МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ЕКОНОМІКО-ПРАВНИЧИЙ ФАХОВИЙ КОЛЕДЖ
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ»

Циклова комісія математичних дисциплін та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «РОЗРОБКА СИСТЕМИ ДОМОПОГИ МЕНЕДЖЕРУ ПРОЄКТІВ»

Виконав: здобувач освіти 4 курсу, групи К121-21

Спеціальність 121 Інженерія програмного забезпечення
(шифр і спеціальність)

Денис ДЕРИВЕДМІДЬ
(ім'я та ПРІЗВИЩЕ)

Керівник Роман КОТОВ
(ім'я та ПРІЗВИЩЕ)
доцент кафедри «Інформаційна безпека та наноелектроніка» НУ «Запорізька політехніка»,

Рецензент к.т.н., доцент Ганна НЕЛАСА
(посада, вчене звання, науковий ступінь, ім'я та ПРІЗВИЩЕ)

Запоріжжя
2025

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ВСП «Економіко-правничий фаховий коледж ЗНУ»

Освітньо-професійний ступінь фаховий молодший бакалавр

Спеціальність 121 Інженерія програмного забезпечення

(шифр і назва)

ЗАТВЕРДЖУЮ
Голова циклової комісії математичних дисциплін та інформаційних технологій

_____ Тетяна СМОЛЯНКОВА
(підпис)

“ 14 ” червня 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

ДЕРИВЕДМІДЮ Денису Михайловичу
(ПРІЗВИЩЕ ім’я та по-батькові)

1. Тема роботи Розробка системи допомоги менеджеру проєктів

Керівник роботи КОТОВ Роман Олександрович
(посада, науковий ступінь, вчене звання, ПРІЗВИЩЕ ім’я та по-батькові)

Затверджено наказом ВСП ЕПФК ЗНУ від « 26» листопада 2024 р № 1852-с

2. Строк подання студентом роботи 13.06.2025

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1.Сучасні інформаційні системи
2. Розробка проєкту інформаційної системи
3. Програмна реалізація інформаційної системи

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація до захисту

6. Консультанти розділів роботи

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

7. Дата видачі завдання 01.12.2024

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи:		
	Постановка задачі	Грудень 2024	виконано
2.	Збір вихідних даних, обробка методичних		
	та теоретичних джерел	Січень 2025	виконано
3.	Розробка першого розділу:		
	Огляд інформаційних систем та технологій для їхнього створення	Лютий 2025	виконано
4.	Розробка другого розділу:		
	Розробка проєкту інформаційної системи та вибір технологій	Квітень 2025	виконано
5.	Розробка третього розділу:		
	Розробка застосунку	Травень 2025	виконано
6.	Оформлення і нормоконтроль кваліфікаційної роботи та перевірка на плагіат	Червень 2025	виконано
7.	Захист кваліфікаційної роботи	20.06.2025	виконано

Здобувач освіти _____ Денис ДЕРИВЕДМІДЬ
(підпис) (ім'я та ПРІЗВИЩЕ)

Керівник роботи _____ Роман КОТОВ
(підпис) (ім'я та ПРІЗВИЩЕ)

Нормоконтроль пройдено

Нормоконтролер _____ Юлія БОРИСОВСЬКА
(підпис) (ім'я та ПРІЗВИЩЕ)

РЕФЕРАТ

Кваліфікаційна робота: 48 сторінок, 6 зображень, 8 джерел в переліку посилань.

Об’єкт дослідження: процес розробки, впровадження та використання інформаційних систем у сфері управління проєктами.

Предмет дослідження: система для управління проєктами.

Мета роботи: розробка та реалізація вебзастосунку, який забезпечить менеджеру проєктів інструменти для ефективної організації задач, контролю їх виконання та координації роботи команди.

Методи дослідження: аналітичний, експериментальний, порівняльний аналіз.

У кваліфікаційній роботі проведено детальний аналіз предметної області та розроблено цілісний процес створення вебзастосунку. Використовуючи теоретичні відомості про розробку інформаційних систем, а також мову програмування Python і технології веброзробки (HTML, CSS, формат даних JSON), було створено функціональну систему під назвою «Project Manager Assistant». В рамках дослідження ретельно спроєктовано та реалізовано інтерфейс системи, що повністю задовольняє вимоги користувачів і забезпечує ефективну взаємодію. З використанням клієнт-серверної архітектури (серверна частина на Python, клієнтська – вебінтерфейс HTML/CSS) застосунок «Project Manager Assistant» успішно реалізовано та протестовано.

Ця система є результатом ґрунтовного вивчення питань автоматизації управління проєктами, а його функціональність спрямована на надання менеджерам проєктів можливостей оптимізувати планування та контроль завдань. Робота демонструє не лише знання теоретичних аспектів розробки програмного забезпечення й управління проєктами, але й уміння успішно втілити їх у практичному застосуванні.

У результаті проведеного дослідження досягнуто поставленої мети – розроблено та впроваджено систему, яка забезпечує менеджеру проєктів ефективні засоби керування Проєктами та командною роботою. Створена система може бути використана на практиці для підвищення ефективності процесів управління Проєктами та продуктивності командної роботи.

PYTHON, HTML, CSS, JSON, API, КЛІЄНТСЬКА ЧАСТИНА, СЕРВЕР, УПРАВЛІННЯ ПРОЄКТАМИ.

SUMMARY

Qualification paper: 48 pages, 8 references, 6 images.

Research object: the process of developing, implementing, and using information systems in project management.

Research subject: system for project management.

Purpose of the work: development and implementation of a system that provides project managers with tools for efficient task organization, execution control, and team coordination.

Research methods: analytical, experimental, comparative analysis.

In this qualification paper, a comprehensive analysis of the domain was conducted, and an integrated process for creating an application was developed. Using theoretical insights into information systems development, as well as the Python programming language and web development technologies (HTML, CSS, JSON data format), a functional web application named "Project Manager Assistant" was successfully created. Within the research framework, the application interface was thoroughly designed and implemented, fully meeting user requirements and ensuring efficient interaction. Employing a client-server architecture (server-side implemented with Python and client-side via HTML/CSS), the "Project Manager Assistant" application was successfully realized and tested.

This application is the outcome of an extensive study of automation issues in project management, with functionality designed to provide project managers with capabilities for optimized task planning and control. The research demonstrates not only theoretical knowledge in software development and project management but also the ability to successfully apply these concepts practically.

As a result of the conducted research, the stated objective was achieved—the web application that offers project managers effective tools for managing projects and teamwork was developed and implemented. The created system can be practically used to enhance efficiency in project management processes and improve team productivity.

PYTHON, HTML, CSS, JSON, API, CLIENT SIDE, SERVER, PROJECT MANAGEMENT.

ЗМІСТ

Завдання на кваліфікаційну роботу студенту 2
Реферат 4
Summary 6
Вступ 8
1 Огляд предметної області 10
1.1 Огляд систем допомоги менеджеру Проєктів 11
1.2 Основні компоненти систем допомоги менеджеру Проєктів 13
1.3 Висновки до розділу 1 16
2 Проєктування системи 18
2.1 Опис технічних вимог 18
2.2 Вибір архітектури 21
2.3 Вибір бази даних 23
2.4 Вибір серверної частини 25
2.5 Проєктування інтерфейсу 28
2.6 Висновки до розділу 2 31
3 Програмна реалізація системи 33
3.1 Створення серверної частини 33
3.2 Створення клієнтської частини 36
3.3 Тестування системи 41
3.4 Висновки до розділу 3 44
Висновки 46
Перелік використаних джерел 48

ВСТУП

Сучасний розвиток інформаційних технологій супроводжується стрімким зростанням кількості ІТ-Проєктів, що потребують ефективного управління. Це підтверджується прогнозами світових фахівців: за даними Інституту управління Проєктами (PMI), до 2027 року кількість спеціалістів, необхідних для роботи у сфері управління Проєктами, сягне близько 88 мільйонів осіб (на 33% більше порівняно з 2017 роком) [1]. Постійне збільшення масштабу і складності Проєктної діяльності значно ускладнює роботу менеджера Проєкту: потрібно одночасно контролювати виконання великої кількості задач, дотримуватися визначених графіків і бюджетів, координувати команду та оперативно реагувати на зміни вимог і виникнення ризиків. В таких умовах виникає нагальна потреба в автоматизації процесів управління Проєктами.

Впровадження сучасних систем управління Проєктами допомагає підвищити продуктивність і точність роботи: програмні рішення беруть на себе значну частину відстеження прогресу, надсилання нагадувань про дедлайни та формування звітності. Це дозволяє менеджеру Проєкту зосередитися на стратегічних аспектах – роботі з командою, плануванні та аналізі ризиків – замість виконання монотонних адміністративних дій. Крім того, автоматизація забезпечує прозорість командної роботи: усі члени команди і зацікавлені сторони можуть у режимі реального часу бачити актуальний стан Проєкту (прогрес виконання задач, відповідальні виконавці, пріоритети тощо). Така прозорість покращує координацію і підвищує ймовірність успішного завершення Проєкту.

Отже, метою даної дипломної роботи є розробка повноцінного вебдодатку – системи допомоги менеджеру Проєктів, яка надаватиме керівнику Проєкту інструменти для ефективної організації задач, контролю їх виконання та координації роботи команди. Для досягнення цієї мети необхідно

було створити програмний продукт із клієнтською та серверною частинами, що забезпечуватиме управління Проєктами і їхніми завданнями на всіх етапах – від постановки та планування до відстеження прогресу й внесення змін у разі потреби.

Завдання дослідження. Для реалізації поставленої мети в роботі потрібно було вирішити такі основні завдання:

Огляд предметної області. Дослідити роль і функції менеджера Проєкту та проаналізувати основні труднощі його професійної діяльності. Розглянути існуючі аналогічні системи управління Проєктами (Jira, Trello, ClickUp) – широко відомі інструменти – та визначити спільні елементи таких програмних рішень (автоматизоване планування, управління ресурсами, термінами, ризиками, звітністю тощо).

Проєктування системи. Розробити концепцію та архітектуру програмної системи допомоги менеджеру Проєктів. На цьому етапі сформулювати технічні вимоги до функціоналу системи, обґрунтувати вибір архітектури, способу зберігання даних (тип бази даних) і технологій для серверної та клієнтської частин, а також сПроєктувати інтерфейс користувача у вигляді прототипу.

Програмна реалізація. Створити робочий вебдодаток згідно з Проєктним планом: реалізувати серверну частину (сервер застосунку, базу даних та API для взаємодії) і клієнтську частину (вебінтерфейс користувача), забезпечивши обробку даних та необхідний функціонал керування Проєктами й задачами. Провести тестування системи, перевіривши відповідність реалізованого функціоналу визначеним вимогам і надійну роботу програми.

Аналіз результатів і висновки. Оцінити отримані результати розробки та ступінь досягнення поставленої мети. У підсумковому розділі (висновках) узагальнити основні переваги створеного рішення для менеджера Проєктів (спрощення планування і контролю, підвищення ефективності командної роботи тощо) та окреслити можливі напрями подальшого розвитку і вдосконалення розробленої системи.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

Проєктний менеджер (менеджер Проєкту) – це фахівець, який організовує, планує та керує виконанням Проєкту, працюючи в рамках визначених обмежень (часових, фінансових тощо). Він відповідає за формування команди, визначення цілей, комунікацію із зацікавленими сторонами та успішне завершення Проєкту.

У своїй діяльності менеджери Проєктів стикаються з численними труднощами: необхідно одночасно контролювати виконання багатьох завдань, дотримуватися затверджених графіків і бюджетів, координувати команду та своєчасно реагувати на зміни вимог.

Часто використовується декілька розрізнених інструментів для планування, обліку та комунікації, і надмірна складність таких засобів ускладнює роботу. Брак єдиного простого рішення призводить до витрат часу на навчання команди різним системам і знижує ефективність управління. Внаслідок цього з’являється потреба в більш повній автоматизації управління Проєктними завданнями.

Автоматизація допомагає підвищити продуктивність, зменшити кількість помилок і обсяг рутинної роботи, покращити дисципліну праці та розширити можливості команди. Сучасні програмні рішення беруть на себе значну частину контролю прогресу, нагадувань про терміни та формування звітності, що дозволяє керівнику Проєкту зосередитися на стратегічних завданнях і роботі з людьми замість монотонних адміністративних дій.

Важливим є те, що автоматизовані системи забезпечують прозорість: усі члени команди і стейкхолдери бачать актуальний стан Проєкту (виконані та невиконані задачі, відповідальні, пріоритети, строки тощо) у реальному часі. Таким чином, впровадження автоматизованого управління завданнями суттєво спрощує роботу менеджера Проєкту та підвищує ймовірність успішного завершення Проєкту.

1.1 Огляд систем допомоги менеджеру Проєктів

Для підтримки діяльності менеджерів Проєктів розроблено багато спеціалізованих систем управління Проєктами. Ці програмні комплекси допомагають планувати Проєкти, призначати та відстежувати виконання завдань, управляти ресурсами і строками. Серед найбільш відомих прикладів – платформи Jira, Trello та ClickUp.

Jira (розробник – Atlassian) [2] – програмний продукт для відстеження помилок, керування завданнями та гнучкого управління Проєктами. Jira широко використовується в IT для ведення беклогу задач, підтримки методологій Scrum і Kanban, а також для управління випусками програмного забезпечення.

До переваг Jira належать потужні засоби обліку та класифікації задач (гнучкі дошки, настроювані робочі процеси), розвинені можливості звітності та аналітики (діаграми згоряння, звіти спринтів тощо), а також тісна інтеграція з іншими продуктами Atlassian (Confluence, Bitbucket). Водночас система Jira має і недоліки: новим користувачам може бути складно оволодіти всіма функціями через значну складність інтерфейсу; гнучкість налаштувань обмежена визначеними рамками (у деяких конкурентів вона вища); для невеликих команд вартість ліцензії при масштабуванні може стати відчутною.

Таким чином, Jira орієнтована передусім на великі та технічно підготовлені колективи, де потрібен потужний інструмент управління розробкою, але може бути надмірною для простіших Проєктів.

Trello [3] – це веборієнтована система управління завданнями у стилі канбан, що надає візуальний підхід до організації роботи. В основі Trello лежать дошки (boards), розділені на списки, в яких розміщуються картки завдань.

Такий підхід є інтуїтивно зрозумілим: завдання переміщуються між списками, які зазвичай відповідають статусам (наприклад, “До виконання”, “В процесі”, “Завершено”), що дає змогу миттєво оцінити прогрес роботи. Сильними сторонами Trello є простота і зручність користування. Інтерфейс надзвичайно дружній до користувача, що робить цей інструмент привабливим

для малих команд та особистих Проєктів. Базовий функціонал Trello доступний безкоштовно (можна створювати до 10 дощок з необмеженою кількістю карток). Додатково Trello підтримує різні уявлення даних (таймлайн, календар, табличний вигляд) і інтеграції (так звані Power-Ups) для розширення можливостей.

Головним недоліком Trello є обмежена функціональність для великих або складних Проєктів: в ньому бракує багатьох просунутих можливостей, необхідних для масштабного управління (наприклад, повноцінного відстеження ресурсів, розширеної аналітики, керування бюджетом). Тому Trello чудово підходить для візуальної організації простих процесів і командної роботи на початкових етапах, але при зростанні Проєкту чи ускладненні вимог може виникнути потреба перейти на більш потужний інструмент.

ClickUp [4] – сучасна хмарна платформа «все-в-одному» для управління роботою, яка позиціонується як універсальний робочий простір для командної співпраці та підвищення ефективності. Вона підтримує різноманітні функції: управління завданнями з різними видами представлення (список, дошка, календар, діаграма Ґанта тощо), спільну роботу з документами і файлами, вбудований чат для комунікації, відстеження цілей і OKR, побудову звітів, а також має широкий вибір інтеграцій з іншими сервісами.

Сильні сторони ClickUp – висока кастомізація (користувачі можуть налаштовувати статуси завдань, поля, вигляди під свої процеси) та дружній інтерфейс, який попри велику кількість функцій залишається доволі зрозумілим. ClickUp пропонує багато можливостей навіть у безкоштовному тарифному плані та недорогих платних планах, що робить його привабливим для невеликих компаній.

З іншого боку, надмірно широкий набір функцій іноді призводить до «перевантаження» – команди можуть витрачати зайвий час на налаштування або плутатися в численних можливостях платформи. Для дуже великих Проєктів інколи повідомляють про зниження продуктивності системи.

Отже, ClickUp добре підходить тим, хто прагне об’єднати в одному середовищі всі аспекти роботи над Проєктом і готовий інвестувати час у його

конфігурацію під власні потреби.

Проаналізувавши існуючі рішення, можна зробити висновок, що попри різноманітність функцій у подібних системах, часто виникає потреба у створенні власного інструмента. Причинами цього можуть бути специфічні вимоги конкретної команди чи організації, які не покриваються наявними засобами; бажання мати більш простий і «легковаговий» додаток без зайвих залежностей; необхідність інтеграції з внутрішніми процесами компанії.

Багато універсальних систем (як-от розглянуті Jira, Trello, ClickUp) є занадто складними або надлишковими для вузьких задач. Тому розробка персоналізованого помічника менеджера Проєктів здатна забезпечити кращу адаптацію під конкретні потреби користувача і усунути недоліки існуючих аналогів.

1.2 Основні компоненти систем допомоги менеджеру Проєктів

Незважаючи на різноманітність систем управління Проєктами, більшість із них мають схожу базову структуру даних та користувацький інтерфейс. Ключовими поняттями є Проєкт і задача. Проєкт – це сукупність взаємопов’язаних задач, об’єднаних спільною метою та характеристиками (наприклад, назва Проєкту, його опис, дати початку та завершення, склад команди виконавців). Багато систем дозволяють вести одночасно кілька Проєктів і швидко перемикатися між ними; при цьому кожен Проєкт зберігає власну структуру задач, не змішуючись із даними інших Проєктів.

Задача – основна одиниця роботи в Проєктному менеджменті. Задачі містять назву (стислий опис роботи), розгорнутий опис, пріоритет, відповідального виконавця, термін виконання (дедлайн) та інші атрибути. Для кожної задачі фіксується її статус у робочому процесі. Як правило, використовуються типові статуси, що відображають життєвий цикл задачі: наприклад, “Нова” (або To Do, ще не розпочато), “В роботі” (або In Progress, виконується), “Готово” (або Done, завершено). У складніших випадках

додаються проміжні стани на кшталт “На перевірці”, “Призупинено” чи інші – залежно від процесів конкретної команди. Переведення задачі в той чи інший статус фіксує прогрес і забезпечує команді спільне бачення перебігу роботи над Проєктом. Окрім статусу, важливими атрибутами задач є пріоритет (визначає черговість та важливість задачі, наприклад високий, середній або низький пріоритет) та термін виконання – планова дата завершення. Система може автоматично відзначати прострочені задачі або піднімати їх у списках, привертаючи увагу менеджера до критичних пунктів плану. Гнучке редагування властивостей задач – обов’язкова функція: менеджер у будь-який момент може скоригувати опис, змінити дедлайн чи пріоритет, перепризначити відповідального тощо, щоби динамічно реагувати на зміни в Проєкті.

Інтерфейс користувача систем управління Проєктами зазвичай орієнтований на наочне відображення стану Проєкту і зручність виконання основних операцій. Як правило, він містить декілька ключових компонентів. Перший – це панель управління (dashboard) або оглядова сторінка, де відображаються зведені показники Проєкту: загальна кількість задач, скільки з них відкрито, у роботі чи виконано; можуть наочно показуватися графічні індикатори прогресу (наприклад, процент виконання у вигляді прогрес-бару). Також на інформаційній панелі можуть бути представлені нагадування про дедлайни (які задачі скоро мають бути виконані або вже прострочені) та інші статистичні дані, що дають менеджеру “пульс” Проєкту в реальному часі.

Другий ключовий компонент – представлення списку задач. Існує декілька популярних підходів до відображення переліку задач: табличний вигляд та дошка Kanban. У табличному (списковому) вигляді задачі представлені у вигляді рядків таблиці, де колонки відповідають основним атрибутам (назва задачі, статус, виконавець, пріоритет, дедлайн тощо). Користувач може сортувати чи фільтрувати цей список, швидко знаходячи потрібні завдання. Натомість Kanban-дошка відображає задачі у вигляді карток, розкладених по стовпцях, що відповідають статусам. Картки можна

“перетягувати” між стовпцями, змінюючи статус задачі – такий спосіб особливо наочний для командної роботи за методологіями Agile. Багато систем (наприклад, згадані Trello, Jira, ClickUp) підтримують обидва підходи, дозволяючи перемикатися між різними поданнями даних залежно від потреб користувача.

Задачі згруповані за статусами: To Do (заплановані), In Progress (у процесі виконання), Done (завершені). Кожна картка містить назву завдання, короткі деталі, позначки пріоритету чи інші індикатори, і може мати призначеного виконавця (позначається аватаром або ініціалами на картці). Візуальне групування і переміщення карток між колонками спрощує відстеження прогресу та координацію роботи команди.

Третій важливий компонент – засоби створення та редагування. Інтерфейс зазвичай містить зручні форми для додавання нового Проєкту або нової задачі. При створенні Проєкту користувач вводить назву, опис, може визначити учасників команди. При створенні задачі – вказує її назву, деталі, вибирає пріоритет, статус (початково найчастіше “Нова”), призначає відповідального зі списку учасників Проєкту, встановлює дедлайн тощо. Багато систем надають можливості швидкого редагування прямо у списках: наприклад, змінити статус чи виконавця задачі можна випадаючим списком або перетягуванням мишею, без відкриття окремого вікна. Такі інтерактивні елементи економлять час і роблять роботу інтуїтивно зрозумілою. Додатково інтерфейс може включати механізми фільтрації та пошуку задач (за текстом, відповідальним, пріоритетом тощо), що дозволяє користувачу швидко знаходити потрібні елементи навіть у великому Проєкті.

В цілому, основні складові типового програмного засобу для керування Проєктами – це логічна модель даних (Проєкти та задачі з усіма атрибутами, статуси, пріоритети, виконавці) і користувацький інтерфейс (панель показників, список/дошка задач, форми додавання та редагування, засоби навігації). Продумана реалізація цих компонентів дозволяє ефективно планувати роботу та відстежувати її виконання в реальному часі.

1.3 Висновки до розділу 1

У сучасному менеджменті Проєктів існує велика кількість інструментів, що полегшують працю керівника Проєкту, проте актуальність завдання розробки власної системи залишається високою. По-перше, діяльність менеджера Проєктів є складною і багатоаспектною: необхідно керувати командою, строками, завданнями, ризиками, комунікаціями. Без допоміжних програмних засобів ефективно виконувати всі ці обов’язки вкрай важко, особливо у великих чи розподілених командах. Автоматизація рутинних процесів (відстеження статусів задач, нагадування про дедлайни, формування звітів) є актуальною потребою, що підвищує ймовірність успіху Проєкту та зменшує навантаження на менеджера.

По-друге, доступні аналоги у вигляді відомих систем (Jira, Trello, ClickUp та інші) надають широкі можливості, але не завжди оптимально підходять для кожної організації. Як було показано, одні системи можуть бути занадто складними у впровадженні та використанні, інші – надто простими і не охоплюють усіх необхідних функцій. Універсальні інструменти часто потребують адаптації бізнес-процесів під себе або дорогого розширення для отримання потрібних можливостей. Це породжує ситуацію, коли менеджери Проєктів змушені користуватися кількома різними програмами одночасно, що фрагментує інформацію і ускладнює робочий процес.

Отже, виникає необхідність створення власної системи, орієнтованої на конкретні вимоги та умови використання. Власний вебдодаток «Project Manager Assistant» задуманий як легке рішення без зайвої складності, що включає тільки найважливіші функції управління Проєктами та завданнями. Він має поєднати ключові переваги існуючих систем (наочність, спільний доступ до даних, простоту оновлення статусів та контролю прогресу) і одночасно бути адаптованим під специфічні потреби користувача (зокрема, враховувати локальні вимоги, уникати надмірних функцій, не залежати від зовнішніх сервісів). Більшість систем дійсно мають схожі базові можливості,

проте деталі реалізації та гнучкість налаштування визначають успішність їх впровадження. У власному рішенні можна врахувати всі виявлені недоліки аналогів та забезпечити необхідну масштабованість і зручність.

Підсумовуючи, розробка персональної системи підтримки менеджера Проєктів є виправданою, коли існуючі інструменти не повністю відповідають вимогам конкретного користувача. Така система повинна містити стандартний набір функцій (управління Проєктами, завданнями, статусами, строками, виконавцями) та інтуїтивний інтерфейс, а її головна перевага – адаптація під конкретний контекст діяльності. Це забезпечить ефективну підтримку менеджера Проєкту в його роботі та підвищить результативність управління Проєктами в цілому.

2 ПРОЄКТУВАННЯ СИСТЕМИ

Етап Проєктування системи є критичним кроком у розробці програмного забезпечення, оскільки саме на цьому етапі визначаються структура і поведінка майбутньої системи. Під час Проєктування виконуються розробка концепції рішення, опрацювання архітектури та вибір технічних рішень, необхідних для успішної реалізації Проєкту. Проєктувальна команда встановлює чіткі критерії та стандарти, планує компоненти системи і їх взаємодію.

Основні цілі цього етапу – однозначно визначити вимоги до системи, поставити зрозумілі завдання розробникам та задокументувати ключові рішення. У результаті Проєктування формується повне технічне завдання (ТЗ) з описом усіх вимог і обраних рішень, а також створюються допоміжні матеріали (діаграми, макети інтерфейсу), які спрощують подальшу реалізацію.

Таким чином, Проєктування системи забезпечує надійний фундамент для розробки, мінімізує ризики непорозумінь і дозволяє впевнитися, що кінцевий продукт відповідатиме поставленим вимогам.

2.1 Опис технічних вимог

Чітке визначення технічних вимог є запорукою успішного розвитку системи. Технічні вимоги описують очікувану функціональність та обмеження системи, виконуючи роль своєрідного контракту між замовником і розробником. Вони визначають, що саме має робити програмний продукт і за яких умов він це виконує. Якісно сформульовані вимоги полегшують процес дизайну та реалізації, а також спрощують перевірку готової системи на відповідність очікуванням.

Для системи «Project Manager Assistant» були визначені й реалізовані

такі ключові технічні вимоги:

- Підтримка кількох Проєктів: система має оперувати множиною Проєктів, дозволяючи створювати нові Проєкти та перемикатися між ними. Кожен Проєкт ізольовано містить свої дані (список задач, команду виконавців тощо), тобто інформація різних Проєктів не змішується.

- Керування переліком задач у Проєкті: у межах кожного Проєкту користувач може додавати необхідну кількість задач. Для кожної задачі зберігаються основні атрибути: заголовок, опис, пріоритет, статус, відповідальний, дедлайн та дата створення. Це дозволяє вести облік різноманітних робочих завдань у контексті одного Проєкту.

- Змінювані статуси задач: кожна задача повинна мати статус життєвого циклу (наприклад, Open, In Progress, Done), який можна змінювати у процесі роботи. Це забезпечує відстеження прогресу виконання завдань та гнучке управління їхнім станом протягом часу.

- Відсутність видалення задач: у системі навмисно не передбачено фізичне видалення задач. Завдання, навіть виконані, не вилучаються з бази, а залишаються у статусі “Done”. Такий підхід гарантує цілісність історичних даних і дозволяє надалі аналізувати всі виконані завдання, зберігаючи повну картину Проєкту.

- Мінімальні зовнішні залежності: реалізація повинна обходитися без потреби встановлення сторонніх бібліотек або сервісів. Увесь необхідний функціонал досягається засобами стандартної мови Python [5] та вбудованих модулів, а на клієнті – стандартними вебтехнологіями (HTML, CSS, JavaScript). Такий підхід спрощує розгортання системи та усуває потенційні проблеми сумісності, оскільки не залежить від сторонніх компонентів.

- Наявність лаунчера для запуску: передбачено спеціальний запускний модуль (“лаунчер”), який одним кроком запускає серверну частину і відкриває клієнтський інтерфейс у браузері. Це підвищує зручність для кінцевого користувача, дозволяючи легко розпочати роботу з застосунком без виконання складних налаштувань.

Клієнт-серверна архітектура: система повинна бути побудована за архітектурою «клієнт–сервер». Вебінтерфейс (клієнтська частина) працює у браузері користувача, звертаючись до серверної частини через мережеві запити. Така розподілена структура підвищує гнучкість та масштабованість рішення, чітко розділяючи логіку зберігання даних і презентації інформації.

- Запуск у браузері: користувацький інтерфейс має функціонувати як вебсторінка, що відкривається у стандартному браузері. Таким чином, система не потребує встановлення окремого клієнта – достатньо відкрити локальну вебсторінку, яку генерує сервер. В результаті забезпечується крос-платформність та зручність доступу до системи на будь-якому пристрої з браузером.

- Створення та редагування задач: система надає можливість додавати нові задачі через інтерфейс (шляхом заповнення форми з полями задачі). Введені користувачем дані передаються на сервер, де створюється новий запис у базі даних (файлі JSON) [8]. Також повинна бути можливість за потреби коригувати інформацію про існуючу задачу (наприклад, змінити її статус або інші атрибути) – ця функціональність реалізована через відповідні запити до серверного API.

- Перемикання між Проєктами: інтерфейс повинен мати зручний механізм вибору поточного Проєкту (наприклад, випадаючий список доступних Проєктів). Після обрання іншого Проєкту користувач бачить і керує задачами саме в контексті цього вибраного Проєкту. На серверній частині зберігається інформація про активний Проєкт і повертаються відповідні дані при запитах.

-Відображення дашборду прогресу: для кожного Проєкту формується візуальна інформаційна панель (дашборд) зі зведенням ключових показників: загальна кількість задач, кількість відкритих, у роботі та виконаних задач, а також кількість прострочених завдань. Дашборд відображає процент виконання (співвідношення завершених задач) у вигляді прогрес-бару. Ця функціональність покликана надати користувачу швидкий огляд стану

Проєкту в режимі реального часу.

Перелічені вимоги були враховані при Проєктуванні системи та згодом реалізовані в коді. Вони охоплюють усю основну функціональність, необхідну для «Project Manager Assistant», і гарантують, що кінцевий продукт задовольнить потреби користувача у керуванні Проєктами та завданнями. У наступних підрозділах пояснюється, яким чином ці вимоги вплинули на вибір архітектури, технологій і рішень для системи.

2.2 Вибір архітектури

Для даної системи обрано клієнт-серверну архітектуру, виходячи з вимог до функціональності та зручності використання. Клієнт-серверна модель передбачає розподіл ролей: клієнтська частина відповідає за взаємодію з користувачем, тоді як серверна частина обробляє бізнес-логіку і зберігає дані. Такий підхід забезпечує чітке розділення функцій між компонентами системи і спрощує їх розробку та підтримку.

У межах обраної архітектури серверна частина реалізована як автономний вебсервер, що надає API для основних операцій. Сервер відповідає за зберігання даних (використовуючи JSON-файл як просту базу даних) та за обробку запитів від клієнта: створення Проєктів, додавання задач, зміну статусів, перемикання поточного Проєкту, отримання списку задач і показників дашборду. Фактично сервер виступає єдиним джерелом правдивих даних (single source of truth) для клієнта: усі операції з даними проходять через чітко визначені кінцеві точки API (HTTP-запити типу GET або POST).

Клієнтська частина системи – це інтерактивна вебсторінка, що відображає інтерфейс користувача та взаємодіє із сервером через мережеві запити. При запуску застосунку сервер надсилає клієнту початкову HTML-сторінку з усіма необхідними скриптами та стилями. У браузері ця сторінка надає динамічний інтерфейс: відображає список Проєктів і задач, дашборд,

форми введення даних. Клієнтська частина виконує запити до серверного API (наприклад, GET /api/projects для отримання переліку Проєктів, POST /api/tasks для створення нової задачі тощо) та оновлює відображення на основі відповіді. Така взаємодія реалізована у стилі RESTful, що сприяє простоті й надійності комунікації між фронтендом і бекендом. Ви можете побачити діаграму розгортання додатку на рисунку 2.1.

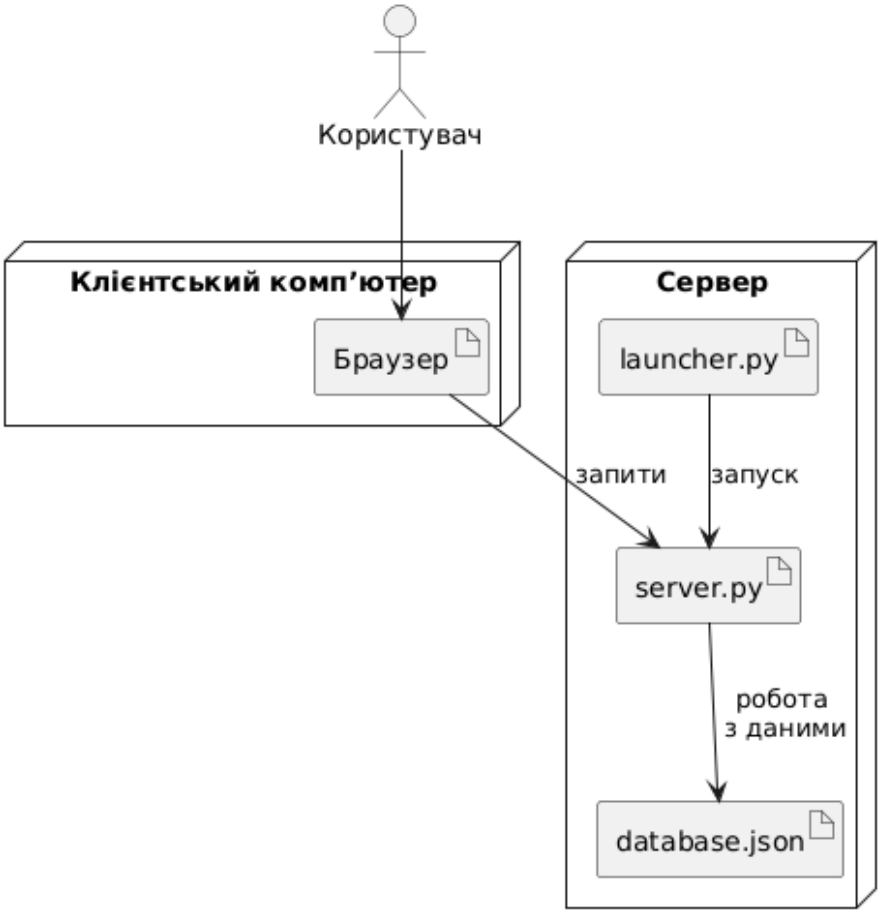


Рисунок 2.1 – Діаграма розгортання

Вибір клієнт-серверної архітектури обґрунтовується низкою переваг. По-перше, масштабованість – за потреби серверну частину можна розгорнути на віддаленому хості, забезпечивши багатокористувацький доступ, тоді як клієнти (браузери) зможуть підключатися до неї через мережу. По-друге, підтримуваність – розділення на клієнт і сервер дозволяє незалежно модифікувати інтерфейс або серверну логіку без взаємного впливу, що

полегшує внесення змін у майбутньому. Нарешті, зручність розповсюдження – користувачу достатньо передати лише невеликий серверний застосунок; інтерфейс же відкривається у будь-якому браузері, не потребуючи складної інсталяції чи налаштування.

Для наочності архітектурне рішення і процес розгортання системи були відображені на діаграмі розгортання. На ній показано, що серверна програма Project Manager Assistant запускається на локальному комп'ютері (в середовищі Python), відкриваючи мережевий порт (стандартно, наприклад, 8080). Клієнтська частина представлена вебзастосунком, який підключається до серверу за адресою http://localhost:8080. Діаграма демонструє взаємозв'язок компонентів: користувач працює з браузером, браузер надсилає HTTP-запити серверу, а сервер звертається до JSON-бази даних і повертає результати назад клієнту. Така схема розгортання підтверджує, що компоненти системи логічно розділені, але фізично можуть працювати на одній машині (або бути легко перенесені на різні машини при масштабуванні системи).

Отже, клієнт-серверна архітектура дозволила повністю задовольнити визначені технічні вимоги, забезпечивши гнучкість, модульність і зручність використання системи «Project Manager Assistant». Вибір цієї архітектури став основою для подальших рішень щодо вибору бази даних та технологій реалізації як серверної, так і клієнтської частини.

2.3 Вибір бази даних

На етапі Проєктування було вирішено використовувати JSON-файл як базу даних для зберігання інформації про Проєкти та задачі. Формат JSON (JavaScript Object Notation) обрано через його простоту, легкість та достатність для поставлених завдань. JSON є текстовим форматом, що підтримується всіма сучасними мовами програмування і легко читається як людиною, так і машиною. У контексті невеликої системи, де обсяг даних відносно малий і

одночасний доступ здійснюється лише локально, текстовий файл JSON є цілком ефективним рішенням. Таке рішення не потребує налаштування окремого сервера баз даних і виключає додаткові залежності, тобто відповідає вимозі мінімальних залежностей.

Структура JSON-бази даних у системі Project Manager Assistant сПроєктована таким чином, щоб відображати ієрархію “Проєкт – задачі”. База даних складається з об’єкта верхнього рівня projects, який містить множину Проєктів. Кожен Проєкт у цьому файлі представлений окремим вкладеним об’єктом, ключем якого є назва Проєкту. Наприклад, можуть бути два записи: "Website Redesign" і "Mobile App Development" – назви Проєктів. Кожен об’єкт Проєкту має такі поля: name (назва Проєкту, рядок), description (опис Проєкту, текст), tasks (масив задач, пов’язаних із цим Проєктом; кожен елемент масиву – це об’єкт, що відповідає одній задачі з її атрибутами: ідентифікатор, заголовок, опис, пріоритет, статус, виконавець, строк виконання, дата створення), team_members (список учасників команди; для простоти реалізації може залишатися порожнім або використовуватися для зазначення відповідальних осіб) і created_at (дата створення Проєкту).

Таким чином, всі дані Проєкту (включно з усіма його задачами) компактно зберігаються у форматі JSON як вкладена структура. Додатково на верхньому рівні файлу присутнє поле current_project, що зберігає назву активного (поточного) Проєкту, з яким наразі працює користувач. Це поле дозволяє системі пам’ятати контекст між запусками програми (після перезапуску додатку останній відкритий Проєкт залишиться обраним активним).

Варто зазначити, що використання JSON як сховища даних спрощує процес серіалізації та десеріалізації. Серверна частина може легко перетворювати об’єкти Проєктів і задач у JSON-рядок для запису у файл і назад у об’єкти при завантаженні (завдяки стандартним бібліотекам Python для роботи з JSON). За потреби файл JSON можна відкрити і переглянути вручну – зрозуміла текстова структура полегшує налагодження і перевірку даних.

Для формалізації структури даних та взаємозв’язків між об’єктами була також розроблена UML-діаграма класів. На цій діаграмі відображено основні класи системи та зв’язки між ними. Зокрема, клас ProjectManagerAssistant (відповідає за керування всією системою) містить колекцію об’єктів Project. Кожен об’єкт класу Project має список своїх Task. Діаграма класів показує, що між Project та Task існує відношення “один-до-багатьох” (один Проєкт містить багато задач), а також відображає атрибути кожного класу. Наприклад, клас Task містить поля id, title, description, priority, status, assignee, due_date, created_at. Наявність UML-діаграми класів на етапі Проєктування допомогла чітко спланувати структуру даних у коді та переконатися, що всі необхідні сутності й їх атрибути враховані. Крім того, така діаграма служить елементом документації, який полегшує розуміння системи іншим розробникам або зацікавленим особам.

У підсумку, вибір JSON-файлу як бази даних виявився виправданим з точки зору простоти та достатності для даного Проєкту. Він відповідає вимогам щодо легкості й автономності системи, а сПроєктована структура даних (підтверджена UML-діаграмою) забезпечує зручну роботу з задачами і Проєктами в коді.

2.4 Вибір серверної частини

Як мову та платформу для реалізації серверної частини було обрано Python. Це рішення прийнято з огляду на низку переваг, які надає Python для наших цілей:

Вбудований HTTP-сервер: стандартна бібліотека Python містить модуль http.server, що дозволяє швидко створити простий вебсервер без використання сторонніх фреймворків. У Проєкті Project Manager Assistant застосовано цей модуль для обробки HTTP-запитів. Це суттєво скоротило обсяг коду та спростило розгортання – серверна програма може працювати «з коробки», без

додаткових налаштувань.

Підтримка JSON: Python має інтегровані засоби для роботи з JSON (модуль json). Завдяки цьому легко серіалізувати об’єкти (Проєкти, задачі) у JSON-формат і зберігати їх у файл, а також читати файл і перетворювати дані назад у об’єкти. Така нативна підтримка спрощує реалізацію збереження даних і відповідає вимозі легкої роботи з JSON-базою.

Популярність і простота: Python є однією з найпоширеніших мов програмування, відомою своєю лаконічністю та простим синтаксисом. Завдяки цьому розробка серверної логіки (класів для Проєктів і задач, API-методів) проходила швидко і зрозуміло. Код на Python відзначається високою читабельністю, що полегшує подальшу підтримку або доопрацювання системи.

Невелике навантаження: Очікуване навантаження на систему Project Manager Assistant – невелике, оскільки вона призначена для локального використання одним менеджером чи невеликою командою. Python повністю справляється з обробкою таких обсягів даних та кількості запитів, не створюючи проблем з продуктивністю. Для випадку багатокористувацької високонавантаженої системи, можливо, знадобилося б використати більш спеціалізовані рішення, але для нашого Проєкту Python є оптимальним вибором.

Відсутність додаткових залежностей: реалізація серверної частини на Python не потребує встановлення сторонніх бібліотек чи середовищ. Весь функціонал досягнуто за рахунок стандартних можливостей мови. Це значно спрощує поширення програми – достатньо, щоб на комп’ютері був встановлений інтерпретатор Python відповідної версії. Користувач може запустити файл pm_launcher.py, який активує сервер, і відразу отримати працюючу систему. Такий «легковаговий» підхід узгоджується з вимогами мінімальних залежностей та простоти запуску.

Серверна частина була побудована з використанням кількох ключових класів. Зокрема, клас ProjectManagerAssistant відповідає за високорівневі

операції (завантаження/збереження JSON-файлу, створення Проєктів тощо). Клас Project інкапсулює дані Проєкту і містить методи для маніпуляції задачами (додавання нової задачі, отримання задач за статусом, оновлення статусу). Клас Task представляє окрему задачу з її властивостями. Обробка HTTP-запитів здійснюється шляхом перевизначення методів do_GET та do_POST у класі-обробнику, який наслідує BaseHTTPRequestHandler. У цьому обробнику реалізовані маршрути (ендпоїнти) – наприклад, /api/projects, /api/tasks, /api/dashboard, /api/switch-project тощо – які викликають відповідні методи класу ProjectManagerAssistant і повертають клієнту потрібні дані у форматі JSON. Ви можете побачити діаграму класів на рисунку 2.2.

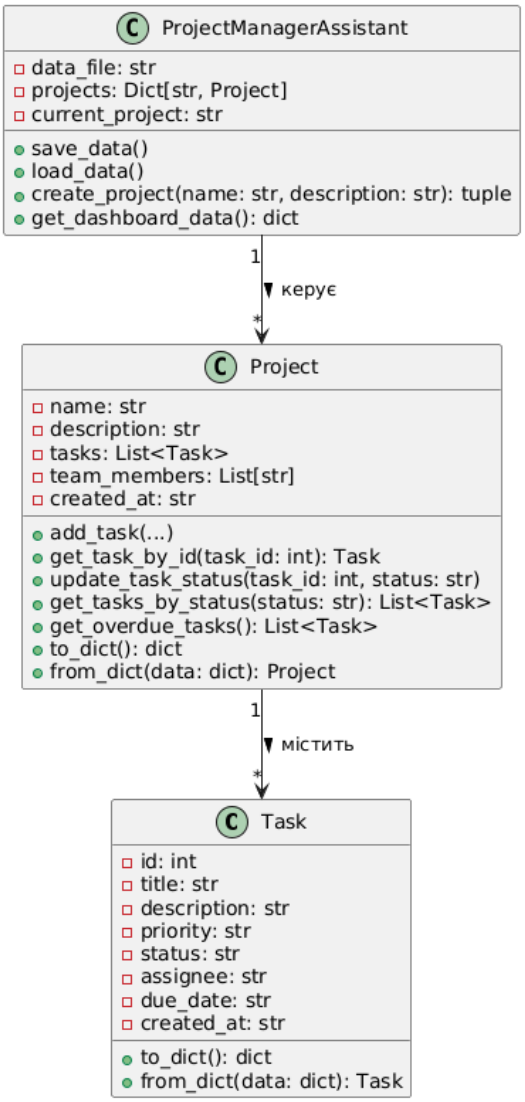


Рисунок 2.2 – Діаграма класів

Вибір Python позитивно вплинув і на швидкість розробки інтерфейсу: сервер динамічно формує HTML-сторінку і відправляє її клієнту. Завдяки простоті роботи зі строками в Python, шаблон HTML/CSS/JS було безпосередньо вбудовано в код сервера, що дозволило обійтися без окремих шаблонізаторів чи складних налаштувань. Хоча такий підхід придатний переважно для демонстраційних та невеликих Проєктів, у рамках Project Manager Assistant він виявився достатньо зручним і швидким у реалізації.

Зважаючи на всі ці фактори, Python став раціональним вибором для серверної частини: він забезпечив необхідну функціональність, стабільність і простоту розробки, не створюючи зайвих бар'єрів для користувача та розробників.

2.5 Проєктування інтерфейсу

Проєктування інтерфейсу користувача є важливою складовою розробки, адже від нього залежить зручність та ефективність роботи кінцевого користувача із системою. На цьому етапі визначається, як саме інформація та функції будуть представлені користувачу, які елементи керування необхідні та як вони взаємодіятимуть між собою. Добре продуманий інтерфейс зменшує кількість помилок при введенні даних, полегшує навчання роботі з системою і підвищує продуктивність користувача.

Для Project Manager Assistant було вирішено реалізувати вебінтерфейс, використовуючи стандартні технології HTML [6] і CSS [7] (та мінімальний JavaScript для динамічної взаємодії). Вибір на користь зв'язки HTML+CSS зумовлений їх універсальністю та крос-платформенністю – такий інтерфейс працюватиме у будь-якому сучасному браузері без додаткових плагінів або налаштувань. Крім того, використання стандартних вебтехнологій дає змогу швидко створити прототип інтерфейсу і при потребі легко внести в нього зміни.

На етапі Проєктування інтерфейсу було визначено основні компоненти, які він має містити:

Шапка застосунку: у верхній частині вебсторінки відображається заголовок системи – назва Project Manager Assistant та короткий слоган або опис. Цей елемент дозволяє користувачу швидко зрозуміти, з яким додатком він працює, і формує єдиний стиль оформлення інтерфейсу.

Перемикач Проєкту: елемент інтерфейсу, що дозволяє обрати активний Проєкт для роботи. Реалізовано у вигляді випадаючого списку під міткою “Current Project”. Список динамічно заповнюється назвами всіх наявних Проєктів із бази даних; обрання Проєкту призводить до автоматичного завантаження його даних (через виклик відповідного API) і оновлення всіх пов’язаних секцій інтерфейсу (дашборду, списку задач тощо).

Інформаційна панель (дашборд): центральний блок для моніторингу статусу Проєкту. На дашборді у зручній візуальній формі відображаються основні статистичні показники: загальна кількість задач у Проєкті, кількість відкритих, в процесі і виконаних задач, а також число прострочених задач. Кожен показник представлений у вигляді картки з коротким заголовком і числовим значенням. Під картками розташована смуга прогресу, яка графічно демонструє відсоток виконаних задач (наприклад, заповнення смуги на 50% при виконанні половини задач). Дашборд автоматично оновлюється при кожній зміні в задачах, надаючи актуальну інформацію про прогрес Проєкту.

Форми для введення даних: інтерфейс містить дві основні форми – форму створення нового Проєкту і форму додавання нової задачі. Форма створення Проєкту складається з полів “Project Name” та “Description”, що дозволяє користувачу вказати назву нового Проєкту і його опис. При відправленні цієї форми (натисканні кнопки "Create Project") генерується запит на сервер для створення Проєкту, після чого список Проєктів та інші компоненти інтерфейсу оновлюються. Форма додавання задачі містить поля для введення назви задачі, опису, пріоритету (з випадаючого списку значень Low/Medium/High), імені виконавця та дати дедлайну. Після натискання

кнопки "Add Task" введена інформація надсилається на сервер, який додає нову задачу до поточного Проєкту. Обидві форми розташовані поруч (в окремих колонках секції форм) для зручності та чітко позначені заголовками (“Create New Project” та “Add New Task”).

Список задач: нижче форм знаходиться секція зі списком задач поточного Проєкту. Вона відображає всі задачі вибраного Проєкту у вигляді вертикального списку карток (або рядків таблиці). Кожна задача показує свою назву та коротку мета-інформацію – пріоритет, статус, виконавця і дату дедлайну. Якщо у задачі є детальний опис, він також може відображатися під назвою. Важливим інтерактивним елементом у кожній задачі є випадаючий список для зміни статусу: користувач може безпосередньо зі списку перевести задачу, наприклад, з Open в In Progress або Done. Цей випадаючий список відображає поточний статус і дозволяє вибрати інший; при зміні статусу генерується запит до серверу для оновлення даних, після чого список задач і дашборд перезавантажуються, відображаючи актуальну інформацію. Завдяки такому механізму зміни статусів відбуваються динамічно, і результат одразу видно – це покращує досвід користувача (немає потреби вручну оновлювати сторінку).

Перш ніж реалізовувати інтерфейс у коді, був створений чорновий макет інтерфейсу (wireframe). Цей каркасний макет являв собою спрощене схематичне зображення сторінки, де було показано розташування основних блоків: місце заголовка, випадаючого списку Проєктів, блоку дашборду, форм і списку задач. Макет допоміг перевірити логічну організацію інтерфейсу ще до написання коду, а також обговорити його з потенційними користувачами (наприклад, менеджерами Проєктів) для отримання зворотного зв’язку. На основі цього макету було підтверджено, що обрані компоненти і їх розміщення є інтуїтивно зрозумілими.

При стилізації інтерфейсу за допомогою CSS особливу увагу приділено зрозумілому візуальному кодуванню інформації. Використано спокійну, нейтральну кольорову гаму з акцентами: наприклад, для позначення статусів

задач застосовуються різні кольори (високий пріоритет або прострочені задачі підсвічуються теплими відтінками, виконані задачі – зеленим тощо). Елементи інтерфейсу (кнопки, поля форм) оформлені в єдиному стилі, завдяки чому система сприймається цілісно. CSS забезпечує також адаптивність: використано гнучкі сітки та відносні розміри, тому інтерфейс підлаштовується під різний розмір вікна браузера, зберігаючи читабельність і зручність користування як на великих моніторах, так і на ноутбуках чи планшетах.

Отже, Проєктування інтерфейсу завершилося створенням прототипу, що відповідає всім вимогам до системи. Розроблений інтерфейс відображає всю потрібну інформацію (Проєкти, задачі, статуси, показники прогресу), надає засоби для введення нових даних та керування ними, і при цьому залишається простим та зрозумілим у використанні. Обрані технології HTML+CSS (у поєднанні з JavaScript) забезпечили швидку та ефективну реалізацію цього дизайну.

2.6 Висновки до розділу 2

У даному розділі було здійснено повне Проєктування системи «Project Manager Assistant». Спочатку сформовано технічне завдання і визначено основні технічні вимоги: систему сПроєктовано для підтримки кількох Проєктів і множини завдань, з можливістю змінювати статуси задач, при цьому дані фіксуються у JSON-файлі і видалення задач не передбачене; реалізація мала мінімізувати зовнішні залежності та забезпечити зручний запуск через лаунчер. На основі цих вимог обґрунтовано вибір архітектури: клієнт-серверна модель дозволила розділити логіку на бекенд (сервер на Python) і фронтенд (вебінтерфейс у браузері), що забезпечило гнучкість і масштабованість рішення. Розглянуто вибір бази даних – використання JSON-файлу як простого сховища даних виправдало себе в контексті невеликої системи, спростивши роботу з даними та уникнувши розгортання окремої

СУБД; структура бази (Проєкти з вкладеними задачами) і відповідні класи були сПроєктовані та проілюстровані на UML-діаграмі. Також обрано технологію серверної частини – мову Python, яка завдяки вбудованому HTTP-серверу та підтримці JSON повністю задовольнила потреби системи, дозволивши уникнути зайвих залежностей і забезпечивши достатню продуктивність для поставлених задач. Нарешті, виконано Проєктування інтерфейсу користувача: визначено всі необхідні компоненти (дашборд, перемикач Проєктів, форми введення, список задач) і їх розташування, підготовлено wireframe-макет та згодом реалізовано його засобами HTML/CSS. Всі ці кроки Проєктування стали міцною основою для наступного етапу – безпосередньої програмної реалізації – оскільки на момент початку імплементації вже існувало чітке бачення структури системи, її функцій і зовнішнього вигляду.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

Розділ 3 присвячено переходу від Проєктування до створення працездатного продукту – вебзастосунку Project Manager Assistant. На цьому етапі було реалізовано серверну та клієнтську частини системи відповідно до розробленого Проєкту. Результатом є готовий прототип додатка, яким можуть користуватися кінцеві користувачі. У підрозділах наведено основні кроки реалізації: створення серверної частини (бекенд), створення клієнтської частини (фронтенд), проведення тестування застосунку, а також підсумки й обговорення труднощів, що виникли під час реалізації.

3.1 Створення серверної частини

Архітектура серверної частини. Серверна частина системи реалізована мовою Python у вигляді легкої вебслужби. Вона відповідає за збереження даних про Проєкти та задачі, обробку HTTP-запитів до API та повернення відповідей клієнту. Структурно сервер складається з двох основних компонентів – безпосередньо серверного модуля та спеціального лаунчера (програми запуску). Лаунчер автоматизує старт серверу і відкриття клієнтського застосунку в браузері користувача. Зокрема, файл pm_launcher.py спочатку перевіряє наявність серверного файлу, після чого запускає його як підпроцес через інтерпретатор Python. Перед запуском у консоль виводиться інформація про старт серверу та основні можливості демо-версії (створення Проєктів, відстеження задач, оновлення даних у реальному часі тощо). Такий підхід дозволяє користувачу запустити систему подвійним кліком по launcher-файлу: він автоматично піднімає сервер і відкриває сторінку застосунку у веббраузері (для відкриття URL використовується модуль webbrowser).

Основні функції серверу. Сервер побудований з використанням

стандартного модуля Python http.server. За замовчуванням він запускається на локальному хості (http://localhost) на порту 8080. Сервер працює на базі класу запитів BaseHTTPRequestHandler – у Проєкті створено підклас PMRequestHandler, який перевизначає обробники GET і POST запитів. Таким чином бекенд надає простий API для взаємодії з даними Проєктів та задач. Зокрема, сервер обробляє такі запити: GET / (повертає головну HTML-сторінку клієнтської частини); GET /api/projects (повертає список Проєктів і позначення поточного активного Проєкту); GET /api/tasks (повертає перелік задач поточного Проєкту); GET /api/dashboard (повертає агреговані дані для дашборду – кількість задач за статусами, відсоток виконання тощо); POST /api/projects (створює новий Проєкт); POST /api/tasks (створює нову задачу в межах активного Проєкту); POST /api/tasks/update (оновлює статус існуючої задачі) та POST /api/switch-project (перемикає активний Проєкт).

Для обслуговування цих запитів у класі PMRequestHandler реалізовано методи do_GET і do_POST, які аналізують шлях запиту та викликають відповідні функції обробки. Наприклад, при запиті GET /api/projects викликається метод get_projects(), що формує відповідь у вигляді JSON-об’єкта з переліком Проєктів та назвою поточного активного Проєкту. Якщо в системі збережено два Проєкти, скажімо “Website Redesign” і “Mobile App Development”, відповідь міститиме масив Проєктів із їхніми назвами та поле current_project з позначенням активного Проєкту. Аналогічно, метод get_tasks() звертається до менеджера Проєктів і отримує список задач активного Проєкту у вигляді масиву об’єктів (словників) JSON. Для агрегованої інформації метод get_dashboard() повертає статистику задач (скільки відкрито, в роботі, виконано, прострочено тощо) разом із відсотком виконання Проєкту.

Ключові компоненти та збереження даних. Серверний код використовує допоміжні класи Task (представляє задачу) і Project (представляє Проєкт). Ці класи інкапсулюють властивості задач (назва, опис, пріоритет, статус, відповідальний, строк тощо) і Проєктів (назва, опис, список задач, учасники

команди). Для управління всіма даними створено клас ProjectManagerAssistant, який містить словник Проєктів та назву поточного Проєкту. При старті сервер завантажує початкові дані з файлу pm_data.json – локального JSON-файлу, що зберігає стан Проєктів. У ньому структура даних відповідає описаній у розділі 2: секція "projects" (Проєкти з вкладеними задачами) та поле "current_project" для збереження вибору активного Проєкту. У випадку першого запуску, якщо файл даних відсутній або порожній, сервер автоматично генерує демо-дані – створює кілька прикладних Проєктів і задач для демонстрації функціональності. Усі зміни, які вносить користувач під час роботи (нові Проєкти, задачі, оновлення статусів), зберігаються через метод save_data(), що перетворює поточні об’єкти у відповідну структуру словників і перезаписує JSON-файл. Таким чином, при наступному запуску програми метод load_data() відновлює всі дані, збережені раніше.

Приклад реалізації API та формування відповіді. На сервері всі відповіді API формуються у форматі JSON з відповідними HTTP-заголовками. Розглянемо приклад реалізації створення нового Проєкту: клієнтська частина надсилає POST-запит на шлях /api/projects з JSON-даними, що містять назву (name) і опис (description) Проєкту. У методі do_POST цей запит перенаправляється до функції create_project(), яка викликає відповідний метод менеджера Проєктів і отримує результат успішності операції. Після цього сервер формує JSON-відповідь з полем success (булеве значення) та повідомленням message. Відправка здійснюється через допоміжний метод send_json_response(), який встановлює код відповіді 200 OK, тип контенту application/json і передає рядок JSON у вихідний потік. Така структура відповіді уніфікована для більшості POST-запитів (створення/оновлення), що спрощує обробку результатів на фронтенді.

Підтримка CORS. У процесі розробки виникла потреба забезпечити можливість виконання запитів до API з різних джерел (наприклад, для налагодження або потенційного розширення функціональності). Для цього було реалізовано підтримку механізму Cross-Origin Resource Sharing (CORS).

CORS дозволяє браузеру відмінити політику єдиного походження і надати доступ до ресурсів сервера з іншого домену чи порту, якщо сервер належним чином налаштований. У нашому сервері CORS-запити дозволені шляхом встановлення спеціальних HTTP-заголовків у відповідях. Зокрема, в класі PMRequestHandler визначено метод _set_cors_headers(), який додає до заголовків відповіді такі директиви: Access-Control-Allow-Origin: * (дозвіл звернення з будь-якого домену), Access-Control-Allow-Methods: GET, POST, OPTIONS (дозволені типи запитів) і Access-Control-Allow-Headers: Content-Type (дозволений заголовок Content-Type). Метод _set_cors_headers() викликається для всіх JSON-відповідей, а також в обробнику OPTIONS-запитів (на preflight-запит браузера сервер повертає код 200 OK, сигналізуючи, що основний запит можна виконувати). У нашому випадку клієнтська частина звертається до сервера з того самого хоста, тому CORS більшою мірою потрібен для сценаріїв розробки. Втім, його реалізація підвищує гнучкість системи на майбутнє.

Таким чином, серверна частина Project Manager Assistant перетворює розроблену модель даних на робочий вебсервіс із HTTP API. Запустивши лаунчер, користувач отримує у браузері інтерфейс, а сервер у фоновому режимі обробляє всі дії користувача – створення Проєктів і задач, перемикання Проєктів, зміну статусів завдань тощо.

3.2 Створення клієнтської частини

Доставка клієнтського застосунку через HTTP. Клієнтська (фронтенд) частина застосунку автоматично завантажується з серверу при відкритті сторінки в браузері. Коли користувач звертається до кореневого URL (наприклад, http://localhost:8080/), сервер повертає заздалегідь підготовлений HTML-документ з усіма необхідними стилями та скриптами. Цей документ формує єдиний сторінковий інтерфейс (single-page application), який після

завантаження динамічно звертається до API для отримання даних. Передача HTML безпосередньо з сервера (без окремих статичних файлів .html, .css, .js) спрощує розгортання – достатньо запустити Python-сервер, і він сам видасть клієнту всю сторінку.

Структура HTML-сторінки. Вихідний код сторінки містить розмітку інтерфейсу менеджера Проєктів. У заголовку документа визначено метатеги та базові стилі CSS, що задають оформлення елементів (в нашій реалізації стилі для спрощення вбудовані прямо у HTML). Далі, в тілі HTML, розміщено основні секції додатку:

Шапка (Header): містить заголовок із назвою застосунку та коротким слоганом.

Вибір Проєкту: випадаючий список <select> для перемикання між Проєктами, розміщений у блоці з написом "Current Project". На момент початкового завантаження сторінки список Проєктів порожній – він буде наповнений скриптом після отримання даних від серверу. Для елемента вибору Проєкту встановлено обробник події onchange="switchProject()" – тобто при виборі іншого Проєкту викликається функція switchProject(), яка відправляє відповідний запит до сервера.

Інформаційна панель (Dashboard): блок, що відображає назву поточного Проєкту та ключові статистичні показники. Спочатку цей блок прихований (має стиль display: none) і з’являється тільки після вибору або створення Проєкту. Він містить сітку карток зі статистикою: загальна кількість задач, скільки з них відкрито, в прогресі, завершено та прострочено. Нижче розташована смуга прогресу, яка графічно показує відсоток виконаних задач. Після завантаження даних з серверу скрипт вставляє актуальні значення у елементи цієї панелі (кількість задач, заповнення прогрес-бару, текст відсотка виконання).

Форми створення: секція з двох форм – зліва форма створення нового Проєкту, справа форма додавання нової задачі. Вони дозволяють вводити назву і опис Проєкту, а також параметри нової задачі (заголовок, опис,

пріоритет, відповідальний, дата дедлайну). Кожна форма має кнопку підтвердження: "Create Project" для створення Проєкту та "Add Task" для додавання задачі. Наприклад, у формі створення Проєкту поле "Project Name" має атрибут required, тому браузер не дозволить відправити порожню назву (це базова валідація на рівні HTML). Кнопка відправки має CSS-клас btn, який стилізує її як синю кнопку. Вигляд форм створення можна побачити на рисунку 3.1.

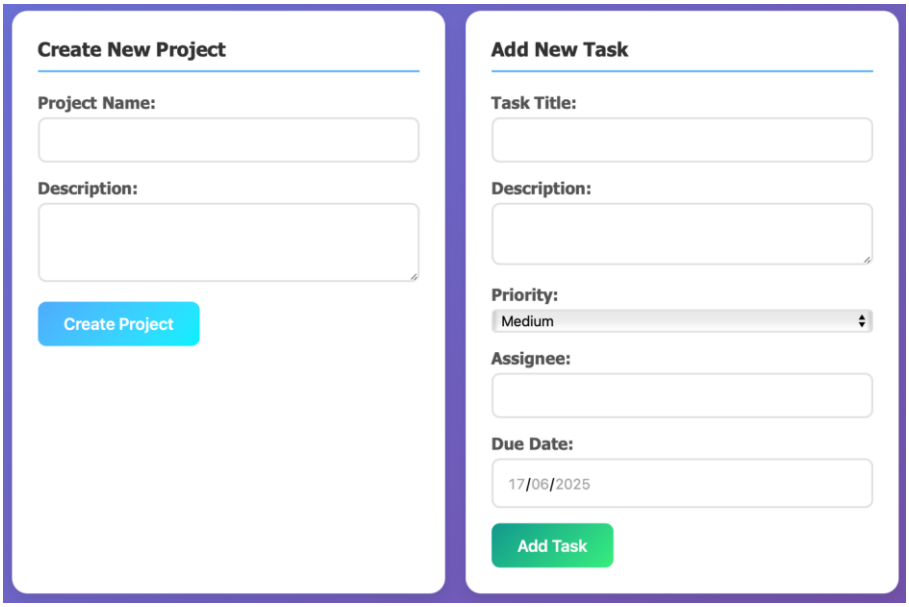


Рисунок 3.1 – Форми редагування

Список задач: нижче форм розташована секція Tasks, яка містить заголовок списку задач і контейнер #tasksList. Під час початкового завантаження сторінки цей контейнер порожній. Після вибору Проєкту або створення задач скрипт на фронтенді отримує список задач від серверу і динамічно наповнює #tasksList елементами. Кожна задача відображається у вигляді окремого блоку з класом task-item та кольоровою міткою зліва (смужка, колір якої залежить від пріоритету та статусу задачі). Вміст блоку задачі включає назву задачі (у тегу <h4>), мета-дані (пріоритет, статус, відповідальний, дедлайн – у блоці <p> з класом task-meta), опис (якщо задано) та випадаючий список для зміни статусу. Нижче наведено спрощений приклад

HTML-розмітки однієї задачі, що генерується скриптом:

```
        <div class="task-item high-priority">
            <div class="task-info">
                <h4>Create wireframes</h4>
                <p class="task-meta">Priority: High | Status: In Progress | Assigned to: Alice | Due: 2024-12-15</p>
                <p>Design new layout and user flow</p>
            </div>
            <div class="task-actions">
                <select onchange="updateTaskStatus(1, this.value)">
                    <option value="Open">Open</option>
                    <option value="In Progress" selected>In Progress</option>
                    <option value="Done">Done</option>
                </select>
            </div>
        </div>
```

У цьому прикладі задача "Create wireframes" має високий пріоритет (що відображається червоною смужкою зліва завдяки класу high-priority) і статус "In Progress" (у випадаючому списку вибрано відповідний пункт). Користувач може змінити статус задачі прямо зі списку: при зміні значення викликається JavaScript-функція updateTaskStatus(), яка відправляє POST-запит на сервер для оновлення статусу. Вигляд списку задач можна побачити на рисунку 3.2.



Рисунок 3.2 – Список задач

Клієнтський скрипт. Наприкінці HTML-документа розміщено сценарій на JavaScript, який керує динамічною поведінкою інтерфейсу. Після завантаження сторінки (подія window.onload) скрипт виконує початкові запити до сервера: завантажує список Проєктів, дані для дашборду та список задач для поточного Проєкту через відповідні виклики API (наприклад, fetch('/api/projects'), fetch('/api/dashboard') тощо). Отримавши дані, скрипт заповнює елементи інтерфейсу наступним чином:

Наповнює випадаючий список Проєктів елементами <option> згідно з отриманим JSON (data.projects) та встановлює вибір на активному Проєкті (data.current_project).

Відображає панель дашборду (роблячи її видимою) і підставляє числа задач у відповідні поля (#totalTasks, #openTasks тощо), а також оновлює заповнення прогрес-бару.

Створює HTML-елементи для кожної задачі і додає їх у контейнер #tasksList. Генерація задач здійснюється через функцію createTaskElement(task), логіку якої наведено вище у прикладі.

Крім завантаження, скрипт обробляє події форм. Наприклад, при сабміті форми створення Проєкту (projectForm) зчитуються введені назва та опис і відправляються POST-запитом на сервер. У відповідь сервер повертає повідомлення про успішність; після цього скрипт оновлює список Проєктів і дашборд, щоб відобразити новостворений Проєкт. Аналогічно, при додаванні

нової задачі обробник форми задач (taskForm) зчитує всі поля задачі і відправляє їх на сервер POST-запитом, після чого оновлює список задач та дашборд відповідно до відповіді.

Отже, клієнтська частина забезпечує повноцінний інтерактивний інтерфейс, який динамічно оновлюється на основі даних із серверу. Вона реалізує весь необхідний функціонал для взаємодії користувача із системою у зручний спосіб, реагуючи на дії користувача та відображаючи актуальну інформацію без перезавантаження сторінки.

3.3 Тестування системи

Після реалізації серверної і клієнтської частин було проведено всебічне тестування системи Project Manager Assistant. Метою тестування було впевнитися у стабільності роботи системи, крос-браузерній сумісності, адаптивності інтерфейсу та коректній роботі функцій додатку відповідно до вимог.

Тестування в різних браузерах. Система перевірена в актуальних версіях популярних браузерів: Google Chrome, Mozilla Firefox, Microsoft Edge. В усіх цих браузерах інтерфейс відображався коректно, стилі застосовувалися однаково, а функціональність працювала без збоїв (пересилання форм, перемикання Проєктів, оновлення статусів задач тощо). Оскільки в реалізації використовувалися стандартизовані технології HTML5 і сучасний API fetch для JavaScript (який підтримується згаданими браузерами), проблем сумісності не виникло. Додатково було здійснено пробний запуск у Safari на macOS – інтерфейс також працював належним чином.

Тестування на мобільних пристроях. Для перевірки адаптивності інтерфейсу було протестовано відкриття застосунку на смартфоні (браузер Chrome на Android). Завдяки застосованим у стилях правилам адаптивності (*CSS media queries*) інтерфейс автоматично перебудовується під вузький

екран. Зокрема, блоки форм, які на широкому екрані розташовані поруч, на смартфоні відображаються один під одним, а список задач змінює компоновку: елементи *.task-item* стають вертикально орієнтованими з відступами між ними. Під час тесту на телефоні всі елементи інтерфейсу залишалися читабельними: текст достатнього розміру, кнопки і поля вводу зручні для натискання.



Рисунок 3.3 – Десктопний вигляд

Перевірено також, що використані градієнтні фони та стилі коректно відображаються у мобільному браузері. Таким чином, можна стверджувати, що розроблений інтерфейс є респонсивним (адаптивним) і придатним для використання як на десктопі, так і на мобільних пристроях. Вигляд додатку на мобільних пристроях та на десктопі можна побачити на рисунках 3.3 та 3.4.

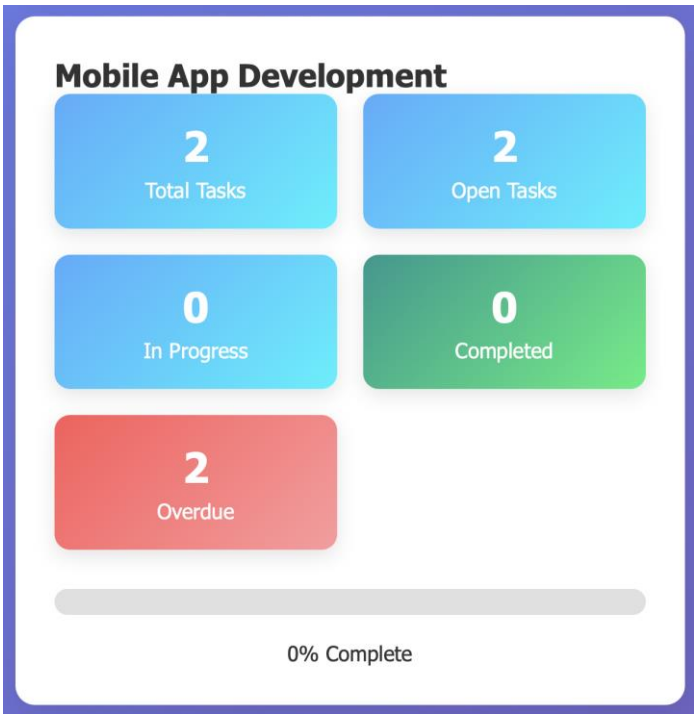


Рисунок 3.4 – Мобільний вигляд

Перевірка багатокористувацького доступу. Хоча система не є багатокористувацьким онлайн-сервісом, було змодельовано одночасне відкриття програми у декількох вкладках браузера на одному комп'ютері. При цьому всі вкладки звертаються до одного бекенду і спільного файлу даних. Перевірено, що дії в одній вкладці (наприклад, створення нової задачі або зміна статусу) впливають на загальний стан: інші вкладки при наступному зверненні до серверу отримують вже оновлені дані (наприклад, якщо перезавантажити сторінку або виконати дію, що тригерить *fetch*-запит). Прямих конфліктів при паралельному додаванні Проєктів/задач не виявлено – сервер обробляє запити послідовно, присвоюючи коректні ідентифікатори новим задачам. Таким чином, стабільність даних між сесіями збережено.

Методи валідації та обробка помилок. Реалізовані механізми валідації показали свою ефективність під час тестування. HTML5-атрибути (такі як *required* на полях форм) не дозволяли відправити порожні поля – браузер одразу повідомляє про необхідність заповнення, попереджаючи некоректне введення. На стороні серверу також додано базову перевірку: усі API-запити очікують валідний JSON, і у разі помилки розбору даних сервер повертає код

400 Bad Request. Перевірено обробку некоректних ситуацій – наприклад, при відправленні неповних даних сервер відповідає повідомленням про помилку, а клієнтський скрипт виводить це повідомлення користувачу через функцію showAlert() (угорі сторінки з'являється червона панель із текстом помилки). В ході тестування не трапилося критичних збоїв або винятків – додаток виявився доволі стійким до некоректних дій користувача.

Перевірка продуктивності. В рамках тестування було також оцінено швидкодію системи. Завантаження сторінки і початкових даних відбувається майже миттєво, оскільки обсяг передаваних даних невеликий (HTML-сторінка менше 100 КБ, JSON з демо-Проєктами – кілька кілобайт). Оновлення інтерфейсу після дій користувача також виконується швидко завдяки тому, що JavaScript працює локально, без перезавантаження сторінки. При тестуванні з десятками задач у Проєкті інтерфейс залишався чуйним. Засоби розробника в браузері не показали суттєвих витоків пам'яті чи помилок виконання скриптів.

Отже, тестування підтвердило, що реалізована система відповідає висунутим вимогам: він сумісний з основними браузерами, коректно працює на різних пристроях (є адаптивним) і стабільно обробляє основні сценарії використання.

3.4 Висновки до розділу 3

У третьому розділі дипломної роботи здійснено реалізацію програмної частини системи відповідно до сПроєктованих раніше вимог. Створено серверну частину, яка надає HTTP API для управління Проєктами і задачами, та клієнтську частину – інтуїтивний вебінтерфейс, через який користувач взаємодіє із системою. Система успішно перейшла зі стадії Проєкту до стадії працездатного програмного продукту.

В процесі реалізації було подолано низку труднощів. Одним із викликів стало забезпечення постійного зберігання даних між сеансами – для цього

використано простий, але ефективний підхід із JSON-файлом на сервері. Виникали питання синхронізації даних при одночасних зверненнях – їх вирішено за рахунок послідовної обробки запитів і блокування доступу до файлу на час запису (особливості GIL у Python і однопоточна природа нашого сервера спростили цю задачу). Ще одним нюансом була необхідність реалізації CORS: спочатку при розробці фронтенду окремо від сервера браузер блокував запити до API через політику безпеки, тому було додано відправку CORS-заголовків у відповідях сервера, що успішно вирішило проблему. Значна увага приділялася й адаптивності інтерфейсу – медіа-запити CSS не одразу було налаштовано правильно, але шляхом ітераційного тестування на реальних пристроях вдалося досягти коректного відображення на різних розмірах екрану.

Реалізація системи підтвердила життєздатність розробленого дизайну: всі заплановані функції (створення та редагування Проєктів і задач, відображення статистики, зміна статусів, збереження даних між сеансами) успішно працюють. Отриманий вебдодаток Project Manager Assistant є працездатним прототипом, що може слугувати основою для подальшого вдосконалення. Етап програмної реалізації став вирішальним кроком, який перетворив абстрактний Проєкт на конкретний продукт. Завдяки цьому етапу стало можливим перейти до повноцінного тестування та підготовки системи до реального використання, що і було продемонстровано в даному розділі.

ВИСНОВКИ

Таким чином, у ході виконання дипломної роботи всі поставлені завдання були успішно виконані, що дозволило досягти визначеної мети розробки. На етапі огляду предметної області було проаналізовано професію менеджера проєктів, описано основні складнощі його роботи та розглянуто наявні програмні аналоги (системи Jira, Trello, ClickUp). Проведений аналіз підтвердив доцільність створення власної системи, враховуючи специфічні потреби команд і бажання мати простіший інструмент без зайвої складності чи залежностей.

На етапі проєктування системи сформульовано технічні вимоги до функціоналу та обґрунтовано вибір ключових технологій. Зокрема, було вирішено використовувати клієнт-серверну архітектуру: серверна частина реалізована мовою Python, а клієнтська – у вигляді веб-інтерфейсу (HTML та CSS). Дані зберігаються у форматі JSON, що спростило зберігання інформації для невеликої системи і дозволило уникнути розгортання окремої бази даних. У рамках проєктування також створено необхідні діаграми (UML, діаграма розгортання) та прототип інтерфейсу користувача (макет), які забезпечили чітке уявлення про структуру і вигляд системи перед початком реалізації.

На етапі реалізації всі компоненти програмного комплексу були розроблені відповідно до проєкту. Створено серверну частину з REST API засобами Python, яка керує даними про проєкти й задачі та зберігає їх у JSON-файлі. Паралельно розроблено інтерактивну клієнтську частину – веб-застосунок з інтуїтивним інтерфейсом (HTML/CSS та JavaScript) для взаємодії менеджера проєктів із системою. Додано всі необхідні інтерфейсні елементи (дашборд з показниками, списки задач, форми для створення проєктів і задач тощо), реалізовано автоматизований запуск системи через спеціальний лаунчер, а також проведено комплексне тестування роботи застосунку на відповідність визначеним вимогам.

У результаті виконаної роботи створено повнофункціональну MVP-систему «Project Manager Assistant» для управління проєктами, яка допоможе малим і середнім командам ефективно організовувати роботу без використання складних сторонніх рішень. Усі заплановані функції успішно реалізовані та перевірені в дії, що підтверджує успішність практичної реалізації поставленої концепції. Розроблена система демонструє вдале застосування теоретичних знань на практиці та може слугувати основою для подальшого розвитку або навчальним прикладом для майбутніх проєктів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Звіт про зростання кількості робочих місць та дефіцит талантів у сфері управління Проєктами. URL : https://www.pmi.org/learning/careers/job-growth (дата звернення : 06.06.2025).

2. Jira. URL : https://www.atlassian.com/company (дата звернення : 06.06.2025).

3. Trello. URL : https://trello.com/uk/tour (дата звернення : 06.06.2025).

4. ClickUp. URL : https://clickup.com/about (дата звернення : 01.03.2024).

5. Python. URL : https://www.python.org/ (дата звернення : 06.06.2025).

6. HTML. URL : https://html.com/ (дата звернення : 06.06.2025).

7. CSS. URL : https://css.in.ua/ (дата звернення : 06.06.2025).

8. JSON. URL : https://www.json.org/json-en.html (дата звернення : 06.06.2025).

Декларація

академічної доброчесності

здобувача освіти ВСП «Економіко-правничого фахового коледжу ЗНУ»

Я, Дериведмідь_Денис_Михайлович, здобувач освіти 4 курсу, спеціальності/освітньо-професійної програми 121 Інженерія програмного забезпечення/Розробка програмного забезпечення, групи К121-21, адреса електронної пошти denisderivedmed7@gmail.com

- підтверджую, що написана мною кваліфікаційна робота на тему «Розробка системи допомоги менеджеру проєктів» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений;

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

- згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата _____ Підпис _____ Денис ДЕРИВЕДМІДЬ

Дата _____ Підпис _____ Роман КОТОВ