ЛЬВІВСЬКИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук, прикладної та вищої математики

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Проєктування та реалізація вебсервісу для оптимізації планування робочими завданнями з допомогою ШІ

Виконав: здобувач першого (бакалаврського)
рівня вищої освіти
ОПП "Комп'ютерні науки"
спеціальності 122 "Комп'ютерні науки"
_____ Заришняк Вадим Владиславович
(підпис)

Науковий керівник:
_____ Рехлецький Євген Анатолійович
(підпис)

Львів - 2026

ЛЬВІВСЬКИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Факультет міжнародних економічних відносин та інформаційних технологій
Кафедра комп'ютерних наук, прикладної та вищої математики
Перший (бакалаврський) рівень вищої освіти
Освітньо-професійна програма "Комп'ютерні науки"
Спеціальність 122 "Комп'ютерні науки"

"**ЗАТВЕРДЖУЮ**"
Завідувач кафедри комп'ютерних та наук, прикладної та вищої математики

_____ Новосад З. Г.
(підпис) (прізвище та ініціали)

13.10.2025 р.

ЗАВДАННЯ
до кваліфікаційної роботи здобувачу
Заришняку Вадиму Владиславовичу

1. Тема проєкту «Проєктування та реалізація вебсервісу для оптимізації планування робочими завданнями з допомогою ШІ»;
   керівник кваліфікаційної роботи Рехлецький Євген Анатолійович, к. е. н., доц.;
   затверджена наказом по Університету ***№ 991/03 від 06 жовтня 2025 р.***
2. Термін здачі здобувачем проєкту 05 червня 2025 р.
3. Вхідні дані до проєкту: література, документація, друковані матеріали, сайти Internet, що пов'язані з дослідженням сучасного стану предметної області; аналогічні дані по платформах, що дозволяють на них реалізацію проєкту.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
   1.Теоретичні засади предметної області.
   2. Вибір можливих платформ для реалізації проєкту, розробка інструментарію обраних платформ.
   3. Розробка моделі та реалізація проєкту.
   Перелік графічного матеріалу: до 11 слайдів у форматі PowerPoint
5. Консультанти розділів проєкту

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

6. Дата видачі завдання 13.10.2025 р.

Календарний план проєкту

№ з/п	Найменування етапів робіт	Термін виконання	Примітка
1.	Отримання завдання і аналіз інформаційних джерел за темою проєкту.	до 11.12.2024	
2.	Дослідження існуючих платформ управління завданнями; створення розгорнутого плану роботи.	до 13.01.2025	
3.	Аналіз та вибір технологічного стеку (React, Vite, Zustand, Groq API); формування списку літературних джерел.	до 07.02.2025	
4.	Проєктування архітектури застосунку та моделі даних; налаштування структури проєкту.	до 01.04.2025	
5.	Реалізація компонентів інтерфейсу та інтеграція Groq API; розробка чорнової версії застосунку.	до 15.04.2025	
6.	Тестування реалізованого проєкту.	до 22.04.2025	
7.	Оформлення кваліфікаційної роботи. Подання проєкту на перевірку керівнику	до 16.05.2025	
8.	Доопрацювання кваліфікаційної роботи з врахуванням зауважень керівника. Подання проєкту для перевірки в системі “Антиплагіат”	до 20.05.2025	
9.	Подання кваліфікаційної роботи керівнику для повторної перевірки і отримання відгуку.	до 27.05.2025	
10.	Представлення проєкту на зовнішнє рецензування. Подання кваліфікаційної роботи на кафедру комп'ютерних наук	до 5.06.2025	
11.	Попередній захист кваліфікаційної роботи	05.06.2025-06.06.2025	
12.	Захист кваліфікаційної роботи на засіданні кваліфікаційної комісії	15.06.2026 - 19.06.2026 (згідно графіку роботи ЕК)	

Здобувач вищої освіти _____ Заришняк В.В.

Керівник роботи _____ Рехлецький Є.А.

РЕФЕРАТ

У кваліфікаційній роботі представлено процес проєктування та реалізації веб-сервісу для оптимізації планування робочих завдань з використанням штучного інтелекту. Розглянуто теоретичні основи систем управління завданнями, проведено аналіз існуючих платформ та обґрунтовано вибір технологій. Реалізовано клієнтську частину застосунку на основі бібліотеки React 18, системи управління станом Zustand та інструменту збірки Vite. Застосунок включає Kanban-дошку з фільтрацією за пріоритетом, режим списку з сортуванням за AI-score, аналітичний модуль та AI-асистента на базі моделі Llama 3.3 70B через Groq API, здатного аналізувати завдання та надавати рекомендації українською мовою.

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

Об'єктом дослідження є процес проєктування та розробки веб-сервісу для управління робочими завданнями з інтегрованим модулем штучного інтелекту.

Предметом дослідження є методи та технології реалізації інтелектуального веб-застосунку для планування завдань на основі React, Zustand та Groq API.

Метою дослідження є проєктування та практична реалізація повнофункціонального веб-сервісу для оптимізації планування робочих завдань із сучасним інтерфейсом та AI-функціональністю.

Ключові слова: вебсервіс, штучний інтелект, планування завдань, оптимізація, управління проєктами, машинне навчання, API.

SUMMARY

This qualification work presents the process of designing and implementing a web service for optimizing work task planning using artificial intelligence. The theoretical foundations of task management systems are examined, existing platforms are analyzed, and the technology choices are justified. The client-side of the application is built using React 18, Zustand for state management, and Vite as the build tool. The application includes a Kanban board with priority filtering, a list view sorted by AI-score, an analytics module, and an AI assistant powered by the Llama 3.3 70B model via Groq API, capable of analyzing tasks and providing recommendations in Ukrainian.

The work consists of an introduction, three chapters, conclusions, a list of references, and appendices.

The object of the research is the process of designing and developing a web service for work task management with an integrated artificial intelligence module.

The subject of the research is the methods and technologies for implementing an intelligent web application for task planning based on React, Zustand, and Groq API.

The aim of the research is to design and practically implement a fully

functional web service for optimizing work task planning with a modern interface and AI functionality.

KeyWords: web service, artificial intelligence, task scheduling, optimization, project management, recommendation system, machine learning, API.

ЗМІСТ

ВСТУП 7

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ПРОЄКТУВАННЯ ВЕБ-СЕРВІСІВ ДЛЯ УПРАВЛІННЯ ЗАВДАННЯМИ 10

1.1. Поняття та класифікація систем управління завданнями . 10

1.2. Роль штучного інтелекту в оптимізації планування робочих процесів 14

1.3. Огляд існуючих рішень та аналіз аналогів 18

РОЗДІЛ 2: ВИБІР ТЕХНОЛОГІЙ ТА ПРОЄКТУВАННЯ ВЕБ-СЕРВІСУ 25

2.1. Обґрунтування вибору технологічного стеку 25

2.2. Архітектура застосунку та моделі даних 31

2.3. Проєктування інтерфейсу користувача та UX/UI-рішень 37

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ВЕБ-СЕРВІСУ 46

3.1. Реалізація клієнтської частини на основі React 46

3.2. Управління станом застосунку засобами Zustand 53

3.3. Інтеграція AI-модуля на базі Groq API 60

ВИСНОВКИ 68

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ 71

ДОДАТКИ 74

ВСТУП

Сучасне ділове середовище характеризується постійно зростаючим обсягом завдань, проєктів та інформаційних потоків, якими щодня доводиться керувати як окремим фахівцям, так і цілим командам. Ефективне планування робочого часу та правильна розстановка пріоритетів стають критично важливими навичками в умовах, коли кількість одночасно активних задач перевищує можливості людської уваги. Саме тому інструменти управління завданнями перетворились із допоміжних утиліт на повноцінні стратегічні рішення для підвищення продуктивності.

Паралельно з цим стрімкий розвиток технологій штучного інтелекту відкриває принципово нові можливості для автоматизації рутинних когнітивних процесів. Сучасні великі мовні моделі (Large Language Models, LLM) здатні аналізувати контекст, виявляти залежності між завданнями, оцінювати ризики дедлайнів та формулювати конкретні рекомендації щодо порядку виконання робіт. Інтеграція таких моделей безпосередньо у веб-сервіси управління завданнями дозволяє перейти від пасивного відображення списку задач до активного AI-асистування у прийнятті рішень.

Актуальність теми кваліфікаційної роботи обумовлена поєднанням двох чинників. По-перше, попит на зручні та інтелектуальні інструменти планування продовжує зростати як серед індивідуальних користувачів, так і серед малих команд, яким не підходять громіздкі корпоративні системи. По-друге, поява доступних безкоштовних API для роботи з великими мовними моделями — зокрема Groq API з моделлю Llama 3.3 70B — робить практичну реалізацію AI-функціональності досяжною навіть у межах навчального проєкту. Таким чином, тема відповідає актуальним тенденціям розвитку інформаційних технологій та має очевидне практичне застосування.

Об'єктом дослідження є процес проєктування та розробки веб-сервісу для управління робочими завданнями з інтегрованим модулем штучного інтелекту.

Предметом дослідження є методи, технології та архітектурні підходи до реалізації інтелектуального веб-застосунку для планування завдань на основі бібліотеки React, системи управління станом Zustand та Groq API.

Метою кваліфікаційної роботи є проєктування та практична реалізація повнофункціонального веб-сервісу для оптимізації планування робочих завдань, що поєднує зручний сучасний інтерфейс із можливістю AI-аналізу, пріоритизації та рекомендацій на основі реальних даних користувача.

Для досягнення поставленої мети визначено такі завдання:

- дослідити теоретичні засади управління завданнями та роль штучного інтелекту в оптимізації робочих процесів;

- провести аналіз існуючих рішень у сфері task management та визначити їхні переваги, недоліки та функціональні стандарти галузі;

- здійснити порівняльний аналіз технологій та обрати оптимальний стек для реалізації проєкту;

- спроєктувати архітектуру веб-сервісу, структуру компонентів, модель даних та логіку управління станом;

- розробити та реалізувати інтерфейс користувача з Kanban-дошкою, списком завдань та аналітичним модулем;

- інтегрувати Groq API для забезпечення функціональності AI-асистента, здатного аналізувати завдання та надавати рекомендації;

- перевірити коректність роботи розробленого застосунку та оцінити якість реалізованого рішення.

Практичне значення роботи полягає в тому, що розроблений застосунок TaskAI є повнофункціональним веб-сервісом, готовим до реального використання. Він поєднує зручний Kanban-інтерфейс із живим AI-асистентом на базі моделі Llama 3.3 70B, що дозволяє користувачам не лише відстежувати завдання, а й отримувати конкретні рекомендації щодо пріоритетів та ризиків дедлайнів. Розроблена архітектура на базі React та Zustand є масштабованою та може слугувати основою для подальшого розширення функціональності.

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі розглядаються теоретичні засади управління завданнями та роль штучного інтелекту в оптимізації планування. У другому розділі здійснюється вибір технологічного стеку та проєктування системи. У третьому розділі описується практична реалізація веб-сервісу та результати його тестування. Завершують роботу висновки та список літературних джерел.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ПРОЄКТУВАННЯ ВЕБ-СЕРВІСІВ ДЛЯ УПРАВЛІННЯ ЗАВДАННЯМИ

1.1. Поняття та класифікація систем управління завданнями

Управління завданнями (task management) є однією з ключових дисциплін у сфері організації праці та підвищення продуктивності. В широкому розумінні під управлінням завданнями розуміють сукупність методів, інструментів та практик, спрямованих на планування, відстеження, пріоритизацію та успішне завершення одиниць роботи — завдань — у визначені терміни та відповідно до встановлених критеріїв якості. Незважаючи на удавану простоту, ця дисципліна має глибоке теоретичне підґрунтя та тісно пов'язана з такими науковими галузями, як теорія організації, когнітивна психологія, інформаційний менеджмент та інженерія програмного забезпечення.

З точки зору інформаційних технологій система управління завданнями — це програмний продукт або веб-сервіс, що забезпечує користувачеві можливість створювати завдання, призначати їм атрибути (пріоритет, дедлайн, категорію, виконавця), відстежувати їхній поточний стан та аналізувати прогрес виконання. Сучасні системи цього класу, як правило, реалізують значно ширший набір функцій: інтеграцію з іншими інструментами, командну співпрацю, автоматизацію рутинних операцій, аналітичні звіти та, дедалі частіше, функціональність на основі штучного інтелекту.

Еволюція систем управління завданнями пройшла кілька характерних етапів. На першому етапі, що охоплює кінець 1980-х — початок 2000-х років, домінували настільні застосунки, орієнтовані на індивідуального користувача: Microsoft Outlook із вбудованим модулем завдань, Palm Pilot та подібні рішення для персонального планування. Ці інструменти були ізольованими,

не передбачали командної роботи та мали обмежені можливості для синхронізації даних між пристроями.

Другий етап розпочався з масовим поширенням широкосмугового інтернету та хмарних обчислень у 2000-х роках. Саме тоді з'явились перші веб-орієнтовані системи управління завданнями — Basecamp (2004), Remember the Milk (2004), Toodledo (2006) — які перенесли дані у хмару та зробили можливою синхронізацію між пристроями та спільний доступ до завдань у командах [30]. Цей перехід принципово змінив парадигму: з персонального інструменту система управління завданнями перетворилась на платформу для колаборації.

Третій, актуальний на сьогодні, етап пов'язаний із мобільною революцією, появою методологій гнучкої розробки (Agile, Scrum, Kanban) та інтеграцією штучного інтелекту. Сучасні лідери ринку — Notion, ClickUp, Linear, Asana — пропонують комплексні робочі простори, що поєднують управління завданнями з базами знань, документами, автоматизацією та AI-функціональністю в єдиному середовищі [30].

Класифікація систем управління завданнями

Різноманітність існуючих рішень у цій сфері зумовлює потребу в їхній систематизації. Розглянемо основні підходи до класифікації.

За цільовою аудиторією системи управління завданнями поділяються на три групи. Персональні системи орієнтовані на індивідуального користувача та зосереджені на особистій продуктивності: Todoist, Things 3, Microsoft To Do. Командні системи призначені для спільної роботи невеликих груп і забезпечують розподіл завдань між виконавцями, відстеження прогресу кожного учасника та комунікацію в контексті завдань: Trello, Asana, Basecamp. Корпоративні системи розраховані на великі організації з розгалуженою ієрархією, складними процесами затвердження та інтеграцією з ERP-системами: Jira, ServiceNow, Microsoft Project.

За моделлю відображення завдань виділяють кілька архетипів. Списочна модель (list-based) є найпростішою і найпоширенішою: завдання

відображаються у вигляді ієрархічного списку з можливістю вкладення підзавдань. Kanban-модель, що бере свій початок з виробничої системи Toyota, організовує завдання у вигляді карток, що переміщуються між колонками, які відповідають етапам робочого процесу. Calendar-модель відображає завдання в прив'язці до дат та часових слотів, що зручно для планування з жорсткими дедлайнами. Matrix-модель, заснована на матриці Ейзенгауера, класифікує завдання за двома осями — терміновість та важливість — і розподіляє їх по чотирьох квадрантах. Комбінована модель, яку реалізують сучасні платформи на кшталт Notion та ClickUp, дозволяє користувачеві обирати між різними режимами відображення тих самих даних.

За способом розгортання системи поділяються на хмарні (SaaS), що не потребують встановлення та доступні через браузер з будь-якого пристрою, локальні, що розгортаються на власній інфраструктурі організації, та гібридні, що поєднують обидва підходи. Для сучасних веб-сервісів домінуючою є хмарна модель.

За ступенем інтелектуалізації — критерій, що набуває дедалі більшого значення, — системи можна розподілити на традиційні, де вся логіка пріоритизації покладається на користувача, системи з елементами автоматизації, що виконують прості правила (наприклад, автоматично позначають прострочені завдання), та інтелектуальні системи, що використовують методи машинного навчання або великі мовні моделі для аналізу завдань, прогнозування термінів виконання та формування рекомендацій. Класифікацію систем управління завданнями за основними критеріями наведено у таблиці 1.1.

Таблиця 1.1

Класифікація систем управління завданнями

Критерій класифікації	**Тип**	**Приклади**
Цільова аудиторія	Персональні	Todoist, Things 3, Microsoft To Do
	Командні	Trello, Asana, Basecamp
	Корпоративні	Jira, ServiceNow, MS Project
Модель відображення	Списочна	Todoist, Microsoft To Do
	Kanban	Trello, Jira
	Календарна	Google Tasks, Fantastical
	Комбінована	Notion, ClickUp, Linear
Спосіб розгортання	Хмарна (SaaS)	Asana, Trello, Notion
	Локальна	Jira Server, Redmine
	Гібридна	Confluence, GitLab
Ступінь інтелектуалізації	Традиційна	Trello, Basecamp
	З автоматизацією	Asana, Monday.com
	Інтелектуальна	Notion AI, ClickUp AI, Motion

Розроблюваний у даній кваліфікаційній роботі веб-сервіс TaskAI належить до категорії командно-персональних систем з комбінованою моделлю відображення (Kanban-дошка, список, аналітика), хмарним способом розгортання та високим ступенем інтелектуалізації завдяки інтеграції великої мовної моделі Llama 3.3 70B через Groq API.

Ключовою відмінністю сучасних систем управління завданнями від їхніх попередників є зміщення акценту з фіксації стану завдань на активне сприяння їх виконанню. Якщо традиційні інструменти виступали пасивним сховищем інформації, то сучасні платформи прагнуть стати інтелектуальними помічниками, що попереджають про ризики, підказують оптимальний порядок дій та автоматизують рутинні операції з планування. Саме цей вектор розвитку визначає архітектурні та функціональні рішення, прийняті при проєктуванні розроблюваного сервісу.

1.2. Роль штучного інтелекту в оптимізації планування робочих процесів

Штучний інтелект як інструмент підвищення продуктивності праці перетворився з теоретичної концепції на практично застосовуване рішення впродовж останніх кількох років. Ключовим каталізатором цього переходу стала поява великих мовних моделей — нового класу систем штучного інтелекту, здатних розуміти природну мову, аналізувати контекст та генерувати змістовні відповіді на складні запити. На відміну від вузькоспеціалізованих алгоритмів машинного навчання, що потребують великих обсягів розмічених тренувальних даних для кожної конкретної задачі, великі мовні моделі є універсальними інструментами, застосовними до широкого кола завдань без додаткового навчання [19].

У контексті управління завданнями штучний інтелект може виконувати принципово різні ролі залежно від рівня інтеграції та складності застосованих підходів. Розглянемо ці ролі детально, від найпростіших до найскладніших.

Автоматична категоризація та тегування завдань. Найпростіший рівень застосування штучного інтелекту полягає в автоматичному визначенні категорії, пріоритету та релевантних тегів для нового завдання на основі аналізу його назви та опису. Замість того щоб вручну призначати кожному завданню категорію «Backend», «Frontend» чи «DevOps», система може автоматично визначати її за ключовими словами та контекстом. Хоча такий підхід не потребує повноцінних великих мовних моделей і може бути реалізований класичними методами класифікації тексту, сучасні LLM роблять його значно точнішим та гнучкішим.

Інтелектуальна пріоритизація завдань. Більш складна роль штучного інтелекту полягає в аналізі всього списку завдань та формуванні рекомендацій щодо порядку їх виконання. При цьому враховуються не лише очевидні параметри — дедлайн та заявлений пріоритет, — а й менш формалізовані фактори: залежності між завданнями, поточний контекст та

навантаження, стратегічна важливість тих чи інших напрямів роботи. Саме цю роль відіграє AI-асистент у розроблюваному сервісі TaskAI: при зверненні користувача модель отримує повний список завдань із їхніми атрибутами та формулює конкретні рекомендації щодо того, що варто виконати першим і чому.

Виявлення ризиків та попередження. Штучний інтелект здатний проактично виявляти потенційні проблеми у плані завдань ще до того, як вони стануть критичними. Аналізуючи дедлайни, поточний прогрес та типові темпи виконання, система може попереджати про завдання з реалістичним ризиком прострочення, виявляти «вузькі місця» в робочому процесі та рекомендувати перерозподіл зусиль. Цей аспект особливо цінний для командної роботи, де один блокуючий елемент може гальмувати виконання цілого ланцюжка залежних завдань.

Декомпозиція складних завдань. Великі мовні моделі здатні допомагати у розбитті масштабних, погано визначених завдань на конкретні виконувані підзавдання. Якщо користувач створює завдання на кшталт «Розробити систему автентифікації», AI-асистент може запропонувати детальний план із підзавдань: проєктування схеми бази даних, реалізація JWT-токенів, написання middleware для перевірки прав, розробка форм входу та реєстрації, тестування. Така декомпозиція перетворює розпливчасту мету на чіткий виконуваний план.

Природномовний інтерфейс до даних. Замість того щоб вручну переглядати списки та застосовувати фільтри, користувач може просто запитати систему у вільній формі: «Що я маю зробити сьогодні?», «Які завдання найбільше ризикують не вкластись у дедлайн?», «Що блокує завершення модуля оплати?». Обробка таких запитів є природною сильною стороною великих мовних моделей і забезпечує значно нижчий когнітивний бар'єр для отримання потрібної інформації порівняно з традиційними інтерфейсами фільтрації.

Технічні підходи до інтеграції AI у веб-сервіси

З технічної точки зору інтеграція штучного інтелекту у веб-застосунки може здійснюватись за кількома принципово різними схемами, кожна з яких має свої переваги та обмеження.

Перший підхід — власне навчання та розгортання моделі — передбачає тренування або тонке налаштування (fine-tuning) моделі на власних даних з подальшим її розгортанням на власній інфраструктурі. Цей підхід забезпечує максимальний контроль над поведінкою моделі та конфіденційністю даних, однак вимагає значних обчислювальних ресурсів, експертизи в галузі машинного навчання та суттєвих фінансових витрат. Для більшості прикладних розробок він є недоцільним.

Другий підхід — використання API зовнішніх провайдерів — є домінуючим у сучасній індустрії. Розробник отримує доступ до потужних мовних моделей через стандартизований HTTP API без необхідності піклуватись про інфраструктуру, оновлення моделей та масштабування. Провідними провайдерами таких послуг є OpenAI (моделі серії GPT та o1), Anthropic (моделі Claude), Google (моделі Gemini) та Meta (відкриті моделі серії Llama).

Особливе місце серед провайдерів займає Groq — компанія, що спеціалізується на надшвидкому інференсі мовних моделей завдяки власним спеціалізованим процесорам LPU (Language Processing Unit). Groq API надає доступ до відкритих моделей, зокрема Llama 3.3 70B від Meta та Mixtral від Mistral AI, з затримкою відповіді в десятки разів меншою порівняно з GPU-базованими рішеннями. Безкоштовний тарифний план Groq дозволяє виконувати до 30 запитів на хвилину та до 14 400 запитів на добу, що цілком достатньо для навчальних та прототипних застосунків. Саме Groq API обрано для реалізації AI-функціональності у розроблюваному сервісі TaskAI з огляду на поєднання безкоштовного доступу, високої швидкості відповідей та якості моделі Llama 3.3 70B [18].

Третій підхід — використання відкритих моделей з локальним розгортанням через такі інструменти як Ollama або LM Studio — є компромісом між першим і другим: розробник розгортає готову відкриту модель на власному обладнанні без необхідності її тренування. Цей підхід забезпечує повну конфіденційність даних та відсутність залежності від зовнішніх сервісів, однак вимагає достатньо потужного апаратного забезпечення для прийнятної швидкості роботи.

Метрики ефективності AI-асистування у плануванні

Оцінка ефективності впровадження штучного інтелекту в системи управління завданнями є важливим, однак складним питанням. На відміну від традиційних програмних функцій, де ефективність вимірюється об'єктивними показниками продуктивності, якість AI-рекомендацій залежить від суб'єктивних факторів та контексту конкретного користувача. Проте дослідники та практики виділяють кілька ключових метрик.

Точність пріоритизації вимірює відповідність між рекомендованим AI порядком виконання завдань та оптимальним порядком, визначеним ретроспективно після завершення проєкту. Дослідження компанії McKinsey свідчать, що впровадження AI-асистування у робочі процеси планування скорочує час, витрачений на організаційні рішення, на 20–30%, вивільняючи його для безпосереднього виконання завдань [26].

Рівень прийняття рекомендацій відображає, яку частину AI-підказок користувачі фактично виконують. Цей показник є індикатором релевантності та якості рекомендацій: якщо рівень прийняття низький, це свідчить про те, що система не враховує важливих контекстуальних факторів.

Зниження кількості прострочених завдань є найбільш практично значущою метрикою. Системи з активним AI-моніторингом дедлайнів демонструють скорочення частки прострочених завдань на 15–25% порівняно з традиційними інструментами, де відповідальність за відстеження термінів повністю покладається на користувача.

Таким чином, штучний інтелект трансформує системи управління завданнями з пасивних інструментів фіксації інформації на активних помічників, здатних аналізувати ситуацію, виявляти ризики та пропонувати конкретні рекомендації. Розуміння різних рівнів та підходів до AI-інтеграції є необхідною теоретичною основою для обґрунтування архітектурних рішень, прийнятих у наступних розділах даної кваліфікаційної роботи.

1.3. Огляд існуючих рішень та аналіз аналогів

Перед проєктуванням будь-якого програмного продукту необхідно провести ґрунтовний аналіз існуючих рішень у відповідній предметній галузі. Такий аналіз дозволяє виявити функціональні стандарти ринку, визначити сильні та слабкі сторони наявних платформ, уникнути повторення відомих помилок та сформулювати конкурентні переваги розроблюваного рішення. У контексті даної кваліфікаційної роботи розглянемо як міжнародні платформи-лідери ринку управління завданнями, так і їхні характеристики з точки зору інтеграції штучного інтелекту.

Trello

Trello є одним із найвпізнаваніших інструментів управління завданнями у світі, заснованим у 2011 році компанією Fog Creek Software та придбаним Atlassian у 2017 році. Платформа побудована виключно на Kanban-моделі відображення: завдання представлені картками, що переміщуються між колонками дошки відповідно до поточного стану виконання. Простота концепції стала головною конкурентною перевагою Trello — новий користувач може освоїти базовий функціонал за лічені хвилини без жодного навчання.

З технічної точки зору Trello реалізований як SPA-застосунок на базі React із серверною частиною на Node.js [14]. Платформа підтримує розширення функціональності через систему Power-Ups — плагінів, що додають інтеграції з іншими сервісами, автоматизацію та додаткові елементи інтерфейсу. Саме через Power-Ups реалізована базова AI-функціональність:

Atlassian Intelligence, доступний у преміальних тарифах, забезпечує автоматичне заповнення описів карток, пропозиції щодо чеклістів та базовий аналіз дошки.

Серед недоліків Trello варто відзначити відсутність нативного відображення у вигляді списку або календаря без додаткових Power-Ups, обмежену глибину аналітики в безкоштовному тарифі, а також те, що AI-функціональність доступна лише на платних планах починаючи від 10 доларів на місяць за користувача.

Asana

Asana, заснована у 2008 році співзасновником Facebook Дастіном Московіцем, є одним із найфункціональніших інструментів командного управління завданнями на ринку. На відміну від Trello, Asana підтримує множинні режими відображення — список, дошка, календар, часова шкала та портфоліо проєктів — що робить її придатною для значно ширшого кола сценаріїв використання.

Ключовою технологічною перевагою Asana є розвинена система правил автоматизації, що дозволяє налаштовувати складні тригери та дії без написання коду: наприклад, автоматично призначати завдання конкретному виконавцю при переміщенні картки у певну колонку або надсилати сповіщення при наближенні дедлайну. Asana Intelligence, представлена у 2023 році, забезпечує AI-підсумки проєктів, автоматичне оновлення статусів та інтелектуальні відповіді на запитання про стан завдань природною мовою.

Суттєвим обмеженням Asana є висока вартість: повноцінний доступ до AI-функцій та розширеної аналітики доступний лише на тарифах Business та Enterprise, вартість яких становить від 24,99 долара на місяць за користувача. Безкоштовний тарифний план суттєво обмежує кількість учасників проєкту та доступні функції.

Notion

Notion є найбільш концептуально відмінним серед розглянутих інструментів. Заснований у 2016 році, він позиціонує себе не як

спеціалізований task manager, а як універсальний «all-in-one workspace» — поєднання бази знань, документів, таблиць та управління завданнями в єдиному гнучкому середовищі. Кожен елемент у Notion є блоком, що може бути довільно організований у сторінки, бази даних та вкладені структури.

Notion AI, запущений у 2023 році та включений у всі платні плани, є одним із найбільш зрілих AI-рішень у цій категорії [14]. Він підтримує генерацію та редагування тексту, автоматичне заповнення властивостей баз даних, створення підсумків документів, відповіді на запитання про вміст робочого простору та автоматизацію повторюваних операцій. Технічно Notion AI базується на моделях OpenAI та Anthropic, інтегрованих безпосередньо в редактор.

Головним недоліком Notion з точки зору управління завданнями є надмірна гнучкість, що перетворюється на складність: налаштування ефективної системи task management у Notion вимагає значного часу та досвіду. Крім того, продуктивність платформи при великих обсягах даних поступається спеціалізованим task manager-ам.

ClickUp

ClickUp, заснований у 2017 році, є найамбітнішим конкурентом у цьому сегменті з гаслом «замінити всі інші застосунки». Платформа підтримує понад 15 різних режимів відображення завдань, включаючи унікальні варіанти на кшталт Mind Map та Whiteboard, надзвичайно гнучку систему налаштування полів завдань та розвинену автоматизацію.

ClickUp Brain, представлений у 2023 році, є найбільш комплексним AI-рішенням серед розглянутих платформ [14]. Він об'єднує три компоненти: AI Knowledge Manager для відповідей на запитання про вміст робочого простору, AI Project Manager для автоматичного оновлення статусів та формування звітів, та AI Writer for Work для генерації та редагування текстового контенту. ClickUp Brain базується на власній інфраструктурі та моделях від OpenAI.

Основним недоліком ClickUp є висока складність платформи, що ускладнює початкове освоєння та може призводити до «паралічу від

надмірності можливостей» у нових користувачів. AI-функціональність доступна за окрему плату — 7 доларів на місяць за користувача на додаток до базового тарифу.

Linear

Linear є порівняно молодим гравцем ринку, заснованим у 2019 році, однак здобув значну популярність серед розробників програмного забезпечення завдяки поєднанню мінімалістичного дизайну, надзвичайної швидкості роботи та глибокої інтеграції з інструментами розробки. Linear орієнтований на технічні команди та реалізує Scrum та Kanban-методології з підтримкою циклів (спринтів), дорожніх карт та інтеграції з GitHub, GitLab та Figma.

Linear AI, запущений у 2024 році, фокусується на автоматизації специфічних для розробки завдань: автоматичне прив'язування завдань до pull request-ів, генерація описів завдань на основі коду та автоматичне визначення пріоритетів на основі активності в репозиторії.

Motion

Motion є найбільш радикальним прикладом AI-орієнтованого підходу до управління завданнями. На відміну від усіх попередніх інструментів, де AI виступає додатковою функцією поверх традиційного task manager-a, Motion будує весь продукт навколо AI-планувальника. Система автоматично складає розклад робочого дня користувача, розподіляючи завдання по вільних слотах календаря з урахуванням пріоритетів, дедлайнів та тривалості, що оцінюється AI. При виникненні несподіваних змін — нова термінова зустріч або затримка завдання — Motion автоматично перебудовує весь розклад.

Підхід Motion є інноваційним, однак має суттєве обмеження: повна автоматизація планування підходить далеко не всім користувачам і може сприйматись як надмірний контроль. Крім того, Motion є одним із найдорожчих рішень у категорії — від 19 доларів на місяць за особистий план.

Для структурованого порівняння розглянутих платформ та визначення орієнтирів для проєктування власного рішення проведемо функціональний

аналіз за ключовими параметрами. Порівняльний аналіз функціональності існуючих платформ подано у таблиці 1.2.

Таблиця 1.2

Порівняльний аналіз функціональності існуючих систем управління завданнями

Функціональність	**Trello**	**Asana**	**Notion**	**ClickUp**	**TaskAI**
Kanban-дошка	✓	✓	✓	✓	✓
Список завдань	Power-Up	✓	✓	✓	✓
Аналітика та звіти	обмежено	✓	✓	✓	✓
AI-пріоритизація	платно	платно	платно	платно	✓
AI-чат з даними	—	платно	платно	платно	✓
AI-оптимізація списку	—	—	частково	платно	✓
Безкоштовний AI	—	—	—	—	✓
Фільтрація за пріоритетом	✓	✓	✓	✓	✓
Відкритий код	—	—	—	—	✓
Швидкість роботи	висока	середня	середня	низька	висока

Проведений огляд дозволяє сформулювати кілька ключових висновків, що стануть орієнтиром при проєктуванні розроблюваного сервісу.

По-перше, усі розглянуті платформи монетизують AI-функціональність як преміальну опцію, роблячи її недоступною для безкоштовних користувачів. Це формує очевидну нішу для рішення, що пропонує реальний AI-асистент у повністю безкоштовному форматі. Розроблюваний сервіс TaskAI закриває цю нішу завдяки використанню безкоштовного Groq API.

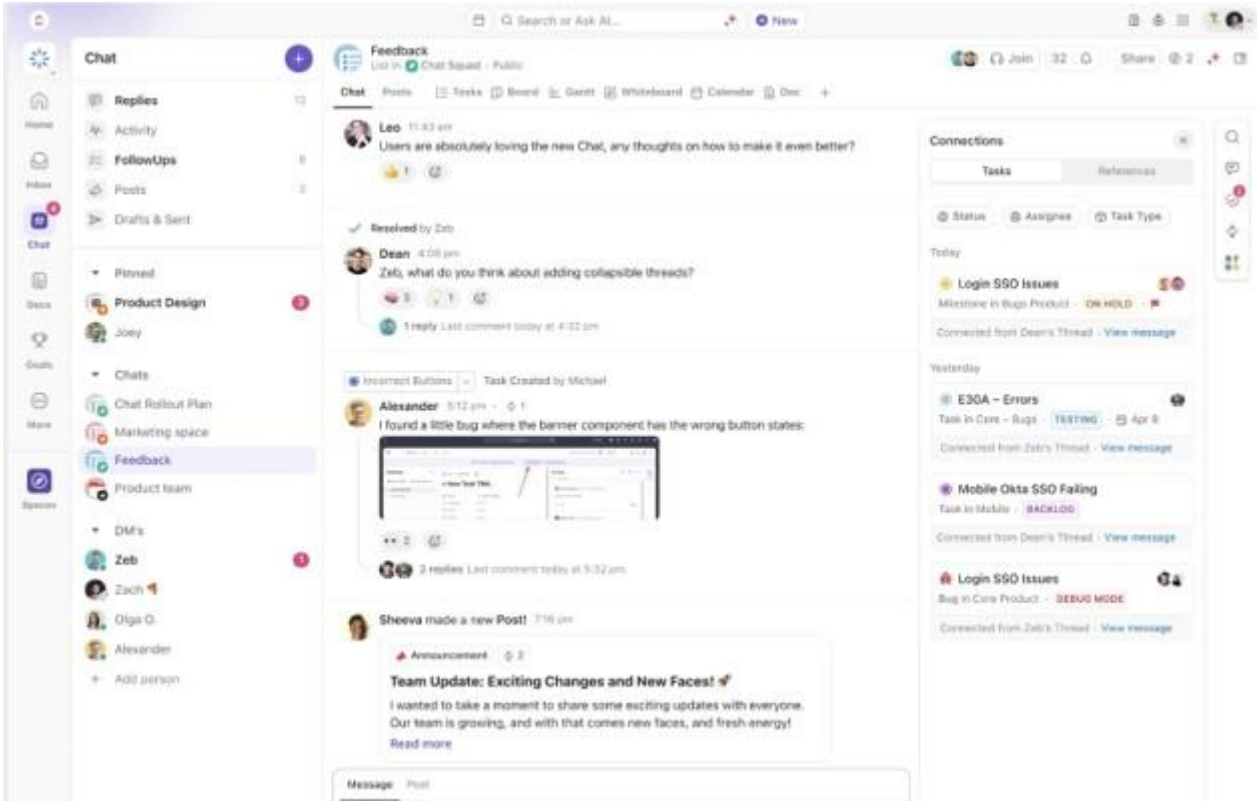
По-друге, більшість платформ страждають від надмірної складності та когнітивного перевантаження інтерфейсу. Trello є надто простим і не масштабується для складних проєктів, тоді як ClickUp та Notion є надто складними для швидкого старту. TaskAI прагне зайняти оптимальну середину: достатньо функціональний для реального використання, але не перевантажений зайвими налаштуваннями.

По-третє, жодна з розглянутих безкоштовних платформ не надає можливості прямого діалогу з AI про конкретні завдання користувача в реальному часі. Більшість AI-функцій є автоматизованими та не

передбачають інтерактивної взаємодії. Вбудований чат-асистент TaskAI, що має повний доступ до актуального списку завдань та відповідає на довільні запити українською мовою, є суттєвою диференціюючою характеристикою розроблюваного рішення.

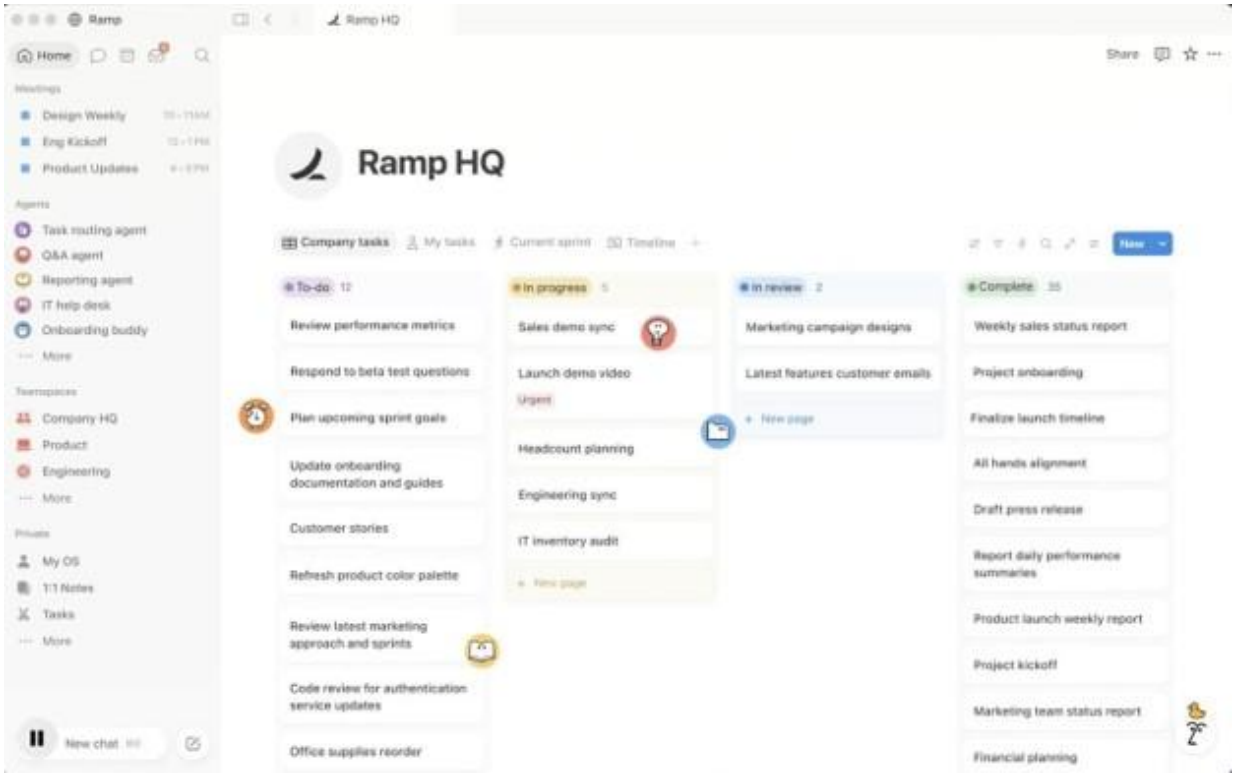
По-четверте, швидкість роботи інтерфейсу є критично важливою характеристикою для інструментів продуктивності. Повільний інтерфейс підриває саму ідею підвищення ефективності. Використання Vite як інструменту збірки та React з оптимізованим управлінням станом через Zustand забезпечує TaskAI суб'єктивно миттєву реакцію на дії користувача.

Зазначені висновки є теоретичним підґрунтям для прийняття проєктних рішень у другому розділі кваліфікаційної роботи, де буде обґрунтовано вибір технологічного стеку та визначено архітектуру розроблюваного веб-сервісу. Інтерфейс платформи Trello зображено на малюнку 1.1.



Малюнок 1.1 - скріншот інтерфейсу ClickUp

Інтерфейс платформи ClickUp зображено на малюнку 1.2.



Малюнок 1.2 - скріншот інтерфейсу Notion

РОЗДІЛ 2: ВИБІР ТЕХНОЛОГІЙ ТА ПРОЄКТУВАННЯ ВЕБ-СЕРВІСУ

2.1. Обґрунтування вибору технологічного стеку

Вибір технологічного стеку є одним із найважливіших архітектурних рішень при розробці будь-якого веб-застосунку. Від правильності цього вибору залежать продуктивність системи, зручність розробки та підтримки, масштабованість, швидкість завантаження інтерфейсу, а також доступність кваліфікованих фахівців для подальшої роботи над проєктом. Хибний вибір на цьому етапі може призвести до суттєвих витрат часу та ресурсів на пізніших стадіях розробки, коли зміна базових технологій стає надзвичайно дорогою операцією. У даному підрозділі розглянемо альтернативні варіанти для кожного ключового технологічного рівня та обґрунтуємо прийняті рішення.

2.1.1. Вибір мови програмування

Сучасна фронтенд-розробка базується на JavaScript — єдиній мові програмування, що нативно підтримується всіма веб-браузерами без додаткової компіляції чи встановлення плагінів. JavaScript пройшов довгий шлях від простої мови для додавання інтерактивності на веб-сторінках до повноцінної універсальної мови програмування, що використовується як на клієнтській стороні у браузері, так і на серверній через середовище виконання Node.js.

Поряд із класичним JavaScript широкого поширення набув TypeScript — мова програмування, розроблена компанією Microsoft, що є надбудовою над JavaScript із додаванням статичної типізації. TypeScript компілюється у звичайний JavaScript і дозволяє виявляти цілий клас помилок ще на етапі написання коду, до запуску програми. Для великих проєктів із командами розробників TypeScript суттєво підвищує надійність кодової бази та спрощує рефакторинг.

Для даного проєкту обрано JavaScript як основну мову розробки. Це рішення обумовлене кількома факторами. По-перше, розроблюваний застосунок є проєктом середньої складності, де переваги TypeScript щодо

великих кодових баз є менш критичними. По-друге, JavaScript забезпечує більш швидкий старт та менший обсяг шаблонного коду, що є важливим в умовах обмеженого часу на виконання кваліфікаційної роботи. По-третє, всі використовувані бібліотеки мають відмінну підтримку JavaScript і не вимагають TypeScript для повноцінного використання.

2.1.2. Порівняльний аналіз інструментів збірки

Сучасна фронтенд-розробка неможлива без інструментів збірки — програмних засобів, що перетворюють вихідний код розробника на оптимізовані файли, готові до розгортання в браузері. Вони виконують транспіляцію JSX у звичайний JavaScript, об'єднання модулів у бандли, мінімізацію коду, оптимізацію зображень та багато інших операцій.

Webpack є найстарішим і найпоширенішим інструментом збірки для JavaScript-застосунків, що з'явився у 2012 році. Webpack відрізняється надзвичайною гнучкістю та потужною системою плагінів, однак має суттєвий недолік — повільний холодний старт та час перезбірки при внесенні змін у великих проєктах. Саме з цим недоліком пов'язана поява нового покоління інструментів.

Vite, розроблений Еваном Ю — автором Vue.js — у 2020 році, використовує принципово інший підхід: замість попередньої збірки всього застосунку Vite у режимі розробки роздає модулі безпосередньо браузеру через нативний ESM (ECMAScript Modules), трансформуючи лише ті файли, що реально запитуються в даний момент [16]. Це забезпечує практично миттєвий холодний старт незалежно від розміру проєкту. Для виробничої збірки Vite використовує Rollup — зрілий і добре оптимізований бандлер. За даними опитування State of JS 2024, Vite є найбільш задоволеним серед розробників інструментом збірки вже третій рік поспіль [23].

Create React App (CRA) — офіційний інструмент ініціалізації React-проєктів від Meta, що використовує Webpack під капотом. Попри зручність початкового налаштування, CRA суттєво поступається Vite за швидкістю у

режимі розробки та наразі офіційно не рекомендується командою React для нових проєктів.

Для даного проєкту обрано Vite як інструмент збірки. Миттєвий старт сервера розробки, швидке оновлення модулів (HMR) та сучасна конфігурація за замовчуванням роблять його оптимальним вибором для React-застосунку.

2.1.3. Порівняльний аналіз JavaScript-фреймворків

Ринок фронтенд-розробки пропонує три основні рішення для побудови складних інтерактивних інтерфейсів: React, Vue.js та Angular. Кожен із них має свою філософію, переваги та типові сфери застосування.

React — бібліотека для побудови інтерфейсів користувача, розроблена компанією Meta та вперше представлена у 2013 році. Технічно React зосереджений виключно на шарі представлення та використовує концепцію компонентів — незалежних, перевикористовуваних блоків інтерфейсу, кожен з яких управляє власним станом та відображенням. React використовує JSX — розширення синтаксису JavaScript, що дозволяє описувати розмітку інтерфейсу безпосередньо у коді [8], [14]. Ключовою технічною інновацією React є віртуальний DOM — легка копія реального DOM у пам'яті JavaScript, що дозволяє мінімізувати кількість дорогих операцій із реальним DOM через порівняння попереднього та нового стану. За даними опитування Stack Overflow Developer Survey 2024, React залишається найпопулярнішою фронтенд-бібліотекою у світі, яку використовують понад 40% веб-розробників [22].

Vue.js — прогресивний JavaScript-фреймворк, створений Еваном Ю у 2014 році. Vue відрізняється м'якшою кривою навчання порівняно з React, чіткою структурою однофайлових компонентів та вбудованою реактивною системою. Vue особливо популярний серед розробників, що переходять з jQuery або серверних шаблонізаторів, а також у азіатських ринках розробки.

Angular — повноцінний фреймворк, розроблений компанією Google у 2016 році. Angular є найбільш «важким» з трьох рішень: він включає власну систему впровадження залежностей, модульну систему, маршрутизатор та

HTTP-клієнт. Angular орієнтований на TypeScript як основну мову та найкраще підходить для великих корпоративних застосунків із великими командами розробників. Порівняльний аналіз React, Vue.js та Angular наведено у таблиці 2.1.

Таблиця 2.1

Порівняльний аналіз React, Vue.js та Angular

Критерій	**React**	**Vue.js**	**Angular**
Розробник	Meta	Evan You	Google
Рік виходу	2013	2014	2016
Тип	Бібліотека	Фреймворк	Повний фреймворк
Основна мова	JS / TS	JS / TS	TypeScript
Крива навчання	Середня	Низька	Висока
Розмір бандлу	Малий	Малий	Великий
Популярність	★★★★★	★★★★	★★★
Екосистема	Дуже велика	Велика	Велика
Підходить для	SPA, великих проєктів	SPA, середніх проєктів	Корпоративних застосунків

На основі проведеного аналізу для реалізації даного проєкту обрано React версії 18. Ключовими аргументами є: домінуюча популярність бібліотеки та найбільша екосистема допоміжних пакетів; компонентна модель, що ідеально відповідає задачі побудови Kanban-інтерфейсу, де картки завдань є природними перевикористовуваними компонентами; підтримка Concurrent Features у React 18, що забезпечує плавнішу роботу інтерфейсу при асинхронних операціях, зокрема при запитах до Groq API.

2.1.4. Вибір системи управління станом

Управління станом є однією з ключових архітектурних задач у фронтенд-розробці. Стан застосунку включає всі дані, що можуть

змінюватись під час роботи: список завдань, активний фільтр пріоритету, поточне представлення (дошка, список, аналітика), повідомлення AI-чату та стан завантаження. При зростанні складності застосунку виникає потреба у централізованому сховищі, доступному з будь-якої частини застосунку без явної передачі через props на кожному рівні дерева компонентів.

Context API — вбудований механізм React — добре підходить для даних, що рідко змінюються: теми оформлення, мови інтерфейсу. Однак при частих оновленнях Context API викликає перерендеринг усіх компонентів-споживачів, що може негативно впливати на продуктивність.

Redux Toolkit є офіційно рекомендованим способом використання Redux і суттєво спрощує роботу з ним завдяки функції createSlice. Redux Toolkit забезпечує чітку структуру та відмінні інструменти налагодження через Redux DevTools, однак вимагає більшого обсягу шаблонного коду навіть після спрощень, які він вносить.

Zustand — легковагова бібліотека управління станом, розроблена командою Pmndrs у 2019 році. Zustand реалізує мінімалістичний підхід: весь стан описується у єдиному хуку з простим API для читання та оновлення. Бібліотека не вимагає провайдерів або обгортання кореневого компонента, має мінімальний розмір (близько 1 КБ після мінімізації) та повністю сумісна з React DevTools [15]. При цьому Zustand підтримує middleware для персистентності стану через localStorage та DevTools-інтеграцію.

Jotai та Recoil є бібліотеками атомарного управління станом, що базуються на концепції незалежних атомів стану. Вони добре підходять для гранульованого управління станом, однак мають меншу спільноту та екосистему порівняно з Zustand та Redux.

Для даного проєкту обрано Zustand як систему управління станом. Це рішення обґрунтоване тим, що архітектура застосунку передбачає відносно невелику кількість доменів стану — завдання, UI-стан та повідомлення AI-чату — для яких лаконічний API Zustand є оптимальним. Відсутність

необхідності у провайдерах та мінімальний розмір бібліотеки відповідають загальній філософії проєкту щодо уникнення надлишкової складності.

2.1.5. Вибір бібліотеки іконок та допоміжних інструментів

Для відображення іконок у інтерфейсі обрано Tabler Icons — відкриту бібліотеку із понад 5000 іконок у консистентному стилі, доступну у форматі веб-шрифту. Перевагами цього рішення є відсутність необхідності імпортувати кожну іконку окремо, простота використання через CSS-класи та відмінна якість відображення на екранах із високою щільністю пікселів.

Для маршрутизації між сторінками застосунку обрано React Router DOM версії 6 — стандартну бібліотеку клієнтської маршрутизації для React [17]. React Router забезпечує декларативне визначення маршрутів та плавну навігацію без перезавантаження сторінки, що є ключовою характеристикою Single Page Application. Стек проєкту TaskAI подано у таблиці 2.2.

Таблиця 2.2

Підсумковий технологічний стек проєкту TaskAI

Рівень	**Технологія**	**Версія**	**Призначення**
Мова	JavaScript	ES2022+	Основна мова розробки
Збірка	Vite	6.x	Інструмент збірки та сервер розробки
UI-бібліотека	React	18.x	Побудова компонентного інтерфейсу
Управління станом	Zustand	5.x	Централізоване сховище даних
Маршрутизація	React Router DOM	6.x	Клієнтська навігація між сторінками
Стилізація	CSS Modules	—	Ізольовані стилі компонентів
Іконки	Tabler Icons	latest	Бібліотека векторних іконок
AI-інтеграція	Groq API	—	Доступ до моделі Llama 3.3 70B
Персистентність	localStorage	—	Збереження завдань між сесіями

Обраний технологічний стек є сучасним, добре підтримуваним та широко застосовуваним у промисловій розробці. Поєднання Vite, React та Zustand забезпечує оптимальний баланс між продуктивністю розробки, швидкістю роботи застосунку та простотою архітектури, що є важливим як для навчального проєкту, так і для подальшого розширення функціональності.

2.2. Архітектура застосунку та моделі даних

Проєктування архітектури є фундаментальним етапом розробки програмного забезпечення, що передує написанню коду. Грамотно спроєктована архітектура забезпечує зрозумілість структури проєкту, зручність підтримки та розширення функціональності, мінімізує кількість помилок при реалізації та спрощує онбординг нових розробників у кодову базу. У даному підрозділі розглянемо архітектурні рішення, прийняті для розроблюваного веб-сервісу TaskAI, обґрунтуємо вибір підходів та детально опишемо модель даних.

2.2.1. Загальна архітектура застосунку

Розроблюваний застосунок реалізований за архітектурним патерном SPA (Single Page Application) із клієнтською маршрутизацією [11]. Це означає, що при першому завантаженні браузер отримує єдиний HTML-файл разом із JavaScript-бандлом, після чого вся навігація між розділами сервісу відбувається на стороні клієнта без звернень до сервера за новими сторінками. Такий підхід забезпечує швидку та плавну навігацію, характерну для нативних застосунків, та є стандартом для сучасних продуктивних веб-інструментів.

Вибір SPA-архітектури для сервісу управління завданнями є природним та обґрунтованим. Користувач, що працює з task manager-ом, виконує численні швидкі дії — додавання завдань, зміна фільтрів, перегляд аналітики, відправка запитів до AI-асистента — і будь-яка затримка,

пов'язана з перезавантаженням сторінки, руйнувала б потік роботи та суперечила б самій ідеї інструменту продуктивності.

З точки зору поділу відповідальності застосунок організовано за принципом розшарованої архітектури, де кожен шар має чітко визначену роль та мінімальну залежність від деталей реалізації інших шарів.

Шар представлення (Presentation Layer) реалізований засобами React-компонентів та CSS Modules. Цей шар відповідає виключно за відображення даних та обробку взаємодій користувача — кліків, введення тексту, вибору фільтрів. Компоненти цього шару отримують дані зі стору Zustand через хуки та викликають відповідні дії для оновлення стану, але не містять бізнес-логіки.

Шар управління станом (State Management Layer) реалізований засобами Zustand. Цей шар є централізованим сховищем усіх даних застосунку, що можуть змінюватись під час роботи. Він містить три незалежних домени стану: tasks для управління завданнями, ui для управління поточним виглядом та фільтрами, та ai для зберігання повідомлень чату та стану завантаження. Завдяки middleware persist стан завдань автоматично зберігається у localStorage та відновлюється при наступному відкритті застосунку [15].

Шар бізнес-логіки (Business Logic Layer) у даному застосунку реалізований частково у сторі Zustand через функції-екшени, частково у спеціалізованих хуках. Зокрема, хук useAI інкапсулює всю логіку взаємодії з Groq API: формування системного промпту з поточними завданнями, відправку запиту, обробку відповіді та управління станом завантаження.

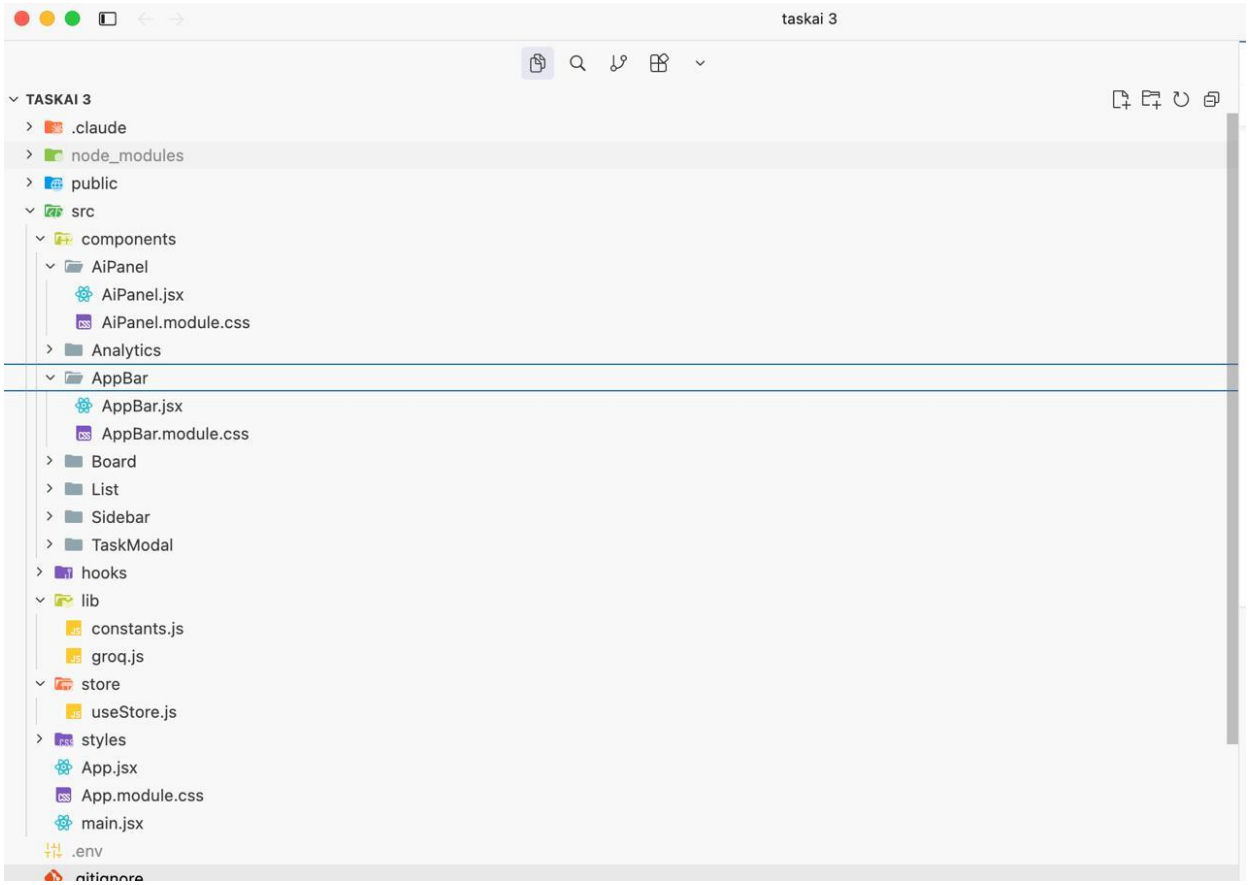
Шар даних (Data Layer) включає модуль взаємодії з Groq API та механізм персистентності через localStorage. У реальному комерційному проєкті цей шар містив би повноцінний API-клієнт для взаємодії з серверним бекендом та базою даних, однак у межах даної кваліфікаційної роботи зосередимось на клієнтській архітектурі.

2.2.2. Структура файлів та директорій

Організація файлової структури проєкту безпосередньо відображає його архітектуру та суттєво впливає на зручність навігації по кодовій базі в процесі розробки та підтримки. Для даного проєкту прийнята структура, що базується на функціональному поділі за типом відповідальності кожного модуля.

Коренева директорія проєкту містить конфігураційні файли: package.json із переліком залежностей та скриптами для запуску, vite.config.js із налаштуваннями інструменту збірки та index.html як єдину HTML-сторінку застосунку.

Директорія src організована таким чином: файл main.jsx є точкою входу і підключає кореневий компонент до DOM; файл App.jsx є кореневим компонентом застосунку і визначає маршрути та загальний макет; директорія components містить перевикористовувані компоненти, кожен у власній піддиректорії з файлом компонента та відповідним CSS Module; директорія store містить Zustand-стор; директорія hooks містить кастомні React-хуки; директорія lib містить допоміжні утиліти, константи та модуль для роботи з Groq API; директорія styles містить глобальні CSS-стилі та змінні. Структуру файлів проєкту у редакторі VS Code зображено на малюнку 2.1.



Малюнок 2.1 - Структура файлів проєкту у редакторі VS Code.

2.2.3. Проєктування моделі даних

Модель даних визначає структуру об'єктів, з якими працює застосунок. Для сервісу управління завданнями ключовою сутністю є завдання (task). Визначимо його структуру та обґрунтуємо вибір кожного поля. Модель даних об'єкта «Завдання» наведено у таблиці 2.3.

Таблиця 2.3

Модель даних об'єкта «Завдання» (Task)

Поле	Тип	Обов'язкове	Опис
id	Number	Так	Унікальний числовий ідентифікатор завдання
title	String	Так	Назва завдання
desc	String	Ні	Детальний опис завдання
priority	String	Так	Пріоритет: «high», «med» або «low»
cat	String	Так	Категорія: «Backend», «Frontend», «DevOps» тощо
due	String	Ні	Дедлайн у форматі ISO 8601 (YYYY-MM-

Поле	Тип	Обов'язкове	Опис
			DD)
status	String	Так	Статус: «todo», «inprogress» або «done»
score	Number	Так	AI-оцінка пріоритетності від 1 до 10

Поле score заслуговує окремого пояснення. Це числове значення від 1 до 10, що генерується автоматично при створенні завдання та відображає початкову оцінку його пріоритетності на основі комбінації заявленого пріоритету та випадкової варіації, що імітує реальну неоднорідність у межах одного рівня пріоритету. Це поле використовується для сортування завдань у режимі списку та відображається на картках як «AI-score», що додає ігровий елемент та стимулює користувача звертати увагу на пріоритизацію.

Модель стану Zustand-стору є центральним елементом архітектури. Загальний стан застосунку складається з трьох незалежних доменів, кожен з яких управляється власним набором екшенів. Структуру глобального стану Zustand-стору подано у таблиці 2.4.

Таблиця 2.4

Структура глобального стану Zustand-стору

Домен	Поле	Тип	Початкове значення	Призначення
tasks	tasks	Array	DEFAULT_TASKS	Масив усіх завдань
tasks	nextId	Number	6	Лічильник для генерації id
ui	view	String	«board»	Поточне відображення
ui	filterPriority	String	«all»	Активний фільтр пріоритету
ui	modalOpen	Boolean	false	Стан модального вікна
ai	messages	Array	[...]	Повідомлення чату з AI
ai	aiLoading	Boolean	false	Стан очікування відповіді AI

2.2.4. Проєктування маршрутизації

Попри те що розроблюваний сервіс є переважно односторінковим застосунком із перемиканням між режимами відображення без зміни URL, базова структура маршрутизації через React Router DOM збережена для дотримання архітектурних стандартів та спрощення потенційного розширення. Поточна версія застосунку реалізує єдиний кореневий маршрут, що відображає головний компонент App із усіма підкомпонентами. Перемикання між дошкою, списком та аналітикою відбувається через зміну стану view у Zustand-сторі, а не через зміну URL, що є свідомим рішенням на користь спрощення для поточного функціонального обсягу.

2.2.5. Проєктування потоків даних

Потік даних у Zustand-застосунку є більш гнучким порівняно з Redux, однак зберігає принцип уникнення прямих мутацій стану компонентами. Розглянемо типові потоки даних для основних сценаріїв використання сервісу.

Сценарій «Додавання нового завдання» реалізується таким чином: користувач натискає кнопку «Завдання» у верхній панелі; компонент AppBar викликає екшен openModal зі стору; компонент TaskModal отримує modalOpen зі стору та відображається; користувач заповнює форму та підтверджує; компонент TaskModal викликає екшен addTask із об'єктом нового завдання; Zustand оновлює масив tasks та nextId; завдяки middleware persist оновлений стан автоматично зберігається у localStorage; всі компоненти, що підписані на tasks через useStore, автоматично перерендеряться з новими даними.

Сценарій «Фільтрація за пріоритетом» реалізується так: користувач натискає на пункт «Критичні» у лівій панелі навігації; компонент Sidebar викликає екшен setFilterPriority зі значенням «high»; Zustand оновлює filterPriority у домені ui; компоненти Board, List та Analytics отримують filteredTasks — обчислюване значення, що автоматично перераховується на

основі поточного filterPriority; відображення оновлюється без жодних додаткових дій.

Сценарій «Запит до AI-асистента» є найбільш складним з точки зору потоку даних: користувач вводить текст у поле чату та надсилає запит; компонент AiPanel викликає функцію ask із хука useAI; хук useAI встановлює aiLoading у true через екшен setAILoading та додає повідомлення користувача через addMessage; формується системний промпт із поточним списком завдань та відправляється HTTP-запит до Groq API; при отриманні відповіді aiLoading скидається у false та відповідь AI додається до масиву messages; компонент AiPanel автоматично оновлює відображення чату.

Таким чином, архітектура розроблюваного сервісу базується на чіткому розподілі відповідальності між шарами, однонаправленому потоці даних та централізованому управлінні станом через Zustand. Прийняті архітектурні рішення забезпечують зрозумілість кодової бази, передбачуваність поведінки системи та зручність подальшого розширення функціональності.

2.3. Проєктування інтерфейсу користувача та UX/UI-рішень

Якість інтерфейсу користувача є одним із визначальних факторів успіху будь-якого веб-сервісу, і особливо це стосується інструментів продуктивності. Парадоксально, але саме застосунки, покликані підвищувати ефективність роботи, найбільш чутливі до недоліків у дизайні: кожна секунда, витрачена на пошук потрібної кнопки, кожен зайвий клік або неочевидна навігаційна структура безпосередньо знецінює головну цінність продукту. Дослідження Nielsen Norman Group свідчать, що користувачі формують перше враження про веб-застосунок протягом 50 мілісекунд, а 88% користувачів не повертаються до сервісу після негативного досвіду взаємодії з інтерфейсом [24], [25]. Тому проєктування UX (User Experience) та UI (User Interface) має передувати написанню коду і базуватись на розумінні потреб цільової аудиторії та перевірених принципах дизайну.

2.3.1. Визначення цільової аудиторії та користувацьких сценаріїв

Цільова аудиторія розроблюваного сервісу TaskAI охоплює кілька ключових груп користувачів. Основну групу складають фахівці у сфері інформаційних технологій — розробники програмного забезпечення, дизайнери, QA-інженери та DevOps-спеціалісти, що ведуть активну проєктну діяльність та потребують зручного інструменту для відстеження власних завдань. Другу групу складають студенти технічних спеціальностей, що виконують курсові та дипломні проєкти і потребують інструменту для планування етапів роботи. Третю групу утворюють менеджери невеликих команд, яким потрібен легкий та швидкий у впровадженні інструмент без тривалого налаштування, характерного для корпоративних систем.

Спільними характеристиками цих груп є технічна грамотність та звичка до сучасних веб-застосунків, орієнтація на ефективність та небажання витрачати час на освоєння складних інструментів, а також вимогливість до швидкості роботи інтерфейсу. Більшість представників цільової аудиторії використовує застосунок з десктопного браузера в процесі роботи, хоча мобільна адаптивність залишається важливою вимогою.

На основі аналізу цільової аудиторії визначено чотири основні користувацькі сценарії, що мають бути реалізовані в застосунку.

Сценарій 1 — «Швидке додавання завдання»: користувач згадує нове завдання у процесі роботи і хоче зафіксувати його якнайшвидше, не відволікаючись від поточної діяльності. Очікуваний шлях — одне натискання кнопки «Завдання», введення назви, підтвердження Enter. Ціль — не більше 10 секунд від думки до збереженого завдання.

Сценарій 2 — «Планування робочого дня»: на початку дня користувач переглядає всі завдання, оцінює дедлайни та звертається до AI-асистента за рекомендацією щодо порядку виконання. Очікуваний шлях — відкриття застосунку, перегляд дошки або списку, натискання кнопки «AI-оптимізація» та отримання конкретних рекомендацій.

Сценарій 3 — «Фільтрація за пріоритетом»: користувач хоче зосередитись лише на критичних завданнях, ігноруючи решту. Очікуваний шлях — одне натискання пункту «Критичні» у лівій панелі, миттєве оновлення відображення.

Сценарій 4 — «Аналіз прогресу»: наприкінці тижня користувач хоче оцінити, скільки завдань виконано, яке розподілення за категоріями та чи є критичні завдання, що потребують термінової уваги. Очікуваний шлях — перехід до розділу «Аналітика», швидкий перегляд зведених показників.

2.3.2. Принципи UX-проєктування

При проєктуванні інтерфейсу розроблюваного сервісу застосовано низку ключових UX-принципів, що формують основу сучасного підходу до дизайну продуктивних інструментів.

Принцип мінімального зусилля передбачає, що найчастіші дії користувача мають виконуватись якнайшвидше і з мінімальною кількістю кроків. У TaskAI цей принцип реалізовано на кількох рівнях: кнопка додавання завдання завжди доступна у верхній панелі незалежно від поточного розділу; швидке додавання безпосередньо у колонку Kanban-дошки через кнопку «Додати» не потребує відкриття модального вікна; фільтрація пріоритетів виконується одним кліком у бічній панелі.

Принцип зворотного зв'язку означає, що інтерфейс має негайно реагувати на дії користувача та повідомляти про результат. При додаванні завдання через модальне вікно AI-асистент автоматично підтверджує додавання повідомленням у чаті. При відправці запиту до Groq API відображається анімований індикатор введення тексту (три пульсуючі крапки), що сигналізує про активну обробку запиту та запобігає подвійним натисканням.

Принцип консистентності передбачає однаковий вигляд та поведінку однотипних елементів по всьому застосунку. Всі картки завдань мають однакову структуру незалежно від колонки. Кольорове кодування

пріоритетів — червоний для критичного, помаранчевий для середнього, зелений для низького — витримується в усіх режимах відображення: на картках дошки, у рядках списку, у фільтрах бічної панелі та в аналітиці. Завдяки CSS-змінним ця консистентність підтримується автоматично.

Принцип ієрархії інформації визначає, що найважливіша інформація має бути найбільш виразною. На картці завдання ієрархія побудована так: назва завдання є найбільшим та найжирнішим текстом; теги пріоритету та категорії розміщені під назвою у вигляді кольорових мітках; AI-score та дедлайн розташовані внизу картки меншим шрифтом; прогрес-бар є найменш виразним елементом, що доповнює загальну картину, але не домінує.

Принцип попередження помилок передбачає, що інтерфейс має унеможливлювати або мінімізувати ймовірність помилкових дій. Кнопка підтвердження у формі додавання завдання залишається неактивною до введення назви. При натисканні поза межами модального вікна воно закривається, що є стандартною та очікуваною поведінкою для більшості користувачів. Введення дедлайну реалізоване через нативний date picker браузера, що унеможливлює введення некоректного формату дати.

2.3.3. Система дизайну та візуальна ідентичність

Візуальна ідентичність TaskAI базується на принципі «функціональної нейтральності» — інтерфейс має бути достатньо виразним, щоб бути приємним у використанні, але не настільки яскравим, щоб відволікати від змісту завдань. Цей принцип відрізняє інструменти продуктивності від розважальних застосунків і пояснює, чому більшість успішних task manager-ів використовують стримані, переважно монохромні палітри.

Кольорова схема. Основний фон застосунку реалізований через CSS-змінну --bg-tertiary з відтінком #EDEDEB — теплий світло-сірий, що на відміну від чистого білого не стомлює очі при тривалій роботі. Панелі та картки використовують --bg-secondary (#F5F5F3) та --bg-primary (#FFFFFF) відповідно, створюючи природну глибину без використання тіней.

Акцентний колір --c-accent (#7F77DD) — приглушений фіолетовий — використовується для логотипу та активних елементів навігації, надаючи інтерфейсу характер без агресивної яскравості. Семантичні кольори для пріоритетів засновані на загальноприйнятій схемі: червоний (#FCEBEB / #A32D2D) для критичних, помаранчевий (#FAEEDA / #854F0B) для середніх, зелений (#EAF3DE / #3B6D11) для низьких. Кожен семантичний колір реалізований парою фон/текст для забезпечення достатнього контрасту відповідно до стандарту WCAG 2.1 [29]. Кольорову палітру інтерфейсу TaskAI наведено у таблиці 2.5

Таблиця 2.5

Кольорова палітра інтерфейсу TaskAI

Змінна	Світла тема	Призначення
--bg-primary	#FFFFFF	Фон карток та модальних вікон
--bg-secondary	#F5F5F3	Фон бічної панелі та колонок
--bg-tertiary	#EDEDEB	Загальний фон застосунку
--text-primary	#1A1A18	Основний текст
--text-secondary	#6B6B67	Другорядний текст
--text-tertiary	#9B9B97	Підказки та мітки часу
--c-accent	#7F77DD	Акцент, логотип, активні елементи
--c-err-bg / txt	#FCEBEB / #A32D2D	Критичний пріоритет
--c-warn-bg / txt	#FAEEDA / #854F0B	Середній пріоритет
--c-ok-bg / txt	#EAF3DE / #3B6D11	Низький пріоритет
--c-info-bg / txt	#E6F1FB / #185FA5	AI-score, інформаційні елементи

Важливою архітектурною рисою кольорової системи є повна підтримка темної теми через медіа-запит prefers-color-scheme: dark. Всі кольори визначені через CSS-змінні та мають окремий набір значень для темної теми, що перевизначається автоматично без жодного JavaScript. Це дозволяє застосунку коректно відображатись відповідно до системних налаштувань користувача без додаткових налаштувань.

Типографіка. Для всього інтерфейсу використовується системний шрифтовий стек: -apple-system, BlinkMacSystemFont, 'Segoe UI', sans-serif.

Такий підхід забезпечує відображення нативного системного шрифту на кожній платформі — San Francisco на macOS та iOS, Segoe UI на Windows, Roboto на Android — що гарантує максимальну читабельність та природну інтеграцію з операційним середовищем без завантаження зовнішніх шрифтів [27].

Форми та заокруглення. Інтерактивні елементи мають помірне заокруглення кутів: картки завдань — 8px (--r), модальні вікна — 12px (--rl), теги пріоритетів — 8px. Такий підхід надає інтерфейсу сучасного та дружнього характеру, уникаючи при цьому надмірної «пухнастості», що може виглядати непрофесійно в контексті ділового інструменту.

2.3.4. Проєктування ключових екранів

Розглянемо детальніше структуру та логіку побудови ключових екранів застосунку.

Загальний макет (App Shell). Застосунок використовує фіксований макет із трьох вертикальних зон: верхня панель AppBar висотою 50px, що містить логотип та кнопки дій; робоча область, що займає всю висоту, що залишилась, та поділена на три горизонтальні зони — бічна панель навігації (215px), основна контентна область (flex: 1) та AI-панель (285px). Фіксована структура макету забезпечує передбачуваність інтерфейсу: користувач завжди знає, де знаходяться елементи навігації та AI-асистент, незалежно від поточного розділу.

Kanban-дошка є центральним функціональним екраном застосунку та реалізована у вигляді трьох рівноширих колонок: «До виконання», «В процесі» та «Готово». Кожна колонка має заголовок із лічильником завдань та кнопку швидкого додавання. Картки завдань містять назву, опис, теги пріоритету та категорії, AI-score, дедлайн та прогрес-бар. При відсутності завдань у колонці відображається порожній стан з іконкою та текстом — це важливий елемент дизайну, що запобігає відчуттю «зламаного» інтерфейсу при першому запуску або після агресивної фільтрації.

Список завдань відображає ті самі завдання у вигляді впорядкованого списку, відсортованого за AI-score у спадному порядку. Такий режим зручний для користувачів, що надають перевагу лінійному перегляду та хочуть одразу бачити рейтинг пріоритетності. Кожен рядок містить порядковий номер, назву, теги та score.

Аналітика відображає чотири зведені показники у вигляді карток — загальна кількість завдань, кількість виконаних, середній AI-score та кількість критичних — та два розділи з горизонтальними прогрес-барами: розподілення за статусами та розподілення за категоріями. Цей екран реагує на активний фільтр пріоритету, показуючи аналітику лише для відфільтрованих завдань.

AI-панель розташована у правій частині екрану та є постійно видимою незалежно від активного розділу. Вона містить заголовок з індикатором підключення, область повідомлень чату з автопрокруткою, рядок швидких запитів у вигляді чіпів та поле введення з кнопкою відправки. Повідомлення відображаються у вигляді бульбашок: повідомлення користувача вирівняні праворуч із темним фоном, відповіді AI — ліворуч із фоном вторинного кольору.

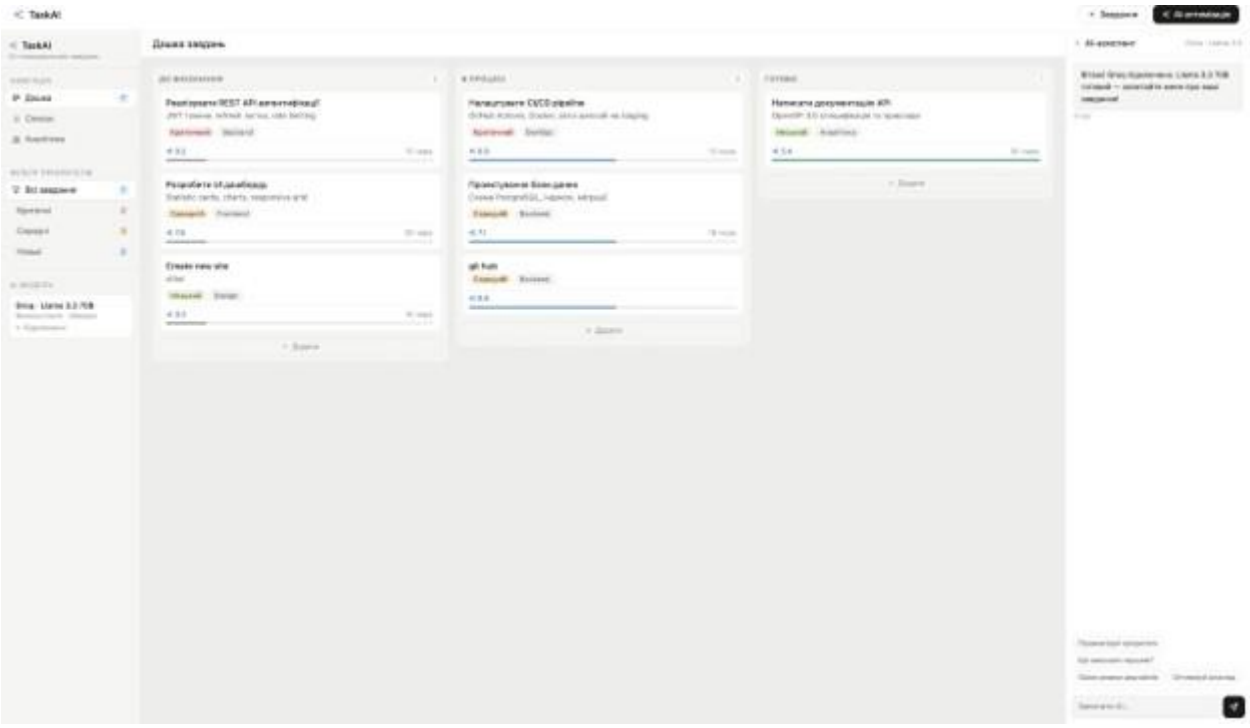
Модальне вікно додавання завдання відкривається поверх основного контенту з напівпрозорим оверлеєм та анімацією появи. Форма містить поля назви, опису, вибору пріоритету та категорії через dropdown, а також вибору дедлайну через date picker. За замовчуванням дедлайн встановлюється на 7 днів від поточної дати, що відповідає типовому горизонту планування.

2.3.5. Адаптивний дизайн

Попри те що основною платформою для TaskAI є десктопний браузер, адаптивність інтерфейсу є обов'язковою вимогою сучасного веб-застосунку. Адаптація реалізована через CSS-медіа-запити без використання зовнішніх CSS-фреймворків.

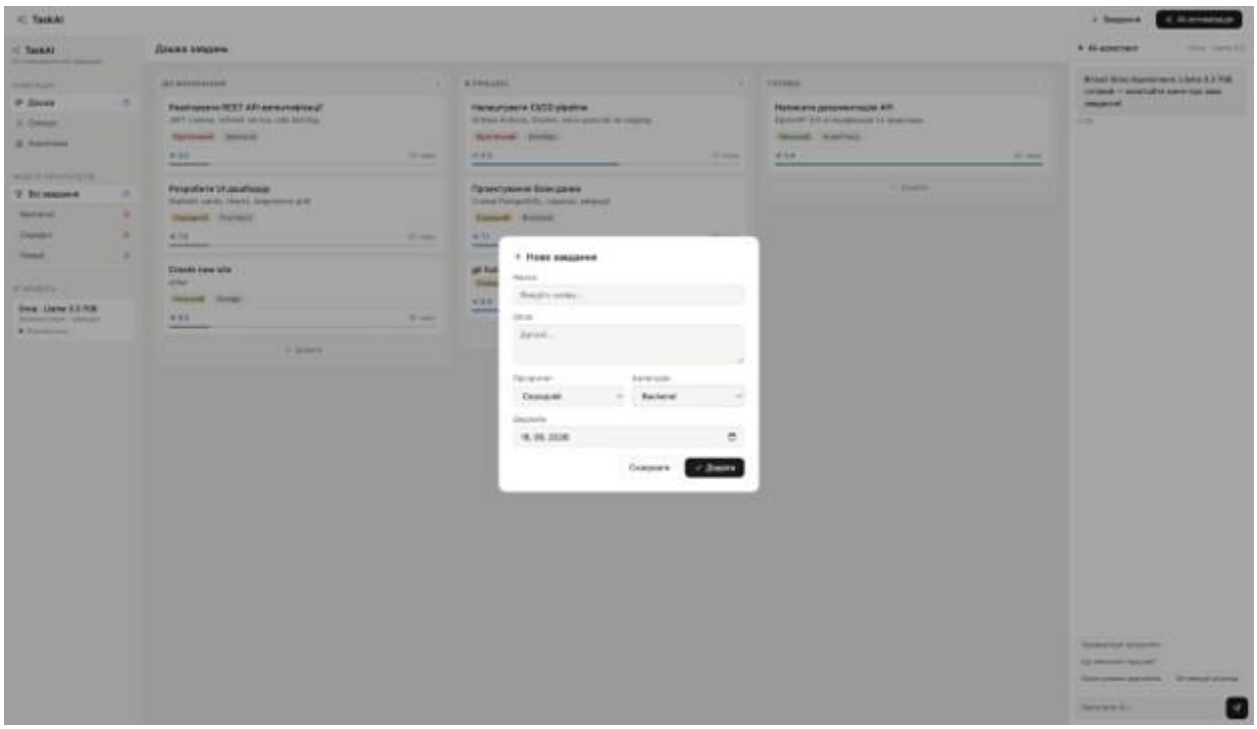
На екранах із шириною менше 1024px бічна панель навігації приховується та замінюється компактною горизонтальною смугою навігації. AI-панель переміщується у нижню частину екрану та може бути згорнута. На екранах менше 768px Kanban-дошка перемикається з триколонкового на одноколонковий макет, відображаючи колонки послідовно по вертикалі.

Таким чином, проєктування інтерфейсу розроблюваного сервісу базується на глибокому розумінні потреб цільової аудиторії, перевірених принципах UX-проєктування та продуманій візуальній системі. Прийняті дизайнерські рішення забезпечують баланс між функціональністю та простотою, між виразністю та нейтральністю, що є оптимальним для інструменту продуктивності. Детальна реалізація спроєктованих екранів описана у третьому розділі кваліфікаційної роботи. Спроєктований макет Kanban-дошки застосунку зображено на малюнку 2.2.



Малюнок 2.2 - скріншот Kanban-дошки застосунку

Макет модального вікна додавання завдання зображено на малюнку 2.3.



Малюнок 2.3 - скріншот модального вікна додавання завдання.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ВЕБ-СЕРВІСУ

Третій розділ є практичним ядром кваліфікаційної роботи і присвячений безпосередній реалізації веб-сервісу TaskAI на основі архітектурних та дизайнерських рішень, визначених у другому розділі. У розділі детально описується реалізація компонентної структури React, системи управління станом Zustand, інтеграції Groq API для забезпечення AI-функціональності, а також результати перевірки коректності роботи розробленого застосунку.

3.1. Реалізація клієнтської частини на основі React

Компонентний підхід є фундаментальною концепцією React, що визначає спосіб організації коду інтерфейсу. Замість монолітної розмітки інтерфейс розбивається на незалежні, перевикористовувані блоки — компоненти, кожен з яких відповідає за певну частину відображення та власну логіку взаємодії [7]. Кожен компонент отримує дані через підписку на Zustand-стор або через пропси від батьківського компонента, відображає відповідний фрагмент інтерфейсу та реагує на дії користувача, викликаючи екшени стору. У даному підрозділі розглянемо реалізацію кожного компонента застосунку, обґрунтуємо прийняті технічні рішення та наведемо ключові фрагменти коду.

3.1.1. Точка входу та кореневий компонент

Файл main.jsx є точкою входу застосунку — саме з нього починається виконання коду після завантаження сторінки браузером. У цьому файлі відбувається монтування кореневого React-компонента до DOM-елемента з ідентифікатором root, підключення глобальних стилів та шрифту іконок Tabler Icons. На відміну від Redux-застосунків, що потребують обгортання у компонент Provider, Zustand не вимагає жодних провайдерів на кореневому рівні — стор доступний з будь-якого компонента через імпорт хука useStore.

Кореневий компонент App.jsx визначає загальний макет застосунку та підключає всі компоненти верхнього рівня. Структурно він організований у вигляді фіксованого flex-контейнера з вертикальним напрямком, що займає повну висоту вікна браузера. Зверху розміщується компонент AppBar, нижче — робоча область у вигляді горизонтального flex-контейнера, що містить Sidebar, основну контентну зону та AiPanel. Компонент TaskModal розміщений поза основним макетом як фіксований оверлей — він рендериться лише тоді, коли modalOpen у сторі має значення true, що реалізовано через умовний рендеринг.

Умовний рендеринг активного режиму відображення в основній контентній зоні реалізований через тернарний ланцюжок на основі значення view зі стору: при значенні «board» відображається компонент Board, при «list» — List, при «analytics» — Analytics. Такий підхід є простим та ефективним для трьох фіксованих режимів і не потребує складнішої системи маршрутизації.

3.1.2. Компонент AppBar

Компонент AppBar є постійним елементом інтерфейсу, що відображається у верхній частині застосунку на всіх режимах відображення. Він виконує дві ключові функції: надає доступ до дії додавання нового завдання через кнопку «Завдання» та запускає процедуру AI-оптимізації через кнопку «AI-оптимізація».

З технічної точки зору AppBar є одним із найпростіших компонентів застосунку — він підписується лише на два екшени зі стору: openModal для відкриття модального вікна та optimize з хука useAI для запуску AI-аналізу. Кнопка «AI-оптимізація» отримує проп disabled зі значенням aiLoading, що унеможливлює повторне натискання під час очікування відповіді від Groq API та запобігає дублюванню запитів.

Стилізація компонента реалізована через AppBar.module.css із використанням CSS-змінних для кольорів та border-radius. Нижня межа

панелі реалізована через border-bottom замість box-shadow, що відповідає сучасному «пласкому» стилю дизайну та забезпечує більш чіткий розподіл між панеллю та робочою областю.

3.1.3. Компонент Sidebar

Компонент Sidebar виконує подвійну функцію: навігація між режимами відображення та фільтрація завдань за пріоритетом. Це поєднання є свідомим архітектурним рішенням, що дозволяє розмістити обидві функції в єдиному постійно видимому місці без необхідності окремих елементів навігації.

Навігаційна частина містить три пункти — «Дошка», «Список» та «Аналітика» — кожен із яких при натисканні викликає екшен setView відповідним значенням. Активний пункт відображається з виділеним стилем через умовне застосування класу navItemActive з CSS Module замість navItem [21].

Секція фільтрів пріоритету містить чотири пункти: «Всі завдання», «Критичні», «Середні» та «Низькі». Кожен пункт відображає лічильник кількості завдань відповідного пріоритету, що обчислюється безпосередньо у компоненті через фільтрацію масиву tasks зі стору. Лічильники оновлюються реактивно при кожній зміні списку завдань, що забезпечує актуальність інформації в реальному часі.

У нижній частині бічної панелі розміщено інформаційну картку активної AI-моделі з назвою «Groq · Llama 3.3 70B», статусом підключення та пульсуючим зеленим індикатором. Анімація пульсації реалізована через CSS-анімацію @keyframes pulse, що циклічно змінює opacity елемента, імітуючи живий пульс активного з'єднання.

3.1.4. Компонент Board та TaskCard

Компонент Board є центральним функціональним елементом застосунку та реалізує Kanban-дошку у вигляді трьох колонок. Він отримує відфільтрований масив завдань через виклик filteredTasks зі стору — це

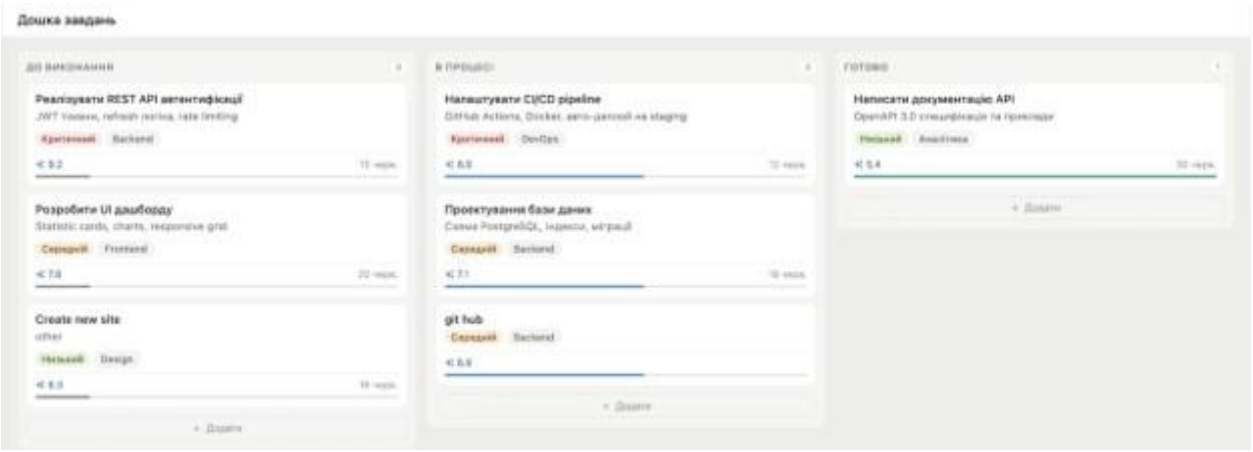
функція-геттер, що обчислює результат на льоту на основі поточного filterPriority, та розподіляє їх по колонках відповідно до значення поля status.

Розподіл завдань по колонках реалізований через метод reduce, що формує об'єкт із трьома масивами: todo, inprogress та done. Цей підхід є ефективнішим за три окремі виклики filter, оскільки вимагає лише одного проходу по масиву незалежно від його розміру.

Кожна колонка рендериться з заголовком, лічильником завдань та кнопкою швидкого додавання. При кліку на кнопку швидкого додавання відображається стандартний браузерний prompt для введення назви — це свідоме рішення на користь простоти, що дозволяє уникнути складнішого вбудованого поля редагування без суттєвої втрати зручності для швидких операцій.

Компонент TaskCard є найбільш перевикористовуваним елементом застосунку та відображає всю ключову інформацію про завдання в компактному форматі. Він приймає єдиний проп task — об'єкт завдання — та самостійно отримує необхідні константи з модуля lib/constants.js для відображення людиночитних назв пріоритетів та статусів.

Клас тегу пріоритету визначається через об'єкт-мапу TAG_CLASS, що зіставляє значення priority з відповідним CSS-класом з модуля: high відповідає tagHigh, med — tagMed, low — tagLow. Такий підхід є більш декларативним та читабельним порівняно з ланцюжком умовних операторів. Прогрес-бар реалізований через об'єкти STATUS_PROGRESS та STATUS_COLORS з константного модуля, що повертають числове значення ширини та колір заливки відповідно до статусу завдання. Kanban-дошку із завданнями у всіх трьох колонках зображено на малюнку 3.1.



Малюнок 3.1 - Kanban-дошка крупно з картками.

3.1.5. Компоненти List та Analytics

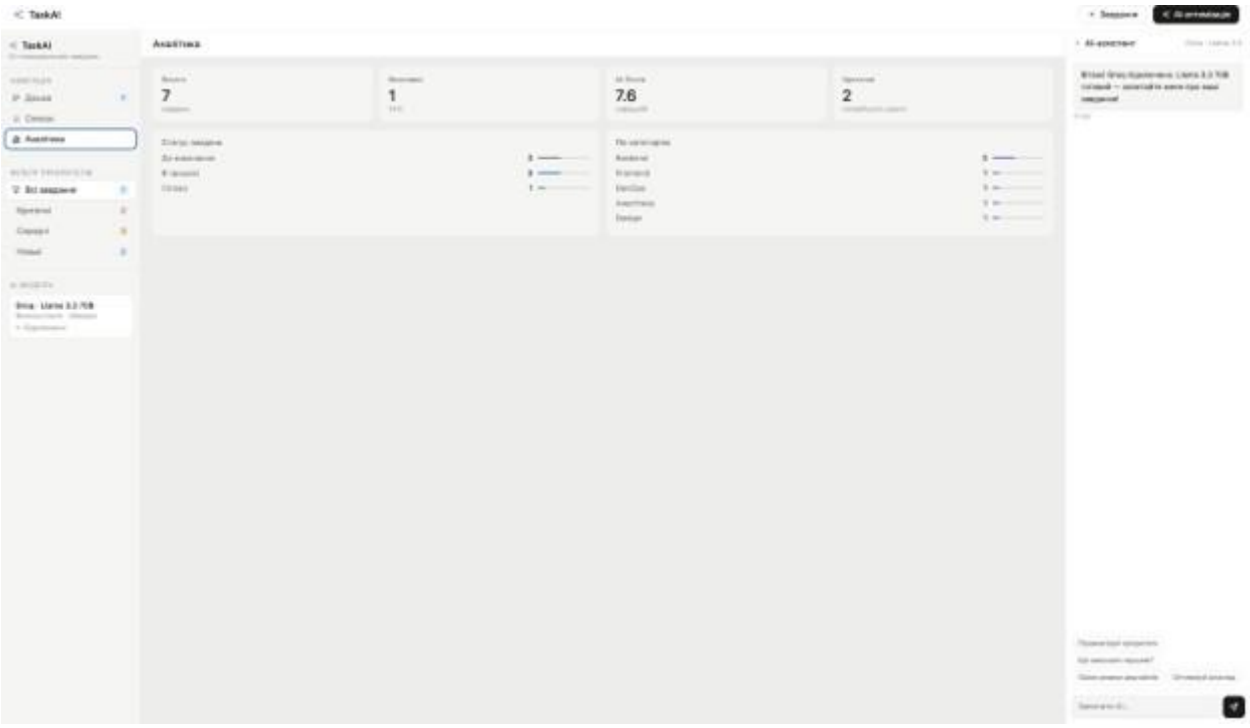
Компонент List відображає ті самі завдання, що й Board, однак у вигляді впорядкованого вертикального списку. Перед рендерингом компонент сортує відфільтровані завдання за полем score у спадному порядку через метод sort із порівняльною функцією. Важливою деталлю є те, що sort викликається на копії масиву — через spread-оператор [...filteredTasks] — а не на оригіналі, що є критичним для уникнення мутацій стану, несумісних з реактивністю Zustand.

При порожньому масиві відфільтрованих завдань компонент відображає порожній стан із іконкою та повідомленням, що пропонує змінити фільтр або додати нові завдання. Цей патерн empty state є обов'язковим елементом якісного UX та запобігає відчуттю «зламаного» інтерфейсу.

Компонент Analytics обчислює всі статистичні показники безпосередньо під час рендерингу на основі поточного масиву відфільтрованих завдань. Загальна кількість, кількість виконаних завдань, кількість завдань у процесі виконання, кількість критичних завдань та середній AI-score обчислюються через методи filter та reduce. Розподіл за категоріями формується через метод reduce, що будує об'єкт-лічильник із назвами категорій як ключами та кількістю завдань як значеннями.

Відсоткові показники для прогрес-барів обчислюються як відношення кількості завдань певного типу до загальної кількості, помножене на 100. При

нульовій загальній кількості завдань компонент відображає відповідний порожній стан замість числень ділення на нуль, що є прикладом захисного програмування. Розділ аналітики із заповненими показниками зображено на малюнку 3.2.



Малюнок 3.2 - розділ аналітики.

3.1.6. Компонент TaskModal

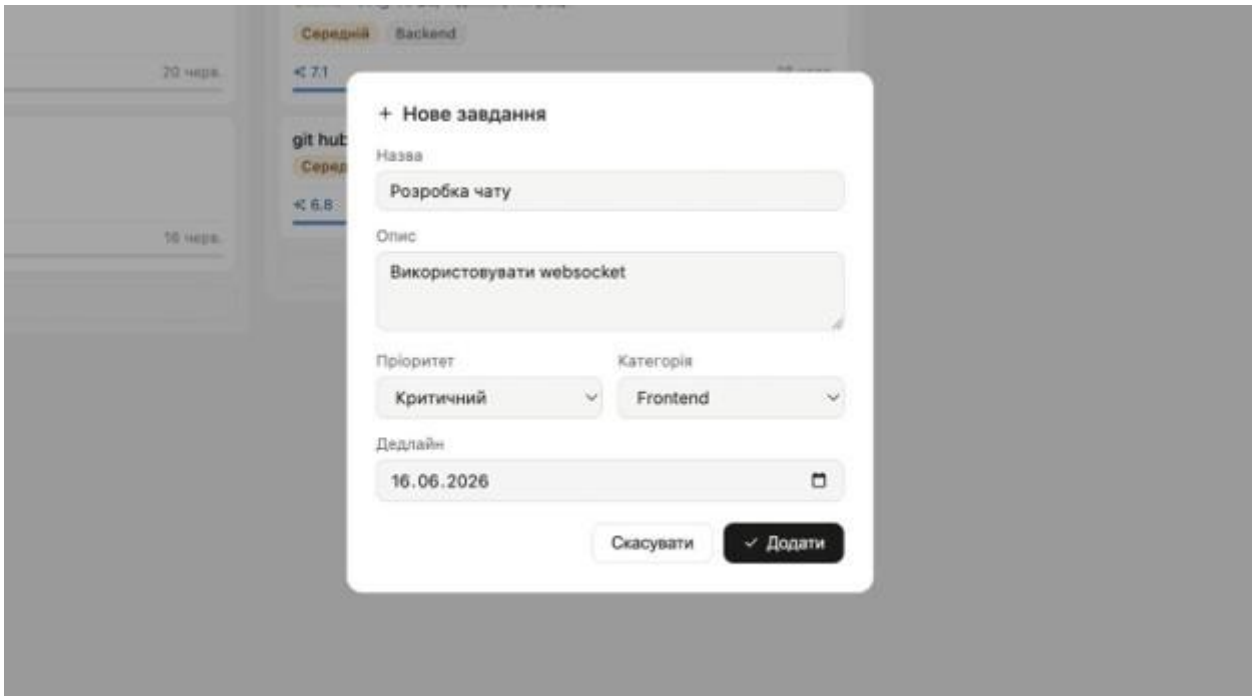
Компонент TaskModal реалізує форму додавання нового завдання у вигляді модального вікна. Він демонструє принцип локального управління станом форми: поля форми зберігаються у локальному стані через useState, тоді як глобальний стор залучається лише у момент підтвердження для збереження нового завдання.

Стан форми організований як єдиний об'єкт form із полями title, desc, priority, cat та due. Функція set, визначена всередині компонента, оновлює одне конкретне поле через spread-оператор, зберігаючи решту значень незмінними. Такий підхід є лаконічнішим за окремий useState для кожного поля і добре масштабується при додаванні нових полів.

Функція handleSubmit виконує базову валідацію — перевіряє наявність непорожньої назви — та при успішній перевірці викликає екшен addTask зі стору, передаючи поточний стан форми. Після збереження завдання форма

скидається до початкових значень через присвоєння INITIAL_STATE, а модальне вікно закривається через екшен closeModal. Додатково через екшен addMessage у AI-панель надсилається підтвердження про успішне додавання завдання з його назвою та згенерованим score.

Анімація появи та зникнення модального вікна реалізована через CSS-анімації fadeIn для оверлею та slideUp для самого модального вікна, визначені в TaskModal.module.css. Анімація slideUp переміщує вікно з позиції translateY(8px) до translateY(0) одночасно зі зміною opacity від 0 до 1, що створює природний ефект «спливання» з нижнього краю. Модальне вікно додавання завдання із заповненими полями зображено на малюнку 3.3.



Малюнок 3.3 - модальне вікно з формою.

3.1.7. Компонент AiPanel

Компонент AiPanel реалізує інтерфейс чату з AI-асистентом та є одним із найбільш технічно насичених компонентів застосунку. Він поєднує відображення статичного списку повідомлень із динамічним оновленням при надходженні нових відповідей та управління полем введення.

Автоматична прокрутка до останнього повідомлення реалізована через useRef та useEffect: реф bottomRef вказує на порожній div-елемент у кінці

списку повідомлень, а ефект викликає scrollIntoView при кожній зміні масиву messages або значення aiLoading. Це забезпечує автоматичне відображення найновіших повідомлень без необхідності ручного прокручування.

Анімований індикатор введення тексту (typing indicator) відображається як окремий елемент у кінці списку повідомлень при aiLoading === true. Він реалізований через три span-елементи з CSS-анімацією dot, що почергово змінюють opacity з затримкою 0.2 секунди між кожним, створюючи ефект послідовного пульсування.

Швидкі запити (чіпи) реалізовані як масив рядків AI_CHIPS з константного модуля. При кліку на чіп його текст встановлюється як значення поля введення та одразу викликається функція відправки, минаючи ручне підтвердження. Це значно прискорює типові взаємодії з AI-асистентом для найпоширеніших запитів.

Обробка натискання Enter у полі введення реалізована через обробник onKeyDown: при натисканні клавіші Enter без Shift викликається handleSend та запобігається стандартна дія браузера (перенесення рядка) через e.preventDefault(). При натисканні Shift+Enter вставляється новий рядок, що дозволяє формулювати багаторядкові запити.

Таким чином, компонентна структура застосунку реалізована відповідно до принципу єдиної відповідальності: кожен компонент виконує одну чітко визначену функцію, є незалежним та максимально перевикористовуваним. Чіткий розподіл між компонентами, що підписані на глобальний стан, та компонентами з локальним станом забезпечує зрозумілість архітектури та спрощує подальше розширення функціональності сервісу.

3.2. Управління станом застосунку засобами Zustand

Управління станом є одним із найважливіших аспектів архітектури будь-якого складного фронтенд-застосунку. У даному підрозділі детально розглянемо практичну реалізацію системи управління станом розроблюваного сервісу засобами Zustand: організацію стору, реалізацію

кожного домену стану, логіку екшенів та механізм персистентності даних між сесіями.

3.2.1. Організація Zustand-стору

Центральним елементом Zustand-архітектури є стор — єдине джерело правди для всього динамічного стану застосунку. На відміну від Redux, де стор конфігурується через окрему функцію configureStore і підключається до застосунку через компонент Provider, Zustand-стор є простим JavaScript-модулем, що експортує хук useStore [15]. Будь-який компонент може підписатись на будь-яку частину стану через деструктуризацію з цього хука без жодного додаткового налаштування.

Стор розроблюваного застосунку визначений у файлі store/useStore.js та обгортається двома middleware: persist для автоматичного збереження стану у localStorage та devtools для інтеграції з браузерним розширенням Zustand DevTools. Middleware persist отримує конфігурацію з іменем ключа у localStorage — «taskai-store» — та функцією partialize, що визначає, яка частина стану підлягає збереженню [15]. Зокрема, зберігаються лише поля tasks та nextId із домену завдань: UI-стан та повідомлення AI-чату свідомо виключені з персистентності, оскільки їхнє відновлення між сесіями не є доцільним.

Внутрішня організація стору базується на концепції слайсів — логічно відокремлених груп стану та екшенів, що відповідають за певний домен. У даному проєкті виділено три слайси: tasks, ui та ai. Кожен слайс реалізований як окрема функція-фабрика, що приймає set та get як аргументи та повертає об'єкт із початковим станом та екшенами. Фінальний стор об'єднує всі три слайси через spread-оператор у єдиному виклику create.

3.2.2. Реалізація домену tasks

Домен tasks є найважливішим у застосунку та управляє повним масивом завдань та лічильником для генерації унікальних ідентифікаторів. Початковий стан містить масив DEFAULT_TASKS з п'яти демонстраційних завдань, що завантажуються при першому запуску, та nextId зі значенням 6.

Екшен addTask реалізує логіку додавання нового завдання. Він приймає об'єкт із полями форми, доповнює його автоматично генерованими полями id та score та додає до масиву tasks через spread-оператор. Значення score генерується функцією randomScore з модуля constants.js, що повертає випадкове число від 6 до 10 з одним знаком після коми — діапазон обраний таким чином, щоб усі нові завдання мали відносно високий початковий рейтинг, що стимулює користувача до взаємодії з AI для більш точної пріоритизації. Функція повертає створений об'єкт завдання, що дозволяє компоненту TaskModal відобразити його назву та score у підтверджуючому повідомленні AI-чату.

Екшен quickAddTask призначений для швидкого додавання завдання безпосередньо з колонки Kanban-дошки. Він приймає три параметри: status визначає колонку, до якої додається завдання, title є назвою, priority за замовчуванням приймає значення «med». На відміну від addTask, quickAddTask не відкриває модальне вікно і не потребує заповнення додаткових полів, забезпечуючи мінімальне тертя при швидких операціях планування.

Функція filteredTasks є геттером — функцією, що обчислює похідне значення на основі поточного стану. Вона зчитує tasks та filterPriority з відповідних доменів та повертає відфільтрований масив: при filterPriority === «all» повертається повний масив, в іншому випадку — масив, відфільтрований за полем priority. Геттер викликається кожного разу, коли компонент звертається до нього через useStore, що гарантує актуальність результату без необхідності у мемоїзації для масивів поточного розміру.

3.2.3. Реалізація домену ui

Домен ui управляє всіма аспектами стану інтерфейсу, що не є даними предметної області. Він містить три поля: view з початковим значенням «board», filterPriority з початковим значенням «all» та modalOpen з початковим значенням false.

Екшени setView та setFilterPriority є тривіальними — кожен з них оновлює відповідне поле стану переданим значенням через виклик set. Попри свою простоту, винесення цих екшенів у стор замість локального useState у компонентах є архітектурно правильним рішенням, оскільки поточний режим відображення та активний фільтр є спільним станом, що впливає на кілька компонентів одночасно: Sidebar відображає активний пункт навігації, основна контентна область відображає відповідний компонент, заголовок у mhead відображає відповідну назву розділу.

Екшени openModal та closeModal керують видимістю модального вікна додавання завдання. Компонент TaskModal підписується на поле modalOpen та рендериться лише при його значенні true. Такий підхід є поширеним патерном для управління видимістю модальних вікон у React-застосунках і є кращою альтернативою передачі колбеків через ланцюжок пропсів.

3.2.4. Реалізація домену ai

Домен ai управляє станом AI-чату: масивом повідомлень та прапорцем завантаження. Початковий стан масиву messages містить одне вітальне повідомлення від AI-асистента з поточним часом, що генерується у момент ініціалізації стору.

Кожне повідомлення є об'єктом із трьома полями: role приймає значення «user» або «ai», text містить текст повідомлення та ts містить відформатовану рядкову мітку часу, що генерується через toLocaleTimeString при додаванні повідомлення. Зберігання мітки часу у рядковому форматі замість об'єкта Date є свідомим рішенням для спрощення серіалізації — попри те що домен ai виключений з persist, це забезпечує сумісність у разі майбутньої зміни конфігурації.

Екшен addMessage додає нове повідомлення до масиву через spread-оператор та автоматично генерує мітку часу через допоміжну функцію now, визначену локально в модулі стору. Екшен setAILoading встановлює прапорець завантаження, що використовується компонентом AiPanel для

відображення typing indicator та компонентом AppBar для блокування кнопки AI-оптимізації.

3.2.5. Персистентність стану через localStorage

Механізм персистентності є критично важливою функцією для застосунку управління завданнями: користувач очікує, що його завдання збережуться між сесіями роботи у браузері. Zustand реалізує персистентність через middleware persist, що автоматично серіалізує визначену частину стану у JSON та зберігає її у localStorage при кожному оновленні, а при ініціалізації стору зчитує збережені дані та відновлює стан.

Конфігурація middleware persist включає функцію partialize, що отримує повний стан та повертає лише ту його частину, що підлягає збереженню. У даному проєкті зберігаються tasks та nextId, тоді як view, filterPriority, modalOpen, messages та aiLoading відновлюються до початкових значень при кожному запуску. Це є коректним рішенням: користувач завжди починає сесію з дошкою без активних фільтрів, а повідомлення попередніх сесій з AI не є релевантними у новому контексті.

Важливим аспектом персистентності є обробка ситуації, коли збережені дані у localStorage не відповідають поточній структурі стану — наприклад, при оновленні застосунку. Zustand persist підтримує поле version та функцію migrate для обробки таких ситуацій, однак у поточній версії TaskAI ці механізми не задіяні, оскільки структура даних є стабільною.

3.2.6. Підписка на стан у компонентах

Ефективна підписка на стан є важливим аспектом продуктивності Zustand-застосунку. При некоректному використанні компоненти можуть перерендерюватись надто часто, що призводить до помітного зниження плавності інтерфейсу.

Основний принцип полягає в тому, що компонент має підписуватись лише на ті частини стану, що реально використовуються у його рендерингу. Zustand досягає цього через селектори — функції, що передаються як

аргумент у useStore та повертають конкретну частину стану або екшен. Компонент перерендериться лише тоді, коли значення, повернуте селектором, змінилось, а не при будь-якій зміні стору.

Наприклад, компонент AppBar підписується лише на openModal та optimize і не перерендериться при зміні списку завдань, активного фільтру або повідомлень AI-чату. Компонент Sidebar підписується на view, filterPriority, tasks та countByPriority і перерендериться при зміні будь-якого з цих значень, що є коректним, оскільки всі вони відображаються у компоненті. Компонент AiPanel підписується на messages та aiLoading і реагує лише на зміни у домені ai. Підписки компонентів на домени стану Zustand наведено у таблиці 3.1.

Таблиця 3.1

Підписки компонентів на домени стану Zustand

Компонент	**Домен tasks**	**Домен ui**	**Домен ai**
AppBar	—	openModal	—
Sidebar	tasks, countByPriority	view, filterPriority, setView, setFilter	—
Board	filteredTasks, quickAddTask	filterPriority	—
List	filteredTasks	—	—
Analytics	filteredTasks	—	—
AiPanel	—	—	messages, aiLoading
TaskModal	addTask	modalOpen, closeModal	addMessage

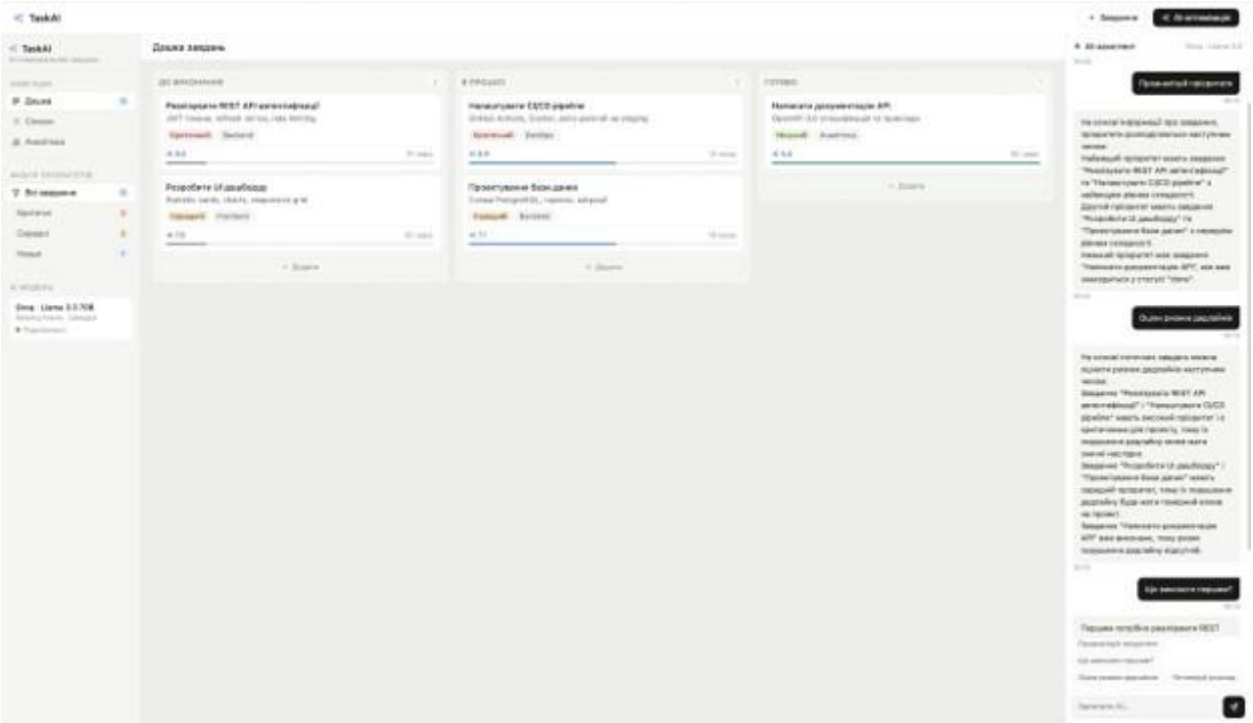
Такий розподіл підписок мінімізує кількість перерендерингів та забезпечує оптимальну продуктивність застосунку навіть при частих оновленнях стану під час активної роботи з AI-асистентом.

3.2.7. Налагодження за допомогою Zustand DevTools

Zustand підтримує інтеграцію з браузерним розширенням Redux DevTools через middleware devtools, що дозволяє використовувати потужний інструментарій налагодження, розроблений для Redux, разом із Zustand [15].

Розширення Redux DevTools доступне для браузерів Chrome та Firefox та надає розробнику повний контроль над станом застосунку в реальному часі.

У вкладці Actions відображається хронологічний журнал усіх змін стану з моменту завантаження застосунку. Для кожної зміни показується назва екшену та нове значення відповідних полів. Функція Time Travel дозволяє переміщатись між попередніми станами застосунку, що надзвичайно зручно для налагодження складних послідовностей взаємодій — наприклад, для перевірки коректності стану після серії додавань та фільтрацій завдань. Функція Diff показує різницю між поточним та попереднім станом після кожної зміни, дозволяючи точно бачити, які поля були оновлені. Стан Zustand-стору у браузерному розширенні Redux DevTools зображено на малюнку 3.4.



Малюнок 3.4 - скріншот Redux DevTools із Zustand-стором.

Таким чином, система управління станом, реалізована засобами Zustand, забезпечує централізоване, передбачуване та зручне для налагодження управління всіма динамічними даними застосунку. Поділ стану на три незалежних домени, використання геттерів для обчислення похідних значень та коректна конфігурація middleware persist формують надійну основу для подальшого розширення функціональності веб-сервісу.

3.3. Інтеграція AI-модуля на базі Groq API

Інтеграція штучного інтелекту є ключовою диференціюючою характеристикою розроблюваного сервісу TaskAI, що відрізняє його від традиційних інструментів управління завданнями. У даному підрозділі детально розглянемо практичну реалізацію AI-модуля: архітектуру взаємодії з Groq API, формування контекстних промптів, обробку відповідей моделі та інтеграцію AI-функціональності у загальну архітектуру застосунку.

3.3.1. Архітектура взаємодії з Groq API

Groq API реалізує стандартизований інтерфейс, сумісний з OpenAI API, що є поширеною галузевою практикою серед провайдерів мовних моделей [18]. Це означає, що формат запитів та відповідей є ідентичним до OpenAI Chat Completions API, що суттєво спрощує потенційну заміну провайдера у майбутньому без суттєвих змін у коді застосунку.

Взаємодія з API побудована за схемою «системний промпт плюс повідомлення користувача». Системний промпт визначає роль та поведінку моделі і передається у кожному запиті у полі system масиву messages з роллю «system». Повідомлення користувача передається з роллю «user». Модель повертає відповідь з роллю «assistant» у полі choices[0].message.content результуючого об'єкта.

Весь код взаємодії з Groq API інкапсульований у файлі lib/groq.js, що експортує єдину асинхронну функцію callGroq. Вона приймає два рядкових аргументи — system та user — та повертає Promise, що розв'язується у рядок відповіді моделі або відхиляється з об'єктом помилки при невдалому запиті. Такий підхід реалізує принцип єдиної відповідальності: весь мережевий код зосереджений в одному місці та повністю відокремлений від компонентів інтерфейсу та логіки стору.

HTTP-запит реалізований через нативний Fetch API браузера без використання додаткових HTTP-бібліотек. Тіло запиту формується як JSON-об'єкт із полями model, max_tokens та messages. Значення model встановлено

як «llama-3.3-70b-versatile» — це ідентифікатор моделі Llama 3.3 70B у реєстрі Groq [19]. Значення max_tokens встановлено рівним 1024, що забезпечує достатньо довгі та детальні відповіді для аналізу завдань без надмірних витрат квоти API.

У заголовках запиту передаються Content-Type зі значенням «application/json» та Authorization у форматі Bearer з API-ключем. API-ключ зберігається безпосередньо у файлі groq.js як константа, що є прийнятним рішенням для навчального проєкту, однак у виробничому середовищі ключ слід зберігати у змінних середовища та не включати до системи контролю версій.

3.3.2. Хук useAI

Хук useAI є центральним елементом AI-модуля та інкапсулює всю бізнес-логіку взаємодії із Groq API. Він визначений у файлі hooks/useAI.js та надає два методи: ask для відправки довільного запиту від користувача та optimize для запуску процедури AI-оптимізації всього списку завдань.

Хук отримує необхідні дані та екшени зі стору через useStore: масив tasks для формування контексту запиту, addMessage для додавання повідомлень до чату, setAILoading для управління станом завантаження та loading для перевірки активного запиту. Перевірка if (loading) return на початку обох методів запобігає відправці паралельних запитів при швидких повторних натисканнях.

Функція ask виконує такий алгоритм: спочатку додає повідомлення користувача до чату через addMessage, потім встановлює aiLoading у true, формує системний промпт із поточним списком завдань, відправляє запит через callGroq та при отриманні відповіді додає її до чату та скидає aiLoading у false. Весь асинхронний код обгортається у try-catch для обробки можливих помилок мережі або API, при яких у чат додається повідомлення з текстом помилки.

Функція optimize відрізняється від ask лише форматом промпту: вона автоматично формує запит на аналіз та оптимізацію без участі користувача та використовує інший системний промпт, орієнтований на формування структурованих рекомендацій у вигляді нумерованого списку.

3.3.3. Формування контекстних промптів

Якість відповідей мовної моделі безпосередньо залежить від якості промптів. Ретельне проєктування системних промптів є важливим етапом розробки AI-інтегрованих застосунків, що часто називають «prompt engineering» — інженерією промптів. У даному проєкті застосовано кілька ключових принципів цієї дисципліни [26].

Принцип чіткого визначення ролі. Системний промпт починається з однозначного визначення ролі моделі: «Ти AI-асистент планувальника завдань TaskAI». Це налаштовує модель на відповідний контекст та запобігає невідповідним відповідям, що виходять за межі предметної області сервісу.

Принцип включення контексту. Системний промпт включає поточний список завдань у структурованому текстовому форматі, що генерується функцією tasksSummaryForAI з модуля constants.js. Кожне завдання відображається рядком формату «- [назва] [[пріоритет], [категорія], [статус], score:[значення]]». Такий компактний формат передає всі ключові атрибути завдання в мінімальній кількості токенів, що важливо для ефективного використання контекстного вікна моделі.

Принцип обмеження формату відповіді. Для функції ask системний промпт містить вимогу відповідати коротко — у 2-4 реченнях — та по-діловому українською мовою. Обмеження обсягу відповіді є важливим для підтримання природного темпу діалогу та уникнення надмірно довгих відповідей, що ускладнюють сприйняття у форматі чат-інтерфейсу.

Принцип структурованості для аналітичних запитів. Для функції optimize системний промпт містить вимогу формулювати відповідь у вигляді «топ-3 рекомендацій нумерованим списком». Структурований формат

полегшує сприйняття аналітичних рекомендацій та дозволяє користувачеві швидко виділити ключові дії.

3.3.4. Обробка відповідей та відображення у чаті

Після отримання відповіді від Groq API хук useAI витягує текст із поля choices[0].message.content результуючого об'єкта та передає його в екшен addMessage зі роллю «ai». Компонент AiPanel підписаний на масив messages у сторі та автоматично оновлює відображення при додаванні нового елемента.

Відображення повідомлень реалізоване через компонент Message всередині AiPanel, що приймає об'єкт повідомлення та рендерить бульбашку з текстом та міткою часу. CSS-клас бульбашки залежить від ролі: для повідомлень користувача застосовується клас msgUser з темним фоном та вирівнюванням праворуч, для повідомлень AI — стандартний клас із світлим фоном та вирівнюванням ліворуч. Текст повідомлення рендериться як є, зберігаючи переноси рядків, що є важливим для коректного відображення структурованих відповідей із нумерованими списками від функції optimize.

Обробка помилок реалізована таким чином, що користувач завжди отримує зрозуміле повідомлення незалежно від причини невдачі. При помилці мережі або неправильному API-ключі у чат додається повідомлення від AI з текстом помилки, що дозволяє користувачеві зрозуміти проблему без необхідності відкривати консоль розробника.

3.3.5. Тестування AI-функціональності

Перевірка коректності роботи AI-модуля проводилась за кількома напрямами, що охоплюють як функціональні аспекти, так і якість відповідей моделі.

Перевірка базової функціональності чату. Для перевірки базової роботи чату було відправлено серію запитів різних типів: прості запитання про поточний стан завдань («Скільки завдань у мене в процесі виконання?»), запити на рекомендації («Що варто виконати першим?») та запити на аналіз ризиків («Які завдання найімовірніше будуть прострочені?»). У всіх випадках

модель коректно відображала актуальний стан завдань у своїх відповідях, що підтверджує правильність формування контекстного промпту.

Перевірка функції AI-оптимізації. При натисканні кнопки «AI-оптимізація» модель коректно аналізувала повний список завдань та формувала структуровані рекомендації у вигляді нумерованого списку з трьох пунктів. Рекомендації були конкретними та релевантними: модель правильно виділяла завдання з критичним пріоритетом та наближеними дедлайнами як першочергові.

Перевірка реакції на зміни стану. Після додавання нового завдання через модальне вікно наступний запит до AI-асистента коректно враховував нове завдання у своїх відповідях, що підтверджує, що системний промпт формується динамічно на основі актуального стану стору в момент відправки запиту, а не кешується.

Перевірка обробки помилок. При відключенні мережевого з'єднання та відправці запиту застосунок коректно відображав повідомлення про помилку у чаті без зависання інтерфейсу. Кнопка відправки та кнопка AI-оптимізації коректно розблоковувались після обробки помилки, що дозволяє повторити запит після відновлення з'єднання.

Перевірка стану завантаження. Під час очікування відповіді від Groq API typing indicator коректно відображався у чаті, кнопка відправки блокувалась та кнопка AI-оптимізації в AppBar переходила у стан disabled. Після отримання відповіді всі елементи керування коректно повертались до активного стану. Результати перевірки функціональності AI-модуля наведено у таблиці 3.2.

Таблиця 3.2

Результати перевірки функціональності AI-модуля

Сценарій перевірки	Очікуваний результат	Фактичний результат	Статус
Запит про стан завдань	Коректна відповідь з даними зі стору	Відповідь враховує актуальні завдання	✓ Пройдено

Сценарій перевірки	Очікуваний результат	Фактичний результат	Статус
Запит рекомендацій	Конкретні рекомендації з обґрунтуванням	Нумерований список рекомендацій	✓ Пройдено
AI-оптимізація	Топ-3 рекомендації нумерованим списком	Структурована відповідь з 3 пунктів	✓ Пройдено
Запит після додавання завдання	Нове завдання враховане у відповіді	Нове завдання присутнє у контексті	✓ Пройдено
Запит при відсутності мережі	Повідомлення про помилку у чаті	Помилка відображена, UI не завис	✓ Пройдено
Повторний запит під час завантаження	Запит ігнорується	Дублювання запитів відсутнє	✓ Пройдено
Typing indicator під час очікування	Анімація трьох крапок у чаті	Індикатор відображається коректно	✓ Пройдено

3.3.6. Загальне тестування застосунку

Окрім перевірки AI-модуля, проведено загальне тестування всього застосунку за основними користувацькими сценаріями, визначеними у підрозділі 2.3.1.

Тестування Kanban-дошки. Перевірено коректність відображення завдань у колонках відповідно до їхнього статусу, відображення порожнього стану при відсутності завдань у колонці, коректність роботи кнопки швидкого додавання у кожній колонці та відображення кольорових тегів пріоритетів.

Тестування фільтрації. При активації фільтру «Критичні» Kanban-дошка, список та аналітика коректно відображають лише завдання з priority === «high». Лічильники у бічній панелі оновлюються при додаванні нових завдань. Перемикання між фільтрами відбувається миттєво без будь-яких затримок.

Тестування персистентності. Після додавання кількох завдань та перезавантаження сторінки браузера всі завдання коректно відновлювались зі стану localStorage. UI-стан коректно скидався до початкових значень: дошка відкривалась без активних фільтрів.

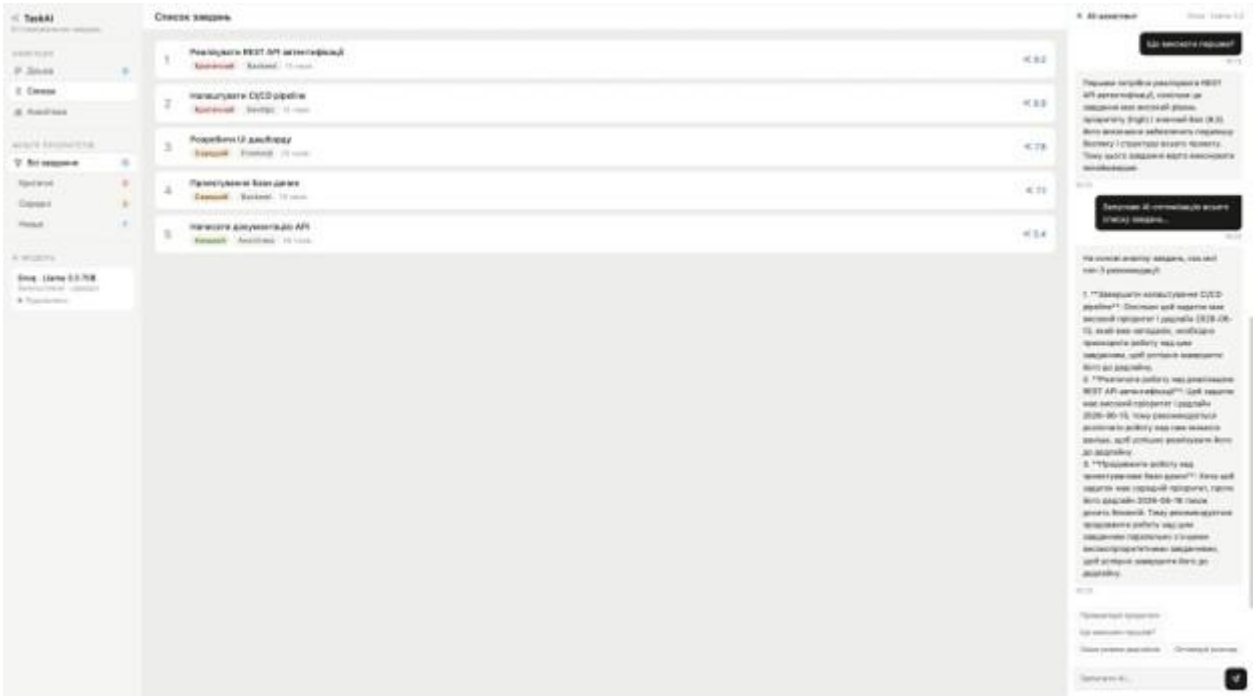
Тестування темної теми. При зміні системних налаштувань операційної системи на темну тему застосунок автоматично перемикається на темну кольорову схему завдяки медіа-запиту prefers-color-scheme: dark у глобальному CSS. Всі елементи інтерфейсу коректно відображаються в обох темах без проблем із контрастністю. Зведені результати тестування застосунку TaskAI подано у таблиці 3.3.

Таблиця 3.3 -

Зведені результати тестування застосунку TaskAI

Функціональний модуль	**Кількість перевірених сценаріїв**	**Пройдено**	**Не пройдено**
Kanban-дошка	6	6	0
Список завдань	4	4	0
Аналітика	4	4	0
Фільтрація за пріоритетом	5	5	0
Модальне вікно додавання	5	5	0
Персистентність даних	3	3	0
AI-чат	7	7	0
Темна тема	2	2	0
Разом	**36**	**36**	**0**

Розділ «Список» із завданнями, відсортованими за AI-score, зображено на малюнку 3.5.



Малюнок 3.5 - розділ «Список».

Таким чином, інтеграція AI-модуля на базі Groq API реалізована як повноцінна функціональність, що органічно вписується в архітектуру застосунку. Чіткий поділ між мережевим кодом у lib/groq.js, бізнес-логікою у hooks/useAI.js та відображенням у компоненті AiPanel забезпечує зрозумілість реалізації та спрощує потенційну заміну AI-провайдера або моделі. Проведене тестування підтверджує коректність роботи всіх компонентів системи та готовність застосунку до реального використання.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи на тему «Проєктування та реалізація веб-сервісу для оптимізації планування робочих завдань за допомогою штучного інтелекту» було досягнуто поставленої мети та вирішено всі визначені завдання. Підсумуємо основні результати проведеного дослідження та практичної розробки.

У першому розділі здійснено комплексний аналіз теоретичних основ предметної галузі. Досліджено поняття та еволюцію систем управління завданнями — від ізольованих настільних застосунків кінця 1980-х до сучасних хмарних платформ із AI-функціональністю. Розроблено класифікацію таких систем за чотирма критеріями: цільовою аудиторією, моделлю відображення, способом розгортання та ступенем інтелектуалізації. Визначено п'ять рівнів AI-інтеграції у системи управління завданнями та досліджено технічні підходи до впровадження великих мовних моделей у веб-застосунки. Проведено порівняльний аналіз провідних платформ — Trello, Asana, Notion, ClickUp, Linear та Motion — що дозволив виявити ключову ринкову нішу: жодна з безкоштовних платформ не надає повноцінного інтерактивного AI-асистента, здатного аналізувати конкретні завдання користувача у реальному часі та відповідати українською мовою. Саме цю нішу заповнює розроблений сервіс TaskAI.

У другому розділі здійснено вибір технологічного стеку та спроєктовано архітектуру застосунку. На основі порівняльного аналізу обґрунтовано вибір React 18 як основної бібліотеки для побудови інтерфейсу, Vite як інструменту збірки з практично миттєвим холодним стартом, Zustand як системи управління станом із мінімалістичним API та нативною підтримкою персистентності, а також Groq API з моделлю Llama 3.3 70B як єдиного безкоштовного провайдера з достатньою якістю відповідей та підтримкою української мови. Спроєктовано розшаровану архітектуру застосунку з чітким поділом на шар представлення, шар управління станом,

шар бізнес-логіки та шар даних. Визначено модель даних завдання з вісьмома полями та структуру Zustand-стору з трьома незалежними доменами. Розроблено систему дизайну з кольоровою палітрою на основі CSS-змінних, що забезпечує повну підтримку світлої та темної теми без жодного JavaScript. Спроєктовано чотири основні користувацькі сценарії та обґрунтовано UX-рішення для кожного з них відповідно до потреб цільової аудиторії — фахівців у сфері IT та студентів технічних спеціальностей.

У третьому розділі реалізовано повнофункціональний веб-сервіс для оптимізації планування завдань. Розроблено компонентну структуру застосунку із семи основних компонентів, кожен з яких реалізований з власним CSS Module для повної ізоляції стилів. Реалізовано систему управління станом на основі трьох Zustand-доменів та AI-модуль у вигляді файлу lib/groq.js із функцією callGroq та хука hooks/useAI.js із методами ask та optimize. Застосовано принципи prompt engineering для формування контекстних системних промптів, що включають актуальний список завдань користувача та визначають чіткий формат відповіді моделі. Проведено комплексне тестування застосунку за 36 сценаріями, що охоплюють усі функціональні модулі — Kanban-дошку, список, аналітику, фільтрацію, модальне вікно, персистентність, AI-чат та темну тему — всі сценарії успішно пройдено без виявлення критичних помилок.

Практична значимість виконаної роботи полягає в тому, що розроблений застосунок TaskAI є повнофункціональним веб-сервісом, готовим до реального використання. Він поєднує повний базовий функціональний набір сучасного task manager-a — Kanban-дошку, режим списку, аналітику, фільтрацію за пріоритетом та персистентність даних — з живим AI-асистентом на базі моделі Llama 3.3 70B, що відповідає на довільні запити українською мовою на основі актуального списку завдань користувача. Використання безкоштовного Groq API робить повноцінну AI-функціональність доступною без жодних фінансових витрат, що є суттєвою конкурентною перевагою порівняно з усіма розглянутими аналогами, де AI

доступний лише на платних тарифах. Застосований технологічний стек на базі React, Vite та Zustand відповідає актуальним стандартам промислової фронтенд-розробки та забезпечує масштабованість архітектури для подальшого розширення.

Перспективами подальшого розвитку проєкту є: реалізація повноцінного серверного бекенду на основі Node.js та NestJS із базою даних PostgreSQL для синхронізації даних між пристроями та командної роботи; впровадження системи автентифікації та особистих кабінетів користувачів; розробка мобільного застосунку на базі React Native із спільною бізнес-логікою з веб-версією; реалізація функціональності drag-and-drop для переміщення карток між колонками Kanban-дошки; інтеграція з популярними сервісами — GitHub для автоматичного створення завдань із issues та Google Calendar для синхронізації дедлайнів; впровадження більш складних AI-функцій, зокрема автоматичної декомпозиції завдань та предиктивного аналізу ризиків дедлайнів на основі накопичених даних.

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Попович Т. М., Гавриш І. В. Проєктування інформаційних систем електронної комерції : навч. посіб. Львів : Видавництво ЛТЕУ, 2022. 256 с.
2. Семенченко А. І. Архітектура сучасних фронтенд-застосунків. Комп'ютерні науки та інформаційні технології. 2023. № 1. С. 112–119.
3. Тесленко О. М. Управління станом у React-застосунках. Інформаційні технології та комп'ютерна інженерія. 2022. № 4. С. 78–85.
4. Чорний В. С. Розробка веб-застосунків на основі сучасних JavaScript-фреймворків. Вісник Львівського університету. Серія «Прикладна математика та інформатика». 2023. № 31. С. 134–142.
5. Банкрашков А. В. Веб-технології та веб-дизайн : навч. посіб. Київ : Університет «Україна», 2021. 318 с.
6. Костенко А. В., Новосад З. Г. Проєктування програмних систем : навч. посіб. Львів : Видавництво ЛТЕУ, 2023. 198 с.
7. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. Sebastopol : O'Reilly Media, 2020. 310 p.
8. Wieruch R. The Road to React: Your Journey to Master React in JavaScript. Self-published, 2023. 246 p.
9. Mardan A. React Quickly: Painless Web Apps with React, JSX, Redux, and GraphQL. 2nd ed. Shelter Island : Manning Publications, 2023. 440 p.
10. Simpson K. You Don't Know JS Yet: Get Started. 2nd ed. Sebastopol : O'Reilly Media, 2020. 278 p.
11. Fink G., Flatow I. Pro Single Page Application Development. New York : Apress, 2022. 326 p.
12. Larsen E. The Road to React: Your Journey to Master React in JavaScript. Self-published, 2023. 246 p.
13. Dabit N. Full Stack Serverless: Modern Application Development with React, AWS, and GraphQL. Sebastopol : O'Reilly Media, 2020. 220 p.

14. React — офіційна документація [Електронний ресурс]. – Режим доступу: https://react.dev
15. Zustand — офіційна документація [Електронний ресурс]. – Режим доступу: https://docs.pmnd.rs/zustand
16. Vite — офіційна документація [Електронний ресурс]. – Режим доступу: https://vite.dev
17. React Router — офіційна документація [Електронний ресурс]. – Режим доступу: https://reactrouter.com
18. Groq API — офіційна документація [Електронний ресурс]. – Режим доступу: https://console.groq.com/docs
19. Meta AI. Llama 3.3 Model Card [Електронний ресурс]. – Режим доступу: https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_3
20. Tabler Icons — офіційна документація [Електронний ресурс]. – Режим доступу: https://tabler.io/icons
21. CSS Modules — специфікація [Електронний ресурс]. – Режим доступу: https://github.com/css-modules/css-modules
22. Stack Overflow Developer Survey 2024 [Електронний ресурс]. – Режим доступу: https://survey.stackoverflow.co/2024
23. State of JS 2024: Front-end Frameworks [Електронний ресурс]. – Режим доступу: https://2024.stateofjs.com/en-US/libraries/front-end-frameworks
24. Nielsen Norman Group. 10 Usability Heuristics for User Interface Design [Електронний ресурс]. – Режим доступу: https://www.nngroup.com/articles/ten-usability-heuristics
25. Nielsen Norman Group. Response Times: The 3 Important Limits [Електронний ресурс]. – Режим доступу: https://www.nngroup.com/articles/response-times-3-important-limits
26. McKinsey Global Institute. The Economic Potential of Generative AI [Електронний ресурс]. – Режим доступу: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai

27. MDN Web Docs. JavaScript Guide [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
28. MDN Web Docs. CSS Custom Properties (Variables) [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_custom_properties
29. Web Content Accessibility Guidelines (WCAG) 2.1. W3C Recommendation [Електронний ресурс]. – Режим доступу: https://www.w3.org/TR/WCAG21
30. Atlassian. What is Kanban? [Електронний ресурс]. – Режим доступу: https://www.atlassian.com/agile/kanban.

ДОДАТКИ

Додаток 1

Повний вихідний код файлу src/store/useStore.js

```
import { create } from 'zustand';
import { persist, devtools } from 'zustand/middleware';
import { DEFAULT_TASKS, randomScore } from '../lib/constants';

// ── helpers ──────────────────────────────────────────

function now() {
  return new Date().toLocaleTimeString('uk-UA', {
    hour: '2-digit',
    minute: '2-digit',
  });
}

// ── tasks slice ──────────────────────────────────────

const createTasksSlice = (set, get) => ({
  tasks:  DEFAULT_TASKS,
  nextId: DEFAULT_TASKS.length + 1,

  addTask: (fields) => {
    const task = {
      ...fields,
      id:    get().nextId,
      score: randomScore(),
      status: 'todo',
    };
    set(s => ({ tasks: [...s.tasks, task], nextId: s.nextId + 1 }));
    return task;
  },

  quickAddTask: (status, title, priority = 'med') => {
    const task = {
      id:       get().nextId,
      title,
      desc:     '',
      priority,
      cat:      'Backend',
      due:      '',
      status,
      score:    randomScore(),
    };
    set(s => ({ tasks: [...s.tasks, task], nextId: s.nextId + 1 }));
  },
```

```
    filteredTasks: () => {
      const { tasks, filterPriority } = get();
      return filterPriority === 'all'
        ? tasks
        : tasks.filter(t => t.priority === filterPriority);
    },

    countByPriority: (p) =>
      get().tasks.filter(t => t.priority === p).length,
  });

  // ── ui slice ──────────────────────────────────────────

  const createUISlice = (set) => ({
    view:           'board',
    filterPriority: 'all',
    modalOpen:      false,

    setView:           (view)           => set({ view }),
    setFilterPriority: (filterPriority) => set({ filterPriority }),
    openModal:  () => set({ modalOpen: true }),
    closeModal: () => set({ modalOpen: false }),
  });

  // ── ai slice ──────────────────────────────────────────

  const createAISlice = (set) => ({
    messages: [
      {
        role: 'ai',
        text: 'Вітаю! Groq підключено. Llama 3.3 70B готовий _ '
            + 'запитайте мене про ваші завдання!',
        ts: now(),
      },
    ],
    aiLoading: false,

    addMessage:   (msg) =>
      set(s => ({ messages: [...s.messages, { ...msg, ts: now() }] })),
    setAILoading: (v)   => set({ aiLoading: v }),
  });

  // ── combined store ────────────────────────────────────

  export const useStore = create(
    devtools(
      persist(
        (set, get) => ({
          ...createTasksSlice(set, get),
```

```
        ...createUISlice(set),
        ...createAISlice(set),
      }),
      {
        name: 'taskai-store',
        partialize: (s) => ({
          tasks:  s.tasks,
          nextId: s.nextId,
        }),
      }
    ),
    { name: 'TaskAI Store' }
  )
);
```

Додаток 2

Повний вихідний код файлів src/lib/groq.js та src/hooks/useAI.js

```
// ── src/lib/groq.js ──────────────────────────────────────────

const                          API_KEY                          =
'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';
const                          API_URL                          =
'https://api.groq.com/openai/v1/chat/completions';
const MODEL   = 'llama-3.3-70b-versatile';

/**
 * Відправляє запит до Groq API та повертає текст відповіді
моделі.
 * @param {string} system   — системний промпт (роль та
контекст моделі)
 * @param {string} user     — повідомлення користувача
 * @returns {Promise<string>}
 */
export async function callGroq(system, user) {
  const response = await fetch(API_URL, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${API_KEY}`,
    },
    body: JSON.stringify({
      model:      MODEL,
      max_tokens: 1024,
      messages: [
        { role: 'system', content: system },
        { role: 'user',   content: user   },
      ],
    }),
  });

  const data = await response.json();

  if (data.error) {
    throw new Error(data.error.message);
  }

  return data.choices?.[0]?.message?.content ?? '—';
}

// ── src/hooks/useAI.js ──────────────────────────────────────

import { useStore }            from '../store/useStore';
import { callGroq }            from '../lib/groq';
import { tasksSummaryForAI,
         tasksListForAI }      from '../lib/constants';

/**
 * Хук, що інкапсулює логіку взаємодії з Groq API.
```

```
 * Надає методи ask() та optimize() для компонентів.
 */
export function useAI() {
  const tasks      = useStore(s => s.tasks);
  const addMessage = useStore(s => s.addMessage);
  const setLoading = useStore(s => s.setAILoading);
  const loading    = useStore(s => s.aiLoading);

  // Системний промпт для режиму чату
  const chatSystemPrompt = (summary) =>
    `Ти AI-асистент планувальника завдань TaskAI. ` +
    `Відповідай коротко (2-4 речення), по-діловому, ` +
    `українською мовою.\n` +
    `Поточні завдання:\n${summary}`;

  // Системний промпт для режиму оптимізації
  const optimizeSystemPrompt =
    `Ти AI-оптимізатор завдань. Відповідай по-діловому, ` +
    `українською мовою. ` +
    `Давай топ-3 рекомендації нумерованим списком.`;

  /**
   * Відправляє довільний запит користувача до моделі.
   * @param {string} text — текст запиту
   */
  async function ask(text) {
    if (loading) return;

    addMessage({ role: 'user', text });
    setLoading(true);

    const summary = tasksSummaryForAI(tasks);

    try {
      const reply = await callGroq(chatSystemPrompt(summary),
text);
      addMessage({ role: 'ai', text: reply });
    } catch (e) {
      addMessage({    role:    'ai',    text:    `Помилка:
${e.message}` });
    } finally {
      setLoading(false);
    }
  }

  /**
   * Запускає AI-аналіз та оптимізацію всього списку завдань.
   */
  async function optimize() {
    if (loading) return;

    addMessage({
      role: 'user',
```

```
          text:   'Запускаю   AI-оптимізацію   всього   списку
завдань...',
        });
        setLoading(true);

        const list = tasksListForAI(tasks);

        try {
          const reply = await callGroq(
            optimizeSystemPrompt,
            `Проаналізуй   ці   завдання   і   дай   топ-3
рекомендації:\n${list}`,
          );
          addMessage({ role: 'ai', text: reply });
        } catch (e) {
          addMessage({    role:    'ai',    text:    `Помилка:
${e.message}` });
        } finally {
          setLoading(false);
        }
      }

      return { ask, optimize, loading };
    }
```

Додаток 3

Повний вихідний код файлу src/styles/global.css

```
/* ── Reset ──────────────────────────────────────────────────────────── */
*, *::before, *::after {
  box-sizing: border-box;
  margin:     0;
  padding:    0;
}

/* ── CSS-змінні: світла тема ──────────────────────────────────────── */
:root {
  /* Фони */
  --bg-primary:   #ffffff;
  --bg-secondary: #f5f5f3;
  --bg-tertiary:  #ededeb;

  /* Текст */
  --text-primary:   #1a1a18;
  --text-secondary: #6b6b67;
  --text-tertiary:  #9b9b97;

  /* Межі */
  --border-light: rgba(0, 0, 0, 0.10);
  --border-mid:   rgba(0, 0, 0, 0.18);

  /* Семантичні кольори */
  --c-info-bg:  #E6F1FB;  --c-info-txt:  #185FA5;
  --c-warn-bg:  #FAEEDA;  --c-warn-txt:  #854F0B;
  --c-ok-bg:    #EAF3DE;  --c-ok-txt:    #3B6D11;
  --c-err-bg:   #FCEBEB;  --c-err-txt:   #A32D2D;

  /* Акцент */
  --c-accent: #7F77DD;

  /* Border-radius */
  --r:  8px;
  --rl: 12px;

  /* Шрифт */
  --font:  -apple-system,  BlinkMacSystemFont,  'Segoe  UI',
sans-serif;
}

/* ── CSS-змінні: темна тема ──────────────────────────────────────── */
@media (prefers-color-scheme: dark) {
  :root {
    --bg-primary:   #1e1e1c;
    --bg-secondary: #252523;
    --bg-tertiary:  #2c2c2a;
```

```
      --text-primary:   #f0f0ee;
      --text-secondary: #a0a09c;
      --text-tertiary:  #6a6a66;

      --border-light: rgba(255, 255, 255, 0.08);
      --border-mid:   rgba(255, 255, 255, 0.14);

      --c-info-bg:  #0C447C;  --c-info-txt:  #B5D4F4;
      --c-warn-bg:  #633806;  --c-warn-txt:  #FAC775;
      --c-ok-bg:    #27500A;  --c-ok-txt:    #C0DD97;
      --c-err-bg:   #791F1F;  --c-err-txt:   #F7C1C1;
    }
  }

  /* ── Базові стилі ──────────────────────────────────────────
*/
  html, body, #root {
    height:   100%;
    overflow: hidden;
  }

  body {
    font-family: var(--font);
    background:  var(--bg-tertiary);
    color:       var(--text-primary);
    font-size:   14px;
    line-height: 1.5;
    -webkit-font-smoothing:  antialiased;
    -moz-osx-font-smoothing: grayscale;
  }

  /* ── Скролбар ──────────────────────────────────────────────
*/
  ::-webkit-scrollbar {
    width: 5px;
  }
  ::-webkit-scrollbar-track {
    background: transparent;
  }
  ::-webkit-scrollbar-thumb {
    background:    var(--border-mid);
    border-radius: 3px;
  }
  ::-webkit-scrollbar-thumb:hover {
    background: var(--text-tertiary);
  }

  /* ── Утиліти ───────────────────────────────────────────────
*/
  button {
    font-family: var(--font);
    cursor:      pointer;
  }
```

```
input, textarea, select {
  font-family: var(--font);
}

a {
  color:           var(--c-accent);
  text-decoration: none;
}
a:hover {
  text-decoration: underline;
}
```