МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

ІВАШЕНЮК Сергій Олександрович

Веб-застосунок для керування завданнями в малих робочих групах / Web Application for Task Management in Small Working Groups

спеціальність: 122 - Комп'ютерні науки
освітньо-професійна програма - Комп'ютерні науки

Кваліфікаційна робота

Виконав студент групи КН-41
Івашенюк С.О.

_____

Науковий керівник
к.т.н., доцент Д.І. Загородня

Кваліфікаційну роботу допущено до захисту
«___»_____ 20___ р.
Завідувач кафедри
_____ Н. В. Дзюбановська

Тернопіль – 2026

Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність 122 – Комп'ютерні науки
освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Н.В. Дзюбановська
«____» _____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
ІВАШЕНЮК Сергій Олександрович
(прізвище, ім'я, по батькові)

1. Веб-застосунок для керування завданнями в малих робочих групах / Web Application for Task Management in Small Working Groups
керівник роботи Д. І. Загородня к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 20 грудня 2025 р. № 938.
2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2026 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити:

– здійснити аналіз предметної області керування завданнями в малих робочих групах та огляд існуючих веб-застосунків (Trello, Asana, Jira), визначити їх функціональні можливості;

– розробити архітектурне, алгоритмічне та інформаційне забезпечення системи, сформувати загальну структуру веб-застосунку;

– реалізувати програмне забезпечення веб-застосунку, створити інтерфейс користувача та провести тестування системи.

5. Перелік графічного матеріалу в роботі:

– структурна схема (архітектура) веб-застосунку для керування завданнями;

– діаграма послідовності взаємодії користувачів із системою;

– результати тестування програмного забезпечення (таблиці, діаграми, скріншоти роботи системи).

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 1 грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи	до 01.01. 2026 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.02. 2026 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 15.03.2026 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 01.05. 2026 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2026 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2026 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2026 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 10.06.2026 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 10.06. 2026 р.	

Студент_____ С.О. Івашенюк_____
(підпис) (прізвище та ініціали)

Керівник кваліфікаційної роботи_____ Д. І. Загородня
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Веб-застосунок для керування завданнями в малих робочих групах» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки» написана обсягом у 62 сторінки і містить 30 ілюстрацій, 3 таблиці, 3 додатки та 25 використаних джерел.

Метою роботи є дослідження сучасних підходів до розробки веб-застосунків для керування завданнями в малих робочих групах та створення програмного засобу, який забезпечує ефективну організацію, розподіл і контроль виконання завдань між користувачами.

Методами розроблення обрано метод аналізу (для дослідження існуючих систем керування завданнями), метод синтезу (для поєднання функціональних компонентів системи), методи проєктування інформаційних систем (для побудови архітектури веб-застосунку), метод моделювання (для розробки логіки взаємодії користувачів із системою), а також експериментальні методи тестування (для перевірки функціональності та зручності використання системи).

Внаслідок виконання роботи обґрунтовано доцільність використання веб-технологій для організації спільної роботи в малих групах, розроблено архітектуру та реалізовано веб-застосунок для керування завданнями, який забезпечує створення, редагування та контроль виконання завдань, а також взаємодію між учасниками проєкту.

Ключові слова: ВЕБ-ЗАСТОСУНОК, КЕРУВАННЯ ЗАВДАННЯМИ, МАЛІ РОБОЧІ ГРУПИ, СИСТЕМА УПРАВЛІННЯ, ІНТЕРФЕЙС КОРИСТУВАЧА, БАЗА ДАНИХ, JAVASCRIPT, API.

ANNOTATION

The qualification work on the topic "Web application for task management in small work groups" for obtaining the educational degree "bachelor" in specialty 122 "Computer Science" of the educational program "Computer Science" is written in a volume of 62 pages and contains 30 illustrations, 3 tables, 3 appendices and 25 used sources.

The purpose of the work is to study modern approaches to the development of web applications for task management in small work groups and to create a software tool that provides effective organization, distribution and control of task performance between users.

The development methods chosen are the analysis method (for studying existing task management systems), the synthesis method (for combining functional components of the system), information systems design methods (for building the architecture of the web application), the modeling method (for developing the logic of user interaction with the system), as well as experimental testing methods (for testing the functionality and usability of the system).

As a result of the work, the feasibility of using web technologies for organizing joint work in small groups was substantiated, the architecture was developed and a web application for task management was implemented, which provides the creation, editing and control of task execution, as well as interaction between project participants.

Keywords: WEB APPLICATION, TASK MANAGEMENT, SMALL WORK GROUPS, MANAGEMENT SYSTEM, USER INTERFACE, DATABASE, JAVASCRIPT, API.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

ORM – Object-Relational Mapping

UML – Unified Modeling Language

ERD – Entity-Relationship Diagram

DFD – Data Flow Diagram

DB – Database (база даних)

UI – User Interface

UX – User Experience

HTTP – Hyper Text Transfer Protocol

REST – Representational State Transfer

PK – Primary Key

FK – Foreign Key

CRUD – Create, Read, Update, Delete

ВСТУП

На сучасному етапі розвитку інформаційних технологій та цифровізації бізнес-процесів критично важливим стає ефективне управління командною роботою. Збільшення обсягу проєктів, підвищення вимог до швидкості їх виконання та необхідність постійної комунікації між учасниками зумовлюють потребу у застосуванні спеціалізованих програмних рішень для керування завданнями.

Ця проблема є особливо гострою для невеликих робочих груп, які характеризуються обмеженим складом та часто недостатніми ресурсами для впровадження громіздких корпоративних систем. У таких умовах ключовим є надання простого, зручного та водночас функціонального інструменту, що дозволить ефективно планувати діяльність, розподіляти обов'язки, моніторити їх виконання та налагоджувати взаємодію між членами команди.

На сьогодні на ринку програмного забезпечення представлена значна кількість продуктів для управління завданнями, таких як Trello, Asana, Monday та інші. Проте більшість із них або пропонують обмежений функціонал у безоплатних версіях, або є занадто складними для освоєння невеликими колективами, що знижує ефективність їхнього застосування у малих групах.

Отже, постає необхідність у створенні веб-застосунку, який би оптимально поєднував легкість використання, достатній функціонал та ефективні механізми управління завданнями, орієнтовані на потреби невеликих робочих груп.

Основною метою даної дипломної роботи є проєктування та розробка веб-застосунку для управління завданнями в малих робочих групах, здатного автоматизувати процеси планування, розподілу та контролю їх виконання.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– проаналізувати предметну область управління завданнями;

– дослідити наявні програмні аналоги та їх функціонал;

– сформулювати функціональні та нефункціональні вимоги до майбутньої системи;

– розробити архітектуру веб-застосунку;
– спроєктувати інформаційну модель даних;
– здійснити програмну реалізацію системи;
– провести тестування створеного продукту.

Об'єктом дослідження є процес організації та управління завданнями в контексті малих робочих груп.

Предметом дослідження виступають методології, моделі та програмні засоби, що використовуються для реалізації веб-застосунків, призначених для керування та моніторингу виконання завдань.

Практична цінність отриманих результатів полягає у можливості застосування розробленого веб-застосунку для підвищення ефективності роботи малих команд, оптимізації процесів планування та посилення контролю за виконанням завдань.

Структура дипломної роботи складається з трьох логічно послідовних розділів, у яких послідовно висвітлено аналіз предметної області, етапи проєктування системи, програмну реалізацію та результати її тестування.

Загальний обсяг роботи складає 61 сторінки і містить 30 рисунок, три таблиці, три додатки та 25 використаних джерела.

Апробацію дослідження проведено на Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Інтелектуальні комп’ютерні системи та мережі», яка відбулася 20 травня 2026р. (м. Тернопіль, Україна). Копія публікації представлена в додатку А.

Програмна реалізація розробленої системи представлена у вигляді вихідного коду, який наведено в додатках. Основні фрагменти програмного коду подано у додатку Б,В.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧ ПРОЄКТУВАННЯ ВЕБ-ЗАСТОСУНОКУ

1.1 Опис предметної області Веб

Проблема ефективного управління завданнями виникла ще на початкових етапах організації колективної праці. До появи спеціалізованого програмного забезпечення, невеликі робочі групи застосовували звичні методи планування:

– ручні записи та візуальні дошки. На початковому етапі команди фіксували завдання у блокнотах, на дошках або плакатах, розподіляли їх серед учасників та відмічали прогрес. Хоча цей метод був простим, він ускладнював оперативну координацію діяльності та не надавав можливостей для зручного аналізу результативності.

– цифрові таблиці (наприклад, Excel). У 1990-х та 2000-х роках електронні таблиці набули популярності для моніторингу завдань. Вони надавали можливість зберігати значний обсяг даних, сортувати та фільтрувати їх, проте не підтримували інтерактивної взаємодії між учасниками та не надавали сповіщень про оновлення.

– комунікаційні засоби (месенджери та електронна пошта). Для обміну інформацією про завдання використовувалися поштові сервіси, чати або форуми. Це сприяло обміну відомостями, але ускладнювало централізований контроль і збільшувало ймовірність втрати суттєвої інформації.

З розвитком інтернет-технологій почали з'являтися перші спеціалізовані веб-орієнтовані системи для управління завданнями. Їхня еволюція відбувалася у кілька стадій:

1. Початкова стадія - елементарні онлайн-списки справ (to-do lists): Ці системи надавали користувачам функціонал для створення завдань, встановлення термінів та фіксації їх завершення, але не передбачали можливостей для спільної роботи команди.

2. Етап візуального менеджменту (Kanban-дошки): Програми, подібні до Trello, запровадили інтуїтивно зрозуміле візуальне відображення завдань за

допомогою дощок, колонок і карток, що значно спростило контроль над процесом та моніторинг ходу виконання.[5]

3. Сучасна стадія - багатофункціональні платформи для командної співпраці (наприклад, Asana, Monday): Об'єднання аналітичних функцій, автоматизації та комунікаційних засобів дозволяє не лише моніторити завдання, а й планувати ресурси, контролювати навантаження учасників команди та оцінювати загальну продуктивність.[7]

Таким чином, еволюція рішень для управління завданнями пройшла шлях від ручних методів та електронних таблиць до сучасних веб-додатків з вбудованими інструментами контролю, аналізу та комунікації. Для малих колективів основними критеріями результативності залишаються простота використання, швидкість адаптації та чіткість відображення статусу завдань, що й лягло в основу розробки сучасних веб-рішень.

Дана робота зосереджена на процесі управління завданнями в невеликих робочих групах за допомогою веб-додатків. У колективах невеликого розміру (від 3 до 15 осіб) часто постає питання ефективного розподілу функцій, моніторингу їх завершення та обміну відомостями між співробітниками. Застосування спеціалізованого веб-додатку дає змогу автоматизувати зазначені процеси та покращити результативність командної взаємодії.

Ключовою метою такої системи є організація, планування та моніторинг виконання завдань членами робочої групи. Веб-додаток гарантує централізоване збереження даних щодо завдань, користувачів, термінів, поточних станів та підсумків діяльності.

До ключових функцій процесу управління завданнями відносяться:

– формування та зміна завдань;
– призначення відповідальних осіб;
– визначення пріоритетності та встановлення термінів;
– моніторинг поточного стану виконання;
– взаємообмін коментарями та файлами між учасниками;
– огляд переліку завдань та можливість їх фільтрації;

– генерування звітів про виконану роботу.

У процесі управління завданнями виокремлюють транзакційний та аналітичний компоненти.

При моделюванні даної предметної області необхідно визначити основні функціональні можливості системи управління завданнями. До них належать управління обліковими записами користувачів, створення та керівництво проєктами, обробка завдань, відстеження ходу виконання робіт, взаємодія між учасниками колективу та аналіз результатів діяльності.

Для відображення функціональної структури системи використовується функціональна блок-схема, що ілюструє ключові функціональні модулі веб-додатку.

На рисунку 1.1 представлена схема відображає основні функціональні компоненти системи. Її ядром є система управління завданнями для малих робочих груп, від якої відгалужуються такі ключові функціональні модулі: керування користувачами, менеджмент проєктів та команд, розподіл завдань, моніторинг виконання, обговорення завдань та формування аналітичних звітів.

Рисунок 1.1- Дерево функцій системи керування завданнями

Операційний блок - це частина системи управління проєктами, яка відповідає за основні дії з інформацією та допомагає системі виконувати її головні можливості. Його мета - обробляти запити від користувачів, працювати з базою даних та підтримувати дані в системі в актуальному стані.

Серед його функцій - створення нових проєктів і завдань, зміна вже наявних записів та видалення старої або непотрібної інформації. Крім того, цей блок дає змогу змінювати статуси завдань відповідно до етапів їх виконання, що допомагає стежити за ходом роботи над проєктом у реальному часі.

Важливою частиною операційного блоку є керування користувачами та їхніми ролями. Для цього передбачені механізми реєстрації, входу в систему та перевірки користувачів, а також визначення, хто до яких можливостей системи має доступ.

Також операційний блок дозволяє працювати з додатковою інформацією щодо завдань, наприклад, додавати коментарі, прикріплювати файли та призначати відповідальних за їх виконання. Усі внесені зміни автоматично зберігаються і стають доступними для інших учасників проєкту.

Щоб ефективно контролювати роботу, операційний блок збирає та оновлює дані про поточні та завершені завдання. Це дозволяє користувачам отримувати свіжу інформацію про стан проєкту та приймати обґрунтовані рішення щодо керування ним.

Аналітичний інструмент призначений для збирання, обробки та аналізу інформації, що накопичується під час функціонування системи управління проєктами. Його ключове завдання - надавати користувачам актуальні відомості про прогрес виконання завдань та підтримувати ухвалення керівних рішень, ґрунтуючись на об'єктивних даних.

Цей інструмент дозволяє аналізувати хід виконання завдань за різноманітними показниками. Наприклад, користувачі можуть відстежувати активність та результати діяльності протягом визначених часових інтервалів, що допомагає оцінювати розвиток проєктів, виявляти періоди найвищої ефективності та можливі відставання.

Крім цього, аналітичний інструмент дозволяє оцінювати ефективність роботи індивідуальних виконавців. На підставі зібраної інформації надаються відомості про кількість завершених, поточних та прострочених завдань, що сприяє кращому розподілу робочого навантаження серед членів команди.

Однією зі значущих функцій модуля є аналіз даних у контексті окремих проєктів або робочих команд. Це дає змогу менеджерам відслідковувати прогрес реалізації кожного проєкту, контролювати досягнення поставлених цілей та оцінювати продуктивність команд.

Додатково, модуль проводить аналіз завдань за їхнім поточним станом та ступенем пріоритетності. Це дозволяє користувачам оперативно виявляти ключові завдання, моніторити їхнє виконання та вчасно реагувати на можливі ризиби, пов'язані зі зволіканнями або перевантаженням співробітників.

Отже, створення веб-додатку для управління завданнями дозволяє автоматизувати процеси планування та контролю в невеликих колективах, підвищити прозорість виконання задач та забезпечити надійну інформаційну підтримку для прийняття управлінських рішень.

1.2 Огляд і дослідження аналогів сервісів для керування завданнями

Для створення веб-сервісу для керування завданнями необхідно проаналізувати доступні програмні рішення зі схожим функціоналом. Серед провідних інструментів для управління проектами та завданнями можна назвати Trello, Asana і Monday [7].

Проведений аналіз існуючих систем (таблиця 1.1) показав, що платформи Trello, Asana та Monday забезпечують широкий набір інструментів для управління проєктами та завданнями. Водночас для невеликих робочих груп їх використання не завжди є доцільним через надлишковий функціонал, складність налаштування або обмеження безкоштовних тарифних планів.

На відміну від розглянутих аналогів, розроблюваний веб-застосунок орієнтований саме на невеликі команди та передбачає спрощений інтерфейс,

мінімальну кількість дій для створення і контролю завдань, а також швидке впровадження без додаткового навчання користувачів. Це дозволяє підвищити ефективність командної взаємодії та скоротити час на організацію робочого процесу.

Таблиця 1.1 – Порівняння сервісів для керування завданнями

Критерій	Trello	Asana	Monday
Тип інтерфейсу	Канбан-дошки	Списки,таймлайни, канбан	Таблиці,канбан, таймлайн
Простота використання	Дуже проста	Середня	Складна
Гнучкість	Обмежена	Висока	Дуже висока
Функціональність	базова	розширена	Дуже широка
Підходить для	Малих команд	Середніх команд	Великих команд
Візуалізація	Канбан	Канбан+графіки	Канбан+таблиці +дашборди
Ціна	Є безкоштовний план	Є безкоштовний план	Обмежений безкоштовний

1.2.1 Аналіз платформи Trello

Trello є веб-додатком, призначеним для управління проектами та завданнями, що використовує підхід Kanban-дошок. Він дозволяє візуально відстежувати хід робіт, організовуючи завдання у вигляді дощок, списків та карток.[5]

Основні можливості платформи Trello сприяють продуктивній організації робочих процесів та контролю за виконанням завдань у колективах будь-якого розміру. Передусім, система дає змогу створювати індивідуальні дошки для кожного проєкту, що дозволяє систематизувати діяльність відповідно до напрямків роботи або стадій виконання. Кожна така дошка є візуальним середовищем, де можна впорядкувати всі завдання, пов'язані з конкретним проєктом.

Для уточнення деталей роботи застосовуються картки, кожна з яких призначається для окремого завдання. У картці можна викласти основний зміст завдання, долучити потрібні файли та встановити критерії її виконання. Ключовою опцією є функція призначати виконавців, це сприяє ясному розподілу функцій та збільшує відповідальність учасників команди.

Крім того, платформа дає змогу визначати кінцеві дати для завдань, що допомагає відстежувати терміни і гарантує своєчасну реалізацію проєктів. Для оптимізації взаємодії між членами команди доступна функція додавання коментарів, де можна з'ясовувати нюанси, дискутувати щодо прогресу або залишати відгуки. Користувачі також мають змогу прикріплювати документи, це істотно полегшує обмін файлами та іншими матеріалами.

Додатково Trello надає підтримку для застосування позначок та списків перевірки. Позначки дають можливість сортувати завдання за встановленими параметрами у той час як списки перевірки сприяють поділу комплексних завдань на менші етапи і контролювати хід їх реалізації.

Користувацький інтерфейс системи є легким для освоєння та містить декілька ключових компонентів. Головною складовою є загальна інформаційна панель, на якій представлений список усіх проєктів. Це дає можливість оперативно перемикатися між ними і відстежувати загальний прогрес діяльності. Ключовим робочим інструментом є дошка із задачами, що типово організовані за стадіями виконання, як-от “До виконання”, “У роботі” та “Завершено”. Цей принцип втілює засади канбан-методики і забезпечує прозорість робочого потоку.

Для кожної задачі передбачена окрема розгорнута картка, яка містить вичерпну інформацію: опис, додані файли, коментарі, призначених виконавців та кінцеві терміни. Такий підхід дозволяє зібрати всі релевантні дані в єдиному сховищі і запобігти втраті критично важливих відомостей.

Проте, система також має деякі слабкі сторони. Наприклад, безкоштовний варіант Trello пропонує обмежену кількість функцій, що може виявитися неадекватним для масштабніших проєктів. Окрім того, її ефективність знижується при управлінні великими або багатоступеневими проєктами, для яких необхідні

більш просунуті засоби планування та моніторингу. Слід також зазначити лімітовані аналітичні функції платформи, це ускладнює ретельний аналіз продуктивності або передбачення результатів.

Отже, Trello представляє собою практичний та дієвий інструмент для організації роботи невеликих колективів та проєктів середньої складності. Його легкість у використанні, візуальна доступність та основний функціонал роблять його найкращим рішенням для тих користувачів, кому потрібне оперативне впровадження системи контролю за завданнями без значних часових витрат на освоєння. [5]

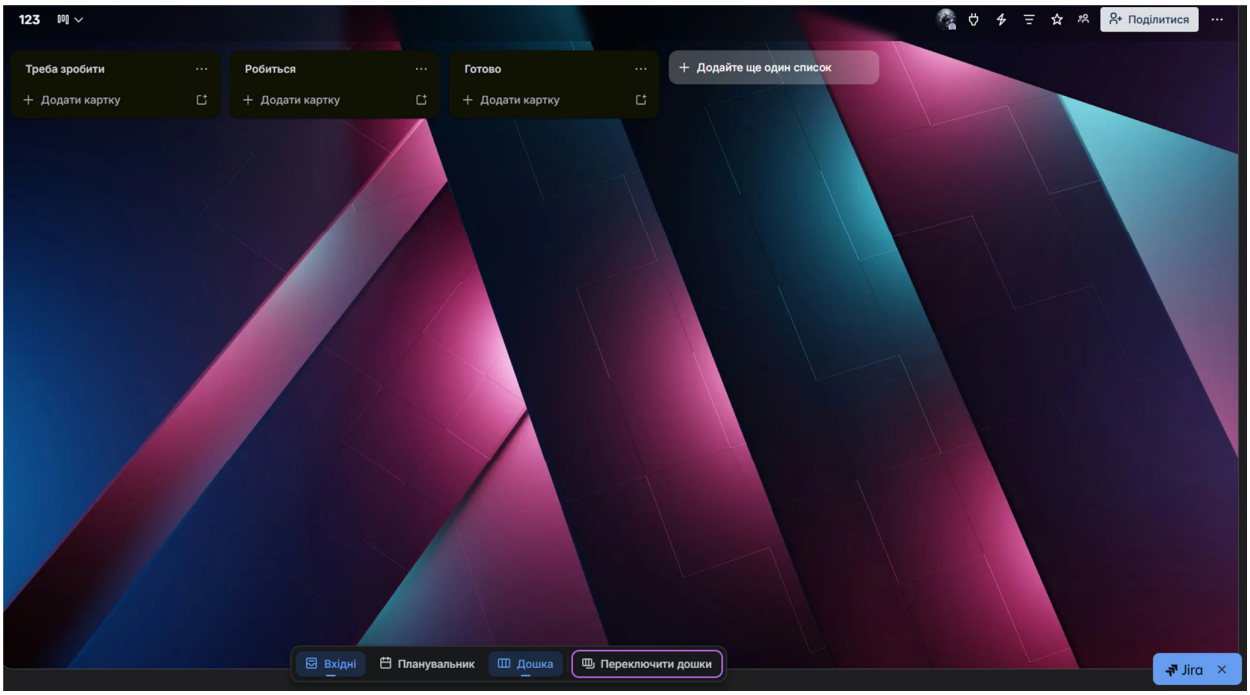

Рисунок 1.2 - Інтерфейс Trello

1.2.2 Аналіз платформи □s□n□

Asana - це інструмент для управління завданнями та проектами, який допомагає командам ефективно планувати діяльність, відстежувати її прогрес та узгоджувати співпрацю між учасниками.[6]

Ключові можливості платформи призначені для оптимізованого керування завданнями та налагодження взаємодії в межах проєктів. Однією з головних можливостей є формування та впорядкування завдань, що дозволяє

упорядковувати робочий потік, поділяючи його на чіткі кроки. Кожне завдання може бути розширене за вимогою користувачів, що допомагає краще усвідомити поставлені цілі.

Суттєвою особливістю є можливість призначення виконавців за певні завдання. Це дозволяє чітко окреслити обов'язки в команді, покращує організацію та гарантує відкритість виконання робіт. Додатково платформа пропонує засоби для визначення часових обмежень, що дає змогу відстежувати терміни та графікувати виконання робіт.

Можливість створення списків і календарів завдань сприяє комфортному об'єднанню завдань та їх наочному представленню. Списки дозволяють організовувати завдання відповідно до критеріїв, наприклад, за етапами чи пріоритетом, тоді як календар допомагає слідкувати за датами завершення та розподіляти робоче навантаження.

Для налагодження співпраці в команді доступні засоби для обміну думками щодо завдань через систему коментарів. Це дозволяє учасникам проєкту ділитися відомостями, конкретизувати подробиці, вирішувати спірні моменти та швидко ухвалювати рішення, не вдаючись до інших каналів зв'язку.

Інтерфейс платформи є зрозуміло побудованим і простим в освоєнні. Одним із головних компонентів є основний робочий простір який пропонує всебічний погляд на всі активні проєкти та дозволяє оперативно змінювати фокус між ними. У межах кожного проєкту відображається список завдань, що дає змогу користувачам бачити весь обсяг завдань та стежити за прогресом.

Кожне завдання має окрему картку з розгорнутою інформацією, яка містить ключові дані: опис, актуальний стан, призначених виконавців та коментарі. Це забезпечує єдине місце для зберігання інформації та полегшує пошук необхідних даних. Додатково платформа містить календар, який використовується для контролю дат завершення завдань, що сприяє раціональному розподілу часу та виконанню завдань вчасно.

Отже, описаний функціонал та компоненти системи створюють цілісний підхід до управління проєктами, об'єднуючи засоби планування, контролю та комунікації на одній платформі.

До недоліків Asana відносять:

– складніший інтерфейс у порівнянні з простішими інструментами управління завданнями;

– обмежені функції в безоплатній версії;

– потенційно надлишковий набір функцій для малих колективів.

Платформа Asana пропонує широкий функціонал, що робить її придатною для управління комплексними проектами. Однак велика кількість деталей може ускладнити використання для невеликих груп із незначною кількістю завдань.[6]

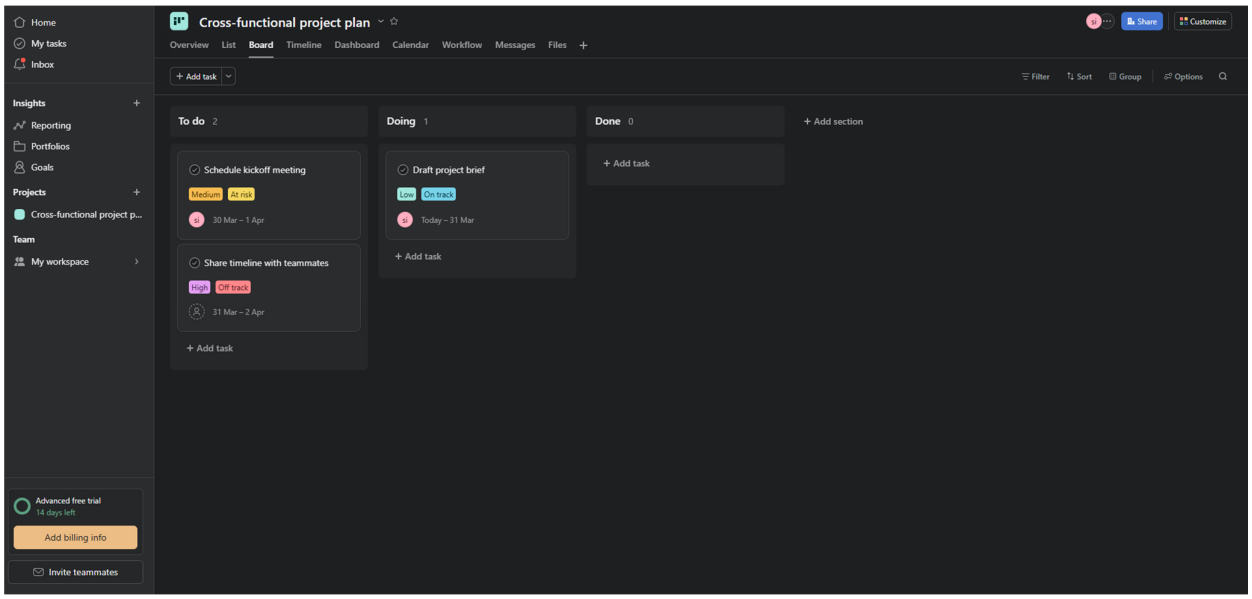

Рисунок 1.3 - Інтерфейс Asana

1.2.3 Аналіз платформи□ □nd□□

Monday - це сучасний інструмент для управління та координації робочих процесів, що пропонує візуальні засоби для контролю виконання завдань та оптимізації командної комунікації.[7]

Основні можливості платформи Monday спрямовані на всебічне адміністрування проєктів та підвищення ефективності робочих процесів.

Насамперед, система надає змогу створювати проєкти й окремі завдання, що дозволяє організувати діяльність організації згідно з визначеними цілями. Кожен проєкт може містити безліч взаємопов'язаних завдань, які деталізують етапи його виконання.

Ключовим елементом функціоналу є опція призначення відповідальних виконавців для кожного завдання. Це сприяє чіткому розподілу ролей у команді, підвищує рівень підзвітності та забезпечує прозорість виконання робіт. Крім того, платформа дозволяє встановлювати кінцеві терміни для завдань, що допомагає контролювати дотримання дедлайнів та раціонально планувати робочий час.

Значна увага приділяється автоматизації рутинних операцій. Monday.com пропонує інструменти для автоматичного здійснення повторюваних дій, таких як зміна статусів, відправлення сповіщень або оновлення інформації. Це суттєво зменшує завантаженість користувачів і дає змогу сфокусуватися на більш важливих аспектах роботи. Додатково система забезпечує можливість моніторингу загального прогресу проєкту, що дозволяє своєчасно виявляти труднощі та вносити корективи до плану дій.

Користувацький інтерфейс платформи є сучасним і багатофункціональним. Центральним компонентом є головна інформаційна дошка яка надає зведені дані про всі активні проєкти та їхній поточний стан. Завдання можуть бути представлені у вигляді таблиці або дошки, що забезпечує гнучкість у виборі способу візуалізації даних відповідно до потреб користувача.

Для кожного завдання передбачено окреме вікно редагування деталей, де можна змінювати опис, статус, призначених осіб та інші характеристики. Це дозволяє зберігати всю ключову інформацію в одному місці та оперативно її актуалізувати. Також платформа пропонує аналітичні діаграми, які використовуються для оцінки ходу виконання проєктів, аналізу продуктивності команди та ухвалення управлінських рішень на основі зібраної інформації.

Водночас, платформа Monday має й певні недоліки. Зокрема, вона відзначається порівняно високою вартістю платних тарифних планів, що може бути стримуючим фактором.

Перевагою цієї платформи є її розширені можливості та потужні інструменти автоматизації процесів. Водночас Monday краще підходить для середніх та великих команд.[7]

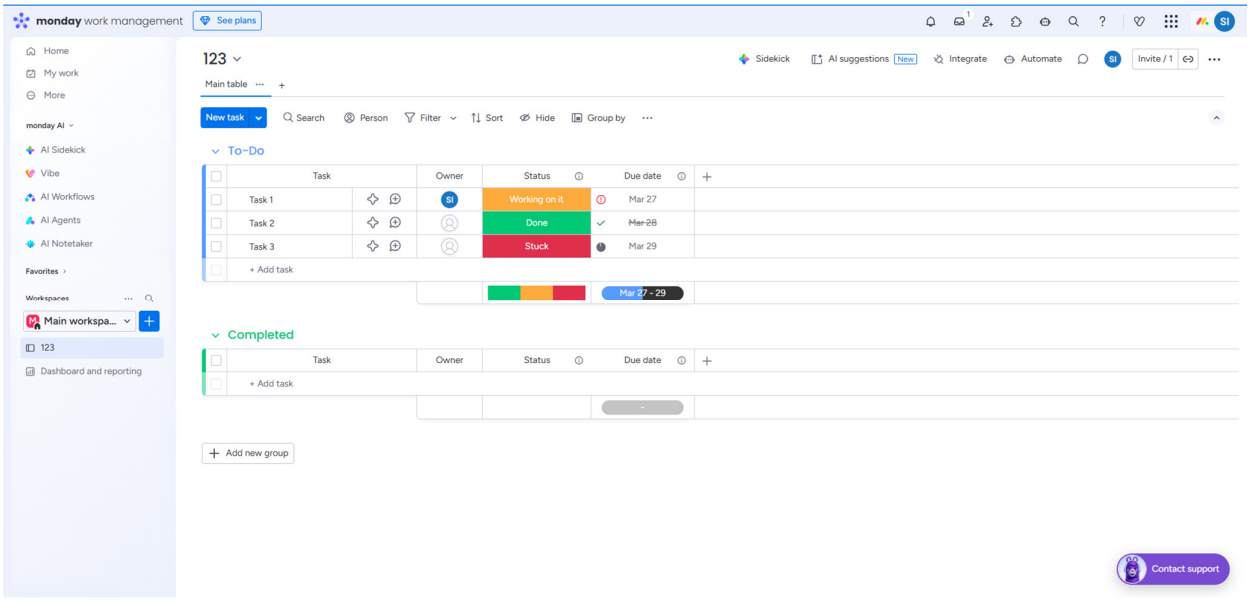

Рисунок 1.4-Інтерфейс Monday

1.3 Постановка задачі дослідження

За результатами проведеного аналізу предметної області та існуючих аналогічних систем управління завданнями було сформульовано ключові вимоги до майбутнього веб-додатка для організації роботи з дорученнями в невеликих робочих колективах.[7]

Об'єктом цього дослідження є процедура організації та моніторингу виконання завдань у невеликих робочих групах. Предметом дослідження виступають методики та програмні рішення для візуалізації процесу керування дорученнями за допомогою веб-додатків.

Ключовою проблемою в діяльності невеликих команд є відсутність уніфікованої системи для зберігання інформації про завдання, відстеження термінів виконання та забезпечення ефективної комунікації між учасниками. Нерідко дані обмінюються за допомогою різноманітних месенджерів або електронної пошти, що

ускладнює відстеження прогресу роботи та може призвести до втрати цінної інформації.

Для вирішення цієї проблеми пропонується розробити веб-додаток для управління завданнями, що забезпечить централізоване керівництво роботою команди, оптимізацію планування та ефективний контроль за виконанням доручень.

Основними функціональними вимогами до системи є:

– реєстрація та вхід користувачів до системи;
– створення, редагування та видалення завдань;
– призначення виконавців для кожного завдання;
– визначення пріоритету та встановлення термінів виконання;
– оновлення статусу завдань (наприклад: нове, у роботі, виконано);
– перегляд списку завдань та можливість їх фільтрації;
– можливість додавати коментарі до завдань;
– ведення історії змін та результатів виконання завдань.

До головних нефункціональних вимог відносяться:

– інтуїтивно зрозумілий та зручний користувацький інтерфейс;
– доступність системи через будь-який веб-браузер без потреби в додатковому програмному забезпеченні;
– гарантування збереження та захисту даних;
– підтримка одночасної роботи кількох користувачів.

Розроблювана система повинна підвищити ефективність організації роботи в невеликих колективах шляхом автоматизації процесів планування, розподілу та контролю за виконанням завдань.

Для формалізації вимог до програмного продукту було визначено основні ролі користувачів системи:

– адміністратор системи;
– керівник проєкту;
– учасник команди.

Адміністратор відповідає за підтримку працездатності системи та керування користувачами. Керівник проєкту здійснює створення проєктів, призначення виконавців та контроль виконання завдань. Учасник команди виконує призначені завдання, оновлює їх статус та бере участь в обговоренні робочих питань.

Отже, основною метою дослідження є проєктування та впровадження веб-додатку для управління завданнями, який гарантуватиме зручний моніторинг робочих процесів, збільшить продуктивність команди та посилить контроль за виконанням поставлених доручень.

2. АЛГОРИТМІЧНЕ ТА ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ВЕБ-ЗАСТОСУНОКУ

2.1 Загальна структура проєктованої системи

Програма надає централізоване керування відомостями про користувачів, доручення, зауваження, часові межі виконання та стани робіт. Вона дає змогу організувати дієву взаємодію між членами команди, полегшує процес планування роботи та посилює нагляд за виконанням визначених завдань.

Запроєктована програма утілюється у вигляді веб-застосунку, що функціонує за клієнт-серверною будовою. Користувачі спілкуються із програмою через веб-інтерфейс у браузері, а опрацювання запитів відбувається на сервері. Відомості зберігаються у сховищі даних.

Загальна будова програми містить такі головні складові:

– клієнтська ланка (Frontend) - забезпечує взаємодію користувача із програмою через веб-інтерфейс;

– серверна ланка (Backend) - виконує опрацювання запитів користувачів та реалізує бізнес-логіку;

– сховище даних - забезпечує збереження та опрацювання відомостей про користувачів, проєкти та доручення;

Головні функціональні блоки програми:

1. Блок керування користувачами. Забезпечує:
   – реєстрація користувачів;
   – вхід у програму;
   – керування особистими даними користувачів;
2. Блок керування проєктами. Забезпечує:
   – створення проєктів;
   – зміна відомостей про проєкт;
   – додавання учасників до проєкту;
3. Блок керування дорученнями. Забезпечує:
   – створення доручень;

– призначення виконавців;
– встановлення крайніх термінів;
– зміна стану виконання;

4. Блок комунікацій. Забезпечує:
– додавання коментарів до доручень;
– обмін файлами;
– сповіщення між учасниками.

Інформаційні зв’язки між частинами програми забезпечуються через передачу даних між клієнтською ланкою, сервером та сховищем даних.

Для проєктування програми застосовано об’єктно-орієнтований метод, який дозволяє представити програму як взаємодію об’єктів предметної сфери.

Основними об’єктами програми є:

– user
– project;
– task
– comments;
– file;

Кожен об’єкт має набір ознак та методів, що окреслюють його функціонування у програмі.

Для моделювання будови програми можуть застосовуватися такі схеми:

- схема класів - для опису будови об’єктів програми;
- схема потоків даних (DFD) - для відображення потоків інформації;
- ER-схема - для проєктування сховища даних.

Потоки даних у системі організовані за принципом клієнт-серверної взаємодії. Користувач формує запит через веб-інтерфейс, після чого запит надходить до серверної частини для обробки. Сервер виконує перевірку коректності даних, реалізує бізнес-логіку та взаємодіє з базою даних. Після завершення обробки результат повертається користувачеві у вигляді оновленого інтерфейсу або повідомлення про виконання операції.

2.2 Алгоритмічне забезпечення

Програмне забезпечення комплексу визначає логіку роботи веб-застосунку та описує послідовність виконання дій під час взаємодії користувача з системою. Головним алгоритмом комплексу є алгоритм управління задачами.

Узагальнений алгоритм роботи системи (рисунок 2.1):

1. Користувач відкриває веб-застосунок.
2. Система звіряє авторизацію користувача.
3. Якщо користувач не авторизований - виконується процедура входу чи реєстрації.
4. Після авторизації відкривається головна сторінка комплексу.
5. Користувач може здійснити одну з наступних дій:
   - створити новий проєкт;
   - переглянути перелік проєктів;
   - створити задачу;
   - змінити наявну задачу;
   - додати примітку до задачі;
   - змінити стан виконання.
6. Після виконання операції відомості передаються на сервер.
7. Сервер обробляє запит та фіксує відомості у базі даних.
8. Користувач одержує свіжу інформацію на екрані.

Алгоритм створення задачі (рисунок 2.2):

1. Користувач відкриває сторінку проєкту.
2. Натискає кнопку “Створити задачу”.
3. Вводить відомості:
   - назву задачі;
   - деталізацію;
   - відповідального;
   - кінцевий термін.
4. Система звіряє правильність введених відомостей.

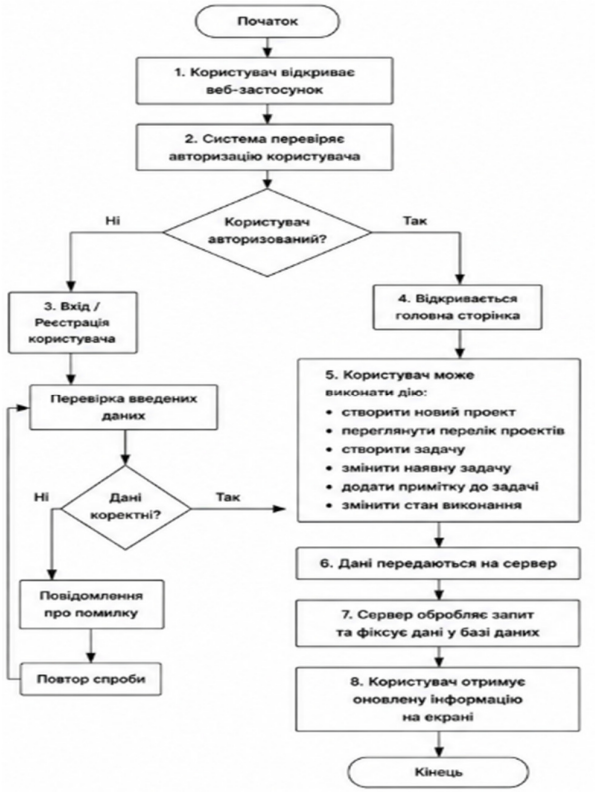

Рисунок 2.1 – Загальний алгоритм роботи системи

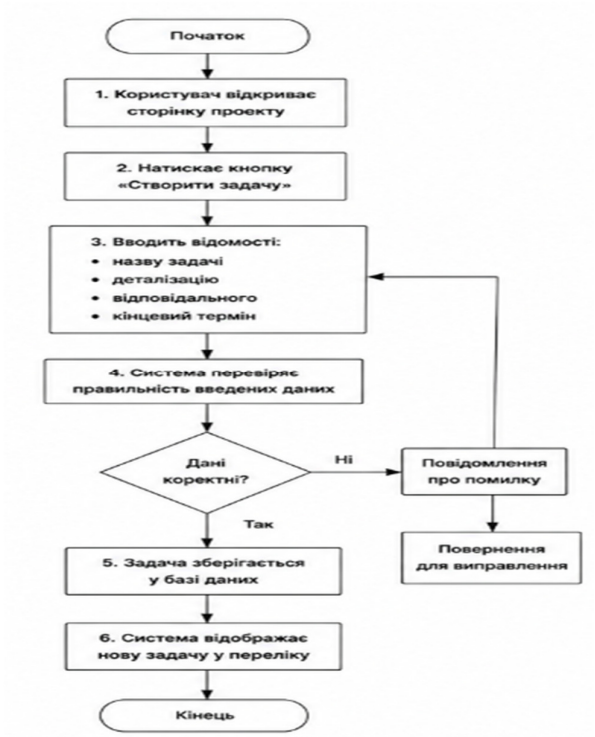

Рисунок 2.2 – Алгоритм створення задачі

5. Якщо відомості правильні - задача зберігається у базі даних.
6. Система показує нову задачу у переліку.

Алгоритм зміни стану задачі:

1. Користувач відкриває перелік завдань.
2. Обирає потрібну задачу.
3. Змінює стан (наприклад: нове - у роботі - завершено).
4. Система реєструє зміну стану.
5. Відомості оновлюються у базі даних.
6. Оновлена інформація відображається у системі.

Алгоритми мусять брати до уваги можливі збої, наприклад:

- введення неправильних відомостей;
- брак доступу до проєкту;
- збої зв'язку із сервером.

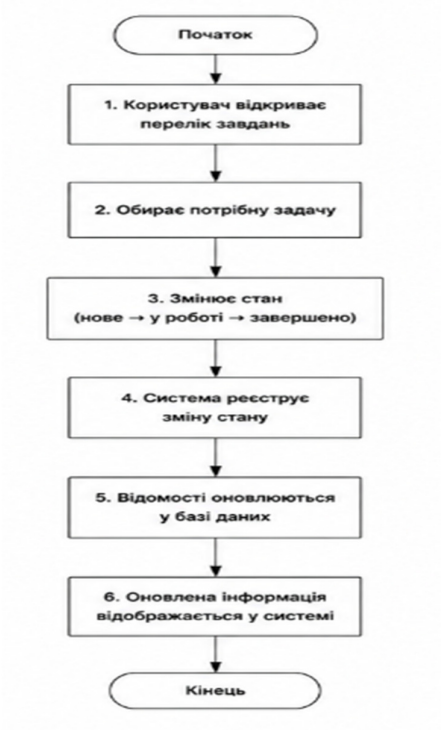

Рисунок 2.3 – Алгоритм зміни стану задачі

2.3 Інформаційне забезпечення

Інформаційне забезпечення системи містить організацію зберігання, обробки та передачі відомостей, потрібних для функціонування веб-застосунку.

2.3.1 Опис вхідних та вихідних відомостей

Вхідна інформація є набором даних, які користувач надає системі для подальшого опрацювання та застосування. Ці відомості становлять основу роботи платформи і гарантують реалізацію її ключових функцій. До основних вхідних даних відносяться реєстраційні відомості про користувача, що містять особисту інформацію, контактні дані та інші обов'язкові атрибути для створення профілю.

Крім того, ключове значення має інформація для входу а саме логін та пароль, які дозволяють отримати доступ до системи і забезпечують безпеку особистих даних. Суттєву частину введених відомостей складають дані про завдання: їхнє найменування, опис, ступінь важливості, строки реалізації та інші параметри, необхідні для планування діяльності.

Також до вхідної інформації зараховуються коментарі до задач, що слугують для взаємодії між учасниками проєкту, уточнення подробиць та обговорення робочих процедур. Додатково користувачі надають відомості про проєкти, включно з їхньою назвою, характеристикою, організацією та переліком учасників, що дає змогу системі ефективно структурувати робочий простір.

Вихідні дані - це результат опрацювання введених відомостей, який представляється користувачеві у зрозумілій та легкодоступній формі. Вони демонструють актуальний стан системи та сприяють прийняттю управлінських рішень. До ключових вихідних даних входить перелік завдань, що охоплює всі поточні та виконані задачі в рамках проєкту.

Важливим компонентом є статус виконання задач, який дозволяє моніторити етапи їхньої реалізації та оцінювати поступ у роботі. Крім того, система надає інформацію про виконавців, що дає змогу встановити відповідальних осіб та контролювати завантаження.

До вихідних даних також належать статистичні відомості щодо виконання завдань, котрі можуть включати обсяг завершених задач, відповідність строкам та загальну продуктивність. Окрім цього, система генерує системні повідомлення, які інформують користувачів про ключові події, оновлення у завданнях або наближення кінцевих термінів.

Отже, взаємозв'язок між вхідною та вихідною інформацією забезпечує повноцінну роботу системи, формуючи уніфіковане інформаційне середовище для керування проєктами та узгодження дій користувачів.

2.3.2 Концептуальне інфологічне проєктування

На етапі інфологічного проєктування формується концептуальна модель даних, що описує головні об’єкти предметної області та зв’язки між ними.

Головними компонентами системи для керування проєктами є об'єкти, що представляють основні аспекти предметної області та дозволяють зберігати й обробляти інформацію. Кожна така складова характеризується набором властивостей які сприяють взаємодії між різними частинами системи.

Однією з перших і найважливіших складових є користувач (User). Ця складова відповідає за зберігання даних про людей, що використовують систему. Серед її ключових характеристик - унікальний номер (id), ім'я (name), електронна пошта (email) та пароль (password) для підтвердження особи. Саме ці особи створюють проєкти, виконують поставлені завдання та спілкуються один з одним всередині системи.

Далі йде проєкт (Project), який слугує для об'єднання взаємопов'язаних завдань. Він дозволяє організувати роботу відповідно до конкретних цілей чи напрямків. До основних властивостей належать: унікальний номер (id), назва (name), детальний опис (description), а також дата створення (created_at), яка допомагає відслідковувати історію робіт.

Завдання (Task) є ключовою частиною системи, оскільки воно представляє собою окрему одиницю роботи. Кожне завдання має унікальний номер (id), назву (title), опис (description), поточний стан виконання (status) та дату завершення

(deadline). Завдання можуть бути пов'язані як з проєктами, так і з користувачами, що сприяє впорядкованому управлінню робочими процесами.

Для взаємодії між учасниками проєкту передбачена складова "коментар" (Comment). Вона дає можливість додавати текстові записи до завдань, уточнювати подробиці або обговорювати прогрес. Головними властивостями є унікальний номер (id), власне текст (text) та дата створення (created_at).

Окремо виділяється файл або вкладення (Attachment), що слугує для зберігання додаткових матеріалів, які стосуються завдань. Це можуть бути різноманітні документи, зображення чи інші дані, потрібні для виконання роботи. Його властивості включають унікальний номер (id), назву файлу (filename), місце зберігання (filepath), а також ідентифікатор завдання (task_id), який вказує на зв'язок між файлом та конкретним завданням.

Отже, описані складові формують фундамент структури системи, гарантуючи збереження інформації, її логічне впорядкування та взаємозв'язок між основними компонентами процесу керування проєктами.

Зв’язки між сутностями:

- користувач може бути учасником багатьох проєктів;
- проєкт охоплює багато завдань;
- завдання може мати багато коментарів.

2.3.3 Розробка глобальної даталогічної моделі даних

Логічна модель даних описує архітектуру сховища інформації системи управління завданнями та визначає логічні взаємозв'язки між її ключовими об'єктами. Ця модель розробляється на основі інформаційної моделі та ілюструє спосіб збереження відомостей у реляційній базі даних.

Головна мета створення логічної моделі даних полягає у:

– визначенні структури таблиць бази даних;
– встановленні зв'язків між таблицями;
– забезпеченні цілісності та узгодженості даних;
– оптимізації процесу зберігання інформації.

У процесі проєктування загальної логічної моделі даних були використані ER-діаграма у нотації IDEF1X (рисунок 2.4), а також UML-діаграма класів, що надає більш наочне представлення архітектури системи (рисунок 2.5).

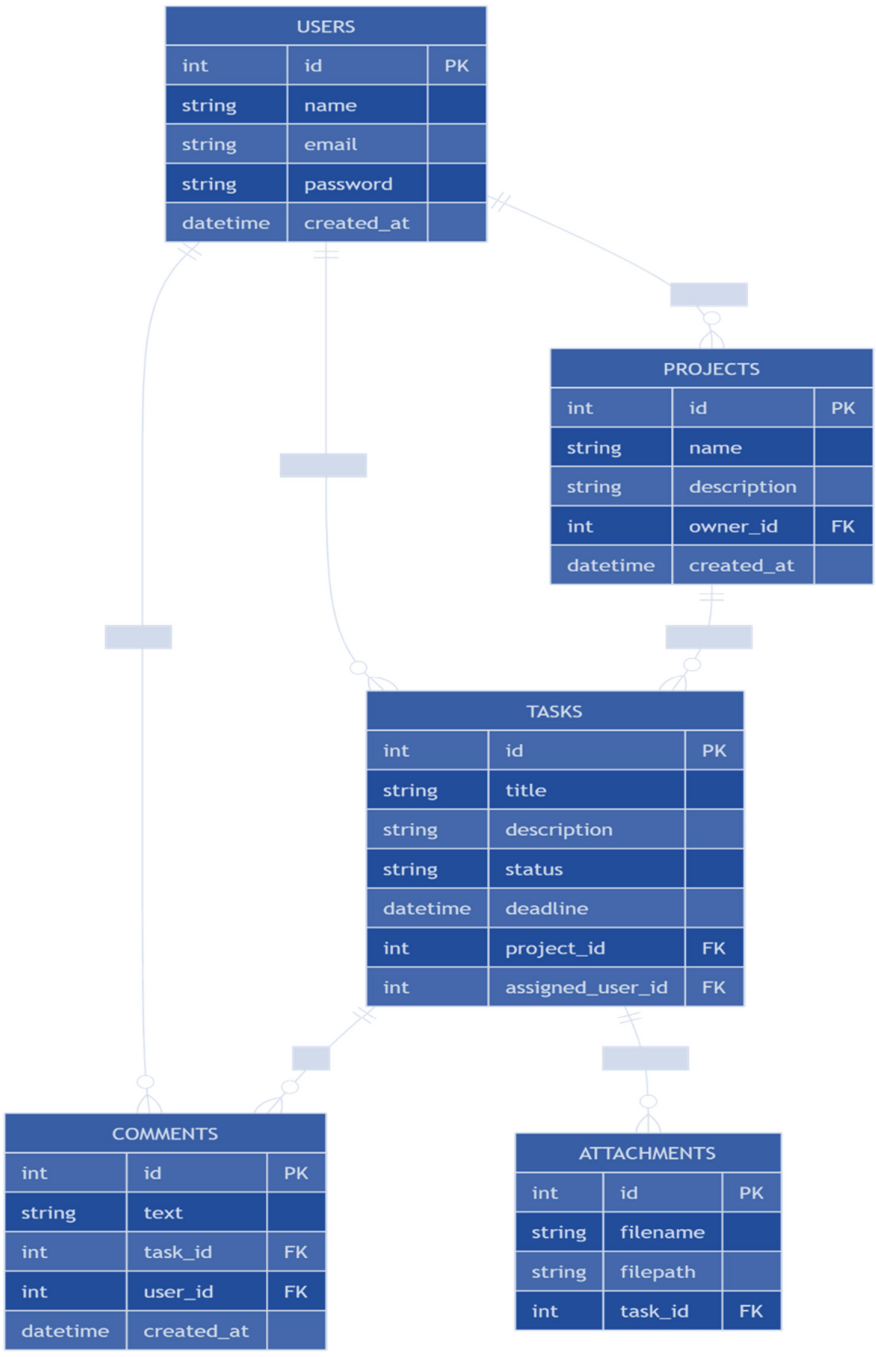

Рисунок 2.4 – □□D – діаграма

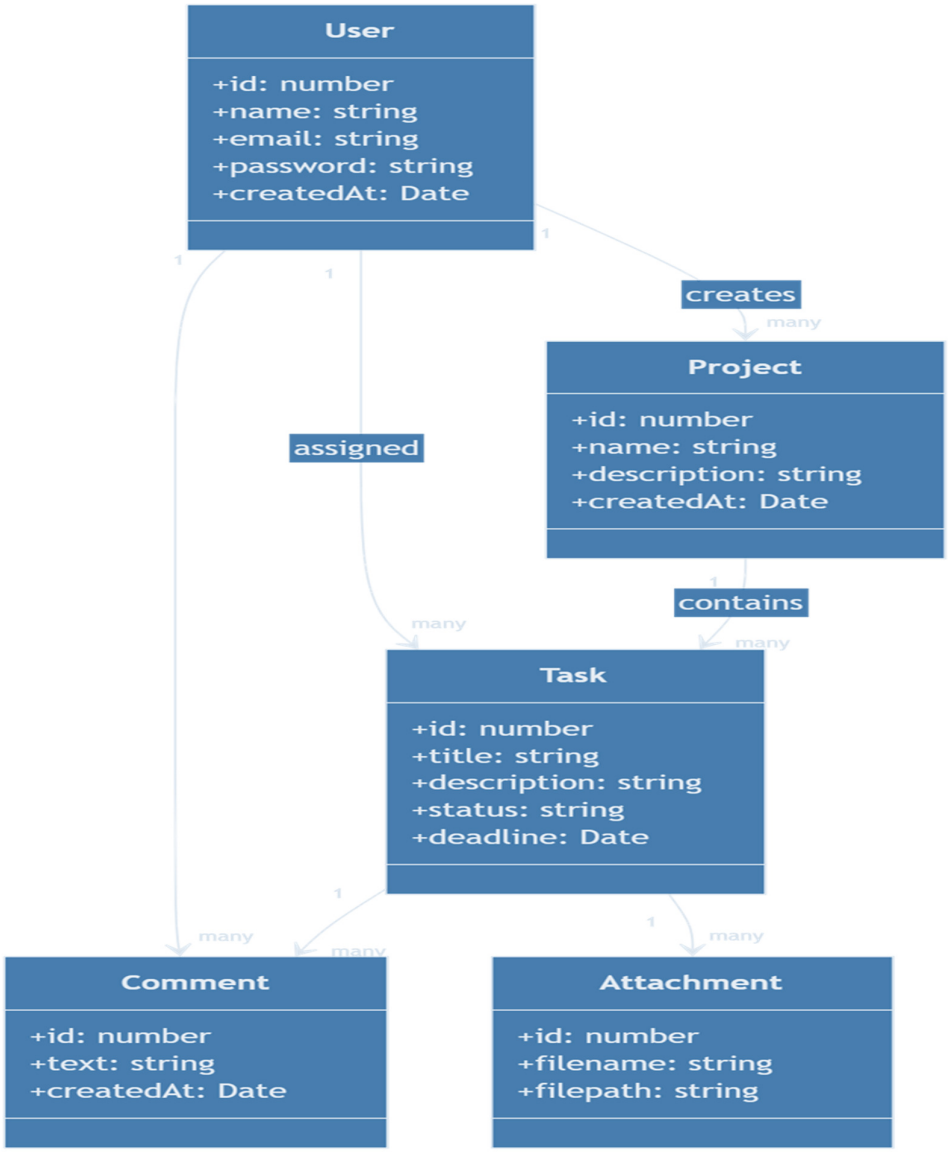

Рисунок 2.5 – U L - діаграма класів

Основними сутностями системи управління завданнями є:

– Користувач (User)
– Проєкт (Project)
– Завдання (Task)
– Коментар (Comment)
– Файл (Attachment)

Взаємозв'язки між сутностями:

– один користувач може створювати багато проєктів;
– один проєкт може містити безліч завдань;

– одне завдання може мати численні коментарі;

– одне завдання може мати багато прикріплених файлів;

– один користувач може бути призначений виконавцем для багатьох завдань.

Для більш докладного опису структури системи застосовується UML-схема класів (рисунок 2.5). Вона демонструє класи, їхні атрибути та взаємозв'язки.

Головні класи системи:

- Us□r;
- □r□j□□t;
- □□sk;
- □□mm□nt;
- Attachment.

2.3.4 Проєктування фізичної моделі даних

Фізична модель даних окреслює спосіб втілення бази даних у певній системі керування базами даних (СУБД). На відміну від логічної та даталогічної моделей, фізична модель враховує технічні особливості зберігання відомостей, структуру таблиць, типи даних, індекси, обмеження та засоби оптимізації доступу до даних.

Для реалізації бази даних системи керування завданнями було обрано реляційну СУБД PostgreSQL, яка гарантує високу продуктивність, надійність збереження даних та підтримку складних запитів. PostgreSQL широко застосовується у сучасних веб-застосунках та підтримує розширені механізми забезпечення цілісності даних.[17]

Фізична модель бази даних формується на підставі глобальної даталогічної моделі та містить конкретні таблиці, їх поля, типи даних та обмеження.

Фізична модель бази даних системи має такі ознаки:

- використовується реляційна структура зберігання відомостей;
- кожна таблиця має первинний ключ (Primary Key);

- зв’язки між таблицями реалізуються за допомогою зовнішніх ключів (Foreign Key);
- застосовуються індекси для оптимізації швидкості пошуку даних;
- використовується нормалізація таблиць, що дозволяє уникнути дублювання відомостей.

Типи даних:

- INTEGER - для ідентифікаторів записів;
- VARCHAR - для текстових полів невеликої довжини;
- TEXT - для зберігання великих текстових даних;
- TIMESTAMP - для збереження дати та часу;
- BOOLEAN - для логічних значень.

Використання відповідних типів даних дозволяє оптимізувати застосування пам’яті та підвищити швидкість обробки запитів.[17]

Безпека інформації є одним із ключових аспектів функціонування веб-застосунку. Для захисту персональних даних користувачів та запобігання несанкціонованому доступу реалізовано комплекс організаційних та програмних заходів.

Основними механізмами захисту є:

– автентифікація користувачів за логіном та паролем;

– зберігання паролів у вигляді хешованих значень;

– перевірка прав доступу до проєктів та завдань;

– використання захищених HTTP-з'єднань;

– валідація вхідних даних на серверній стороні;

– захист від несанкціонованого доступу до API.

Застосування зазначених механізмів дозволяє забезпечити конфіденційність, цілісність та доступність інформації в системі.

3. ПРОГРАМНО-ТЕХНОЛОГІЧНЕ ЗАБЕЗПЕЧЕННЯ ВЕБ-ЗАСТОСУНКУ

3.1 Реалізація програмного забезпечення

Програмний комплекс реалізований на основі клієнт-серверної архітектури. Його основними складовими є клієнтський модуль, серверний модуль та сховище даних.

Структура системи представлена у вигляді трирівневої моделі:

– рівень інтерфейсу користувача (Frontend);
– рівень обробки бізнес-логіки (Backend);
– рівень збереження інформації (Database).

Клієнтський модуль відповідає за комунікацію з користувачем, формування та надсилання запитів до серверного модуля, а також за відображення отриманих результатів. Він реалізує користувацький інтерфейс та забезпечує легку навігацію функціоналом системи.

Серверний модуль розроблений із застосуванням фреймворку NestJS і виконує обробку запитів, імплементацію бізнес-процесів та взаємодію з базою даних. Ключовими структурними елементами серверного модуля є контролери, сервіси та репозиторії.

Контролери приймають HTTP-запити від клієнтської частини та передають їх для подальшої обробки відповідним сервісам. Сервіси містять основні бізнес-правила застосунку та здійснюють операції з даними. Репозиторії забезпечують доступ до сховища даних за допомогою технології ORM.[17]

База даних використовується для зберігання інформації про користувачів, проєкти, завдання, коментарі та вкладені файли. Взаємодія з базою даних здійснюється через ORM, що забезпечує абстрагований доступ до інформації.

Для реалізації веб-застосунку було обрано сучасний стек технологій, що забезпечує високу продуктивність, масштабованість та зручність подальшого супроводу програмного забезпечення.

Серверна частина системи реалізована із використанням фреймворку NestJS. Вибір даного фреймворку обумовлений його модульною архітектурою, підтримкою

мови TypeScript та наявністю вбудованих механізмів для реалізації REST API. Порівняно з Express.js, який є одним із найпоширеніших фреймворків для розробки серверних застосунків на платформі Node.js, NestJS забезпечує більш структурований підхід до організації коду. Використання контролерів, сервісів та модулів дозволяє розділити бізнес-логіку та спростити подальший розвиток програмного продукту. Крім того, NestJS містить вбудовану підтримку механізмів впровадження залежностей (Dependency Injection), що позитивно впливає на якість архітектури програмної системи.

Для зберігання даних було обрано систему керування базами даних PostgreSQL. Серед основних переваг PostgreSQL слід відзначити високу надійність, підтримку складних SQL-запитів, розширені механізми забезпечення цілісності даних та високу продуктивність при роботі з великими обсягами інформації. Порівняно з MySQL, PostgreSQL забезпечує більш розвинуту підтримку транзакцій, зовнішніх ключів та механізмів контролю цілісності даних. Також PostgreSQL підтримує широкий набір сучасних можливостей, включаючи розширені типи даних, повнотекстовий пошук та складні механізми індексації, що є важливим для розробки інформаційних систем корпоративного рівня.

Для взаємодії серверної частини з базою даних використовується ORM-технологія. ORM (Object Relational Mapping) дозволяє працювати з даними у вигляді об'єктів мови програмування без необхідності створення великої кількості SQL-запитів вручну. Серед сучасних ORM-рішень найбільш поширеними є TypeORM та Prisma. У даному проєкті було використано TypeORM завдяки його тісній інтеграції з NestJS та підтримці об'єктно-орієнтованого підходу до проєктування баз даних. На відміну від Prisma, яка орієнтована на генерацію типізованого клієнта доступу до даних, TypeORM забезпечує більш гнучке налаштування зв'язків між сутностями та спрощує реалізацію складних структур даних.

Таким чином, використання NestJS, PostgreSQL та TypeORM дозволило побудувати сучасну, масштабовану та надійну програмну систему, що відповідає вимогам до веб-застосунків для керування завданнями в малих робочих групах та

забезпечує можливість подальшого розширення функціональності без суттєвих змін архітектури.

Потік даних між компонентами системи відбувається за наступною послідовністю представленою на рисунку 3.1. Такий підхід гарантує чітке розділення відповідальності та сприяє покращенню масштабованості рішення.

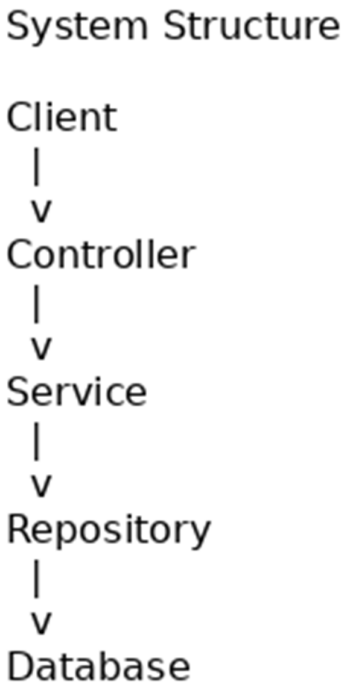

Рисунок 3.1 – Структурна схема

3.1.2 U□ L-Діаграма класів, що реалізують основну бізнес-логіку програмної системи

Клас ‘□□sk’ є фундаментальним компонентом системи, призначеним для моделювання та представлення окремих завдань. Він об'єднує основні властивості, такі як унікальний ідентифікатор, найменування, детальний опис, поточний стан (статус), а також містить посилання на пов'язаного користувача та відповідну категорію. Призначення цього класу полягає у зберіганні та керуванні інформацією про завдання, створені користувачами в системі.

Клас ‘User’ моделює обліковий запис користувача в системі. Він включає ключові атрибути, необхідні для впізнавання та підтвердження особистості користувача, зокрема адресу електронної пошти та пароль. Користувач уповноважений створювати та керувати власними завданнями, що встановлює безпосередній зв'язок між класами `User` та `Task`.

Клас ‘TaskService’ інкапсулює ділову логіку, що стосується управління завданнями. Він надає набір методів для виконання операцій зі створення, вилучення (отримання), оновлення та видалення завдань. Цей клас виконує попередню обробку даних перед їх збереженням у сховищі та здійснює взаємодію з відповідними репозиторіями.

Клас ‘UserService’ несе відповідальність за функціональну логіку, пов'язану з управлінням користувацькими обліковими записами. Він забезпечує реалізацію функцій реєстрації нових користувачів, їх автентифікації (авторизації) та отримання відповідних даних про користувача.

Клас ‘TaskController’ відповідає за обробку вхідних HTTP-запитів, призначених для маніпуляцій із завданнями. Він слугує інтерфейсом, що координує взаємодію між клієнтською частиною програми та відповідними сервісами.

Клас ‘UserController’ керує обробкою мережевих запитів, які стосуються користувацьких операцій, зокрема запитів на реєстрацію нових облікових записів та їх автентифікації.

Приклад наведеної діаграми класів зображена на рисунку 3.2.

3.1.3 UML-діаграма станів інтерфейсу користувача

Діаграма станів UML призначена для візуалізації різноманітних станів, у яких може перебувати користувацький інтерфейс, а також для ілюстрації можливих переходів між цими станами.[15]

Ключовими станами системи є:

– Початковий стан (Start): фаза ініціалізації або первинного налаштування інтерфейсу.

– Стан завантаження (L□□d□ng): період, коли система очікує отримання необхідних даних з віддаленого джерела.

– Стан готовності (L□□d□d): фаза, коли дані відображені, і користувач може повноцінно взаємодіяти з функціоналом системи.

– Стан несправності (□rr□r): виникає у випадку збою або некоректного виконання будь-якого запиту.

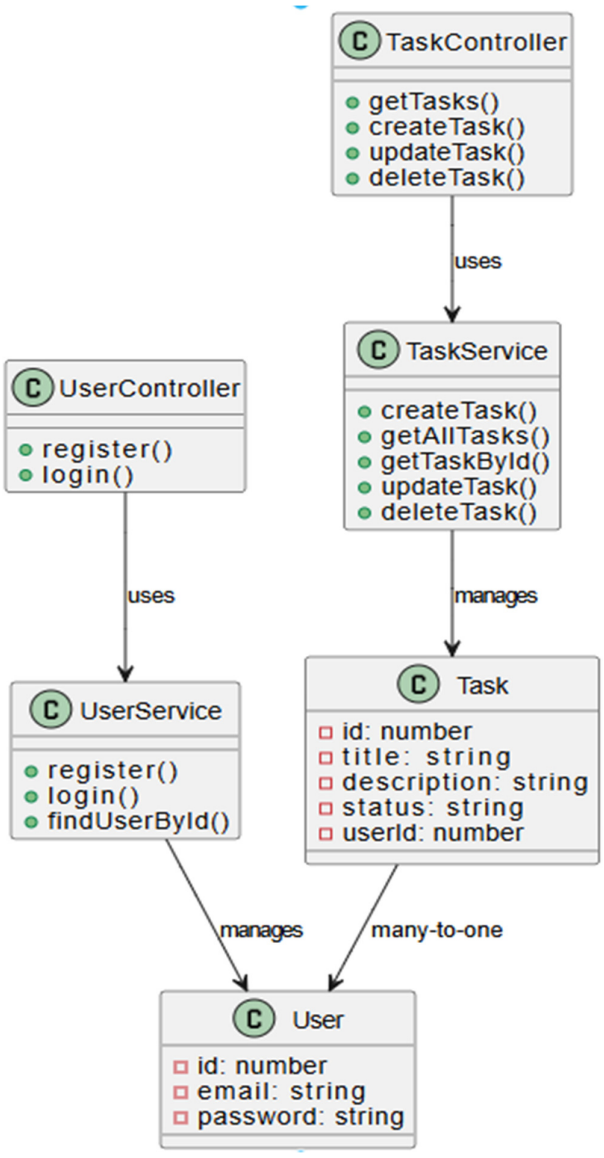

Рисунок 3.2 – Діаграма класів

Зміна станів відбувається на підставі результатів обробки запитів. Наприклад, після ініціалізації інтерфейсу система послідовно переходить до стану завантаження (рисунок 3.3). Залежно від успішності цього процесу, вона далі переміщується або до стану готовності (у разі успішного завершення), або до стану несправності (при виникненні помилки).

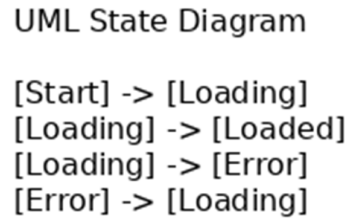

Рисунок 3.3 - Стану завантаження

Ця діаграма дозволяє чітко структурувати поведінку користувацького інтерфейсу та значно поглиблює розуміння логіки його взаємодії з користувачем.

3.1.4 Обґрунтування вибору програмних засобів реалізації

Для розробки серверної частини системи було застосовано фреймворк N□stJS, який функціонує на базі платформи N□d□js та підтримує використання мови програмування □□□S□r□t.[18]

Вибір даного фреймворку був зумовлений такими основними перевагами:

– модульна архітектура, що забезпечує високу організованість та структурованість коду;

– інтегрована підтримка □□□S□r□t, яка підвищує надійність та стабільність програмного забезпечення;

– наявність механізму впровадження залежностей, що спрощує тестування та подальшу підтримку коду;

– можливість створення масштабованих серверних додатків.

Для організації взаємодії з базою даних було використано підхід □□□ (□bj□t-□□l□t□n□ □ □□□ng), який дозволяє працювати з даними на рівні об'єктів та оптимізує процес розробки.[17]

В якості архітектурного шаблону для комунікації між клієнтською та серверною частинами було обрано REST (Representational State Transfer), що гарантує простоту інтеграції та універсальність рішення.

3.2 Інтерфейс користувача

Для гарантування безпечного доступу до можливостей системи застосовується процедура підтвердження особи користувачів (рисунок 3.4).

Після коректного введення ідентифікаційних даних:

– система здійснює перевірку валідності наданих облікових відомостей (наприклад, електронної пошти та пароля);

– користувач отримує доступ до захищених компонентів системи;

– генерується спеціальний маркер доступу, що використовується для подальшої взаємодії з програмним інтерфейсом (API).

Цей механізм дозволів забезпечує розмежування доступу до функціоналу системи, обмежуючи його для користувачів, які не мають відповідних прав.

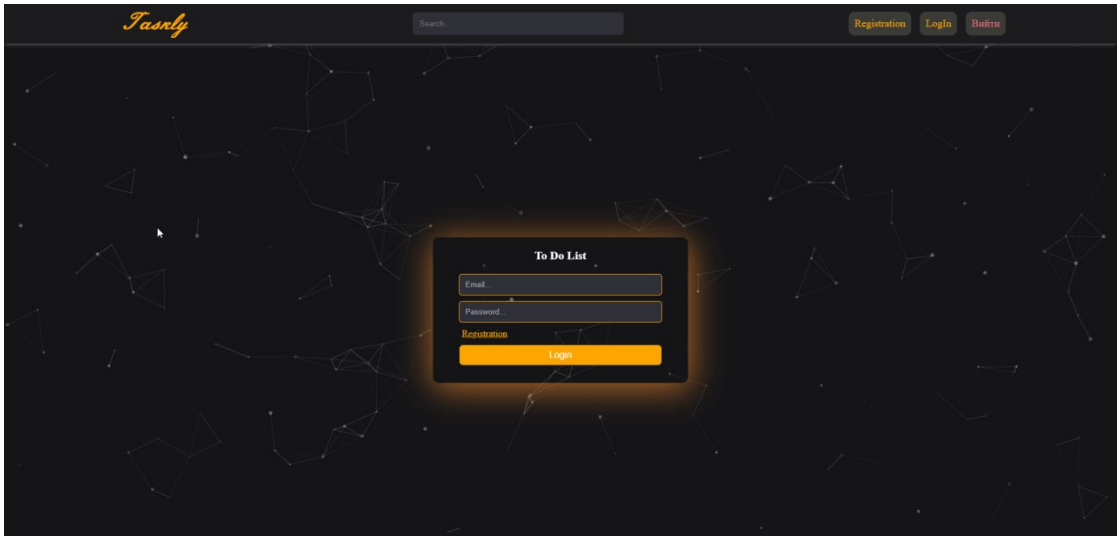

Рисунок 3.4 – Підтвердження особи користувачів

Платформа надає функціонал для реєстрації нових користувачів. У процесі реєстрації користувачі надають обов'язкові персональні відомості, серед яких електронна адреса, пароль, ім’я та прізвище (рисунок 3.5).

Процес реєстрації передбачає виконання таких дій:

– перевірка валідності наданих даних;
– контроль на унікальність електронної адреси;
– занесення інформації про нового користувача до бази даних.

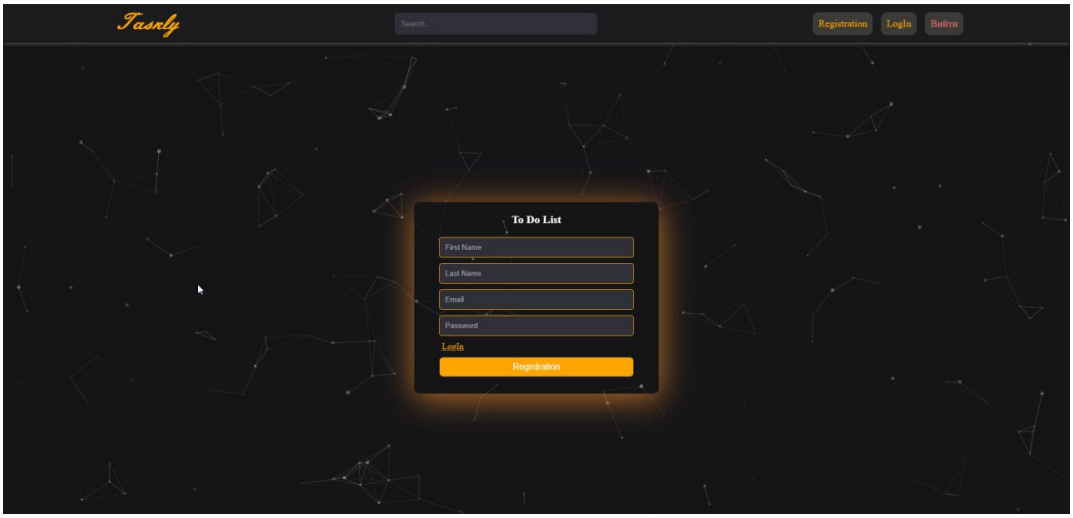

Рисунок 3.5 - Реєстрація нових користувачів

Авторизованому користувачеві надається можливість створювати нові завдання. У процесі формування завдання передбачено вказання таких параметрів:

– детальний опис;
– поточний статус виконання.

Після відправлення відповідного запиту дані надсилаються до серверного компонента, де здійснюється їхня подальша обробка та зберігання у сховищі даних.

Рисунок 3.6 демонструє головне вікно програмної системи після успішної аутентифікації користувача.

Ключовими складовими інтерфейсу є:

– верхня навігаційна панель;
– блок пошуку;

– перелік користувачів організації;

– область керування проєктами;

– кнопки для реєстрації та входу.

У лівій частині інтерфейсу відображаються відомості про поточного користувача та компанію, до якої він належить. Нижче розміщено список членів організації, що дозволяє швидко переглядати учасників команди.

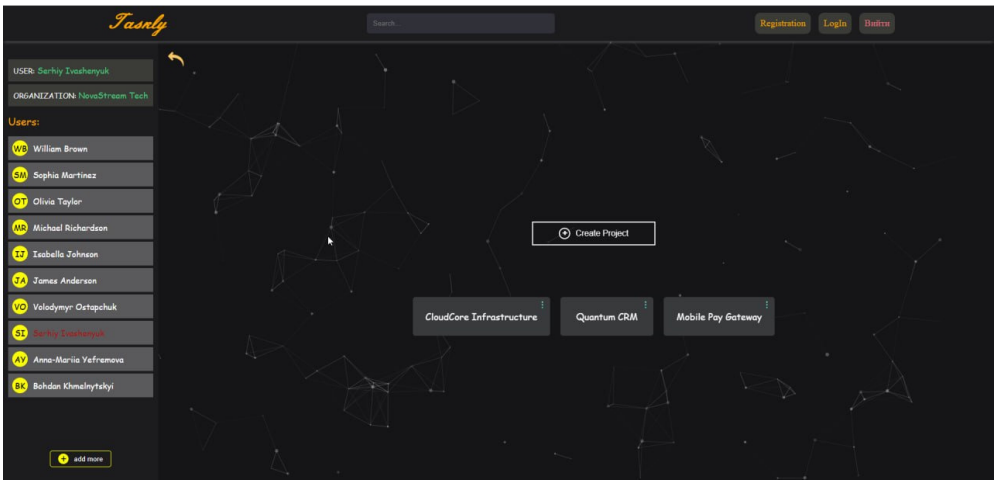

Рисунок 3.6 – Головне вікно програмної системи

Центральна частина інтерфейсу містить функціональність для управління проєктами. Користувачі мають можливість створювати нові проєкти (рисунок 3.7) та переглядати вже існуючі.

Кнопка “Create Project” призначена для створення нового проєкту, а кнопка “add more” використовується для додавання нових членів до організації.

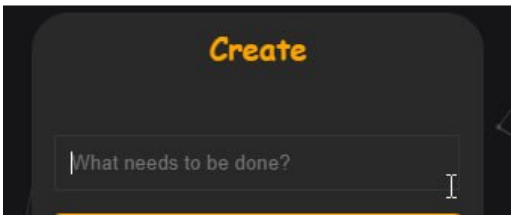

Рисунок 3.7 – Створення нового проєкту

Візуальне оформлення інтерфейсу виконано у темній колірній палітрі, що покращує зорове сприйняття інформації та забезпечує зручність під час тривалої роботи з системою.

На рисунку 3.8 представлено інтерфейс системи керування завданнями у межах окремого проєкту. Цей інтерфейс виконано у вигляді Kanban-дошки, де завдання розподіляються згідно з їхнім статусом виконання.

До основних категорій належать:

– Create;

– □n□r□gr□ss;

– D□n□;

– Verified.

Кожна категорія відображає відповідну стадію життєвого циклу завдання. Для кожного завдання відображається така інформація:

– найменування завдання;

– призначений виконавець;

– додаткові елементи керування.

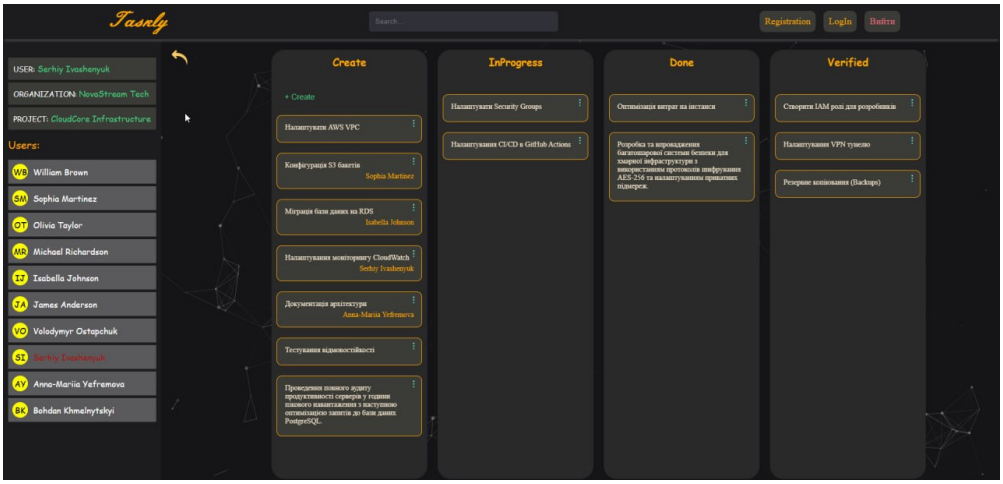

Рисунок 3.8 – Інтерфейс системи керування завданнями

Користувач має можливість:

– створювати нові завдання;

– переміщувати завдання між різними статусами;

– відстежувати процес виконання;

– переглядати призначених осіб.

Застосування методології Kanban забезпечує візуалізацію поточного стану виконання завдань та покращує координацію командної роботи. У лівій частині інтерфейсу додатково розміщено перелік учасників проєкту, що сприяє швидкій взаємодії з командою.

На рисунку 3.9 зображено модальне вікно для призначення виконавця завдання. Цей елемент інтерфейсу призначений для вибору користувача, який буде відповідальним за виконання певного завдання.

У вікні представлений перелік усіх співробітників організації, доступних для призначення на роль виконавця. Кожен учасник має індивідуальний елемент для вибору. Функціональність цього інструменту охоплює:

– призначення основного виконавця;

– зміну відповідальної особи за завданням;

– організацію розподілу завдань між членами команди.

Застосування модального вікна забезпечує швидкий доступ до функції призначення виконавця, усуваючи необхідність переходу між різними розділами системи.

Рисунок 3.9 - Модальне вікно для призначення виконавця завдання

3.3 Тестування розробленої програми

Під час створення програмної системи для керування завданнями було здійснено всебічне тестування її ключових можливостей. Метою цієї перевірки було підтвердження правильності функціонування системи, надійності взаємодії її складових, а також відповідності реалізованих функцій початковим вимогам.

Випробування охоплювали як серверну, так і клієнтську складові програмного продукту, а також способи їхньої взаємодії.

Основними завданнями тестування були:

– підтвердження працездатності окремих функціональних блоків;

– виявлення будь-яких недоліків або помилок у функціонуванні системи;

– перевірка надійності та стабільності зв'язку між клієнтською та серверною частинами;

– контроль за правильністю обробки веб-запитів (HTTP-запитів);

– випробування систем авторизації користувачів та керування завданнями.

У процесі розроблення програмного забезпечення було використано такі типи тестування:

– функціональні випробування (рисунки 3.10 – 3.11);

– інтеграційні випробування;

– перевірка користувацького інтерфейсу;

– тестування програмного інтерфейсу (API);

– тестування перевірки (валідації) даних.

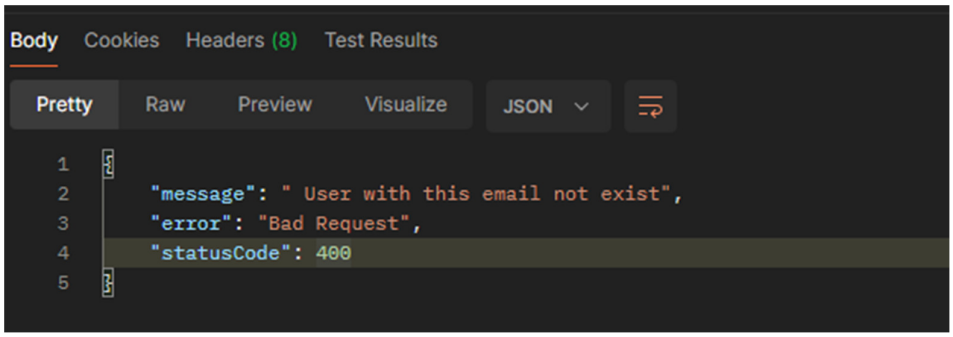

Рисунок 3.10 – Відповідь сервера при введенні невірного e-mail користувача

Функціональні випробування були спрямовані на перевірку реалізації та коректної роботи основних можливостей системи.

Було протестовано такі елементи:

– реєстрація нових користувачів (рисунок 3.12);
– вхід до системи (авторизація) (рисунки 3.13 – 3.15);
– створення нових завдань (рисунок 3.16);
– редагування існуючих завдань (рисунок 3.17);
– видалення завдань;
– призначення відповідальних осіб;
– перегляд переліку завдань;
– зміна статусів завдань.

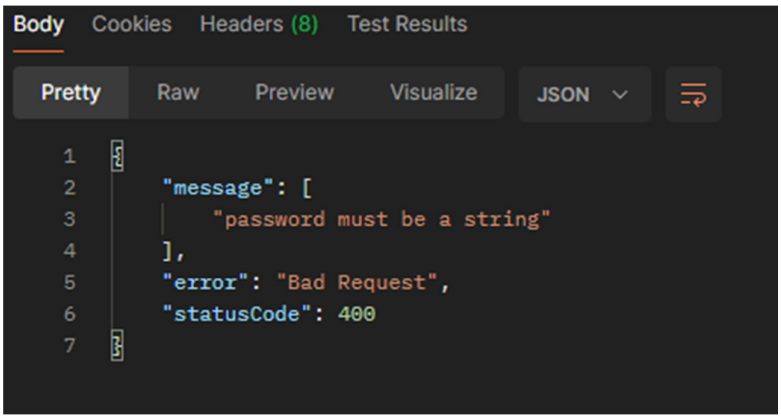

Рисунок 3.11 – Відповідь сервера при введенні невірного паролю користувача

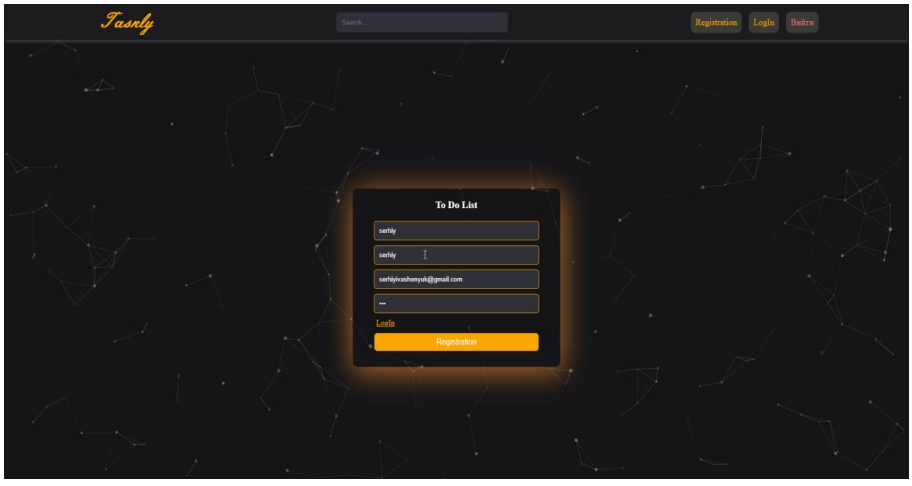

Рисунок 3.12 – Приклад реєстрації нових користувачів

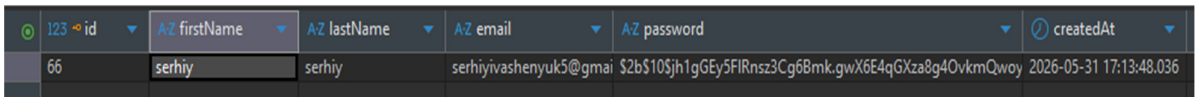

Рисунок 3.13 – Відображення користувача в базі

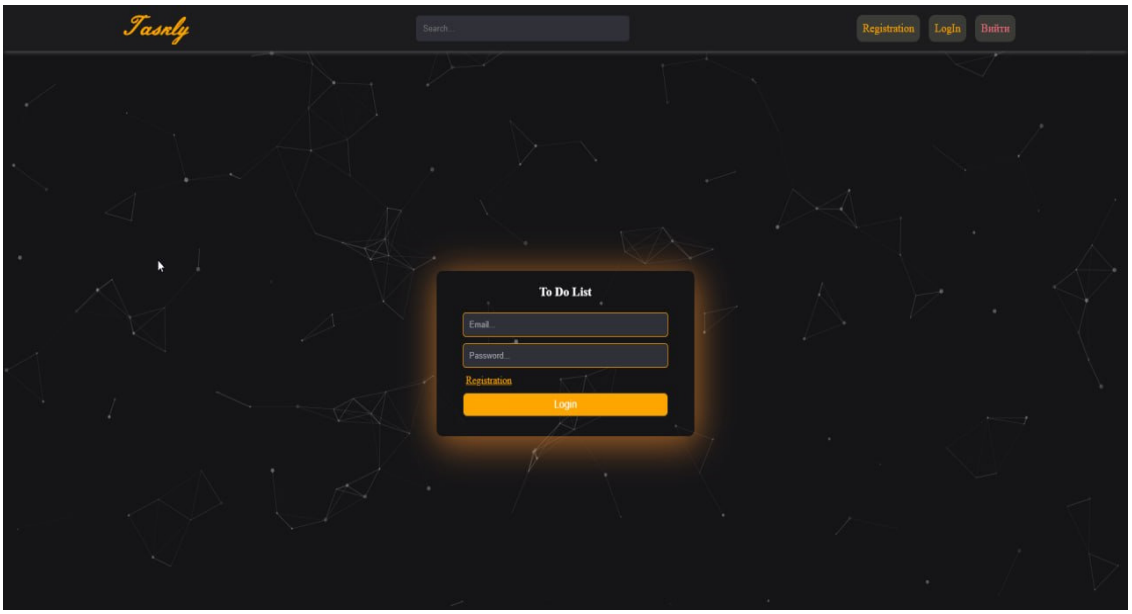

Рисунок 3.14 – Авторизація користувача

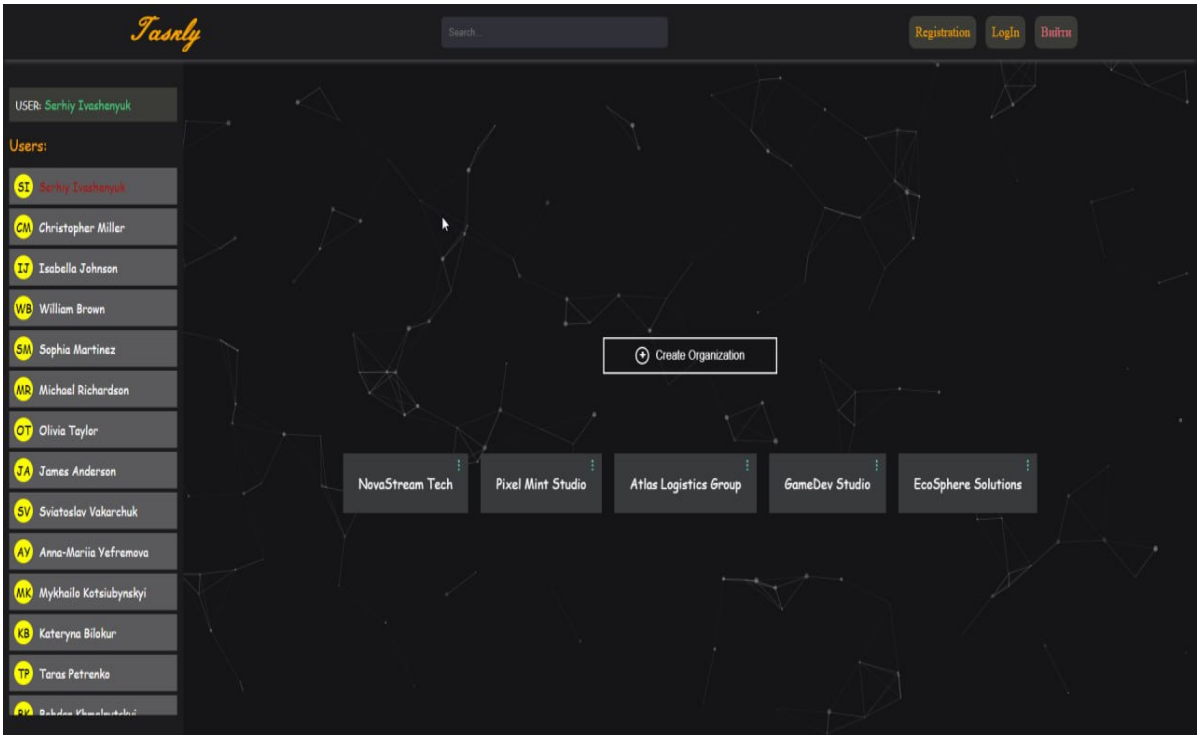

Рисунок 3.15 – Вікно користувача після авторизації

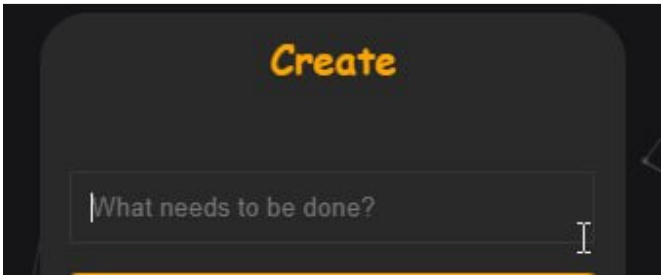

Рисунок 3.16 – Створення нових завдань

Рисунок 3.17 – Редагування існуючих завдань

За результатами проведеного тестування було встановлено, що програмний продукт:

– ефективно виконує свої ключові функції та демонструє стабільність роботи під час типових експлуатаційних режимів;

– забезпечує коректну взаємодію між її клієнтськими та серверними компонентами;

– належним чином здійснює управління завданнями.

При цьому критичних дефектів, які б унеможливлювали експлуатацію системи, виявлено не було.

Під час тестування було перевірено основні сценарії роботи системи: реєстрацію користувачів, авторизацію, створення проєктів, створення завдань, зміну статусів (у додатку б знаходиться код який реалізує функціонал який зображений на Рисунку 3.7, 3.9, 3.16, 3.17), додавання коментарів та перегляд списків завдань. Результати тестування показали коректне виконання всіх основних функцій системи. Помилок, які могли б призвести до втрати даних або порушення логіки роботи застосунку, виявлено не було. Час відгуку системи при виконанні типових операцій не перевищував декількох секунд, що забезпечує комфортну роботу користувачів.

ВИСНОВКИ

1. Було вивчено процес управління завданнями в невеликих робочих колективах, встановлено ключові проблеми, такі як необхідність координації, моніторингу виконання та ефективної взаємодії між учасниками.

2. Проведено аналіз існуючих рішень (Trello, Asana та інші), виявлено їхні переваги (наприклад, зручність, візуалізація) та недоліки (обмежений функціонал або складність), що обґрунтувало доцільність розробки власного програмного продукту.

3. Сформульовано функціональні та нефункціональні вимоги до системи, що охоплюють керування користувачами, проєктами, завданнями, коментарями та вкладеннями, а також критерії безпеки, продуктивності та зручності експлуатації.

4. Спроєктовано клієнт-серверну архітектуру із розподілом на інтерфейсну частину (frontend), серверну частину (backend) та сховище даних (базу даних), що забезпечує масштабованість та розширюваність функціоналу системи.

5. Розроблено інформаційну модель, яка містить ключові сутності (Користувач, Проєкт, Завдання, Коментар, Вкладення) та їхні взаємозв'язки, що гарантує цілісність і логічність збереження інформації.

6. Здійснено програмну реалізацію веб-застосунку із застосуванням актуальних технологій (NestJS, PostgreSQL, TypeORM), що дало змогу створити гнучку та продуктивну систему для керування завданнями.

7. Виконано функціональне тестування програмного продукту, яке підтвердило правильність функціонування ключових модулів, його стабільність та відповідність встановленим вимогам.

8. У підсумку, було успішно створено веб-застосунок, призначений для управління завданнями в невеликих робочих групах. Цей застосунок забезпечує ефективне планування, контроль за виконанням та сприяє взаємодії між користувачами, а також має значний потенціал для подальшого вдосконалення та розвитку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sommerville I. Software Engineering. – Pearson, 2016. – 766 p.
2. Pressman R. Software Engineering: A Practitioner’s Approach. – McGraw-Hill, 2014. – 970 p.
3. Silberschatz A., Korth H., Sudarshan S. Database System Concepts. – McGraw-Hill, 2019. – 1376 p.
4. PostgreSQL Global Development Group. PostgreSQL Documentation [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/
5. Atlassian. Trello Documentation [Електронний ресурс]. – Режим доступу: https://trello.com
6. Asana. Asana Guide. Official Documentation [Електронний ресурс]. – Режим доступу: https://asana.com/guide
7. Monday.com. Documentation [Електронний ресурс]. – Режим доступу: https://monday.com
8. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Addison-Wesley, 1995. – 395 p.
9. Date C. J. An Introduction to Database Systems. – Pearson Education, 2004. – 1024 p.
10. Elmasri R., Navathe S. Fundamentals of Database Systems. – Pearson, 2016. – 1272 p.
11. Martin R. Clean Architecture: A Craftsman’s Guide to Software Structure and Design. – Prentice Hall, 2017. – 432 p.
12. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. – Addison-Wesley, 2004. – 208 p.
13. Freeman E., Robson E. Head First Design Patterns. – O’Reilly Media, 2020. – 694 p.
14. Mozilla Developer Network. Web Development Documentation [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org
15. Sommerville I. Systems and Software Engineering. – Pearson, 2015. – 568 p.

16. Shneiderman B., Plaisant C. Designing the User Interface: Strategies for Effective Human-Computer Interaction. – Pearson, 2016. – 640 p.
17. PostgreSQL Global Development Group. PostgreSQL Official Documentation [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/
18. NestJS. NestJS Documentation [Електронний ресурс]. – Режим доступу: https://docs.nestjs.com
19. Newman S. Building Microservices: Designing Fine-Grained Systems. – O'Reilly Media, 2021. – 600 p.
20. Kleppmann M. Designing Data-Intensive Applications. – O'Reilly Media, 2017. – 616 p.
21. Fowler M. Patterns of Enterprise Application Architecture. – Addison-Wesley, 2002. – 533 p.
22. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Addison-Wesley, 1994. – 395 p.
23. Richards M., Ford N. Fundamentals of Software Architecture. – O'Reilly Media, 2020. – 419 p.
24. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. – Addison-Wesley, 2003. – 560 p.
25. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С., Ліп'яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

Додаток А

Апробація отриманих результатів

Івашенюк С.О.
Студент 4 курсу ФКІТ ЗУНУ
Науковий керівник к.т.н., доцент Загородня Д.І., кафедра ІОСУ ЗУНУ

РОЗРОБКА ВЕБ-ЗАСТОСУНКУ ДЛЯ КЕРУВАННЯ ПРОЄКТНИМИ ЗАВДАННЯМИ

Вступ. Сучасні компанії та команди дедалі частіше застосовують цифрові рішення для організації та управління спільною роботою. Особливу важливість має питання ефективного керування завданнями у невеликих командах, успіх проєктів яких значною мірою залежить від оперативного обміну даними, контролю за дотриманням строків та розподілу функцій між учасниками. Застосування веб-додатків дозволяє автоматизувати ці механізми, забезпечуючи єдиний доступ до даних та можливість колективної роботи в реальному часі [1-4].

Постанова задачі. Основна мета даної роботи полягає у створенні онлайн-системи для управління проєктними завданнями у невеликих робочих колективах, що дозволить автоматизувати ключові етапи планування, розподілу та відстеження виконання робіт. Об'єктом дослідження є процес організації та управління завданнями в контексті малих робочих груп. Предметом дослідження виступають методології, моделі та програмні засоби, що використовуються для реалізації веб-застосунків, призначених для керування та моніторингу виконання завдань.

Основний матеріал. В рамках даного дослідження було проведено детальний аналіз існуючих систем управління проєктами та завданнями, зокрема таких популярних платформ, як Trello, Asana та Monday.com. Дослідження охоплювало вивчення їхніх функціональних можливостей, інтерфейсних рішень, механізмів організації командної взаємодії, планування та контролю виконання завдань. Особливу увагу було приділено інструментам управління життєвим циклом проєктів, засобам моніторингу прогресу, системам сповіщень, інтеграції з зовнішніми сервісами та можливостям візуалізації даних [5]. За результатами проведеного аналізу було визначено основні переваги та обмеження існуючих рішень, ідентифіковано найбільш затребувані функції для ефективної організації роботи користувачів, а також сформульовано функціональні та нефункціональні вимоги до розроблюваного програмного продукту. Отримані результати стали основою для проєктування архітектури системи та вибору оптимальних технологічних рішень для її реалізації.

Загальна структура програми містить клієнтську і серверну ланки та сховище даних. Клієнтська ланка (Frontend) забезпечує взаємодію користувача із програмою через веб-інтерфейс. Серверна ланка (Backend) виконує опрацювання запитів користувачів та реалізує бізнес-логіку. Сховище даних забезпечує збереження та опрацювання відомостей про користувачів, проєкти та доручення.

Головні функціональні блоки програми:

– блок керування користувачами. Забезпечує: реєстрацію користувачів, вхід у програму, керування особистими даними користувачів;

– блок керування проєктами. Забезпечує: створення проєктів, зміну відомостей про проєкт, додавання учасників до проєкту;

– блок керування дорученнями. Забезпечує: створення доручень, призначення виконавців, встановлення крайніх термінів, зміна стану виконання;

– блок комунікацій. Забезпечує: додавання коментарів до доручень, обмін файлами, сповіщення між учасниками.

Інформаційні зв'язки між частинами програми забезпечуються через передачу даних між клієнтською ланкою, сервером та сховищем даних.

Для проєктування програми застосовано об'єктно-орієнтований метод, який дозволяє представити програму як взаємодію об'єктів предметної сфери. Основними об'єктами програми є: user, project, task, comments та file. Кожен об'єкт має набір ознак та методів, що окреслюють його функціонування у програмі.

Створений веб-застосунок надає функціонал для ефективного управління проєктами та завданнями з використанням Kanban-підходу. Система дозволяє формувати робочі області,

створювати проєкти та задачі, змінювати їхні статуси шляхом переміщення між стовпцями Kanban-дошки, призначати відповідальних виконавців та відстежувати хід виконання робіт. На рисунку 1 представлений інтерфейс розробленої системи для керування завданнями.

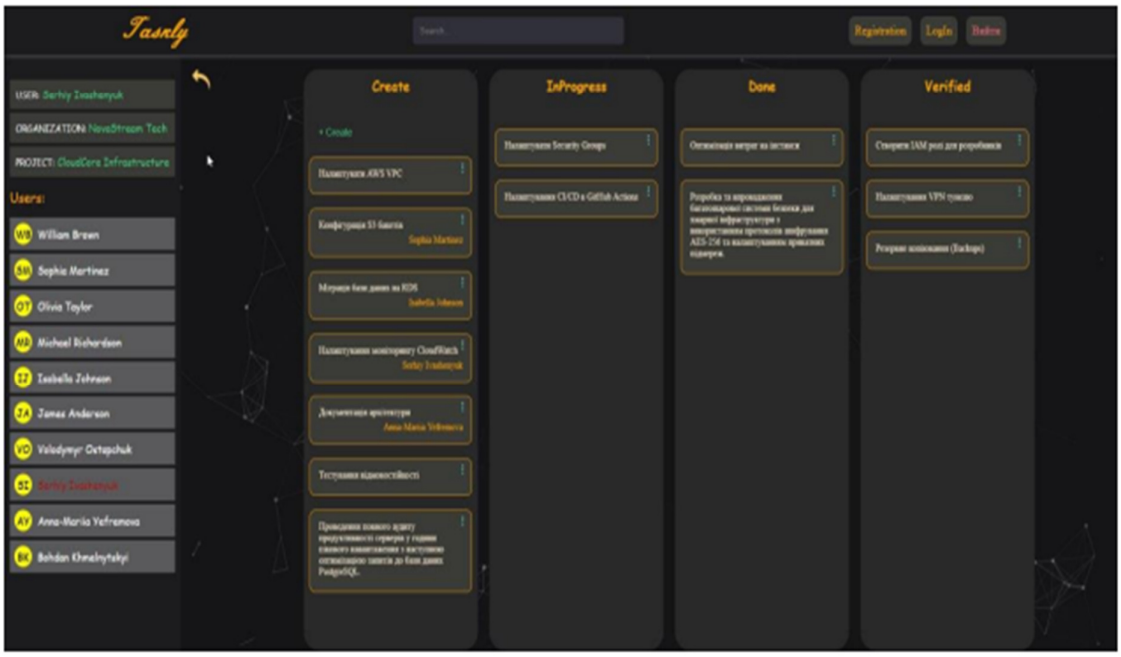

Рисунок 1 – Інтерфейс системи керування завданнями

Клієнтська частина програми реалізована за допомогою фреймворку Nuxt 3, що забезпечує високу швидкодію, зручну архітектуру компонентів та підтримку актуальних стандартів розробки веб-інтерфейсів[3]. Серверна частина побудована на мікросервісній архітектурі з використанням NestJS та GraphQL Gateway, що гарантує масштабованість системи та спрощує подальше розширення її функціональних можливостей.[1]

Для зберігання даних використовується система керування базами даних PostgreSQL[2]. Вона забезпечує надійне збереження інформації, підтримує складні взаємозв'язки між сутностями та демонструє високу продуктивність при роботі зі значними обсягами даних.

Розроблена система підтримує механізми автентифікації та авторизації користувачів, функціонал створення та редагування завдань, їх групування за статусами, а також сприяє ефективній взаємодії між учасниками робочих груп. Застосування Kanban-дошки дозволяє наочно відображати поточний стан виконання проєкту та сприяє підвищенню загальної ефективності командної роботи.

Висновки. У результаті проведеного дослідження було проаналізовано сучасні системи управління проєктами та завданнями, визначено їхні переваги, недоліки та основні функціональні можливості. На основі отриманих результатів сформовано вимоги до розробки веборієнтованої системи керування завданнями для невеликих робочих колективів. Спроєктовано архітектуру програмного забезпечення, що включає клієнтську та серверну частини, а також систему зберігання даних. Програма надає централізоване керування відомостями про користувачів, доручення, зауваження, часові межі виконання та стани робіт. Вона дає змогу організувати дієву взаємодію між членами команди, полегшує процес планування роботи та посилює нагляд за виконанням визначених завдань..

Використання технологій Nuxt 3, NestJS, GraphQL та PostgreSQL забезпечило високу продуктивність, масштабованість і надійність системи. Запровадження Kanban-підходу дозволило візуалізувати стан виконання завдань і підвищити ефективність командної взаємодії. Розроблений програмний продукт може бути використаний для автоматизації управління проєктною діяльністю малих робочих груп та слугувати основою для подальшого розширення функціональних можливостей системи.

Список літератури

1. NestJS. (2026). *NestJS Documentation*. Retrieved May 8, 2026, from https://docs.nestjs.com

2. PostgreSQL Global Development Group. (2026). *PostgreSQL Official Documentation*. Retrieved May 30, 2026, from https://www.postgresql.org/docs/

3. Nuxt.js Team. (2026). *Nuxt Documentation*. Retrieved May 8, 2026, from https://nuxt.com/docs

4. Project Management Institute. (2021). *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)*. Retrieved May 8, 2026, from https://www.pmi.org/pmbok-guide-standards/foundational/pmbok

5. Vue.js Team. (2026). *Vue.js Documentation*. Retrieved May 8, 2026, from

Додаток Б

Сервіс завдань

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { CreateTaskDto } from './dto/createTaskDto';
import { TodoListVersion1 } from './models/todo-list';
import { TokenService } from '../../token/token.service';
import { RefreshJwtTokenEntity } from '../../token/entity/refresh-jwt-token.entity';
import { User } from '../user/models/user.model';
import { JwtService } from '@nestjs/jwt';
import { UpdateTaskDto } from './dto/updateTaskDto';
import { TodoListVersion2 } from './models/todo-list-update';
import { PaginationDto } from './dto/ paginationDto';
import { Status } from '../status/repository/status';
import { Project } from '../project/models/project';
import { InjectRepository } from '@nestjs/typeorm';
import { FindOptionsOrder, Repository } from 'typeorm';

@Injectable()
export class TodolistService {
  constructor(
    @InjectRepository(RefreshJwtTokenEntity)
    private readonly jwtRepository: Repository<RefreshJwtTokenEntity>,
    @InjectRepository(TodoListVersion1)
    private readonly todoListVersion1: Repository<TodoListVersion1>,
    @InjectRepository(TodoListVersion2)
    private readonly todoListVersion2: Repository<TodoListVersion2>,
    private readonly tokenService: TokenService,
    @InjectRepository(User)
    private readonly userRepository: Repository<User>,
    @InjectRepository(Status)
    private readonly todoStatus: Repository<Status>,
    private readonly jwtService: JwtService,
    @InjectRepository(Project)
    private readonly project: Repository<Project>,
  ) {}

  async createTask(dto: CreateTaskDto, token: string) {
    await this.userRepository.findOneOrFail({
      where: { id: dto.reporter_id },
    });

    const decoded = this.jwtService.decode(token);

    const author = await this.userRepository.findOne({
      where: { email: decoded.user.email },
    });
    console.log(decoded);
    const task = this.todoListVersion1.create({
      status_id: dto.status_id,
      author: author.firstName,
      task: dto.task,
      performer_id: author.id,
      reporter_id: dto.reporter_id,
      project_id: dto.project_id,
    });

    await this.todoListVersion1.save(task);
    return task;
  }
```

```
async deleteTask(id: number) {
  return await this.todoListVersion1.delete({ id });
}

async updateTask(id: number, dto: UpdateTaskDto) {
  const sourceData = await this.todoListVersion1.findOneOrFail({
    where: {
      id: id,
    },
  });

  await this.todoListVersion2.save({
    status_id: sourceData.status_id,
    author: sourceData.author,
    reporter_id: sourceData.reporter_id,
    performer_id: sourceData.performer_id,
    task: sourceData.task,
    task_id: sourceData.id,
    project_id: sourceData.project_id,
  });

  if (dto.task) {
    sourceData.task = dto.task;
  }

  if (dto.performer_id) {
    sourceData.performer_id = dto.performer_id;
  }

  if (dto.reporter_id) {
    sourceData.reporter_id = dto.reporter_id;
  }

  await this.todoListVersion1.save(sourceData);

  return sourceData;
}

async allTask(dto: PaginationDto) {
  const { page, limit, sortDirection, sortBy } = dto;
  const offset = (page - 1) * limit;

  const allowedSortFields: (keyof TodoListVersion1)[] = [
    'id',
    'author',
    'performer_id',
    'createdAt',
    'reporter_id',
  ];

  const order: FindOptionsOrder<TodoListVersion1> = {};

  if (allowedSortFields.includes(sortBy as keyof TodoListVersion1)) {
    order[sortBy] = sortDirection.toUpperCase() === 'ASC' ? 'ASC' : 'DESC';
  } else {
    order['createdAt'] = 'DESC'; // дефолтне сортування
  }
  const [items, total] = await this.todoListVersion1.findAndCount({
    take: limit,
    skip: offset,
    order,
  });
```

```
    return {
      data: items,
      total,
      page: page,
      limit: limit,
      lastPage: Math.ceil(total / limit),
    };
  }

  async oneTask(id: number) {
    const findTask = await this.todoListVersion1.findOne({
      where: { id: id },
    });
    if (findTask == null) {
      throw new NotFoundException('task not found');
    }
    return findTask;
  }
}
```

Додаток В

Сервіс авторизації

```
import {
  BadRequestException,
  Injectable,
  UnauthorizedException,
} from '@nestjs/common';
import { UserService } from '../user/user.service';
import { CreateUserDTO } from '../user/dto';
import { AppError } from '../../common/constants/errors';
import { JwtService } from '@nestjs/jwt';
import { TokenService } from '../../token/token.service';
import * as bcrypt from 'bcrypt';
import { AuthUserResponse } from './response';
import { UserLoginDTO } from './dto';
import { AuthResponseDto } from './dto/auth-response.dto';
import { UpdateRefreshDto } from '../../token/dto/updateRefreshDto';
import { ConfigService } from '@nestjs/config';
import { GiveARoleDto } from './dto/give-a-roleDto';
import { User } from '../user/models/user.model';
import { EntityManager, Repository } from 'typeorm';
import { InjectEntityManager, InjectRepository } from '@nestjs/typeorm';
import { PermissionDto } from './dto/permissionDto';
import { RoleRepository } from './repository/role.repository';
import { Permission } from './repository/permission';


@Injectable()
export class AuthService {
  constructor(
    private readonly userService: UserService,
    private readonly tokenService: TokenService,
    private readonly configService: ConfigService,
    private readonly jwtService: JwtService,
    @InjectRepository(User)
    private readonly userRepository: Repository<User>,
    @InjectRepository(RoleRepository)
    private readonly roleRepository: Repository<RoleRepository>,
    @InjectRepository(Permission)
    private readonly permissionRepository: Repository<Permission>,
    @InjectEntityManager()
    private readonly entityManager: EntityManager,
  ) {}

  async registerUsers(
    dto: CreateUserDTO,
    userAgent: string,
  ): Promise<AuthResponseDto> {
    return this.userService.createUser(dto, userAgent);
  }

  async refreshToken(dto: UpdateRefreshDto): Promise<AuthUserResponse> {
    try {
      const payload = await this.jwtService.verifyAsync(dto.refresh_token, {
        secret: this.configService.get('refresh_secret_jwt'),
      });

      const existUser = await this.userService.findUserByEmail(
```

```
      payload.user.email,
    );

    const refresh = await this.tokenService.findByRefreshId(
      dto.refresh_token,
    );

    console.log('токен для оновлення', dto.refresh_token);
    const userData = {
      id: existUser.id,
      name: existUser.firstName,
      email: existUser.email,
      roleId: existUser.role_id,
    };
    if (refresh) {
      const tokens = await this.tokenService.generateJwtToken(userData);
      const userId = existUser.id;
      await this.tokenService.updateAndSaveToken(
        tokens.refresh_token,
        30,
        userId,
      );
      return {
        id: existUser.id,
        email: existUser.email,
        firstName: existUser.firstName,
        lastName: existUser.lastName,
        role_id: existUser.role_id,
        tokens,
      };
    } else {
      throw new UnauthorizedException();
    }
  } catch (e) {
    console.log(e);
    throw new UnauthorizedException();
  }
}

async loginUser(
  dto: UserLoginDTO,
  userAgent: string,
): Promise<AuthUserResponse> {
  const existUser = await this.userService.findUserByEmail(dto.email);
  if (!existUser) throw new BadRequestException(AppError.USER_NOT_EXIST);

  const validatePassword = await bcrypt.compare(
    dto.password,
    existUser.password,
  );
  if (!validatePassword) throw new BadRequestException(AppError.WRONG_DATA);
  const userData = {
    id: existUser.id,
    name: existUser.firstName,
    email: existUser.email,
    roleId: existUser.role_id,
  };
  const tokens = await this.tokenService.generateJwtToken(userData);
  const userId = existUser.id;
  await this.tokenService.saveToken(
    tokens.refresh_token,
    userAgent,
```

```
      30,
      userId,
    );

    return {
      id: existUser.id,
      email: existUser.email,
      firstName: existUser.firstName,
      lastName: existUser.lastName,
      role_id: existUser.role_id,
      tokens,
    };
  }

  async giveARole(dto: GiveARoleDto) {
    const exist = await this.userRepository.findOneByOrFail({
      id: dto.user_id,
    });
    exist.role_id = dto.role_id;
    await this.userRepository.save(exist);
    return { role: dto.role_id };
  }

  async createRole(dto: PermissionDto) {
    return this.entityManager.transaction(async (em) => {
      const role = em.create(RoleRepository, {
        role: dto.nameRole,
        organization_id: dto.organization_id,
      });
      const saveRole = await em.save(role);
      role.permissions = dto.permissions.map((permission) => {
        return em.create(Permission, { permission, role: saveRole });
      });
      await em.save(role.permissions);
      return role;
    });
  }

  async allRole(organization_id: number) {
    return await this.roleRepository.find({
      where: { organization_id: organization_id },
    });
  }

  async deleteRole(role_id: number) {
    await this.roleRepository.delete(role_id);
    return true;
  }
}
```