Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістра

за спеціальності 121 – Інженерія програмного забезпечення

На тему: Дослідження методів автоматизації менеджменту робочого часу та розробка персонального планувальника задач

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-24м
_____/Д.С.Сидоренко/

Керівник
кваліфікаційної роботи _____ /А.М.Стрюк/

Завідувач кафедри _____ /А.М.Стрюк/

Кривий Ріг
2025

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: магістр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А.М.Стрюк

«__» _____ 20__р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-24м Сидоренко Денису Юрійовичу

1. Тема: Дослідження методів автоматизації менеджменту робочого часу та розробка персонального планувальника задач. Затверджено наказом по КНУ №__ від «_» _____ 2025р.
2. Термін подання студентом закінченої роботи : «_» _____ 2025р.
3. Вихідні дані по роботі: У межах дослідження була спроєктована та змодельована система управління задачами, що включає модулі керування завданнями, інтерфейсного відображення, аналітики, нагадувань та централізованого сховища даних. Для опису логіки функціонування застосунку використовувалися UML-діаграми різних типів. Діаграма варіантів використання відобразила ключові сценарії взаємодії користувача із системою, зокрема створення та редагування завдань, перегляд у календарній формі та отримання аналітики.
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): Пояснювальна записка містить опис мети та актуальності дослідження, аналіз застосування методів машинного навчання, моделювання процесів управління проєктами, розроблення архітектури програмного рішення та узагальнення результатів експериментів.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання
1	Огляд літератури та сучасних рішень у сфері менеджменту робочого часу, планувальників задач та UI/UX	01.01.2025 – 20.01.2025
2	Аналіз існуючих методів автоматизації управління задачами, календарного планування та фокус-режимів	21.01.2025 – 10.02.2025
3	Порівняльний аналіз аналогів (Trello, Todoist, Notion, Google Tasks)	11.02.2025 – 28.02.2025
4	Формулювання мети, завдань, актуальності й об’єкта дослідження	01.03.2025 – 10.03.2025
5	Підготовка та оформлення матеріалів першого розділу	11.03.2025 – 25.03.2025
6	Розробка UML-діаграм: Use Case, діаграми класів, активностей; розробка концептуальної архітектури (MVC)	26.03.2025 – 20.04.2025
7	Проєктування структури даних, логіки модулів: задачі, аналітика, фокус-режим, календар	21.04.2025 – 20.05.2025
8	Оформлення матеріалів другого розділу	21.05.2025 – 31.05.2025
9	Розробка інтерфейсу (UI/UX), створення прототипу в Figma та дизайн-системи	01.06.2025 – 25.06.2025
10	Реалізація програмних модулів: задачі, календар, аналітика, режим фокусування	26.06.2025 – 31.08.2025
11	Тестування, відлагодження, оптимізація роботи програмного комплексу	01.09.2025 – 20.09.2025
12	Дослідження ефективності роботи системи, підготовка висновків та узагальнення результатів	21.09.2025 – 10.10.2025
13	Розробка економічного обґрунтування	11.10.2025 – 31.10.2025

№	Найменування етапів кваліфікаційної роботи	Термін виконання
14	Оформлення матеріалів другого, третього й четвертого розділів	01.11.2025 – 20.11.2025
15	Остаточне редагування та оформлення пояснювальної записки	21.11.2025 – 30.11.2025

Дата видачі завдання: «____» _____ 20___ р.
Студент _____ / Д. Ю. Сидоренко /
Керівник роботи _____ / А. М. Стрюк /

УМОВНІ ПОЗНАЧЕННЯ ТА СКОРОЧЕННЯ

1. UML (Unified Modeling Language) – уніфікована мова моделювання, яка використовується для графічного опису структури та логіки роботи системи: діаграми класів, прецедентів, активностей тощо.
2. MVC (Model–View–Controller) – архітектурний підхід, що розділяє додаток на три підсистеми: Model – робота з даними задач, статистикою; View – інтерфейс користувача;Controller – логіка обробки дій користувача.
3. Task Storage (Сховище задач) – програмний модуль або база даних (LocalStorage, IndexedDB, MongoDB), що зберігає інформацію про завдання, категорії, пріоритети та статуси.
4. Analytics Module (Модуль аналітики) – компонент системи, який обробляє дані про виконані й невиконані задачі, формує статистику продуктивності, графіки та коефіцієнти ефективності.
5. Reminder System (Система нагадувань) – модуль, що відстежує дедлайни й генерує сповіщення щодо майбутніх або прострочених задач.

РЕФЕРАТ

АВТОМАТИЗАЦІЯ МЕНЕДЖМЕНТУ РОБОЧОГО ЧАСУ, АВТОМАТИЧНЕ ПЕРЕНЕСЕННЯ НЕВИКОНАНИХ ЗАДАЧ, РОЗШИРЕНИЙ РЕЖИМ ФОКУСУВАННЯ, АНАЛІТИКА ПРОДУКТИВНОСТІ, ПРОЄКТУВАННЯ ПЕРСОНАЛЬНОГО ПЛАНУВАЛЬНИКА ЗАВДАНЬ.

Пояснювальна записка: 75 с., 4 таблиці, 7 схем, 14 джерела**.**

У сучасних умовах цифровізації та швидкого темпу життя проблема ефективного управління робочим часом стає однією з найактуальніших. Багатозадачність, велика кількість інформаційних потоків та постійні перемикання між активностями призводять до зниження продуктивності та хронічного відчуття нестачі часу. Тому розробка інструментів, що допомагають оптимізувати планування, автоматизувати рутинні процеси та підтримувати фокус на пріоритетних завданнях, набуває особливої важливості.У рамках даної кваліфікаційної роботи досліджуються сучасні методи автоматизації планування робочого часу та створюється персональний програмний планувальник завдань, здатний підвищити ефективність організації індивідуальної діяльності. Особлива увага приділяється впровадженню таких механізмів, як автоматичне перенесення невиконаних задач, розширений режим фокусування, система аналітики продуктивності та інструменти моніторингу виконання.Метою роботи є розробка програмного модуля, що забезпечує автоматизоване управління задачами та дозволяє користувачу ефективно організовувати свій робочий час.Дослідження включає - аналіз методів автоматизації менеджменту часу,вивчення підходів до оптимізації планування в умовах багатозадачності,проєктування логічної та програмної архітектури планувальника, розробку та реалізацію функціоналу (управління задачами, автоперенесення.

ABSTRACT

AUTOMATION OF WORKING TIME MANAGEMENT, AUTOMATIC TRANSFER OF UNFULFILLED TASKS, ADVANCED FOCUS MODE, PRODUCTIVITY ANALYTICS, DESIGN OF A PERSONAL TASK PLANNER

Thesis in 75 p,4 tables,7 figure, 14 sources.

In today's digitalization and fast pace of life, the problem of effective work time management is becoming one of the most urgent. Multitasking, a large number of information flows and constant switching between activities lead to a decrease in productivity and a chronic feeling of lack of time. Therefore, the development of tools that help optimize planning, automate routine processes and maintain focus on priority tasks is of particular importance. Within the framework of this qualification work, modern methods of automating work time planning are studied and a personal software task planner is created that can increase the efficiency of organizing individual activities. Special attention is paid to the implementation of such mechanisms as automatic transfer of unfulfilled tasks, an extended focus mode, a performance analytics system and performance monitoring tools. The purpose of the work is to develop a software module that provides automated task management and allows the user to effectively organize their working time.

The study includes - analysis of methods for automating time management,study of approaches to optimizing planning in multitasking, design of the logical and software architecture of the scheduler,development and implementation of functionality (task management, auto-transfer.

ЗМІСТ

ВСТУП

Швидкий ритм життя та великий потік інформації ускладнюють управління часом на роботі. Робота в режимі багатозадачності, постійна зміна діяльності та велика кількість справ призводять до зниження продуктивності, стресу та втрати уваги. Тому потрібні цифрові інструменти, які можуть автоматизувати планування, організовувати завдання та допомагати підтримувати продуктивність.

Зараз важливо бути ефективним як особисто, так і в команді. Планувальник завдань з автоматичним перенесенням невиконаних справ, режимом фокусування, аналізом продуктивності та можливістю налаштування допоможе покращити робочий процес, зменшити навантаження на мозок та краще організувати час. Це особливо важливо для тих, хто працює в умовах, що швидко змінюються, та одночасно виконує багато проєктів.

Мета цієї роботи - вивчити способи автоматизації управління робочим часом і створити планувальник завдань, який робить індивідуальну роботу більш ефективною. Щоб досягти цієї мети, потрібно: проаналізувати наявні системи та методи управління часом, визначити основні потрібні функції, розробити структуру та програмне забезпечення планувальника, а також оцінити його роботу на практиці.

У планувальнику будуть такі функції: автоматичне перенесення невиконаних завдань, режим фокусування з таймером Pomodoro, аналіз продуктивності та можливість налаштування під потреби користувача. В результаті вийде програма, яка допоможе ефективно планувати час, контролювати виконання завдань та підвищувати продуктивність.

1. СУЧАСНИЙ СТАН І АКТУАЛЬНІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1 Формулювання актуальності і завдань роботи

В сучасному світі ефективний управління часом перетворюється на єдиний індикатор успішної професійної діяльності та особистісного розвитку. В умовах зростаючих обсягів інформації, збільшення кількості завдань та швидкої швидкості життя виникає гостра потреба в інструментах, що сприятиме оптимізації планування та контролю часу. Паперові списи завдань та ручне ведення календарів класично застосовуваних методів організації роботи часто виявляються недостатньо ефективними для забезпечення продуктивності при багатозадачності.

Рисунок 1.1 – Тайм менеджмент

Автоматизація процесів менеджерського часу забезпечує зменшення навантаження на людину, зростання точності реалізації планів, уникнення забування критично важливих завдань та оперативне реагування на зміни пріоритетності. Нове цифрове рішення для тайм-менеджменту сприяє

підвищенню ефективності як окремих працівників, так і всієї команди, що є особливо актуальним в умовах конкурентного середовища.

Використання розробки особистого планувальника завдань, спроектованого під індивідуальні потреби користувача, дозволяє не лише застосовувати сучасні методи організації часу, але й передбачати особливості процесу роботи, пріоритети та стиль роботи окремої людини. Підхід такий робить систему більш гнучкою та ефективною в порівнянні з універсальними рішеннями.

У сучасному світі інформації, обсяги щоденних даних ростуть дуже швидко. Фахівці з різних галузей мусять розв'язувати багато завдань одразу, перемикаючись між чатами, поштою та іншими справами. Ця багатозадачність може бути хаотичною, адже важко вибудувати логічну послідовність дій.

Постійні перемикання викликають когнітивне перевантаження, коли мозок не встигає зосередитися після кожного відволікання. Дослідження показують, що після перемикання людині потрібно 20-30 хвилин, аби відновити продуктивність. Отже, час іде не на роботу, а на відновлення уваги.

Відсутність планування призводить до:

- втрати пріоритетів;
- накопичення протермінованих завдань;
- неможливості стежити за прогресом;
- зниження продуктивності та якості роботи.

В умовах зростає потреба в програмах, що впорядковують інформацію та дають можливість управляти задачами. Персональний планувальник допомагає структурувати діяльність, навіть великі обсяги справ, та розумно розподіляти навантаження, зменшуючи вплив багатозадачності. Зараз помітна тенденція: успішні IT-системи роблять акцент на персоналізації. Користувачам потрібні рішення, які враховують їхні потреби, стиль, завантаженість та особисті вподобання. Тому від стандартів переходять до гнучких систем.

- Персоналізація в програмах означає:
- налаштування інтерфейсу;

- вибір зручних методів планування;
- створення індивідуальних правил;
- адаптацію нагадувань та пріоритетів;
- підлаштування до ритму роботи.

Персональний планувальник – це відповідь на цей тренд. Він не просто зберігає інформацію, а й враховує мислення та звички. Наприклад, користувач може налаштувати режими концентрації, типи завдань, методи сортування або інтервали таймера під себе.

Такий підхід покращує результат, тому що людина працює в комфортних умовах, а система підлаштовується під неї. Персоналізація допомагає звикнути до планування, що поліпшує дисципліну та продуктивність надовго.

Персоналізований планувальник стає не просто інструментом, а помічником, який ураховує особливості кожного користувача.

Отже, дослідження методів автоматизації управління робочим часом і створення персоналізованого планувальника задач є надзвичайно актуальними завданнями, що відповідають сучасним вимогам до підвищення продуктивності та якості роботи в різних сферах діяльності.

1.2 Науковий апарат (мета, об'єкт, предмет, задачі та методи дослідження)

Мета дослідження:

Вивчити сучасні методи автоматизації управління робочим часом та розробити індивідуальний планувальник завдань, спрямований на підвищення ефективності організації особистого робочого процесу в умовах багатозадачності та інформаційного перевантаження.

Об’єкт дослідження

Процес управління робочим часом та організація задач в умовах сучасної цифровізації та підвищеної багатозадачності.

Предмет дослідження

Методи автоматизації управління робочим часом, а також програмні засоби для планування, моніторингу та оптимізації виконання завдань.

Завдання дослідження

Провести аналіз сучасних методів і засобів автоматизації процесів управління робочим часом. Визначити ключові вимоги та принципи побудови персонального планувальника завдань. Розробити архітектурне рішення та описати функціональну модель особистого планувальника.

Створити прототип програмного забезпечення для планування завдань.

Реалізувати основні функції системи:

- управління задачами;
- автоматичне перенесення незавершених завдань;
- режим фокусування;
- аналітику продуктивності.
- Провести порівняння розробленого рішення з існуючими аналогами.

Здійснити емпіричне тестування програмного продукту та оцінити його ефективність у реальних умовах використання.

Методи дослідження

Аналіз літератури та огляд доступних рішень - дослідження теоретичних підходів, інструментів та програм для управління часом. Порівняльний аналіз методів - визначення переваг та недоліків різних підходів до автоматизації планування. Дослідження проблем багатозадачності - аналіз когнітивного навантаження, інформаційного перевантаження та необхідності автоматизації.

Моделювання процесу планування робочого часу - побудова логічних схем, архітектури та взаємодії компонентів планувальника. Обґрунтування вибору технологій для створення програмного забезпечення (JavaScript, React, Node.js, MongoDB).

Розробка та програмна реалізація прототипу системи.

Емпіричне тестування роботи планувальника, оцінювання продуктивності та зручності використання.

Організація часу є ключовою організуючою структурою, що забезпечує працездатність на індивідуальному та корпоративному рівнях. Зі збільшенням інформаційного навантаження, з боку кількості завдань постає потреба в інструментах, які оптимізують розподіл часу, слідкують виконанням поставлених завдань і підвищують самої продуктивності. Предметом діяльності є аналіз процесу планування, організації та контролю виконання завдань, що відбуваються у встановленому кінцевому терміні.

У головних елементах очікується постановка цілей, формування списку завдань, пріоритезація, встановлення термінів та персональна оцінка результатів роботи з подальшим зворотнім зв`язком. Разом із розвитком цифрових технологій виконання цих задач стало автоматизованим завдяки різним програмам: планувальникам, трекерам завдань, календарям, системам управління проєктами й ін. Ринок сьогодні розлучений великою кількістю програм з тайм-менеджменту: Todoist, Trello, Notion, Microsoft To Do, Google Calendar тощо. Вони несли аж ніяк не обмежений функціонал: вказання завдань, налаштування нагадувань, ведення щоденника, командна співпраця тощо. Однак більшість або надто складні для того, щоб використовувати їх особисто, або у безкоштовних версіях.

Отже, дедалі раніше йшло розроблення персонального планувальника завдань, рівня пристосованого до особистих запитів, простого, зручного, функціонального, здатного до автоматизації рутинних завдань управління.

Таблиця 1 – Порівняння існуючих аналогів

Назва сервісу	Основні функції	Платформа	Безкоштовна версія	Недоліки
Todoist	Створення задач, проєкти, пріоритети, повторювані події, інтеграції	Web, Android, iOS	Так	Обмеження у безкоштовній версії, відсутність тайм-трекінгу
Trello	Канбан-дошки, списки задач, теги, дедлайни, командна робота	Web, Android, iOS	Так	Ненадійна для великої кількості задач, відсутність чіткого таймлайну
Notion	Блокноти, бази даних, списки задач, календарі, персоналізація	Web, Android, iOS	Так	Стартова складність, потреба часу на налаштування
Microsoft To Do	Просте створення задач, списки, нагадування, інтеграція з Outlook	Web, Android, iOS	Так	Мінімалістичний функціонал, не підходить для складних проєктів
Google Calendar	Календар, події, нагадування, синхронізація з Gmail	Web, Android, iOS	Так	Відсутність гнучкої роботи з задачами, слабка підтримка пріоритетів
TickTick	Задачі, нагадування, помічник Pomodoro, календар, теги	Web, Android, iOS	Так (з обмеженнями)	Деякі функції тільки в платній версії (наприклад, трекінг звичок)

1. Загальні вимоги

Система повинна бути простою у використанні та інтуїтивно зрозумілою для користувача.

Підтримка як стаціонарної (браузерної) версії, так і з адаптацією під мобільні пристрої.

Дані користувача зберігаються локально або в хмарному сховищі (опціонально).

Має працювати офлайн (без підключення до Інтернету)

Таблиця 2 – Функціональні вимоги

Категорія	Опис функціоналу
Управління задачами	- Створення, редагування, видалення задач- Встановлення дедлайнів- Пріоритетність
Категоризація	- Групування задач за проєктами або тегами- Можливість фільтрації та пошуку
Повторювані задачі	- Автоматичне створення щоденних/щотижневих/щомісячних задач
Нагадування та сповіщення	- Встановлення нагадувань (локальні повідомлення або push)
Календар та візуалізація	- Інтегрований календар для перегляду задач у часовому вимірі
Статистика та аналітика	- Відображення завершених задач- Графік продуктивності
Імпорт/експорт даних	- Збереження копії задач у JSON/CSV- Можливість резервного копіювання

Нефункціональні характеристики:

Продуктивність: додаток має працювати без будь-яких затримок, навіть якщо вводиться величезна кількість завдань.

Безпека: забезпечується конфіденційність даних користувача (локальне шифрування, якщо доступна синхронізація).

Зручність інтерфейсу: сучасний адаптивний UI (скажімо, фреймворки Tailwind CSS або Material UI).

Масштабованість: буде додатково розширено за допомогою додаткових функцій (наприклад: синхронізація між пристроями, командний режим)Технології для реалізації (рекомендовані)

Front-End: HTML, CSS (Tailwind), JavaScript (або React)

Зберігання даних: LocalStorage / IndexedDB (опціонально Firebase, Supabase)

Розгортання: GitHub Pages, Vercel або Netlify

Інші бібліотеки: Day.js або date-fns (робота з датами), Chart.js (графіки)

1.3 Ааналіз досліджень з обраної теми

Інформаційне суспільство сьогодні утоваровує кваліфіковану своєчасну організацію робочого часу своєчасно більше ніж будь-коли раніше. Рідкість рівнів інформації, багатозадачність і життя високої інтенсивності закладають у людину можливість кваліфіковано організовувати та регулювати діяльність. Залежно від цього протягом останніх років засіб свій візьмеш за своєчасне автоматизування управління часом, зокрема механізми цифрової організації праці.

1. Теоретичні підстави управління часом

В роботах таких авторів, як Стівен Кові («7 звичок надзвичайно ефективних людей») та Брайан Трейсі («Зроби це зараз»), підкреслюється важливість пріоритизації завдань, розподілу часу за принципом важливості та актуальності, а також необхідності самодисципліни. Їх методи стали основою для розробки ряду цифрових інструментів, що автоматизують планування та управління завданнями.

2. Методи та алгоритми автоматизації

В наукових статтях і технічних дослідження часто розглядаються алгоритми, які лежать в основі цифрових планувальників:

Метод GTD (Getting Things Done) Девіда Аллена - здійснюється у вигляді чеклістів, спонукань, списків робіт;

Метод Канбан - здійснюється для візуалізації процесу виконання робіт (дошки Trello, Jira);

Метод Помідора (Pomodoro Technique) - використовується у багатьох тайм-менеджмент застосунках як метод підвищення ефективності роботи.

З технічної точки зору дослідження мають за мету оптимізувати інтерфейси UI/UX, синхронізувати з календарями (Google Calendar, Outlook), інтегрувати штучний інтелект на підготовку прогнозів ефективності праці та виконати автоматичний розподіл робіт.

3. Огляд наявних систем та додатків

Популярні програмні рішення, такі як Todoist, Notion, Trello, Asana, Microsoft To Do широко досліджуються як приклади реалізації успішного впровадження автоматизованих підходів до планування. Кожен із сервісів має свої особливості:

Todoist - легкість використання, інтеграція з електронною поштою та календарями;

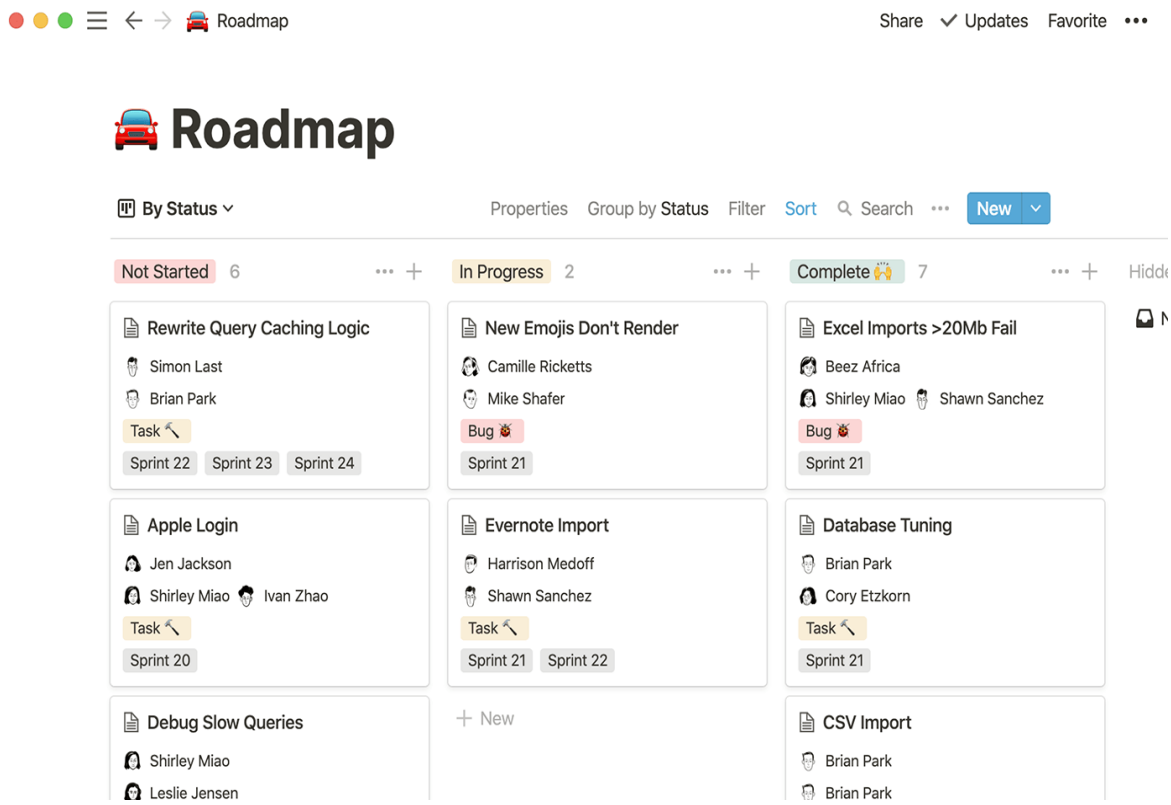

Рисунок 1.2 – Інтерфейс програми todoist

Notion - гнучкість структурування інформації;

Trello - візуальна організація задач;

Asana - призначений для командної роботи;

Microsoft To Do - інтеграція з екосистемою Microsoft.

В більшості випадків користувачі розчаровані проблемами інтерфейсної перевантаженості, складності встановлення та обмеженої персоналізації.

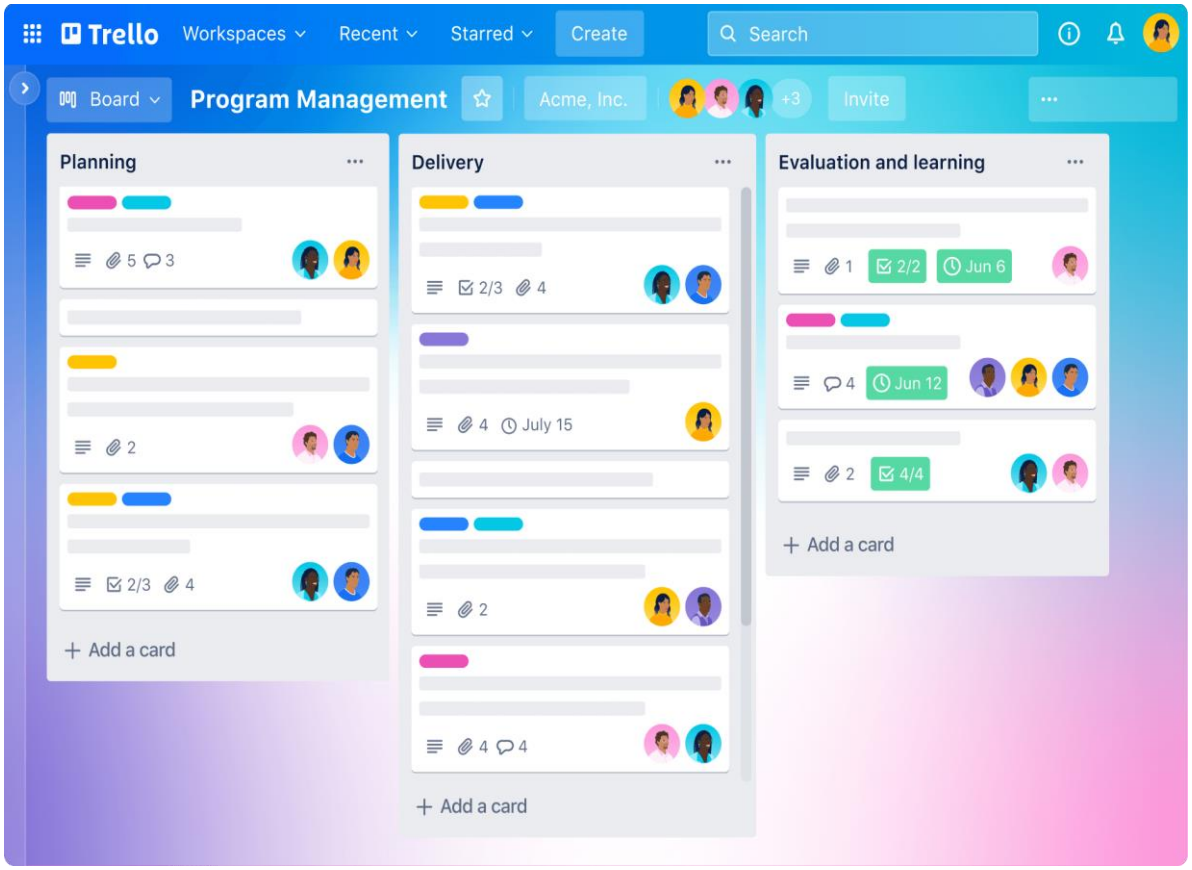

Рисунок 1.3 – Інтерфейс програми Trello

У Trello візуальна Kanban-дошка з картками, колонками і переміщенням завдань між ними є доречною для відображення етапів роботи та відстеження поточного стану задач.

Todoist списки, підзадачі, пріорітети та терміни виконання оптимальні для організації персональних завдань, щоденних справ або процесів з чіткою послідовністю дій.

4. Досвід роботи над особистим планувальником

Почерпнуте в аналізі досліджень кілька напрямів можна виділити для створення особистого інструмента планування:

Виконання адаптації під індивідуальні робочі стани користувача;

Легкоінтерактивний, інтуїтивно зрозумілий інтерфейс;

Інтеграція з іншими послугами та мобільними пристроєм;

Ужиток аналітики для оцінки ефективності виконання завдань;

Працює офлайн-режим та локальне збереження даних.

1.5 Визначення вимог

У залежності від досліджень, аналіз користувацьких потреб та нових технологічних рішень визначені вимоги до розробки персонального планувальника задач. Всі вимоги поділені на функціональні, нефункціональні та системні.

1. Функціональні вимоги:

1.1. Реєстрація та авторизація користувача (з включенням використання сторонніх сервісів - Google, Facebook тощо).

1.2. Створення, редагування, видалення задач з можливістю встановлення дедлайнів, пріоритетів, категорій.

1.3. Візуалізація задач у вигляді списку, календаря або Kanban-дошки.

1.4. Дослідження та фільтрація завдань за різними параметрами (дата, статус, тег тощо).

1.5. Підсічення щодо наближення термінів виконання завдань (з допомогою push-повідомлення або email).

1.6. Автоматичне повторення завдань (щодня, щотижня тощо).

1.7. Синхронізація з зовнішніми календарями (Google Calendar, Outlook).

1.8. Статистика ефективності та аналітика (час, витрачений на завдання, % виконаних завдань тощо).

2. Нефункціональні вимоги:

2.1. Інтерфейс повинен бути адаптивним для використання на ПК, планшетах та смартфонах.

2.2. Аплікація повинна мати інтуїтивно зрозумілий, мінімалістичний дизайн.

2.3. assez високий рівень швидкодії при обробці великої кількості завдань.

2.4. Підтримка режиму офлайн зі збереженням даних локально.

2.5. Забезпечення конфіденційності даних користувача.

3. Системні вимоги:

3.1. Веб-додаток реалізується на основі HTML, CSS, JavaScript (з урахуванням застосування фреймворків при необхідності - React/Vue).

3.2. Сховище даних - місцево (LocalStorage) або в базі даних (наприклад, Firebase чи SQLite в мобільній версії).

3.3. Backend-сервіс (при необхідності) - Node.js або інші сумісні технології.

3.4. Кросбраузерна сумісність: підтримка сучасних версій Chrome, Firefox, Edge, Safari.

Моделювання, проектування та прототипування розробки

На основі сформульованих функціональних і нефункціональних вимог було здійснено етапи моделювання, проєктування архітектури та створення початкового прототипу персонального планувальника задач. Метою цього етапу стало забезпечення структурованого бачення майбутньої системи та перевірка ключових функцій перед початком повноцінної реалізації.

Для моделювання логіки системи застосовувався набір UML-діаграм, кожна з яких відігравала певну роль у структурі проєкту. Діаграма варіантів використання показала, як користувач взаємодіє із застосунком, зокрема, створення та редагування завдань, перегляд календаря, отримання сповіщень і аналіз статистики. Це допомогло визначити важливі точки взаємодії. Діаграма класів детально описала внутрішню структуру з основними об'єктами (Завдання, Користувач, Сповіщення, Статистика) та їх зв'язками, що створило основу для логіки даних. Діаграма активностей відобразила послідовність дій при створенні завдання (введення назви, опису, дати, пріоритету та збереження), що дозволило побачити операцію на рівні бізнес-процесу.

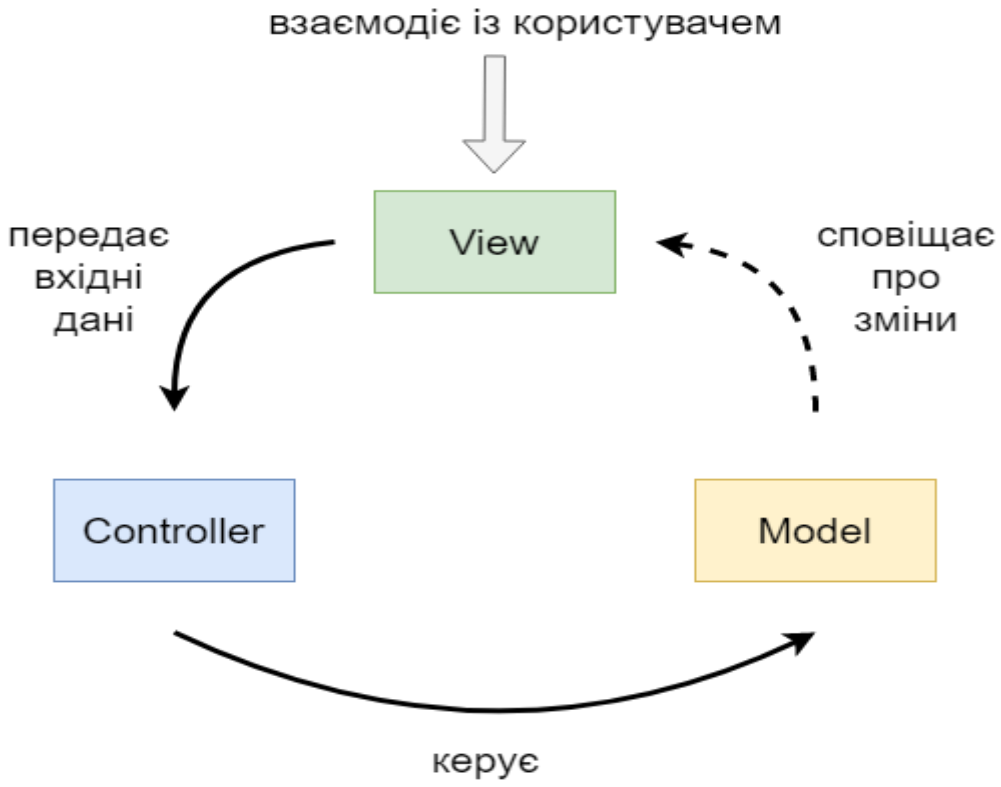

Рисунок 1.4 – Модель MVC

Архітектура застосунку базувалася на моделі MVC, яка розділяє логіку, дані та інтерфейс. Модель містить структури даних завдань, інформацію про користувача та аналітику. Їх реалізація можлива через LocalStorage, IndexedDB (локальна версія) або REST API (серверна частина). Компонент View відображає інтерфейс за допомогою HTML, CSS і JavaScript, формуючи списки завдань, календар, Kanban-дошку та інтерактивні елементи (кнопки, форми, меню).

Контролер керує логікою та подіями: створення завдань, зміна статусу, перемикання між режимами перегляду, фільтрація та сортування. Такий підхід забезпечує гнучкість, спрощує підтримку та дозволяє масштабувати систему.

На етапі прототипування інтерфейсу розробили макети застосунку в Figma або Adobe XD. Прототип містив ключові екрани: головну сторінку зі списком завдань і переходом між днями/тижнями, кнопку додавання завдання (відкриває модальне вікно), панель фільтрів/пошуку та бічне меню для навігації, налаштувань профілю й аналітики. Також створили прототип календарного вигляду з інтерактивними елементами. Це дало змогу оцінити зручність користування, логіку переходів і виявити потрібні зміни.

Прототип тестували серед цільової аудиторії, щоб отримати відгуки про інтерфейс і функціональність. Більшість відзначили інтуїтивність і зручну навігацію, але були і важливі зауваження. Запропонували додати помітну кольорову індикацію пріоритетів і групування завдань за проєктами/категоріями для кращої організації інформації.

Технології реалізації обрали на основі мети та вимог проєкту. Для фронтенду вирішили вживати HTML5 і CSS3 (Flexbox, Grid для адаптивної верстки) та JavaScript ES6+ (для динамічної логіки). У перспективі можливий перехід на React або Vue.js для управління станом і створення складних модулів. Бекенд можна побудувати на Node.js з Express або використати BaaS (Firebase, Supabase). Для збереження даних підійде MongoDB (серверна версія) або LocalStorage/IndexedDB (локальна). Для контролю версій обрали Git і GitHub, а для макетів – Figma, Whimsical або Balsamiq. Розробку вели за Agile-методологією зі спринтами для контролю прогресу.

При проєктуванні інтерфейсу враховувалися принципи UI/UX-дизайну. Ключовою була концепція мінімалізму: акцент на завданнях без зайвих деталей. Інтуїтивність забезпечувалася розташуванням елементів управління, щоб користувач міг швидко виконати дію. Приділили увагу кольоровій доступності, контрастним кольорам для пріоритетів і підтримці темної та світлої теми. Головним було – простота взаємодії: швидке додавання, зручна фільтрація, легка навігація. Це зробило інтерфейс не лише функціональним, а й комфортним.

2 РОЗРОБКА ФУНКЦІОНАЛЬНОЇ СХЕМИ І АЛГОРИТМУ

2.1 Розробка та докладний опис функціональної схеми програми

Для належної роботи персонального планувальника задач розроблено схему програми. Вона показує головні частини системи та їхній зв'язок.

Схема ілюструє, як дані обробляються від початку, коли людина працює з інтерфейсом, і до кінця, коли результати зберігаються у базі даних чи іншому місці.

Головні частини системи:

1.Частина для підтвердження особи.

Вона потрібна для реєстрації, входу в систему та авторизації через інші сервіси (наприклад, Google, Facebook). Вона допомагає зберігати токени доступу та берегти особисті дані.

2.Частина для роботи з задачами.

Тут можна створювати, змінювати, видаляти та показувати задачі.

У задач є такі властивості: назва, опис, термін виконання, важливість, тип, стан виконання, дата створення.

3.Частина для показу інформації.

Вона показує задачі у різних видах: списку, календарі або дошці Kanban. Можна змінювати вигляд даних без втрати інформації чи стану користувача.

4.Частина для пошуку та фільтрування.

Дозволяє шукати та сортувати задачі за різними ознаками: датою, важливістю, типом, станом, тегами.

5.Частина для повідомлень.

Надсилає людині повідомлення про наближення термінів або прострочені задачі.

Можна використовувати push- або email-повідомлення, в залежності від налаштувань.

6.Частина для аналізу.

Відстежує, як людина працює: скільки задач зроблено, скільки часу витрачено, наскільки добре працює за день/тиждень. Результати показуються у вигляді графіків.

7.Місце для збереження даних.

Тут зберігається інформація про людей та їх задачі.

Можна використовувати LocalStorage (у веб-версії) або Firebase/SQLite (у розширеній версії з backend).

8.Backend-сервіс (за потреби).

Він є посередником між клієнтською частиною та базою даних.

Він може бути зроблений на Node.js + Express та надає API-запити CRUD (створення, читання, оновлення, видалення)

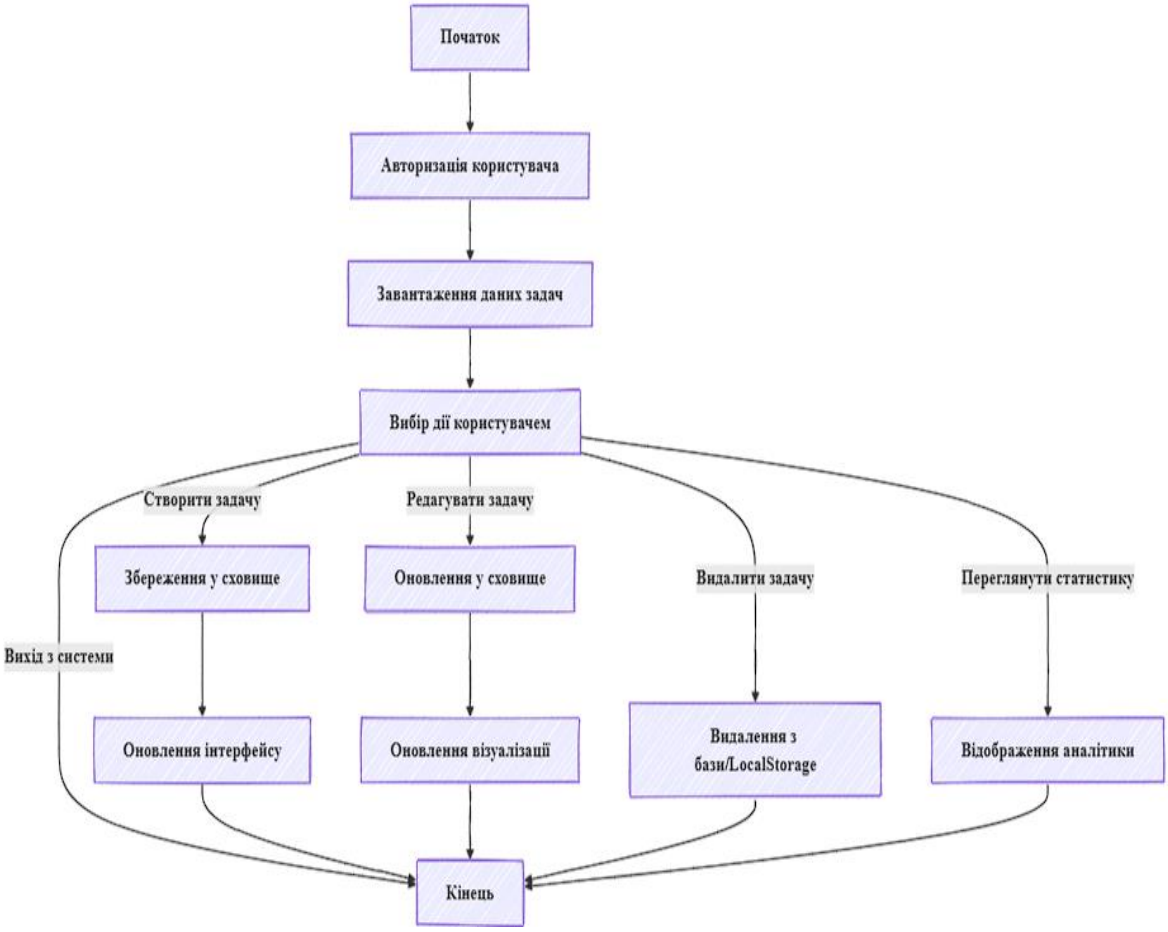

Рисунок 2.1 – Структурна схема програми.

Взаємодія компонентів відбувається так:

1.Користувач працює з інтерфейсом, щоб створити або змінити задачу.

2.Інтерфейс передає запит у модуль управління задачами.

3.Модуль перевіряє дані та зберігає їх.

4.Модуль візуалізації показує оновлення в інтерфейсі.

5.Якщо є дедлайн, модуль сповіщень налаштовує нагадування.

6.Модуль аналітики збирає статистику про виконання задач

2 Контекстна схема системи

Контекстна схема особистого планувальника завдань показує основні об’єкти, з якими взаємодіє система, та як обмінюється інформація. Основна мета схеми – визначити межі системи, вхідні та вихідні потоки даних, а також способи спілкування з користувачем і зовнішніми платформами.

Основні зовнішні об’єкти:

Користувач:

взаємодіє із системою через інтерфейс, створюючи, редагуючи, видаляючи завдання, переглядаючи звіти та налаштовуючи нагадування.

Вхідні дані: логін/пароль або дані Google/Facebook, інформація про завдання.

Вихідні дані: підтвердження входу, список завдань, сигнали, звітність. Платформи перевірки (Google, Facebook). Дають змогу користувачеві ввійти через OAuth.Вхідні дані: запит на перевірку.

Вихідні дані: ключ доступу або відомості про користувача. Служба нагадувань (Електронна пошта / Push). Надсилає користувачеві повідомлення про терміни або пропущені завдання.

Вхідні дані: запит із зазначенням типу та часу сповіщення.

Вихідні дані: надіслане повідомлення або підтвердження відправки. Календарі (наприклад, Google Calendar, Outlook) синхронізують події з планами користувача. Система отримує дані про завдання з кінцевими термінами та надає календарні події або підтвердження синхронізації.

База даних або LocalStorage потрібні для збереження інформації про користувачів, їхні завдання, налаштування та статистику. Дані про створені чи оновлені завдання надходять у систему, а при запуску система надає

дані,результати пошуку та аналітику.

Рисунок 2.2 – Контекстна діаграма

Загальна схема потоків даних (DFD) показує, як обробляється інформація в системі планування задач. На відміну від контекстної діаграми, тут показані як зв'язки із зовнішніми об'єктами, так і внутрішні процеси, наприклад програмні модулі та шляхи передачі даних між ними.

Основні компоненти DFD:

Користувач: головне джерело даних (створення, редагування, видалення задач, перегляд аналітики).

Сервіси автентифікації (наприклад, Google, Facebook): надають дані для входу.

Система сповіщень (Email / Push): отримує запити на надсилання нагадувань. Зовнішні календарі (наприклад, Google Calendar, Outlook): дають можливість синхронізувати задачі.

Внутрішні процеси системи:

Р1 – Авторизація користувача: процес приймає дані для входу або запит на авторизацію через сервіс. Далі передає токен автентифікації внутрішнім модулям для доступу до даних користувача. Р2 – Управління задачами: основний процес системи. Обробляє введення нових задач, зміни та видалення наявних. Також працює з базою даних для збереження інформації. Р3–Фільтрація, пошук і сортування: процес обробляє потреби користувача у пошуку задач за тегами, статусом, пріоритетом або датою.

Р4 – Модуль аналітики: аналізує збережені дані та формує звіти про продуктивність користувача, а саме: кількість зроблених задач, середній час виконання, продуктивність по днях. Р5 – Модуль сповіщень: визначає задачі з дедлайном і створює нагадування для користувача, формуючи запити до системи сповіщень (email або push). Р6 – Синхронізація з календарями: передає інформацію про задачі у зовнішні календарі та отримує оновлення з них

.

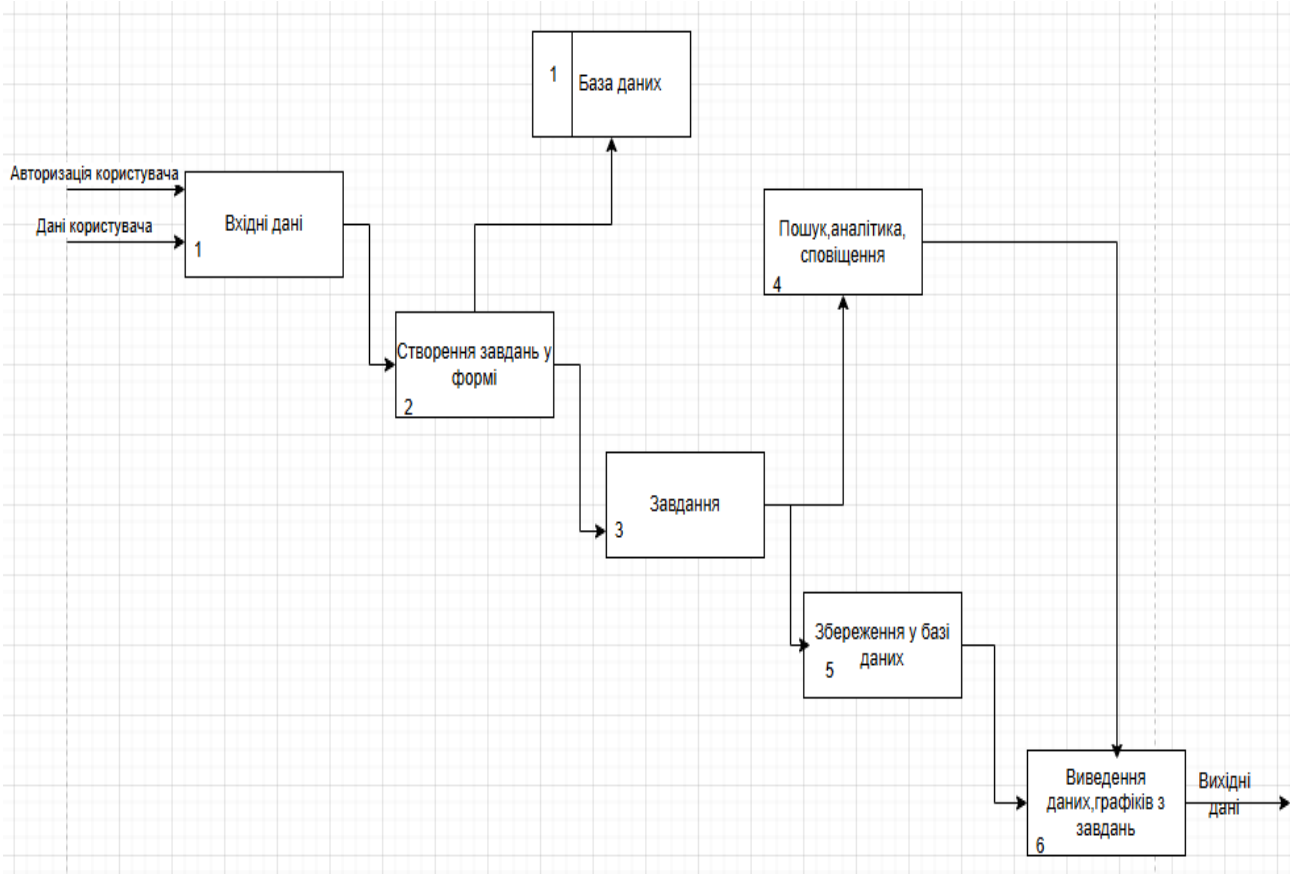

Рисунок 2.3 – Загальна діаграма потоків даних

Діаграма потоків даних (DFD) для системи керування завданнями

Рівень 0: Контекстна схема

Користувач взаємодіє з системою керування завданнями, виконуючи наступні дії:

Система управління завданнями складається з декількох взаємопов'язаних підсистем, кожна з яких має своє призначення. Разом вони забезпечують повний цикл роботи, дозволяючи планувати та контролювати продуктивність.Основна частина системи – модуль управління завданнями. Він дозволяє створювати, змінювати, оновлювати статуси та видаляти завдання. Коли користувач вводить нове завдання, редагує або завершує його, система передає ці дії до сховища даних, де інформація зберігається та оновлюється.

Модуль відображення інтерфейсу надає користувачеві доступ до інформації. Він показує завдання у вигляді списку, дошки Kanban або календаря. Коли користувач відкриває розділ або оновлює сторінку, модуль отримує дані зі

сховища, перетворює їх у візуальний формат і відображає на екрані. Інтерфейс завжди показує актуальний стан системи та спрощує роботу з даними.

Модуль аналітики додає інтелектуальні функції. Він збирає статистику про виконані та невиконані завдання, аналізує продуктивність за різні періоди, визначає динаміку та допомагає оцінити загальну ефективність. На основі цих даних система може показати дні найбільшої активності користувача, кількість виконаних завдань і розрахувати ключові показники. Користувач може надіслати запит на звіт, і система покаже інформацію у зручному вигляді.

Модуль нагадувань аналізує завдання з дедлайнами та контролює терміни виконання. Якщо термін наближається або вже минув, модуль надсилає сповіщення користувачеві, допомагаючи не пропустити важливі події та вчасно завершувати роботу. Всі модулі використовують спільне сховище даних, де зберігається вся інформація про завдання, пріоритети, теги, категорії та історія змін. Сховище забезпечує цілісність і актуальність даних, дозволяючи різним частинам системи швидко отримувати потрібну інформацію.

Отже, робота системи полягає в постійному обміні даними між користувачем, інтерфейсом і внутрішніми модулями. Користувач створює або змінює завдання, модуль управління обробляє запит і оновлює сховище, модуль інтерфейсу показує зміни, модуль аналітики обробляє дані, а модуль нагадувань стежить за термінами. Така структура забезпечує злагоджену та швидку роботу системи, а також дає повний контроль над задачами.

Таблиця 2.1 – Опис основних елементів діаграми DFD

№	Елемент діаграми	Тип	Опис
1	Вхідні дані	Зовнішній об’єкт	Користувач вводить інформацію про завдання: назву, опис, термін, категорію, пріоритет. Це початкова точка процесу.
2	Створення завдань у формі	Процес	Інтерфейс, через який користувач вводить дані про завдання. Тут відбувається перевірка введених значень і підготовка даних до збереження.
3	Завдання	Процес/сутність	Відображає саму сутність «Task» (завдання), яка проходить через усі етапи життєвого циклу - створення, редагування, видалення.
4	Пошук, аналітика, сповіщення	Процес	Модуль, який забезпечує фільтрацію завдань, підрахунок статистики (кількість виконаних, прострочених, активних), а також може формувати нагадування про дедлайни.
5	Збереження у базі даних	Процес	Відповідає за запис нових або змінених завдань у сховище. Виконує операції Create, Update, Delete.
6	Виведення даних, графіків з завдань	Процес	Формує кінцеву візуалізацію: список завдань, календар або графік ефективності. Це кінцева точка взаємодії.
7	База даних	Сховище даних	Місце постійного збереження задач, статусів, тегів і категорій. У вебверсії - LocalStorage або IndexedDB, у мобільній - SQLite.
8	Вихідні дані	Зовнішній об’єкт	Інформація, яку бачить користувач - відфільтрований список завдань, аналітика, графіки або статистика.

Користувач створює або змінює задачу через інтерфейс. Система передає дані в модуль управління задачами, який оновлює сховище. При необхідності, система отримує актуальні дані з локального сховища (LocalStorage/SQLite). На запит користувача модуль відображення формує список задач, календар або дошку Kanban. Якщо активовано модуль аналітики, він обробляє дані та формує статистичні дані. За наявності модуля нагадувань, система надсилає push-повідомлення про наближення термінів виконання задач.

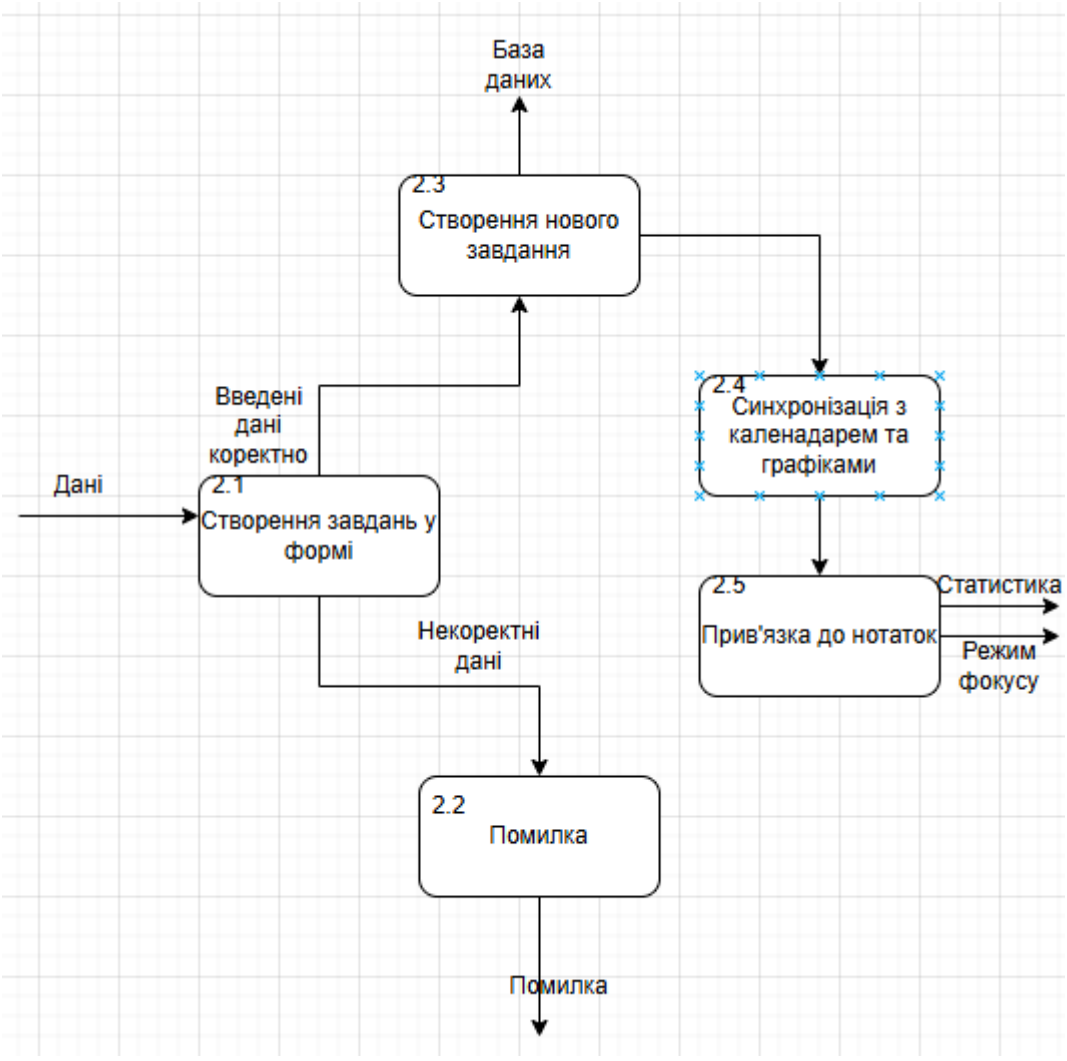

Рисунок 2.4 – Деталізована діаграма потоків даних

Таблиця 2.2 – Основні функціональні модулі

№	Модуль	Призначення
1	Інтерфейс користувача (UI)	Забезпечує введення, редагування, перегляд і видалення завдань. Містить поля введення, кнопки дій, фільтри, панель статистики.
2	Модуль керування задачами	Обробляє створення, редагування, видалення та зміну статусів завдань (“виконано”, “в процесі”, “прострочено”).
3	Модуль збереження даних	Відповідає за запис і читання даних із локального сховища браузера (LocalStorage або IndexedDB).
4	Модуль аналітики та статистики	Обчислює кількість виконаних задач, формує дані для діаграм, визначає продуктивні дні, середню тривалість виконання.
5	Модуль нагадувань (опціонально)	Відстежує дедлайни, формує локальні сповіщення про наближення термінів.
6	Модуль візуалізації	Формує графічне подання списку завдань, діаграм продуктивності, або календарного перегляду.

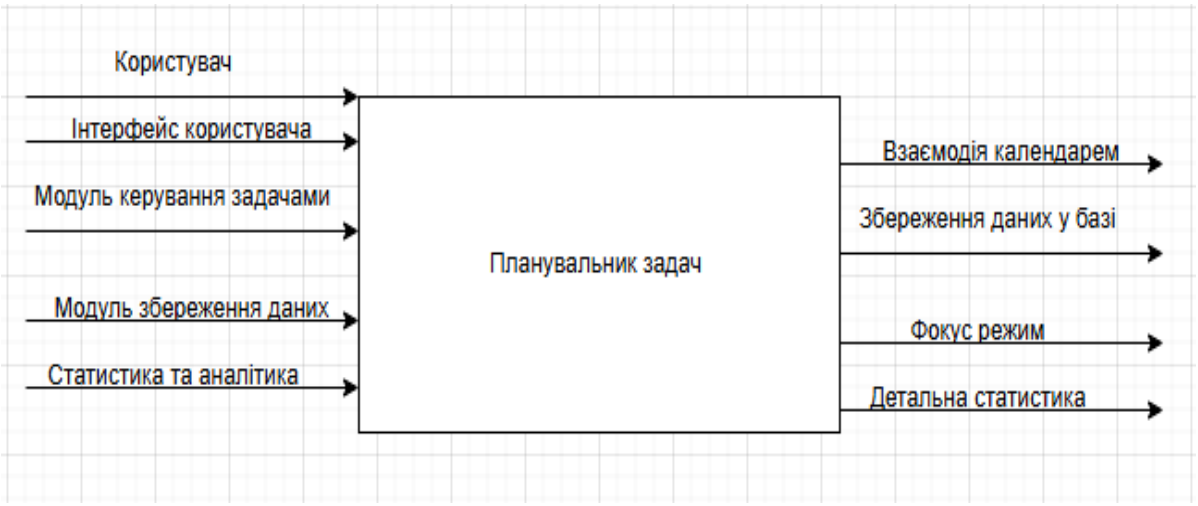

Рисунок 2.5 - Загальна функціональна схема ПЗ

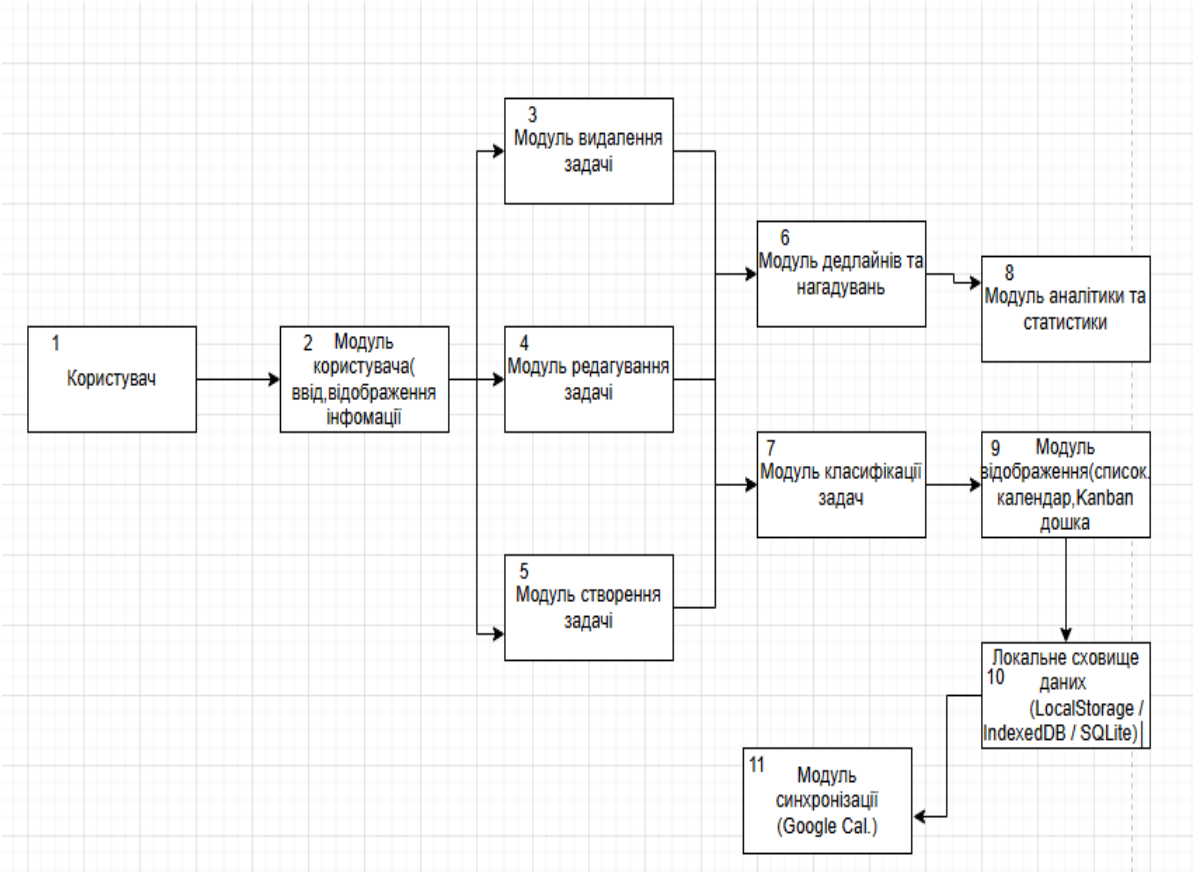

Рисунок 2.6 - Фунціональна схема ПЗ

Функціональна схема показує, як логічно влаштована система та як різні частини персонального планувальника задач працюють разом.

Вона має три рівні:

1.Рівень користувача (де відбувається взаємодія).

2.Рівень обробки даних (де система думає).

3.Рівень зберігання та оновлення даних.

1. Рівень користувача

Блок 1. Користувач

Це той, хто все починає. Він працює з системою через інтерфейс, щоб робити, змінювати, дивитися або прибирати завдання.

Блок 2. Модуль користувача (введення, показ інформації)

Допомагає користувачу спілкуватися з системою. Він бере дані, які вводить користувач (текст, дати, важливість задач), і показує результати роботи

програми (списки, календар, статистику). Тут користувач може робити нові завдання, змінювати старі чи прибирати непотрібні.

2. Логічний рівень обробки даних

Блок 3. Модуль видалення задачі

Прибирає задачу, яку вибрав користувач, з бази даних на комп'ютері. Потім передає оновлену інформацію до модуля класифікації.

Блок 4. Модуль редагування задачі

Дає можливість змінювати, що вже є в задачі (термін, пояснення, важливість). Після зміни дані йдуть до модуля термінів.

Блок 5. Модуль створення задачі

Отримує нові дані від користувача і робить з них об'єкт задачі. Потім відправляє його до модуля класифікації для подальшої роботи.

Блок 6. Модуль дедлайнів та нагадувань

Стежить за тим, коли треба виконати задачу, і повідомляє, коли час добігає кінця. Ця інформація йде до аналітичного модуля.

Блок 7. Модуль класифікації задач

Розкладає задачі по поличках за темами, мітками та важливістю. Потім передає дані до модуля відображення, щоб сформувати зручний список.

Блок 8. Модуль аналітики та статистики

Розглядає дані про те, які задачі зроблено, скільки часу пішло і які були важливі. Робить звіти про те, наскільки добре працює користувач.

Блок 9. Модуль відображення (список, календар, Канбан-дошка)

Відповідає за те, як задачі виглядають на екрані в різних форматах. Показує свіжі дані з урахуванням різних фільтрів.

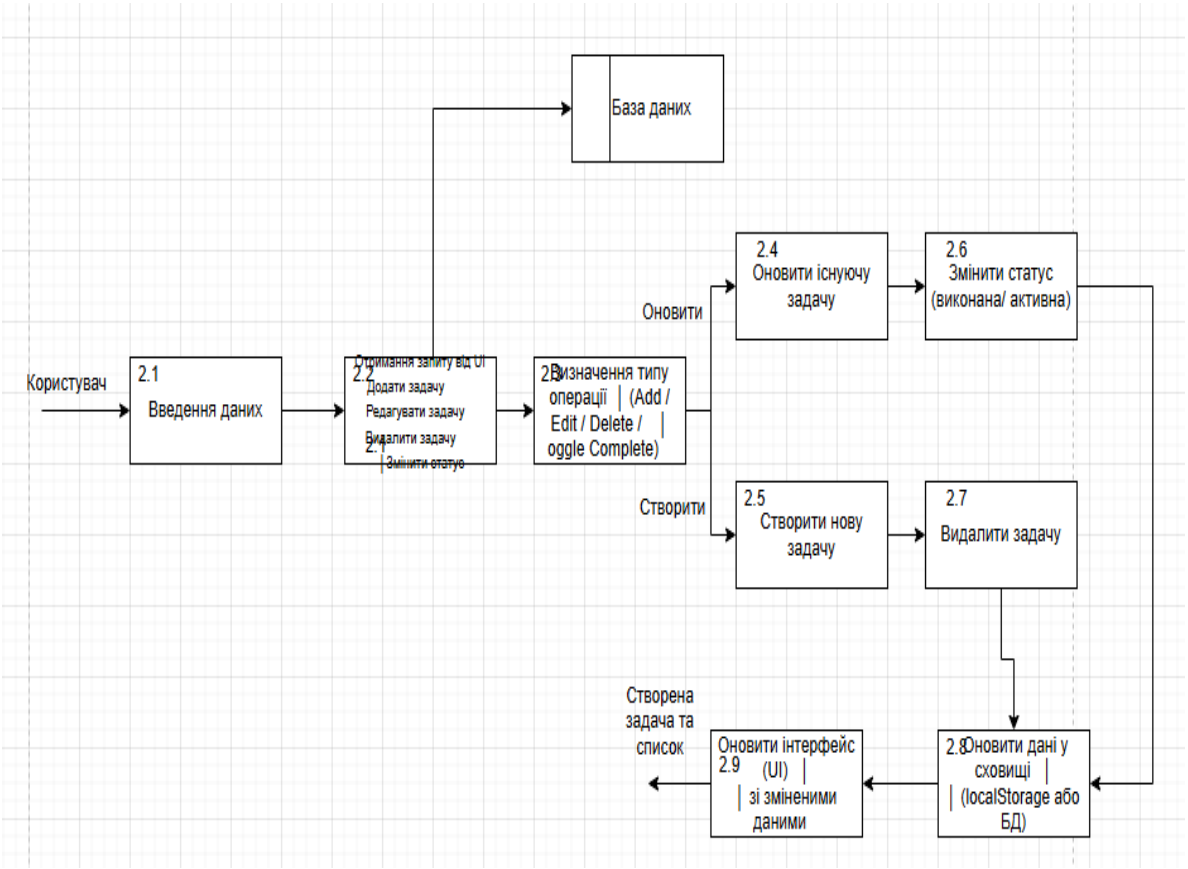

Рисунок 2.7 – Деталізація блоку «Модуль користувача»

Таблиця 2.3 – Модуль користувача

№	Етап	Опис
1	Отримання запиту	Інтерфейс користувача надсилає запит (наприклад, натискання кнопки “Додати” або “Видалити”).
2	Визначення дії	Система визначає, яку саме операцію потрібно виконати.
3	Обробка дії	Виконується додавання, редагування, видалення або зміна статусу задачі.
4	Збереження даних	Зміни записуються у локальне сховище (localStorage) або базу даних.
5	Оновлення інтерфейсу	UI оновлюється, щоб відобразити актуальний стан списку задач.

2.2 Розробка та докладний опис алгоритму роботи програми

2.2.1 Опис функціональних та нефункціональних вимог до систем

Функціональні вимоги визначають ключові можливості програмного забезпечення. Для застосунку персонального планування задач це:

1.Створення та ведення задач:

Можливість заводити нові задачі з назвою, описом, терміном виконання, пріоритетом та категорією. Редагування існуючих задач. Вилучення непотрібних або завершених задач.

2.Встановлення статусу задачі:

Можливість вказувати, чи задача виконана, чи активна. Сортування задач за статусом.

3.Фільтрація та пошук:

Відбір задач за датою, пріоритетом, категорією або статусом. Пошук задач за словами-ключами.

4.Відображення задач:

Показ задач у вигляді списку, календаря або канбан-дошки. Оновлення інформації в режимі реального часу.

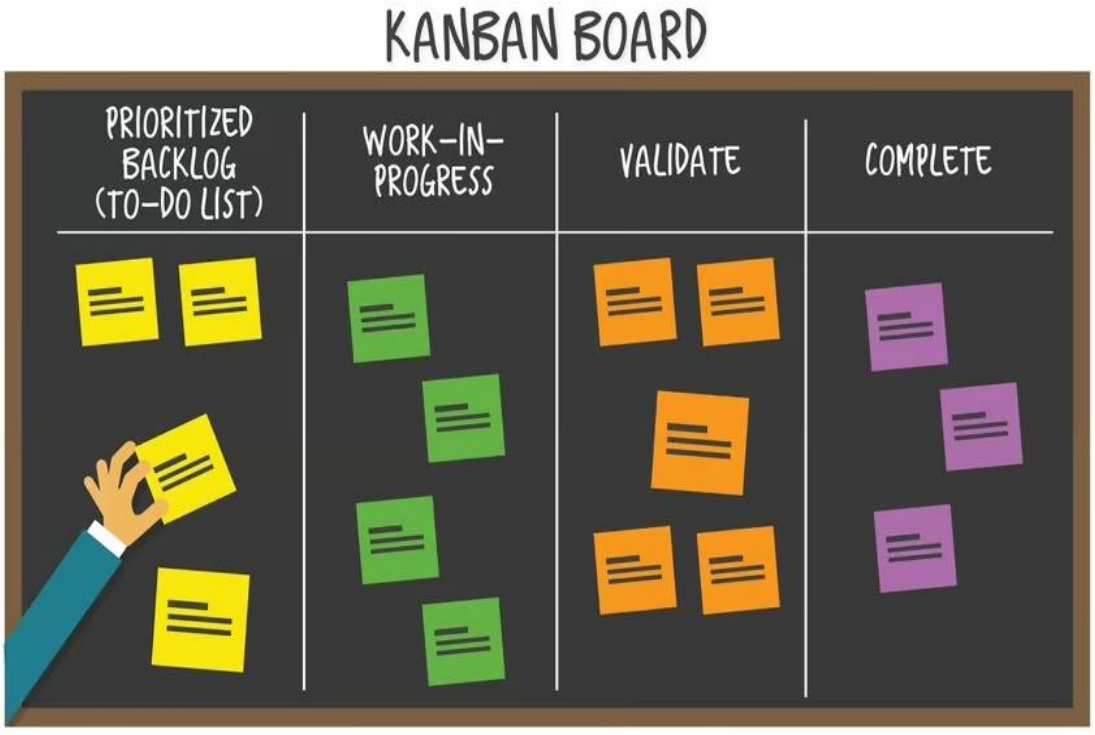

Рисунок 2.8 – Канбан дошка

5.Повідомлення про терміни:

Нагадування про наближення дедлайну (через повідомлення в застосунку або електронною поштою).

6.Аналіз продуктивності:

Обчислення кількості завершених задач за обраний період. Представлення статистики у вигляді діаграм або графіків.

7.Автоматичне повторення задач:

Підтримка задач, які потрібно виконувати регулярно (щодня, щотижня тощо).

8.Збереження даних:

Збереження даних локально (LocalStorage) або в базі даних (Firebase / MongoDB / SQLite). Можливість експорту та імпорту даних.

Нефункціональні вимоги

Нефункціональні вимоги визначають властивості програмного забезпечення, такі як зручність, надійність, швидкість та безпека. Система має коректно працювати на різних пристроях, таких як комп'ютери, планшети та смартфони. Інтерфейс повинен бути зрозумілим, з мінімальним дизайном. Основні дії, наприклад, створення, редагування та видалення, повинні виконуватися за один-два кліки. Система має швидко реагувати, навіть якщо є багато завдань. Доступ до локального чи віддаленого сховища має бути швидким.

Надійність і збереження даних

У разі втрати з'єднання інтерфейс продовжить працювати в автономному режимі. Усі зміни будуть синхронізовані після відновлення мережі. Необхідно забезпечити конфіденційність особистих даних користувачів та захист від несанкціонованого доступу до сховища. Кросплатформеність і кросбраузерність. Програма має підтримувати роботу в сучасних браузерах (Chrome, Firefox, Edge, Safari) та бути сумісна з різними операційними системами (Windows, Android, iOS). Систему можна буде розширити, наприклад, додати спільні завдання або групові календарі.

Таблиця 2.2 – Функціональні та нефункціональні вимоги до системи

№	Назва вимоги	Тип вимоги	Опис	Очікуваний результат
1	Створення задач	Функціональна	Користувач може створювати нові задачі, вказуючи назву, опис, термін виконання, пріоритет та категорію.	Задача зберігається у списку, відображається в інтерфейсі.
2	Редагування задач	Функціональна	Можливість змінювати параметри раніше створеної задачі.	Дані задачі оновлюються без втрати інформації.
3	Видалення задач	Функціональна	Користувач може видалити непотрібну або виконану задачу.	Задача зникає зі списку та з бази даних/LocalStorage.
4	Позначення виконання	Функціональна	Можливість змінювати статус задачі: активна / виконана.	Інтерфейс оновлює відображення статусу.
5	Фільтрація задач	Функціональна	Фільтрація за датою, пріоритетом, категорією, статусом або тегами.	Відображаються лише задачі, що відповідають фільтру.
6	Пошук задач	Функціональна	Пошук задач за ключовими словами.	Швидке знаходження потрібної задачі серед великої кількості.
7	Візуалізація задач	Функціональна	Перегляд задач у вигляді списку, календаря або Kanban-дошки.	Гнучке уявлення процесу роботи користувача.
8	Повідомлення про дедлайн	Функціональна	Автоматичне нагадування користувачу про наближення терміну задачі.	Користувач отримує push- або email-сповіщення.

№	Назва вимоги	Тип вимоги	Опис	Очікуваний результат
9	Автоматичне повторення	Функціональна	Можливість налаштувати повторення задачі (щодня, щотижня, щомісяця).	Задача автоматично додається до списку у новому періоді.
10	Аналітика ефективності	Функціональна	Відстеження виконаних задач, часу, статистики продуктивності.	Користувач бачить графіки виконання задач.
11	Збереження даних	Функціональна	Використання LocalStorage або бази даних (Firebase / MongoDB).	Дані користувача не втрачаються після перезавантаження.
12	Адаптивність інтерфейсу	Нефункціональна	Інтерфейс адаптується під ПК, планшет, смартфон.	Коректне відображення на всіх пристроях.
13	Зручність використання	Нефункціональна	Простий і зрозумілий інтерфейс з мінімумом кроків для основних дій.	Користувач швидко розуміє принцип роботи програми.
14	Швидкодія системи	Нефункціональна	Мінімальна затримка при роботі навіть із великою кількістю задач.	Плавна робота без зависань.
15	Режим офлайн	Нефункціональна	Можливість роботи без інтернету з подальшою синхронізацією.	Користувач може додавати задачі без підключення.
16	Безпека даних	Нефункціональна	Конфіденційність користувацьких даних та захист від стороннього доступу.	Дані зберігаються у захищеному вигляді.
17	Кросбраузерна сумісність	Нефункціональна	Підтримка Chrome, Firefox, Edge, Safari.	Додаток працює стабільно у різних браузерах.

№	Назва вимоги	Тип вимоги	Опис	Очікуваний результат
18	Масштабованість системи	Нефункціональна	Можливість розширення функціоналу (групові задачі, календарі).	Розробка нових модулів без змін у базовій архітектурі.
19	Надійність	Нефункціональна	Система повинна коректно обробляти помилки введення, втрату зв’язку.	Відсутність збоїв при роботі програми.
20	Підтримка локалізації	Нефункціональна	Можливість зміни мови інтерфейсу (українська, англійська).	Інтерфейс адаптується до мови користувача.

2.2.2 Технології та інструменти планувальника задач

Для реалізації програмного забезпечення планувальника задач було використано сучасні веб-технології, які забезпечують стабільну роботу, зручність користування та можливість подальшого масштабування системи.

Основними технологіями клієнтської частини стали **HTML5**, **CSS3** та **JavaScript (ES6+)**. Вони використовуються для створення структури, стилізації та динамічної взаємодії інтерфейсу з користувачем. Для побудови компонентного, гнучкого та реактивного інтерфейсу застосовано фреймворк **React.js**, що дозволяє розробляти модульні елементи та керувати станом додатка. Для маршрутизації між сторінками використовується **React Router**.

Стилізація інтерфейсу реалізована з використанням **Tailwind CSS**, що дозволяє швидко створювати адаптивний та мінімалістичний дизайн. Для додавання плавних переходів і візуальних ефектів застосовано **Framer Motion**.

На серверній стороні (у разі використання backend) застосовується **Node.js** у поєднанні з фреймворком **Express.js**, які забезпечують створення REST API для взаємодії клієнтської частини з базою даних. Для зберігання даних може використовуватись **MongoDB** (через бібліотеку **Mongoose**) або локальне сховище браузера (**LocalStorage**, **IndexedDB**, або **SQLite** у мобільній версії).

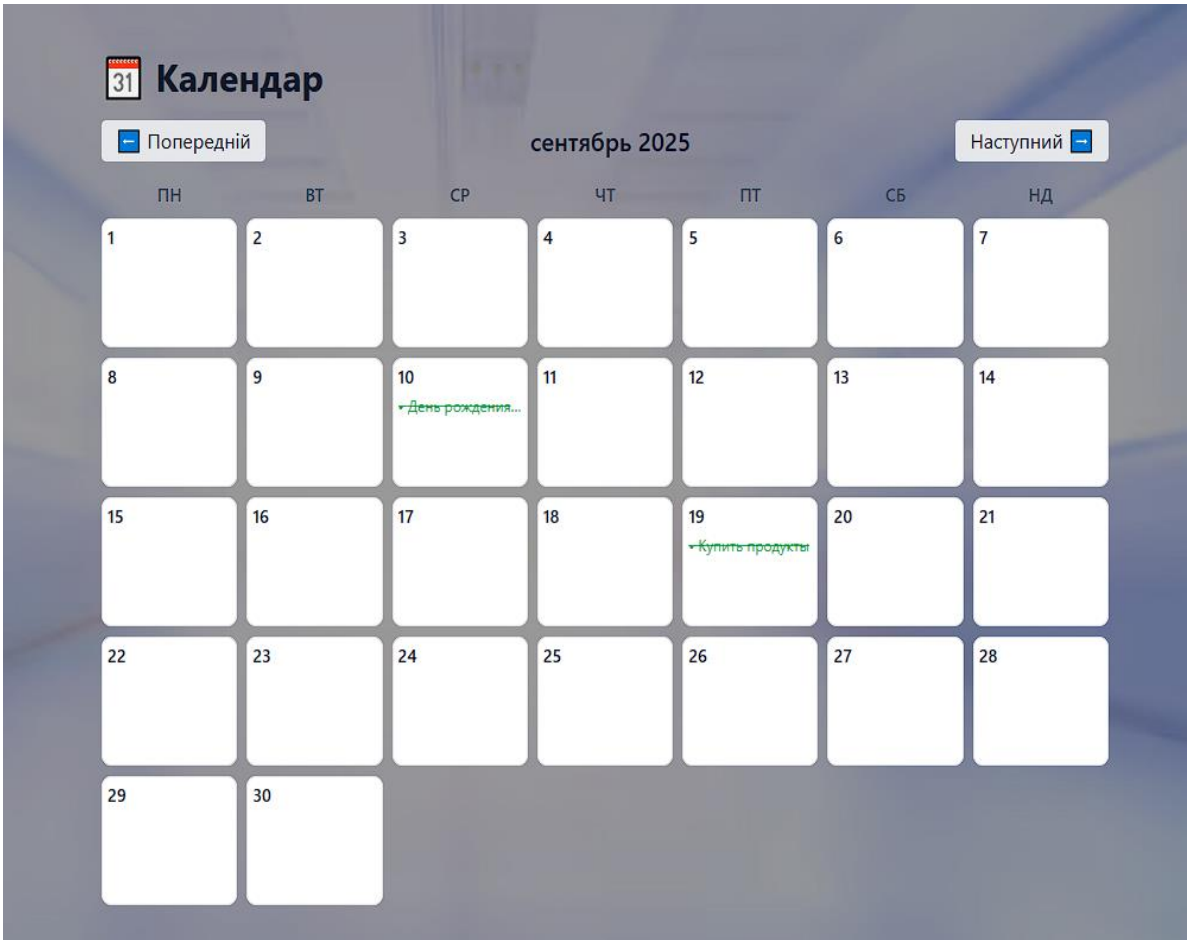

Рисунок 2.9 – Синхронізація задач з календарем

На рисунку зображено процес інтеграції задач із календарним модулем системи. Механізм синхронізації забезпечує автоматичне відображення створених користувачем завдань у відповідних датах та часових інтервалах календаря. Коли користувач додає або редагує задачу, її параметри назва, дедлайн, пріоритет та статус - передаються до календарного компонента, який оновлює відповідну клітинку календарної сітки.

Система також перевіряє наявність конфліктів за часом, дублювання подій або перетинів між задачами. У разі зміни дедлайну або перенесення задачі дані миттєво оновлюються в обох модулях: у списку задач та в календарному представленні. Такий підхід забезпечує цілісність інформації, зручність планування та підвищує точність управління робочим часом.

Для побудови графіків аналітики та статистики ефективності користувача застосовуються бібліотеки **Chart.js** або **Recharts**. Вони дозволяють візуалізувати

відсоток виконаних задач, продуктивність за тиждень або день, а також інші показники.

Інтеграція із зовнішніми сервісами (наприклад, **Google Calendar API**) використовується для синхронізації подій і дедлайнів.

Для тестування API застосовується **Postman**, а для контролю версій - **Git** із хостингом репозиторію на **GitHub**. Розробка проводилась у середовищі **Visual Studio Code** з використанням менеджера пакетів **npm** для встановлення бібліотек та залежностей.

Також під час створення дизайну інтерфейсу застосовувалися інструменти **Figma** або **Canva**, які допомогли створити макети сторінок і загальний стиль додатку.

У результаті використання цих технологій створено зручний, адаптивний та швидкодіючий веб-додаток, який забезпечує ефективне управління задачами, ведення аналітики та підтримку офлайн-режиму.

Таблиця 2.3 – Технології та їх опис

№	Технологія Інструмент	Призначення / Опис
1	React Context API	Використовується для глобального керування станом додатка без потреби у зовнішніх бібліотеках (наприклад, Redux).
2	React Hooks (useState, useEffect, useMemo)	Дозволяють ефективно керувати станом і життєвим циклом компонентів, оптимізують роботу програми.
3	React Router	Забезпечує маршрутизацію між сторінками без перезавантаження вебдодатку (SPA-архітектура).
4	Recharts	Використовується для побудови інтерактивних діаграм аналітики продуктивності користувача.
5	Framer Motion	Бібліотека для створення плавних анімацій і переходів між компонентами.

№	Технологія / Інструмент	Призначення / Опис
6	LocalStorage / IndexedDB	Забезпечують локальне зберігання даних користувача навіть без підключення до інтернету.
7	Mongoose	Інструмент для взаємодії Node.js із базою даних MongoDB, забезпечує моделювання даних і валідацію.
8	Postman	Застосовується для тестування REST API-запитів і налагодження серверної взаємодії.
9	Git + GitHub	Система контролю версій і платформа для зберігання вихідного коду проєкту.
10	npm (Node Package Manager)	Менеджер пакетів для встановлення бібліотек, необхідних для роботи фронтенду й бекенду.
11	Figma / Canva	Використовуються для створення макетів дизайну інтерфейсу користувача.
12	Visual Studio Code	Основне середовище розробки з підтримкою плагінів для React, Tailwind CSS і Git.

2.2.3 Алгоритми роботи планувальника задач

Алгоритми роботи програмного забезпечення планувальника задач визначають логіку взаємодії користувача з інтерфейсом, обробку введених даних, збереження інформації та аналітику ефективності. Нижче подано опис основних алгоритмів системи.

1. Алгоритм додавання задачі

Користувач натискає кнопку **«Додати задачу»**. Відкривається форма введення з полями: назва, опис, дата виконання, пріоритет.Система перевіряє правильність введених даних (наявність назви, коректність дати).

Якщо дані заповнені коректно - задача зберігається у локальній базі (LocalStorage або MongoDB). Інтерфейс оновлюється - нова задача

відображається у списку поточних. У разі помилки система виводить повідомлення користувачу.

2. Алгоритм редагування задачі

Користувач вибирає задачу зі списку. Натискає кнопку **«Редагувати»**. Відкривається форма з поточними даними задачі. Користувач змінює необхідні параметри. Система перевіряє коректність змінених даних. Після підтвердження зміни зберігаються, а інтерфейс оновлюється.

3. Алгоритм видалення задачі

Користувач натискає на кнопку **«Видалити»** поруч із задачею. Відкривається модальне вікно з підтвердженням. Якщо користувач підтверджує дію - задача видаляється з бази даних. Інтерфейс оновлюється без перезавантаження сторінки.

4. Алгоритм зміни статусу задачі (виконано/не виконано)

Користувач позначає задачу як виконану (через чекбокс або кнопку). Система оновлює її статус у базі даних. Інтерфейс оновлює список активних і завершених задач. Дані передаються до модуля аналітики.

5. Алгоритм аналітики продуктивності

Система підраховує кількість завершених задач за день, тиждень або місяць. Формуються статистичні показники ефективності (відсоток виконання, середній час виконання). Дані передаються до модуля візуалізації (Recharts). Відображається діаграма (стовпчикова, кругова або лінійна).

6. Алгоритм фокус-режиму (Pomodoro Mode)

Користувач активує режим фокусування. Система запускає таймер робочого циклу (наприклад, 25 хвилин). Під час роботи інші дії блокуються. Після завершення циклу система подає сигнал і запускає коротку перерву (5 хв).

Дані про цикл записуються в аналітику продуктивності.

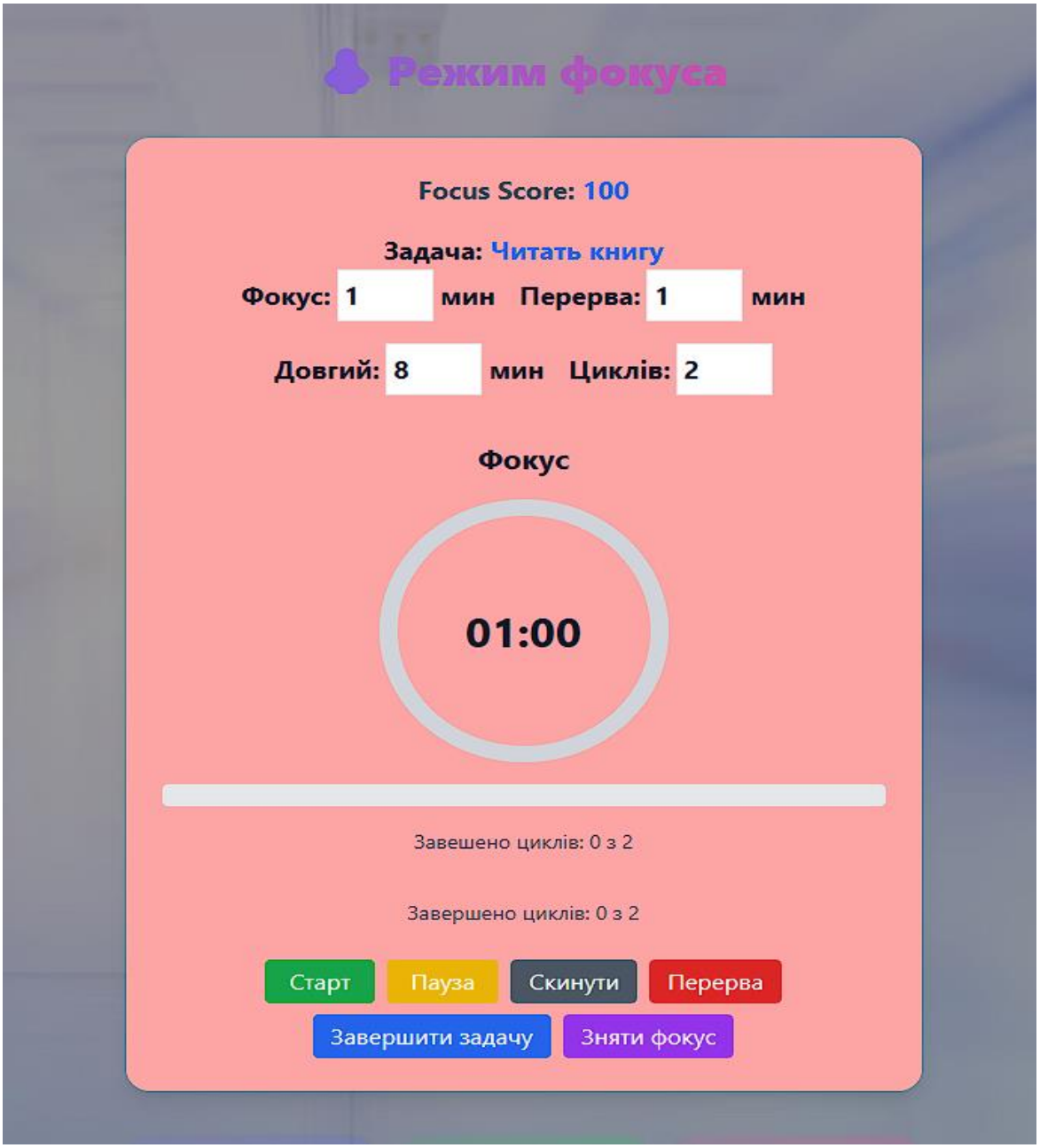

Рисунок 2.10 – Режим фокусу на задачі

На рисунку представлено інтерфейс режиму фокусування, призначеного для роботи з однією вибраною задачею без відволікаючих факторів. У цьому режимі система приховує всі другорядні елементи інтерфейсу - списки задач, меню, панелі інструментів — залишаючи на екрані лише ключову інформацію про поточне завдання.

Режим фокусу включає таймер (наприклад, техніки Pomodoro), який відліковує інтервали концентрації та відпочинку. Протягом активної сесії

користувач отримує мінімум відволікань, а система фіксує тривалість активної роботи над задачею, що надалі враховується в аналітиці продуктивності.

Користувач може керувати сесією через базові елементи управління - запуск, пауза, завершення - а також переглядати прогрес виконання. Такий підхід підвищує зосередженість, сприяє ефективному розподілу часу та допомагає успішніше виконувати складні або пріоритетні задачі.

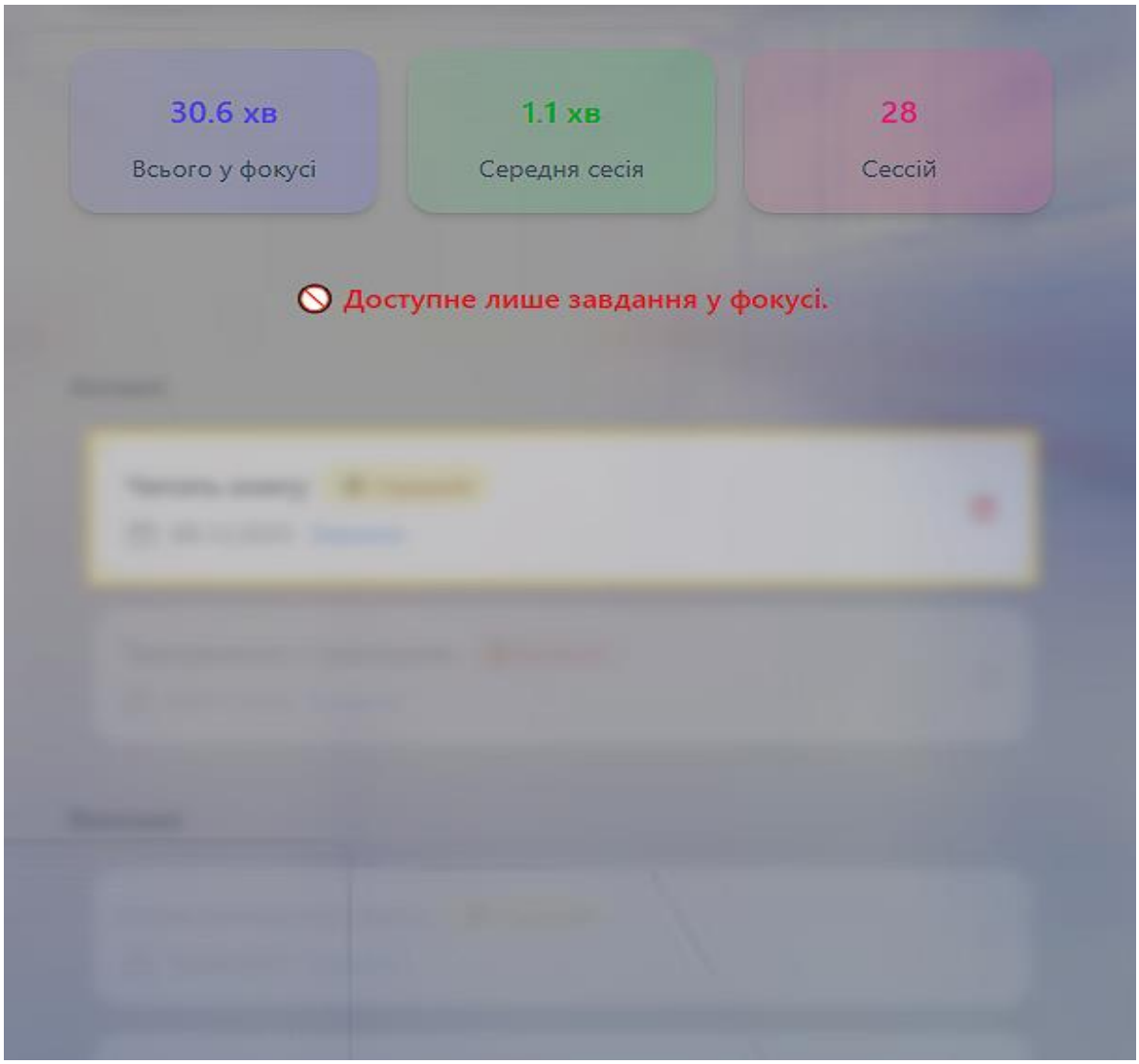

Рисунок 2.11 – Фокусування на одній задачі та блокування інших

На рисунку зображено механізм роботи режиму глибокої концентрації, у якому система надає користувачу можливість зосередитися на виконанні однієї вибраної задачі, одночасно блокуючи доступ до решти елементів інтерфейсу. У цьому режимі всі інші задачі стають недоступними для редагування,

переміщення або запуску, що запобігає переключенню уваги й мінімізує ризик відволікань.

Інтерфейс у режимі блокування підсвічує активну задачу та затемнює або приховує неактивні елементи. Кнопки управління та дії, пов’язані з іншими задачами, відключені або позначені іконками блокування. Такий підхід забезпечує чітку концентрацію на одному завданні та підсилює ефективність роботи користувача.Система фіксує час перебування у фокусі та відстежує прогрес виконання вибраної задачі. Це дозволяє інтегрувати дані в модуль аналітики й отримувати точніші показники продуктивності. Режим блокує будь-які спроби відкриття інших задач до завершення або примусового виходу з фокус-сесії.

Таблиця 2.4 – Основні алгоритми роботи планувальника задач

№	Назва алгоритму	Призначення	Вхідні дані	Вихідні дані	Короткий опис
1	Алгоритм додавання задачі	Додавання нової задачі у систему	Назва, опис, дедлайн, пріоритет	Запис у базу даних, оновлення інтерфейсу	Перевірка введених даних, створення об’єкта задачі та збереження його у сховищі
2	Алгоритм редагування задачі	Оновлення існуючих даних задачі	Ідентифікатор задачі, змінені дані	Оновлені дані в базі	Дозволяє змінювати поля задачі без її видалення
3	Алгоритм видалення задачі	Видалення вибраної задачі	Ідентифікатор задачі	Оновлений список задач	Підтвердження дії користувачем, видалення задачі зі сховища

№	Назва алгоритму	Призначення	Вхідні дані	Вихідні дані	Короткий опис
4	Алгоритм зміни статусу	Позначення задачі як виконаної/активної	Ідентифікатор задачі, новий статус	Оновлений статус задачі	Автоматичне оновлення інтерфейсу та передача даних у модуль аналітики
5	Алгоритм аналітики продуктивності	Розрахунок статистики виконання	Дані про виконані задачі	Діаграми, графіки, відсотки	Аналіз кількості, часу і відсотка виконання задач
6	Алгоритм Pomodoro (фокус-режим)	Підвищення концентрації під час роботи	Обрана задача, час фокус-сесії	Результати циклів, статистика	Запуск таймера, блокування інших дій, підрахунок циклів продуктивності

Алгоритми аналізу продуктивності

У системі планування завдань алгоритми аналізу продуктивності збирають, обробляють і показують статистику про активність користувача. Головне завдання – оцінити, наскільки добре виконуються завдання, знайти певні моделі в роботі користувача та дати поради, як працювати краще.

Спочатку алгоритми збирають дані. Система сама бере інформацію про створені, активні та завершені завдання, записує дату створення, кінцевий термін, важливість і час завершення. Ці дані зберігаються в базі даних і оновлюються, коли змінюється статус завдання.

Далі дані обробляються та аналізуються. Алгоритм обчислює:

- загальну кількість завдань за певний час (день, тиждень, місяць);

- кількість виконаних завдань;
- відсоток виконання (скільки завдань виконано відносно запланованих);
- середній час виконання одного завдання;
- які дні тижня є найбільш productive (на основі кількості завершених завдань);
- як змінюється продуктивність з часом (у вигляді графіка або діаграми).

Нарешті, результати показуються у зручному вигляді. Аналітичний модуль створює графіки та діаграми, щоб показати продуктивність користувача. Це допомагає швидко зрозуміти свій прогрес і побачити, в які дні чи періоди продуктивність була найвищою. Система також може аналізувати себе та помічати дні з низькою активністю, пропонуючи поради: наприклад, збільшити час роботи без перерв або зменшити кількість завдань на день. Так, алгоритми аналізу продуктивності не просто показують результати, а й допомагають користувачам краще організовувати свій час.

```
const now = new Date();
const filteredTasks = tasks.filter(task => {
    const date = new Date( value: task.completedAt || task.updatedAt || task.dueDate);
    if (period === "day") {
        return date.toDateString() === now.toDateString();
    }
    if (period === "week") {
        const weekAgo = new Date();
        weekAgo.setDate(now.getDate() - 7);
        return date >= weekAgo && date <= now;
    }
    if (period === "month") {
        return date.getMonth() === now.getMonth() && date.getFullYear() === now.getFullYear();
    }
    return true;
})

const completedTasks = filteredTasks.filter(t => t.completed);

const tasksByDay = completedTasks.reduce((acc, task) => {
    const date = new Date( value: task.completedAt || task.updatedAt || task.dueDate);
    const day = date.toLocaleDateString( locales: "ru-RU", options: { weekday: "short" });
    acc[day] = (acc[day] || 0) + 1;
    return acc;
}, {} as Record<string, number>);
```

Рисунок 2.9 – Функція фільтрації задач по дням

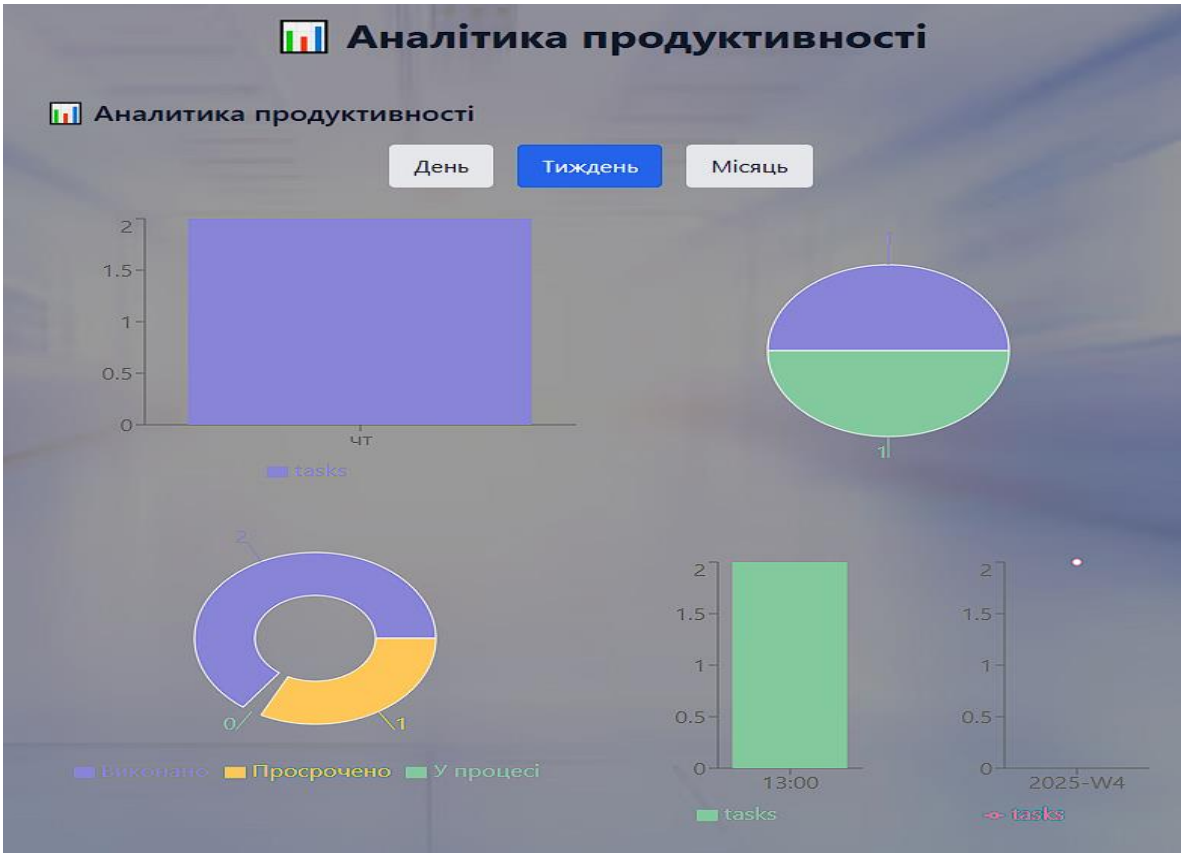

Рисунок 2.10 – Прикалд використання функції у планувальнику задач

Компонент **PomodoroTimer** призначений для організації роботи користувача за методикою Pomodoro у рамках планувальника задач. Він забезпечує точне відлічування часу для фокус-сесій, коротких і довгих перерв, автоматично перемикаючи режими залежно від кількості завершених циклів. Кожен режим має окремі налаштування тривалості, які можна зберігати в локальному сховищі, що дозволяє користувачу налаштувати таймер під свої індивідуальні потреби.

Таймер реалізовано з використанням useState, useEffect та useRef, що забезпечує стабільне оновлення стану та контроль інтервалів без втрати продуктивності. Логіка включає запуск, паузу, скидання та переривання таймера, а також можливість завершення задачі вручну з одночасним відправленням інформації на сервер через API, що дозволяє зберігати статистику виконання.

Візуально прогрес сесії відображається у вигляді анімованого кола, яке плавно змінює заповнення відповідно до часу, що залишився. Крім того, відображається назва поточного режиму, залишок часу у хвилинах і секундах,

кількість завершених циклів і прогрес до довгої перерви. Компонент також інтегрує підкомпоненти для налаштувань (PomodoroSettings), управління таймером (PomodoroControls) та відображення прогресу циклів (PomodoroProgress), що робить його модульним та легко розширюваним.

Такий підхід дозволяє користувачу ефективно керувати своїм часом, підтримувати концентрацію на завданні, уникати перевтоми та отримувати зворотний зв'язок про свою продуктивність у реальному часі. Завдяки зберіганню даних у локальному сховищі та синхронізації з сервером забезпечується безпека інформації та можливість продовжити роботу навіть після перезавантаження сторінки.

```
export default function PomodoroTimer({ task, onStop, onCompleteTask }) {

    const defaultSettings = {
        focus: 25,
        shortBreak: 5,
        longBreak: 15,
        cyclesBeforeLongBreak: 4,
    };

    const [settings, setSettings] = useState( initialState: () => {
        return { ...defaultSettings, ...JSON.parse( text: localStorage.getItem( key: "pomodoroSettings") || "{}") };
    });

    const [seconds, setSeconds] = useState( initialState: settings.focus * 60);
    const [mode, setMode] = useState<"focus" | "shortBreak" | "longBreak">( initialState: "focus");
    const [isRunning, setIsRunning] = useState( initialState: false);
    const [cycles, setCycles] = useState( initialState: 0);
    const intervalRef = useRef( initialValue: null);
    const { log } = useMethodTracking();
    const [animate, setAnimate] = useState( initialState: false);


    useEffect( effect: () => clearInterval(intervalRef.current), deps: []);

    useEffect( effect: () => {
        if (mode === "focus") setSeconds( value: settings.focus * 60);
        if (mode === "shortBreak") setSeconds( value: settings.shortBreak * 60);
        if (mode === "longBreak") setSeconds( value: settings.longBreak * 60);
PomodoroTimer()
```

Рисунок 2.11 – Початок компонента PomodoroTimer

2.3 Опис та порівняння методів планувальника задач та фокусу

2.3.1 Алгоритми

Методи планування задач та фокусування мають не лише теоретичні переваги, а й підтверджуються практичною статистикою ефективності. Наприклад, дослідження показують, що користувачі, які використовують цикли Pomodoro, підвищують продуктивність приблизно на 25–40% за рахунок концентрації на конкретній задачі без відволікань та регулярних коротких перерв. Time Blocking дозволяє зменшити кількість пропущених дедлайнів до 50% і покращує управління робочим часом, оскільки день чітко розбивається на інтервали, які виділені під конкретні завдання.

Методика “Eat That Frog” демонструє, що виконання найважливіших завдань у першій половині дня підвищує відчуття досягнення і скорочує прокрастинацію до 30–35%, що також позитивно впливає на загальну ефективність. Інтеграція цих методів у сучасні цифрові планувальники дозволяє об’єднати структуроване планування та оптимальне фокусування на задачах. За статистикою користувачі, які комбінують управління списками, календарем і Kanban-дошками з методами фокусування, зменшують час на виконання завдань приблизно на 20–25% і збільшують кількість завершених завдань щодня на 30–50%, що робить такі інструменти особливо ефективними для продуктивної роботи.

Алгоритм пріоритезації задач

Опис: цей алгоритм визначає пріоритет кожної задачі на основі термінів виконання, важливості та впливу на загальний прогрес. Він допомагає користувачу фокусуватися на найважливіших завданнях у першочерговому порядку.

Переваги:

Дозволяє ефективніше розподіляти час. Підвищує продуктивність користувача.

Недоліки:

Може не враховувати особисті пріоритети користувача. Потребує правильної оцінки важливості задач для коректної роботи.

2.3.2 Алгоритм розподілу циклів фокусування (Pomodoro)

Опис: цей алгоритм організовує робочий час у цикли фокусування та короткі або довгі перерви, щоб максимізувати продуктивність і запобігти вигоранню. Кількість циклів і тривалість встановлюються користувачем або системою.

Переваги:

Покращує концентрацію та регулярність роботи. Допомагає контролювати час та уникати перевтоми.

Недоліки:

Менш ефективний для задач, що потребують тривалої безперервної уваги. Вимагає дисципліни користувача для дотримання циклів.

Алгоритм нагадувань та дедлайнів

Опис: цей алгоритм відстежує час виконання задач і надсилає користувачу нагадування про наближення дедлайнів або відкладені завдання.

Переваги:

Зменшує ризик пропуску важливих задач. Підвищує дисципліну та відповідальність користувача.

Недоліки:

Може дратувати при надмірній кількості сповіщень. Не враховує пріоритетність задач, якщо нагадування налаштовані стандартно.

2 РОЗРОБКА БАЗИ ДАНИХ І ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз обраної середи програмування

Для створення особистого планувальника завдань я обрав JavaScript з React та Node.js. JavaScript має велику екосистему, є гнучким і працює на різних платформах, що дозволяє зробити веб- чи мобільну версію застосунку.

React спрощує розробку, тестування та повторне використання елементів інтерфейсу завдяки своїй компонентній структурі. Він дає змогу зробити інтерфейс швидким та адаптивним, який одразу реагує на зміни без перезавантаження сторінки.

Node.js є основою для серверної частини застосунку. Він швидко обробляє запити завдяки асинхронній обробці, що важливо при великій кількості користувачів або даних.

Ми додали Tailwind CSS для стилізації, MongoDB для даних та localStorage для локальної роботи, щоб створити баланс між клієнтською та серверною частинами. Tailwind допомагає підтримувати простий дизайн, а MongoDB – зручно зберігати інформацію у форматі JSON.

Ці технології дозволяють нам:

- швидко працювати з великою кількістю завдань;
- адаптувати інтерфейс під різні пристрої;
- розширювати функціонал (додавати аналітику, синхронізацію з календарями, AI);
- легко підтримувати та масштабувати код.

Використання React + Node.js є хорошим рішенням для сучасних вебзастосунків, яким потрібна стабільність, інтерактивність та продуктивність, таких як персональний планувальник завдань.

WebStorm – це інтегроване середовище розробки від JetBrains, зроблене для JavaScript, TypeScript, HTML, CSS і сучасних фреймворків, таких як React, Vue, Angular. Його головна перевага – це тісний зв'язок з JavaScript, що спрощує

роботу з великими проєктами, допомагає швидко знаходити помилки і покращувати код.

Основні можливості WebStorm:

Розумне автодоповнення коду: пропонує підказки для змінних, функцій, методів і компонентів, що прискорює написання коду. Вбудований налагоджувач для коду на стороні клієнта і сервера: дозволяє відстежувати помилки та змінні в реальному часі.

Інтеграція з системами контролю версій (Git, GitHub, GitLab): дає можливість працювати з репозиторіями прямо з IDE. Підтримка npm, Yarn та інших менеджерів пакетів: полегшує встановлення та оновлення. Вбудований термінал і Live Edit: дозволяють бачити зміни у браузері без перезавантаження програми. Застосування WebStorm у розробці позитивно вплинуло на продуктивність, зменшило кількість помилок і прискорило тестування. Завдяки налаштуванню середовища під проєкт, розробник отримує зручне та надійне місце для роботи як з фронтендом (React), так і з бекендом (Node.js).

Отже, поєднання React, Node.js і WebStorm формує хорошу основу для створення персонального планувальника задач. Це забезпечує швидкий процес розробки, зручну відладку, пристосованість до різних пристроїв, а також простоту розширення та підтримки коду в майбутньому.

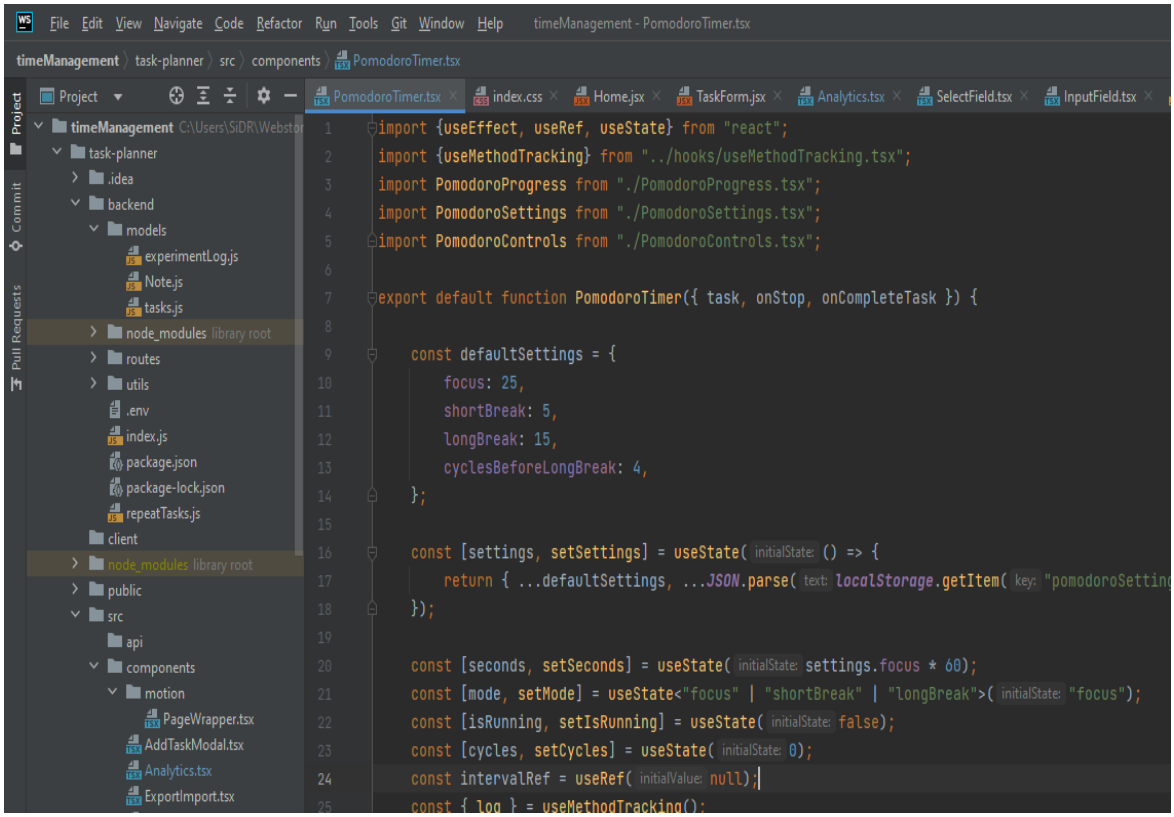

Рисунок 3.1 – Інтерфейс середовища WebStorm

Термінал у WebStorm є дуже зручним для розробників, оскільки дає змогу виконувати всі потрібні команди прямо в IDE, без потреби переходити між вікнами або іншими програмами. Це особливо допомагає при роботі з JavaScript, Node.js і React.

Він вбудований у WebStorm і підтримує ті ж команди, що і звичайна командна оболонка вашої операційної системи, наприклад PowerShell, Bash або Zsh. Щоб його відкрити, перейдіть у меню View → Tool Windows → Terminal або скористайтеся клавішами Alt + F12.

За допомогою термінала ви можете:

- запускати сервери розробки командами `npm start` або `npm run dev`;
- ставити бібліотеки та пакети через `npm install` або `yarn add`;
- використовувати команди Git, такі як `commit`, `push`, `pull` і `branch`, безпосередньо у WebStorm;
- керувати процесом збірки або тестування (наприклад, через `npm test`);
- працювати зі скриптами Node.js, не виходячи з середовища розробки.

Термінал також має декілька вкладок, що дозволяє одночасно запускати різні процеси, наприклад, фронтенд-сервер React і бекенд Node.js. Це робить розробку зручнішою та швидшою.

Ще один позитив – інтеграція з IDE: термінал автоматично відкривається в головній папці проєкту, показує структуру папок, і всі зміни відразу видно у файловій системі WebStorm.

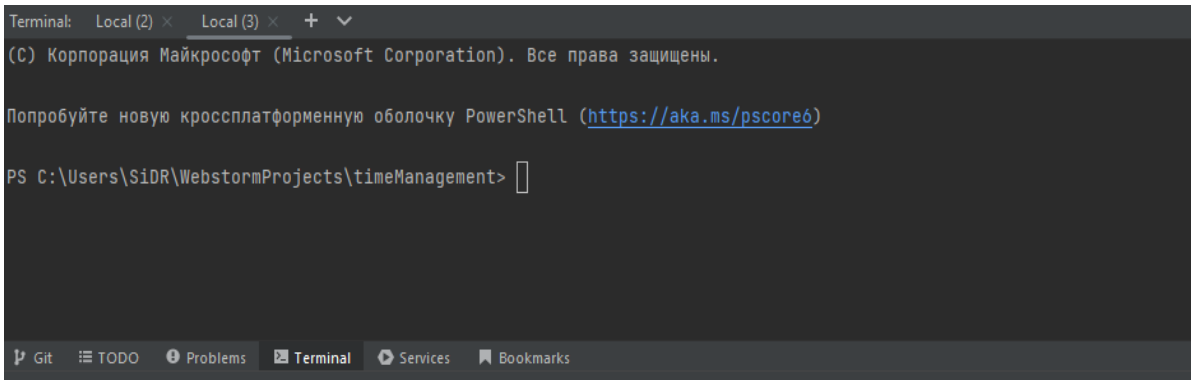

Рисунок 3.2 – Термінал середовища WebStorm

3.2 Розробка бази даних

Для створення планувальника задач на Node.js я застосував базу даних MongoDB та бібліотеку Mongoose. Mongoose – це корисний ORM (інструмент для роботи з базами даних), що спрощує взаємодію між застосунком та базою.

Mongoose дає змогу визначати структуру документів через схеми, створювати моделі для роботи з даними та виконувати основні операції з ними (створення, читання, оновлення, видалення).

У планувальнику задач база даних зберігає таку інформацію:

Task (Задача) – містить назву, опис, важливість, дату виконання та статус.

PomodoroSession (Сесія Pomodoro) – зберігає дані про кількість виконаних циклів Pomodoro для кожної задачі.

Statistics (Статистика) – зберігає інформацію про продуктивність користувача, наприклад, скільки задач він виконав за день або тиждень.

```
import mongoose from "mongoose";

const taskSchema = new mongoose.Schema({
  title: { type: String, required: true },
  description: { type: String },
  priority: { type: String, enum: ["низький", "середній", "високий"], default: "середній" },
  dueDate: { type: Date },
  completed: { type: Boolean, default: false },
  pomodoroCycles: { type: Number, default: 0 },
}, { timestamps: true });

export default mongoose.model("Task", taskSchema);
```

Рисунок 3.3 – Приклад схеми задачі

Ця структура допомагає зручно керувати завданнями: додавати нові, змінювати їхній стан, лічити закінчені Pomodoro-цикли та аналізувати продуктивність.

Чому Mongoose корисний:

Mongoose спрощує роботу з MongoDB завдяки чітким схемам і перевірці даних. Він підтримує проміжне ПЗ, що дає змогу робити додаткові дії (наприклад, автоматично оновлювати дату зміни запису) до або після збереження документа.

Плюс, Mongoose працює з асинхронними запитами, що дає змогу швидко обробляти дані без зупинки основного процесу. Це важливо для веб-програм, як-от планувальник завдань, де потрібна швидка реакція на дії користувача.

Рисунок 3.4 –Підключення до БД та роут для отримання задач

Збереження даних у Mongoose

У Mongoose збереження даних відбувається через роботу з об'єктами, які показують документи в колекціях MongoDB. Це дає можливість розробникам працювати з базою даних як зі звичайними JavaScript об'єктами, не застосовуючи великих запитів MongoDB.

Ось як відбувається процес збереження даних:

1.Створення схеми: Схема описує структуру документа, типи його полів, чи обов'язкові дані, стандартні значення та інші характеристики.

2.Створення моделі: На основі схеми створюється модель. Вона показує конкретну колекцію в базі даних та забезпечує доступ до CRUD дій.

3.Створення об'єкта моделі: Створюється новий об'єкт (документ), який потім можна зберегти у колекції.

4.Виклик методу .save(): Цей метод безпосередньо зберігає дані в MongoDB.

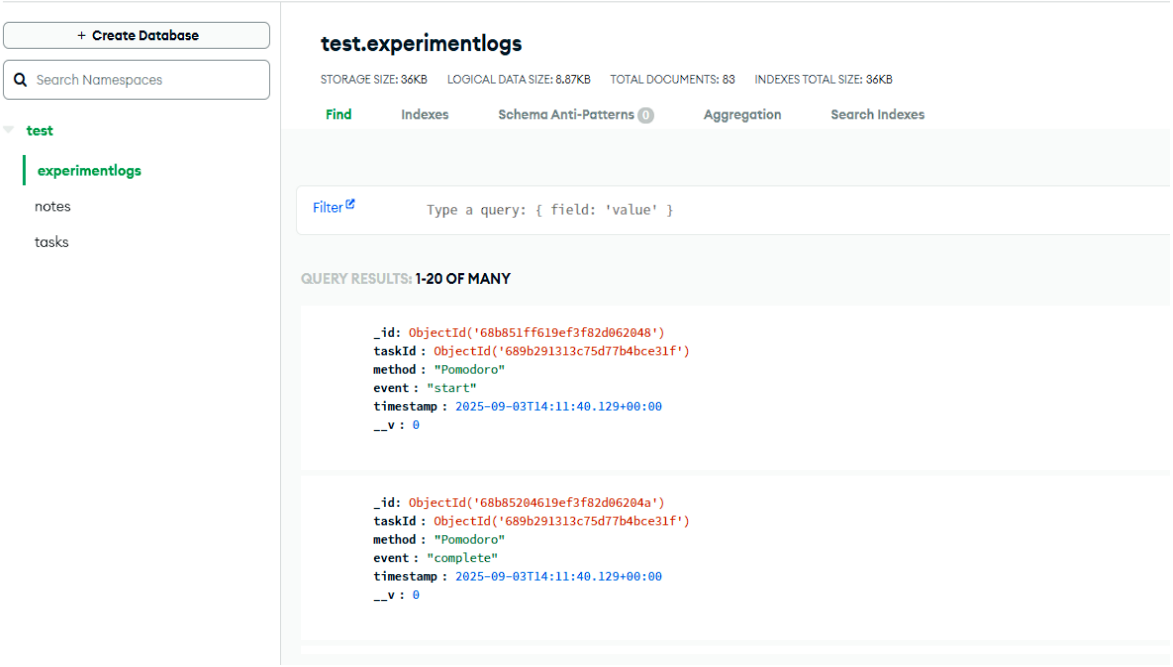

Рисунок 3.5 – Зберігання даних у БД MongooseDB

3.3 Програмна реалізація основних функцій

У персональному планувальнику задач основні функції реалізовані з допомогою React, Node.js (Express) і MongoDB (через Mongoose). Кожен компонент відповідає за певну частину роботи програми: від отримання даних від користувача до їхнього збереження та аналізу.

1.Створення задачі

Інтерфейс створення задач – це React-компонент, який збирає введені користувачем дані (назву, опис, пріоритет, кінцевий термін). Після заповнення форми дані передаються на сервер через HTTP-запит.

На сервері Express отримує ці дані і за допомогою Mongoose створює новий запис у базі даних задач.

```
const handleSubmit = async () => {
  const newTask = { title, description, priority, dueDate };
  await fetch("/api/tasks", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(newTask)
  });
};
```

Рисунок 3.6 – Створення задачі на frontend частині

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

Рисунок 3.7 – Створення задачі на backend частині

2.Редагування та внесення змін до завдань

Користувачі мають можливість змінювати вже існуючі завдання. Це досягається через функцію оновлення, що ініціює PUT-запит до сервера. Сервер ідентифікує завдання за ID та вносить зміни у відповідні поля.

```
app.put("/api/tasks/:id", async (req, res) => {
  try {
    const updatedTask = await Task.findByIdAndUpdate(req.params.id, req.body, { new: true });
    res.json(updatedTask);
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
});
```

Рисунок 3.8 – Редагування задачі

3. Вилучення завдання

Функція вилучення реалізована за допомогою DELETE-запиту. Користувачі можуть вилучати непотрібні або завершені завдання.

```
app.delete("/api/tasks/:id", async (req, res) => {
  try {
    await Task.findByIdAndDelete(req.params.id);
    res.json({ message: "Задачу видалено" });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

Рисунок 3.9 – Видалення задач

4.Позначення задачі як виконаної

В додатку є функція, за допомогою якої користувач може позначити задачу як завершену. Це потрібно для відстеження особистої продуктивності та збереження історії виконаних задач.

У базі даних кожна задача має логічне поле completed зі значенням false. Коли користувач натискає кнопку Виконано, інтерфейс надсилає PATCH-запит на сервер, який змінює значення поля completed на true.

```
app.patch("/api/tasks/:id/complete", async (req, res) => {
  try {
    const task = await Task.findByIdAndUpdate(req.params.id, { completed: true }, { new: true });
    res.json(task);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

Рисунок 3.10 – Позначення задачі як виконанної

Це дозволяє оперативно змінювати стан задач без повного перезавантаження сторінки. Оновлений список задач відображається на інтерфейсі автоматично, а виконані задачі можуть бути відфільтровані або позначені візуально (наприклад, сірим кольором або перекресленим текстом).

5. Аналіз даних і статистика

Щоб допомогти користувачам стежити за своєю працездатністю, ми проаналізували дані задач. Система рахує, скільки задач було зроблено за день, тиждень чи місяць. Також вона показує середній час виконання задач і дні, коли користувач працював найкраще.

Для цього беремо дані з бази MongoDB, використовуючи фільтри Mongoose:

const completedTasks = await Task.find({ completed: true });

Потім ми обробляємо ці дані та відправляємо їх на фронтенд у вигляді масиву. Там вони перетворюються на стовпчикові, лінійні або кругові діаграми за допомогою бібліотеки Recharts. Це допомагає користувачам швидко зрозуміти, наскільки добре вони працюють, і краще планувати свій час, спираючись на реальні цифри.

6. Режим концентрації

Режим концентрації працює за методом Pomodoro, де потрібно чергувати роботу та відпочинок. Коли користувач вмикає таймер, система прибирає все, що може відволікти (інші кнопки, переходи тощо), щоб зосередитись лише на одній задачі.

Таймер працює через React Hook useEffect, який кожну секунду показує, скільки часу залишилось. Коли час закінчується, система автоматично пропонує зробити перерву або почати новий цикл концентрації.

7. Збереження локальних параметрів

Щоб зробити користування застосунком зручнішим, я додав функцію збереження локальних даних через localStorage. Завдяки цьому користувач не

втрачає свої налаштування після перезавантаження сторінки чи повторного входу до застосунку.

Зокрема, у localStorage можуть зберігатися:

включені фільтри задач (наприклад, всі, зроблені, в роботі);

вибрана тема інтерфейсу (світла або темна);

налаштування таймера в режимі фокусування;

інформація про останній відкритий список задач.

```
}

export default function KeyMetrics({ tasks, period }: KeyMetricsProps ) {
    const now = new Date()

    const filtered = tasks.filter(task => {
        const date = new Date( value: task.completedAt || task.updatedAt || task.dueDate);
        if (period === "day") {
            return date.toDateString() === now.toDateString();
        }
        if (period === "week") {
            const weekAgo = new Date();
            weekAgo.setDate(now.getDate() - 7);
            return date >= weekAgo && date <= now;
        }
        if (period === "month") {
            return (
                date.getMonth() === now.getMonth() &&
                date.getFullYear() === now.getFullYear()
            );
        }
        return true;
    });

    const completed = filtered.filter(t => t.completed);
    const total = filtered.length;
```

Рисунок 3.11 – Компонент відстеження вчасно/ невчасно виконаних завдань

```
export default function FocusStatsCard({sessions, totalTime, avgSession, onReset}) {

    const [hovered, setHovered] = useState( initialState: false);
    return (
        <div
            className="p-6 bg-gradient-to-br from-yellow-50 to-yellow-100
            rounded-2xl shadow-md border border-yellow-200
            transition transform hover:scale-[1.02 hover:shadow-lg"
            onMouseEnter={() => setHovered( value: true)}
            onMouseLeave={() => setHovered( value: false)}
        >
            <h3 className="font-bold mb-2">⏱ Focus Mode Stats</h3>
            <p>Усього: {(totalTime / 3600000).toFixed( fractionDigits: 1)} ч</p>
            <p>Середня сессія: {(avgSession / 60000).toFixed( fractionDigits: 1)} хв</p>
            <p> Сессій: {sessions.length}</p>
        </div>
    )
}
```

Рисунок 3.12 – Компонент фокусування на задачі

3.4 Методика роботи користувача

Додаток «Персональний планувальник задач» розроблено для простої та результативної роботи з завданнями. Він має зрозумілий інтерфейс, яким дуже просто користуватися.

Відразу після запуску, користувач бачить перелік всіх своїх задач. Тут можна додати нове завдання, змінити старе або відмітити його як зроблене. Щоб додати нову задачу, потрібно натиснути кнопку «Додати задачу» і вказати назву, опис, важливість і кінцевий термін. Ця інформація зберігається в базі даних MongoDB через сервер Node.js.

У додатку можна сортувати задачі за статусом: виконані, активні або прострочені. Також, є пошук за словами, щоб швидко відшукати потрібне завдання навіть у великому списку.

Для кращої концентрації є режим фокусування (Focus Mode). У ньому запускається таймер Pomodoro, який дозволяє працювати певний час (наприклад, 25 хвилин), а потім автоматично дає короткий або довгий відпочинок. Під час фокусування зайві елементи зникають з екрану, щоб нічого не відволікало.

Після того, як задача виконана, її можна відмітити як зроблену. Тоді її статус змінюється в базі даних, а в інтерфейсі вона стає, наприклад, сірою або перекресленою. Це дає змогу легко відстежувати хід роботи.

В розділі аналітики є статистика по задачам: скільки зроблено за день, тиждень чи місяць, середня продуктивність і найбільш продуктивні дні. Інформація подана у вигляді графіків і діаграм, зроблених за допомогою Recharts. Налаштування користувача (наприклад, параметри Pomodoro, фільтри, тема оформлення) зберігаються у localStorage, щоб при наступному запуску програми все залишалось так, як було. Це зручно, адже можна одразу продовжити роботу з того місця, де зупинились.

Щоб користувачі не забували про важливі справи, я додав функцію автоматичного перенесення задач. Якщо задача не була виконана до кінця дня, система автоматично перенесе її на наступний день зі статусом Активна.

Оновлення відбувається на сервері за допомогою Node.js. Спеціальна функція запускається в певний час (наприклад, опівночі) і перевіряє, чи є задачі із запізнілим терміном виконання. Для кожної такої задачі оновлюється дата, щоб вона знову була актуальною.

Це дає змогу:

Не втрачати важливі завдання. Кожен день мати актуальний список справ. Менше переносити і планувати вручну. Дотримуватися дисципліни та чіткості в роботі. Це частина автоматичного керування часом, оскільки система сама упорядковує список завдань користувача. Розширений режим фокусування

Крім таймера Pomodoro, у застосунку є розширений режим, який допомагає ще більше зосередитися.

- Блокування зайвих елементів
- Коли Pomodoro запущено, програма переходить у спеціальний режим:
- Приховуються або блокуються всі зайві задачі.
- Вимкнені сповіщення та інші речі, які можуть відволікати.
- На екрані тільки таймер і назва поточної задачі.

Це допомагає не перемикати увагу та створює атмосферу, де ніщо не відволікає.

Лічильник циклів Pomodoro

Система рахує, скільки циклів фокусування завершено. Це корисно для:

Відстеження продуктивності протягом дня. Автоматичного визначення, коли потрібна тривала перерва. Формування звички працювати регулярно.

Алгоритм працює так:

Після кожного сеансу фокусування лічильник збільшується. Після 4 циклів користувачеві пропонується зробити велику перерву. Інформація зберігається у localStorage або базі даних.

Автоматичне створення статистики фокуса

Усі завершені сеанси фокусування автоматично записуються в систему аналітики. Для кожного циклу зберігається:

- Дата і час.
- Тривалість.
- Кількість Pomodoro за день.
- Загальний час у стані фокусування.
- Ця інформація використовується в розділі аналітики, де користувач може подивитися:
- Загальний час фокусування за день, тиждень або місяць.
- Найбільш продуктивні дні.
- Графіки тривалості сесій фокусування.
- Як добре виконуються окремі задачі.
- Отже, розширений режим допомагає краще зосередитися і веде детальну історію продуктивності.

ВИСНОВОК

Під час створення персонального планувальника задач ми розробили сучасний веб-застосунок. Він об'єднує керування задачами, фокусування та аналіз продуктивності. Проєкт включав аналіз потреб, розробку архітектури, бази даних, алгоритмів та інтерфейсу користувача.

Система дає змогу створювати, змінювати, видаляти задачі та відмічати їх як зроблені. Можна ставити дедлайни, визначати пріоритети та категорії. Режим Focus Mode (Pomodoro) допомагає зосередитись, а модуль аналітики показує ефективність за допомогою графіків і статистики.Ми використовували React, Node.js, MongoDB (через Mongoose) та localStorage. Це дало високу швидкість, зручне зберігання даних та адаптивність застосунку. Інтерфейс зроблено в простому стилі для комфортної роботи на різних пристроях. У підсумку, ми розробили планувальник, який допомагає в організації робочого часу та збільшує продуктивність завдяки інструментам для концентрації та аналізу. Цей проєкт можна розширити, додавши функції спільної роботи, інтеграцію з календарями та систему сповіщень в режимі реального часу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Mozilla Developer Network (MDN Web Docs). HTML, CSS, JavaScript Documentation. - https://developer.mozilla.org/
2. React Official Documentation. Building User Interfaces with React. - https://react.dev/
3. Node.js Official Documentation. Server-side JavaScript Runtime. - https://nodejs.org/
4. MongoDB Documentation. NoSQL Database for Modern Applications. - https://www.mongodb.com/docs/
5. Mongoose Documentation. Elegant MongoDB Object Modeling for Node.js. - https://mongoosejs.com/docs/
6. Recharts. Simple Charting Library for React. - https://recharts.org/en-US/
7. Tailwind CSS. Utility-First CSS Framework. - https://tailwindcss.com/
8. W3Schools. Web Development Tutorials and References. - https://www.w3schools.com/
9. Pomodoro Technique Official Site. Francesco Cirillo – The Pomodoro Technique. - https://francescocirillo.com/pages/pomodoro-technique
10. Чмир І. О. Моделювання систем у середовищі UML (Unified Modeling Language) : навч. посібник / І. О. Чмир, М. Ф. Ус ; Черкаськ. акад. менеджменту. – Черкаси : ЧАМ, 2004. – 100 с.
11. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process. — Prentice Hall, 2004. — 736 p.
12. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. — Addison-Wesley, 1994. — 395 p.
13. Burbeck S. Applications Programming in Smalltalk-80: How to use Model–View–Controller (MVC). University of Illinois, 1992.
14. Krug S. Don’t Make Me Think: A Common Sense Approach to Web Usability. — New Riders, 2014. — 216 p.
15. Nielsen J. Usability Engineering. — Morgan Kaufmann, 1994. — 362 p.

16. Wazlawick R. S. Object-Oriented Analysis and Design for Information Systems. — Elsevier, 2014. — 388 p.
17. Mozilla Developer Network (MDN). HTML, CSS, JavaScript Documentation. — URL: https://developer.mozilla.org
18. React Official Documentation. React.js Guide. — URL: https://react.dev/
19. Shneiderman B. Designing the User Interface: Strategies for Effective Human-Computer Interaction. — Pearson, 2016. — 624 p.

ДОДАТОК А

App.js

```
      import {useEffect, useState} from "react";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Layout from "./layouts/Layout";
import Home from "./pages/Home";
import Calendar from "./pages/Calendar.tsx";
import Tasks from "./pages/Tasks";
import Settings from "./pages/Settings";
import dayjs from "dayjs";
import FocusMode from "./pages/FocusMode.tsx";
import ProductivityPage from "./pages/ProductivityPage.tsx";
import TaskHistory from "./components/TaskHistory.tsx";
import {autoMoveUnfinishedTasks} from "./utils/taskUtils";
import NotesPage from "./pages/NotesPage.tsx";
import ExportImportPage from "./pages/ExportImportPage.tsx";
import ResearchPage from "./pages/ResearchPage.tsx";
import {AnimatePresence, motion} from "framer-motion";
import PageWrapper from "./components/motion/PageWrapper.tsx";
import KnowledgeBase from "./pages/KnowledgeBasePage.tsx";

function App() {
const [tasks, setTasks] = useState([]);

const [maxTasksPerDay, setMaxTasksPerDay] = useState(() => {
const saved = localStorage.getItem("maxTasksPerDay");
return saved ? Number(saved) : 3;
})

useEffect(() => {
fetch("http://localhost:5000/api/tasks")
.then(res => res.json())
.then(data =>
{
const updated = autoMoveUnfinishedTasks(data);
setTasks(updated)
})
},[])
```

```
const reorderTasks = (newOrder) => {
setTasks(newOrder);
};

const suggestDate = () => {
const taskCountByDate = {};

tasks.forEach((task) => {
taskCountByDate[task.dueDate] = (taskCountByDate[task.dueDate] || 0) + 1;
});

for (let i = 0; i < 7; i++) {
const date= dayjs().add(i, "day").format("YYYY-MM-DD")
if ((taskCountByDate[date] || 0) < 3) return date;
}

return dayjs().add(1, "day").format("YYYY-MM-DD");
}

const getBestDay = () => {
const taskCountByDate = {};

tasks.forEach(task => {
taskCountByDate[task.dueDate] = (taskCountByDate[task.dueDate] || 0) + 1;
});

for (let i = 0; i < 7; i++) {
const date = dayjs().add(i, "day").format("YYYY-MM-DD");
if ((taskCountByDate[date] || 0) < maxTasksPerDay) {
return date;
}
}

return dayjs().add(1, "day").format("YYYY-MM-DD");

}

const addTask = async ({ title, dueDate, priority = "medium", tags = [] })
=> {
const finalDueDate = dueDate || suggestDate();
```

```
const countForDate = tasks.filter((t) => t.dueDate === finalDueDate).length;

if (countForDate >= maxTasksPerDay) {
alert(`✕ Ви вже досягли ліміту задач (${maxTasksPerDay}) на
${finalDueDate}`);
return;
}

const newTask = {
id: Date.now(),
title,
completed: false,
dueDate: finalDueDate,
priority,
tags,
};

const res = await fetch("http://localhost:5000/api/tasks", {
method: "POST",



     headers: {"Content-Type" : "application/json"},
body: JSON.stringify(newTask)
})

const savedTask = await res.json();
setTasks(prev => [savedTask, ...prev]);
};

const toggleTask =async (id) => {
const task = tasks.find(t => t._id === id || t.id === id);
if (!task) return;
const updatedTask = { ...task, completed: !task.completed };

await fetch(`http://localhost:5000/api/tasks/${id}`, {
method: "PUT",
headers: {"Content-Type": "application/json"},
body: JSON.stringify({ completed: updatedTask.completed }),
});
```

```
setTasks(prev => prev.map(t => (t._id === id || t.id === id) ? updatedTask :
t));
};



const deleteTask = async (id) => {
try {
await fetch(`http://localhost:5000/api/tasks/${id}`, {
method: "DELETE",
});
setTasks(tasks.filter(task => task._id !== id));
} catch (error) {
console.error("Ошибка при удалении задачи:", error);
}
};

const bestDay = getBestDay();

return (
<BrowserRouter>
<AnimatePresence mode="wait">
<Routes>
<Route path="/" element={<Layout />}>
<Route
index
element={
<PageWrapper>
<Home
tasks={tasks}
addTask={addTask}
toggleTask={toggleTask}
deleteTask={deleteTask}
bestDay={bestDay}
reorderTasks={reorderTasks}
/>
</PageWrapper>

}
/>
```

```
<Route path="calendar" element={
<PageWrapper>
<Calendar tasks={tasks} onAddTask={addTask} />
</PageWrapper>} />

<Route
path="tasks"
element={
<PageWrapper>
<Tasks
tasks={tasks}
addTask={addTask}
toggleTask={toggleTask}
deleteTask={deleteTask}
/>);
}

      </PageWrapper>
}
/>
<Route path="settings" element={
<PageWrapper>
<Settings
maxTasksPerDay={maxTasksPerDay}
setMaxTasksPerDay={setMaxTasksPerDay}
/>
</PageWrapper>} />
<Route
path="focus-mode"
element={
<PageWrapper>
<FocusMode
tasks={tasks}
setTasks={setTasks}
toggleTask={toggleTask}
deleteTask={deleteTask}
/>
</PageWrapper>
}
/>
<Route
```

```
path="productivity"
element={
<PageWrapper>
<ProductivityPage tasks={tasks} />
</PageWrapper>
}
/>

<Route
path="archive"
element={
<PageWrapper>
<TaskHistory tasks={tasks} />
</PageWrapper>
}
/>
<Route
path="notes"
element={
<PageWrapper>
<NotesPage />
</PageWrapper>
}
/>

</Route>

<Route
path="export-import"
element={
<PageWrapper>
<ExportImportPage
tasks={tasks}
setTasks={setTasks}
/>
</PageWrapper>}
/>
<Route
path="research"
element={
<PageWrapper>
```

```
<ResearchPage
/>
</PageWrapper>}
/>

<Route
path="knowledge"
element={
<PageWrapper>
<KnowledgeBase
/>
</PageWrapper>}
/>

</Routes>
</AnimatePresence>
</BrowserRouter>


export default App;
```

PomodoroTimer.tsx

```
import {useEffect, useRef, useState} from "react";
import {useMethodTracking} from "../hooks/useMethodTracking.tsx";
import PomodoroProgress from "./PomodoroProgress.tsx";
import PomodoroSettings from "./PomodoroSettings.tsx";
import PomodoroControls from "./PomodoroControls.tsx";

export default function PomodoroTimer({ task, onStop, onCompleteTask }) {

const defaultSettings = {
focus: 25,
shortBreak: 5,
longBreak: 15,
cyclesBeforeLongBreak: 4,
};

const [settings, setSettings] = useState(() => {
return { ...defaultSettings,
...JSON.parse(localStorage.getItem("pomodoroSettings") || "{}") };
});
```

```
const [seconds, setSeconds] = useState(settings.focus * 60);
const [mode, setMode] = useState<"focus" | "shortBreak" |
"longBreak">("focus");
const [isRunning, setIsRunning] = useState(false);
const [cycles, setCycles] = useState(0);
const intervalRef = useRef(null);
const { log } = useMethodTracking();
const [animate, setAnimate] = useState(false);


useEffect(() => clearInterval(intervalRef.current), []);

useEffect(() => {
if (mode === "focus") setSeconds(settings.focus * 60);
if (mode === "shortBreak") setSeconds(settings.shortBreak * 60);
if (mode === "longBreak") setSeconds(settings.longBreak * 60);
}, [settings, mode]);


      const getModeTime = (m) => {
if (m === "focus") return settings.focus * 60;
if (m === "shortBreak") return settings.shortBreak * 60;
if (m === "longBreak") return settings.longBreak * 60;
return 0;
};

const start = () => {
if (isRunning) return;
setIsRunning(true);
logEvent("start");

};

const pause = () => {
if (!isRunning) return;
setIsRunning(false);
logEvent("pause");
};

const reset = () => {
setIsRunning(false);
```

```
setSeconds(getModeTime(mode));
logEvent("reset");
};

const interrupt = () => {
setIsRunning(false);
logEvent("interrupt");
}

useEffect(() => {
if (!isRunning) return;

intervalRef.current = setInterval(() => {
setSeconds((prev) => {
if (prev <= 1) {
clearInterval(intervalRef.current!);
handleFinish();
return 0;
}
return prev - 1;
});
}, 1000);

return () => clearInterval(intervalRef.current!);
}, [isRunning]);

const handleFinish = async () => {
setIsRunning(false);
logEvent("complete", mode);

const audio = new Audio("/notify.mp3");
audio.play().catch(() => {});

setAnimate(true);
setTimeout(() => setAnimate(false), 1500);

if (mode === "focus") {
const newCycles = cycles + 1;
setCycles(newCycles);

if (task) {
```

```
await fetch(`http://localhost:5000/api/tasks/${task._id}`, {
method: "PUT",
headers: { "Content-Type": "application/json" },
body: JSON.stringify({ pomodoroCycles: newCycles }),
});
}

const nextMode =
newCycles % settings.cyclesBeforeLongBreak === 0 ? "longBreak" :
"shortBreak";
setMode(nextMode);
setSeconds(getModeTime(nextMode));
} else {
setMode("focus");
setSeconds(getModeTime("focus"));
}
};

const logEvent = (event: string, extraMode: string | null = null) => {
if (!task) return;
log({
taskId: task._id,
method: task.method,
event,
mode: extraMode || mode,
});
};

const finishTask = () => {
setIsRunning(false);
clearInterval(intervalRef.current!);
logEvent("complete", mode);

if (onCompleteTask && task) {
onCompleteTask(task._id);
}
onStop?.(task);
};

const mm = String(Math.floor(seconds / 60)).padStart(2, "0");
const ss = String(seconds % 60).padStart(2, "0");
```

```
const totalTime = getModeTime(mode);
const progress = Math.min(((totalTime - seconds) / totalTime) * 100, 100);

const bgColors = {
focus: isRunning ? "bg-red-200" : "bg-red-300",
shortBreak: isRunning ? "bg-green-200" : "bg-green-300",
longBreak: isRunning ? "bg-blue-200" : "bg-blue-300",
};


return (
<div
className={`p-6 rounded-2xl shadow-lg flex flex-col items-center gap-4
transition-colors duration-700 ${bgColors[mode]}`}
>
<h2 className="text-lg font-bold text-center">
Задача: <span className="text-blue-600">{task?.title}</span>
<PomodoroSettings settings={settings} setSettings={setSettings} />
</h2>

<h2 className="text-xl font-bold capitalize">
{mode === "focus"
      ? "Фокус"
: mode === "shortBreak"
? "Перерва"
: "Довга перерва"}
</h2>

<div className="relative w-48 h-48 flex items-center justify-center">
<svg className="w-48 h-48 transform -rotate-90">
<circle
cx="96"
cy="96"
r="90"
stroke="currentColor"
strokeWidth="12"
fill="none"
className="text-gray-300"
/>
<circle
```

```
cx="96"
cy="96"
r="90"
stroke="currentColor"
strokeWidth="12"
fill="none"
className="text-green-500 transition-all duration-500"
strokeDasharray={2 * Math.PI * 90}
strokeDashoffset={
2 * Math.PI * 90 * (1 - progress / 100)
}
/>
</svg>
<div className="absolute text-3xl font-bold">
{mm}:{ss}
</div>
</div>

<PomodoroProgress
cycles={cycles}
cyclesBeforeLongBreak={settings.cyclesBeforeLongBreak}
/>

<p className="text-sm text-gray-700">
Завершено циклів: {cycles} з {settings.cyclesBeforeLongBreak}
</p>

<PomodoroControls
start={start}
pause={pause}
reset={reset}
interrupt={interrupt}
finishTask={finishTask}
onStop={onStop}
/>
</div>
);
```

KeyMetrics.tsx

```
import {str} from "ajv";

interface KeyMetricsProps {
    tasks: any[];
    period: "day" | "week" | "month";
}

export default function KeyMetrics({ tasks, period }: KeyMetricsProps ) {
    const now = new Date()

    const filtered = tasks.filter(task => {
        const date = new Date(task.completedAt || task.updatedAt ||
task.dueDate);
        if (period === "day") {
            return date.toDateString() === now.toDateString();
        }
        if (period === "week") {
            const weekAgo = new Date();
            weekAgo.setDate(now.getDate() - 7);
            return date >= weekAgo && date <= now;
        }
        if (period === "month") {
            return (
                date.getMonth() === now.getMonth() &&
                date.getFullYear() === now.getFullYear()
            );
        }
        return true;
    });

    const completed = filtered.filter(t => t.completed);
    const total = filtered.length;

    const completionRate = total > 0 ? Math.round((completed.length / total)
* 100) : 0;
    const onTime = completed.filter(t => t.dueDate && new
Date(t.completedAt) <= new Date(t.dueDate));

    const onTimeRate = completed.length > 0 ? Math.round((onTime.length /
completed.length) * 100) : 0;
    const avgTime = completed.length > 0
    ? Math.round(
        completed.reduce((acc, t) => {
            const created = new Date(t.createdAt).getTime();
            const finished = new Date(t.completedAt).getTime();
            return acc + (finished - created);
        }, 0) /
            completed.length /
            (1000 * 60 * 60)
        )
        : 0;

    const dates = Array.from(
        new Set(completed.map(t => new Date(t.completedAt).toDateString()))
```

```
    ).sort((a,b) => new Date(a).getTime() - new Date(b).getTime());

    let streak = 0;
    let maxStreak = 0;
    for (let i = 0; i < dates.length; i++ ){
        if (i === 0) {
            streak = 1;
        } else {
            const prev = new Date(dates[i - 1]);
            const curr = new Date(dates[i]);
            const diff = (curr.getTime() - prev.getTime()) / (1000 * 60 * 60
* 24);
            if (diff === 1) {
                streak++;
            }else {
                maxStreak = Math.max(maxStreak, streak);
                streak = 1;
            }
        }
    }
    maxStreak = Math.max(maxStreak, streak);

    let prevFiltered: any[] = [];
    if (period === "week") {
        const prevWeekStart = new Date();
        prevWeekStart.setDate(now.getDate() - 14);
        const prevWeekEnd = new Date();
        prevWeekEnd.setDate(now.getDate() - 7);

        prevFiltered = tasks.filter(t => {
            const date = new Date(t.completedAt || t.updatedAt ||
t.dueDate);
            return date >= prevWeekStart && date < prevWeekEnd;
        });
    }

    const prevCompleted = prevFiltered.filter(t => t.completed).length;
    const diff = prevCompleted > 0
    ? Math.round(((completed.length - prevCompleted) / prevCompleted) * 100)
        : 0;

    return (
        <div className="grid grid-cols-2 md:grid-cols-4 gap-2 my-6">
            <div className="p-4 bg-blue-100 rounded-2xl shadow-md text-
center flex flex-col justify-center items-center">
                <div className="text-lg font-bold">{completionRate}%</div>
                <div className="text-sm text-gray-600">Completion Rate</div>
            </div>
            <div className="p-4 bg-yellow-100 rounded-2xl shadow-md text-
center flex flex-col justify-center items-center">
                <div className="text-lg font-bold">{onTimeRate}%</div>
                <div className="text-sm text-gray-600">Вчасно виконані</div>
            </div>
            <div className="p-4 bg-purple-100 rounded-2xl shadow-md text-
center flex flex-col justify-center items-center">
                <div className="text-lg font-bold">{avgTime} год</div>
                <div className="text-sm text-gray-600">Середній час
```

```
виконання</div>
            </div>
            <div className="p-4 bg-blue-100 rounded-2xl shadow-md text-
center flex flex-col justify-center items-center">
                <div className="text-lg font-bold">{maxStreak} дн.</div>
                <div className="text-sm text-gray-600">Streak</div>
            </div>
            {period === "week" && (
                <div className="col-span-2 md:col-span-4 p-4 bg-pink-100
rounded-lg shadow text-center">
                    <div className="text-lg font-bold">
                        {diff >= 0 ? `+${diff}%` : `${diff}%`}
                    </div>
                    <div className="text-sm text-gray-600">Порівняно з
минулим тижнем</div>
                </div>
            )}
        </div>
    );
}
```

TaskHistory.tsx

```
import {useMemo, useState} from "react";
import {
  ResponsiveContainer,
  BarChart,
  Bar,
  LineChart,
  Line,
  XAxis,
  YAxis,
  CartesianGrid,
  Tooltip
} from "recharts";
import {Link} from "react-router-dom";

export default function TaskHistory({ tasks}) {
  const completedTasks = tasks.filter(task => task.completed);
  const [filter, setFilter] = useState("all");
  const now = new Date();

  const getPriorityIcon = (priority: string) => {
    switch ( priority ) {
      case "high" : return "🔥";
      case "medium": return "⚡";
      case "low": return "💤";
      default: return "📌";
    }
  }

  const getStatus = (task) => {
```

```
    if (!task.completed) return "□ У процесі";
    return "✓ Завершено";
}

const chartData = useMemo(() => {
    const grouped: Record<string, number[]> = {};


    completedTasks.forEach(task => {
        const end = task.completed ? new Date(task.completedAt) : null
        if (!end) return;
        const day = end.toLocaleDateString("uk-UA", { day: "2-digit", month: "2-digit" });
        const start = new Date(task.startedAt || task.createdAt);
        const duration = end.getTime() - start.getTime();

        if (!grouped[day]) grouped[day] = [];
        grouped[day].push(duration / 60000);
    });

    return Object.keys(grouped).map(day => {
        const durations = grouped[day];
        const avg = durations.reduce((a, b) => a + b, 0) / durations.length;
        return {
            day,
            count: durations.length,
            avg: parseFloat(avg.toFixed(1))
        };
    });
}, [completedTasks]);

const filteredTasks = completedTasks.filter(task => {
    if (!task.completedAt) return false;
    const completedDate = new Date(task.completedAt);

    switch (filter) {
        case "today": {
            return completedDate.toDateString() === now.toDateString()
        }
        case "week": {
            const weekAgo = new Date();
            weekAgo.setDate(now.getDate() - 7);
            return completedDate >= weekAgo && completedDate <= now;
        }
        case "month": {
            return (
                completedDate.getMonth() === now.getMonth() &&
                    completedDate.getFullYear() === now.getFullYear()
            );
        }
        default:
```

```
            return true;
        }
    }
  })

  return (
    <div className="mt-10">
      <h2 className="text-xl font-semibold mb-4">📜 История выполнения</h2>

      <select
        value={filter}
        onChange={(e) => setFilter(e.target.value)}
        className="border rounded px-2 py-1"
      >
        <option value="all">Все</option>
        <option value="today">Сьогодні</option>
        <option value="week">Ця неділя</option>
        <option value="month">Цей місяць</option>
      </select>
      <div className="overflow-x-auto">
        <table className="table-auto w-full border border-gray-300 mt-5 ">
          <thead className="bg-gray-100">
            <tr>
              <th className="px-4 py-2 border">Назва</th>
              <th className="px-4 py-2 border">Пріоритет</th>
              <th className="px-4 py-2 border">Дата завершення</th>
              <th className="px-4 py-2 border">Час виконання</th>
              <th className="px-4 py-2 border">Статус</th>
            </tr>
          </thead>
          <tbody>
          {filteredTasks.map(task => {
            const start = new Date(task.startedAt || task.createdAt);
            const end = task.completedAt ? new Date(task.completedAt) : null;

            if (!end || isNaN(end.getTime())) {
              return (
                <tr key={task._id || task.id}>
                  <td className="px-4 py-2 border">{task.title}</td>
                  <td className="px-4 py-2 border">{task.priority}</td>
                  <td className="px-4 py-2 border">Не завершено</td>
                  <td className="px-4 py-2 border">-</td>
                </tr>
              )
            }

            const durationMs = end.getTime() - start.getTime();
            const hours = Math.floor(durationMs / (1000 * 60 * 60));
            const minutes = Math.floor((durationMs / (1000 * 60)) % 60);
```

```
            return (
              <tr key={task.id}>
                <td className="px-4 py-2 border">{task.title}</td>
                <td className="px-4 py-2 border">{getPriorityIcon(task.priority)}</td>
                <td className="px-4 py-2 border">{end.toLocaleString()}</td>
                <td className="px-4 py-2 border">
                  {hours > 0 ? `${hours} ч `: ""}
                  {minutes} мин
                </td>
                <td className="px-4 py-2 border">{getStatus(task)}</td>

              </tr>
            )
          })}
        </tbody>
      </table>
    </div>

    <div className="mt-10 grid grid-cols-1 md:grid-cols-2 gap-6">
      <div className="p-4 bg-white rounded-xl shadow">
        <h3 className="font-bold mb-2">📊 Заверешені задачі по днях</h3>
        <ResponsiveContainer>
          <BarChart data={chartData}>
            <CartesianGrid strokeDasharray="3 3"/>
            <XAxis dataKey="day"/>
            <YAxis/>
            <Tooltip/>
            <Bar dataKey="count" fill="#4f46e5" radius={[6, 6, 0, 0]} />
          </BarChart>
        </ResponsiveContainer>
      </div>

      <div className="p-4 bg-white rounded-xl shadow">
        <h3 className="font-bold mb-2">⏱ Середній час виконання </h3>
        <ResponsiveContainer width="100%" height={250}>
          <LineChart data={chartData}>
            <CartesianGrid strokeDasharray="3 3"/>
            <XAxis dataKey="day"/>
            <YAxis/>
            <Tooltip/>
            <Line type="monotone" dataKey="avg" stroke="#16a34a" strokeWidth={2} dot/>
          </LineChart>
        </ResponsiveContainer>
      </div>
    </div>

  </div>
);
```