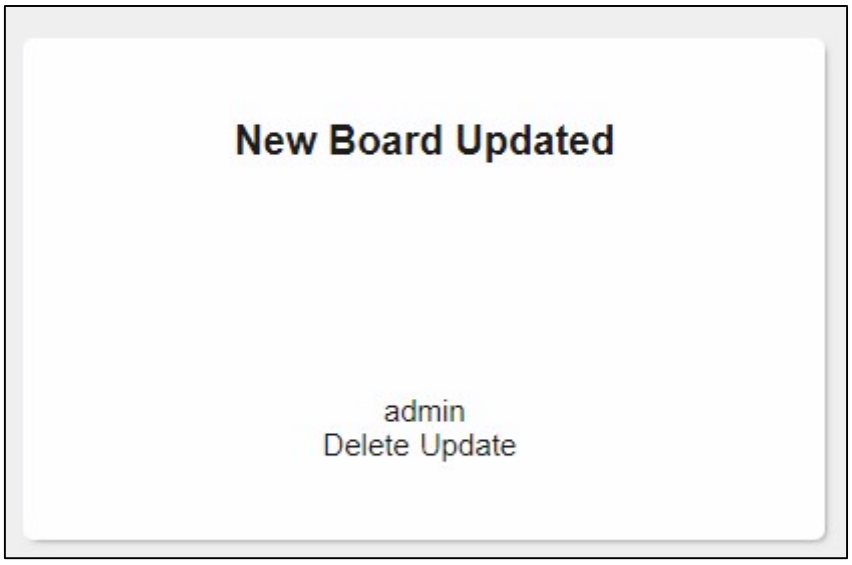

Рисунок 3.15 – Оновлена дошка

Тестування функції видалення дошки. Пріоритет — середній. Серйозність — середня. Кроки відтворення:

— Натиснути кнопку «Delete» на дошці.

Очікуваний результат: дошка буде видалена із списку.

Фактичний результат: дошка була видалена із списку.

Результат продемонстровано на рисунку 3.16.

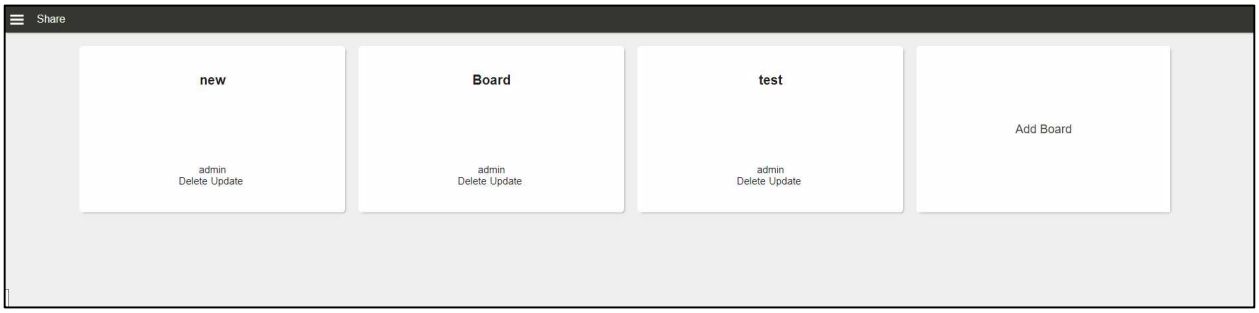

Рисунок 3.16 – Дошку було видалено

Тестування функції створення списку. Пріоритет — високий. Серйозність – критична. Кроки відтворення:

— Натиснути кнопку «Add List»

— Ввести назву списку «New List».

— Натиснути клавішу «Enter» на клавіатурі.

Очікуваний результат: список буде створений та добавлений на дошку.

Фактичний результат: список був створений та добавлений на дошку.

Результат продемонстровано на рисунку 3.17.

Рисунок 3.17 – Список був створений та добавлений на дошку

Тестування функції оновлення списку. Пріоритет — середній. Серйозність – середня. Кроки відтворення:

— Натиснути кнопку «Update»

— Ввести нову назву списку «New List Updated».

— Натиснути кнопку «Save».

Очікуваний результат: ім’я списку буде змінено.

Фактичний результат: ім’я списку було змінено.

Результат продемонстровано на рисунку 3.18.

Тестування функції видалення списку. Пріоритет — високий. Серйозність – критична. Кроки відтворення:

— Натиснути кнопку «Delete»

Очікуваний результат: список буде видалено.

Фактичний результат: список був видалений.

Результат продемонстровано на рисунку 3.19.

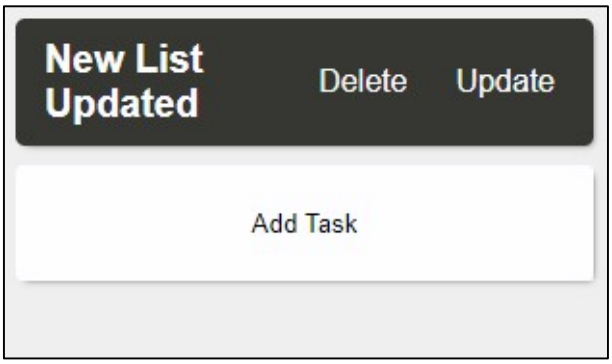

Рисунок 3.18 – Ім’я списку було змінено

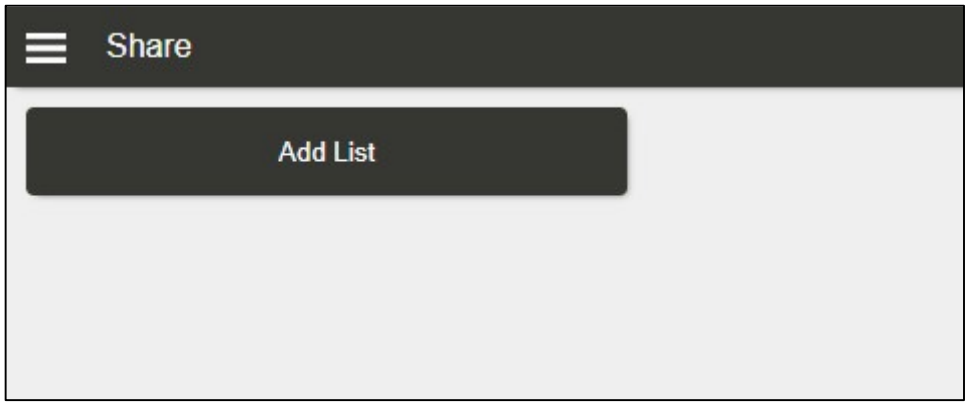

Рисунок 3.19 – Список був видалений

Тестування функції створення задачі. Пріоритет — високий. Серйозність – критична. Кроки відтворення:

— Натиснути кнопку «Add Task»

— Ввести назву «New Task»

— Натиснути клавішу «Enter» на клавіатурі.

Очікуваний результат: задачу буде створено та добавлено до списку.

Фактичний результат: задачу було створено та добавлено до списку.

Результат продемонстровано на рисунку 3.20.

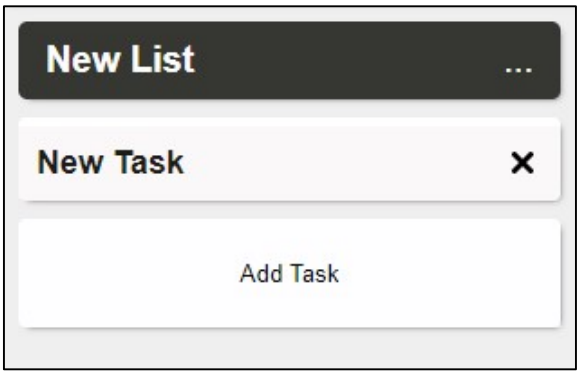

Рисунок 3.20 – Задача була створена

Тестування функції видалення задачі. Пріоритет — високий. Серйозність – критична. Кроки відтворення:

— Натиснути на кнопку «X»

Очікуваний результат: задачу буде видалено.

Фактичний результат: задачу було видалено.

Результат продемонстровано на рисунку 3.21.

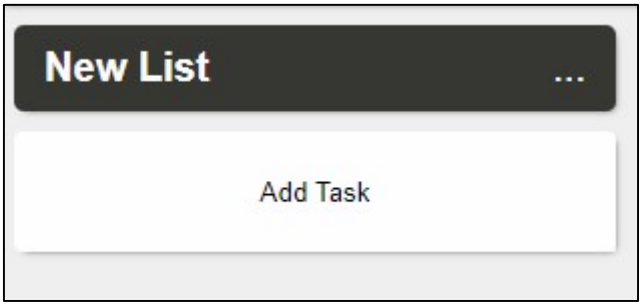

Рисунок 3.21 – Задачу було видалено

Тестування функції оновлення задачі. Пріоритет — високий. Серйозність – критична. Кроки відтворення:

— Перейти до сторінки задачі.

— Натиснути на кнопку «Update».

— Ввести нове ім’я задачі «New Task Updated» (рис 5.15).

— Ввести опис задачі «New Description».

— Натиснути кнопку «Save».

Очікуваний результат: задачу буде оновлено.

Фактичний результат: задачу було оновлено.

Результат продемонстровано на рисунку 3.22.

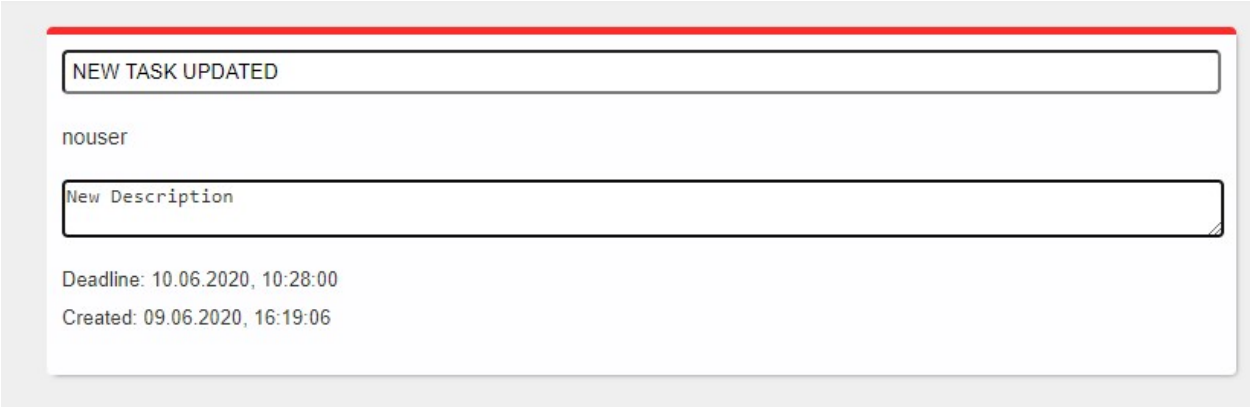

Рисунок 3.22 – Оновлення задачі

Рисунок 3.23 – Задачу було оновлено

Тестування функції оновлення дати дедлайну. Пріоритет – середній. Серйозність – середня. Кроки відтворення:

— Натиснути кнопку «Due Date».

— Вибрати дату (рис 5.17).

— Натиснути кнопку «Submit».

Очікуваний результат: дату дедлайну буде оновлено.

Фактичний результат: дату дедлайну було оновлено.

Результат продемонстровано на рисунку 3.24.

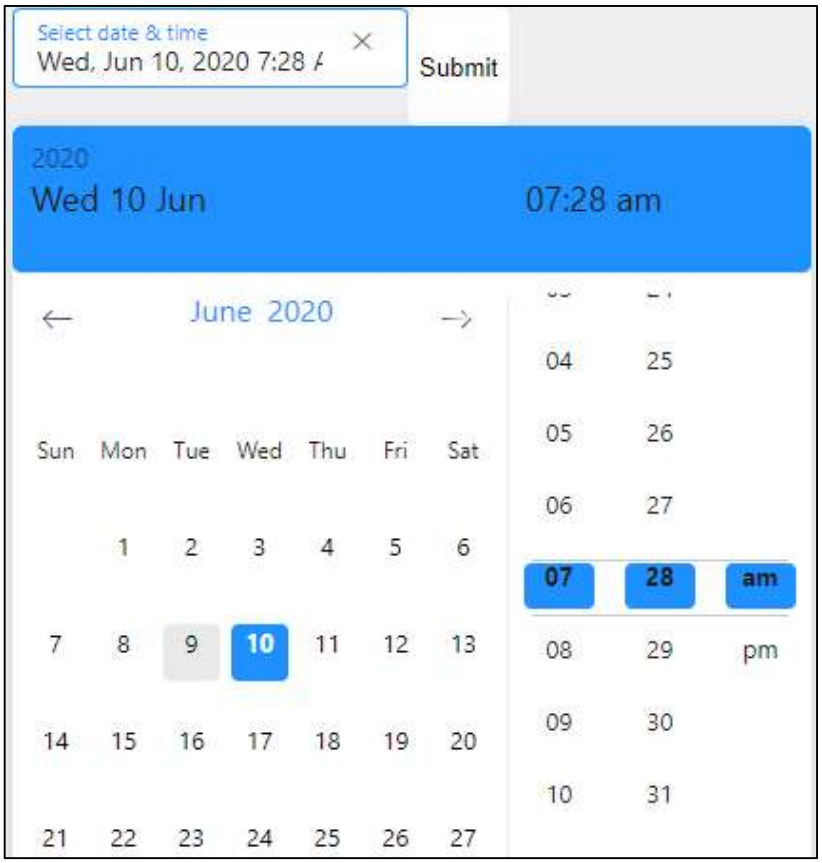

Рисунок 3.24 – Форма вибору дати дедлайну

Рисунок 3.25 – Дату дедлайну було оновлено

Тестування функції зміни кольору задачі. Пріоритет —середній. Серйозність — середня. Кроки відтворення:

— Натиснути кнопку «Change Color».

— Вибрати колір на палітрі (рис 5.19).

— Натиснути кнопку «Save».

Очікуваний результат: колір задачі буде змінено.

Фактичний результат: колір задачі було змінено.

Результат продемонстровано на рисунку 3.26.

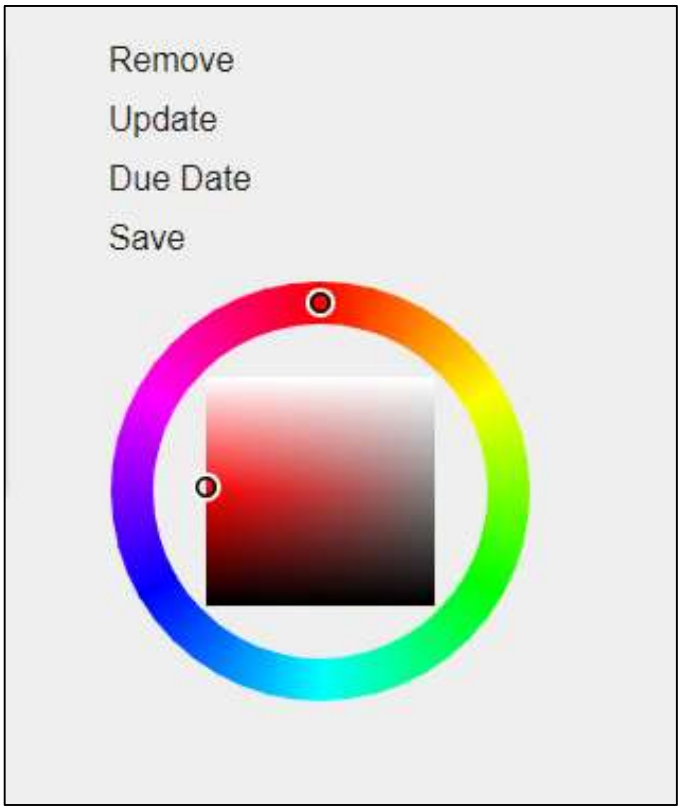

Рисунок 3.26 – Палітра вибору кольорів

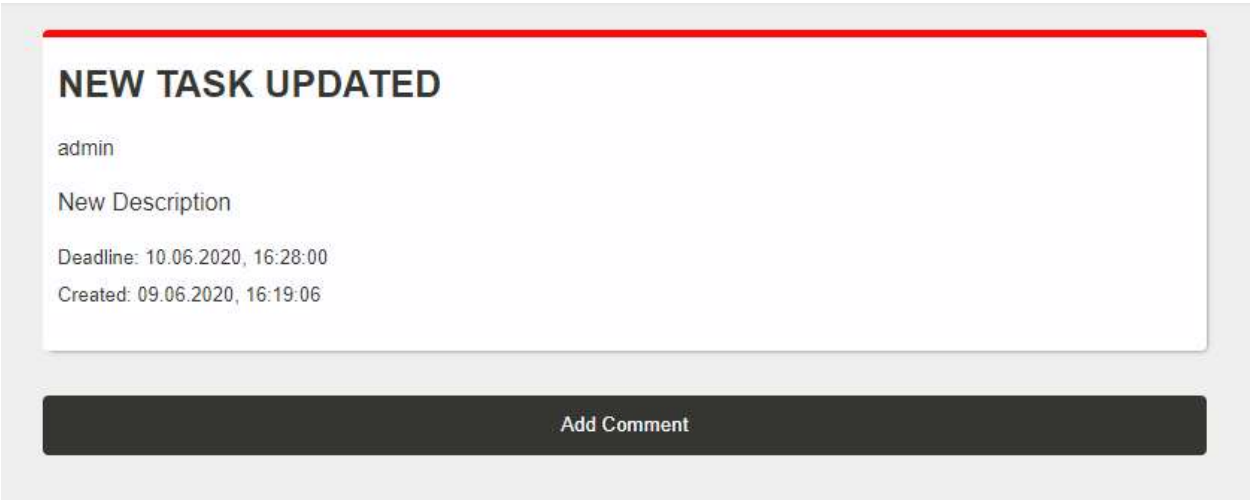

Рисунок 3.27 – Результат зміни кольору задачі

Тестування функції зміни власника задачі. Пріоритет – середній. Серйозність – середня. Кроки відтворення:

— Натиснути кнопку «Change Owner» (рис 5.21).

— Вибрати користувача «nouser» із випадаючого списку.

— Натиснути кнопку «Save».

Очікуваний результат: власника задачі буде змінено.

Фактичний результат: власника задачі було змінено.

Результат продемонстровано на рисунку 3.28.

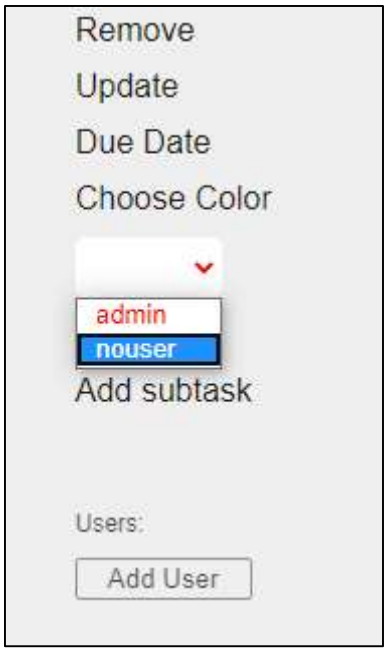

Рисунок 3.28 – Вибір користувача із списку

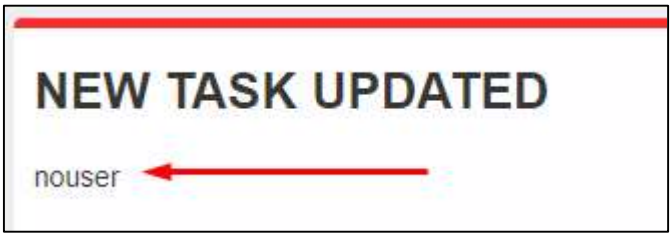

Рисунок 3.29 – Власника задачі змінено

Тестування функції створення підзадачі. Пріоритет – середній. Серйозність – середня. Кроки відтворення:

— Натиснути кнопку «Add Subtask».

— Ввести назву «New Subtask» (рис 5.23).

— Натиснути клавішу «Enter» на клавіатурі.

Очікуваний результат: буде створена підзадача, користувача буде переадресовано до створенної підзадачі.

Фактичний результат: була створена підзадача, користувача було переадресовано.

Результат продемонстровано на рисунку 3.30.

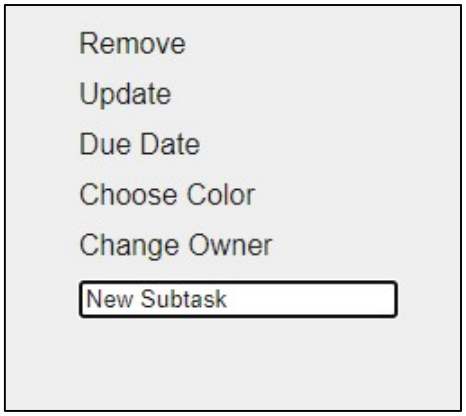

Рисунок 3.30 – Створення підзадачі

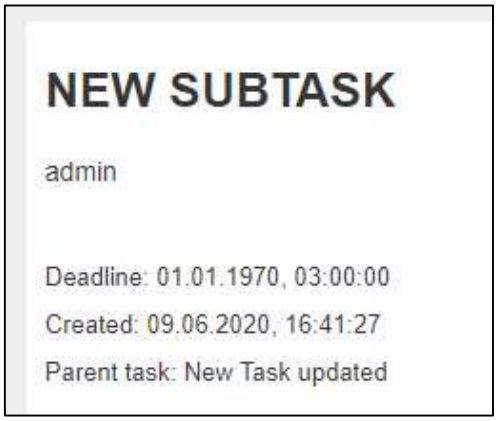

Рисунок 3.31 – Підзадача створена

Тестування функції створення коментарів. Пріоритет – середній. Серйозність – середня. Кроки відтворення:

— Натиснути кнопку «Add Comment».

— Ввести назву «New Comment» (рис 3.32).

— Натиснути клавішу «Enter» на клавіатурі.

Очікуваний результат: коментар буде створено.

Фактичний результат: коментар було створено.

Результат продемонстровано на рисунку 3.32.

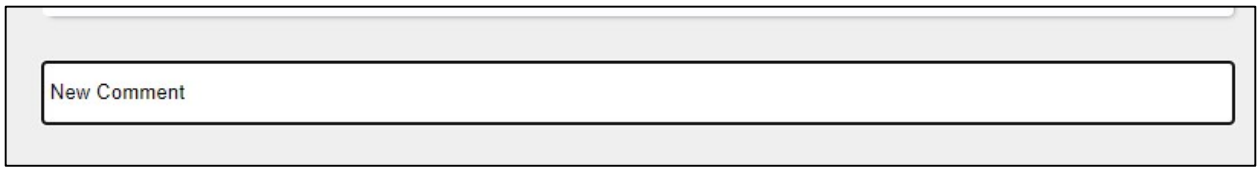

Рисунок 3.32 – Створення нового коментарю

Рисунок 3.33 – Коментар було створено

Тестування функції оновлення коментарів. Пріоритет – середній. Серйозність – середня. Кроки відтворення:

— Натиснути кнопку «Update».

— Ввести назву «New Comment Updated» (рис 5.27).

— Натиснути кнопку «Save».

Очікуваний результат: коментар буде оновлено.

Фактичний результат: коментар було оновлено.

Результат продемонстровано на рисунку 3.34.

Рисунок 3.34 – Оновлення коментарю

Рисунок 3.35 – Коментар було оновлено

Тестування функції видалення коментарів. Пріоритет – середній. Серйозність – середня. Кроки відтворення:

— Натиснути кнопку «Delete».

Очікуваний результат: коментар буде видален.

Фактичний результат: коментар було видалено.

Результат продемонстровано на рисунку 3.36.

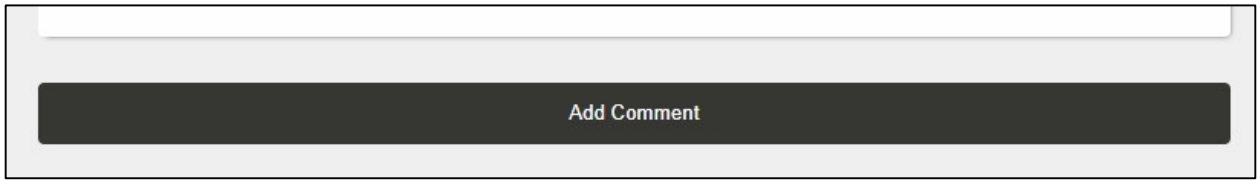

Рисунок 3.36 – Коментар було оновлено

Під час тестування інформаційної системи, було проведено перевірку заявленого функціоналу. За результатами тестування можна зазначити, що фактичний результат виконання відповідає очікуваному. Основуючись на отриманих даних, можна зробити висновок, що в системі був реалізований весь заявлений функціонал. Застосунок готовий до використання та дозволяє ефективно здійснювати управління задачами невеликих або середніх масштабів.

ВИСНОВКИ

У кваліфікаційній роботі було проаналізовано представлені на ринку існуючі рішення, розроблено логічну структуру бази даних, сформульовано задачі користувачів системи, спроєктовано структуру та алгоритміку інформаційної системи для управління задачами.

За результатами розробки інформаційної системи, завдяки функціоналу надання загального доступу, система може бути ефективно використана в особистих цілях та в невеликих проектних командах.

Дане рішення має хороший потенціал для подальшого розвитку, впроваджуючи який, він збільшить свою конкурентоспроможність та зможе залучити нову аудиторію користувачів.

Нижче представлені рекомендації для подальшого розвитку системи:

1. Впровадження гнучких налаштувань доступу

На даний момент в системі є два рівні доступу: адміністратор дошки та робітник. Якщо реалізувати можливість налаштовувати ролі доступу модульно через інтерфейс адміністратора, це сприятиме розвитку нових шляхів для застосування системи.

2. Впровадження системи звітності та контролю робочого часу.

Архітектура системи дозволяє впровадити рішення для контролю за робочим часом співробітників та звістності щодо стану проектів, що може значно вплинути на ефективність робочого процесу.

3. Налаштування PUSH-повідомлень

PUSH-повідомлення доможуть користувачам системи не пропускати важливі події у задачах, будуть сповіщувати про відповіді на їх комментарі або дії з задачами, до яких вони залучені.

4. Розробка мобільних додатків

Швидкий темп життя та робочих процесів робить користувачів вимогливими до скорості завантаження сайтів. Мобільні додатки швидше мобільних сайтів, та в них є й інші переваги:

- доступ до нативних функцій телефону, таких як камера і мікрофон;
- зручність у використанні та найкраща адаптація під мобільні пристрої.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Asana Guide [Електронний ресурс]. - Режим доступу до ресурсу: https://asana.com/guide
2. Як використовувати Trello, повний посібник [Електронний ресурс]. - Режим доступу до ресурсу: https://uk.gadget-info.com/81494-how-to-use-trello-a-complete-guide
3. To Do help & learning [Електронний ресурс]. - Режим доступу до ресурсу: https://support.office.com/en-us/todo
4. SWOT аналіз [електронний ресурс] – режим доступу: https://www.pharmencyclopedia.com.ua/article/1421/metod-swot
5. Метод "Модель Кано" [Електронний ресурс]. – Режим доступу до ресурсу: https://www.inventech.ru/pub/methods/metod-0022/
6. Three-Tier Architecture [Електронний ресурс]. – Режим доступу до ресурсу: https://www.techopedia.com/definition/24649/three-tier-architecture
7. What is Vue.js? [Електронний ресурс]. – Режим доступу до ресурсу: https://vuejs.org/v2/guide/
8. Про Node.js [Електронний ресурс]. – Режим доступу до ресурсу: https://nodejs.org/uk/about/
9. Use Cases [Електронний ресурс]. – Режим доступу до ресурсу: https://systems.education/use-case/
10. PostgreSQL: Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://www.postgresql.org/docs/

ДОДАТКИ

Додаток А. ПРОГРАМНИЙ КОД

App.js

```
import { useEffect } from 'react';
import './App.css';
import TodoForm from './components/TodoForm';
import TodoList from './components/TodoList';
import useStore from './hooks/useStore';

useStore.getState().fetchSettings();
useStore.getState().fetchTodos();

function App() {
  const { settings, loading, error } = useStore(
    ({ settings, loading, error }) => ({ settings, loading, error })
  );

  useEffect(() => {
    if (Object.keys(settings).length) {
      document.documentElement.style.fontSize =
settings.base_font_size;
      document.body.className = settings.theme;
    }
  }, [settings]);

  if (loading) return <h2>Loading...</h2>;

  if (error)
    return (
      <h3 className='error'>
        {error.message || 'Something went wrong. Try again later'}
```

```
      </h3>
    );

  return (
    <div className='App'>
      <h1>New Client</h1>
      <h2>{settings.greetings}</h2>
      <TodoForm />
      <TodoList />
    </div>
  );
}

export default App;
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

todo.api.js

```
const API_URI = 'http://localhost:5000/api/todo';

// метод для получения задач
const fetchTodos = async () => {
  try {
    const response = await fetch(API_URI);
```

```
    if (!response.ok) throw response;
    return await response.json();
  } catch (e) {
    throw e;
  }
};

// метод для создания новой задачи
const addTodo = async (newTodo) => {
  try {
    const response = await fetch(API_URI, {
      method: 'POST',
      body: JSON.stringify(newTodo),
      headers: {
        'Content-Type': 'application/json',
      },
    });
    if (!response.ok) throw response;
    return await response.json();
  } catch (e) {
    throw e;
  }
};

const updateTodo = async (id, changes) => {
  try {
    const response = await fetch(`${API_URI}/${id}`, {
      method: 'PUT',
      body: JSON.stringify(changes),
      headers: {
        'Content-Type': 'application/json',
      },
    });
    if (!response.ok) throw response;
    return await response.json();
```

```
  } catch (e) {
    throw e;
  }
};

// метод для удаления задачи
const removeTodo = async (id) => {
  try {
    const response = await fetch(`${API_URI}/${id}`, {
      method: 'DELETE',
    });
    if (!response.ok) throw response;
  } catch (e) {
    throw e;
  }
};

const todoApi = { fetchTodos, addTodo, updateTodo, removeTodo };

export default todoApi;
```

settings.api.js

```
const API_URI = 'http://localhost:5000/api/settings';

const fetchSettings = async () => {
  try {
    const response = await fetch(API_URI);
    if (!response.ok) throw response;
    return await response.json();
  } catch (e) {
    throw e;
  }
};

const settingsApi = { fetchSettings };
```

```
export default settingsApi;
```

TodoList.js

```
import useStore from '../hooks/useStore';

export default function TodoList() {
  const { todos, updateTodo, removeTodo } = useStore(
    ({ todos, updateTodo, removeTodo }) => ({ todos, updateTodo,
removeTodo })
  );

  return (
    <ul>
      {todos.map(({ id, text, done }) => (
        <li key={id}>
          <input
            type='checkbox'
            checked={done}
            onChange={() => {
              updateTodo(id, { done: !done });
            }}
          />
          <span>{text}</span>
          <button
            onClick={() => {
              removeTodo(id);
            }}
          >
            Remove
          </button>
        </li>
      ))}
    </ul>
  );
```

```
}
```

TodoForm.js

```
import { useState, useEffect } from 'react';
import useStore from '../hooks/useStore';

export default function TodoForm() {
  const addTodo = useStore(({ addTodo }) => addTodo);
  const [text, setText] = useState('');
  const [disable, setDisable] = useState(true);
  useEffect(() => {
    setDisable(!text.trim());
  }, [text]);
  const onChange = ({ target: { value } }) => {
    setText(value);
  };
  const onSubmit = (e) => {
    e.preventDefault();
    if (disable) return;
    const newTodo = {
      text,
      done: false,
    };
    addTodo(newTodo);
  };

  return (
    <form onSubmit={onSubmit}>
      <label htmlFor='text'>New todo text</label>
      <input type='text' id='text' value={text} onChange={onChange}
/>
      <button className='add'>Add</button>
    </form>
  );
}
```

useStore.js

```
import create from 'zustand';
// API для настроек
import settingsApi from '../api/settings.api';
// API для задач
import todoApi from '../api/todo.api';

const useStore = create((set, get) => ({
  settings: {},
  todos: [],
  loading: false,
  error: null,
  fetchSettings() {
    set({ loading: true });
    settingsApi
      .fetchSettings()
      .then((settings) => {
        set({ settings });
      })
      .catch((error) => {
        set({ error });
      })
      .finally(() => {
        set({ loading: false });
      });
  },
  fetchTodos() {
    set({ loading: true });
    todoApi
      .fetchTodos()
      .then((todos) => {
        set({ todos });
      })
      .catch((error) => {
```

```
      set({ error });
    })
    .finally(() => {
      set({ loading: false });
    });
},
addTodo(newTodo) {
  set({ loading: true });
  todoApi
    .addTodo(newTodo)
    .then((newTodo) => {
      const todos = [...get().todos, newTodo];
      set({ todos });
    })
    .catch((error) => {
      set({ error });
    })
    .finally(() => {
      set({ loading: false });
    });
},
updateTodo(id, changes) {
  set({ loading: true });
  todoApi
    .updateTodo(id, changes)
    .then((updatedTodo) => {
      const todos = get().todos.map((todo) =>
        todo.id === updatedTodo.id ? updatedTodo : todo
      );
      set({ todos });
    })
    .catch((error) => {
      set({ error });
    })
    .finally(() => {
```

```
        set({ loading: false });
      });
  },
  // удаления задачи
  removeTodo(id) {
    set({ loading: true });
    todoApi
      .removeTodo(id)
      .then(() => {
        const todos = get().todos.filter((todo) => todo.id !== id);
        set({ todos });
      })
      .catch((error) => {
        set({ error });
      })
      .finally(() => {
        set({ loading: false });
      });
  },
}));

export default useStore;
```