МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ОСОБИСТИМИ ПРОЕКТАМИ З ВИКОРИСТАННЯМ КАНБАН-ДОШКИ ТА ТАЙМ-ТРЕКІНГУ»**

Виконав: студент 4 курсу, групи 6.1222
спеціальності 122 комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми комп'ютерні науки
(назва освітньої програми)
Р. Р. Ігнатченко
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, к.т.н. Добровольський Г.А.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Мильцев О. М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 комп'ютерні науки
(шифр і назва)

Освітня програма комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри комп'ютерних наук, к.пед.н., доцент

_____ Пшенична О.С.
(підпис)

" 08 " грудня 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Ігнатченку Ростиславу Романовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Проектування та реалізація вебзастосунку для управління особистими проектами з використанням канбан-дошки та тайм-трекінгу

керівник роботи Добровольський Геннадій Анатолійович, к.т.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 04.06.2026

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Теоретичні основи розробки вебзастосунків для управління проєктами та тайм- менеджменту
2. Проєктування вебзастосунку для управління особистими проєктами
3. Розробка та тестування вебзастосунку для управління особистими проєктами

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 05.12.2025

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	17.02.2026	
2.	Збір вихідних даних.	13.03.2026	
3.	Обробка методичних та теоретичних	20.03.2026	
	джерел.		
4.	Розробка першого та другого розділу.	05.04.2026	
5.	Розробка третього розділу.	17.05.2026	
6.	Оформлення та нормоконтроль	04.06.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	19.06.2026	

Студент _____ Р. Р. Ігнатченко
(підпис) (ініціали та прізвище)

Керівник роботи _____ Г. А. Добровольський
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Проектування та реалізація вебзастосунку для управління особистими проектами з використанням канбан-дошки та тайм-трекінгу»: 68 с., 16 рис., 9 табл., 23 джерел, 2 додатків.

БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, КАНБАН-ДОШКА, ТАЙМ-МЕНЕДЖМЕНТ, ТАЙМ-ТРЕКІНГ, УПРАВЛІННЯ ПРОЄКТАМИ, SERVER ACTIONS, SQLITE, REACT, USER AUTHENTICATION, UI/UX DESIGN.

Об’єктом дослідження є процеси управління особистими проєктами та організації часу користувача в цифровому середовищі.

Предметом дослідження є методи та програмні засоби розробки вебзастосунків для управління завданнями з використанням канбан-дошки та тайм-трекінгу.

Метою кваліфікаційної роботи є проєктування та реалізація вебзастосунку для управління особистими проєктами з використанням канбан-дошки та тайм-трекінгу.

Методи дослідження: аналіз літератури та існуючих рішень, порівняльний і системний аналіз, методи проєктування програмного забезпечення, об’єктно-орієнтоване програмування та тестування системи.

У роботі проаналізовано сучасні системи управління проєктами та інструменти тайм-менеджменту, визначено вимоги до вебзастосунку. Спроєктовано архітектуру системи, базу даних і користувацький інтерфейс. Реалізовано основні функціональні модулі, зокрема автентифікацію користувачів, канбан-дошку з підтримкою drag-and-drop, систему тайм-трекінгу та збереження історії витраченого часу. Проведене тестування підтвердило коректність роботи системи, її безпеку та продуктивність.

SUMMARY

Bachelor's Thesis «Design and Implementation of a Web Application for Managing Personal Projects Using a Kanban Board and Time Tracking»: 68 pages, 16 figures, 9 tables, 23 sources, 2 appendices.

DATABASE, WEB APPLICATION, KANBAN BOARD, TIME MANAGEMENT, TIME TRACKING, PROJECT MANAGEMENT, SERVER ACTIONS, SQLITE, REACT, USER AUTHENTICATION, UI/UX DESIGN.

The object of the research is the processes of personal project management and user time organization in a digital environment.

The subject of the research is the methods and software tools for developing web applications for task management using a Kanban board and time tracking.

The aim of the thesis is to design and implement a web application for personal project management using a Kanban board and time tracking functionality.

Research methods include literature analysis and review of existing solutions, comparative and system analysis, software design methods, object-oriented programming, and system testing.

The work analyzes modern project management systems and time management tools and defines requirements for the web application. The system architecture, database, and user interface were designed. The main functional modules were implemented, including user authentication, a Kanban board with drag-and-drop support, a time tracking system, and time history storage. Testing confirmed the correctness, security, and performance of the system.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2

Реферат 4

Summary 5

Вступ..... 7

1 Теоретичні основи розробки вебзастосунків для управління проєктами та тайм-менеджменту 9

1.1 Поняття та особливості систем управління особистими проєктами ... 9

1.2 Аналіз сучасних вебзастосунків для тайм-менеджменту та канбан-систем…. 11

1.3 Визначення вимог до вебзастосунку для управління особистими проєктами з канбан-дошкою та тайм-трекінгом..... 15

2 Проєктування вебзастосунку для управління особистими проєктами..... 18

2.1 Аналіз предметної області та проєктування архітектури системи 18

2.2 Проєктування бази даних та основних функціональних модулів 23

2.3 Проєктування користувацького інтерфейсу 30

2.4 Обґрунтування вибору технологій розробки 34

3 Розробка та тестування вебзастосунку для управління особистими проєктами 37

3.1 Реалізація вебзастосунку..... 37

3.2 Розробка функціоналу канбан-дошки та тайм-трекінгу 44

3.3 Тестування системи та аналіз результатів роботи..... 47

3.4 Інструкція користувача..... 49

Висновки 51

Перелік посилань..... 53

Додаток А Лістинг реалізації вебзастосунку 56

Додаток Б Інтерфейс вебзатосунку 65

ВСТУП

Сучасний розвиток інформаційних технологій та широке впровадження вебзастосунків у повсякденну діяльність користувачів суттєво змінили підходи до організації роботи, планування часу та управління особистими проєктами. Умови високої інформаційної навантаженості, багатозадачності та постійного дефіциту часу зумовлюють потребу в ефективних цифрових інструментах, що дозволяють структурувати завдання, контролювати їх виконання та підвищувати продуктивність. Особливої актуальності набувають системи управління особистими проєктами, які поєднують методології візуального планування (зокрема канбан-дошки) та інструменти контролю витрат часу (тайм-трекінг). Такі підходи активно використовуються у сучасних IT-компаніях та сервісах організації роботи, зокрема у рішеннях типу Trello, Jira, Asana, Toggl та інших. Однак більшість існуючих платформ орієнтовані або на командну роботу, або мають надмірну складність для індивідуального користувача, що створює потребу у розробці більш простих, адаптованих до персонального використання вебзастосунків. Актуальність роботи зумовлена необхідністю створення зручного, інтуїтивного та функціонального вебзастосунку для управління особистими проєктами, який поєднує канбан-підхід із можливістю відстеження часу виконання завдань. Такий інструмент дозволяє підвищити рівень самоорганізації користувача, оптимізувати розподіл часу між завданнями та забезпечити прозорість процесу виконання особистих проєктів.

Метою кваліфікаційної роботи є проєктування та реалізація вебзастосунку для управління особистими проєктами з використанням канбан-дошки та тайм-трекінгу.

Для досягнення поставленої мети визначено такі завдання:

- проаналізувати сучасні підходи та існуючі вебзастосунки для управління проєктами та тайм-менеджменту;

- визначити функціональні та нефункціональні вимоги до системи;
- спроєктувати архітектуру вебзастосунку та структуру бази даних;
- розробити користувацький інтерфейс системи з використанням принципів UX/UI-дизайну;
- обґрунтувати вибір технологій реалізації вебзастосунку;
- реалізувати основний функціонал системи, включаючи канбан-дошку та тайм-трекінг;
- провести тестування розробленого програмного продукту та проаналізувати результати його роботи.

Об’єктом дослідження є процеси управління особистими проєктами та організації часу користувача в цифровому середовищі.

Предметом дослідження є методи та програмні засоби розробки вебзастосунків для управління завданнями з використанням канбан-дошки та тайм-трекінгу.

Методи дослідження: аналіз літератури та існуючих рішень, порівняльний і системний аналіз, методи проєктування програмного забезпечення, об’єктно-орієнтоване програмування та тестування системи.

Практичне значення отриманих результатів полягає у створенні готового вебзастосунку, який може бути використаний для планування та контролю особистих завдань, управління навчальними або робочими проєктами, а також для підвищення продуктивності користувачів у щоденній діяльності.

Отримані результати можуть бути застосовані при розробці подібних систем управління завданнями, у навчальному процесі при вивченні вебтехнологій, а також як основа для подальшого удосконалення персональних інструментів тайм-менеджменту з використанням аналітики продуктивності та елементів штучного інтелекту.

1 ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ВЕБЗАСТОСУНКІВ ДЛЯ УПРАВЛІННЯ ПРОЄКТАМИ ТА ТАЙМ-МЕНЕДЖМЕНТУ

1.1 Поняття та особливості систем управління особистими проєктами

Система управління особистими проєктами (англ. Personal Project Management System) являє собою сукупність програмних засобів, методологій та організаційних підходів, що призначені для планування, організації та контролю виконання індивідуальних завдань і досягнення особистих цілей користувача. Такі системи є підвидом загальних систем управління проєктами, проте орієнтовані не на командну роботу, а на індивідуального користувача, який самостійно виступає і виконавцем, і менеджером проєкту.

У загальному розумінні особистий проєкт можна визначити як комплекс взаємопов’язаних дій, спрямованих на досягнення конкретної мети в межах обмеженого часу та ресурсів. До таких проєктів належать навчальні цілі, професійний розвиток, самоосвіта, фінансове планування або реалізація особистих задумів. Управління ними передбачає координацію завдань, контроль термінів та оптимізацію використання часу [1].

З точки зору системного підходу, система управління проєктами розглядається як цілісна структура, що складається з об’єкта управління (самого проєкту), суб’єкта управління (користувача), а також інструментів планування, контролю та аналізу. Її основними компонентами є цілі, функції, методи управління та інформаційне забезпечення процесів.

Особисті системи управління проєктами мають низку характерних особливостей. Насамперед вони відрізняються простотою та орієнтацією на швидке використання без складного навчання. На відміну від корпоративних рішень, такі системи не потребують складної ієрархії ролей чи розподілу відповідальності, що робить їх більш доступними для індивідуальних користувачів [2].

Важливою особливістю є баланс між простотою та функціональністю. Ефективна система повинна забезпечувати швидке створення завдань, їх структурування, контроль виконання та зручну навігацію між проєктами. При цьому надмірна складність інтерфейсу або функціоналу може знижувати продуктивність користувача, тому мінімалізм є одним із ключових принципів таких систем.

Ще однією особливістю є індивідуалізація. Кожен користувач формує власну модель управління завданнями відповідно до своїх звичок, стилю роботи та цілей. Тому сучасні системи часто передбачають можливість налаштування робочих процесів, пріоритетів та візуальних інструментів планування [2].

Серед ключових функціональних елементів таких систем можна виділити управління завданнями, планування часу, візуалізацію прогресу та ведення нотаток. Особливо важливими є методи візуального управління, зокрема канбан-дошки, які дозволяють відображати етапи виконання завдань у зручній та зрозумілій формі. Також широко застосовується тайм-трекінг, який забезпечує контроль витрат часу на виконання окремих задач.

Окремо слід відзначити сучасні тенденції розвитку систем управління особистими проєктами. У 2026 році спостерігається активна інтеграція технологій штучного інтелекту, які дозволяють автоматизувати планування, прогнозувати терміни виконання завдань та надавати персоналізовані рекомендації користувачу. Це підвищує ефективність використання таких систем та зменшує когнітивне навантаження [3].

До поширених інструментів управління особистими проєктами належать Notion, Trello, Todoist, ClickUp та інші платформи, що поєднують функції планування, організації завдань та аналітики продуктивності. Вони демонструють тенденцію до переходу від простих списків задач до комплексних екосистем управління особистою ефективністю [3].

Таким чином, системи управління особистими проєктами є важливим інструментом підвищення ефективності індивідуальної діяльності. Вони дозволяють структурувати завдання, оптимізувати використання часу та

забезпечують прозорість процесу досягнення особистих цілей. Їх подальший розвиток пов’язаний із впровадженням інтелектуальних алгоритмів та розширенням можливостей персоналізації.

1.2 Аналіз сучасних вебзастосунків для тайм-менеджменту та канбан-систем

Методологія канбан бере своє коріння у виробничій системі японської корпорації Toyota, розробленій інженером Тайіті Оно наприкінці 1940-х років. Спочатку метод задумувався як спосіб регулювання потоку деталей та матеріалів відповідно до попиту – для усунення надвиробництва та мінімізації відходів [4]. У сфері програмного забезпечення та розумової праці канбан трансформувався у систему візуального управління процесами, яка допомагає приймати рішення щодо того, що, коли і в якій кількості виробляти [5].

У 2004 році Девід Дж. Андерсон розробив канбан-систему для інженерної групи Microsoft XIT Sustaining Engineering Group, що стало першим значущим застосуванням принципів канбан у сфері інтелектуальної праці. Це адаптування зберегло ключові принципи візуалізації, обмежень незавершеної роботи (WIP) та управління потоком, але пристосувало їх до невиробничих контекстів [5].

У 2003 році Мері та Том Поппендік опублікували книгу «Lean Software Development: An Agile Toolkit», яка перенесла принципи ощадливого виробництва Toyota у сферу розробки програмного забезпечення. Книга акцентувала увагу на усуненні відходів, картографуванні потоку цінності та системах витягування [4]. Ці публікації заклали теоретичне підґрунтя для сучасних цифрових канбан-систем.

Найпростіша канбан-дошка містить три колонки: «Зробити», «У роботі», «Готово». Кожен проєкт поділено на різні етапи, і скільки етапів – стільки й стовпців у канбан-дошці. Такі дошки можна використовувати і для особистого тайм-менеджменту, і для проєктного планування [6].

Було виконано огляд провідних вебзастосунків:

Trello – один із найпоширеніших інструментів для реалізації канбан-підходу у вебсередовищі. Дошка представляє проєкт, списки відповідають етапам робочого процесу, а картки є завданнями [7]. Серед ключових функцій – дошки та картки для візуалізації робочого процесу з можливістю перетягування карток між списками, чеклісти для деталізації завдань, дати виконання з нагадуваннями, кольорові мітки для категоризації завдань, прикріплення файлів та коментарі [8].

Trello пропонує п’ять інструментів управління проєктами: канбан-дошки, часові шкали проєктів, календарі, дашборди та таблиці. Розширена функціональність потребує підключення платних плагінів (Power-Ups) [7]. Суттєве обмеження платформи – відсутність вбудованого тайм-трекінгу, розширеної звітності та підтримки залежностей між завданнями [9].

Jira від Atlassian позиціонується як рішення для команд, що працюють за Agile-методологіями. Платформа надає нативні Scrum- та Kanban-дошки, планування спринтів, управління беклогом, відстеження швидкості роботи команди, діаграми burndown та CFD-діаграми [10]. Серед сильних сторін – потужна система відстеження помилок, розширені дорожні карти та комплексні інструменти звітності й аналітики для моніторингу продуктивності команди [10].

Для індивідуального або малого проєктного використання Jira може виявитися надлишково складним інструментом через значну криву навчання та орієнтованість переважно на командну роботу в IT-сфері.

Asana має чистий та інтуїтивний інтерфейс, ідеальний для команд, що покладаються на списки, дедлайни та індикатори прогресу. Платформа легко масштабується зі зростанням проєктів і пропонує численні інтеграції [11]. Водночас Asana не включає вбудованого тайм-трекінгу, що є суттєвим обмеженням для команд, яким необхідна ця функціональність [11].

Monday.com дотримується візуального підходу: налаштовувані дошки використовують кольорові статуси та віджети, що дозволяє легко виявляти вузькі місця. Платформа пропонує канбан-дошки, часові шкали та календарний

вигляд з функцією перетягування. Тайм-трекінг доступний у тарифному плані Pro і дозволяє командам контролювати витрачений на завдання час [11].

ClickUp позиціонується як «все в одному» для управління проєктами. Він дозволяє переглядати проєкти у вигляді списку, канбан-дошки, діаграми Ганта, календаря, часової шкали та інших форматів [12]. Платформа має вбудований тайм-трекінг, безпосередньо інтегрований із завданнями, а також підтримує розширену автоматизацію на основі тригерів [13].

Notion є комбінованим інструментом для нотаток, баз даних та управління проєктами. Кожен елемент сторінки можна налаштовувати; підтримується перетягування, вставлення зображень та відео. Нативну автоматизацію запроваджено у 2024 році, проте при роботі з більш ніж 5000 записів спостерігаються помітні уповільнення продуктивності [12]. Тайм-трекінг реалізується лише через сторонні інтеграції.

Для систематизації результатів аналізу складено порівняльну таблицю ключових характеристик розглянутих платформ (таблиця 1.1).

Таблиця 1.1 – Порівняльна характеристика вебзастосунків для управління проєктами

Платформа	**Канбан-дошка**	**Вбудований тайм-трекінг**	**Безкоштовний план**	**Орієнтація**
Trello	✓	✗	✓	Малі команди / особисте використання
Jira	✓	✗ (плагін)	✓ (до 10 ос.)	Agile-команди розробників
Asana	✓	✗	✓	Командні проєкти
Monday.com	✓	✓ (Pro)	✓ (до 2 ос.)	Візуальне управління
ClickUp	✓	✓	✓	Команди будь-якого розміру
Notion	✓	✗	✓ (1 ос.)	Нотатки + БД + проєкти

Аналіз провідних вебзастосунків виявив низку спільних недоліків, особливо критичних у контексті управління особистими проєктами.

1. Відсутність або обмеженість вбудованого тайм-трекінгу. ClickUp пропонує нативний тайм-трекінг із можливістю запуску/зупинки таймера,

перегляду таймшитів та встановлення оцінок часу. Натомість більшість платформ покладаються на зовнішні інтеграції для цієї функціональності [13]. Trello та Asana взагалі не мають нативного тайм-трекінгу.

2. Надмірна функціональна складність. Інструменти корпоративного класу вимагають безлічі доповнень для роботи поза базовими робочими процесами. Для управління залежностями, розподілом ресурсів або детальною аналітикою необхідне підключення множини платних плагінів [9]. Для індивідуального користувача подібна складність є зайвою та підвищує поріг входу.

3. Слабке довгострокове планування. Канбан-підхід має суттєві обмеження: він не підходить для проєктів зі складним процесом роботи, а довгострокове планування в ньому майже нереальне [6]. Більшість існуючих інструментів не вирішують цю проблему на рівні архітектури застосунку.

4. Обмежена персоналізація для особистого використання. Платформи класу «enterprise» (Jira, Monday.com) надмірно орієнтовані на командну роботу та не адаптовані під потреби одного користувача. Trello ідеально підходить для управління індивідуальними проєктами або фрілансу, але лише для найпростіших сценаріїв [8].

5. Відсутність єдиного інтегрованого рішення. Жоден із розглянутих застосунків не поєднує органічно канбан-дошку, вбудований тайм-трекінг і аналітику витраченого часу в рамках єдиного легковагого інтерфейсу, орієнтованого на особисте використання [8].

Проведений аналіз засвідчує, що ринок вебзастосунків для управління проєктами є насиченим, однак існуючі рішення здебільшого орієнтовані або на корпоративне командне використання (Jira, Monday.com), або на надзвичайно спрощену модель (Trello без плагінів). Розрив між цими двома полюсами зумовлює актуальність розроблення спеціалізованого вебзастосунку для управління особистими проєктами, який органічно поєднує: (1) гнучку канбан-дошку з налаштовуваними колонками; (2) вбудований тайм-трекінг на рівні задач без потреби у сторонніх інтеграціях; (3) мінімалістичний інтерфейс, вільний від функціональної надмірності, характерної для корпоративних рішень.

1.3 Визначення вимог до вебзастосунку для управління особистими проєктами з канбан-дошкою та тайм-трекінгом

Визначення вимог до програмної системи є базовим етапом її проєктування, оскільки саме на цьому рівні формуються функціональні можливості майбутнього вебзастосунку та встановлюються критерії його якості. Для вебзастосунку управління особистими проєктами, що базується на канбан-дошці та тайм-трекінгу, вимоги доцільно розділити на дві основні групи: функціональні та нефункціональні. Такий поділ дозволяє чітко структурувати очікувану поведінку системи та її технічні характеристики.

Перш за все розглянемо функціональні вимоги (табл. 1.2), які визначають, що саме система повинна робити для забезпечення роботи користувача з завданнями та проєктами.

Таблиця 1.2 – Функціональні вимоги до вебзастосунку

Код	Вимога	Опис
FR-1	Реєстрація та автентифікація	Створення облікового запису за email і паролем, забезпечення безпечного входу та виходу користувача
FR-2	Управління сесіями	Підтримка авторизованого стану користувача між запитами за допомогою cookie або токенів сесії
FR-3	Управління завданнями	Створення, редагування та видалення завдань із зазначенням назви, опису та пріоритету
FR-4	Канбан-візуалізація	Відображення завдань у вигляді колонок за статусами (Backlog, To Do, In Progress, Review, Done)
FR-5	Переміщення завдань	Зміна статусу завдань шляхом drag-and-drop взаємодії між колонками канбан-дошки
FR-6	Тайм-трекінг	Запуск та зупинка таймера для фіксації часу, витраченого на виконання конкретного завдання
FR-7	Зведена статистика	Формування узагальнених показників щодо кількості завдань та загального витраченого часу

У наведеній таблиці представлено ключовий функціонал системи, який забезпечує повний цикл роботи користувача з особистими проєктами. Перші вимоги (FR-1 та FR-2) відповідають за базову безпеку та доступ до системи,

оскільки без реєстрації та збереження сесії неможливо забезпечити персоналізацію даних.

Наступна група вимог (FR-3) описує базову модель роботи із завданнями, яка є ядром усієї системи. Саме через створення, редагування та видалення задач користувач формує структуру свого проєкту.

Далі вимоги FR-4 та FR-5 стосуються канбан-візуалізації, яка дозволяє представити завдання у вигляді зрозумілої та логічної структури колонок. Такий підхід забезпечує наочність процесу виконання проєкту та дозволяє користувачу швидко оцінювати поточний стан задач. Функція перетягування завдань між колонками робить взаємодію з системою інтуїтивною та швидкою.

Окрему роль відіграє тайм-трекінг (FR-6), який дозволяє фіксувати фактичний час виконання кожного завдання. Це дає змогу користувачу аналізувати власну продуктивність та планувати майбутні задачі більш ефективно. Завершує функціональний блок вимога FR-7, яка передбачає формування загальної статистики, що дозволяє отримати узагальнену картину роботи над проєктами.

Після визначення функціональних можливостей системи доцільно розглянути нефункціональні вимоги (табл. 1.3), які описують якісні характеристики вебзастосунку та впливають на його стабільність, безпеку та зручність використання.

Таблиця 1.3 – Нефункціональні вимоги до вебзастосунку

Код	Категорія	Вимога
NFR-1	Безпека	Хешування паролів, використання httpOnly-cookie, ізоляція даних користувачів та захист від SQL-ін’єкцій
NFR-2	Продуктивність	Час відгуку інтерфейсу не більше 200 мс завдяки оптимізації запитів та використанню оптимістичних оновлень
NFR-3	Зручність	Адаптивний інтерфейс (mobile-first), підтримка української мови, світла та темна тема
NFR-4	Надійність	Забезпечення цілісності даних через використання зовнішніх ключів та каскадного видалення
NFR-5	Супровідність	Модульна архітектура та використання статичної типізації для спрощення підтримки системи

Нефункціональні вимоги визначають рівень якості роботи системи та її придатність до практичного використання. Зокрема, вимога безпеки (NFR-1) передбачає захист персональних даних користувача шляхом хешування паролів та використання захищених механізмів зберігання сесій, що мінімізує ризики несанкціонованого доступу.

Вимога продуктивності (NFR-2) визначає швидкодію системи, яка є критично важливою для комфортної взаємодії користувача з інтерфейсом. Завдяки оптимізації запитів та використанню сучасних підходів до оновлення даних забезпечується швидка реакція системи.

Зручність використання (NFR-3) передбачає адаптивність інтерфейсу до різних пристроїв, підтримку української мови та можливість перемикання між темами оформлення, що підвищує доступність системи для широкого кола користувачів.

Надійність (NFR-4) гарантує коректність збереження та обробки даних завдяки використанню зв’язків у базі даних та механізмів каскадного видалення, що запобігає появі неконсистентних записів.

Остання вимога – супровідність (NFR-5) – забезпечує можливість подальшого розвитку системи. Модульна архітектура та структурований код дозволяють легко змінювати або розширювати функціонал без значних переробок.

Отже, сформульовані функціональні та нефункціональні вимоги визначають повний набір характеристик вебзастосунку для управління особистими проєктами. Функціональні вимоги описують безпосередню поведінку системи, включаючи роботу із завданнями, канбан-дошку та тайм-трекінг, тоді як нефункціональні вимоги встановлюють критерії якості, що забезпечують безпеку, продуктивність та зручність використання. На основі цих вимог у подальших розділах виконується проєктування архітектури та реалізація програмного продукту.

2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ОСОБИСТИМИ ПРОЄКТАМИ

2.1 Аналіз предметної області та проєктування архітектури системи

Аналіз предметної області вебзастосунку для управління особистими проєктами дозволяє визначити ключові процеси, що підлягають автоматизації. До таких процесів належать планування завдань, їх структуризація за станами виконання, візуалізація робочого процесу, а також облік витраченого часу. У сучасних умовах користувачі потребують простих, інтуїтивних інструментів, які дозволяють ефективно організовувати власну діяльність без надмірної складності корпоративних систем.

Основною сутністю предметної області є особистий проєкт, який складається з набору завдань, що мають життєвий цикл: створення, виконання, переміщення між статусами та завершення. Додатковим елементом є контроль часу виконання завдань, що реалізується через тайм-трекінг. Таким чином, система повинна забезпечувати повний цикл управління задачами від їх створення до аналізу результатів виконання.

Для формалізації взаємодії користувача з системою використовується діаграма варіантів використання (Use Case), яка відображає основні сценарії роботи та визначає функціональні межі вебзастосунку.

Візуальне представлення діаграми наведено на рисунку 2.1:



Рисунок 2.1 – Діаграма варіантів використання

З наведеної діаграми видно, що єдиним актором системи виступає Користувач, який взаємодіє з усім функціоналом вебзастосунку. Першочерговими діями є реєстрація та вхід у систему, які забезпечують доступ до персоналізованого середовища.

Основна група сценаріїв пов’язана з управлінням завданнями. Користувач має можливість створювати нові завдання, редагувати їх у разі зміни умов, а також видаляти ті, що втратили актуальність. Це формує базову модель роботи із проєктами та дозволяє підтримувати актуальність інформації в системі.

Окремий блок функціональності становить робота з канбан-дошкою. Система дозволяє переміщувати завдання між колонками, що відповідають різним статусам виконання. Такий підхід забезпечує наочну візуалізацію робочого процесу та дозволяє користувачу швидко оцінювати прогрес виконання задач.

Додатково система підтримує тайм-трекінг, який реалізується через запуск і зупинку таймера для кожного завдання. Це дозволяє фіксувати фактичні витрати часу та аналізувати продуктивність користувача. Завершальним сценарієм є перегляд статистики, який надає узагальнену інформацію про виконану роботу, а також вихід із системи, що завершує сесію користувача.

Архітектура вебзастосунку розроблена за багаторівневим (layered) принципом, що забезпечує чіткий поділ відповідальності між компонентами системи та спрощує її масштабування і супровід. Реалізація базується на фреймворку Next.js (App Router), який поєднує серверний рендеринг і клієнтську взаємодію в межах єдиної структури.

Особливістю архітектури є використання Server Actions, які дозволяють виконувати серверну логіку без створення окремого REST API, що спрощує взаємодію між клієнтською та серверною частинами застосунку.

Загальна структура системи відображена на компонентній діаграмі (рис. 2.2):

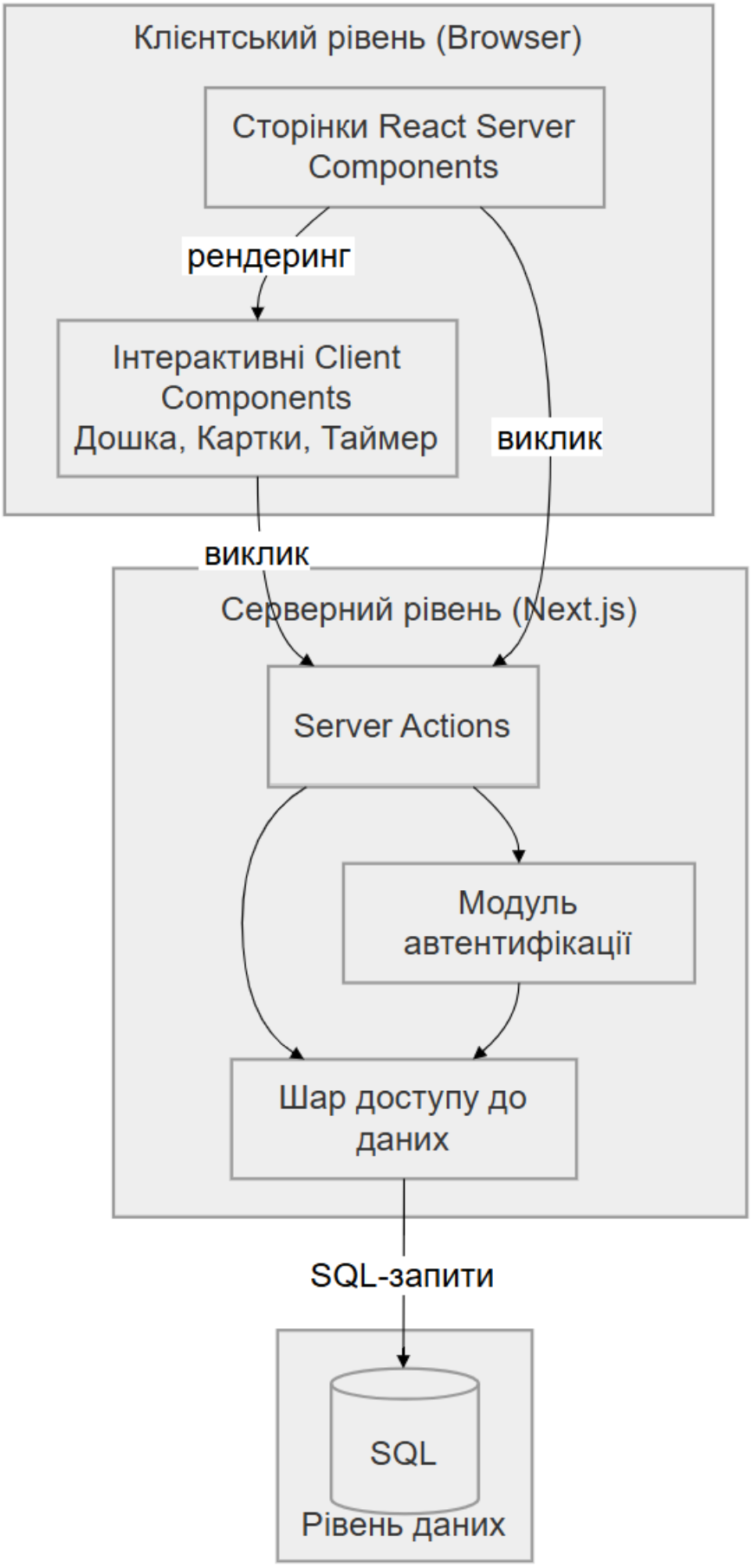


Рисунок 2.2 – Архітектура системи

З діаграми видно, що система складається з трьох основних рівнів: клієнтського, серверного та рівня доступу до даних. Клієнтський рівень відповідає за відображення інтерфейсу користувача, включаючи канбан-дошку, картки завдань та таймер. Саме на цьому рівні відбувається безпосередня взаємодія користувача із системою.

Серверний рівень реалізує бізнес-логіку застосунку. Тут обробляються дії користувача, виконується перевірка даних, а також здійснюється авторизація. Основними компонентами цього рівня є Server Actions та модуль автентифікації, які забезпечують виконання операцій над завданнями та користувачами.

Рівень доступу до даних відповідає за взаємодію з базою даних. Він реалізує виконання SQL-запитів, отримання та збереження інформації. Для зберігання даних використовується SQLite як легка та зручна реляційна база даних.

У результаті запропонована архітектура забезпечує чіткий розподіл відповідальності між рівнями системи, що підвищує її масштабованість, зручність супроводу та стабільність роботи.

У межах підрозділу було здійснено аналіз предметної області вебзастосунку для управління особистими проєктами, визначено ключові сутності та основні процеси взаємодії користувача із системою, зокрема управління завданнями, візуалізація їхнього стану за допомогою канбан-дошки та облік витраченого часу. На основі аналізу сформовано модель використання системи у вигляді діаграми варіантів використання, що дозволило формалізувати основні сценарії роботи користувача.

Також було спроєктовано архітектуру системи, яка базується на багаторівневому підході з поділом на клієнтський рівень, рівень бізнес-логіки та рівень доступу до даних. Запропонована архітектура забезпечує чіткий розподіл відповідальності між компонентами системи, що підвищує її масштабованість, зрозумілість і зручність подальшого супроводу.

2.2 Проєктування бази даних та основних функціональних модулів

Проєктування бази даних та функціональних модулів є ключовим етапом розробки вебзастосунку, оскільки саме на цьому рівні визначається структура збереження інформації та логіка взаємодії між основними сутностями системи. Для забезпечення ефективної роботи застосунку необхідно чітко визначити дані, що зберігаються, їх взаємозв’язки, а також модулі, які реалізують основний функціонал системи.

У межах даного розділу здійснено концептуальне та логічне проєктування бази даних, а також визначено структуру функціональних модулів, які забезпечують реалізацію ключових можливостей вебзастосунку.

На основі аналізу предметної області було визначено чотири основні сутності системи: користувач (User), сесія (Session), проєкти (Task) та запис часу (TimeEntry). Ці сутності відображають базові об’єкти, з якими працює система, та формують основу для побудови реляційної бази даних.

Користувач є центральною сутністю, яка зберігає дані про обліковий запис.

Сесія забезпечує механізм авторизації та збереження стану входу.

Завдання є основною одиницею роботи в системі.

Запис часу фіксує інформацію про витрачений час на виконання завдань.

Концептуальну модель даних подано у вигляді ER-діаграми (рис. 2.3).

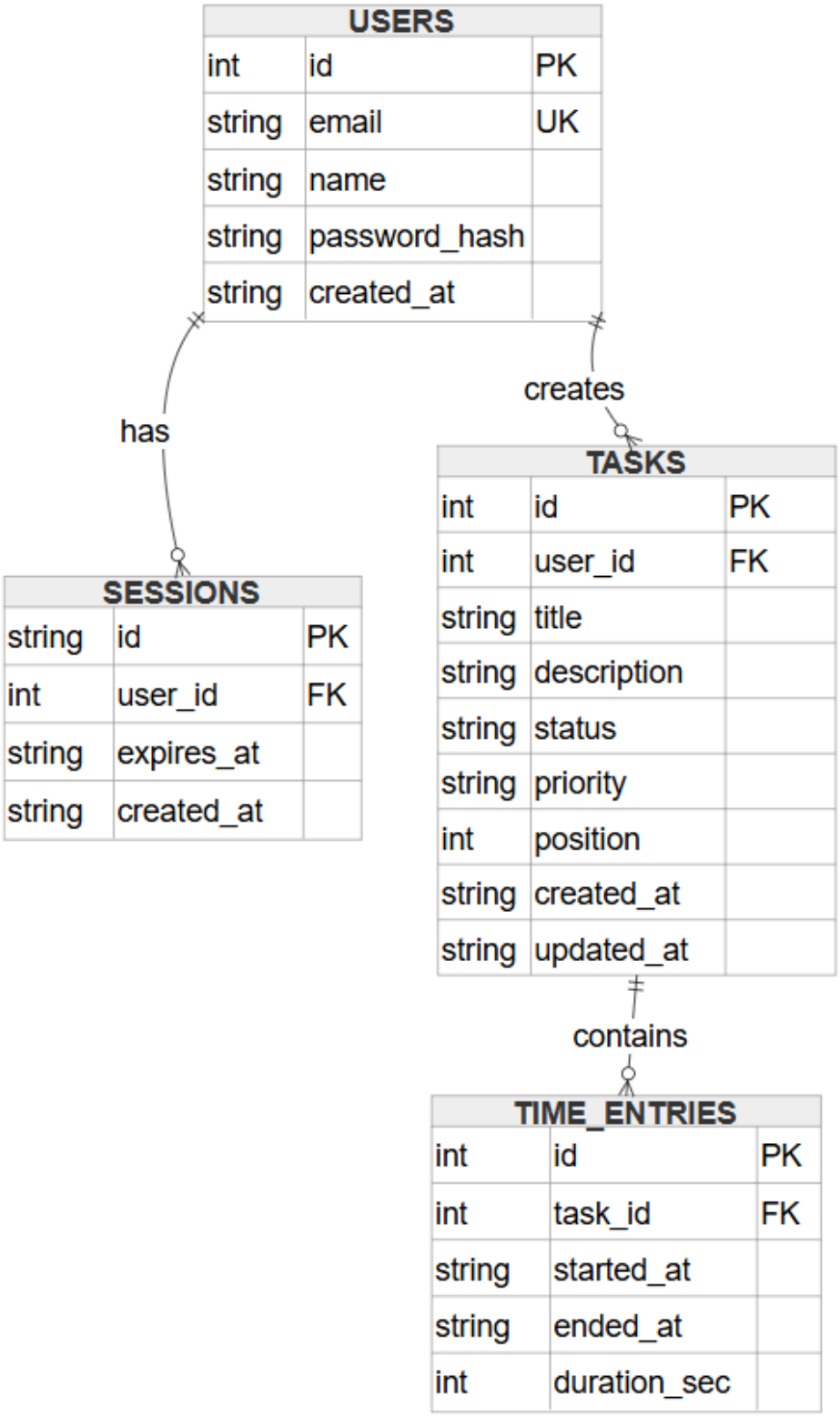


Рисунок 2.3 – Концептуальну модель даних

З наведеної діаграми видно, що між сутностями встановлено такі зв’язки:

- один користувач може мати багато сесій;
- один користувач може створювати багато завдань;

- кожне завдання/проєкт може мати багато записів часу;
- записи часу безпосередньо прив’язані до конкретного проєкту.

Таким чином, модель даних має ієрархічну структуру з чітким розподілом зв’язків «один-до-багатьох», що забезпечує логічну цілісність системи.

Після визначення концептуальної моделі було спроєктовано логічну структуру бази даних, яка реалізується у реляційній СУБД SQLite. Вона представлена у вигляді таблиць із визначеними типами даних, первинними та зовнішніми ключами.

Таблиця users (табл. 2.1) є базовою сутністю системи, оскільки саме вона зберігає інформацію про всіх зареєстрованих користувачів. Кожен запис містить унікальний ідентифікатор id, який використовується для зв’язку з іншими таблицями. Поле email виконує роль логіна та має обмеження унікальності, що виключає можливість дублювання акаунтів.

Ім’я користувача зберігається у полі name та використовується для відображення в інтерфейсі системи. Поле password_hash містить не сам пароль, а його захешоване значення, що підвищує рівень безпеки зберігання даних. Дата створення облікового запису фіксується у полі created_at і дозволяє відстежувати момент реєстрації користувача.

Таблиця 2.1 – Таблиця користувачів (users)

Поле	Тип	Обмеження	Опис
id	INTEGER	PK, AUTOINCREMENT	Унікальний ідентифікатор користувача
email	TEXT	UNIQUE, NOT NULL	Електронна пошта для входу
name	TEXT	NOT NULL	Ім’я користувача
password_hash	TEXT	NOT NULL	Хешований пароль
created_at	TEXT	DEFAULT now	Дата створення облікового запису

Таблиця sessions (табл. 2.2) використовується для реалізації механізму автентифікації та збереження стану входу користувача в систему. Кожна сесія має унікальний токен id, який зберігається у cookie та дозволяє ідентифікувати користувача без повторного введення логіна і пароля.

Поле user_id встановлює зв’язок із таблицею користувачів і визначає, кому належить конкретна сесія. Атрибут expires_at визначає момент завершення дії сесії, після чого користувач автоматично втрачає доступ до системи. Поле created_at фіксує час створення сесії та використовується для контролю активності входів.

Таблиця 2.2 – Сесії (sessions)

Поле	Тип	Обмеження	Опис
id	TEXT	PK	Токен сесії
user_id	INTEGER	FK – users.id	Власник сесії
expires_at	TEXT	NOT NULL	Час завершення сесії
created_at	TEXT	DEFAULT now	Час створення

Таблиця tasks (табл. 2.3) є основною робочою таблицею системи, оскільки саме вона зберігає всі проєкти користувачів. Кожний проєкт має унікальний ідентифікатор id та прив’язане до конкретного користувача через поле user_id.

Поле title містить назву проєкта, яка відображається на канбан-дошці, а description дозволяє зберігати детальний опис задачі. Атрибут status визначає поточний етап виконання проєкту (наприклад: Backlog, To Do, In Progress, Done) і є основою канбан-логіки.

Поле priority використовується для визначення важливості проєкту, а position відповідає за його розташування всередині колонки, що особливо важливо для реалізації drag-and-drop механізму. Поля created_at та updated_at дозволяють відстежувати життєвий цикл проєкту та моменти його змін.

Таблиця 2.3 – Завдання/проєкт (tasks)

Поле	Тип	Обмеження	Опис
id	INTEGER	PK, AUTOINCREMENT	Ідентифікатор завдання
user_id	INTEGER	FK – users.id	Власник завдання
title	TEXT	NOT NULL	Назва завдання
description	TEXT	–	Опис завдання
status	TEXT	NOT NULL	Статус у канбан-дошці
priority	TEXT	DEFAULT ‘medium’	Пріоритет виконання
position	INTEGER	NOT NULL	Позиція у колонці
created_at	TEXT	DEFAULT now	Дата створення
updated_at	TEXT	DEFAULT now	Дата оновлення

Таблиця time_entries (табл. 2.4) призначена для реалізації функціоналу тайм-трекінгу та збереження історії роботи над проєктами. Кожен запис у цій таблиці відповідає окремій сесії роботи над конкретним проєктами.

Поле task_id встановлює зв’язок із таблицею завдань і визначає, до якої саме задачі відноситься запис часу. Атрибут started_at фіксує момент запуску таймера, а ended_at – момент його зупинки.

Якщо значення ended_at дорівнює NULL, це означає, що таймер наразі активний і облік часу триває. Поле duration_sec зберігає підсумкову тривалість виконання в секундах і заповнюється після завершення сесії.

Таблиця 2.4 – Облік часу (time_entries)

Поле	**Тип**	**Обмеження**	**Опис**
id	INTEGER	PK, AUTOINCREMENT	Ідентифікатор запису
task_id	INTEGER	FK – tasks.id	Завдання
started_at	TEXT	NOT NULL	Час початку
ended_at	TEXT	–	Час завершення
duration_sec	INTEGER	DEFAULT 0	Тривалість у секундах

Функціональність системи реалізована у вигляді окремих модулів, кожен з яких відповідає за певну підсистему вебзастосунку. Такий підхід дозволяє забезпечити модульність, масштабованість та простоту супроводу коду.

Загальну структуру модулів подано на схемі на рисунку 2.4.

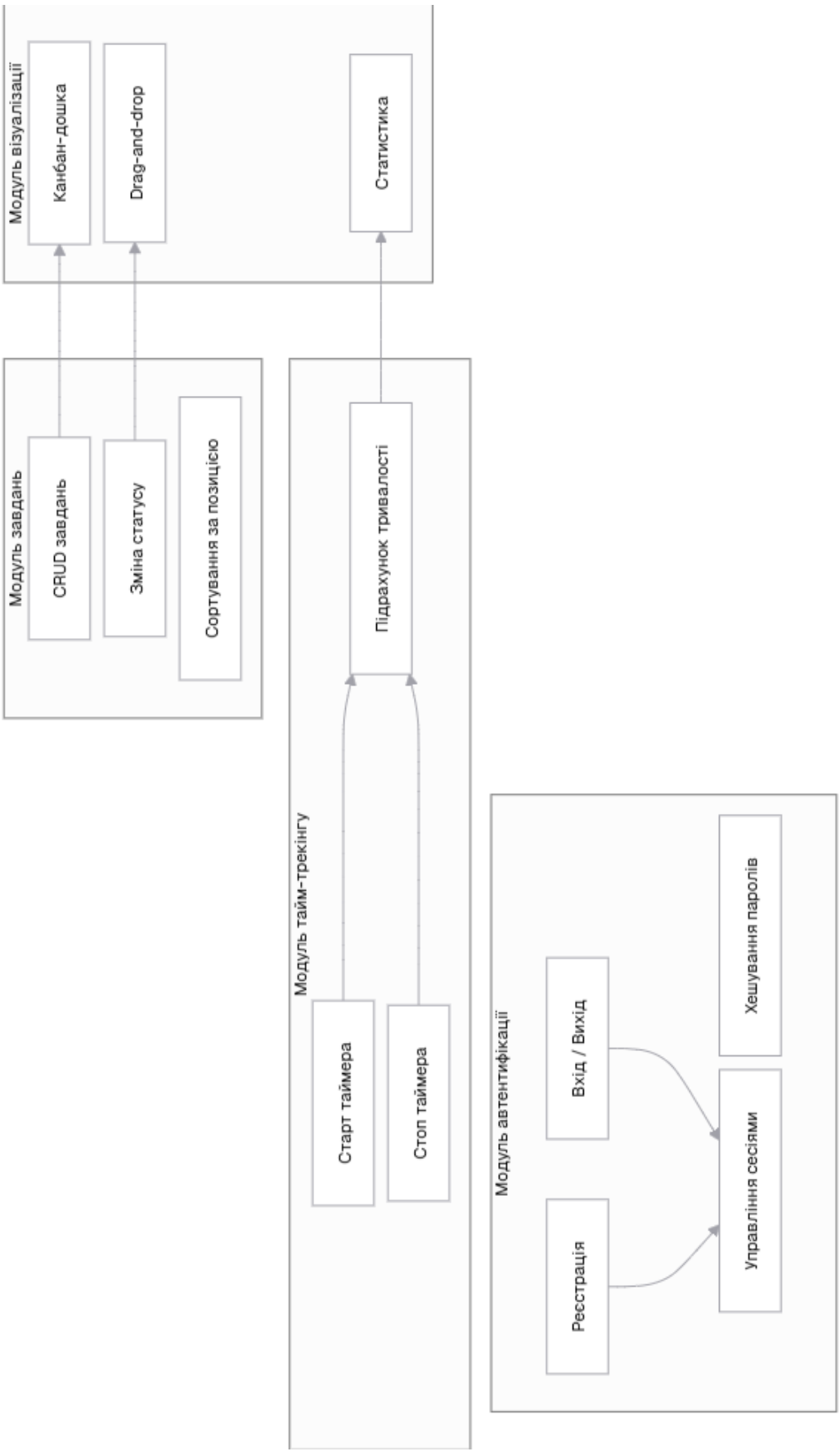


Рисунок 2.4 – Структура функціональних модулів

З діаграми видно, що система складається з чотирьох основних функціональних модулів:

Модуль автентифікації відповідає за реєстрацію користувачів, вхід у систему, вихід та управління сесіями. Також він забезпечує безпечне зберігання паролів через хешування. Модуль завдань реалізує CRUD-операції, зміну статусів завдань та їх впорядкування. Модуль тайм-трекінгу забезпечує запуск і зупинку таймера та підрахунок витраченого часу. Модуль візуалізації відповідає за відображення канбан-дошки, drag-and-drop взаємодію та побудову статистики.

Таким чином, модульна структура дозволяє розділити логіку системи на незалежні компоненти, що спрощує її розвиток та тестування.

Одним із ключових функціональних елементів системи є тайм-трекінг, який дозволяє фіксувати витрачений час на виконання завдань.

Логіку його роботи відображає діаграма послідовності (рис. 2.5):

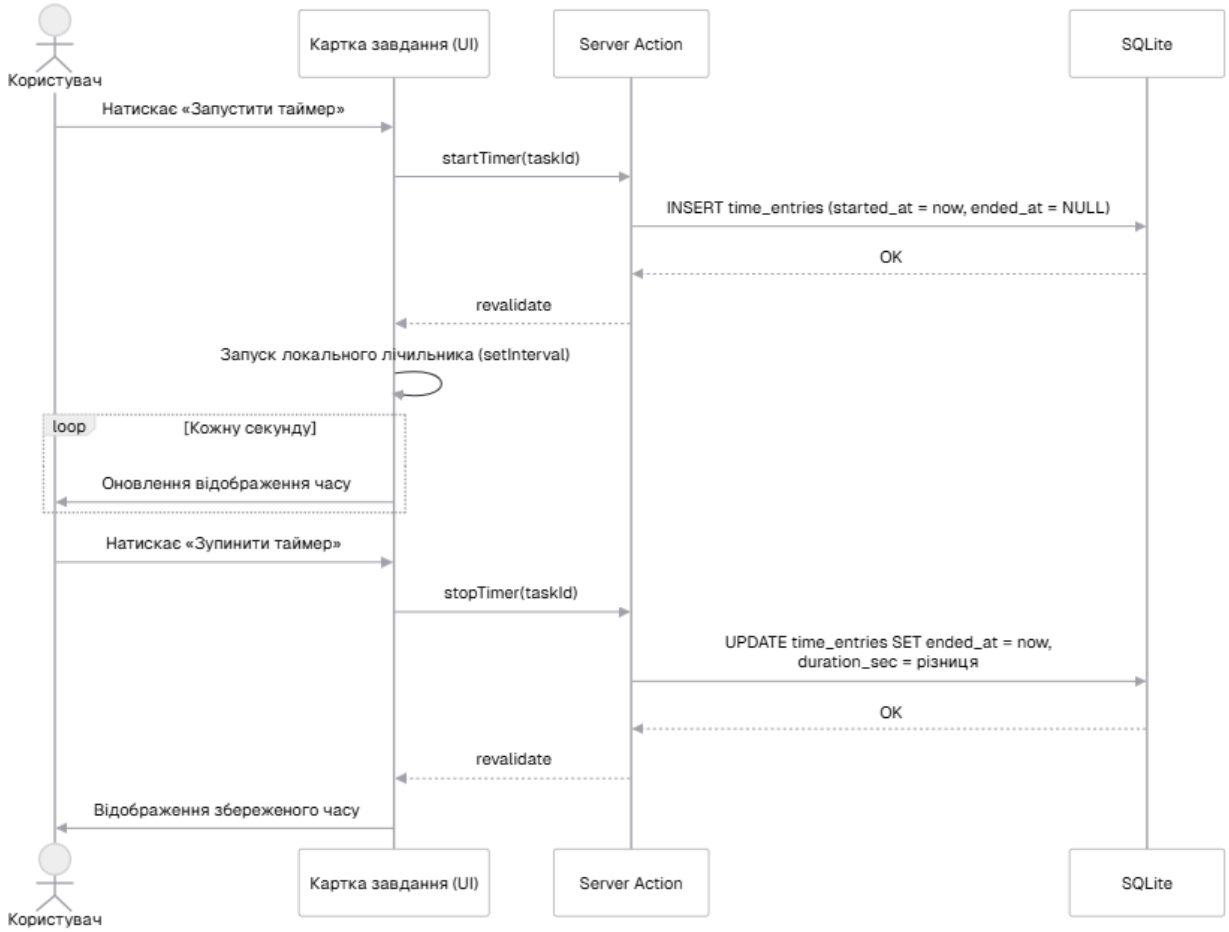


Рисунок 2.5 – Алгоритм роботи тайм-трекінгу

. З діаграми видно, що при запуску таймера створюється новий запис у таблиці time_entries, де фіксується час початку. Після зупинки таймера запис оновлюється шляхом додавання часу завершення та обчислення тривалості виконання завдання.

Загальний час виконання завдання розраховується за формулою:

$$T_{total} = \sum_{i=1}^{n} (ended_i - started_i)$$

де n – кількість сесій обліку часу для конкретного завдання.

У результаті виконання роботи було спроєктовано логічну структуру бази даних, визначено основні сутності та їх взаємозв'язки, а також розроблено функціональну декомпозицію системи. Запропонована модель даних забезпечує цілісність, масштабованість та ефективність зберігання інформації, а модульна архітектура програмної системи дозволяє гнучко розширювати її функціональність у майбутньому.

2.3 Проєктування користувацького інтерфейсу

Проєктування користувацького інтерфейсу є важливим етапом розробки вебзастосунку, оскільки саме інтерфейс визначає зручність, швидкість та ефективність взаємодії користувача із системою. Основною метою цього етапу було створення інтуїтивного, мінімалістичного та адаптивного інтерфейсу для роботи з канбан-дошкою та функцією тайм-трекінгу.

Інтерфейс проєктувався з урахуванням сучасних принципів UX/UI-дизайну та вимог до вебзастосунків реального часу.

Інтерфейс розроблено відповідно до принципів юзабіліті, зокрема: видимість стану системи, відповідність між системою та реальним світом, контроль і свобода дій користувача, а також естетичний мінімалізм.

Видимість стану системи реалізована через відображення активного таймера, статусів завдань та змін у реальному часі. Користувач завжди отримує зворотний зв’язок після виконання будь-якої дії.

Відповідність системи реальному світу забезпечується використанням зрозумілих термінів та логічної структури канбан-дошки, яка відображає типовий робочий процес: від створення задачі до її виконання.

Свобода дій користувача реалізована через можливість редагування, переміщення та видалення завдань без обмежень, а також через можливість скасування змін.

Інтерфейс побудовано за принципом мінімалізму, що дозволяє уникнути перевантаження елементами та зосередити увагу користувача на основному функціоналі.

Для візуального оформлення обрано темну колірну схему (табл. 2.5) з акцентним оранжевим кольором, що забезпечує комфортну роботу при тривалому використанні системи та зменшує навантаження на зір.

Таблиця 2.5 – Колірна палітра інтерфейсу

Призначення	**Колір**	**Застосування**
Тло (background)	Темно-сірий	Основний фон інтерфейсу
Поверхні (card)	Сірий	Картки завдань, блоки та колонки
Акцент (primary)	Оранжевий	Кнопки дій, активні елементи, таймер
Основний текст	Світло-сірий	Заголовки та основний контент
Другорядний текст	Приглушений сірий	Підписи, допоміжна інформація

Обмеження палітри до 3–5 основних кольорів дозволяє досягти візуальної цілісності та уникнути надмірної фрагментації інтерфейсу.

Наступним етапом проєктування стало визначення логіки навігації між основними екранами системи. Навігаційна модель відображає повний цикл взаємодії користувача – від авторизації до роботи з канбан-дошкою та управління завданнями (рис. 2.6).



Рисунок 2.6 – Навігаційна структура користувацького інтерфейсу

Описана діаграма демонструє основні стани системи та переходи між ними. Початковою точкою є сторінка входу, після чого користувач може перейти до реєстрації або безпосередньо до канбан-дошки у разі успішної авторизації. Додатково передбачено повернення до сторінки входу після виходу із системи, що забезпечує коректний цикл сесійної взаємодії.

Головний екран системи реалізовано у вигляді канбан-дошки, яка є центральним елементом взаємодії користувача із завданнями. Його структура побудована за принципом вертикальної ієрархії: спочатку відображається заголовкова частина з ключовою інформацією, після чого розміщується робоча зона у вигляді колонок (рис. 2.7).

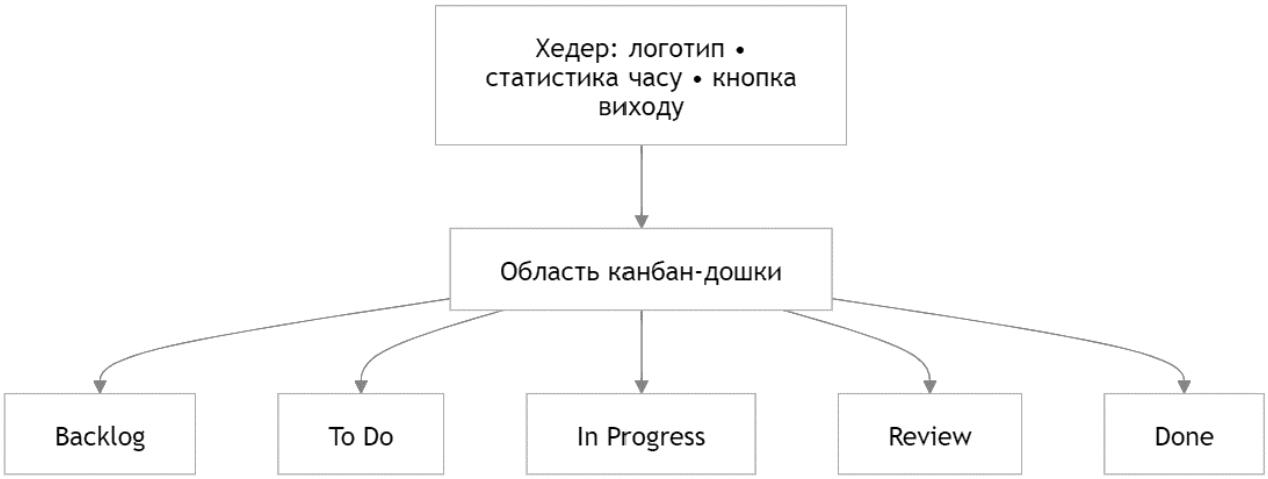


Рисунок 2.7 – Структура головного екрана

Картка завдання є базовим функціональним елементом системи, який об’єднує всю ключову інформацію про конкретне завдання та забезпечує взаємодію з ним (табл. 2.6).

Таблиця 2.6 – Структура картки завдання

Елемент	**Призначення**
Бейдж пріоритету	Візуальне позначення важливості завдання (низький, середній, високий)
Назва завдання	Основний заголовок картки
Опис	Короткий зміст завдання
Індикатор часу	Відображення сумарного витраченого часу у форматі ГГ:ХХ:СС
Кнопка таймера	Запуск та зупинка обліку часу
Меню дій	Редагування та видалення завдання

Крім інформаційної складової, картка підтримує інтерактивність, зокрема механізм перетягування (drag-and-drop), що дозволяє змінювати статус завдання шляхом переміщення між колонками канбан-дошки. При цьому застосовується принцип оптимістичного оновлення, за якого зміни відображаються миттєво без очікування підтвердження сервера, що підвищує швидкодію інтерфейсу та покращує користувацький досвід.

У результаті проєктування користувацького інтерфейсу було сформовано логічно структуровану, ергономічну та адаптивну систему взаємодії користувача із застосунком. Використання принципів юзабіліті, продуманої навігації та

компонентної структури дозволило створити інтерфейс, орієнтований на зручність, швидкодію та інтуїтивність використання.

2.4 Обґрунтування вибору технологій розробки

У процесі проєктування та реалізації програмної системи важливим етапом стало обґрунтування вибору технологічного стеку, оскільки саме від нього залежить ефективність розробки, стабільність роботи застосунку та можливості його подальшого розвитку. Вибір технологій здійснювався комплексно з урахуванням вимог до функціональності системи, її продуктивності, рівня безпеки, доступності документації, активності спільноти розробників, а також доцільності використання в межах індивідуального дипломного проєкту.

Передусім було визначено основні критерії, на які орієнтувався вибір технологій. До них належать відповідність функціональним вимогам системи, здатність забезпечити високу продуктивність вебзастосунку, достатній рівень безпеки під час обробки персональних даних, простота розгортання та супроводу, а також наявність зрілої екосистеми та якісної документації. Окремо враховувався фактор зручності використання технологій в умовах обмеженого часу та ресурсів, характерних для навчального проєкту.

На основі цих критеріїв було обрано фреймворк Next.js [14], побудований на базі бібліотеки React [15], для реалізації клієнтської та серверної частини вебзастосунку. Порівняно з класичним підходом на основі SPA з використанням лише React, Next.js надає суттєві переваги, зокрема підтримку серверного рендерингу, файлову маршрутизацію та можливість використання Server Actions. Саме Server Actions дозволяють викликати серверну логіку безпосередньо з компонентів інтерфейсу без необхідності створення окремого REST API, що значно спрощує архітектуру системи, зменшує кількість проміжних шарів і пришвидшує розробку.

Альтернативні рішення, такі як Vue [16] або Nuxt [17], також розглядалися, однак у контексті даного проєкту вони не надали суттєвих переваг. Хоча ці

фреймворки підтримують подібний функціонал, екосистема React і Next.js є більш розвиненою та краще інтегрується з TypeScript [18], який використовується у проєкті для підвищення надійності коду.

Наступним важливим рішенням став вибір системи управління базою даних. Для зберігання даних обрано SQLite [19] як легку вбудовану реляційну СУБД. Її основною перевагою є відсутність необхідності у розгортанні окремого серверного процесу, що суттєво спрощує інсталяцію та запуск застосунку. У порівнянні з PostgreSQL [20] або MongoDB [21], які потребують окремої інфраструктури та додаткової конфігурації, SQLite є значно більш доцільним рішенням для індивідуального або навчального проєкту. При цьому вона забезпечує повну підтримку реляційної моделі даних, транзакційність та достатній рівень продуктивності для задач подібного масштабу. У межах реалізації також використано вбудований механізм роботи з базою даних у середовищі Node.js [22], що дозволило уникнути зайвих зовнішніх залежностей.

Після аналізу окремих компонентів було сформовано фінальний технологічний стек, який забезпечує повний цикл розробки системи – від побудови інтерфейсу до реалізації серверної логіки та роботи з базою даних. У якості основної мови програмування використано TypeScript [18], що дозволяє підвищити надійність коду завдяки статичній типізації та зменшити кількість потенційних помилок ще на етапі компіляції. Фронтенд реалізовано на основі Next.js з використанням React, що забезпечує компонентний підхід до побудови інтерфейсу, зручну маршрутизацію та можливість серверного рендерингу. Для стилізації інтерфейсу застосовано Tailwind CSS [23], який дозволяє швидко створювати адаптивний дизайн за допомогою утилітарного підходу. Додатково використано бібліотеку shadcn/ui для швидкого впровадження готових UI-компонентів, а також lucide-react для уніфікованої системи іконок. Як систему зберігання даних застосовано SQLite, тоді як автентифікація реалізована за допомогою алгоритму scrypt у поєднанні з cookie-сесіями.

Окрему увагу приділено питанням безпеки, оскільки система передбачає роботу з персональними даними користувачів. Для цього реалізовано

багаторівневий підхід до захисту. Паролі користувачів зберігаються виключно у хешованому вигляді із застосуванням алгоритму scrypt та унікальної солі для кожного запису, що унеможливлює їх відновлення у разі витоку бази даних. Для управління сесіями використано httpOnly cookie, що захищає сесійні токени від доступу через клієнтський JavaScript і знижує ризик XSS-атак. Крім того, всі запити до бази даних додатково фільтруються за ідентифікатором користувача, що забезпечує ізоляцію даних між користувачами. Використання параметризованих SQL-запитів виключає можливість SQL-ін’єкцій, що є одним із базових вимог безпечної роботи з реляційними базами даних.

Таким чином, обраний технологічний стек є збалансованим і оптимально відповідає вимогам проєкту, забезпечуючи поєднання простоти реалізації, високої продуктивності, достатнього рівня безпеки та зручності подальшого супроводу системи.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ОСОБИСТИМИ ПРОЄКТАМИ

3.1 Реалізація вебзастосунку

Реалізацію вебзастосунку виконано відповідно до архітектурних рішень, визначених у попередньому розділі, з дотриманням принципів модульності, масштабованості та розділення відповідальностей. Загальна структура проєкту побудована на основі можливостей фреймворку Next.js (App Router), що дозволяє поєднувати клієнтську та серверну логіку в межах єдиної кодової бази та забезпечує зручну організацію компонентів за функціональними модулями.

У межах реалізації застосунку використано ієрархічну файлову структуру, де кожен каталог відповідає за окремий рівень логіки системи. Центральним елементом є директорія app, яка містить маршрути застосунку, включаючи сторінки авторизації, реєстрації та головну сторінку канбан-дошки. Такий підхід забезпечує природну маршрутизацію без необхідності додаткових конфігурацій.

Окремо виділено директорію actions, у якій реалізовано серверні функції (Server Actions), що відповідають за взаємодію з базою даних та виконання бізнес-логіки. Це дозволяє викликати серверні операції безпосередньо з клієнтських компонентів, зменшуючи складність архітектури та усуваючи потребу у побудові окремого REST API.

Компоненти користувацького інтерфейсу структуровано у директоріях components/board та components/ui. Перша містить специфічні компоненти канбан-дошки, такі як колонки, картки завдань та діалоги, тоді як друга включає універсальні UI-елементи, побудовані на основі бібліотеки shadcn/ui. Додатково директорія lib містить допоміжні модулі, зокрема підключення до бази даних, функції автентифікації та утиліти форматування.

Для наочного представлення архітектури файлової структури застосунку використано діаграму 3.1.

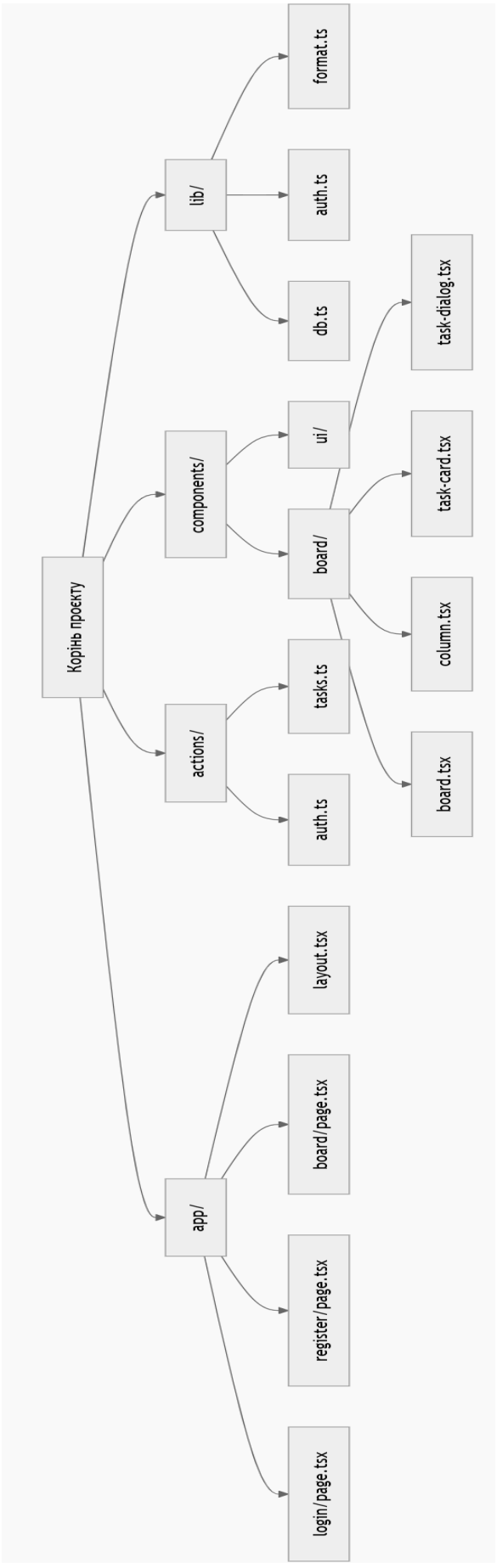


Рисунок 3.1 – Структура проєкту вебзастосунку

У межах реалізації системи зберігання даних було розроблено модуль ініціалізації та конфігурації бази даних, який відповідає за створення підключення до SQLite, формування структури таблиць, а також первинне наповнення системи демонстраційними даними. Повний вихідний код цього модуля наведено у Додатку А.1.

Даний програмний модуль виконує ключову роль у стартовій конфігурації застосунку, оскільки забезпечує централізоване управління базою даних та гарантує коректну ініціалізацію всіх необхідних структур перед початком роботи системи.

На початковому етапі роботи модуля створюється або перевіряється наявність директорії для зберігання бази даних, що забезпечує можливість запису даних у файлову систему середовища виконання. Далі формується шлях до файлу бази даних, після чого ініціалізується підключення до SQLite через модуль node:sqlite.

Особливістю реалізації є використання патерну Singleton, який дозволяє повторно використовувати вже створене з’єднання з базою даних у межах одного життєвого циклу застосунку. Це особливо важливо в умовах середовища Next.js, де можливі повторні ініціалізації модулів під час гарячого перезавантаження.

Після створення підключення виконується конфігурація параметрів SQLite, зокрема увімкнення режиму журналювання WAL (Write-Ahead Logging), що підвищує продуктивність роботи з даними, а також активація зовнішніх ключів для забезпечення цілісності зв’язків між таблицями.

Наступним етапом є виконання функції міграції, у якій декларативно описується структура бази даних. У межах цієї функції створюються основні таблиці системи: користувачів, сесій, завдань та обліку часу виконання. Для кожної таблиці визначено первинні та зовнішні ключі, що забезпечує логічну зв’язаність даних і запобігає виникненню неузгоджених записів.

Додатково у модулі реалізовано механізм створення індексів для часто використовуваних полів, зокрема ідентифікаторів користувача та завдань. Це

дозволяє оптимізувати продуктивність SQL-запитів при роботі з великими обсягами даних.

Окремим елементом реалізації є функція первинного заповнення бази даних демонстраційними значеннями. Під час першого запуску системи автоматично створюється тестовий користувач із попередньо визначеними обліковими даними, а також формується набір прикладних завдань різних статусів і пріоритетів. Це дозволяє одразу перевірити роботу основних функціональних модулів застосунку без необхідності ручного введення даних.

Серверна логіка вебзастосунку побудована на основі механізму Server Actions, який є сучасним підходом Next.js до організації серверної взаємодії. Server Actions реалізуються як асинхронні функції з директивою "use server" і можуть викликатися безпосередньо з клієнтських компонентів без проміжного рівня API. Це дозволяє істотно спростити архітектуру застосунку, оскільки усувається необхідність у проєктуванні та підтримці окремого REST API, а також зменшується кількість точок взаємодії між клієнтом і сервером.

Додатковою перевагою такого підходу є автоматична серіалізація даних та тісна інтеграція з серверним середовищем виконання, що забезпечує більш безпечний і контрольований доступ до ресурсів системи.

Загальний принцип обробки запиту у межах серверних дій відображено на діаграмі послідовності (рис. 3.2). Вона демонструє повний цикл взаємодії між клієнтською частиною застосунку, серверними діями та базою даних.

Діаграма ілюструє типовий сценарій виконання операції, наприклад створення або оновлення завдання. Процес починається на стороні клієнта (React-компонента), який ініціює виклик серверної дії. Далі запит передається до Server Action, де першим етапом виконується перевірка автентифікації користувача через функцію getCurrentUser(), яка зчитує сесію з HTTP-only cookie та визначає ідентичність користувача.

У разі успішної авторизації серверна дія переходить до взаємодії з базою даних SQLite, де виконується відповідний SQL-запит. Важливим аспектом є те,

що всі операції додатково фільтруються за user_id, що забезпечує ізоляцію даних між користувачами та виключає можливість доступу до сторонніх записів.

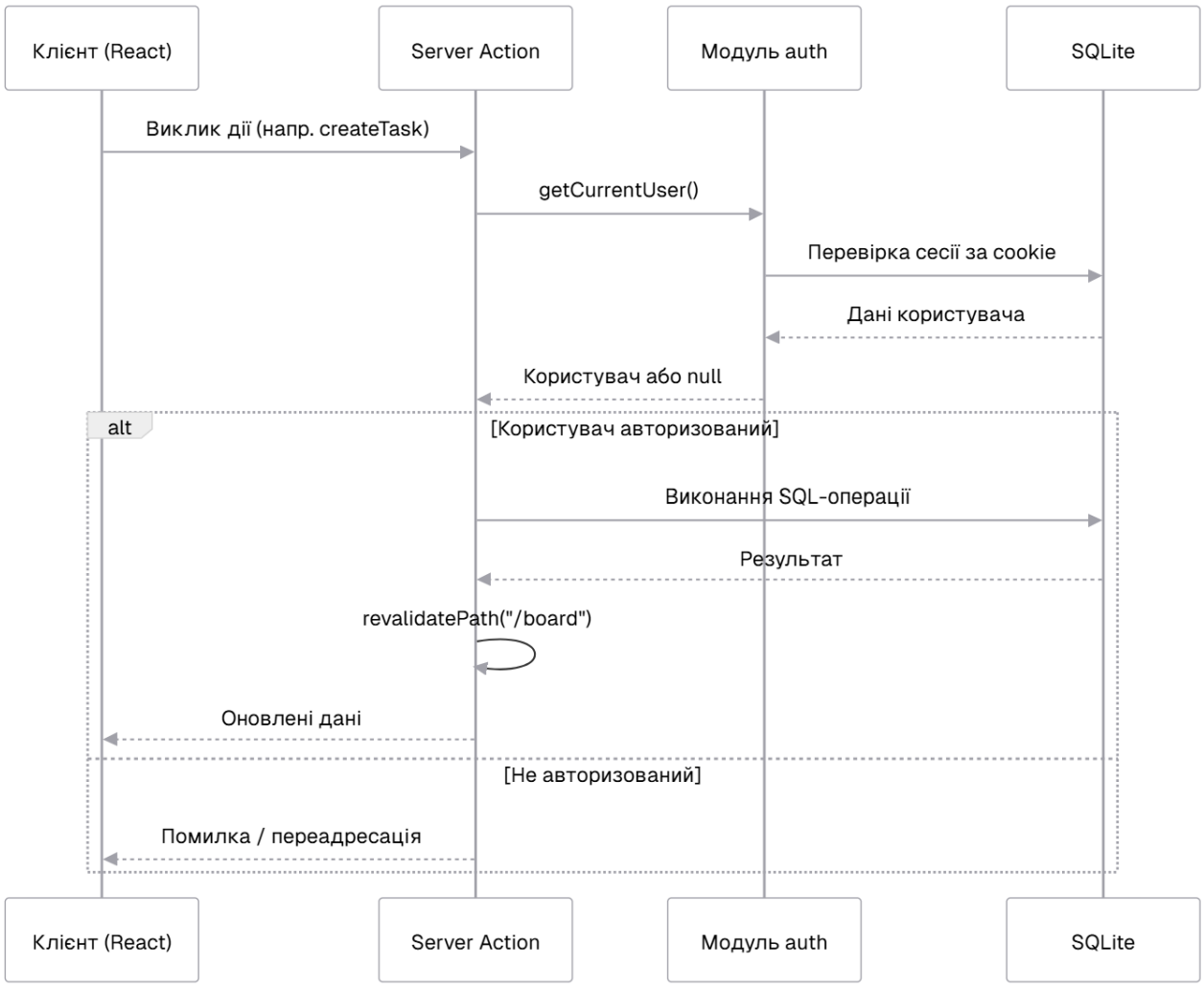


Рисунок 3.2 – Реалізація серверної логіки

Після виконання операції база даних повертає результат виконання, який обробляється серверною дією. У випадку змін даних додатково виконується виклик revalidatePath("/board"), що ініціює оновлення кешу та забезпечує синхронізацію інтерфейсу користувача з актуальним станом даних без необхідності перезавантаження сторінки.

У разі відсутності авторизації або неактивної сесії повертається значення null або виконується редирект на сторінку входу, що додатково підсилює рівень захисту системи.

Таким чином, представлена діаграма відображає повний цикл обробки запиту – від ініціації дії на клієнті до виконання операції в базі даних і оновлення інтерфейсу користувача, демонструючи цілісність та безпечність архітектури застосунку.

Окрему увагу приділено реалізації системи автентифікації користувачів, яка побудована на основі хешування паролів із використанням алгоритму scrypt. Для кожного користувача генерується унікальна сіль, що унеможливлює використання однакових хешів для однакових паролів.

Наведений у додатку А.2 програмний код реалізує повноцінний модуль автентифікації користувачів, а також механізм створення, перевірки та завершення сесій у вебзастосунку. Він є одним із ключових компонентів серверної частини системи, оскільки забезпечує контроль доступу до всіх функціональних можливостей застосунку та гарантує ізоляцію даних між користувачами. Основу модуля становлять криптографічні операції, реалізовані за допомогою вбудованого пакета node:crypto. Зокрема, для захисту паролів використовується алгоритм scrypt, який є стійким до атак перебору завдяки високій обчислювальній складності. Під час реєстрації кожного користувача генерується унікальна сіль (salt) за допомогою функції randomBytes, після чого пароль хешується та зберігається у базі даних у вигляді пари «хеш + сіль».

Функція hashPassword відповідає за первинне перетворення пароля у захищений формат, тоді як verifyPassword виконує перевірку введеного пароля під час авторизації. Для підвищення безпеки порівняння хешів здійснюється через timingSafeEqual, що унеможливлює атаки, пов’язані з аналізом часу виконання операцій. Реєстрація нового користувача реалізована у функції createUser. На цьому етапі виконується перевірка унікальності електронної пошти, після чого створюється новий запис у таблиці users із збереженням хешу пароля та службових даних. У разі спроби створення облікового запису з уже існуючою електронною адресою генерується помилка, що запобігає дублюванню користувачів.

Процес авторизації реалізований у функції authenticate, яка здійснює пошук користувача в базі даних за електронною поштою, після чого перевіряє коректність пароля. У разі успішної перевірки повертається об’єкт користувача, що надалі використовується для створення сесії. Механізм сесій реалізовано через функцію createSession. Після успішної автентифікації створюється унікальний ідентифікатор сесії (UUID), який зберігається у таблиці sessions разом із терміном її дії. Одночасно цей ідентифікатор записується в HTTP-only cookie, що забезпечує його недоступність для JavaScript-коду на клієнтській стороні та підвищує стійкість до XSS-атак. Термін дії сесії встановлено на 30 днів, що визначається константою SESSION_DURATION.

Функція getCurrentUser відповідає за визначення поточного авторизованого користувача. Вона зчитує ідентифікатор сесії з cookie, перевіряє його наявність у базі даних та контролює термін дії. У випадку завершення строку сесії вона автоматично видаляється, після чого доступ користувача вважається недійсним. Якщо сесія є валідною, повертаються базові дані користувача, необхідні для роботи застосунку. Завершення сесії реалізовано у функції destroySession, яка видаляє відповідний запис із бази даних та очищує cookie на стороні клієнта. Це забезпечує повний вихід користувача із системи та запобігає подальшому використанню старого ідентифікатора сесії.

Тобто структура бази даних включає таблиці користувачів, сесій, завдань та обліку часу виконання. Зв’язки між таблицями реалізовано через зовнішні ключі, що забезпечує цілісність даних та запобігає появі неузгоджених записів.

Особливістю реалізації є автоматичне створення демонстраційного користувача та тестового набору завдань під час першого запуску системи. Це дозволяє одразу оцінити функціональність застосунку без необхідності ручного введення даних.

Кожне завдання містить основні атрибути, такі як назва, опис, статус, пріоритет, позиція в списку, а також часові мітки створення та оновлення. Додатково реалізовано окрему таблицю для зберігання записів тайм-трекінгу, що дозволяє відстежувати тривалість роботи над кожним завданням.

Отже, реалізовано вебзастосунок відповідно до попередньо спроєктованої архітектури. Розробка виконувалась із використанням Next.js (App Router), що дозволило ефективно поєднати клієнтську та серверну логіку в єдиній структурі та забезпечити зручну організацію коду за модулями. Було реалізовано основні компоненти системи, зокрема структуру проєкту, механізм роботи з базою даних SQLite, серверну бізнес-логіку на основі Server Actions, а також модуль автентифікації користувачів і керування сесіями. Окрему увагу приділено безпеці, ізоляції даних та оптимізації взаємодії між клієнтом і сервером.

У результаті створено функціональний вебзастосунок, який забезпечує виконання основних вимог до системи та має потенціал для подальшого розширення і масштабування.

3.2 Розробка функціоналу канбан-дошки та тайм-трекінгу

Реалізація канбан-дошки та модуля тайм-трекінгу є ключовим функціональним ядром вебзастосунку, оскільки саме ці компоненти забезпечують інтерактивну роботу користувача із завданнями, їхнім статусом та часовими витратами. Архітектура побудована на поєднанні React-компонентів на клієнтській стороні та серверних дій (Server Actions), що дозволяє досягти балансу між продуктивністю інтерфейсу та узгодженістю даних у базі.

Канбан-дошка реалізована як багаторівнева система React-компонентів, де кожен компонент відповідає за окрему частину інтерфейсу та логіки (рис. 3.3). У центрі структури знаходиться компонент Board, який виконує роль головного контейнера стану. Він відповідає за зберігання списку завдань, управління їх статусами та координацію взаємодії між дочірніми компонентами. Саме через нього здійснюється передача даних до всіх елементів дошки та синхронізація змін із серверною частиною.

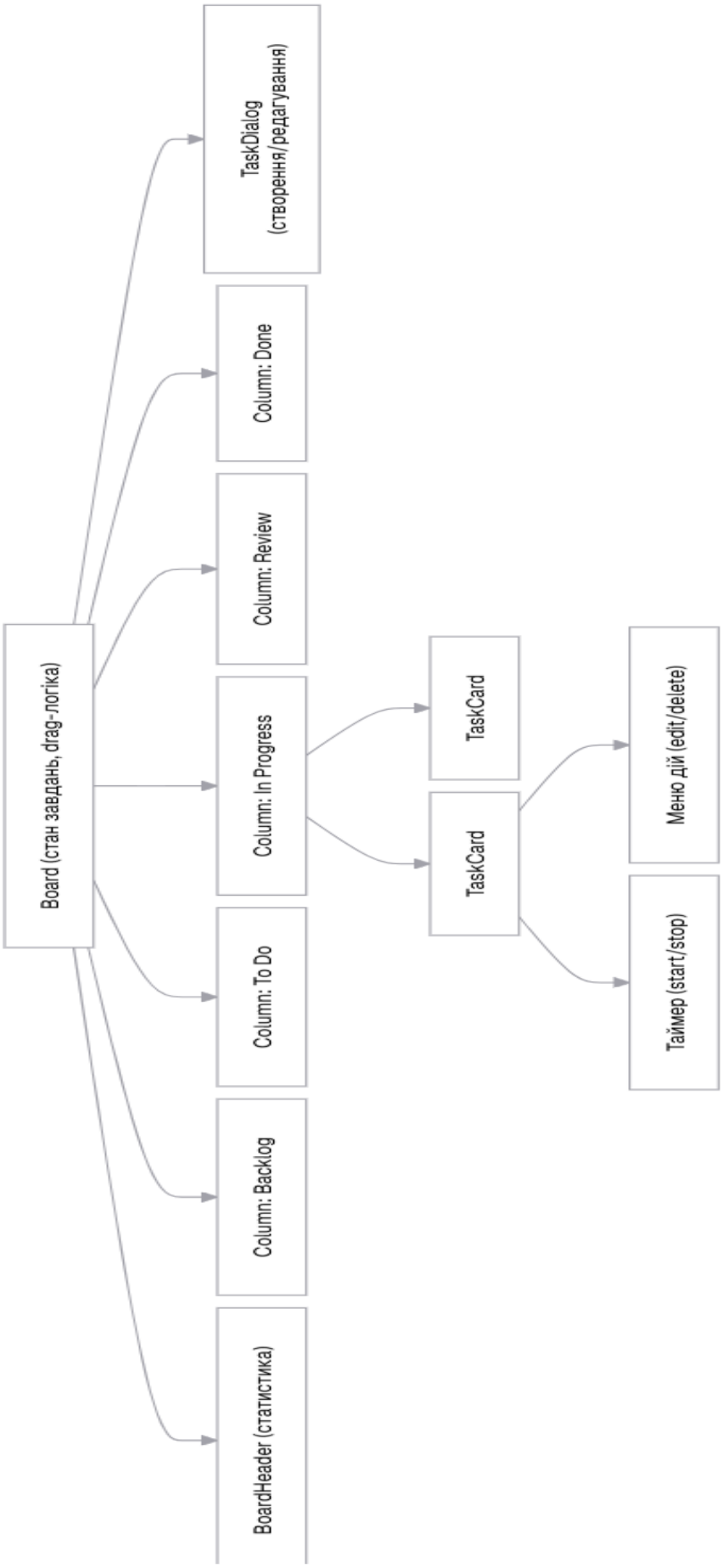


Рисунок 3.3 – Архітектура компонентів дошки

Безпосередньо під Board розташований компонент BoardHeader, який відображає узагальнену інформацію про стан системи, зокрема кількість завдань у різних статусах та загальну статистику виконання. Цей компонент не змінює дані напряму, а лише візуалізує агреговані показники.

Основну частину інтерфейсу формують п’ять колонок канбан-дошки: Backlog, To Do, In Progress, Review та Done. Кожна колонка виконує роль логічного контейнера для завдань певного статусу та одночасно є drop-зоною для механізму drag-and-drop. Це дозволяє користувачу змінювати стан завдання шляхом його переміщення між колонками.

Всередині колонок розміщуються компоненти TaskCard, які представляють окремі завдання. Картка містить основну інформацію про задачу, а також інтерактивні елементи керування. Зокрема, на рівні TaskCard реалізовано таймер (start/stop), який відповідає за облік часу роботи над завданням, та меню дій (edit/delete) для редагування або видалення запису.

Окремим допоміжним компонентом виступає TaskDialog, який використовується для створення та редагування завдань. Він відкривається як модальне вікно і дозволяє користувачу вводити або змінювати дані задачі без переходу на іншу сторінку. Механізм переміщення завдань між колонками реалізовано на основі нативного HTML5 Drag and Drop API без використання сторонніх бібліотек. Це дозволило зменшити залежності проєкту та забезпечити контроль над поведінкою інтерфейсу. Процес переміщення побудовано на принципі оптимістичного оновлення: після початку drag-and-drop стан інтерфейсу змінюється миттєво, а синхронізація з сервером виконується асинхронно. Це забезпечує відчуття швидкої взаємодії та покращує користувацький досвід. Алгоритм роботи включає кілька етапів: ініціалізація перетягування, збереження ідентифікатора задачі, визначення цільової колонки, оновлення локального стану та виклик серверної функції для збереження змін у базі даних. Після успішного виконання серверної операції виконується повторна синхронізація інтерфейсу через механізм revalidation сторінки. Окремо слід

зазначити, що перевірка зміни статусу дозволяє уникнути зайвих запитів, якщо завдання було відпущене у тій самій колонці, де вже знаходилося.

Функціонал тайм-трекінгу реалізовано як комбінацію серверної логіки та клієнтського оновлення інтерфейсу в реальному часі. Основою є таблиця time_entries, у якій зберігаються записи про початок і завершення роботи над завданням. Запуск таймера ініціює створення нового запису з полем started_at, тоді як зупинка призводить до оновлення цього запису та фіксації duration. Активна сесія визначається як запис, у якому поле ended_at є порожнім, що дозволяє точно відрізняти поточну роботу від завершеної.

Клієнтська частина таймера реалізована через періодичне оновлення стану за допомогою setInterval, що дозволяє відображати секунди в реальному часі без повторних запитів до сервера. Це знижує навантаження на базу даних і підвищує продуктивність системи.

Сумарний час виконання завдання розраховується як сума всіх завершених сесій плюс тривалість активної сесії, яка обчислюється на стороні клієнта. Такий підхід забезпечує точність обліку часу навіть у разі тривалих сесій роботи.

Для відображення тривалості роботи над завданнями реалізовано утиліту форматування часу, яка перетворює кількість секунд у зручний для сприйняття формат HH:MM:SS.

У підсумку, реалізований функціонал канбан-дошки та тайм-трекінгу забезпечує інтерактивне управління завданнями, їх гнучке переміщення між статусами та точний облік витраченого часу, що є основою продуктивної роботи в межах розробленого вебзастосунку.

3.3 Тестування системи та аналіз результатів роботи

Тестування розробленого вебзастосунку здійснювалося як комплексний процес, спрямований на перевірку коректності функціонування всіх основних модулів, взаємодії між клієнтською та серверною частинами, а також

відповідності реалізації нефункціональним вимогам. Загальна стратегія тестування включала функціональну, інтеграційну, безпекову, UI- та продуктивнісну перевірки, що дозволило всебічно оцінити якість системи в умовах, наближених до реального використання.

У межах функціонального тестування було сформовано набір сценаріїв, що охоплюють ключові користувацькі дії: реєстрацію, авторизацію, створення, редагування та видалення завдань, роботу з канбан-дошкою, а також використання тайм-трекінгу. Усі сценарії були успішно виконані, що підтвердило коректність реалізації бізнес-логіки та відповідність системи визначеним вимогам. Особливо важливо, що перевірка охоплювала як позитивні, так і негативні сценарії, зокрема спроби входу з некоректними даними та доступ до захищених сторінок без авторизації, які були коректно оброблені системою.

Окрему увагу було приділено безпековому тестуванню, оскільки система працює з персональними даними користувачів та сесіями. Перевірка показала, що паролі зберігаються виключно у хешованому вигляді із використанням алгоритму scrypt, що унеможливлює їх відновлення у відкритому вигляді. Сесійні дані захищені через httpOnly-cookie, що зменшує ризик їх викрадення на стороні клієнта. Крім того, доступ до даних жорстко обмежений за user_id, що гарантує ізоляцію інформації між користувачами та унеможливлює доступ до чужих записів. Також усі SQL-запити параметризовані, що виключає можливість SQL-ін’єкцій.

Під час тестування було виявлено кілька технічних проблем, серед яких некоректна серіалізація об’єктів, отриманих із node:sqlite, а також конфлікт вкладених інтерактивних елементів, що спричиняв помилки гідратації у React. Ці дефекти були оперативно усунуті шляхом приведення даних до звичайних JavaScript-об’єктів та рефакторингу компонентів інтерфейсу відповідно до вимог React щодо вкладеності елементів.

Аналіз продуктивності, виконаний на основі показників Core Web Vitals, продемонстрував стабільну та швидку роботу застосунку. Завдяки використанню оптимістичних оновлень інтерфейсу користувач отримує

миттєвий відгук на свої дії, тоді як серверні операції виконуються асинхронно у фоновому режимі. Додатково застосування серверного рендерингу та локальної бази даних SQLite забезпечує мінімальні затримки при завантаженні та обробці даних.

У результаті проведеного тестування встановлено, що розроблений вебзастосунок повністю відповідає поставленим функціональним і нефункціональним вимогам, демонструє стабільну роботу, належний рівень безпеки та високу продуктивність.

3.4 Інструкція користувача

Для роботи із вебзастосунком користувачу не потрібно встановлювати додаткове програмне забезпечення, оскільки система працює безпосередньо у сучасному вебпереглядачі. Достатньо мати актуальну версію браузера та стабільне підключення до інтернету, що забезпечує коректне завантаження інтерфейсу та взаємодію із серверною частиною системи.

Після відкриття застосунку користувач потрапляє на сторінку автентифікації, де може або увійти до вже створеного облікового запису, або пройти процедуру реєстрації. У випадку створення нового акаунта користувач вводить ім’я, електронну пошту та пароль, після чого система автоматично створює профіль і виконує вхід із перенаправленням на головну сторінку канбан-дошки. При повторних відвідуваннях достатньо ввести свої облікові дані для доступу до системи.

Основна робоча зона застосунку представлена канбан-дошкою, яка поділена на кілька статусних колонок, що відображають життєвий цикл завдань. Користувач може створювати нові завдання, редагувати або видаляти існуючі, а також переміщувати їх між колонками шляхом перетягування. Такий підхід забезпечує інтуїтивно зрозумілу взаємодію та дозволяє швидко змінювати статуси задач без додаткових форм або переходів між сторінками.

Окремим функціональним елементом є система тайм-трекінгу, інтегрована безпосередньо у картки завдань. Користувач може запускати та зупиняти відлік часу, при цьому кожна активна сесія фіксується у базі даних, а відображення часу оновлюється в режимі реального часу. Після зупинки таймера накопичений час зберігається та додається до загальної статистики виконання завдання, що дозволяє відстежувати продуктивність роботи над кожною задачею окремо.

Завершення роботи із системою здійснюється через вихід із облікового запису, після чого сесія користувача закривається, а доступ до захищених сторінок блокується до повторної авторизації. Такий підхід забезпечує безпеку даних і виключає можливість несанкціонованого доступу до робочого середовища.

ВИСНОВКИ

У межах кваліфікаційної роботи було здійснено комплексне проєктування, розробку та тестування вебзастосунку для управління особистими проєктами з використанням канбан-дошки та тайм-трекінгу. Отримані результати дозволяють стверджувати, що поставлена мета роботи досягнута повністю, а всі визначені завдання виконано у повному обсязі.

У першому розділі було розглянуто теоретичні основи створення вебзастосунків для управління проєктами та організації тайм-менеджменту. Було визначено сутність систем особистої продуктивності, їх роль у структуризації задач та підвищенні ефективності користувача. Окремо проаналізовано принципи роботи канбан-систем, які базуються на візуалізації робочого процесу, обмеженні одночасних завдань та поступовому переміщенні задач між стадіями виконання. Також розглянуто сучасні інструменти тайм-трекінгу, що дозволяють фіксувати витрачений час і формувати об’єктивну картину продуктивності. На основі проведеного аналізу було сформовано вимоги до майбутнього застосунку, визначено його основні функціональні можливості, а також окреслено ключові нефункціональні вимоги, серед яких продуктивність, безпека, зручність використання та масштабованість.

У другому розділі виконано проєктування вебзастосунку, яке охоплює як архітектурні рішення, так і логічну структуру системи. Було проаналізовано предметну область та обґрунтовано вибір клієнт-серверної архітектури, що забезпечує розділення відповідальності між інтерфейсом користувача та серверною логікою. Спроєктовано базу даних, яка включає основні сутності: користувачів, сесії, завдання та записи тайм-трекінгу. Визначено зв’язки між таблицями, що забезпечують цілісність даних та підтримку принципів ізоляції користувацької інформації. Також розроблено структуру основних функціональних модулів, зокрема канбан-дошки, системи автентифікації та модуля обліку часу. Особливу увагу приділено проєктуванню користувацького

інтерфейсу, який забезпечує інтуїтивну взаємодію з системою, а також обґрунтовано вибір сучасного технологічного стеку, що дозволив досягти балансу між продуктивністю, простотою розробки та безпекою.

Третій розділ присвячено реалізації та тестуванню розробленого вебзастосунку. Було реалізовано серверну логіку на основі Server Actions, що дозволило значно спростити взаємодію між клієнтською та серверною частинами системи та уникнути необхідності створення окремого REST API. Реалізовано основні функції системи, включаючи керування завданнями, drag-and-drop переміщення карток між колонками канбан-дошки, а також систему тайм-трекінгу з можливістю запуску та зупинки обліку часу. Окремо реалізовано механізм збереження історії часових записів, що дозволяє аналізувати витрати часу на виконання конкретних завдань.

Проведене тестування охоплювало функціональні, інтеграційні, безпекові та продуктивнісні перевірки. Результати підтвердили коректну роботу всіх основних функцій системи та відповідність встановленим вимогам, зокрема надійність автентифікації, ізоляцію даних між користувачами, коректність SQL-запитів і серверної логіки, а також швидкий відгук інтерфейсу завдяки оптимістичним оновленням.

У результаті було створено повноцінний вебзастосунок, який поєднує канбан-дошку та тайм-трекінг у єдиному середовищі. Система дозволяє ефективно організовувати проєкти, візуалізувати робочий процес, контролювати виконання завдань і відстежувати витрачений час, що сприяє підвищенню особистої продуктивності та покращенню навичок планування.

Серед напрямів подальшого розвитку системи можна виділити впровадження аналітики продуктивності, розширення командної роботи, підтримку спільних проєктів та інтеграцію із зовнішніми сервісами. Це дозволить перетворити застосунок на повноцінну платформу для управління проєктами різного масштабу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Шведа Н. М. Система управління проектами в Україні. *Актуальні задачі сучасних технологій : зб. тез доповідей міжнар. наук.-техн. конф. молодих учених та студентів (Тернопіль, 25–26 листоп. 2015).* Тернопіль : ТНТУ, 2015. С. 246–247.

2. Ferreira L., Fortes P., De Castro Lobo Dos Santos H. Gestão de projetos pessoais: um framework para organizar a vida e concretizar objetivos. *Research Society and Development.* 2020. Vol. 9, no. 11. e62991110361. URL : https://doi.org/10.33448/rsd-v9i11.10361 (дата звернення : 20.03.2026).

3. Bittnet Training. 6 найкращих програм для управління особистими проектами у 2026 році. URL : https://www.bittnet.ro/uk/noutati/top-6-aplicatii-de-gestionare-proiecte-personale-in-2026/ (дата звернення : 20.03.2026).

4. KanbanTool. History of Kanban. URL : https://kanbantool.com/kanban-guide/kanban-history (дата звернення : 20.03.2026).

5. Arda Cards. History of Kanban: From Toyota’s Factory Floor to Modern Manufacturing. URL : https://www.arda.cards/post/a-deep-dive-into-the-history-of-kanban-from-toyotas-factory-floor-to-modern-manufacturing (дата звернення : 20.03.2026).

6. Wezom. Канбан Board: принципи роботи та переваги для команди. URL : https://wezom.com.ua/ua/blog/kanban-scho-tse-take-ta-yak-pratsyuje-sistema (дата звернення : 20.03.2026).

7. Buehring S. Trello Kanban. agileKRC. URL : https://agilekrc.com/agile/kanban/kanban-tools-and-software/trello-kanban (дата звернення : 20.03.2026).

8. Tristancho C. Trello Kanban Board: Key Features, Pros, Cons & Alternatives. ProjectManager. URL : https://www.projectmanager.com/blog/trello-kanban-board (дата звернення : 20.03.2026).

9. TaskRhino. Trello Review 2026: Expert Analysis & Feature Breakdown. URL : https://www.taskrhino.ca/blog/trello-review/ (дата звернення : 20.03.2026).

10. Vact. Jira vs. Asana vs. Monday.com: Which PM Tool Is Right for You? URL : https://vact.com/jira-vs-asana-vs-monday-comparison/ (дата звернення : 20.03.2026).

11. Getint. Asana vs Monday: Comparison. URL : https://www.getint.io/blog/asana-vs-monday-comparison (дата звернення : 20.03.2026).

12. Productive Temply. Notion vs ClickUp. URL : https://www.productivetemply.com/blog/notion-vs-clickup (дата звернення : 20.03.2026).

13. Everhour. ClickUp vs Todoist: Breakdown for Project Management & Time Tracking. URL : https://everhour.com/blog/clickup-vs-todoist/ (дата звернення : 20.03.2026).

14. Next.js by Vercel – the React framework. URL : https://nextjs.org/ (дата звернення : 20.03.2026).

15. React. URL : https://react.dev/ (дата звернення : 20.03.2026).

16. Vue.js. The Progressive JavaScript Framework. URL : https://vuejs.org/ (дата звернення : 20.03.2026).

17. Nuxt. The Full-Stack Vue Framework. URL : https://nuxt.com/ (дата звернення : 20.03.2026).

18. TypeScript: JavaScript with Syntax for Types. URL : https://www.typescriptlang.org/ (дата звернення : 20.03.2026).

19. SQLite Documentation. URL : https://sqlite.org/docs.html (дата звернення : 20.03.2026).

20. PostgreSQL Documentation. URL : https://www.postgresql.org/ (дата звернення : 20.03.2026).

21. MongoDB. The world’s leading modern data platform. URL : https://www.mongodb.com/ (дата звернення : 20.03.2026).

22. Node.js – Run JavaScript everywhere. URL : https://nodejs.org/en (дата звернення : 20.03.2026).

23. Tailwind CSS Documentation. URL : https://tailwindcss.com/ (дата звернення : 20.03.2026).

ДОДАТОК А

ЛІСТИНГ РЕАЛІЗАЦІЇ ВЕБЗАСТОСУНКУ

Лістинг А.1 – db.ts

```
import { DatabaseSync } from "node:sqlite"
import { randomBytes, scryptSync, randomUUID } from "node:crypto"
import path from "node:path"
import fs from "node:fs"

// Demo credentials shown on the login screen.
export const DEMO_EMAIL = "demo@flowdesk.app"
export const DEMO_PASSWORD = "demo1234"

// Persist the SQLite database file in a writable data directory.
const dataDir = path.join(process.cwd(), "data")
if (!fs.existsSync(dataDir)) {
  fs.mkdirSync(dataDir, { recursive: true })
}
const dbPath = path.join(dataDir, "app.db")

// Reuse a single connection across hot reloads in development.
const globalForDb = globalThis as unknown as { __db?: DatabaseSync }

function createDb(): DatabaseSync {
  const database = new DatabaseSync(dbPath)
  database.exec("PRAGMA journal_mode = WAL;")
```

```
  database.exec("PRAGMA foreign_keys = ON;")
  migrate(database)
  seedDemo(database)
  return database
}

function migrate(database: DatabaseSync) {
  database.exec(`
    CREATE TABLE IF NOT EXISTS users (
      id TEXT PRIMARY KEY,
      email TEXT NOT NULL UNIQUE,
      name TEXT NOT NULL,
      password_hash TEXT NOT NULL,
      salt TEXT NOT NULL,
      created_at INTEGER NOT NULL
    );

    CREATE TABLE IF NOT EXISTS sessions (
      id TEXT PRIMARY KEY,
      user_id TEXT NOT NULL,
      expires_at INTEGER NOT NULL,
      created_at INTEGER NOT NULL,
      FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE
CASCADE
    );

    CREATE TABLE IF NOT EXISTS tasks (
      id TEXT PRIMARY KEY,
      user_id TEXT NOT NULL,
      title TEXT NOT NULL,
```

```
        description TEXT NOT NULL DEFAULT ‘‘,
        status TEXT NOT NULL DEFAULT ‘backlog’,
        priority TEXT NOT NULL DEFAULT ‘medium’,
        position INTEGER NOT NULL DEFAULT 0,
        created_at INTEGER NOT NULL,
        updated_at INTEGER NOT NULL,
        FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE
CASCADE
      );

      CREATE TABLE IF NOT EXISTS time_entries (
        id TEXT PRIMARY KEY,
        task_id TEXT NOT NULL,
        user_id TEXT NOT NULL,
        started_at INTEGER NOT NULL,
        ended_at INTEGER,
        duration INTEGER,
        FOREIGN KEY (task_id) REFERENCES tasks(id) ON DELETE
CASCADE,
        FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE
CASCADE
      );

      CREATE INDEX IF NOT EXISTS idx_tasks_user ON tasks(user_id);
      CREATE INDEX IF NOT EXISTS idx_sessions_user ON sessions(user_id);
      CREATE INDEX IF NOT EXISTS idx_time_entries_task ON
time_entries(task_id);
      CREATE INDEX IF NOT EXISTS idx_time_entries_user ON
time_entries(user_id);
    `)
```

```
}

export const db: DatabaseSync = globalForDb.__db ?? createDb()
if (process.env.NODE_ENV !== "production") {
  globalForDb.__db = db
}

// Create a ready-to-use demo account with sample tasks on first run.
function seedDemo(database: DatabaseSync) {
  const exists = database.prepare("SELECT id FROM users WHERE email = ?").get(DEMO_EMAIL) as
    | { id: string }
    | undefined
  if (exists) return

  const salt = randomBytes(16).toString("hex")
  const hash = scryptSync(DEMO_PASSWORD, salt, 64).toString("hex")
  const userId = randomUUID()
  const now = Date.now()
  database
    .prepare(
      "INSERT INTO users (id, email, name, password_hash, salt, created_at) VALUES (?, ?, ?, ?, ?, ?)",
    )
    .run(userId, DEMO_EMAIL, "Демо користувач", hash, salt, now)

  const sample: Array<{ title: string; description: string; status: string; priority: string }> = [
    { title: "Дослідити вимоги проєкту", description: "Зібрати та проаналізувати вимоги", status: "done", priority: "high" },
```

```
    { title: "Спроєктувати базу даних", description: "ER-діаграма та схема таблиць", status: "done", priority: "medium" },
    { title: "Реалізувати канбан-дошку", description: "Drag-and-drop між колонками", status: "in_progress", priority: "high" },
    { title: "Додати тайм-трекінг", description: "Таймер старт/стоп на картках", status: "in_progress", priority: "medium" },
    { title: "Написати тести", description: "Функціональне та інтеграційне тестування", status: "review", priority: "medium" },
    { title: "Підготувати документацію", description: "Інструкція користувача", status: "todo", priority: "low" },
    { title: "Оптимізувати продуктивність", description: "Перевірити Web Vitals", status: "backlog", priority: "low" },
  ]

  const insert = database.prepare(
    "INSERT INTO tasks (id, user_id, title, description, status, priority, position, created_at, updated_at) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)",
  )
  sample.forEach((t, i) => {
    insert.run(randomUUID(), userId, t.title, t.description, t.status, t.priority, i, now, now)
  })
}
```

Лістинг А.2 – auth.ts

```
import { randomBytes, scryptSync, timingSafeEqual, randomUUID } from "node:crypto"
import { cookies } from "next/headers"
```

```
import { db } from "./db"

const SESSION_COOKIE = "session_id"
const SESSION_DURATION = 1000 * 60 * 60 * 24 * 30 // 30 days

export type User = {
  id: string
  email: string
  name: string
}

export function hashPassword(password: string): { hash: string; salt: string } {
  const salt = randomBytes(16).toString("hex")
  const hash = scryptSync(password, salt, 64).toString("hex")
  return { hash, salt }
}

export function verifyPassword(password: string, hash: string, salt: string): boolean {
  const candidate = scryptSync(password, salt, 64)
  const stored = Buffer.from(hash, "hex")
  if (candidate.length !== stored.length) return false
  return timingSafeEqual(candidate, stored)
}

export function createUser(email: string, name: string, password: string): User {
  const existing = db.prepare("SELECT id FROM users WHERE email = ?").get(email.toLowerCase())
  if (existing) {
    throw new Error("Користувач з таким email вже існує")
```

```
  }
  const { hash, salt } = hashPassword(password)
  const id = randomUUID()
  db.prepare(
    "INSERT INTO users (id, email, name, password_hash, salt, created_at) VALUES (?, ?, ?, ?, ?, ?)",
  ).run(id, email.toLowerCase(), name, hash, salt, Date.now())
  return { id, email: email.toLowerCase(), name }
}

export function authenticate(email: string, password: string): User | null {
  const row = db
    .prepare("SELECT id, email, name, password_hash, salt FROM users WHERE email = ?")
    .get(email.toLowerCase()) as
    | { id: string; email: string; name: string; password_hash: string; salt: string }
    | undefined
  if (!row) return null
  if (!verifyPassword(password, row.password_hash, row.salt)) return null
  return { id: row.id, email: row.email, name: row.name }
}

export async function createSession(userId: string) {
  const id = randomUUID()
  const now = Date.now()
  const expiresAt = now + SESSION_DURATION
  db.prepare("INSERT INTO sessions (id, user_id, expires_at, created_at) VALUES (?, ?, ?, ?)").run(
    id,
    userId,
```

```
    expiresAt,
    now,
  )
  const cookieStore = await cookies()
  cookieStore.set(SESSION_COOKIE, id, {
    httpOnly: true,
    secure: process.env.NODE_ENV === "production",
    sameSite: "lax",
    expires: new Date(expiresAt),
    path: "/",
  })
}

export async function getCurrentUser(): Promise<User | null> {
  const cookieStore = await cookies()
  const sessionId = cookieStore.get(SESSION_COOKIE)?.value
  if (!sessionId) return null

  const session = db
    .prepare("SELECT user_id, expires_at FROM sessions WHERE id = ?")
    .get(sessionId) as { user_id: string; expires_at: number } | undefined

  if (!session) return null
  if (session.expires_at < Date.now()) {
    db.prepare("DELETE FROM sessions WHERE id = ?").run(sessionId)
    return null
  }

  const user = db.prepare("SELECT id, email, name FROM users WHERE id =
?").get(session.user_id) as
```

```
    | User
    | undefined
  // node:sqlite returns null-prototype objects; convert to a plain object.
  return user ? { id: user.id, email: user.email, name: user.name } : null
}

export async function destroySession() {
  const cookieStore = await cookies()
  const sessionId = cookieStore.get(SESSION_COOKIE)?.value
  if (sessionId) {
    db.prepare("DELETE FROM sessions WHERE id = ?").run(sessionId)
    cookieStore.delete(SESSION_COOKIE)
  }
}
```

ДОДАТОК Б

ІНТЕРФЕЙС ВЕБЗАТОСУНКУ

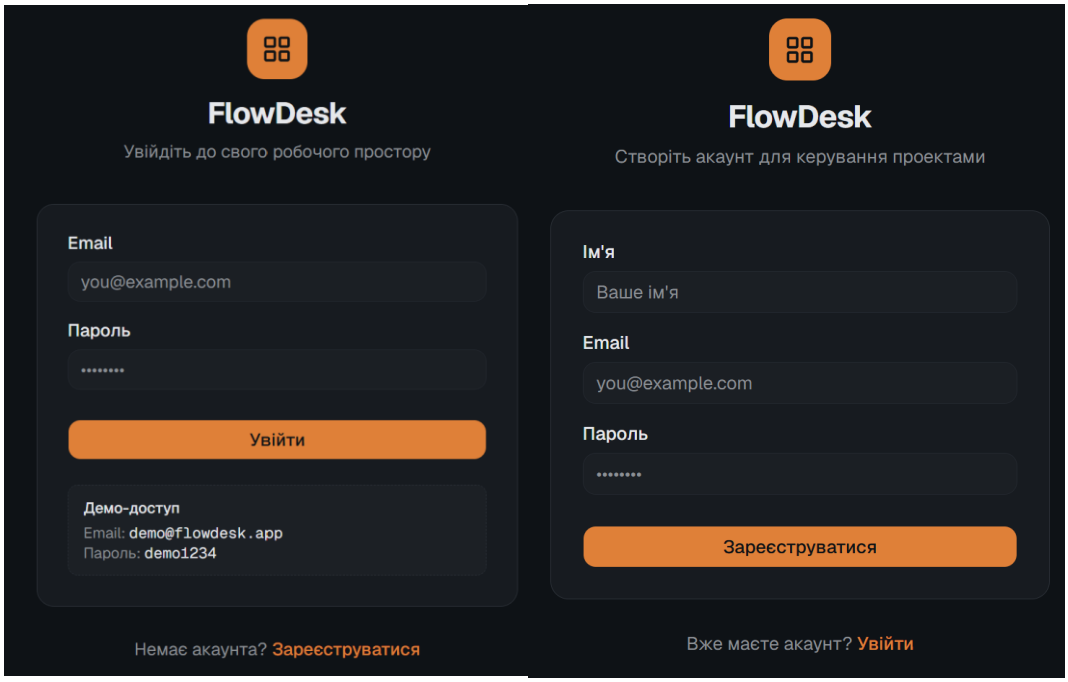


Рисунок Б.1 – Сторінка входу / реєстрації

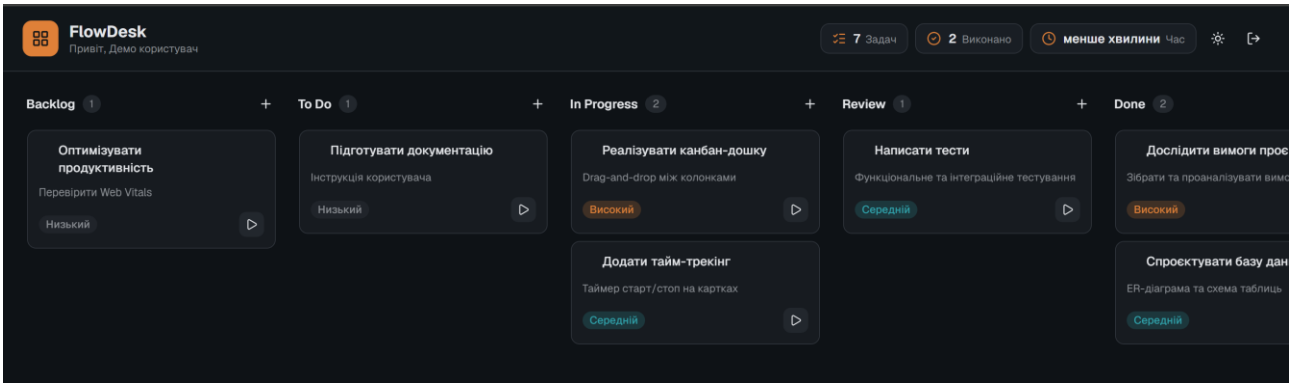


Рисунок Б.2 – Канбан-дошка

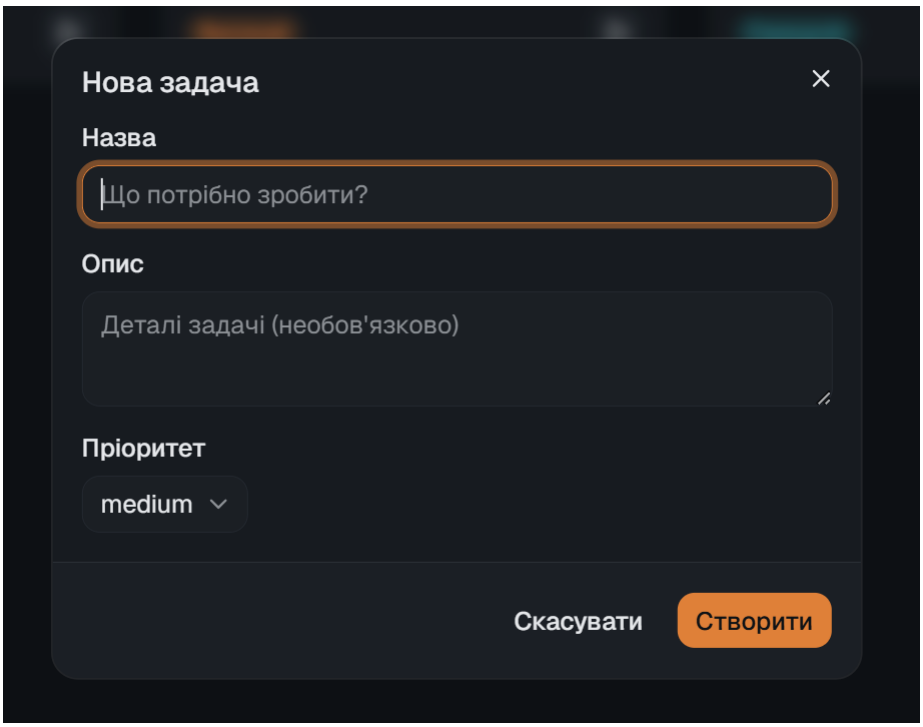


Рисунок Б.3 – Вікно додавання завдання

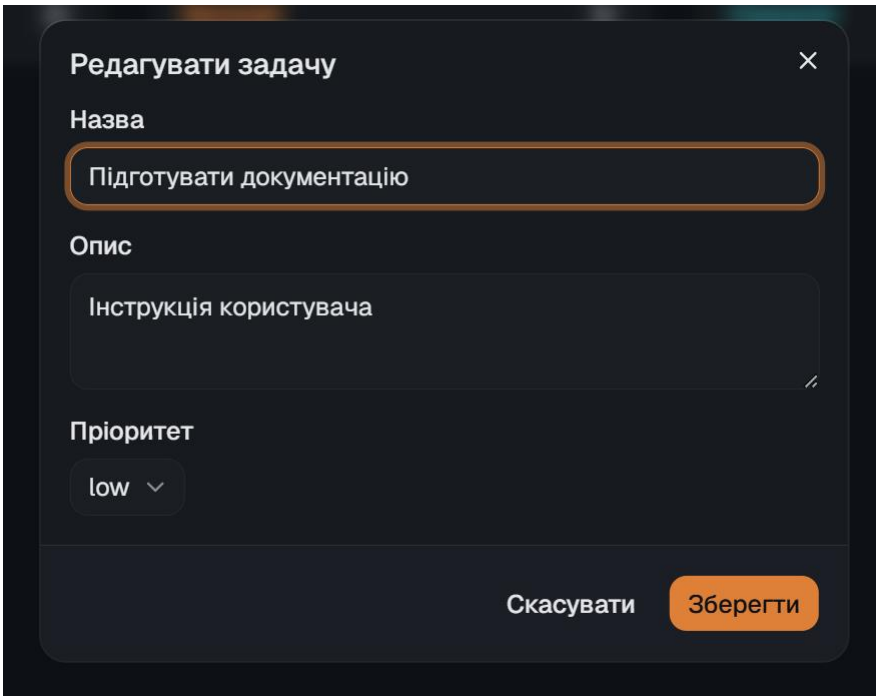


Рисунок Б.4 – Вікно редагування завдання

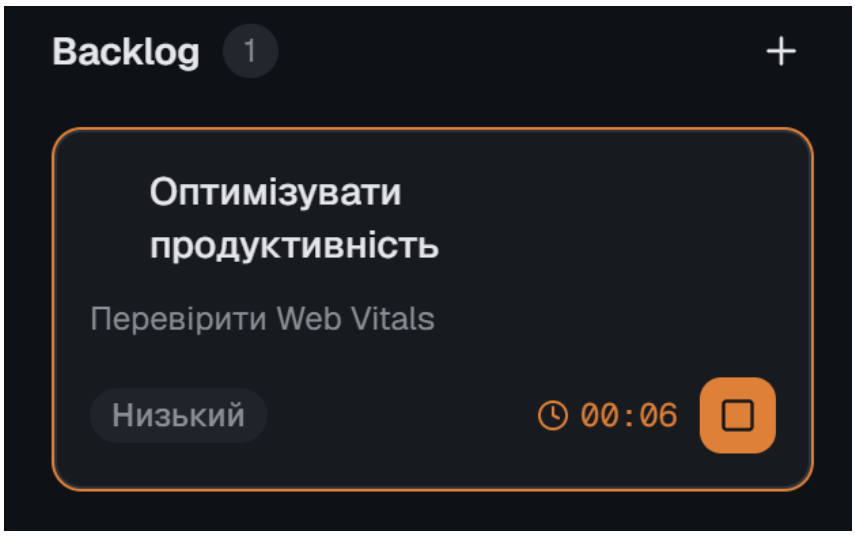


Рисунок Б.5 –Таймер виконання завдання

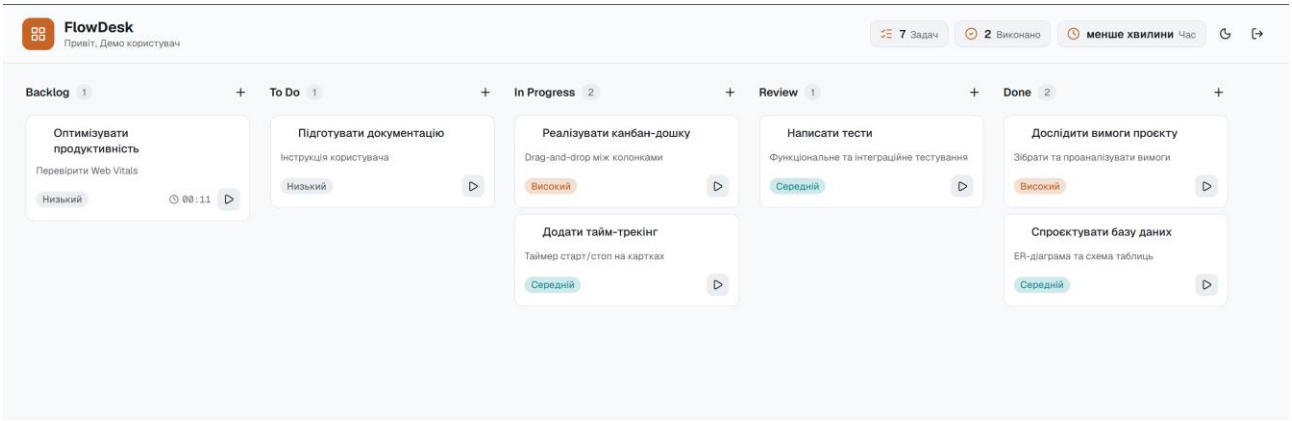


Рисунок Б.6 – Канбан-дошка (світла тема)