

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему РОЗРОБКА ІНТЕЛЕКТУАЛЬНОГО ПЛАНУВАЛЬНИКА
ПЕРСОНАЛЬНИХ ЗАВДАНЬ
DEVELOPMENT OF AN INTELLIGENT PERSONAL
TASK MANAGEMENT SYSTEM

Виконав(ла): студент(ка) 2 курсу, групи КНТ-114м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ВОЛОШИН М.Ю.

(ПРИЗВИЩЕ та ініціали)

Керівник ЗАЙКО Т.А.

(ПРИЗВИЩЕ та ініціали)

Рецензент КОРОТКИЙ О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ВОЛОШИНА Микити Юрійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка інтелектуального планувальника персональних завдань.

Development of an Intelligent Personal Task Management System

керівник проєкту (роботи) к.т.н., доцент, ЗАЙКО Тетяна Анатоліївна,

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “30” вересня 2025 року № 447

2. Строк подання студентом проєкту (роботи) 01 грудня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження.
2. Матеріали та методи. 3. Опис програми. 4. Керівництво програміста.
5. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ЗАЙКО Т.А., доцент		
Нормоконтролер	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання “ 30 ” вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	4-5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

_____ Микита ВОЛОШИН
(підпис) (Імя ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Тетяна ЗАЙКО
(підпис) (Імя ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
112 с., 7 табл., 33 рис., 3 дод., 25 джерел.

ІНТЕЛЕКТУАЛЬНИЙ ПЛАНУВАЛЬНИК, ПЕРСОНАЛЬНІ ЗАВДАННЯ, МЕТОД ПЛАНУВАННЯ ПЕРСОНАЛЬНИХ ЗАВДАНЬ, БАЗА ДАНИХ, ВЕБСАЙТ, РОЗРОБКА, ДІАГРАМА, АВТОРИЗАЦІЯ, ВЗАЄМОДІЯ.

Об'єкт дослідження – процес розробки інтелектуального планувальника персональних завдань.

Предмет дослідження – методи та програмні засоби для автоматизованого планування та управління персональними завданнями.

Мета роботи – підвищення ефективності управління персональними завданнями, зокрема скорочення часу на їх планування та підвищення продуктивності користувача, шляхом дослідження методів інтелектуального планування та розробки програмного забезпечення інтелектуального планувальника персональних завдань.

Матеріали, методи та технічні засоби: методи планування персональних завдань, фреймворк Vue.js, мова програмування JavaScript, Node.js, мова розмітки HTML, таблиця каскадних стилів CSS, база даних MongoDB, середовище розробки Visual Studio Code.

Результати. Розроблено програмне забезпечення для інтелектуального планування персональних завдань із використанням JavaScript, Vue.js, Node.js та MongoDB у середовищі Visual Studio Code.

Практична цінність роботи полягає у створенні програмного забезпечення, яке допомагає користувачам автоматизовано визначати алгоритм дій для ефективного виконання особистих завдань та підвищення продуктивності.

Висновки. Вконано проєктування та реалізацію програмного забезпечення для спрощення управління персональними завданнями; розробку інтерфейсу для користувачів; тестування функціоналу інтелектуального планувальника.

Галузь використання – особисте планування, корпоративна продуктивність, дистанційна робота, онлайн-навчання та самоорганізація.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 112 pages, 7 tables, 33 figures, 3 appendixes, 25 sources.

INTELLECTUAL PLANNER, PERSONAL TASKS, PERSONAL TASKS PLANNING METHOD, DATABASE, WEBSITE, DEVELOPMENT, DIAGRAM, AUTHORIZATION, INTERACTION.

The object of the study is the process of developing an intelligent personal task planner.

The subject of the study is methods and software for automated planning and management of personal tasks.

The purpose of the work is to increase the efficiency of personal task management, in particular, reducing the time for their planning and increasing user productivity, by studying methods of intelligent planning and developing software for an intelligent personal task planner.

Materials, methods and technical means: personal task planning methods, Vue.js framework, JavaScript programming language, Node.js, HTML markup language, CSS cascading style sheet, MongoDB database, Visual Studio Code development environment.

Results. Software for intelligent planning of personal tasks using JavaScript, Vue.js, Node.js and MongoDB in the Visual Studio Code environment has been developed.

The practical value of the work lies in creating software that helps users automatically determine the algorithm of actions for the effective completion of personal tasks and increasing productivity.

Conclusions. Software has been designed and implemented to simplify the management of personal tasks; user interface development; testing of the functionality of the intelligent planner.

Field of use - personal planning, corporate productivity, remote work, online learning and self-organization.

ЗМІСТ

	С.
Перелік скорочень та умовних познач	9
Вступ	10
1 Аналіз проблеми та постановка завдань дослідження	12
1.1 Основні поняття і актуальність планування персональних завдань.....	12
1.2 Програмні аналоги додатків планування персональних завдань.....	13
1.3 Висновки за розділом 1	18
2 Матеріали та методи	19
2.1 Вибір мови програмування та інших технологій розробки	19
2.2 Вибір середовища розробки.....	24
2.3 Метод планування персональних завдань	26
2.4 Висновки за розділом 2	30
3 Опис програми.....	32
3.1 Структура додатку планувальника персональних завдань	32
3.2 Проектування бази даних планувальника персональних завдань	33
3.3 Функціонування системи планувальника персональних завдань.....	37
3.4 Авторизація.....	37
3.5 Метод інтелектуального планування персональних завдань	41
3.6 Додавання завдань.....	50
3.7 Висновки за розділом 3	53
4 Керівництво програміста	54
4.1 Призначення й умови застосування програми.....	54
4.2 Характеристики розробленого продукту	55
4.3 Структурна схема компонентів вебсервісу	56
4.4 Висновки за розділом 4	57
5 Експлуатація, тестування та експериментальне дослідження програми	58
5.1 Опис роботи з програмою	58
5.1.1 Звернення до програми	58
5.1.2 Вхідні й вихідні дані.....	58

5.1.3 Повідомлення.....	59
5.2 Інструкція з використання	59
5.3 Експериментальне дослідження (тестування)	66
5.4 Висновки за розділом 5	67
Висновки.....	68
Перелік джерел посилань.....	70
Додаток А Технічне завдання.....	72
Додаток Б Текст програми.....	76
Додаток В Слайди презентації	104

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CSS	– Cascading Style Sheets;
DB	– Data Base;
HTML	– HyperText Markup Language;
HTTPS	– HyperText Transfer Protocol Secure;
JS	– JavaScript;
OS	– Operating System;
VS	– Visual Studio;
БД	– база даних;
СКБД	– система керування базами даних.

ВСТУП

Раціональне планування часу є ключовим аспектом ефективної організації особистої та професійної діяльності людини. У сучасному світі, де щоденно зростає обсяг інформації, завдань і обов'язків, коректний розподіл часу стає необхідною умовою досягнення високої продуктивності та балансу між роботою і відпочинком. Люди часто стикаються з труднощами у організації справ, дотриманні дедлайнів та визначенні пріоритетів, що призводить до втрати ефективності та перевантаження.

Традиційні засоби планування, такі як паперові блокноти або звичайні електронні календарі, не завжди виконують сучасні вимоги користувачів. Вони не беруть до уваги індивідуальні особливості, зміни у пріоритетах чи гнучкість робочих процесів. Тому актуальною є розробка інтелектуальних систем, здатних не лише зберігати перелік завдань, а й аналізувати їх, формувати гнучкий розклад, визначати пріоритетність і надавати рекомендації щодо ефективного розподілу часу.

Інтелектуальний планувальник персональних завдань є інструментом, який допомагає користувачеві оптимізувати виконання своїх справ, підвищити організованість та ефективність. Така система дозволяє автоматизувати процес управління завданнями, нагадувати про важливі події, коригувати розклад відповідно до змін і пропонувати користувачеві зручний спосіб взаємодії з власним планом дій.

Створення подібних програмних рішень сприяє покращенню якості життя користувачів, адже допомагає ефективно використовувати час, запобігати надмірному навантаженню та досягати поставлених цілей. Особливо важливою така система є в умовах багатозадачності, де потрібна чітка структура, адаптивність і гнучкість планування.

Таким чином, розробка інтелектуального планувальника персональних завдань є актуальною задачею, спрямованою на створення

інноваційного інструменту, що поєднує можливості автоматизації, аналізу та індивідуального підходу до організації діяльності користувача.

Для досягнення поставленої мети у кваліфікаційній роботі магістра необхідно розв'язати такі задачі:

- виконати аналіз предметної області та існуючих програмних засобів для планування завдань;
- розробити модель взаємодії користувача з інтелектуальним планувальником;
- здійснити проєктування архітектури програмного забезпечення;
- створити програмне забезпечення для інтелектуального планування персональних завдань;
- провести тестування розробленого програмного продукту та оцінити його ефективність.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Основні поняття і актуальність планування персональних завдань

Персональне завдання – завдання, що стосується лише однієї особи, не передається іншим і не є елементом командної роботи.

Планування персональних завдань є однією з важливих складових ефективного керування діяльністю людини, що забезпечує впорядкованість дій, досягнення поставлених цілей та раціональне використання часу. Це систематичний процес, який передбачає визначення цілей, формування переліку завдань, розстановку пріоритетів і здійснення контролю над їх виконанням [1].

Управління завданнями – процес, за допомогою якого організації або чи окремі особи систематично планують роботу, розставляють пріоритети та здійснюють моніторинг виконання завдань [2].

Планування особистих завдань можна визначити як процес організації власної діяльності шляхом визначення цілей, розподілу ресурсів і встановлення часових меж виконання дій. Основною метою такого планування є досягнення максимальної ефективності при мінімальних витратах часу та зусиль.

В умовах сучасного світу актуальність планування персональних завдань постійно зростає. Людина щодня стикається з великою кількістю інформації, необхідністю виконання різноманітних дій і прийняття швидких рішень. Без належного планування це призводить до хаосу, перевантаження, втрати продуктивності та зниження якості виконання обов'язків.

Ефективне планування дозволяє структурувати діяльність, визначити пріоритетність справ, встановити часові рамки та уникнути перевитрати ресурсів. Воно сприяє підвищенню самодисципліни, допомагає уникати стресових ситуацій, пов'язаних із невиконанням завдань у встановлені

терміни, а також створює умови для досягнення балансу між роботою, навчанням і відпочинком [3].

Важливим аспектом сучасного планування є його гнучкість – можливість коригувати розклад відповідно до нових умов. Це особливо важливо в умовах постійних змін, коли користувачеві потрібно оперативно реагувати на непередбачувані події або зміни в пріоритетах.

Розвиток інформаційних технологій створює нові можливості для автоматизації процесів планування. Сучасні програмні засоби здатні не лише фіксувати завдання, а й аналізувати їх, пропонувати оптимальний розподіл часу, враховувати рівень завантаженості користувача, важливість справ та інші фактори. Такі системи дозволяють зробити процес планування більш ефективним, гнучким та персоналізованим.

Отже, планування персональних завдань є невід’ємною частиною раціональної організації діяльності сучасної людини. А створення інтелектуальних програмних засобів для його реалізації є актуальним напрямом у галузі інженерії програмного забезпечення, який сприяє підвищенню ефективності, зручності та автоматизації процесу управління особистими справами.

1.2 Програмні аналоги додатків планування персональних завдань

У сучасному світі ефективне управління часом та персональними завданнями є одним із ключових чинників продуктивності людини. З розвитком цифрових технологій з’явилося багато програмних рішень, що допомагають користувачам планувати власну діяльність, контролювати виконання завдань, встановлювати пріоритети та відстежувати прогрес. До найпопулярніших серед них належать Todoist, Trello, Microsoft To Do та Notion.

Todoist – це одна з найвідоміших систем управління завданнями, яка поєднує простоту інтерфейсу з розширеними можливостями планування [4]. Додаток дозволяє створювати завдання, підзавдання, встановлювати пріоритети, нагадування та дедлайни. Завдяки підтримці міток і фільтрів користувач може швидко орієнтуватися у великій кількості справ. Todoist також інтегрується з календарями Google, Outlook та іншими сервісами, що робить його універсальним інструментом для роботи й особистого життя.

На рисунку 1.1 продемонстровано інтерфейс Todoist.

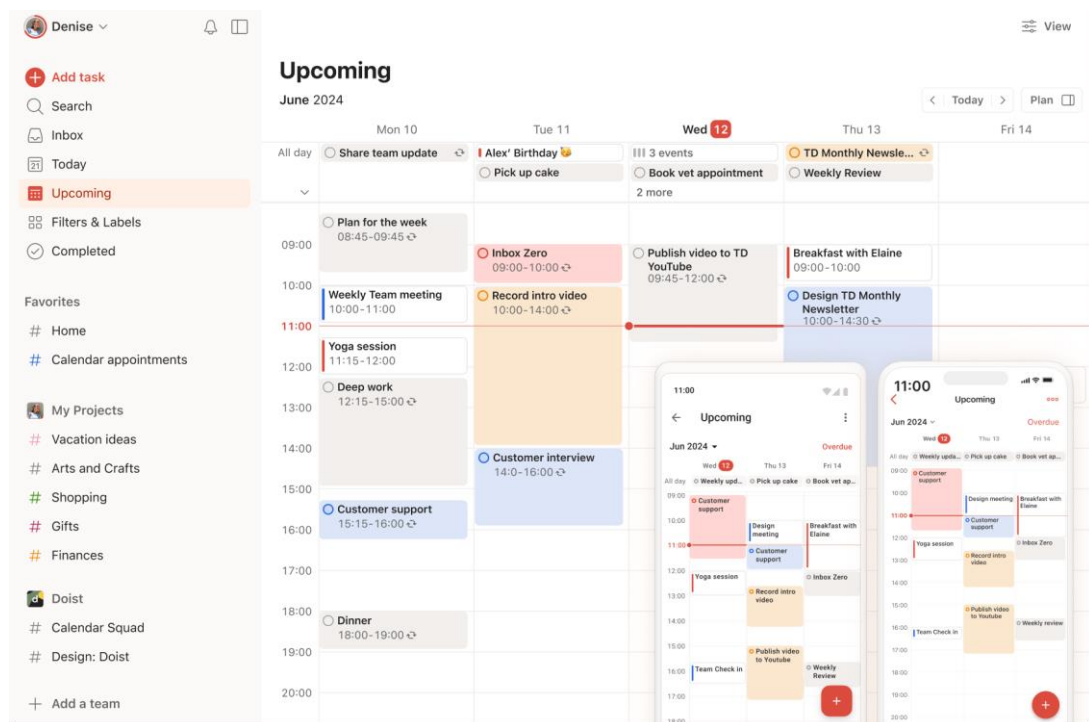


Рисунок 1.1 – Інтерфейс системи управління завданнями Todoist [4]

Trello – це візуальний інструмент керування завданнями, який базується на методології Kanban [5]. Його головна особливість полягає у використанні панелей, списків і карток, які дозволяють користувачу наочно бачити процес виконання завдань. Система підтримує спільну роботу в командах, коментарі, прикріплення файлів та інтеграцію з сервісами Google Drive, Slack, Jira тощо. Trello особливо зручний для проєктного планування, коли потрібно відстежувати етапи виконання роботи та її стан у режимі реального часу. На рисунку 1.2 продемонстровано інтерфейс Trello.

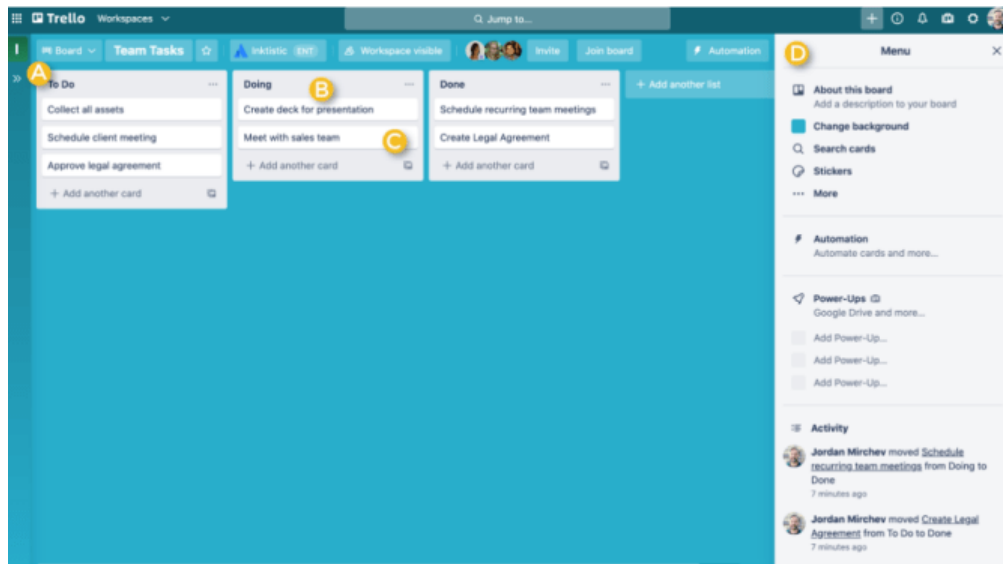


Рисунок 1.2 – Інтерфейс Trello [5]

Microsoft To Do – це безкоштовний додаток від корпорації Microsoft, орієнтований на особисте та командне планування [6]. Головний акцент приділяється зручності користування та синхронізації між пристроями за допомогою облікового запису Microsoft. Додаток дозволяє створювати списки завдань, встановлювати дедлайни, додавати нотатки та пріоритети. Microsoft To Do інтегрується з Outlook, Teams та іншими корпоративними інструментами, що робить його корисним для працівників офісів і студентів.

На рисунку 1.3 продемонстровано інтерфейс Microsoft To Do.

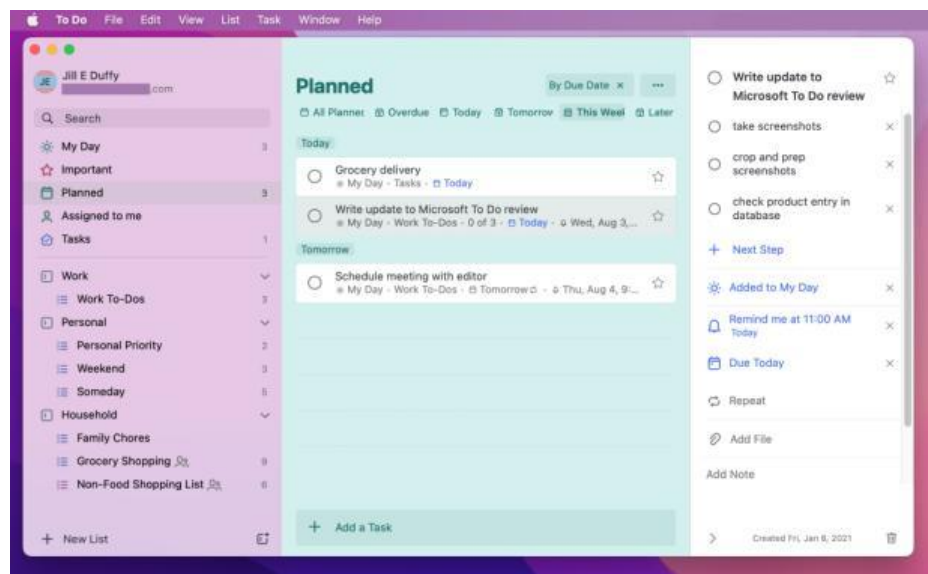


Рисунок 1.3 – Інтерфейс Microsoft To Do [6]

Notion – це багатозначна платформа для організації інформації, яка поєднує в собі функції нотатника, менеджера завдань, бази даних і системи спільної роботи [7]. Користувач може створювати власні сторінки, таблиці, бази даних і зв'язки між ними, що дозволяє гнучко адаптувати Notion під особисті потреби. Додаток активно використовується для ведення персональних планів, навчальних нотаток, а також для управління великими проєктами.

На рисунку 1.4 продемонстровано інтерфейс платформи Notion.

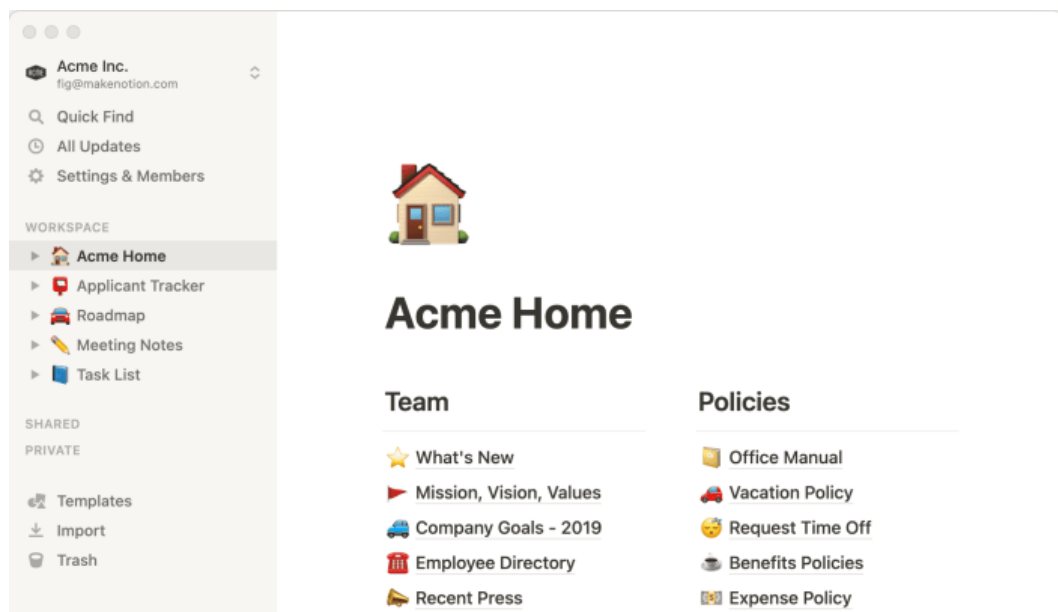


Рисунок 1.4 – Інтерфейс платформи Notion [7]

Для порівняльного аналізу особливостей розглянутих систем сформовано таблицю 1.1.

Таблиця 1.1 – Порівняльна характеристика популярних планувальників персональних завдань

Характеристика	Todoist	Trello	Microsoft To Do	Notion
Тип платформи	Хмарна	Хмарна	Хмарна	Хмарна / локальна
Рівень кастомізації	Середній	Високий	Низький	Дуже високий

Продовження таблиці 1.1

Характеристика	Todoist	Trello	Microsoft To Do	Notion
Відстеження прогресу	Так, за допомогою нагадувань та фільтрів	Так, візуально через картки та статуси	Так, базові списки та завершені завдання	Так, через бази даних і таблиці
Наявність шаблонів	Так, обмежена кількість	Так, багато шаблонів для проєктів	Обмежена	Дуже велика кількість
Аналітика та статистика	Так, базова продуктивність	Обмежена	Немає	Так, можна налаштувати через бази даних
Безпека та конфіденційність	Стандартна хмарна безпека	Стандартна хмарна безпека	Інтеграція з обліковим записом Microsoft	Висока, можна локально зберігати дані
Підтримка спільної роботи	Так	Так	Частково	Так
Інтеграції	Google Calendar, Slack, Outlook	Slack, Jira, Google Drive	Outlook, Teams	Google Calendar, Slack, GitHub

Аналіз наведених аналогів показує, що існуючі програмні рішення для планування завдань мають широкий функціонал, однак більшість із них зосереджені на зовнішньому контролі виконання задач, а не на інтелектуальній підтримці користувача. Вони не враховують індивідуальні особливості, поведінкові шаблони або пріоритети користувача, не пропонують автоматичне формування оптимального графіка виконання завдань.

Отже, актуальність розробки інтелектуального планувальника персональних завдань полягає у створенні системи, здатної не лише організовувати діяльність користувача, але й аналізувати його поведінку,

прогнозувати навантаження, автоматично формувати оптимальний розклад та адаптуватися до змін у пріоритетах.

1.3 Висновки за розділом 1

У результаті проведеного аналізу з'ясовано, що сучасний ринок програмного забезпечення пропонує широкий спектр додатків для планування персональних завдань, таких як Todoist, Trello, Microsoft To Do, Notion тощо. Вони забезпечують базові можливості організації часу, встановлення цілей, пріоритезації завдань і відстеження прогресу користувача.

Однак, попри значну кількість існуючих рішень, більшість з них не мають інтелектуальних механізмів підтримки користувача, які б дозволяли автоматично адаптувати план до змін пріоритетності, рівня завантаженості чи ефективності виконання завдань. Такі системи часто зосереджені лише на візуалізації процесу чи зберіганні інформації, не надаючи гнучких інструментів прогнозування або персоналізованих рекомендацій.

На основі проведеного аналізу визначено основні функціональні вимоги до майбутнього програмного забезпечення:

- створення, редагування та відстеження завдань і подій;
- встановлення пріоритетів і дедлайнів;
- позначення завдань як виконаних;
- формування графіків продуктивності за категоріями завдань;
- аналіз показників продуктивності користувача з метою формування інтелектуальних рекомендацій щодо оптимального розподілу часу, визначення періодів найвищої активності та побудови персоналізованих графіків роботи.

2 МАТЕРІАЛИ ТА МЕТОДИ

2.1 Вибір мови програмування та інших технологій розробки

Вибір мови програмування та відповідних технологій є ключовим кроком у процесі розробки програмного забезпечення. У цьому проєкті для розробки використовувалися такі мови програмування, як Vue.js, JavaScript та Node.js [8].

Node.js було обрано основною мовою для серверної частини проєкту. Це рішення зумовлене популярністю Node.js у сфері веброзробки, а також його здатністю ефективно обробляти велику кількість запитів завдяки своїй неблокуючій моделі вводу/виводу [9]. Важливою перевагою є те, що Node.js побудований на JavaScript, що дозволяє використовувати одну мову для написання як серверної, так і клієнтської частин проєкту, що значно спрощує процес розробки. Крім того, Node.js підтримує широкий спектр модулів та бібліотек, що полегшують створення вебдодатків, а також інтеграцію з базами даних або іншими сервісами [10].

Vue.js – це JavaScript-фреймворк, який використовується для створення клієнтської частини вебзастосунку в рамках цього проєкту. Вибір на користь Vue.js був зумовлений його простотою та гнучкістю, що надають можливість розробляти інтерактивний та динамічний інтерфейс користувача [11]. Цей фреймворк дозволяє ефективно керувати станом компонентів та забезпечувати безперебійну взаємодію з середовищем користувача без необхідності перезавантаження вебсторінок. Додатковою перевагою стала простота реалізації таких функцій, як перевірка даних у формах, інтерактивні елементи та асинхронні запити до сервера, що значно підвищило функціональність застосунку [12].

У процесах розробки також використовувалися передові вебтехнології, зокрема HTML5 та CSS3 [13]. HTML5, як сучасна версія мови розмітки HTML, приніс значний набір інноваційних функцій. Так, він забезпечує підтримку вбудованих аудіо- та відеоелементів, роботу з

графікою за допомогою canvas, доступ до апаратних ресурсів пристрою тощо, що значно розширює можливості вебзастосунків [14].

CSS3, у свою чергу, є розширеною версією Cascading Style Sheets, яка додала нові інструменти для розробників [15]. Серед них – технології анімації, плавні переходи, адаптивні макети (Flexbox та Grid), а також покращене керування візуальними ефектами, такими як тіні або багат шарові фонові зображення. Використання цих технологій дозволило створити естетичний, функціональний та адаптивний інтерфейс, що відповідає вимогам сучасної веброзробки [16].

Одним з аналогів фреймворку Vue.js є React.js. Обидва інструменти спрямовані на розробку інтерактивних вебдодатків та мають свої особливості. Vue.js вирізняється простотою та високою гнучкістю, тоді як React надає розробникам доступ до потужної екосистеми та великого набору бібліотек управління станами. Обидва фреймворки можуть бути оптимальними для різних проєктів залежно від конкретних вимог та цілей.

Vue.js – це сучасний та гнучкий фреймворк для розробки користувацьких інтерфейсів, який здобув популярність завдяки низькому порогу входу та високій продуктивності. Однією з головних причин його широкого використання є не лише продуманість основної бібліотеки, але й наявність багатьох екосистем інструментів, які значно розширюють можливості програміста. Vue.js характеризується дружньою, зрозумілою документацією, а його структура відрізняється логікою та послідовністю. Навіть для новачка у фронтенд-розробці ознайомлення з Vue набагато простіше завдяки відсутності зайвої складності чи надмірно громіздких структур. Тому цей фреймворк часто рекомендують тим, хто тільки-но починає свій шлях програмування [17].

Vue.js можна легко інтегрувати в існуючий проєкт, додавши його через простий тег `<script>` без необхідності реорганізації архітектури. У той же час, цей фреймворк підтримує розробку складних SPA-додатків за допомогою інструментів Vue CLI або Vite. Така універсальність дозволяє

ефективно використовувати Vue як у невеликих проєктах, так і великомасштабних системах. Реактивна система Vue автоматично відстежує зміни даних та оновлює інтерфейс без необхідності вручну вносити зміни до DOM. Розробники звільняються від необхідності синхронізувати стан програми та інтерфейсу вручну, оскільки цей процес автоматизовано. Такий підхід дозволяє знизити обсяг коду та зробити його більш зрозумілим.

Vue підтримує розробку інтерфейсів за допомогою модульного підходу, що дозволяє розбити систему на невеликі ізольовані компоненти, такі як кнопки, форми, карти або модальні вікна. Це сприяє повторному використанню компонентів у різних частинах проєкту, що сприяє масштабованості, покращує читабельність та забезпечує зручність обслуговування коду. Використовуючи механізм віртуального DOM Vue, визначає мінімальні зміни, які необхідно внести в інтерфейс для забезпечення оновлення сторінок. Такий підхід допомагає скоротити час рендерингу та підвищити ефективність взаємодії з користувачем, що особливо важливо для великих проєктів, де оптимізація є ключовим аспектом.

Vue має багату екосистему як офіційних, так і сторонніх інструментів, включаючи Vue Router, Pinia/Vuex, Nuxt.js та UI-фреймворки. Велика та активна спільнота розробників регулярно створює нові бібліотеки, пише навчальні посібники та дає технічні консультації, що полегшує вирішення більшості проблем. Завдяки чітко структурованій архітектурі, інструментам керування станом та маршрутизації, а також можливості використання типізації (наприклад, TypeScript), Vue підходить не лише для невеликих сайтів, але й для розробки високонавантажених платформ та корпоративних систем.

Node.js – це платформа, яка надає можливість виконання JavaScript на стороні сервера [18]. З моменту своєї появи вона здобула значну популярність серед бекенд-розробників завдяки низці функцій, які заслуговують на увагу. Node.js базується на механізмі циклу подій та

неблокуючому вводі-виводі (неблокуючому вводі-виводі). Такий підхід дозволяє серверу ефективно обслуговувати числові запити одночасно, не чекаючи завершення операцій. Ця особливість робить платформу надзвичайно корисною для високонавантажених додатків.

Єдина мова програмування – JavaScript. Node.js надає можливість використовувати одну мову як для фронтенд-, так і для бекенд-розробки, що сприяє прискоренню процесу розробки, оптимізації внутрішньої комунікації в команді, скороченню кривої навчання для нових розробників. Ця універсальність є однією з ключових переваг платформи.

Node Package Manager (npm) надає доступ до величезної кількості готових модулів, які охоплюють широкий спектр завдань: від інтеграції з базами даних та авторизації до створення вебсерверів та аналізу даних. Використання npm дозволяє уникнути дублювання зусиль, оскільки більшість стандартних завдань вже вирішені іншими розробниками.

Node.js підтримує як горизонтальну, так і вертикальну масштабованість:

- горизонтальна масштабованість реалізується шляхом додавання нових серверів для розподілу навантажень;
- вертикальна – шляхом збільшення обчислювальних ресурсів однієї системи.

Завдяки цим властивостям платформа підходить як для невеликих вебсайтів, так і для складних багатокомпонентних систем.

Node.js ідеально підходить для завдань, що потребують обробки даних у режимі реального часу, таких як чати, сповіщення, спільне редагування документів або онлайн-таблиці. Це досягається за допомогою WebSocket та моделі подій [19].

Платформа широко використовується для розробки REST та GraphQL API завдяки своїй простоті, продуктивності та легкій масштабованості. Такі API легко інтегруються з мобільними додатками, сторонніми сервісами та системами з мікросервісною архітектурою.

Легка структура та висока швидкість Node.js роблять його оптимальним вибором для побудови мікросервісів, де кожна невелика частина системи виконує своє завдання незалежно від інших компонентів.

Узгодження Node.js з Vue.js створює потужний технологічний стек для веброзробки. Vue.js забезпечує гнучкість у створенні сучасних користувацьких інтерфейсів, тоді як Node.js формує стабільну та продуктивну основу на стороні сервера. У взаємодії ці технології можуть розробляти функціональні, масштабовані та високоякісні вебдодатки: від простих односторінкових додатків (SPA) до великих багатофункціональних систем. Обґрунтування вибору мови програмування для розробки програмного забезпечення інтелектуального планувальника персональних завдань наведено в таблиці 2.1.

Таблиця 2.1 – Обґрунтування вибору мови програмування

Критерій	Vue.js	React.js
Підхід до побудови компонентів	Оптимізований, шаблони + JS	Повністю на JavaScript / JSX
Крива входження	Низька	Середня
Офіційні інструменти	Багато (Router, Pinia, CLI)	Менше, часто сторонні
Документація	Дуже структурована	Потужна, але більш технічна
Інтеграція в існуючі проєкти	Дуже проста	Середня
Робота з анімаціями	Зручні вбудовані можливості	Переважно сторонні бібліотеки
Розмір ядра	Малий	Середній
Гнучкість налаштувань проєкту	Висока	Дуже висока
Популярність у великих корпораціях	Середня	Дуже висока
Підтримка TypeScript	Відмінна (Vue 3)	Дуже добра

Для розробки програмного забезпечення інтелектуального планувальника персональних завдань було обрано технології Vue.js та Node.js. Фреймворк Vue.js використовується для реалізації клієнтської частини програми, тоді як серверна частина створена за допомогою Node.js. Кожна з цих технологій має свої унікальні переваги та дозволяє вирішувати широкий спектр завдань.

Поєднання Vue.js та Node.js сприяло реалізації повнофункціонального, інтерактивного вебзастосунку, який відповідає потребам користувачів. Вибираючи мову програмування або платформу для розробки вебзастосунків, важливо враховувати специфіку проєкту, а також технічний досвід та компетенцію розробника. У багатьох випадках саме ця комбінована технологія демонструє високу ефективність для виконання різних завдань, що реалізуються в процесі створення програмних продуктів.

2.2 Вибір середовища розробки

Visual Studio Code – це легкий, але водночас дуже функціональний текстовий редактор, створений корпорацією Microsoft. Головною метою цього інструменту є підтримка роботи з ефективними мовами програмування, а також надання широкого спектру функцій, що сприяють його популярності серед розробників [20].

Однією з визначальних характеристик Visual Studio Code є його кросплатформність. Програма сумісна з ключовими операційними системами, включаючи Windows, macOS та Linux, що робить її доступною для більшості користувачів, незалежно від типу обладнання. Через це розробники не можуть працювати з одним інструментом, незалежно від специфіки операційного середовища [21].

Редактор є потужним інструментом, що забезпечує підтримку роботи з широким спектром мов програмування, що робить його надзвичайно універсальним як для новачків, так і для досвідчених фахівців у сфері

програмування. Visual Studio Code надає інтегровану підтримку для програмування багатьох популярних мов, а за допомогою системного розширення цей перелік можна розширити практично необмежено [22].

Однією з ключових переваг редактора є його інтелектуальні функції, серед яких виділяється IntelliSense. Цей інструмент надає можливість автодоповнення коду, а також виправлення та виправлення синтаксичних помилок у реальному часі. Завдяки цій значній швидкості написання коду, знижується ризик виникнення типів помилок, а процес розробки стає більш ефективним. Крім того, система підсвічування синтаксису сприяє кращій читабельності коду, що має особливе значення для розробників, які працюють над масштабними проєктами [23].

Іншою важливою функцією є інтеграція з Git – провідною системою контролю версій. Це забезпечує можливість одночасно в редакції переглядати зміни в коді, створювати комісії, виконувати перемикання між головками проєкту та розв'язувати конфлікти. Така інтеграція суттєво полегшує процес співпраці в командних проєктах і забезпечує більш ефективний контроль над ходом розробки.

Visual Studio Code слугує універсальним інструментом, який задовольняє потреби різних типів розробників: від тих, хто працює над невеликими особистими проєктами, до команди, що створює масштабні корпоративні системи. Інтуїтивно зрозумілий інтерфейс полегшує освоєння редактора для початківців, тоді як досвідчені фахівці гідно оцінюють його розширені функціональні можливості та інструменти, спрямовані на оптимізацію робочих процесів.

Обґрунтування середовища розробки для створення програмного забезпечення інтелектуального планувальника персональних завдань наведено в таблиці 2.2.

Таблиця 2.2 - Обґрунтування середовища розробки для створення програмного забезпечення інтелектуального планувальника персональних завдань

Характеристика	VS Code	Sublime Text
Вартість	Безкоштовний	Платна ліцензія (є пробна)
Швидкий старт без налаштувань	–	+
Споживання оперативної пам'яті	Вище	Дуже низьке
Частота оновлень	Дуже висока	Помірна
Робота з великими файлами	Середня	Дуже швидка
Підтримка Git	Вбудована	Через плагіни
Інструменти для веброботи	Максимально розвинені	Базові
Персоналізація інтерфейсу	Висока	Середня
Пошук по проєкту	Швидкий	Дуже швидкий
Автоматичні форматтери (Prettier тощо)	Легко інтегруються	Через сторонні плагіни

Таким чином, Visual Studio Code поєднує в собі легкість, адаптивність і високий рівень продуктивності, демонструючи переваги над багатьма іншими текстовими редакторами та інтегрованими середовищами розробки за низкою ключових параметрів. Цей інструмент забезпечує оптимальне співвідношення між функціональністю та швидкістю роботи, надаючи розробникам необхідний набір можливостей для ефективного виконання завдань, незалежно від обраної мови програмування чи характеру проєкту.

2.3 Метод планування персональних завдань

Метод, що використовується при створенні інтелектуального персонального планувальника завдань, базується на аналізі поведінкових та продуктивних даних користувача. Головною метою є впровадження системи, яка автоматично збирає інформацію про виконання завдань, оцінює їх

класифікацію, визначає проблемні області та надає індивідуальні рекомендації щодо підвищення ефективності управління часом.

Такий підхід допоможе користувачеві приймати обґрунтовані рішення щодо своєї продуктивності, оптимізувати робоче навантаження та розробляти реалістичні стратегії планування. Система не лише відстежує виконані або пропущені завдання, але й виконує свої дії через невдачі та знаходить шляхи покращення власної роботи.

Ключові елементи методу.

Усі завдання розподіляються за чіткими категоріями, такими як «Робота», «Особисте», «Навчання» або «Інше».

Цей підхід дозволяє:

- оцінювати рівень продуктивності в кожній окремій сфері;
- виявляти наявні дисбаланси;
- аналізувати розподіл навантаження між різними аспектами життя.

Завдяки категоризації можна, наприклад, побачити, що в категорії Робота виконано всі заплановані завдання, а в категорії «Особисте» залишається значна кількість невиконаних справ. Це свідчить про нерівномірне використання часу та ресурсів.

Система автоматично фіксує:

- кількість створених завдань;
- відсоток виконаних завдань;
- пропущені завдання;
- вибір варіантів (низький, середній, високий);
- графіки виконання на тиждень або інший вибраний період;
- рівень продуктивності в кожній категорії.

Зібрані дані об'єднуються та готуються до подальшого аналізу, який включає візуалізацію у вигляді графіків та статистичних карток.

На основі зібраних даних система генерує ключові показники:

- загальний рівень виконання завдань;
- продуктивність у кожній категорії;

- стабільність виконання завдань по днях;
- кількість прострочених завдань;
- баланс між пріоритетами.

Наприклад, якщо загальний відсоток виконаних завдань становить 78%, але в категорії «Особисте» цей показник дорівнює 0%, система може запропонувати звернути більше уваги на завдання цієї категорії.

Для створення аналітики використовуються такі методи:

- порівняння виконаних та невиконаних завдань;
- аналіз змін динаміки за вибраний період (тиждень, два тижні тощо);
- визначення найуспішніших та найменш продуктивних днів.

Система аналізує закономірності та генерує персоналізовані поради.

Наприклад:

- якщо продуктивність висока, забезпечити підтримку поточного темпу, виконуючи приблизно 0,5 завдання на день;
- якщо є прострочені завдання, система рекомендує виділити 30–60 хвилин на їх перегляд, оновлення або делегування;
- якщо спостерігається помітне зниження активності, є рекомендації почати з невеликих змін, наприклад, додати 60–90 хвилин зосередженої роботи на день та організувати щотижневий огляд.

Ці рекомендації працюють як віртуальний помічник з управління часом, сприяючи ефективному плануванню та розподілу завдань.

Визначити ефективність використання такого планувальника за допомогою порівняльного аналізу:

До впровадження системи:

- кількість виконаних та пропущених завдань;
- рівень загальної дисципліни користувачів;
- відсоток незавершених завдань з високим пріоритетом.

Після впровадження:

- зміна відсотка виконання завдань;
- збільшення продуктивності в певних категоріях;

- більш рівномірний розподіл робочого навантаження;
- зменшення деяких завдань з простроченими термінами.

Такий підхід дозволяє оцінити, чи дійсно система вплинула на поведінку користувача та програмного забезпечення, і які аспекти були покращені.

Метод інтелектуального планування особистих завдань, що базується на аналізі категорій, пріоритетів, продуктивності та динаміки виконання, дозволяє:

- глибше аналізувати власну ефективність;
- розробляти більш обґрунтовані стратегії планування;
- отримувати об'єктивний зворотний зв'язок у вигляді персоналізованих рекомендацій;
- регулярно коригувати рівень робочого навантаження;
- підтримувати баланс між професійними, навчальними та особистими завданнями.

Впровадження такого підходу допоможе підвищити рівень самоорганізації, сформувати корисні робочі звички та покращити загальну ефективність.

Для створення та збірки проєкту було використано Vite – сучасний інструмент, який характеризується надзвичайно швидкою ініціалізацією та зручним процесом розробки. Його популярність пояснюється практично миттєвою компіляцією та ефективною системою гарячої заміни модулів, що значно покращує роботу над інтерфейсом.

Однією з головних переваг Vite є використання нативних ES-модулів. У режимі розробки цей інструмент запускає легкий сервер, який компілює лише ті частини коду, які потрібні на даний момент. Це дозволяє уникнути повної перекомпіляції, що значно скорочує час запуску та оновлення.

Додатково, Vite використовує esbuild для попередньої збірки залежностей. Esbuild – це високошвидкісний пакувальник на базі Go, який значно складніший за продуктивністю порівняно з традиційними

інструментами. Хоча Webpack залишається одним із найбільш широко використовуваних пакувальників у веброзробці, він працює повноцінніше та часто вимагає складнішого налаштування. Порівняльну інформацію про Vite та Webpack можна знайти в таблиці 2.3.

Таблиця 2.3 – Порівняння Vite та Webpack

Критерій	Vite	Webpack
Швидкість розробки	Дуже висока швидкість завдяки динамічному збиранню лише потрібних модулів	Може працювати повільніше, особливо у великих проєктах
Налаштування	Мінімальна конфігурація, інструмент готовий до роботи «з коробки»	Вимагає детальної конфігурації, що ускладнює старт
Спільнота	Спільнота активно зростає, але поки менша	Велика, усталена та дуже активна спільнота
Підтримка ES-модулів	Повністю вбудована підтримка, оптимізована для сучасних стандартів	Підтримує, але потребує додаткових налаштувань
Плагіни	Підтримуються, але більшість функцій доступні без них	Гнучкість через плагіни та ладери
Сфери використання	Малий та середній проєкт, сучасні фреймворки	Найширший спектр проєктів, підходить для масштабних рішень
Конфігурація	Дуже проста, мінімалістична	Як правило, складна та багатоетапна

2.4 Висновки за розділом 2

При розробці програмного забезпечення системи управління освітнім плануванням було обрано технологічний стек, що включає Vue.js для клієнтської частини та Node.js для серверної логіки. Цей вибір зумовлений їхньою сумісністю, високою продуктивністю та гнучкістю. Використання Node.js дозволило організувати ефективну обробку запитів та забезпечити

єдину мовну базу на основі JavaScript, що значно спростило структуру проєкту та прискорило процес розробки. Зі свого боку, Vue.js надає можливість реалізувати сучасний динамічний інтерфейс зі зрозумілою компонентною архітектурою та високою продуктивністю.

Серед аргументів на користь використання Vue.js він відповідає своєму основному конкуренту React.js, оскільки вирішальними були проста інтеграція, доступна документація, низький поріг входу та наявність потужних інструментів. HTML5 та CSS3 гармонійно доповнювали технологічну базу, забезпечуючи створення адаптивного та привабливого інтерфейсу для користувачів.

На особливу увагу заслуговує використання Vite як інструменту для розробки та складання клієнтської частини. Цей інструмент значно підвищує ефективність роботи завдяки швидкій компіляції, швидкому запуску проєкту та підтримці гарячої заміни модулів. Використання нативних ES-модулів у роботі Vite дозволило уникнути зайвої компіляції коду, що виявилось особливо цінним під час активного редагування інтерфейсу. Крім того, використання надшвидкого механізму esbuild для попередньої обробки залежностей забезпечило стабільну продуктивність навіть зі зростанням масштабу проєкту. Порівняно з традиційними асемблерами, такими як Webpack, Vite пропонував набагато комфортніші умови для роботи та запитів на налаштування.

Таким чином, поєднання Vue.js, Node.js та Vite виявилось оптимальним рішенням для створення масштабованого, зручного та високопродуктивного вебресурсу. Обраний технологічний стек відповідав усім вимогам проєкту, забезпечуючи ефективність розробки та перспективи подальшого вдосконалення системи.

3 ОПИС ПРОГРАМИ

3.1 Структура додатку планувальника персональних завдань

Структура програмного забезпечення інтелектуального планувальника персональних завдань наведено на рис. 3.1.

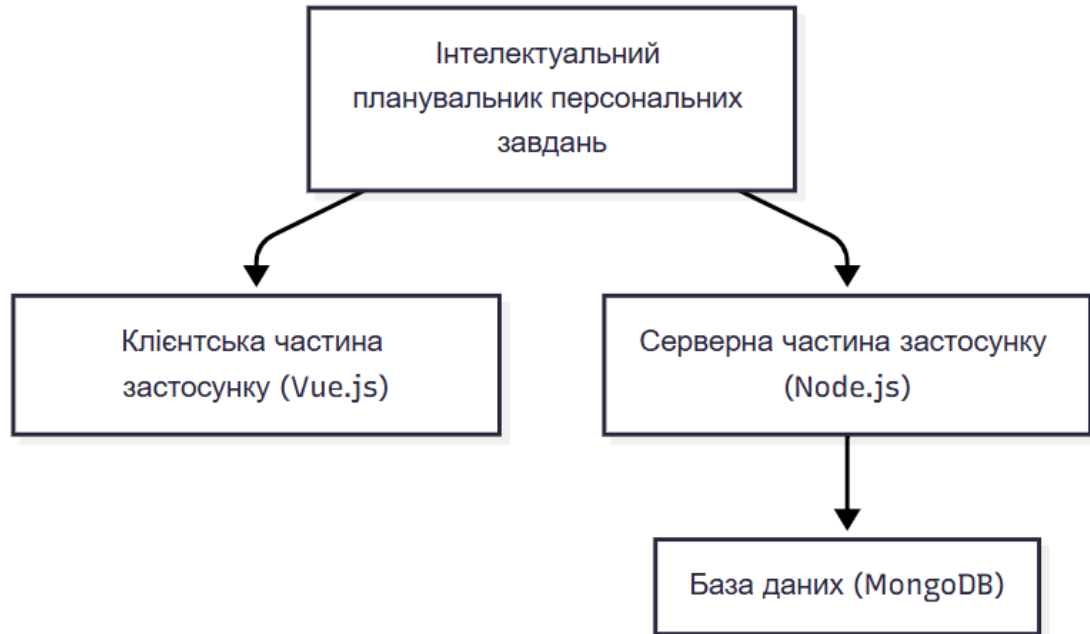


Рисунок 3.1 – Структура додатку інтелектуального планувальника персональних завдань

На рисунку показано основні компоненти програмного забезпечення системи персонального планування завдань:

- клієнтська частина програми;
- серверна частина;
- база даних.

Клієнтська частина складається з інтерфейсу користувача, реалізованого за допомогою фреймворку Vue.js, а також технологій HTML5 та CSS3. Цей модуль відповідає за відображення списку завдань, аналітику, продуктивність графіків та інші елементи взаємодії з користувачем.

Серверна частина реалізована на базі Node.js, що забезпечує обробку запитів, управління логікою планувальника, обробку статистики та генерацію рекомендацій. Доступ до вебсервісу створюється через Інтернет за допомогою протоколу HTTPS, що гарантує безпечний обмін даними між клієнтом та сервером.

База даних MongoDB використовується для зберігання всієї інформаційної системи – завдань, статусів, категорій, пріоритетів, історії виконання. Вона була обрана завдяки високій продуктивності, надійності та простоті використання з документоорієнтованою структурою даних, що ідеально підходить для динамічних вебзастосунків.

3.2 Проєктування бази даних планувальника персональних завдань

У процесі роботи з інтелектуальною системою персонального планувальника завдань виникає потреба зберігати значну кількість інформації – дані про завдання, терміни, категорії, статуси виконання, користувачів, файли та статистику продуктивності. Для коректної роботи всіх сервісних модулів необхідно забезпечити надійність та гнучке управління системою баз даних.

У цій розробці використовувалася база даних MongoDB (таблиця 3.1). MongoDB – це сучасна документоорієнтована NoSQL база даних, створена для зберігання великих обсягів динамічної та неструктурованої інформації, що особливо актуально для вебдодатків та продуктивності системи [24].

До основних переваг MongoDB належать:

- в межах однієї колекції можна зберігати документи різної структури, що дозволяє легко доповнювати та змінювати модель даних без складних міграцій;

- MongoDB підтримує горизонтальне масштабування, завдяки чому дані можна розподіляти між багатьма серверами для підвищення швидкості та надійності;

- система використовує індексацію, кешування та потужний механізм агрегації для забезпечення швидкого виконання запитів та аналітичних операцій;

- дані зберігаються у форматі BSON (розширений JSON), що дозволяє моделювати складні об'єкти з вкладеними структурами – ідеально підходить для завдань, що мають підкатегорії, файли, пріоритети тощо.

MongoDB добре працює в системах, де важливі гнучкість моделі, масштабованість та можливість роботи з великими наборами даних [25]. Він підходить для:

- вебдодатків та мобільних додатків;
- систем аналітичної обробки інформації;
- онлайн-сервісів зі зміненою структурою збережених даних.

Популярність MongoDB зумовлена його здатністю працювати з великими, неоднорідними та швидкозмінними наборами даних, зберігаючи при цьому високу продуктивність та простоту адміністрування.

Таблиця 3.1 – Порівняння MongoDB та реляційних SQL-баз

Характеристика	MongoDB (NoSQL)	SQL (Реляційні БД)
Модель даних	Документна	Таблична
Схема	Безсхемна	Фіксована
Запити	MQL (JSON-подібний формат)	SQL
Інтеграція з великими даними	Висока	Орієнтована на структуровані дані
Типи даних	Гнучкі, включно з вкладеними структурами	Обмежені стандартними типами
Робота зі зв'язками	Через вкладення або посилення (без JOIN)	Розвинені JOIN-запити
Відновлення після збоїв	Просте налаштування реплікації	Реплікація є, але складніша в налаштуванні

MongoDB є оптимальним вибором для систем, що потребують гнучкої структури та швидкого масштабування, особливо у вебзастосунках та платформах з динамічною моделлю даних.

Структура даних компонентів системи зображена у таблиці 3.2-3.3.

Таблиця 3.2 – Структура даних таблиці «Tasks» (Завдання)

Назва ключа	Тип значення	Опис
_id	ObjectId	Унікальний ідентифікатор завдання
title	String	Назва завдання; обов'язкове поле
description	String	Опис або деталі завдання
priority	String	Пріоритет виконання; за замовчуванням – medium
dueDate	Date	Дата дедлайну
category	String	Категорія завдання; за замовчуванням – work
completed	Boolean	Статус виконання (виконано/не виконано); за замовчуванням – false
completedAt	Date	Дата та час завершення завдання (null, якщо не виконано)
userId	ObjectId (ref: User)	Посилання на користувача, якому належить завдання; обов'язкове поле
createdAt	Date	Дата створення запису; генерується автоматично
updatedAt	Date	Дата останнього оновлення запису; змінюється автоматично перед збереженням

Таблиця 3.3 – Структура даних таблиці «User» (Користувач)

Назва ключа	Тип значення	Опис
_id	ObjectId	Унікальний ідентифікатор користувача
username	String	Ім'я користувача; обов'язкове та унікальне поле
email	String	Електронна пошта; обов'язкове та унікальне поле
password	String	Пароль у зашифрованому вигляді; обов'язкове поле
createdAt	Date	Дата реєстрації користувача; встановлюється автоматично

На рисунку 3.2 зображено взаємозв'язок між таблицями системи інтелектуального планувальника персональних завдань.

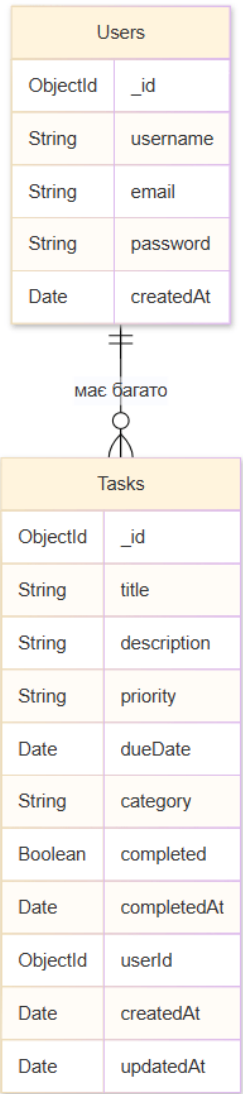


Рисунок 3.2 – Взаємозв'язок між таблицями

За допомогою рисунку можна побачити, що у таблиці «Users» та таблицею «Tasks» встановлений взаємозв'язок типу «один до багатьох». Це означає, що один користувач може мати декілька завдань, але кожне завдання належить лише одному конкретному користувачеві. Такий зв'язок реалізовано за допомогою поля `userId` у таблиці «Tasks», яке містить посилання на унікальний ідентифікатор користувача.

3.3 Функціонування системи планувальника персональних завдань

Схема функціонування додатку інтелектуального планувальника персональних завдань зображено на рис. 3.3.

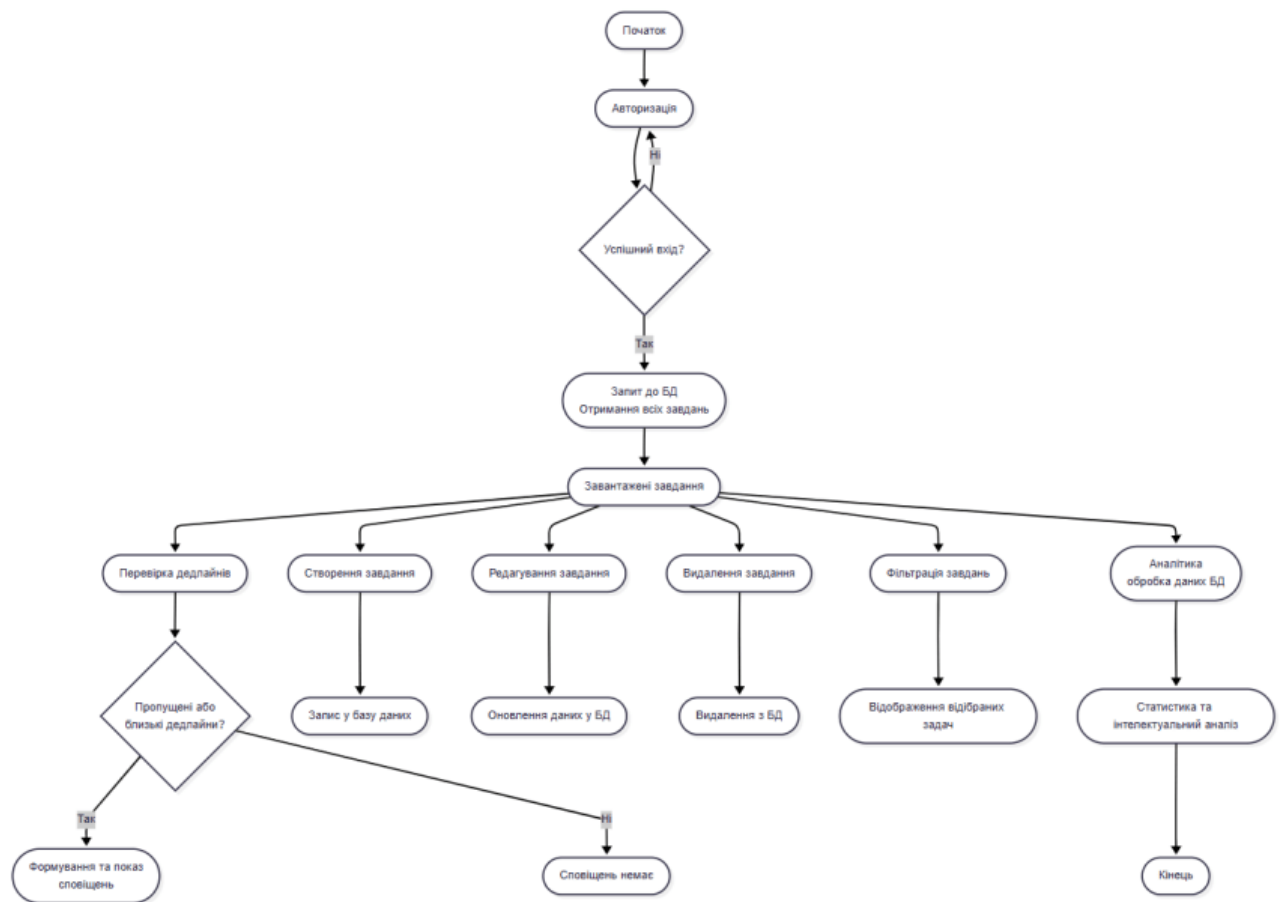


Рисунок 3.3 – Схема функціонування додатку

3.4 Авторизація

Щоб отримати доступ до інтелектуального персонального планувальника завдань, необхідно пройти процедуру авторизації. Новим користувачам спочатку пропонується створити власний обліковий запис (рис.3.4) .

Під час реєстрації необхідно заповнити форму, в якій вказати такі дані:

- ім'я користувача;
- електронна пошта;
- пароль.

Після відправлення форми система перевіряє, чи існує вже обліковий запис із зазначеною електронною адресою. Якщо такий обліковий запис знайдено, на екрані з'являється повідомлення про те, що користувач із цією електронною адресою вже зареєстрований (рис. 3.5).

Якщо реєстрація пройшла успішно, введені дані зберігаються в системі, після чого користувач може увійти в систему.

Для авторизації необхідно ввести:

- електронну пошту;
- пароль.

Після відправлення форми система звіряє зазначену інформацію з даними в базі даних. Якщо пароль введено неправильно, користувач отримує повідомлення про помилку. Якщо електронна пошта та пароль введені правильно, програма «Інтелектуальний планувальник» відкривається в основному інтерфейсі, де можна виконувати різні дії, такі як створення, перегляд, фільтрація та редагування завдань, а також перегляд аналітики.

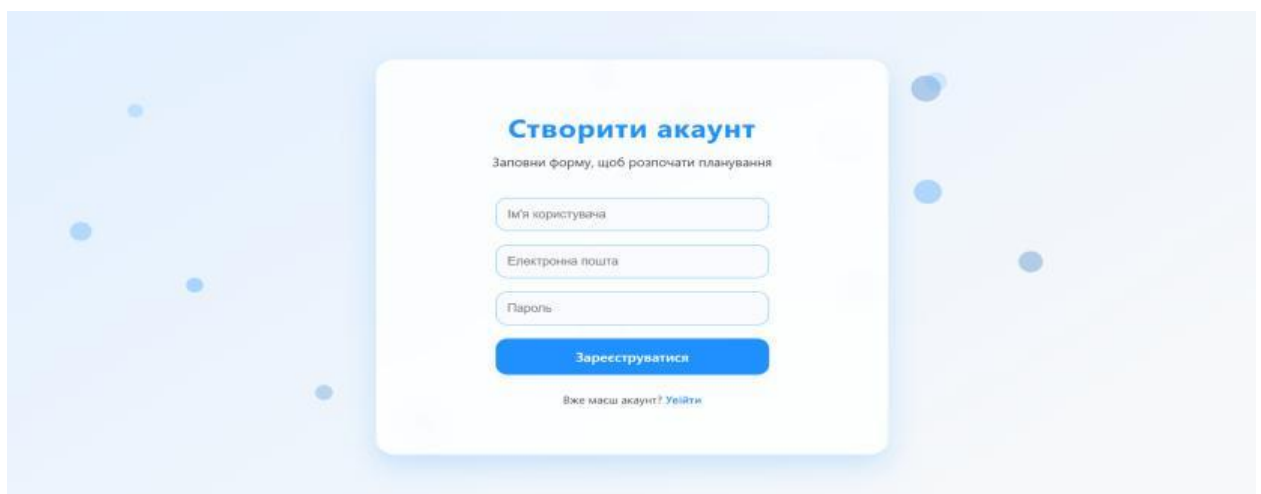


Рисунок 3.4 – Реєстрація користувача

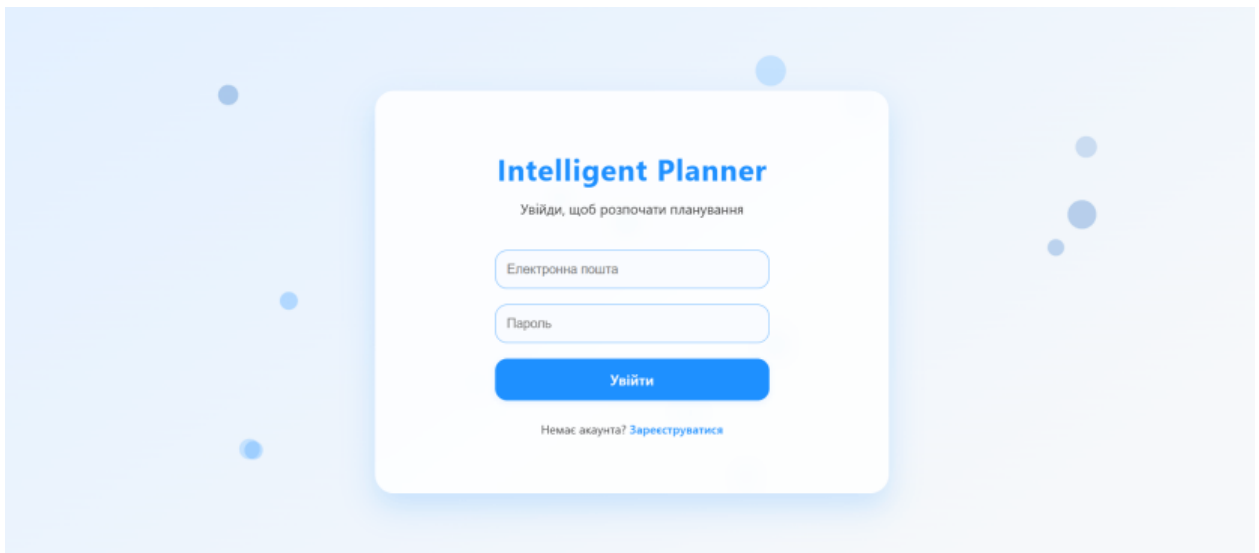


Рисунок 3.5 – Авторизація користувача

```

require('dotenv').config();
const express = require('express');
const cors = require('cors');
const connectDB = require('./config/db');
const User = require('./models/user');
const app = express();
connectDB().then(async () => {
  try {
    let testUser = await User.findOne({ email: 'test@example.com' });
    if (!testUser) {
      testUser = new User({username: 'test',email: 'test@example.com',
password: 'password123'});
      await testUser.save();
      console.log('Test user created with ID:', testUser._id);
    } else {
      console.log('Test user already exists with ID:', testUser._id);
    } catch (error) {
      console.error('Error creating test user:', error);
    }
  });
});

```

```

app.use(cors());
app.use(express.json());
app.post('/api/users/register', async (req, res) => {
  try {
    const existingUser = await User.findOne({ email: req.body.email });
    if (existingUser) {
      return res.status(400).json({
        message: 'Користувач із такою поштою вже існує!',
        field: 'email'});
    }
    const user = new User(req.body);
    const savedUser = await user.save();
    res.status(201).json(savedUser);
  } catch (error) {
    if (error && error.code === 11000) {
      const key = error.keyValue ? Object.keys(error.keyValue)[0] : null;
      const field = key || 'email';
      return res.status(400).json({
        message: `Користувач із таким ${field} вже існує!`,
        field});
    }
    if (error && error.name === 'ValidationError') {
      const firstKey = Object.keys(error.errors)[0];
      const field = firstKey || null;
      const message = error.errors[firstKey] ? error.errors[firstKey].message :
error.message;
      return res.status(400).json({ message, field });
    }
    res.status(400).json({ message: error.message });
  }
}

```

```

});
app.post('/api/users/login', async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(404).json({
        message: 'Користувача з такою поштою не знайдено', field: 'email'});
    }
    if (user.password !== password) {
      return res.status(401).json({
        message: 'Невірний пароль', field: 'password'});
    }
    res.json({
      id: user._id, email: user.email, username: user.username
    });
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});

```

3.5 Метод інтелектуального планування персональних завдань

Метод базується на автоматичному аналізі даних про завдання, отриманих системою. Програма відстежує створені, виконані та прострочені

завдання, враховує їх категорії та пріоритет, а потім розраховує ключові показники ефективності.

Спочатку збираються дані за категоріями, кількістю виконаних та невиконаних завдань, рівнем прострочених завдань та середнім рівнем активності за останні кілька днів. Додатково аналізується динаміка ефективності, зокрема, темп роботи за поточний тиждень порівнюється з попереднім, щоб оцінити, чи користувач досягає прогресу, чи знижує активність.

Якщо в певній категорії виявлено погані результати або накопичується багато прострочених завдань, система ідентифікує це як поточну проблему та генерує персоналізовані рекомендації. Ці рекомендації базуються на реальних показниках: відсотку виконаних завдань, середньодобовій кількості виконаних завдань та динаміці темпу за останні тижні.

Таким чином, метод швидко оцінює рівень ефективності користувача, виявляє недоліки в плануванні та пропонує індивідуальні рекомендації щодо оптимізації управління часом.

```
<template>
<div class="analytics-page" v-if="!taskStore.loading">
<h1>Аналітика продуктивності</h1>
<div class="analytics-grid">
<div class="stats-card overall-stats">
<h2>Загальна статистика</h2>
<div class="stats-content">
<div class="stat-item">
<span class="stat-label">Всього завдань</span>
<span class="stat-value">{{ taskStats.total }}</span></div>
<div class="stat-item">
<span class="stat-label">Відсоток виконання</span>
<span class="stat-value">{{ taskStats.rate }}%</span></div>
```

```

<div class="stat-item"> <span class="stat-label">Виконано</span>
<span class="stat-value success">{{ taskStats.completed }}</span></div>
<div class="stat-item">
<span class="stat-label">Пропущено</span>
<span class="stat-value overdue">{{ overdueCount }}</span></div>
</div>
</div>
<div class="chart-card category-chart">
<h2>Продуктивність за категоріями</h2>
<canvas id="categoryChart"></canvas>
</div>
<div class="chart-card priority-chart">
<h2>Розподіл за пріоритетами</h2>
<canvas id="priorityChart"></canvas>
</div>
<div class="chart-card trends-chart">
<div class="trends-header"><h2>Тренди виконання</h2></div>
<div class="trends-controls">
<button class="period-btn" @click="prevWeek">◀</button>
<select      v-model.number="periodWeeks"      class="period-select"
title="Вибрати період">
  <option :value="1">1 тиждень</option>
  <option :value="2">2 тижні</option>
  <option :value="4">4 тижні</option>
</select>
<button class="period-btn" @click="nextWeek">▶</button>
</div>
<div class="trends-range">{{ weeklyRange }}</div>
<canvas id="trendsChart"></canvas></div>
</div>

```

```

<div class="insights-section"><h2>Інтелектуальний аналіз</h2>
<div class="insights-grid">
  <div class="insight-card" v-for="(suggestion, index) in suggestions"
:key="index" :class="suggestion.type">
    <i :class="getSuggestionIcon(suggestion.type)"></i>
    <p>{{ suggestion.message }}</p></div>
  </div>
</div>
<div class="category-stats">
  <h2>Детальна статистика за категоріями</h2>
  <div class="category-grid">
    <div v-for="(stats, category) in productivityByCategory" :key="category"
class="category-card">
      <h3>{{ getCategoryName(category) }}</h3>
      <div class="category-stats-content">
        <div class="progress-circle" :style="{ background: `conic-gradient(var(--
success-color) ${stats.rate}%, #f0f0f0 0)` }">
          <span>{{ stats.rate }}%</span>
        </div>
        <div class="category-details">
          <p>Всього: {{ stats.total }}</p>
          <p>Виконано: {{ stats.completed }}</p>
        </div>
      </div>
    </div>
  </div>
</div>
</div>
</div>
</div>
<div v-else>
  <p>Завантаження даних...</p>

```

```

</div>
</template>
<script setup>
import { onMounted, ref, computed, watch, nextTick } from 'vue'
import { useTaskStore } from '../store'
import Chart from 'chart.js/auto'
import { subDays, eachDayOfInterval, format, isSameDay, isBefore,
startOfDay, startOfWeek, endOfWeek, addWeeks } from 'date-fns'
import { uk } from 'date-fns/locale'
import analyticsService from '../services/analytics'
const taskStore = useTaskStore()
const charts = ref({})
const weekOffset = ref(0)
const periodWeeks = ref(1)
const taskStats = computed(() => {
const stats = taskStore.getTasksStats
return stats || { total: 0, completed: 0, rate: 0 }
})
const overdueCount = computed(() => {
const tasks = taskStore.getTasks || []
const todayStart = startOfDay(new Date())
return tasks.filter(t => !t.completed && t.dueDate && isBefore(new
Date(t.dueDate), todayStart)).length})
const productivityByCategory = computed(() => {
const categories = taskStore.getCategories || []
const tasks = taskStore.getTasks || []
const result = {}
categories.forEach(c => {
const tasksInCategory = tasks.filter(t => t.category === c.id)
const completed = tasksInCategory.filter(t => t.completed).length

```

```

const total = tasksInCategory.length
const rate = total > 0 ? Math.round((completed / total) * 100) : 0
result[c.id] = { total, completed, rate }})
return result})
const metrics = ref({})
const suggestions = ref([])
const selectedWeekStart = computed(() => {
const base = addWeeks(new Date(), weekOffset.value)
return startOfWeek(base, { weekStartsOn: 1 })
})
const selectedWeekEnd = computed(() => {
const start = selectedWeekStart.value
const end = subDays(addWeeks(start, periodWeeks.value), 1)
return end})
const weeklyRange = computed(() => {
const start = selectedWeekStart.value
const end = selectedWeekEnd.value
return `${format(start, 'dd.MM')}-${format(end, 'dd.MM')}`})
function prevWeek() { weekOffset.value -= Math.max(1,
periodWeeks.value) }
function nextWeek() { weekOffset.value += Math.max(1,
periodWeeks.value) }
function getCategoryName(categoryId) {
const category = taskStore.getCategories?.find(c => c.id === categoryId)
return category ? category.name : categoryId}
function getSuggestionIcon(type) {
const icons = {
warning: 'fas fa-exclamation-triangle',
info: 'fas fa-info-circle',
success: 'fas fa-check-circle'}

```

```

return icons[type] || 'fas fa-lightbulb'}

function createCategoryChart() {
  const ctx = document.getElementById('categoryChart')?.getContext('2d')
  if (!ctx) return null
  const categories = taskStore.getCategories || []
  const stats = productivityByCategory.value
  return new Chart(ctx, {
    type: 'bar',
    data: {
      labels: categories.map(c => c.name),
      datasets: [
        { label: 'Виконано', data: categories.map(c => stats[c.id]?.completed || 0),
          backgroundColor: 'rgba(75, 192, 192, 0.7)'},
        {label: 'Залишилось', data: categories.map(c => (stats[c.id]?.total || 0) -
          (stats[c.id]?.completed || 0)), backgroundColor: 'rgba(255, 99, 132, 0.7)'}]],
    options: {
      responsive: true,
      plugins: {
        legend: { position: 'top' },
        title: { display: false }
      },
      scales: {
        y: { beginAtZero: true, precision: 0 }}}})}

function createPriorityChart() {
  const ctx = document.getElementById('priorityChart')?.getContext('2d')
  if (!ctx) return null
  const priorities = taskStore.getPriorities || []
  const tasks = taskStore.getTasks || []
  const data = priorities.map(priority => ({
    priority: priority.name, color: priority.color,

```

```

count: tasks.filter(t => t.priority === priority.id).length
)))
return new Chart(ctx, {
  type: 'doughnut',
  data: {
    labels: data.map(d => d.priority),
    datasets: [{
      data: data.map(d => d.count),
      backgroundColor: data.map(d => d.color)}}],
  options: {
    responsive: true,
    plugins: {
      legend: { position: 'right' }}}})}
function createTrendsChart() {
  const ctx = document.getElementById('trendsChart')?.getContext('2d')
  if (!ctx) return null
  const start = selectedWeekStart.value
  const end = selectedWeekEnd.value
  const days = eachDayOfInterval({ start, end })
  const tasks = taskStore.getTasks || []
  const completedByDay = days.map(day => {
    return tasks.filter(t => {
      if (!t.completed || !t.completedAt) return false
      const cd = new Date(t.completedAt)
      if (isNaN(cd)) return false
      return isSameDay(cd, day)
    }).length
  })
  return new Chart(ctx, {
    type: 'line',

```

```

data: {
  labels: days.map(d => format(d, 'EEE d', { locale: uk })),
  datasets: [{
    label: 'Виконані завдання',
    data: completedByDay,
    borderColor: 'rgb(75, 192, 192)',
    tension: 0.1}}],
  options: {
    responsive: true,
    plugins: {
      legend: { display: false }
    },
    scales: {
      y: { beginAtZero: true, precision: 0 }}}})
watch(() => taskStore.getTasks, weekOffset, periodWeeks], ([newTasks])
=> {
  if (charts.value.trends) {
    try { charts.value.trends.destroy() } catch (e) { /* ignore */ }
    charts.value.trends = null
  }
  const tasks = taskStore.getTasks || []
  if (tasks.length > 0) {
    charts.value.trends = createTrendsChart()
  }
  }, { deep: true })
  onMounted(async () => {
    await taskStore.loadTasks()
    await nextTick()
    const tasks = taskStore.getTasks || []

```

```

const analyzed = analyticsService.analyzeTasks(tasks)
metrics.value = analyzed.metrics || {}
suggestions.value = analyzed.suggestions || []
if (tasks.length > 0) {
  charts.value.category = createCategoryChart()
  charts.value.priority = createPriorityChart()
  charts.value.trends = createTrendsChart()
}
taskStore.loaded = true
})
watch(() => taskStore.getTasks, (newTasks) => {
  const analyzed = analyticsService.analyzeTasks(newTasks || [])
  metrics.value = analyzed.metrics || {}
  suggestions.value = analyzed.suggestions || []
}, { deep: true })
</script>

```

3.6 Додавання завдань

У додатку інтелектуального планувальника користувач може створювати власні завдання, щоб структурувати свій день та контролювати виконання справ. Під час додавання завдання користувач заповнює назву, опис, вибирає категорію (Робота, Навчання, Особисте тощо), встановлює пріоритет і кінцевий термін виконання.

Якщо завдання з часом буде позначене як виконане, система автоматично фіксує дату завершення, що дозволяє формувати точну аналітику – наприклад, обчислювати продуктивність або визначати, які справи користувач найчастіше відкладає.

```

import { API_URL } from './config';
export const tasksApi = {

```

```

async getAllTasks(userId) {
  const response = await fetch(`${API_URL}/tasks?userId=${userId}`);
  if (!response.ok) throw new Error('Failed to fetch tasks');
  return response.json();
},
async createTask(task) {
  const response = await fetch(`${API_URL}/tasks`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(task),
  });
  if (!response.ok) throw new Error('Failed to create task');
  return response.json();
},
async updateTask(id, updates) {
  const response = await fetch(`${API_URL}/tasks/${id}`, {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(updates),
  });
  if (!response.ok) throw new Error('Failed to update task');
  return response.json();
},
async deleteTask(id) {
  const response = await fetch(`${API_URL}/tasks/${id}`, {
    method: 'DELETE',

```

```

});
if (!response.ok) throw new Error('Failed to delete task');
return response.json();
}
};

const mongoose = require('mongoose');
const taskSchema = new mongoose.Schema({
  title: { type: String, required: true },
  description: { type: String },
  priority: { type: String, enum: ['low', 'medium', 'high'], default: 'medium' },
  dueDate: { type: Date },
  category: { type: String, enum: ['work', 'personal', 'study', 'other'], default:
'work' },
  completed: { type: Boolean, default: false },
  completedAt: { type: Date, default: null },
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true
},
  createdAt: { type: Date, default: Date.now },
  updatedAt: { type: Date, default: Date.now }
});

taskSchema.pre('save', function (next) {
  if (this.isModified('completed')) {
    this.completedAt = this.completed ? new Date() : null;
    this.updatedAt = Date.now();
    next();
  }
});

module.exports = mongoose.model('Task', taskSchema);
app.post('/api/tasks', async (req, res) => {
  try {
    const task = new Task(req.body);
    const savedTask = await task.save();

```

```
res.status(201).json(savedTask);  
} catch (error) {  
res.status(400).json({ message: error.message });});});
```

3.7 Висновки за розділом 3

Було розроблено програмне забезпечення для інтелектуального планувальника персональних завдань. Відображено структуру системи. У ході розробки вебсервісу були визначені його основні складові: частина додатку клієнта, що являє собою інтерфейс, який забезпечує передачу інформації між користувачем та програмними компонентами, серверна частина додатку, яка забезпечую обробку запитів користувачів, авторизацію, метод планування персональних завдань та різноманітні взаємодії з компонентами. Описано схему функціонування інтелектуального планувальника персональних завдань. Здійснено проєктування бази даних системи інтелектуального планувальника персональних завдань.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення й умови застосування програми

Програмне забезпечення розроблено для підвищення особистої продуктивності користувачів, оптимізації управління часом та полегшення управління повсякденними завданнями. Додаток побудовано на сучасних вебтехнологіях, включаючи Vue.js, Node.js, HTML та CSS. Вебсервіс реалізовано за допомогою Microsoft Visual Studio Code, кросплатформного та безкоштовного інструменту, який чудово підходить для розробки вебдодатків. Система управління базами даних MongoDB забезпечує гнучке зберігання інформації про завдання, події та аналітичних даних.

Ключові функції інтелектуального персонального планувальника завдань включають:

- створення, редагування та відстеження персональних завдань і подій;
- встановлення пріоритетів, дедлайнів та категорій завдань;
- можливість позначати завдання як виконані;
- формування рекомендацій щодо порядку виконання завдань відповідно до термінів та пріоритетів;
- відображення графіків продуктивності за різними категоріями робіт;
- автоматичний аналіз активності користувача та пропозиції щодо оптимального планування часу.

Наукова складова системи охоплює збір даних про продуктивність, аналіз завершених і прострочених завдань, виявлення закономірностей у роботі та надання індивідуальних рекомендацій для покращення організації часу. Це дозволяє не лише відслідковувати події, але й отримувати інтелектуальну підтримку в управлінні особистим графіком.

Для ефективного використання інтелектуального планувальника необхідні такі технічні вимоги:

- оперативна пам'ять від 4 ГБ;
- операційна система з підтримкою вебдодатків;
- відеоадаптер DirectX 10;
- процесор з частотою не менше 1 ГГц;
- стабільне підключення до Інтернету;
- актуальний веббраузер.

Функціональність додатку детально описана в технічному завданні (додаток А). Основні програмні механізми роботи планувальника проілюстровані у фрагментах коду, наведених у додатку Б.

4.2 Характеристики розробленого продукту

Програмне забезпечення інтелектуального персонального планувальника завдань працює безперервно, адаптуючись до індивідуальних потреб користувача та зміни його робочого навантаження. Швидкість взаємодії з вебсервісом залежить від пропускну здатності вебшлюзу сервера, якості інтернет-з'єднання користувача, а також продуктивності його комп'ютерного обладнання та програмного забезпечення.

Під час роботи система надає користувачеві інформацію у вигляді динамічно генерованих вебсторінок, на яких відображаються такі функції:

- авторизація та доступ до особистого простору;
- перегляд усіх створених завдань;
- можливість створювати, редагувати та видаляти завдання;
- встановлення термінів, пріоритетів і категорій;
- визначення завдань як виконаних;
- отримання рекомендацій для оптимального планування на основі попередньої активності;
- формування графіків продуктивності за категоріями та загальної статистики виконання;

- побудова діаграмів стану завдань (виконані, у процесі виконання, прострочені);
- автоматичний аналіз ефективності користувача за певний період.

Під час роботи програмне забезпечення інтелектуального планувальника активно використовує можливості сучасної системи управління базами даних MongoDB. База даних забезпечує швидку обробку великої кількості записів, гнучко зберігає структуровану інформацію про завдання та статистику продуктивності, а також підтримує масштабованість вебзастосунку.

4.3 Структурна схема компонентів вебсервісу

Створений додаток умовно можна розділити на дві основні логічні частини.

Перша частина – це клієнтський модуль, який надає користувачам інтерфейс для роботи з їхніми завданнями. Тут можна переглядати особисті списки, створювати нові записи, редагувати або видаляти їх. У цьому модулі також реалізовані функції налаштування параметрів, термінів виконання, визначення виконаних завдань та перегляду продуктивності за допомогою графіків та діаграм. Додаток пропонує користувачам інтелектуальні поради, які допомагають оптимізувати графік з урахуванням попередньої активності та поточних навантажень.

Друга частина – це серверний модуль, основним завданням якого є авторизація користувачів, обробка запитів та взаємодія з базою даних. Після автентифікації сервер відповідає за управління операціями зі створення, зміни та збереження завдань. Крім того, сервер виконує аналітичні алгоритми для генерації рекомендацій та збору даних для статистичних звітів. Клієнтський модуль не має прямого доступу до бази даних – усі операції введення змінюються через сервер. Це забезпечує надійний захист інформації та стабільність системи.

4.4 Висновки за розділом 4

У цьому розділі розглянуто ключове призначення та умови використання додатку, призначеного для інтелектуального планувальника персональних завдань. Представлено основні характеристики вебсервісу та описано зміст основних інформаційних сторінок. Також проаналізовано структурну схему компонентів програми, яка відображає логічну взаємодію елементів, розроблених у додатку.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Опис роботи з програмою

5.1.1 Звернення до програми

Клієнти можуть звертатися до вебсервісу інтелектуального планувальника персональних завдань через будь-який сучасний веббраузер, який підтримує виконання вебскриптів.

5.1.2 Вхідні й вихідні дані

Вхідними даними під час авторизації є логін та пароль користувача.

Вихідними є підтвердження успішного входу або повідомлення про помилку. Після входу система також формує початкові дані для подальшої роботи – ідентифікатор користувача та доступ до його персонального списку завдань.

Завантаження завдань: на вході система отримує ідентифікатор користувача, за яким здійснює запит до бази даних. На виході формується повний перелік особистих завдань, включаючи дедлайни, категорії, пріоритети та їхній поточний статус.

Перевірка дедлайнів: вхідними даними є список задач користувача з датами виконання. Вихідними – список задач із критичними або близькими дедлайнами, що необхідно позначити користувачу. Якщо виявлено прострочені завдання, система надає рекомендації щодо їх обробки.

Аналітика та інтелектуальна обробка даних: на вході система отримує історію виконання завдань, їх кількість, частоту змін дедлайнів, час виконання та інші параметри. На виході формуються статистичні зведення й аналітичні рекомендації: прогнозоване навантаження на тиждень, задачі з найвищим ризиком прострочення, поради щодо оптимального планування.

5.1.3 Повідомлення

Користувач інтелектуального планувальника персональних завдань у процесі взаємодії з вебсервісом може отримувати системні повідомлення, що виникають під час оновлення стану завдань. Повідомлення надсилаються у випадках, коли користувач створює нове завдання, змінює його параметри або позначає його як виконане.

Окрім цього, система інформує про наближення дедлайнів, появу критичних завдань та рекомендації, сформовані аналітичним модулем. Завдяки таким повідомленням користувач оперативно отримує всю необхідну інформацію для ефективного планування та контролю власних задач.

5.2 Інструкція з використання

Після переходу за вебадресою <http://localhost:5173/register>, перед користувачем з'являється сторінка реєстрації (рис. 5.1).

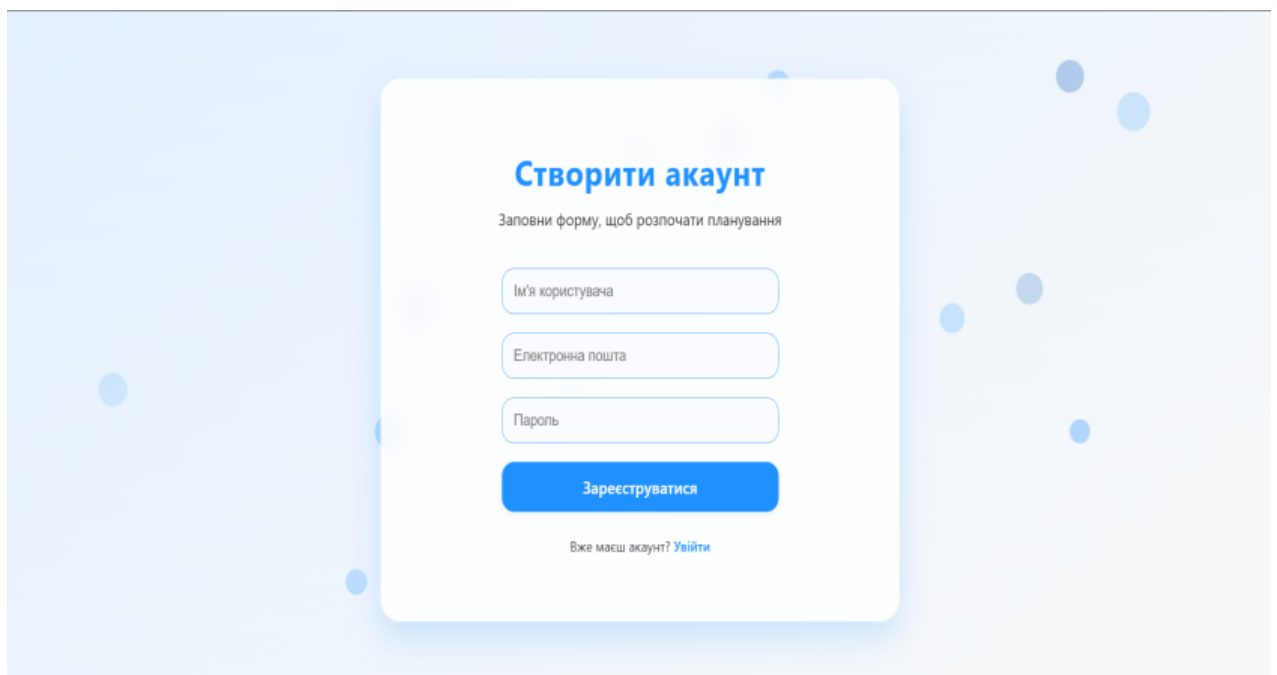
Скріншот веб-сторінки реєстрації. У центрі білий картка на світло-блакитному фоні з декоративними кругами. На картці: заголовок 'Створити акаунт' (синій), підзаголовок 'Заповни форму, щоб розпочати планування' (сірий), три поля вводу (Ім'я користувача, Електронна пошта, Пароль) з сірими підказками, синій кнопка 'Зареєструватися', посилання 'Вже маєш акаунт? Увійти' (синій).

Рисунок 5.1 – Вікно реєстрації

На цій сторінці користувач вводить свої дані для реєстрації (ім'я користувача, електронна пошта та пароль). Якщо у користувача є аккаунт, то користувач вводить свої дані (електронна пошта та пароль).

На цій сторінці (рис.5.2) користувач вводить свої дані: логін та пароль.

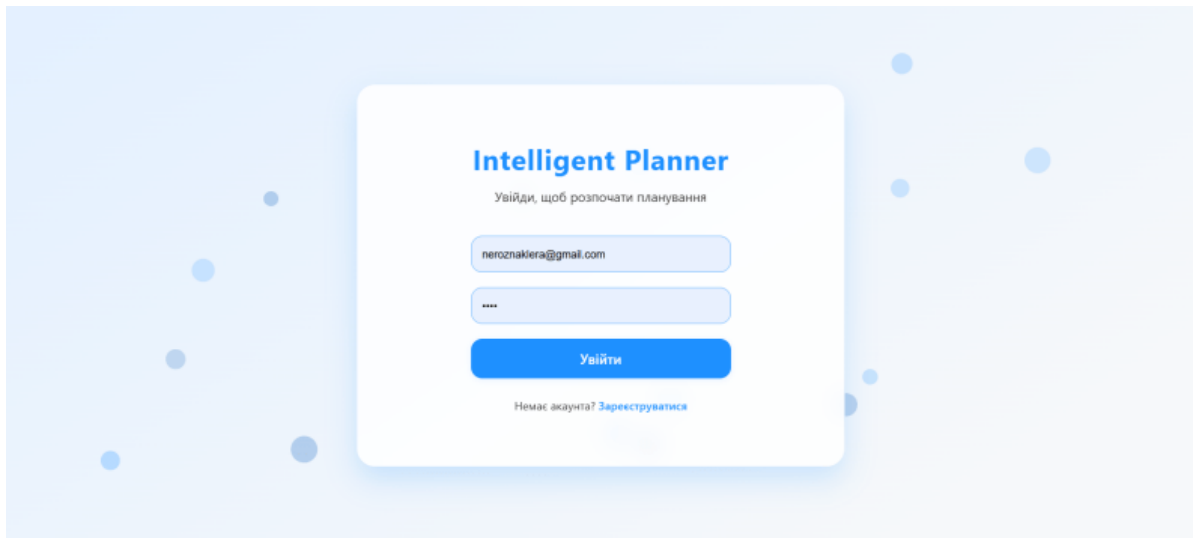


Рисунок 5.2 – Вікно авторизації

Після авторизації, користувач потрапляє на головну сторінку додатку (рис. 5.3).

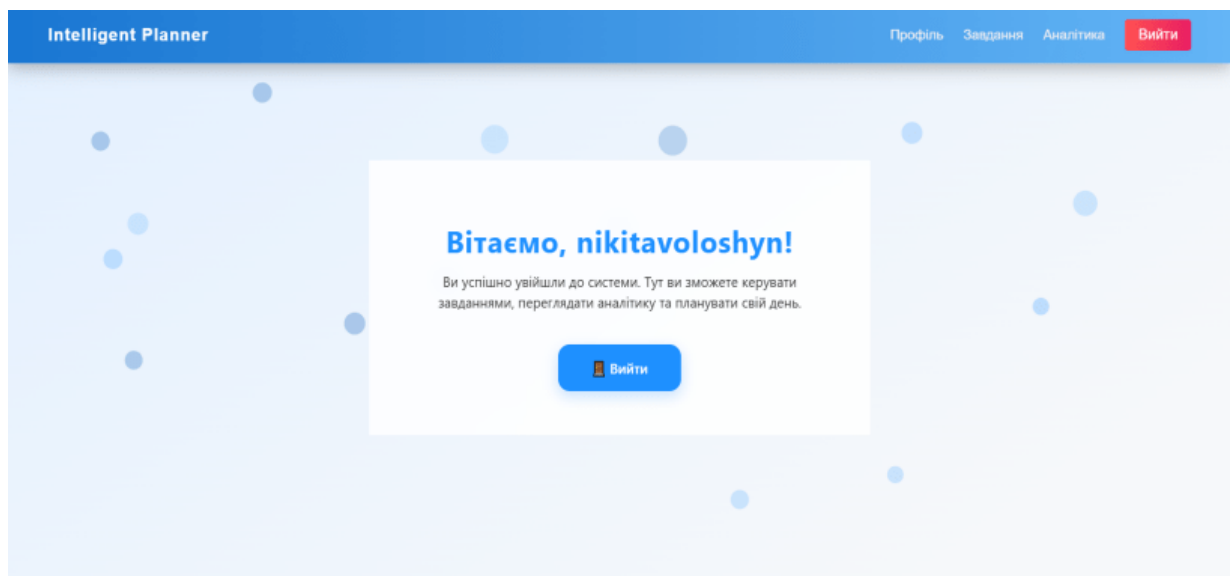


Рисунок 5.3 – Головна сторінка

У верхній частині екрана розташована стандартна навігаційна панель – «Профіль», «Завдання», «Аналітика» та кнопка «Вийти».

Повідомлення про вхід: Текст підтверджує успішну авторизацію і визначає основний функціонал, доступний користувачеві: «Ви успішно увійшли до системи. Тут ви зможете керувати завданнями, переглядати аналітику та планувати свій день.»

На вкладці «Завдання» користувач отримує доступ до повного списку своїх задач, де відображаються як виконані, так і активні та прострочені завдання, що дозволяє оперативно контролювати їхній стан та планувати подальші дії (рис. 5.4).

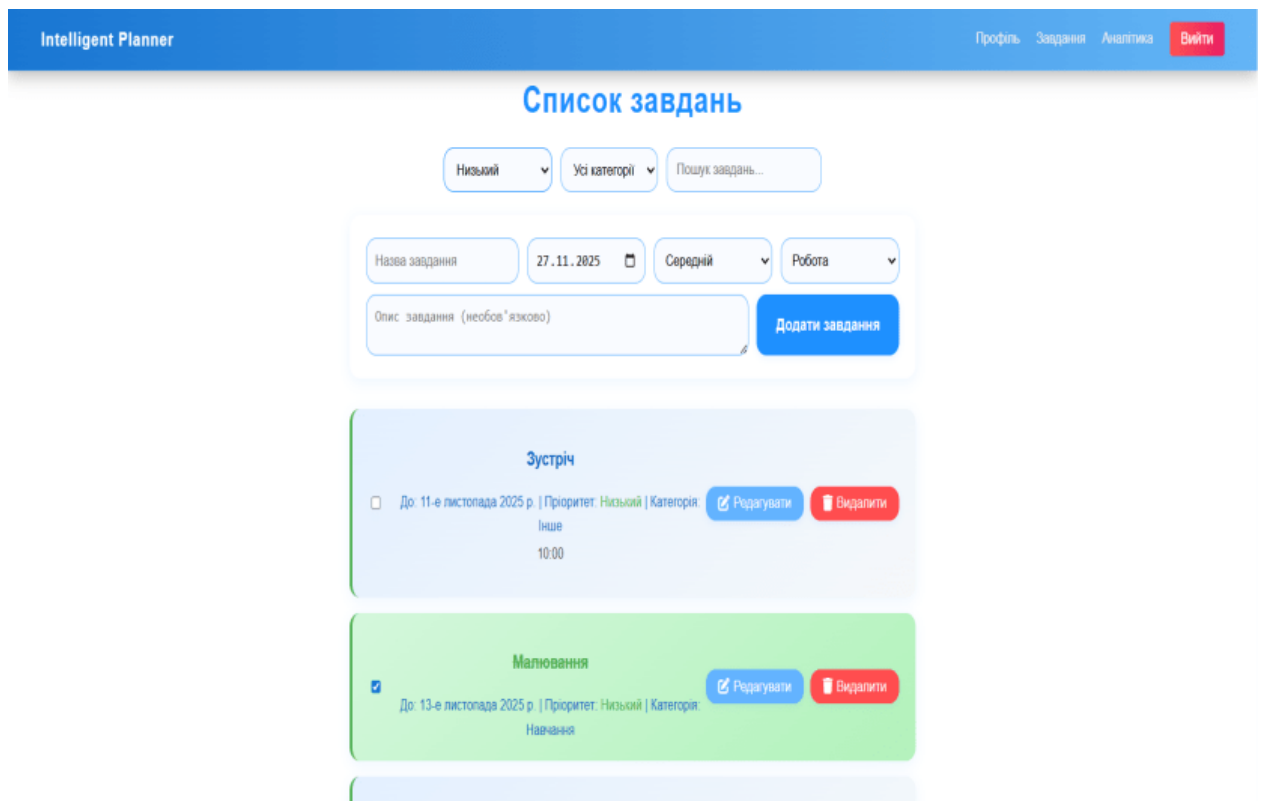


Рисунок 5.4 – Список завдань

Користувач також має можливість застосовувати фільтри за пріоритетом і категорією, а за потреби – виконувати пошук завдань за їх назвою, що забезпечує зручну навігацію та швидке знаходження потрібної інформації (рис.5.5).

Рисунок 5.5 – Фільтрація та пошук завдання

Користувач має можливість змінювати параметри вже створеного завдання. Після відкриття форми редагування у спливаючому вікні відображаються всі поточні дані задачі: назва, дата виконання, пріоритет, категорія та час (рис. 5.6).

Користувач може коригувати будь-який із цих параметрів – наприклад, змінити дедлайн, знизити або підвищити пріоритет, перемістити завдання до іншої категорії або уточнити час виконання. Після внесення змін користувач натискає «Зберегти», і система оновлює інформацію в базі даних та відображає оновлений вигляд задачі.

За потреби він може скасувати внесені зміни, використавши кнопку «Скасувати», після чого форма закривається без жодних коригувань. Такий механізм дозволяє швидко підтримувати актуальність усіх завдань та гнучко реагувати на зміни у плануванні.

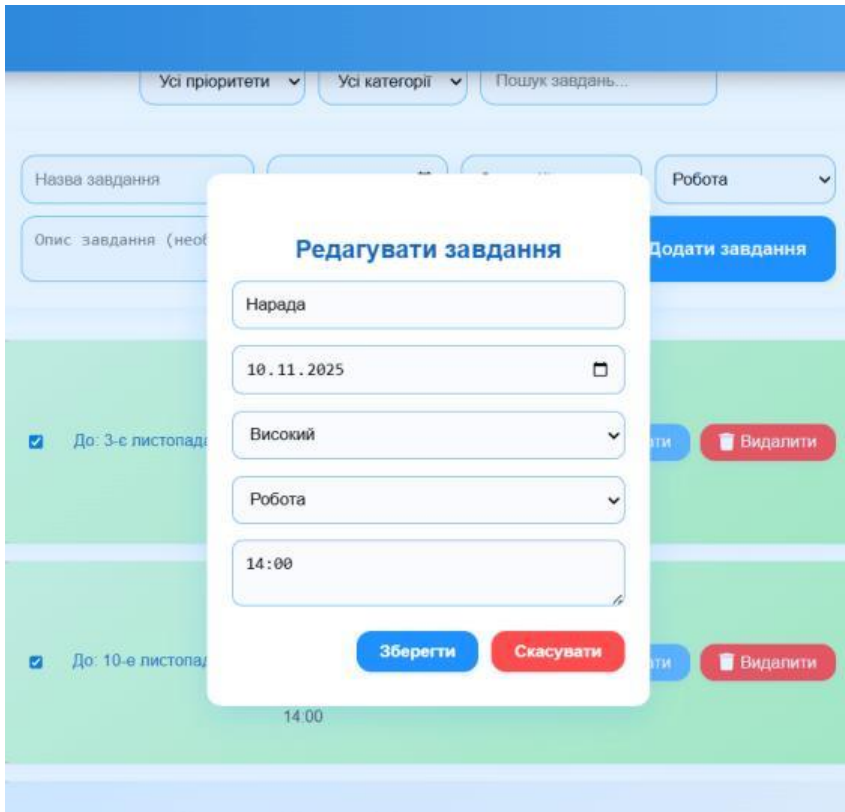


Рисунок 5.6 – Редагування завдання

Під час видалення завдання система відображає діалогове вікно підтвердження, яке запобігає випадковому видаленню важливої інформації (рис. 5.7). У спливаючому повідомленні користувачеві пропонується підтвердити свої дії, обравши «ОК», або скасувати операцію, натиснувши «Скасувати». Лише після підтвердження система видаляє завдання з бази даних та оновлює перелік задач. Такий механізм забезпечує додатковий рівень безпеки та дозволяє уникнути помилкових дій під час управління завданнями.

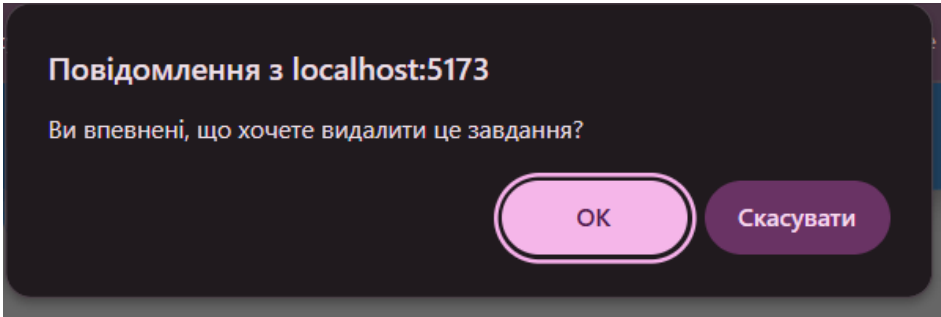


Рисунок 5.7 – Видалення завдання

На вкладці «Аналітика» користувач може переглядати узагальнену статистику своєї продуктивності, включно з кількістю створених, виконаних та пропущених завдань, а також відсотком виконання. Користувач отримує можливість аналізувати ефективність у різних категоріях, спостерігаючи, які типи завдань виконуються найчастіше, а які – залишаються невиконаними.

Додатково доступна візуалізація розподілу задач за пріоритетами, що дає змогу оцінити, наскільки збалансованим є поточне навантаження. Усі графіки та діаграми формуються автоматично на основі даних користувача, що дозволяє швидко оцінити власну продуктивність та виявити можливі проблемні зони в плануванні (рис. 5.8).

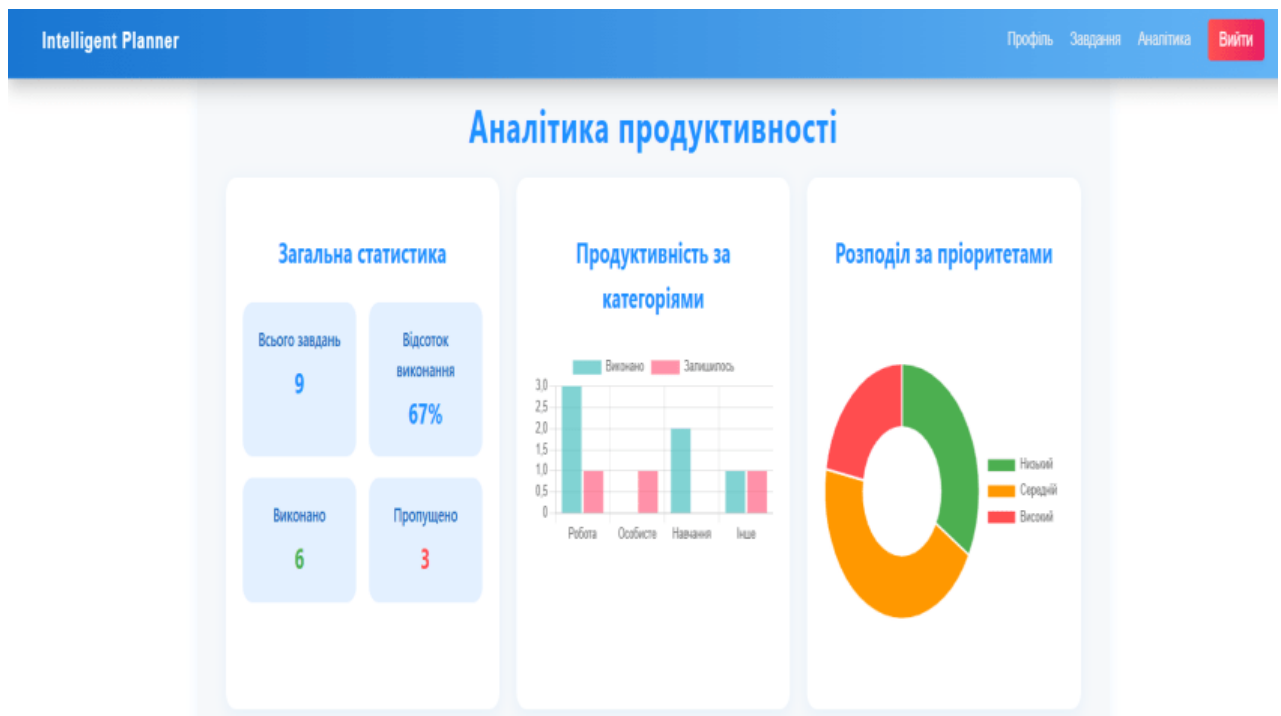


Рисунок 5.8 – Аналітика продуктивності

У нижній частині вкладки «Аналітика» користувач може переглядати розширені дані щодо виконання завдань. Блок «Тренди виконання» відображає динаміку продуктивності за обраний період, показуючи, у які дні користувач виконував найбільше завдань. Це дозволяє оцінити власний ритм роботи та виявити періоди підвищеної або зниженої активності.

Секція «Інтелектуальний аналіз» надає автоматично згенеровані рекомендації та висновки на основі поведінки користувача. Система визначає відсоток виконаних завдань, виявляє прострочені задачі, пропонує оптимізувати роботу з дедлайнами, а також підказує, коли навантаження є надмірним або навпаки — надто низьким. Ці підказки допомагають користувачеві краще організувати планування та своєчасно реагувати на можливі проблеми.

У блоці «Детальна статистика за категоріями» користувач бачить, як виконуються завдання в окремих сферах діяльності — робота, навчання, особисте життя тощо. Для кожної категорії відображається кількість створених і виконаних завдань, а також відсотковий показник успішності. Такий підхід дозволяє легко оцінити, у яких саме сферах користувач працює ефективно, а де необхідно приділити більше уваги (рис. 5.9).

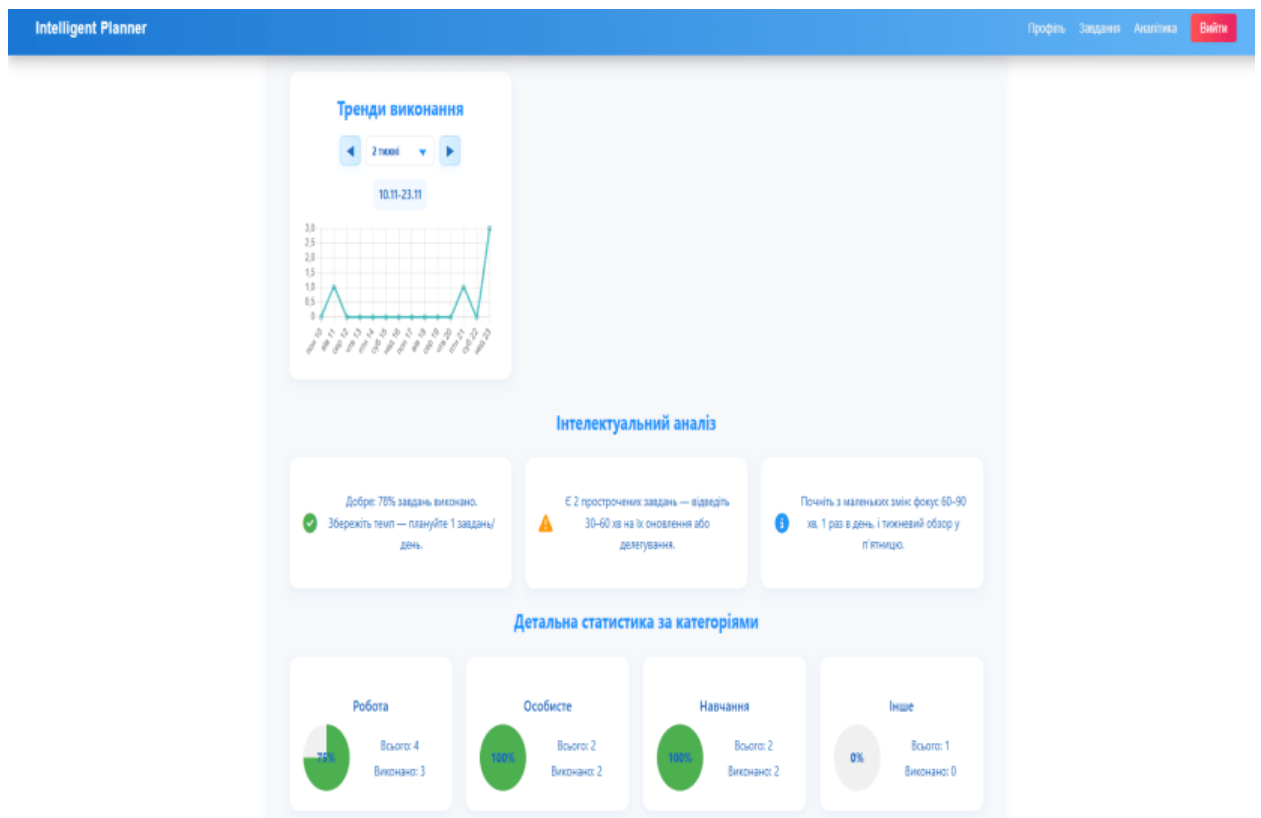


Рисунок 5.9 – Аналітика завдань

5.3 Експериментальне дослідження (тестування)

Основним критерієм оцінки тестування вебзастосунку інтелектуального планувальника персональних завдань є його відповідність визначеним технічним вимогам та стабільність роботи в реальних користувацьких сценаріях.

Відповідно до технічного завдання були встановлені такі функціональні вимоги до розроблюваного застосунку:

- авторизація та доступ до особистого простору;
- перегляд усіх створених завдань;
- можливість створювати, редагувати та видаляти завдання;
- встановлення термінів, пріоритетів і категорій;
- визначення завдань як виконаних;
- отримання рекомендацій для оптимального планування на основі попередньої активності;
- формування графіків продуктивності за категоріями та загальної статистики виконання;
- побудова діаграмів стану завдань (виконані, у процесі виконання, прострочені);
- автоматичний аналіз ефективності користувача за певний період.

У процесі роботи було здійснено аналіз популярних рішень для планування особистих задач (Todoist, Trello, Microsoft To Do, Notion.). Під час розробки враховано їхні переваги та недоліки, а також реалізовано функції, які відрізняють створений додаток від існуючих аналогів, зокрема: візуалізацію детальної статистики виконання, автоматичний інтелектуальний аналіз завдань та формування персональних рекомендацій.

Результати тестування показали, що вебдодаток повністю відповідає технічному завданню та коректно реалізує всі передбачені функції.

Застосунок демонструє стабільну роботу, забезпечує швидку взаємодію з інтерфейсом і коректну обробку даних користувача.

Тестування підтвердило, що інтелектуальний планувальник персональних завдань відповідає вимогам, є зручним у використанні та забезпечує високий рівень функціональності, необхідний для ефективного управління персональними задачами.

5.4 Висновки за розділом 5

У цьому розділі було представлено програмне забезпечення інтелектуального планувальника персональних завдань та наведено інструкцію щодо його використання. Проведене тестування системи включало перевірку функціональності, зручності взаємодії та відповідності вимогам технічного завдання. Отримані результати засвідчили, що розроблений інтелектуальний планувальник повністю відповідає заданим технічним критеріям, коректно виконує функції автоматизованого управління завданнями та демонструє високі показники стабільності й ефективності роботи.

ВИСНОВКИ

У ході виконання дипломної кваліфікаційної роботи магістра було проаналізовано та досліджено методи інтелектуального планування персональних завдань і розроблено програмне забезпечення для їх автоматизованого управління.

Було розглянуто основні концепції тайм-менеджменту, сучасні тенденції розвитку цифрових планувальників та питання популярності систем персональної продуктивності. Проведено огляд та аналіз програмних аналогів, серед яких для детальнішого порівняльного вивчення було виділено такі сервіси, як Todoist, Trello, Microsoft To Do та Notion.

Для реалізації програмного забезпечення інтелектуального планувальника персональних завдань обрано фреймворк Vue.js та платформу Node.js, які широко застосовуються у сучасній веброзробці та забезпечують можливість створення масштабованих, зручних і високопродуктивних вебзастосунків.

Для розробки системи було використано середовище Visual Studio Code, яке підтримує інтеграцію з обраними технологіями, пропонує широкий функціонал для налагодження коду та забезпечує зручні умови для командної та індивідуальної розробки програмного забезпечення.

У межах роботи запропоновано метод інтелектуального планування персональних завдань, заснований на аналізі даних користувача щодо термінів виконання, рівня пріоритетності, тривалості та історії виконання завдань. Алгоритм роботи системи здійснює обробку зазначених параметрів, розраховує індивідуальні пріоритетні коефіцієнти та автоматично формує оптимізований перелік завдань для користувача з урахуванням дедлайнів та поточного навантаження. У разі перевищення допустимих часових меж система генерує сповіщення та рекомендації щодо перерозподілу завдань.

Результати роботи планувальника візуалізуються у вигляді аналітичних панелей та графіків, що відображають рівень виконання завдань, розподіл часу та динаміку щоденної продуктивності. Це дозволяє користувачеві наочно оцінювати ефективність власного планування та коригувати стратегію управління часом.

Ефективність запропонованого методу може бути перевірена шляхом емпіричного аналізу статистики використання системи, зокрема за показниками зростання виконаних завдань, зменшення прострочених дедлайнів та підвищення регулярності планування.

У ході виконання роботи було спроектовано структуру програмного забезпечення інтелектуального планувальника та визначено його основні компоненти: клієнтську частину вебзастосунку, що забезпечує взаємодію користувача з функціоналом сервісу; серверну частину, яка обробляє запити та виконує бізнес-логіку застосунку; базу даних, призначену для зберігання інформації про завдання, користувачів та історію взаємодії із системою.

Було описано функціональну схему роботи додатку, визначено призначення та умови використання інтелектуального планувальника персональних завдань, охарактеризовано ключові можливості системи, а також розроблено структуру та зміст інформаційних вебсторінок інтерфейсу користувача.

Створено повнофункціональне програмне забезпечення інтелектуального планувальника та підготовлено інструкцію з його використання. Проведене тестування підтвердило коректність роботи усіх основних модулів системи та відповідність програмного продукту вимогам технічного завдання. За результатами перевірки встановлено, що розроблений планувальник демонструє високі показники ефективності у порівнянні з існуючими аналогами та забезпечує зручний інструментарій для підвищення особистої продуктивності користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Walek B., Novak V. Intelligent Task Planning System Based on Methods of Fuzzy Natural Logic / B. Walek, V. Novák. – Axioms. – 2023. – Vol. 12, no. 6, art. 545
2. Sajja R., Sermet Y., Cikmaz M., Cwiertny D., Demir I. Artificial Intelligence-Enabled Intelligent Assistant for Personalized and Adaptive Learning in Higher Education / R. Sajja, Y. Sermet, M. Cikmaz, D. Cwiertny, I. Demir. – Information. – 2024. – Vol. 15, art. 596.
3. Marikyan D., Papagiannidis S., Rana O. F., Ranjan R., Morgan G. “Alexa, let’s talk about my productivity”: The impact of digital assistants on work productivity / D. Marikyan, S. Papagiannidis, O. F. Rana, R. Ranjan, G. Morgan. – Journal of Business Research. – 2022. – Vol. 142. – P. 572–584.
4. Todoist [Електрон. ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/Todoist>
5. Trello [Електрон. ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/Trello>
6. Microsoft To Do [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Microsoft_To_Do
7. Notion [Електрон. ресурс]. – Режим доступу: <https://coursebox.info/partner/notion>
8. Flanagan D. JavaScript: The Definitive Guide. – 7th ed. – Sebastopol : O’Reilly Media, 2020. – 706 p.
9. Haverbeke M. Eloquent JavaScript. – 3rd ed. – San Francisco : No Starch Press, 2018. – 456 p.
10. Djirdeh H., Murray N., Lerner A. Fullstack Vue: The Complete Guide to Vue.js. – CreateSpace / Fullstack.io, 2018. – 442 p.
11. Macrae C. Vue.js: Up and Running. Building Accessibility and Performant Web Apps. – Sebastopol : O’Reilly Media, 2018. – 262 p.
12. Ribeiro H. R. Vue.js 3 Cookbook: Discover actionable solutions for

building modern web apps with the latest Vue features and TypeScript. – Packt Publishing, 2020. – 562 p.

13. Robbins J. N. Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics (5-th ed.). – O'Reilly Media, 2018. – 808 p.

14. Duckett J. HTML and CSS: Design and Build Websites. – Indianapolis : Wiley, 2018. – 490 p.

15. Powell, T. HTML & CSS: The Complete Reference, Fifth Edition / T. Powell – New York: Spring, 2010. – 785 p.

16. Meyer, E. CSS: The Definitive Guide, 4th Edition / E. Meyer. – Sebastopol : O'Reilly Media, 2017. – 864 p.

17. Lerner, A. Fullstack Vue: The Complete Guide to Vue.js / A. Lerner, N. Murray, H. Djirdeh. –San Francisco: Spring, 2018. – 332 p.

18. Casciaro M., Mammino L. Node.js Design Patterns. – Packt Publishing, 2020. – 526 p.

19. Casciaro, M. Node.js Design Patterns / M. Casciaro. – Hamilton: Spring, 2016. – 221 p.

20. Johnson, B. Visual Studio Code: End-to-End Editing and Debugging Tools for Web Developers/ B. Johnson – New York: Wiley, 2022. – 15 p.

21. Johnson, B. Visual Studio Code: End-to-End Editing and Debugging Tools for Web Developers/ B. Johnson – New York: Wiley, 2022. – 35 p.

22. Johnson, B. Visual Studio Code: End-to-End Editing and Debugging Tools for Web Developers/ B. Johnson – New York: Wiley, 2022. – 213 p.

23. Johnson, B. Visual Studio Code: End-to-End Editing and Debugging Tools for Web Developers/ B. Johnson – New York: Wiley, 2022. – 291 p.

24. Cbodorow, K. MongoDB: The Definitive Guide/ K. Cbodorow – New York: Spring, 2013. – 386 p.

25. Koplend, R. MongoDB Applied Design Patterns/ R.Koplend: Washington: Spring, 2013. – 209 p.

ДОДАТОК А
Технічне завдання

Вступ

Програмне забезпечення може використовуватися для планування персональних завдань.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Розробка інтелектуального планувальника персональних завдань», затверджене наказом Національного університету «Запорізька політехніка» від 30 вересня 2025 р. за № 447.

A.2 Призначення розробки

Програмний продукт призначений для забезпечення та підтримки процесу планування персональних завдань.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Програмне забезпечення інтелектуального планувальника персональних завдань забезпечує виконання таких функцій:

- авторизація та доступ до особистого простору;
- перегляд усіх створених завдань;
- можливість створювати, редагувати та видаляти завдання;
- встановлення термінів, пріоритетів і категорій;
- визначення завдань як виконаних;
- отримання рекомендацій для оптимального планування на основі попередньої активності;
- формування графіків продуктивності за категоріями та загальної статистики виконання;

- побудова діаграмів стану завдань (виконані, у процесі виконання, прострочені);
- автоматичний аналіз ефективності користувача за певний період.

A.3.2 Вимоги до інтерфейсу програми

Інтерфейс додатку інтелектуального планувальника персональних завдань повинен бути зручним та зрозумілим для користувачів.

A.3.3 Вимоги до надійності

Програмне забезпечення інтелектуального планувальника персональних завдань повинно забезпечити надійне та стабільне функціонування.

A.3.4 Умови експлуатації

Для експлуатації програмного забезпечення інтелектуального планувальника персональних завдань у користувача повинен бути наявний пристрій з встановленою операційною системою, з можливістю взаємодії з програмними інтерфейсами через мережу інтернет.

A.3.5 Вимоги до складу та параметрів технічний засобів

Для комфортного використання програмного забезпечення системи планування начального процесу необхідні такі апаратні та програмні компоненти:

- оперативна пам'ять від 4 ГБ;
- операційна система з підтримкою вебдодатків;
- відеоадаптер DirectX 10;
- процесор з частотою не менше 1 ГГц;
- стабільне підключення до Інтернету;
- актуальний веббраузер.

А.3.6 Вимоги до маркування і пакування

Програму для інтелектуального планувальника персональних завдань можна записати на будь-який носій інформації. На пакування обов'язково повинна бути вказана назва програми: «Програмне забезпечення для інтелектуального планувальника персональних завдань».

А.4 Вхідні дані до роботи

Вхідними даними під час авторизації є логін та пароль користувача.

Вихідними є підтвердження успішного входу або повідомлення про помилку. Після входу система також формує початкові дані для подальшої роботи – ідентифікатор користувача та доступ до його персонального списку завдань.

Завантаження завдань: на вході система отримує ідентифікатор користувача, за яким здійснює запит до бази даних. На виході формується повний перелік особистих завдань, включаючи дедлайни, категорії, пріоритети та їхній поточний статус.

Перевірка дедлайнів: вхідними даними є список задач користувача з датами виконання. Вихідними – список задач із критичними або близькими дедлайнами, що необхідно позначити користувачу. Якщо виявлено прострочені завдання, система надає рекомендації щодо їх обробки.

Аналітика та інтелектуальна обробка даних: на вході система отримує історію виконання завдань, їх кількість, частоту змін дедлайнів, час виконання та інші параметри. На виході формуються статистичні зведення й аналітичні рекомендації: прогнозоване навантаження на тиждень, задачі з найвищим ризиком прострочення, поради щодо оптимального планування.

ДОДАТОК Б
Текст програми

Б.1 Текст файлу index.js

```

require('dotenv').config();
const express = require('express');
const cors = require('cors');
const connectDB = require('./config/db');
const Task = require('./models/task');
const User = require('./models/user');
const app = express();
connectDB().then(async () => {
  try {
    let testUser = await User.findOne({ email: 'test@example.com' });
    if (!testUser) {
      testUser = new User({
        username: 'test',
        email: 'test@example.com',
        password: 'password123'
      });
      await testUser.save();
      console.log('Тестовий користувач створений з ID:', testUser._id);
    } else {
      console.log('Тестовий користувач вже існує з ID:', testUser._id);
    }
  } catch (error) {
    console.error('Помилка при створенні тестового користувача:',
error);
  }
});
app.use(cors());
app.use(express.json());

```

// GET /api/tasks - Отримання всіх завдань користувача за userId

```
app.get('/api/tasks', async (req, res) => {
  try {
    const userId = req.query.userId;
    const tasks = await Task.find({ userId }).populate('userId',
'username');
    res.json(tasks);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});
```

// POST /api/tasks - Створення нового завдання

```
app.post('/api/tasks', async (req, res) => {
  try {
    const task = new Task(req.body);
    const savedTask = await task.save();
    res.status(201).json(savedTask);
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
});
```

// PUT /api/tasks/:id - Оновлення існуючого завдання

```
app.put('/api/tasks/:id', async (req, res) => {
  try {
    const task = await Task.findByIdAndUpdate(
      req.params.id,
      req.body,
      { new: true, runValidators: true }
    );
    if (!task) {
```

```

        return res.status(404).json({ message: 'Завдання не знайдено' });
    }
    res.json(task);
} catch (error) {
    res.status(400).json({ message: error.message });
}
});

// DELETE /api/tasks/:id - Видалення завдання
app.delete('/api/tasks/:id', async (req, res) => {
    try {
        const task = await Task.findByIdAndDelete(req.params.id);
        if (!task) {
            return res.status(404).json({ message: 'Завдання не знайдено' });
        }
        res.json({ message: 'Завдання успішно видалено' });
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

// POST /api/users/register - Реєстрація нового користувача
app.post('/api/users/register', async (req, res) => {
    try {
        const existingUser = await User.findOne({ email: req.body.email });
        if (existingUser) {
            return res.status(400).json({
                message: 'Користувач із такою поштою вже існує!',
                field: 'email'
            });
        }
        const user = new User(req.body);

```

```

    const savedUser = await user.save();
    res.status(201).json(savedUser);
  } catch (error) {
    if (error && error.code === 11000) {
      const key = error.keyValue ? Object.keys(error.keyValue)[0] : null;
      const field = key || 'email';
      return res.status(400).json({
        message: `Користувач із таким ${field} вже існує!`,
        field
      });
    }
    if (error && error.name === 'ValidationError') {
      const firstKey = Object.keys(error.errors)[0];
      const field = firstKey || null;
      const message = error.errors[firstKey] ?
error.errors[firstKey].message : error.message;
      return res.status(400).json({ message, field });
    }
    res.status(400).json({ message: error.message });
  }
});

// POST /api/users/login - Вхід користувача в систему
app.post('/api/users/login', async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(404).json({
        message: 'Користувача з такою поштою не знайдено',
        field: 'email'
      });
    }
  }
});

```

```

    });
  }
  if (user.password !== password) {
    return res.status(401).json({
      message: 'Невірний пароль',
      field: 'password'
    });
  }
  res.json({
    id: user._id,          // MongoDB ID користувача
    email: user.email,      // Email користувача
    username: user.username // Ім'я користувача
  });
} catch (error) {
  res.status(500).json({ message: error.message });
}
});
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Сервер запущений на порту ${PORT}`);
});

```

Б.2 Текст файлу Tasks.vue

```

<template>
  <div class="tasks-page">
    <h1>Список завдань</h1>
    <div v-if="taskStore.loading" class="loading-spinner">Завантаження...
    </div>
    <div v-if="taskStore.error" class="error-message"> {{ taskStore.error }}
  </div>

```

```

</div>
<div class="filters">
  <select v-model="filterPriority">
    <option value="">Усі пріоритети</option>
    <option value="low">Низький</option>
    <option value="medium">Середній</option>
    <option value="high">Високий</option>
  </select>
  <select v-model="filterCategory">
    <option value="">Усі категорії</option>
    <option v-for="category in taskStore.getCategories"
:key="category.id" :value="category.id">
      {{ category.name }}
    </option>
  </select>
  <input type="text" v-model="searchQuery" placeholder="Пошук
завдань..." />
</div>
<form @submit.prevent="addTask" class="task-form">
  <input v-model="newTask.title" placeholder="Назва завдання"
required />
  <input type="date" v-model="newTask.deadline" required />
  <select v-model="newTask.priority">
    <option value="low">Низький</option>
    <option value="medium">Середній</option>
    <option value="high">Високий</option>
  </select>
  <select v-model="newTask.category">
    <option v-for="category in taskStore.getCategories"
:key="category.id" :value="category.id">

```

```

    {{ category.name }}
  </option>
</select>

<textarea      v-model="newTask.description"      placeholder="Опис
завдання (необов'язково)"></textarea>

  <button type="submit" class="add-btn">Додати завдання</button>
</form>

<ul class="task-list">
  <li v-for="task in filteredTasks" :key="task._id"
    :class="{ completed: task.completed }"
    :style="{
      borderLeft: `4px solid ${getPriorityColor(task.priority)}`,
      background: task.completed ? 'linear-gradient(145deg, #d4f7dc,
#b2f2bb)' : 'linear-gradient(145deg, #e3f0ff, #f6f8fa)'
    }">
    <div class="task-info">
      <input      type="checkbox"      :checked="task.completed"
@change="toggleTaskComplete(task._id)" />
      <div>
        <h3>{{ task.title }}</h3>
        <p class="task-meta">
          До: {{ formatDate(task.dueDate) }} |
          Пріоритет: <span :style="{ color: getPriorityColor(task.priority)
}">{{ getPriorityName(task.priority) }}</span> |
          Категорія: {{ getCategoryName(task.category) }}
        </p>
        <p      v-if="task.description"      class="task-description">{{
task.description }}</p>
      </div>
    </div>
  </li>
</ul>

```

```

<div class="task-actions">
  <button @click="openEditModal(task)" class="edit-btn">
    <i class="fas fa-edit"></i> Редагувати
  </button>
  <button @click="deleteTask(task._id)" class="delete-btn">
    <i class="fas fa-trash"></i> Видалити
  </button>
</div>
</li>
</ul>
<div v-if="isModalOpen" class="modal-overlay"
@click.self="closeModal">
  <div class="modal">
    <h2>Редагувати завдання</h2>
    <input v-model="editTaskData.title" placeholder="Назва завдання"
required />
    <input type="date" v-model="editTaskData.deadline" required />
    <select v-model="editTaskData.priority">
      <option v-for="priority in taskStore.getPriorities" :key="priority.id"
:value="priority.id">
        {{ priority.name }}
      </option>
    </select>
    <select v-model="editTaskData.category">
      <option v-for="category in taskStore.getCategories"
:key="category.id" :value="category.id">
        {{ category.name }}
      </option>
    </select>

```

```

    <textarea v-model="editTaskData.description" placeholder="Опис
завдання"></textarea>

    <div class="modal-buttons">
        <button @click="saveEdit" class="save-btn">Зберегти</button>
        <button @click="closeModal" class="cancel-
btn">Скасувати</button>
    </div>
</div>
</div>
</div>
</template>

<script setup>
import { ref, reactive, computed, onMounted } from 'vue'
import { useTaskStore } from '../store'
import { useRouter } from 'vue-router'
import { format } from 'date-fns'
import { uk } from 'date-fns/locale'
const taskStore = useTaskStore()
const router = useRouter()
const filterCategory = ref("")
const filterPriority = ref("")
const searchQuery = ref("")
const newTask = reactive({
  title: "",
  deadline: format(new Date(), 'yyyy-MM-dd'),
  priority: 'medium',
  description: "",
  category: "",
})
const isModalOpen = ref(false)

```

```

const editTaskData = reactive({
  id: null,
  title: "",
  deadline: format(new Date(), 'yyyy-MM-dd'),
  priority: 'medium',
  category: "",
  description: ""
})

onMounted(async () => {
  const user = JSON.parse(localStorage.getItem('user'))
  if (!user?.id) {
    router.push('/login')
    return
  }
  await taskStore.loadTasks()
  if (!newTask.category && taskStore.getCategories.length > 0) {
    newTask.category = taskStore.getCategories[0].id
  }
})

const suggestions = computed(() =>
taskStore.getTimeManagementSuggestions)

const filteredTasks = computed(() => {
  let tasks = taskStore.getTasks
  if (filterPriority.value) {
    tasks = tasks.filter(task => task.priority === filterPriority.value)
  }
  if (filterCategory.value) {
    tasks = tasks.filter(task => task.category === filterCategory.value)
  }
})

```

```

if (searchQuery.value) {
  const query = searchQuery.value.toLowerCase()
  tasks = tasks.filter(task =>
    task.title.toLowerCase().includes(query) ||
    task.description?.toLowerCase().includes(query)
  )
}
return tasks.sort((a, b) => new Date(a.dueDate) - new Date(b.dueDate))
})

function formatDate(date) {
  if (!date) return 'Дата не встановлена'
  try {
    const parsedDate = new Date(date)
    if (isNaN(parsedDate.getTime())) return 'Некоректна дата'
    return format(parsedDate, 'PPP', { locale: uk })
  } catch (error) {
    console.error('Error formatting date:', error)
    return 'Помилка дати'
  }
}

function getPriorityColor(priorityId) {
  const priority = taskStore.getPriorities.find(p => p.id === priorityId)
  return priority ? priority.color : '#000'
}

function getPriorityName(priorityId) {
  const priority = taskStore.getPriorities.find(p => p.id === priorityId)
  return priority ? priority.name : priorityId
}

function getCategoryName(categoryId) {
  const category = taskStore.getCategories.find(c => c.id === categoryId)

```

```

    return category ? category.name : categoryId
  }
function addTask() {
  const user = JSON.parse(localStorage.getItem('user'))
  if (!user?.id) {
    alert('Помилка: користувач не авторизований')
    return
  }
  if (!newTask.category && taskStore.getCategories.length > 0) {
    newTask.category = taskStore.getCategories[0].id
  }
  const taskData = {
    ...newTask,
    dueDate: newTask.deadline,
    userId: user.id // використовуємо ID авторизованого користувача
  }
  delete taskData.deadline
  taskStore.addTask(taskData)
  Object.assign(newTask, {
    title: "",
    deadline: format(new Date(), 'yyyy-MM-dd'),
    priority: 'medium',
    description: "",
    category: ""
  })
}
function deleteTask(taskId) {
  if (confirm('Ви впевнені, що хочете видалити це завдання?')) {
    taskStore.deleteTask(taskId)
  }
}

```

```

}
function toggleTaskComplete(taskId) {
  taskStore.toggleTaskComplete(taskId)
}
function openEditModal(task) {
  Object.assign(editTaskData, {
    id: task._id,
    title: task.title,
    deadline: task.dueDate,
    priority: task.priority,
    category: task.category,
    description: task.description || " ")
  isModalOpen.value = true
}
function closeModal() {
  isModalOpen.value = false
}
function saveEdit() {
  if (editTaskData.id) {
    taskStore.updateTask(editTaskData.id, {
      title: editTaskData.title,
      dueDate: editTaskData.deadline,
      priority: editTaskData.priority,
      category: editTaskData.category,
      description: editTaskData.description
    }) }
  closeModal()
}
</script>

```

Б.3 Текст файла analytics.js

```

function parseDate(d) {
  if (!d && d !== 0) return null
  if (d instanceof Date) return isNaN(d) ? null : d
  if (typeof d === 'number') {
    const ms = d < 1e11 ? d * 1000 : d
    const dt = new Date(ms)
    return isNaN(dt) ? null : dt
  }
  if (typeof d === 'object') {
    if (d.$date) return parseDate(d.$date)
    if (d.$numberLong) return parseDate(Number(d.$numberLong))
    if (d.date) return parseDate(d.date)
    return null
  }
  if (typeof d === 'string') {
    if (/^\d+$/.test(d)) {
      const n = Number(d)
      return parseDate(n)
    }
    const dt = new Date(d)
    return isNaN(dt) ? null : dt
  }
  return null
}

function startOfDay(d) {
  const dt = new Date(d)
  dt.setHours(0,0,0,0)
  return dt
}

```

```

}
function subDays(date, days) {
  if (!date) return null
  const dt = new Date(date)
  dt.setDate(dt.getDate() - days)
  return dt
}
function getDateFromTask(task, candidates) {
  if (!task) return null
  for (const key of candidates) {
    if (task[key] !== undefined && task[key] !== null) return
    parseDate(task[key])
  }
  return null
}
// Основна функція аналізу завдань
export function analyzeTasks(tasks = []) {
  const total = (tasks || []).length
  const completed = (tasks || []).filter(t => t.completed).length
  const completionRate = total > 0 ? Math.round((completed / total) * 100) :
0
  const todayStart = startOfDay(new Date())
  const overdue = (tasks || []).filter(t => {
    const due = getDateFromTask(t, ['dueDate', 'due_date', 'due', 'deadline'])
    return !t.completed && due && due < todayStart
  }).length
  const horizonDays = 14
  const since = new Date(); since.setDate(since.getDate() - horizonDays)
  const completedLastN = (tasks || []).filter(t => {

```

```

    const cd = getDateFromTask(t, ['completedAt', 'completed_at',
'completed'])
    return t.completed && cd && cd >= since
}).length
const avgPerDay = +(completedLastN / Math.max(1,
horizonDays)).toFixed(2)
// Тренд: останні 7 днів в порівнянні з попередніми 7 днями
const now = new Date()
const last7Start = subDays(now, 7)
const prev7Start = subDays(now, 14)
const completedLast7 = (tasks || []).filter(t => {
    const cd = getDateFromTask(t, ['completedAt', 'completed_at',
'completed'])
    return t.completed && cd && cd >= last7Start
}).length
const completedPrev7 = (tasks || []).filter(t => {
    const cd = getDateFromTask(t, ['completedAt', 'completed_at',
'completed'])
    return t.completed && cd && cd >= prev7Start && cd < last7Start
}).length
const avgLast7 = +(completedLast7 / 7).toFixed(2)
const avgPrev7 = +(completedPrev7 / 7).toFixed(2)
const trendFactor = avgPrev7 > 0 ? (avgLast7 / avgPrev7) : (avgLast7 > 0
? 1.5 : 1)
// Об'єкт з основними метриками
const metrics = {
    total,
    completed,
    completionRate,
    overdue,

```

```

    avgPerDay
  }
  // Генеруємо рекомендації на основі метрик
  const suggestions = []
  if (total === 0) {
    suggestions.push({ type: 'info', message: 'У вас немає задач: створіть задачу, щоб почати відстеження продуктивності.' })
    return { metrics, suggestions }
  }
  // Якщо низька продуктивність, рекомендуємо техніку Pomodoro
  if (completionRate < 60) {
    suggestions.push({ type: 'warning', message: `Низький відсоток виконання (${completionRate}%). Спробуйте починати день з 1-2 пріоритетних завдань (time-block 60-90 хв).` })
  } else {
    let suggestedPerDay = Math.max(1, Math.round(avgPerDay || 0))
    if (trendFactor > 1.1 && avgPerDay >= 1) {
      suggestedPerDay = Math.max(suggestedPerDay, Math.round(avgPerDay) + 1)
    }
    suggestions.push({ type: 'success', message: `Добре: ${completionRate}% завдань виконано. Збережіть темп – плануйте ${suggestedPerDay} завдань/день.` })
    if (overdue > 0) {
      suggestions.push({ type: 'warning', message: `Є ${overdue} прострочених завдань - відведіть 30-60 хв на їх оновлення або делегування.` })
    }
    suggestions.push({ type: 'info', message: 'Почніть з маленьких змін: фокус 60-90 хв, 1 раз в день, і тижневий обзор у п'ятницю.' })
  }

```

```

    return { metrics, suggestions } }
export default { analyzeTasks }

```

Б.4 Текст файлу Analytics.vue

```

<template>
  <div class="analytics-page" v-if="!taskStore.loading">
    <h1>Аналітика продуктивності</h1>
    <div class="analytics-grid">
      <div class="stats-card overall-stats">
        <h2>Загальна статистика</h2>
        <div class="stats-content">
          <div class="stat-item">
            <span class="stat-label">Всього завдань</span>
            <span class="stat-value">{{ taskStats.total }}</span>
          </div>
          <div class="stat-item">
            <span class="stat-label">Відсоток виконання</span>
            <span class="stat-value">{{ taskStats.rate }}%</span>
          </div>
          <div class="stat-item">
            <span class="stat-label">Виконано</span>
            <span class="stat-value success">{{ taskStats.completed }}</span>
          </div>
          <div class="stat-item">
            <span class="stat-label">Пропущено</span>
            <span class="stat-value overdue">{{ overdueCount }}</span>
          </div>
        </div>
      </div>
    </div>
  </div>
</template>

```

```

<!-- Графіки -->
<div class="chart-card category-chart">
  <h2>Продуктивність за категоріями</h2>
  <canvas id="categoryChart"></canvas>
</div>

<div class="chart-card priority-chart">
  <h2>Розподіл за пріоритетами</h2>
  <canvas id="priorityChart"></canvas>
</div>

<div class="chart-card trends-chart">
  <div class="trends-header">
    <h2>Тренди виконання</h2>
  </div>
  <div class="trends-controls">
    <button class="period-btn" @click="prevWeek">◀</button>
    <select v-model.number="periodWeeks" class="period-select"
title="Вибрати період">
      <option :value="1">1 тиждень</option>
      <option :value="2">2 тижні</option>
      <option :value="4">4 тижні</option>
    </select>
    <button class="period-btn" @click="nextWeek">▶</button>
  </div>
  <div class="trends-range">{{ weeklyRange }}</div>
  <canvas id="trendsChart"></canvas>
</div>
</div>

<!-- Інтелектуальний аналіз -->
<div class="insights-section">
  <h2>Інтелектуальний аналіз</h2>

```

```

<div class="insights-grid">
  <div class="insight-card" v-for="(suggestion, index) in suggestions"
:key="index"
    :class="suggestion.type">
    <i :class="getSuggestionIcon(suggestion.type)"></i>
    <p>{{ suggestion.message }}</p>
  </div>
</div>
</div>
<!-- Детальна статистика за категоріями -->
<div class="category-stats">
  <h2>Детальна статистика за категоріями</h2>
  <div class="category-grid">
    <div v-for="(stats, category) in productivityByCategory"
:key="category"
      class="category-card">
      <h3>{{ getCategoryName(category) }}</h3>
      <div class="category-stats-content">
        <div class="progress-circle"
          :style="{ background: `conic-gradient(var(--success-color)
${stats.rate}%, #f0f0f0 0)` }">
          <span>{{ stats.rate }}%</span>
        </div>
        <div class="category-details">
          <p>Всього: {{ stats.total }}</p>
          <p>Виконано: {{ stats.completed }}</p>
        </div>
      </div>
    </div>
  </div>
</div>

```

```

    </div>
  </div>
  <!-- Індикатор завантаження -->
  <div v-else>
    <p>Завантаження даних...</p>
  </div>
</template>
<script setup>
import { onMounted, ref, computed, watch, nextTick } from 'vue'
import { useTaskStore } from '../store'
import Chart from 'chart.js/auto'
import { subDays, eachDayOfInterval, format, isSameDay, isBefore,
startOfDay, startOfWeek, endOfWeek, addWeeks } from 'date-fns'
import { uk } from 'date-fns/locale'
import analyticsService from '../services/analytics'
const taskStore = useTaskStore()
const charts = ref({})
const weekOffset = ref(0) // 0 = current week, shifts in weeks
const periodWeeks = ref(1) // number of weeks to show in trends (1,2,4)
const taskStats = computed(() => {
  const stats = taskStore.getTasksStats
  return stats || { total: 0, completed: 0, rate: 0 }
})
// Кількість прострочених (не виконаних) завдань
const overdueCount = computed(() => {
  const tasks = taskStore.getTasks || []
  const todayStart = startOfDay(new Date())
  return tasks.filter(t => !t.completed && t.dueDate && isBefore(new
Date(t.dueDate), todayStart)).length
})

```

```

const productivityByCategory = computed(() => {
  const categories = taskStore.getCategories || []
  const tasks = taskStore.getTasks || []
  const result = {}
  categories.forEach(c => {
    const tasksInCategory = tasks.filter(t => t.category === c.id)
    const completed = tasksInCategory.filter(t => t.completed).length
    const total = tasksInCategory.length
    const rate = total > 0 ? Math.round((completed / total) * 100) : 0
    result[c.id] = { total, completed, rate }
  })
  return result
})

const metrics = ref({})
const suggestions = ref([])

const selectedWeekStart = computed(() => {
  const base = addWeeks(new Date(), weekOffset.value)
  return startOfWeek(base, { weekStartsOn: 1 })
})

const selectedWeekEnd = computed(() => {
  // end = start + periodWeeks weeks, minus one day
  const start = selectedWeekStart.value
  const end = subDays(addWeeks(start, periodWeeks.value), 1)
  return end
})

const weeklyRange = computed(() => {
  const start = selectedWeekStart.value
  const end = selectedWeekEnd.value
  return `${format(start, 'dd.MM')}-${format(end, 'dd.MM')}`
})

```

```

function prevWeek() { weekOffset.value -= Math.max(1,
periodWeeks.value) }
function nextWeek() { weekOffset.value += Math.max(1,
periodWeeks.value) }
// Методи
function getCategoryName(categoryId) {
  const category = taskStore.getCategories?.find(c => c.id === categoryId)
  return category ? category.name : categoryId
}
function getSuggestionIcon(type) {
  const icons = {
    warning: 'fas fa-exclamation-triangle',
    info: 'fas fa-info-circle',
    success: 'fas fa-check-circle'
  }
  return icons[type] || 'fas fa-lightbulb'
}
// Функції створення графіків
function createCategoryChart() {
  const ctx = document.getElementById('categoryChart')?.getContext('2d')
  if (!ctx) return null
  const categories = taskStore.getCategories || []
  const stats = productivityByCategory.value
  return new Chart(ctx, {
    type: 'bar',
    data: {
      labels: categories.map(c => c.name),
      datasets: [
        {
          label: 'Виконано',

```

```

      data: categories.map(c => stats[c.id]?.completed || 0),
      backgroundColor: 'rgba(75, 192, 192, 0.7)'
    },
    {
      label: 'Залишилось',
      data: categories.map(c => (stats[c.id]?.total || 0) -
(stats[c.id]?.completed || 0)),
      backgroundColor: 'rgba(255, 99, 132, 0.7)'
    }
  ]
},
options: {
  responsive: true,
  plugins: {
    legend: { position: 'top' },
    title: { display: false }
  },
  scales: {
    y: { beginAtZero: true, precision: 0 }
  }
})
}

function createPriorityChart() {
  const ctx = document.getElementById('priorityChart')?.getContext('2d')
  if (!ctx) return null
  const priorities = taskStore.getPriorities || []
  const tasks = taskStore.getTasks || []
  const data = priorities.map(priority => ({
    priority: priority.name,

```

```

    color: priority.color,
    count: tasks.filter(t => t.priority === priority.id).length
  )))
return new Chart(ctx, {
  type: 'doughnut',
  data: {
    labels: data.map(d => d.priority),
    datasets: [{
      data: data.map(d => d.count),
      backgroundColor: data.map(d => d.color)
    }]
  },
  options: {
    responsive: true,
    plugins: {
      legend: { position: 'right' }
    }
  }
})
}

function createTrendsChart() {
  const ctx = document.getElementById('trendsChart')?.getContext('2d')
  if (!ctx) return null
  // selected week: Monday - Sunday
  const start = selectedWeekStart.value
  const end = selectedWeekEnd.value
  const days = eachDayOfInterval({ start, end })
  const tasks = taskStore.getTasks || []
  // robust comparison: parse completedAt and use isSameDay
  const completedByDay = days.map(day => {

```

```

return tasks.filter(t => {
  if (!t.completed || !t.completedAt) return false
  const cd = new Date(t.completedAt)
  if (isNaN(cd)) return false
  return isSameDay(cd, day)
}).length
})
return new Chart(ctx, {
  type: 'line',
  data: {
    labels: days.map(d => format(d, 'EEE d', { locale: uk })),
    datasets: [{
      label: 'Виконані завдання',
      data: completedByDay,
      borderColor: 'rgb(75, 192, 192)',
      tension: 0.1
    }]
  },
  options: {
    responsive: true,
    plugins: {
      legend: { display: false }
    },
    scales: {
      y: { beginAtZero: true, precision: 0 }
    }
  }
})
}

```

```

watch(() => taskStore.getTasks, weekOffset, periodWeeks], ([newTasks])
=> {
  if (charts.value.trends) {
    try { charts.value.trends.destroy() } catch (e) { /* ignore */ }
    charts.value.trends = null }
  const tasks = taskStore.getTasks || []
  if (tasks.length > 0) {
    charts.value.trends = createTrendsChart()
  }
}, { deep: true })
onMounted(async () => {
  await taskStore.loadTasks()
  await nextTick()
  const tasks = taskStore.getTasks || []
  const analyzed = analyticsService.analyzeTasks(tasks)
  metrics.value = analyzed.metrics || {}
  suggestions.value = analyzed.suggestions || []
  if (tasks.length > 0) {
    charts.value.category = createCategoryChart()
    charts.value.priority = createPriorityChart()
    charts.value.trends = createTrendsChart()
  }
  taskStore.loaded = true
})
watch(() => taskStore.getTasks, (newTasks) => {
  const analyzed = analyticsService.analyzeTasks(newTasks || [])
  metrics.value = analyzed.metrics || {}
  suggestions.value = analyzed.suggestions || []
}, { deep: true })
</script>

```

ДОДАТОК В
Слайди презентації

Міністерство освіти і науки України
Національний університет «Запорізька політехніка»
Кафедра програмних засобів

**Розробка інтелектуального планувальника
персональних завдань**

Виконав

ст. групи КНТ – 114м

Микита ВОЛОШИН

Керівник

к.т.н. доцент

Тетяна ЗАЙКО

Рисунок В.1 – Слайд 1

2

- ▶ Об'єкт дослідження – процес розробки інтелектуального планувальника персональних завдань.
- ▶ Предмет дослідження – методи та програмні засоби для автоматизованого планування та управління персональними завданнями.
- ▶ Мета роботи – підвищення ефективності управління персональними завданнями, зокрема скорочення часу на їх планування та підвищення продуктивності користувача, шляхом дослідження методів інтелектуального планування та розробки програмного забезпечення інтелектуального планувальника персональних завдань.

Рисунок В.2 – Слайд 2

Завдання роботи:

3

1. Проаналізувати принципи роботи різних систем планування та управління завданнями, які використовуються користувачами;
2. Дослідити методи інтелектуального планування персональних завдань, які застосовують сучасні системи;
3. Виконати проєктування інтелектуального планувальника персональних завдань;
4. Реалізувати програмне забезпечення інтелектуального планувальника персональних завдань.

Рисунок В.3 – Слайд 3

Вибір мови програмування

4

Критерій	Vue.js	React.js
Підхід до побудови компонентів	Оптимізований, шаблони + JS	Повністю на JavaScript / JSX
Крива входження	Низька	Середня
Офіційні інструменти	Багато (Router, Pinia, CLI)	Менше, часто сторонні
Документація	Дуже структурована	Потужна, але більш технічна
Інтеграція в існуючі проєкти	Дуже проста	Середня
Робота з анімаціями	Зручні вбудовані можливості	Переважно сторонні бібліотеки
Розмір ядра	Малий	Середній
Гнучкість налаштувань проєкту	Висока	Дуже висока
Популярність у великих корпораціях	Середня	Дуже висока
Підтримка TypeScript	Відмінна (Vue 3)	Дуже добра

Рисунок В.4 – Слайд 4

Вибір середовища програмування

5

Характеристика	VS Code	Sublime Text
Вартість	Безкоштовний	Платна ліцензія (є пробна)
Швидкий старт без налаштувань	–	+
Споживання оперативної пам'яті	Вище	Дуже низьке
Частота оновлень	Дуже висока	Помірна
Робота з великими файлами	Середня	Дуже швидка
Підтримка Git	Вбудована	Через плагіни
Інструменти для веброзробки	Максимально розвинені	Базові
Персоналізація інтерфейсу	Висока	Середня
Пошук по проєкту	Швидкий	Дуже швидкий
Автоматичні форматтери (Prettier тощо)	Легко інтегруються	Через сторонні плагіни

Рисунок В.5 – Слайд 5

Структура додатку

6



Рисунок В.6 – Слайд 6

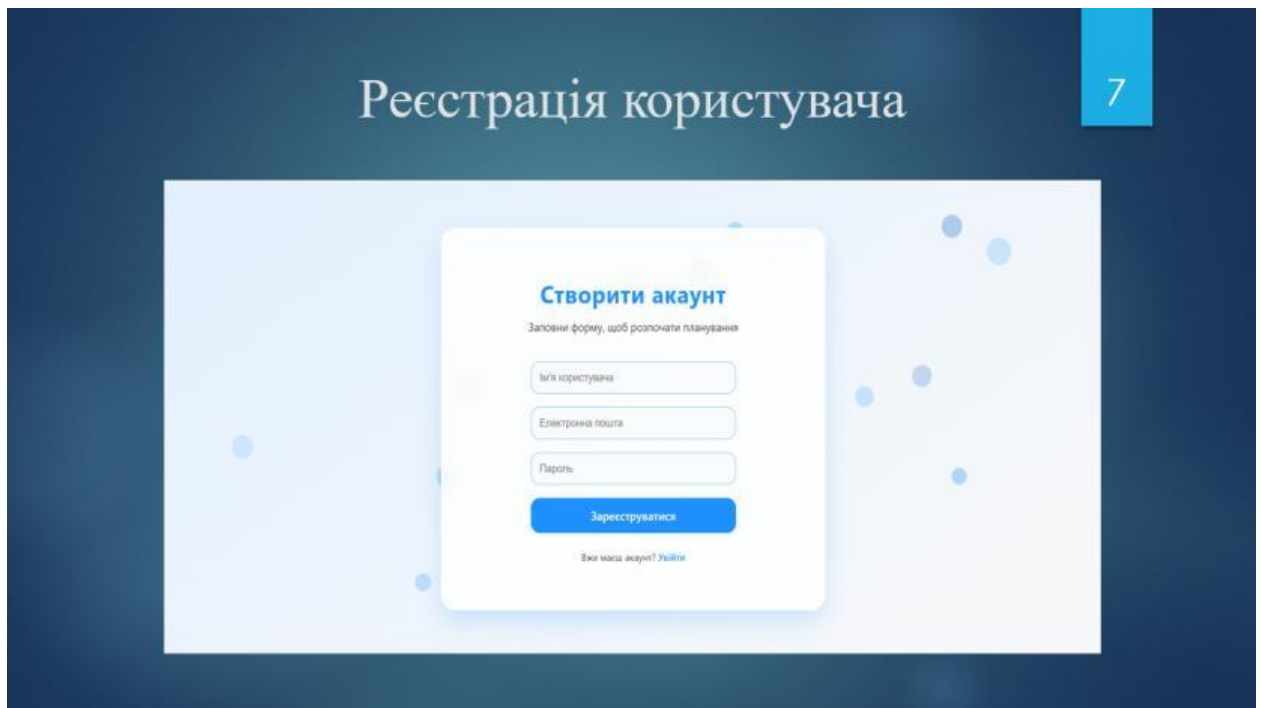


Рисунок В.7 – Слайд 7

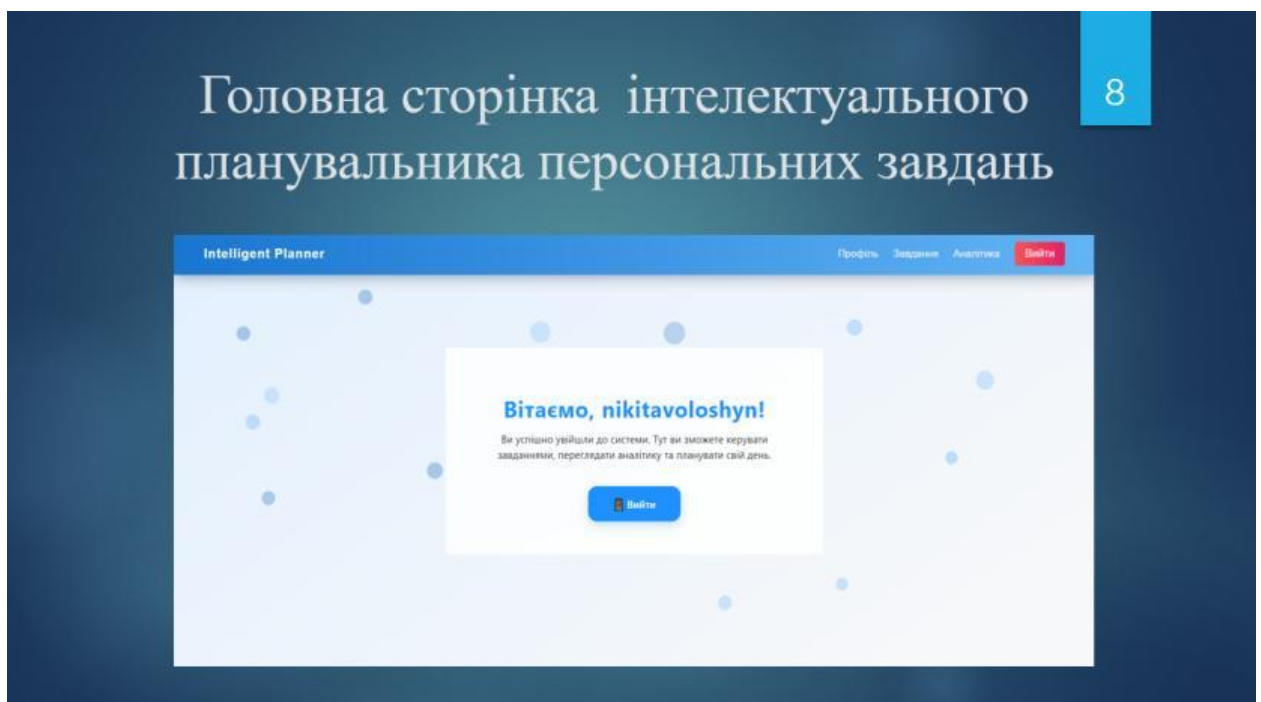


Рисунок В.8 – Слайд 8

Список завдань для користувача

9

Intelligent Planner

Профіль Завдання Аналітика Вийти

Список завдань

Нижній Усі категорії Пошук завдань...

Нижній 27.11.2025 Середній Робота

Опис завдання (необов'язково) **Додати завдання**

Зустріч
 Ді: 15-е листопада 2025 р. | Пріоритет: Нижній | Категорія: Інша | 10:00
 [Роздрукувати] [Видалити]

Малювання
 Ді: 15-е листопада 2025 р. | Пріоритет: Нижній | Категорія: Малювання
 [Роздрукувати] [Видалити]

Рисунок В.9 – Слайд 9

Фільтрація та пошук завдання

10

Intelligent Planner

Профіль Завдання Аналітика Вийти

Список завдань

Середній Особисте Прогулянка

Нижній 27.11.2025 Середній Робота

Опис завдання (необов'язково) **Додати завдання**

Прогулянка
 Ді: 12-е листопада 2025 р. | Пріоритет: Середній | Категорія: Особисте | 10:00
 [Роздрукувати] [Видалити]

Рисунок В.10 – Слайд 10

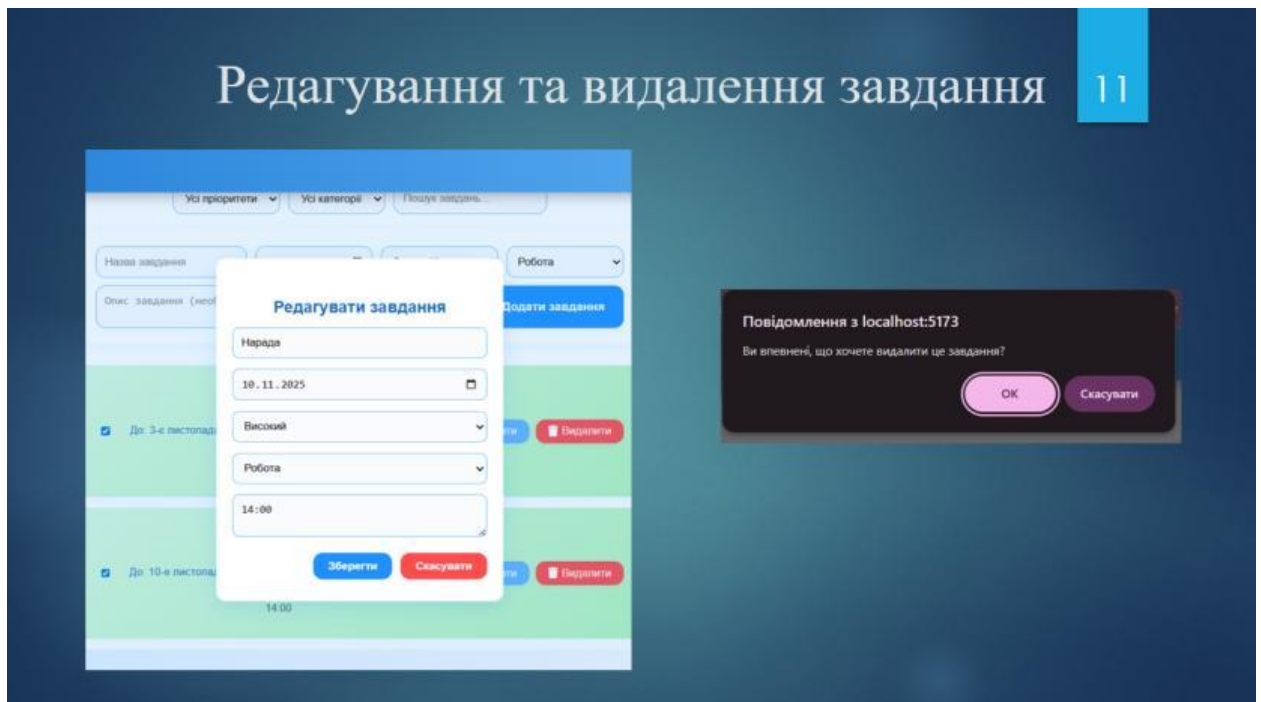


Рисунок В.11 – Слайд 11



Рисунок В.12 – Слайд 12

Інтелектуальний аналіз та статистика 13

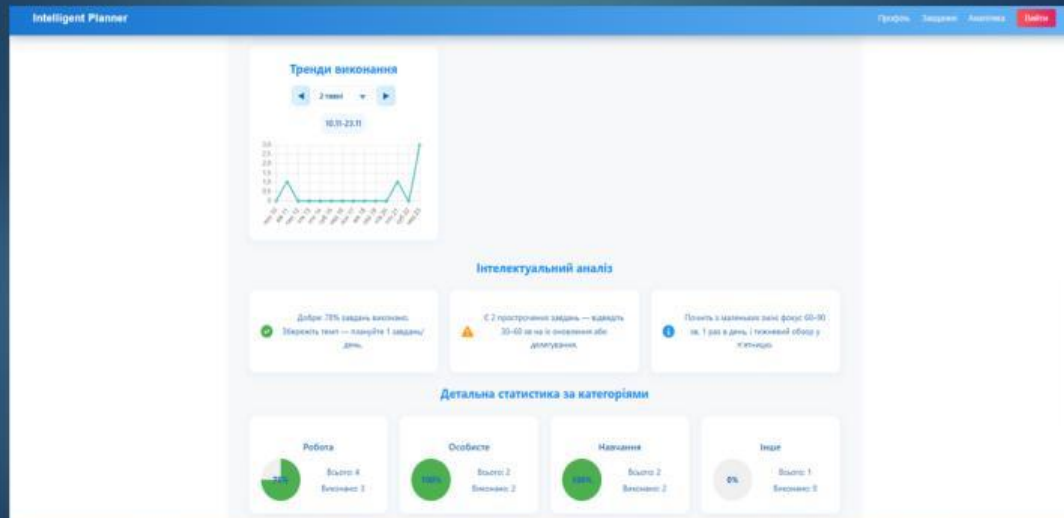


Рисунок В.13 – Слайд 13

Висновки

14

1. У ході виконання дипломної кваліфікаційної роботи магістра було проаналізовано та досліджено методи інтелектуального планування персональних завдань і розроблено програмне забезпечення для їх автоматизованого управління.
2. У межах роботи запропоновано метод інтелектуального планування персональних завдань, заснований на аналізі даних користувача щодо термінів виконання, рівня пріоритетності, тривалості та історії виконання завдань. Алгоритм роботи системи здійснює обробку зазначених параметрів, розраховує індивідуальні пріоритетні коефіцієнти та автоматично формує оптимізований перелік завдань для користувача з урахуванням дедлайнів та поточного навантаження. У разі перевищення допустимих часових меж система генерує сповіщення та рекомендації щодо перерозподілу завдань.

Рисунок В.14 – Слайд 14

Висновки

15

3. Результати роботи планувальника візуалізуються у вигляді аналітичних панелей та графіків, що відображають рівень виконання завдань, розподіл часу та динаміку щоденної продуктивності. Це дозволяє користувачеві наочно оцінювати ефективність власного планування та коригувати стратегію управління часом. Ефективність запропонованого методу може бути перевірена шляхом емпіричного аналізу статистики використання системи, зокрема за показниками зростання виконаних завдань, зменшення прострочених дедлайнів та підвищення регулярності планування.

4. Створено повнофункціональне програмне забезпечення інтелектуального планувальника та підготовлено інструкцію з його використання. Проведене тестування підтвердило коректність роботи усіх основних модулів системи та відповідність програмного продукту вимогам технічного завдання. За результатами перевірки встановлено, що розроблений планувальник демонструє високі показники ефективності у порівнянні з існуючими аналогами та забезпечує зручний інструментарій для підвищення особистої продуктивності користувачів.

Рисунок В.15 – Слайд 15