Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Онлайн планувальник завдань (To-Do-List)

Виконав(ла): студент(ка) IV курсу, групи СПс-41
спеціальності 121– Інженерія програмного забезпечення
(шифр і назва спеціальності)

_____ Теслюк В.О.
(підпис) (прізвище та ініціали)

Керівник _____ Тимків П. О.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю. М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М. Р.
(підпис) (прізвище та ініціали)

Рецензент _____ Тиш Є. В.
(підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Петрик М.Р.
(підпис) (прізвище та ініціали)
« 6 » квітня 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)
за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
студенту Теслюку Василю Олеговичу

1. Тема роботи Онлайн планувальник завдань (To-Do-List)

Керівник роботи доц., к. т. н. Тимків Павло Олександрович
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом по університету від « 06 » квітня 2026 року № 4/9-172
2. Термін подання студентом роботи 22.06.2026
3. Вихідні дані до роботи наукові літературні джерела

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Вступна частина
1. Аналіз предметної області та постановка задачі
2. Проєктування архітектури системи та бази даних
3. Тестування функціоналу та продуктивності
4. Безпека життєдіяльності, основи охорони праці.
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень, слайдів)
1. Тема роботи. 2. Актуальність, мета та завдання дослідження.
3. Аналіз існуючих аналогів.
4. Функціональні та нефункціональні вимоги .5. Загальна архітектура системи.
6. Діаграми класів та варіантів використання.
7. Технологічний стек розробки. 8. Засоби реалізації інтерфейсу системи.
9. Тестування та впровадження. 10. Висновки. 12. Слайди презентації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Мариненко С. Ю. канд. тех. наук, доцент		
Нормоконтроль	Стоянов Ю.М. доц. каф. ПІ		

7. Дата видачі завдання 6 квітня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Розробка технічного завдання*	*6.04 – 12.04*	
2.	*Робота над першим розділом «Аналіз вимог до програмної системи»*	*13.04 – 26.04*	
3.	*Робота над другим розділом «Проєктування та розробка системи»*	*27.04 – 03.05*	
4.	*Робота над третім розділом «Тестування та впровадження»*	*04.05 – 17.05*	
5.	*Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»*	*18.05 – 24.05*	
6.	*Оформлення пояснювальної записки і графічного матеріалу*	*25.05 – 7.06*	
7.	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*8.06 – 14.06*	
8.	*Попередній захист кваліфікаційної роботи бакалавра*	*15.06 – 21.06*	
9.	*Захист кваліфікаційної роботи бакалавра*		

Студент _____ Теслюк В. О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Тимків П. О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Теслюк В. О. Онлайн планувальник завдань (To-Do-List) : робота на здобуття кваліфікаційного ступеня бакалавра : спец. 121 - інженерія програмного забезпечення / наук. кер. Ю. М. Стоянов. Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2026. 105 с. // С. – 105, табл. – 7, рис. – 59, бібліогр. – 21, слайд. – 19, додат. – 0.

Ключові слова: інженерія програмного забезпечення, веб-застосунок, управління завданнями, Next.js, Supabase, Tailwind CSS, TypeScript.

Метою роботи є розробка повнофункціонального веб-застосунку для управління особистими завданнями з підтримкою категорій, пріоритетів, нагадувань та аналітики продуктивності.

У першому розділі проведено аналіз предметної області управління завданнями, досліджено існуючі аналоги (Google Tasks, Trello, Notion), сформульовано функціональні та нефункціональні вимоги, обґрунтовано вибір технологічного стеку.

У другому розділі спроєктовано архітектуру системи, побудовано діаграму класів, ER-діаграму бази даних та діаграми послідовностей. Описано розробку інтерфейсу користувача, клієнтської та серверної частин, механізми інтеграції з Supabase.

У третьому розділі проведено функціональне тестування за 30 сценаріями, виконано вимірювання продуктивності засобами Lighthouse, розглянуто можливості подальшого розширення системи.

У четвертому розділі розглянуто питання безпеки життєдіяльності в умовах воєнного стану та охорони праці оператора комп'ютерного обладнання.

Об'єктом дослідження є процес управління особистими завданнями та робочим часом.

Предметом дослідження є методи, моделі та технології розробки веб-орієнтованої системи для планування завдань із використанням Next.js, Supabase та Tailwind CSS.

ABSTRACT

Teslyk V. O. Online Task Planner (TaskFlow) : Bachelor's qualification thesis : specialty 121 – Software Engineering / supervisor Yu. M. Stoianov. Ternopil : Ternopil Ivan Puluj National Technical University, 2026. 105 p. // Pp. – 105, tables – 7, figures – 59, references – 21, slides – 19, appendices – 0.

Keywords: software engineering, web application, task management, Next.js, Supabase, Tailwind CSS, TypeScript. The aim of the thesis is to develop a fully functional web application for personal task management with support for categories, priorities, reminders, and productivity analytics.

The first chapter analyzes the subject area of task management, examines existing solutions (Google Tasks, Trello, Notion), formulates functional and non-functional requirements, and justifies the choice of the technology stack.

The second chapter presents the system architecture design, including a class diagram, an ER database diagram, and sequence diagrams. The user interface design, client-side and server-side development, and Supabase integration mechanisms are described.

The third chapter covers functional testing with 30 test scenarios and performance measurements using Lighthouse. Opportunities for further system expansion are considered.

The fourth chapter addresses life safety issues under martial law conditions and occupational safety for computer equipment operators.

The object of the research is the process of personal task and time management.

The subject of the research is methods, models, and technologies for developing a web-based task planning system using Next.js, Supabase, and Tailwind CSS.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface (прикладний програмний інтерфейс).

BaaS – Backend-as-a-Service (бекенд як сервіс).

CSS – Cascading Style Sheets (каскадні таблиці стилів).

CSR – Client-Side Rendering (рендеринг на стороні клієнта).

ER – Entity-Relationship (модель «сутність-зв'язок»).

FID – First Input Delay (затримка першої взаємодії).

FK – Foreign Key (зовнішній ключ).

GTD – Getting Things Done (методологія управління завданнями).

JWT – JSON Web Token (веб-токен у форматі JSON).

LCP – Largest Contentful Paint (відображення найбільшого елемента).

PK – Primary Key (первинний ключ).

PWA – Progressive Web Application (прогресивний веб-застосунок).

RLS – Row Level Security (безпека на рівні рядків).

RSC – React Server Components (серверні компоненти React).

SQL – Structured Query Language (мова структурованих запитів).

SSR – Server-Side Rendering (рендеринг на стороні сервера).

TTFB – Time to First Byte (час до отримання першого байта).

UI – User Interface (користувацький інтерфейс).

UML – Unified Modeling Language (уніфікована мова моделювання).

БД – база даних.

ВДТ – візуальний дисплейний термінал.

ПК – персональний комп’ютер.

ДСНС – Державна служба України з надзвичайних ситуацій.

ЗМІСТ

ВСТУП

Цифрове середовище диктує жорсткі вимоги до самоорганізації. Щоденний потік завдань зростає – робочі проєкти, навчальні зобов'язання, особисті справи накладаються одне на одне, створюючи хаос. Без чіткої системи планування людина втрачає контроль над дедлайнами, забуває про пріоритети, витрачає час на другорядне. Саме тому інструменти для управління завданнями перетворилися з додаткового сервісу на базову потребу.

Популярні рішення – Todoist, Trello, Google Tasks – пропонують широкий функціонал. Однак безкоштовні версії цих продуктів суттєво обмежені: частина можливостей заблокована, аналітика недоступна, українська локалізація відсутня або фрагментарна. Повноцінне використання вимагає платної підписки. Це створює запит на власну розробку – легкий, швидкий веб-застосунок із розширеною аналітикою, який не потребує оплати за базову функціональність.

Актуальність теми підсилюється трендом на віддалену роботу. Люди дедалі частіше самостійно розподіляють робочий час, і навички тайм-менеджменту стають критичними. Методологія GTD Девіда Аллена, матриця Ейзенхауера, техніка Pomodoro – ці підходи вимагають інструментальної підтримки. Зручний планувальник не просто фіксує список справ. Він формує звичку до системного мислення, допомагає аналізувати власну продуктивність і коригувати стратегію роботи.

Мета роботи – розробити веб-застосунок «Онлайн планувальник завдань» (TaskFlow) із функціями створення, редагування, категоризації та аналітики завдань, системою пріоритетів, статусів і нагадувань.

Для досягнення мети поставлено такі завдання**:**

1. Проаналізувати предметну область, дослідити існуючі аналоги, виявити їх переваги та недоліки.
2. Сформулювати функціональні та нефункціональні вимоги до системи.
3. Обрати технологічний стек, спроєктувати архітектуру застосунку та структуру бази даних.
4. Реалізувати клієнтську та серверну частини.
5. Провести тестування функціоналу, оцінити відповідність системи вимогам.
6. Розробити інструкцію користувача.

Об'єкт дослідження - процес управління особистими завданнями та робочим часом.

Предмет дослідження - веб-орієнтована програмна система для планування завдань із використанням технологій Next.js, Supabase, Tailwind CSS.

Методи дослідження**:** аналіз науково-технічної літератури, порівняльний аналіз програмних аналогів, методи об'єктно-орієнтованого проєктування, методи веб-розробки, методи функціонального тестування.

Практичне значення одержаних результатів полягає в готовому до використання програмному продукті, який може застосовуватися для особистого планування, а також слугувати основою для подальшого розширення – додавання колективного режиму, інтеграції із зовнішніми календарями, створення мобільної версії.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

У цьому розділі проведено аналіз предметної області управління завданнями, досліджено існуючі програмні аналоги, сформульовано постановку задачі, визначено функціональні та нефункціональні вимоги до системи, обґрунтовано вибір технологічного стеку та архітектурних рішень.

1.1 Аналіз предметної області

Предметна область онлайн-планувальника завдань охоплює організацію, збереження, обробку та аналіз особистих і робочих задач користувача. Щоденна діяльність людини складається з десятків різнорідних завдань. Вони відрізняються пріоритетами, термінами виконання, категоріями, рівнем складності. Без належної систематизації накопичення задач призводить до втрати контролю над робочим процесом, зниження продуктивності та пропуску важливих дедлайнів [1]. Інструменти управління завданнями – не допоміжний сервіс, а необхідна складова роботи як окремих фахівців, так і команд.

Завдання є основною одиницею даних. Воно описує дію, що має бути виконана, та містить набір характеристик: назву, опис, категорію, пріоритет, статус виконання, термін завершення. Користувач створює, редагує та видаляє завдання, контролює їх виконання, змінює статуси залежно від прогресу.

Оскільки в реальній діяльності людина працює з різними напрямами – навчання, робота, побут, – необхідно групувати задачі за категоріями. Категорії пришвидшують орієнтування у великому списку, підвищують структурованість, дозволяють будувати статистику за напрямами.

Під час планування важливо розрізняти завдання за рівнем важливості (низький, середній, високий) та поточним станом (очікує, у процесі, виконано). Це дає змогу визначати першочергові задачі та відстежувати прогрес [2].

Термін виконання – критичний параметр задачі. Він формує вибірку завдань на сьогодні, на завтра, на майбутній період, виявляє прострочені дії.

Система зобов'язана автоматично аналізувати дати та попереджати користувача про актуальні терміни. На цьому будується система нагадувань.

Планувальник має вчасно інформувати про наближення дедлайнів та прострочення. Для великих списків це критично: без автоматичного нагадування користувач пропускає важливі дати.

Предметна область передбачає не лише зберігання задач, а й аналіз виконаної роботи. Потрібна інформація про кількість завершених задач, динаміку активності за тиждень, розподіл за пріоритетами, статусами, категоріями. Така статистика допомагає оцінити власну продуктивність, виявити сильні та слабкі сторони в плануванні, приймати рішення щодо оптимізації часу [1].

Для безпечного доступу та зберігання особистих даних система підтримує індивідуальні облікові записи. Це гарантує, що кожен користувач працює лише зі своїми завданнями та налаштовує інтерфейс відповідно до власних потреб (тема, мова, персональні параметри).

На ринку наявні різні застосунки для управління завданнями. Найбільш відомі – Google Tasks, Trello, Notion. Кожен із них реалізує власний підхід та інструментарій.

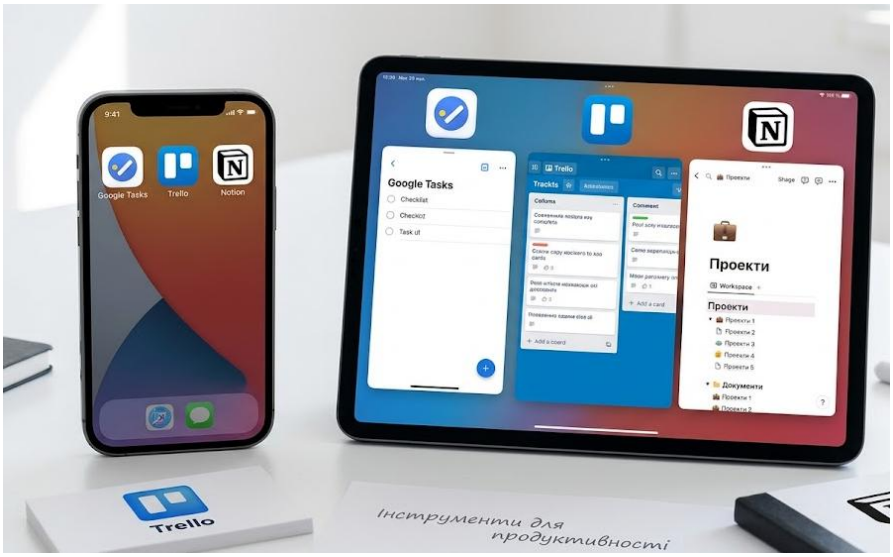

Рисунок 1.1 – Інтерфейс додатків планування завдань

Google Tasks орієнтований на базову роботу із завданнями, інтегрований із Google Calendar та Gmail. Дозволяє створювати прості списки задач, встановлювати терміни та підзадачі. Trello побудований на фреймворку канбан-дошок. Користувачі працюють із картками, переміщуючи їх між колонками. Notion – універсальна платформа для організації даних, де завдання є лише одним із модулів.

Аналіз переваг і недоліків аналогів відображено у таблиці 1.1.1.

Таблиця 1.1.1 – Аналіз існуючих аналогів

Критерій	Google Tasks	Trello	Notion
Тип системи	Простий список задач	Канбан-дошка	Універсальна платформа
Основне призначення	Базове управління завданнями	Візуальне управління проєктами	База знань, нотатки, управління проєктами
Структура	Списки та підзадачі	Дошки, колонки, картки	Сторінки, блоки, бази даних
Категоризація	Відсутня	Мітки, колонки	Бази даних з фільтрами
Пріоритети	Відсутні	Через мітки	Властивості бази даних
Статуси завдань	Виконано / Не виконано	Залежать від колонок	Настроювані
Дедлайни	Так, базові	Так, з календарем	Так, з нагадуваннями
Інтеграції	Google Calendar, Gmail	Багато зовнішніх сервісів	Розширений API
Мобільний застосунок	Так	Так	Так

Google Tasks підходить для мінімальних потреб, Trello – для візуального управління, Notion – для глибокої організації даних. Жоден із них не поєднує одночасно простоту, аналітику та гнучку категоризацію без обмежень, що обґрунтовує доцільність розробки власного рішення.

1.2 Постановка задачі

Мета розробки онлайн-планувальника завдань TaskFlow – створення веб-застосунку, який надає користувачу інструменти для повного циклу управління персональними задачами: від створення до аналізу виконання. Система забезпечує:

- централізоване зберігання та обробку завдань із доступом через веб-браузер;
- структуризацію завдань за категоріями, пріоритетами, статусами та термінами виконання;
- автоматичне інформування про наближення та прострочення дедлайнів;
- надання аналітичних даних у вигляді графіків та числових показників;
- підтримку двох мов інтерфейсу (українська, англійська) та двох візуальних тем (світла, темна).
-

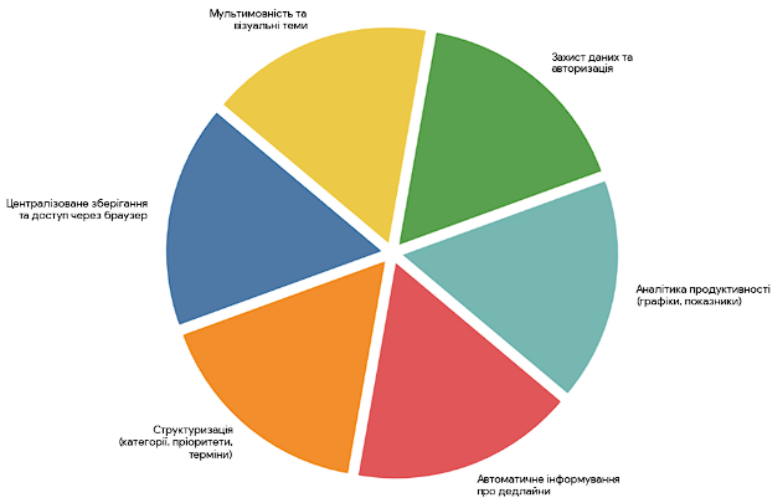

Рисунок 1.2.1 – Функціональна структура системи

У системі передбачено дві ролі користувачів:

1. Користувач – основна роль. Має доступ до повного функціоналу управління власними завданнями:

- створення, редагування та видалення завдань;
- створення, редагування та видалення категорій;
- перегляд дашборду з поточною статистикою;
- перегляд аналітики (графіки за статусами, пріоритетами, категоріями, тижнева активність);
- редагування особистого профілю;
- зміна теми оформлення та мови інтерфейсу;
- отримання нагадувань про дедлайни;
- вихід з облікового запису та видалення акаунта.

2. Адміністратор – роль, що передбачає розширені можливості (на етапі подальшого розвитку системи):

- перегляд списку всіх зареєстрованих користувачів;
- блокування та видалення облікових записів;

Для деталізації процесу управління завданнями та визначення меж відповідальності кожної ролі розроблено модель прецедентів (Use Case Model). Модель демонструє, як Користувач та Адміністратор взаємодіють із функціональними блоками системи TaskFlow. В основу побудови моделі покладено принцип модульності: кожна група функцій (керування завданнями, аналітика, профілювання) виступає окремим сервісом, доступним через інтерфейс браузера.

На Рисунку 1.2.2 представлено діаграму прецедентів, яка візуалізує структуру прав доступу та основні сценарії використання онлайн-планувальника TaskFlow.

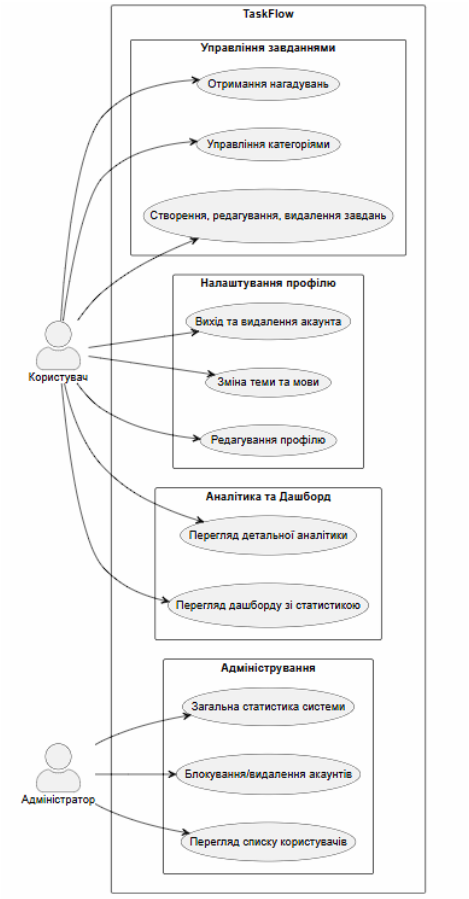

Рисунок 1.2.2 – Діаграма прецедентів системи

Функціонал системи поділяється на такі модулі:

1. Авторизація та реєстрація:

- реєстрація нового облікового запису з підтвердженням через електронну пошту;
- вхід до системи за логіном та паролем;
- відновлення пароля;
- автоматичне перенаправлення авторизованих користувачів на дашборд;

На рисунку 1.2.3 показано діаграму послідовності модуля авторизації:

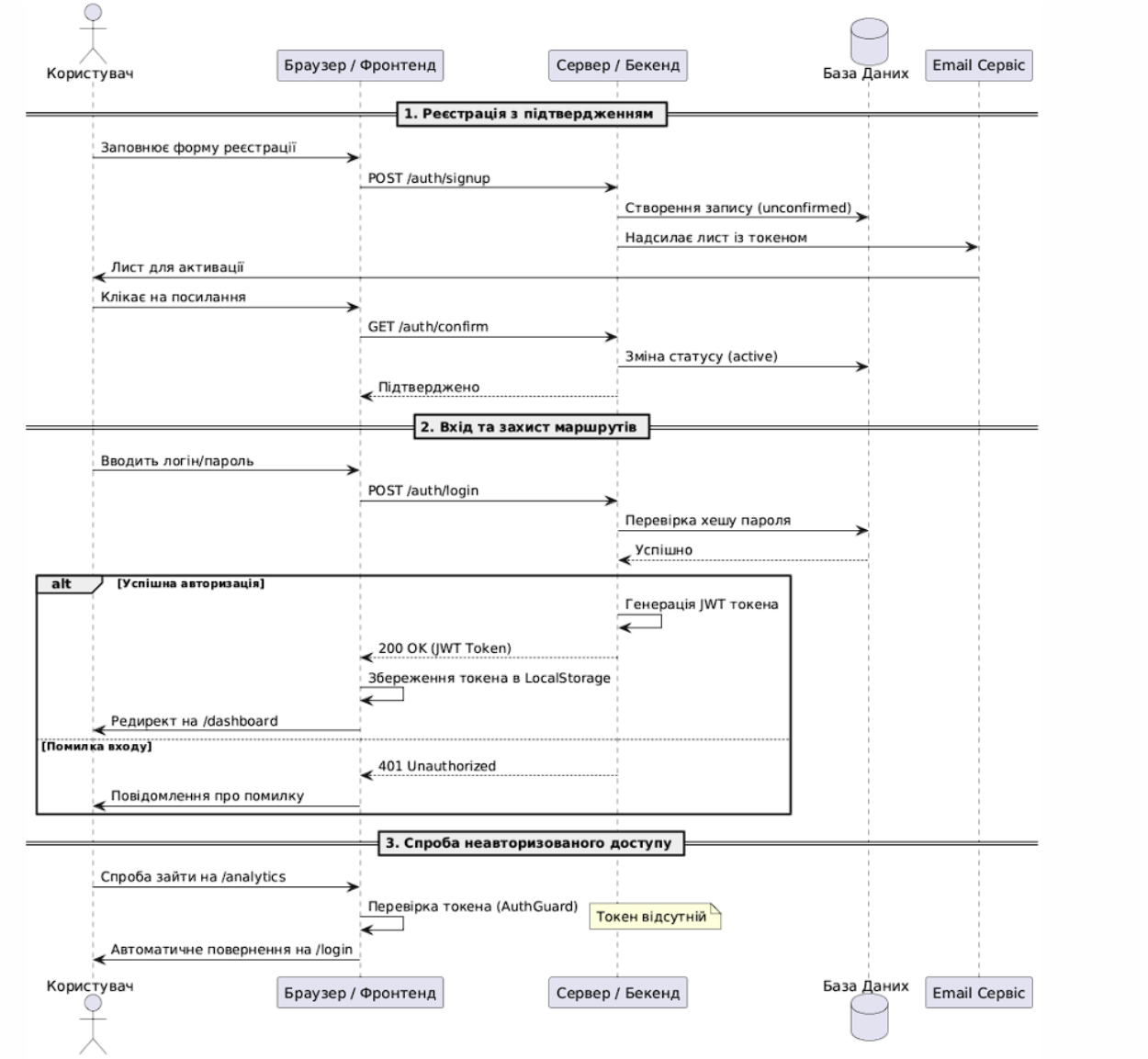

Рисунок 1.2.3 – Діаграма послідовності модуля авторизації

2. Управління завданнями:

- створення завдання з полями: назва, опис, пріоритет (низький, середній, високий), статус (очікує, в процесі, виконано), термін виконання, категорія;
- редагування всіх полів завдання;
- видалення завдання з підтвердженням дії;
- зміна статусу завдання без відкриття форми редагування (через контекстне меню).

На рисунку 1.2.4 відображено схему діаграми послідовності модуля:

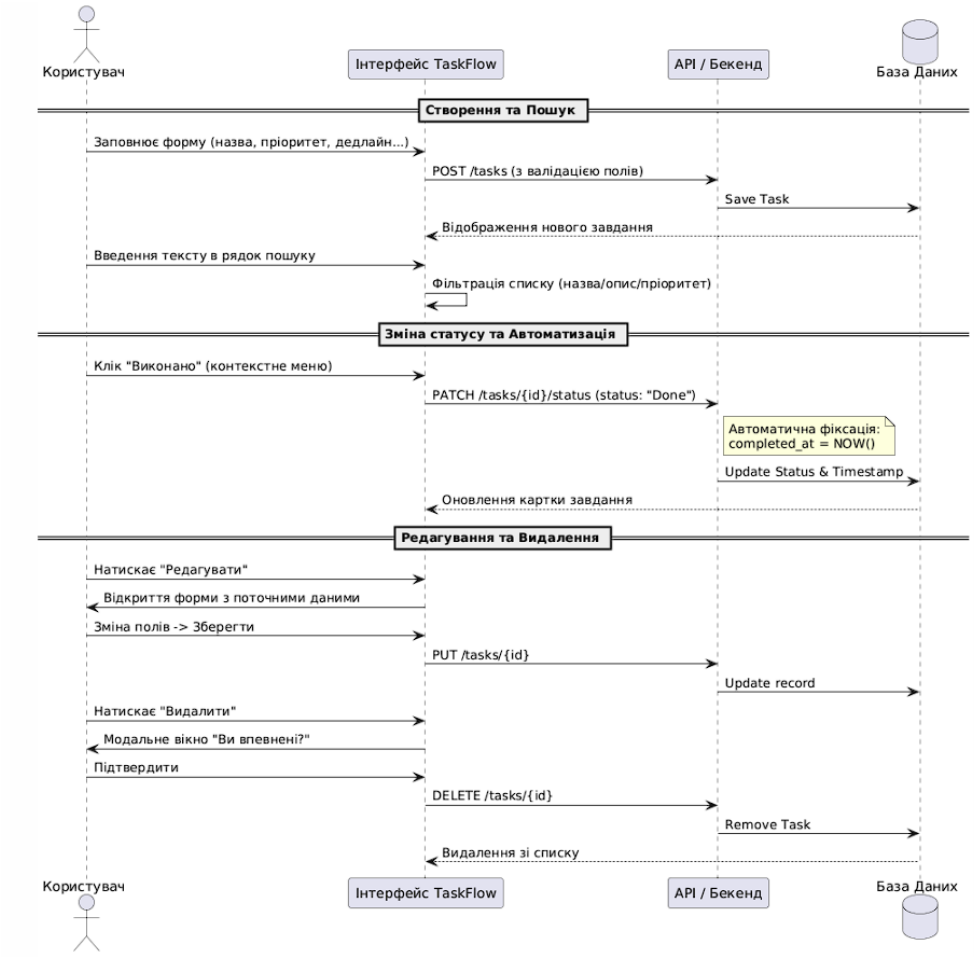

Рисунок 1.2.4 – Діаграма послідовності модуля управління завданнями

3. Управління категоріями:

- створення категорії з вибором назви, кольору (12 варіантів) та іконки (12 варіантів із бібліотеки Lucide);
- редагування параметрів категорії;
- видалення категорії;
- відображення прогресу виконання завдань у межах кожної категорії.

На рисунку 1.2.5 показано діаграму класів для модуля управління категоріями.

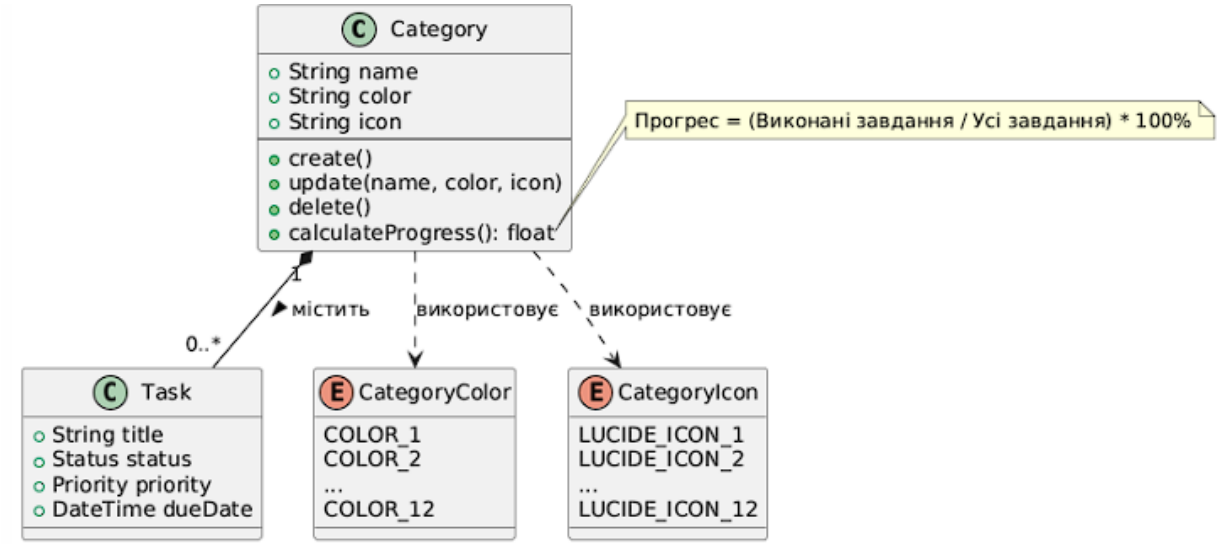

Рисунок 1.2.5 – Діаграма класів управління категоріями

4. Дашборд (головна панель):

- персональне привітання користувача залежно від часу доби;
- зведені показники: загальна кількість завдань, виконаних, у процесі, рівень завершення у відсотках;
- список завдань на сьогодні;
- список найближчих завдань за терміном виконання;
- огляд категорій із прогрес-барами;
- нещодавня активність користувача.

Діаграма компонентів на рисунку 1.2.6 добре показує візуалізацію роботи дашборду, оскільки дашборд – це перш за все агрегатор даних із різних частин системи.

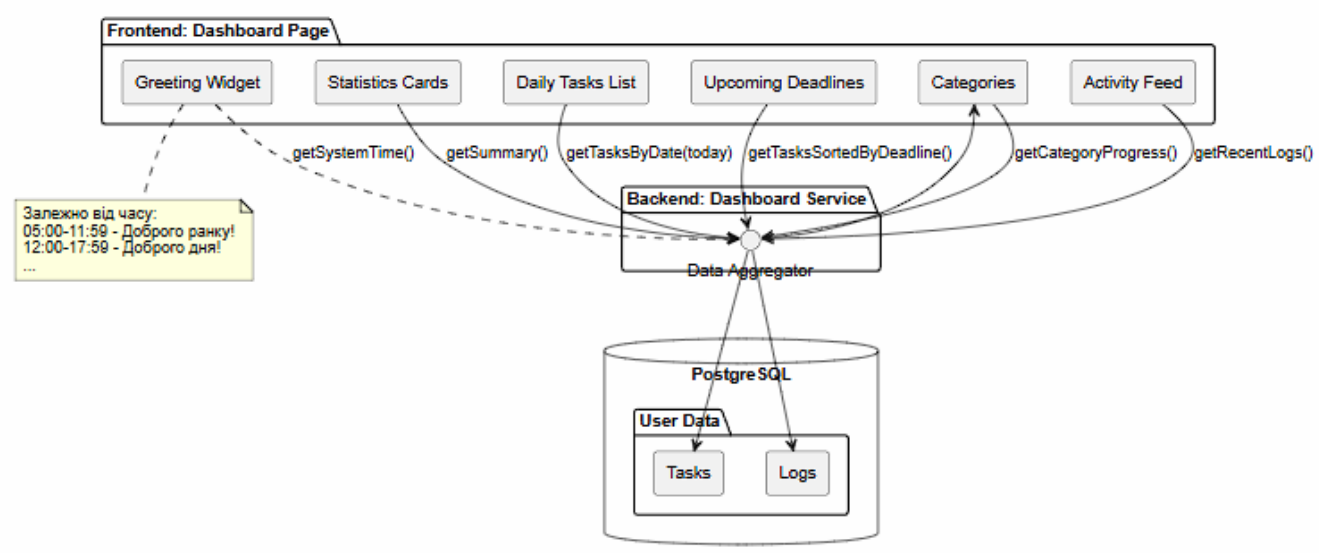

Рисунок 1.2.6 – Діаграма компонентів дашборду

5. Статистика та аналітика:

- кругова діаграма розподілу завдань за статусами;
- кругова діаграма розподілу завдань за пріоритетами;
- стовпчикова діаграма тижневої активності (створені та завершені завдання за останні 7 днів);
- горизонтальна стовпчикова діаграма розподілу завдань за категоріями;
- зведення за дедлайнами: кількість прострочених завдань, завдань на сьогодні та майбутніх завдань.

6. Система нагадувань:

- автоматичне визначення завдань із простроченим терміном, терміном на сьогодні та на завтра;
- відображення спливаючих нагадувань у правому нижньому куті екрана;
- можливість швидко позначити завдання виконаним безпосередньо з нагадування;

7. Профіль користувача:

- перегляд та редагування повного імені;
- відображення електронної пошти та дати реєстрації;
- статистика акаунта: загальна кількість і кількість виконаних завдань;
- завершення сеансу (вихід із системи);
- видалення облікового запису з підтвердженням.

8. Зберігання та безпека даних:

- використання бази даних PostgreSQL через платформу Supabase;
- автентифікація через Supabase Auth із використанням JWT-токенів;
- розмежування доступу на рівні бази даних через Row Level Security (RLS);
- зберігання призначених для користувача налаштувань (тема, мова) у локальному сховищі браузера.

На рисунку 1.2.6 показано діаграму розгортання вона показує, як клієнтська частина взаємодіє із хмарними сервісами через захищені протоколи.

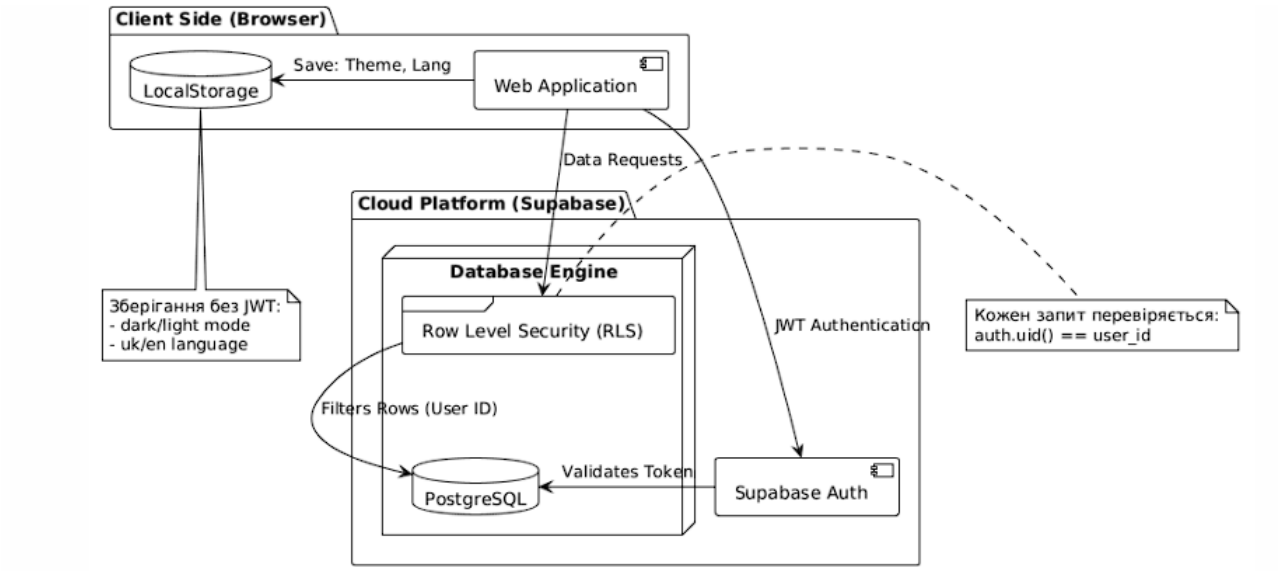

Рисунок 1.2.6 – Діаграма розгортання бази даних

1.3 Функціональні та нефункціональні вимоги

Функціональні вимоги визначають поведінку системи, перелік операцій, які система має виконувати, та реакцію на дії користувача.

FR-01. Автентифікація користувача. Система забезпечує реєстрацію нового користувача за допомогою електронної пошти та пароля, вхід зареєстрованого користувача та вихід з облікового запису.

FR-02. Керування профілем. Система надає можливість перегляду даних профілю (ім'я, електронна пошта, дата реєстрації) та редагування повного імені.

FR-03. Керування завданнями. Система дозволяє створювати завдання із зазначенням назви, опису, пріоритету, статусу, терміну виконання та категорії; редагувати всі поля існуючого завдання; видаляти завдання; змінювати статус (pending, inprogress, completed) без входу в режим редагування.

FR-04. Керування категоріями. Система надає можливість створювати категорії із зазначенням назви, кольору та іконки, редагувати параметри категорії, видаляти категорію.

FR-05. Перегляд завдань. Система відображає список усіх завдань користувача із фільтрацією за статусом (all, pending, inprogress, completed), пріоритетом (all, low, medium, high) та категорією. Забезпечено текстовий пошук за назвою та описом.

FR-06. Головна панель. Система відображає загальну кількість завдань, кількість виконаних, кількість завдань у процесі, відсоток завершення; список завдань на поточну дату; список майбутніх завдань, відсортованих за датою виконання; огляд категорій із прогресом виконання.

FR-07. Статистика та аналітика. Система відображає графік розподілу завдань за статусом (кругова діаграма), за пріоритетом (кругова діаграма), тижневу активність (стовпчикова діаграма), розподіл за категоріями (горизонтальна стовпчикова діаграма), кількість прострочених завдань, завдань на сьогодні та майбутніх.

FR-08. Система нагадувань. Система автоматично визначає прострочені завдання, завдання з терміном на поточну дату та на наступний день; відображає спливаючі нагадування в правому нижньому куті інтерфейсу; дозволяє закрити окреме нагадування або всі одразу; дозволяє позначити завдання виконаним безпосередньо з нагадування.

FR-09. Локалізація інтерфейсу. Система підтримує українську та англійську мови, перемикання виконується без перезавантаження сторінки.

FR-10. Тема оформлення. Система підтримує світлу та темну теми, перемикання виконується без перезавантаження сторінки, вибір теми зберігається між сесіями.

FR-05. Перегляд завдань

Система повинна відображати список усіх завдань користувача. Система повинна забезпечувати фільтрацію завдань за статусом (all, pending, inprogress, completed). Система повинна забезпечувати фільтрацію завдань за пріоритетом (all, low, medium, high). Система повинна забезпечувати фільтрацію завдань за категорією. Система повинна забезпечувати текстовий пошук завдань за назвою та описом.

FR-06. Головна панель (Dashboard)

Система повинна відображати на головній панелі загальну кількість завдань, кількість виконаних завдань, кількість завдань у процесі, відсоток завершення. Система повинна відображати список завдань на поточну дату. Система повинна відображати список майбутніх завдань, відсортованих за датою виконання. Система повинна відображати огляд категорій із прогресом виконання.

FR-07. Статистика та аналітика

Система повинна відображати графік розподілу завдань за статусом (кругова діаграма). Система повинна відображати графік розподілу завдань за пріоритетом (кругова діаграма). Система повинна відображати графік тижневої активності (стовпчикова діаграма створених та виконаних завдань).

FR-08. Система нагадувань

Система повинна автоматично визначати прострочені завдання, завдання з терміном на поточну дату та завдання з терміном на наступний день. Система повинна відображати спливаючі нагадування для визначених завдань у правому нижньому куті інтерфейсу. Система повинна надавати можливість закрити окреме нагадування або всі нагадування. Система повинна надавати можливість позначити завдання виконаним безпосередньо з нагадування.

Нефункціональні вимоги визначають якісні характеристики системи, обмеження та умови функціонування.

NFR-01. Продуктивність. Час відповіді сервера не перевищує 500 мс для операцій читання та 1000 мс для операцій запису за нормальних умов навантаження. First Input Delay – не більше 100 мс.

NFR-02. Надійність. Система коректно обробляє помилки мережевих з'єднань із виведенням повідомлень користувачеві, зберігає цілісність даних при одночасному доступі кількох клієнтів, забезпечує доступність не нижче 99.5% часу роботи серверної інфраструктури.

NFR-03. Безпека. Усі запити до бази даних виконуються з урахуванням належності даних конкретному користувачеві (Row Level Security). Автентифікація – через JWT-токени. Усі з'єднання – через HTTPS. Паролі не зберігаються у відкритому вигляді.

NFR-04. Зручність використання. Інтерфейс інтуїтивно зрозумілий без спеціального навчання. Основні операції (створення завдання, зміна статусу) виконуються не більше ніж за 2 кліки. Система надає візуальний зворотний зв'язок (анімації, зміна стану кнопок, індикатори завантаження).

NFR-05. Сумісність. Клієнтська частина коректно функціонує в останніх версіях Google Chrome, Mozilla Firefox, Safari та Microsoft Edge. Система коректно відображається на пристроях із шириною екрану від 320px до 2560px.

NFR-06. Підтримуваність. Вихідний код структурований за модульним принципом, написаний із використанням TypeScript у режимі суворої типізації. Найменування змінних, функцій та компонентів відповідає їх призначенню.

NFR-07. Масштабованість. Архітектура допускає додавання нових модулів без зміни існуючої структури. База даних підтримує збільшення обсягу даних без деградації продуктивності.

NFR-08. Локалізація. Усі текстові рядки винесено в окремі файли. Додавання нової мови не потребує змін у логіці компонентів.

NFR-09. Розгортання. Система підтримує розгортання на платформі Vercel без додаткової конфігурації серверного середовища. База даних розгортається як керований сервіс Supabase.

1.4 Архітектурні вимоги та вибір технологій

Архітектура системи «TaskFlow» побудована за клієнт-серверним принципом із гібридним рендерингом. Клієнтська частина відповідає за відображення інтерфейсу та обробку дій користувача. Серверна частина – за бізнес-логіку, взаємодію з базою даних та автентифікацію. Такий розподіл забезпечує чітке розмежування відповідальності між рівнями системи [3].

На рисунку 1.4.1 схематично показано архітектуру системи:

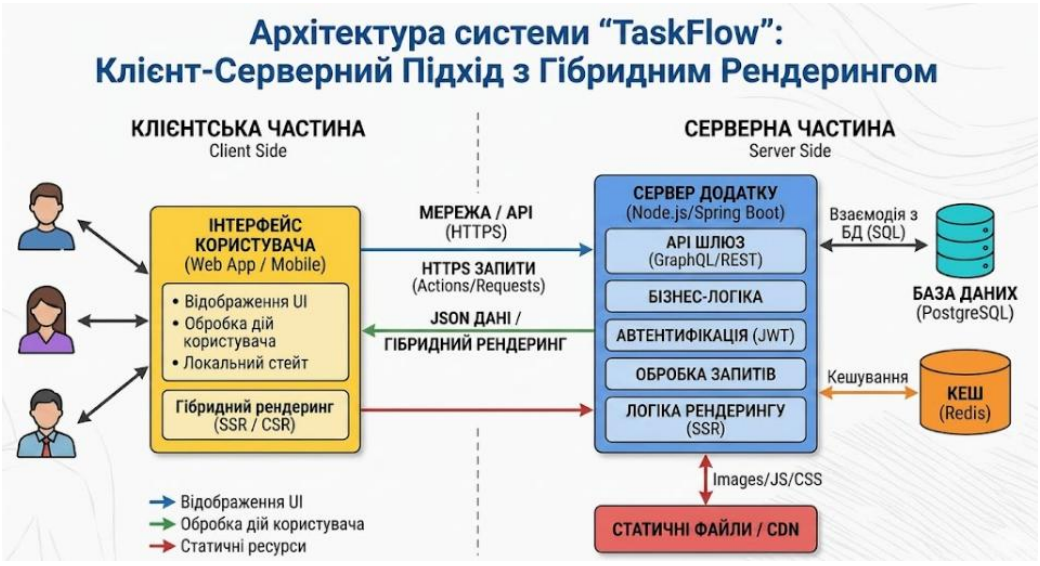

Рисунок 1.4.1 – Клієнт-серверна архітектура в системі

Для реалізації фронтенду обрано Next.js версії 16 з App Router. Next.js – один із провідних фреймворків екосистеми React із вбудованою підтримкою серверного рендерингу (SSR), статичної генерації (SSG) та гібридних підходів. App Router забезпечує файлову маршрутизацію: структура директорій безпосередньо визначає URL-структуру застосунку. Серверні компоненти React дозволяють виконувати запити до бази даних на стороні сервера та передавати готові дані клієнту, зменшуючи обсяг JavaScript у браузері. Next.js має вбудовану оптимізацію зображень, шрифтів та скриптів [3].

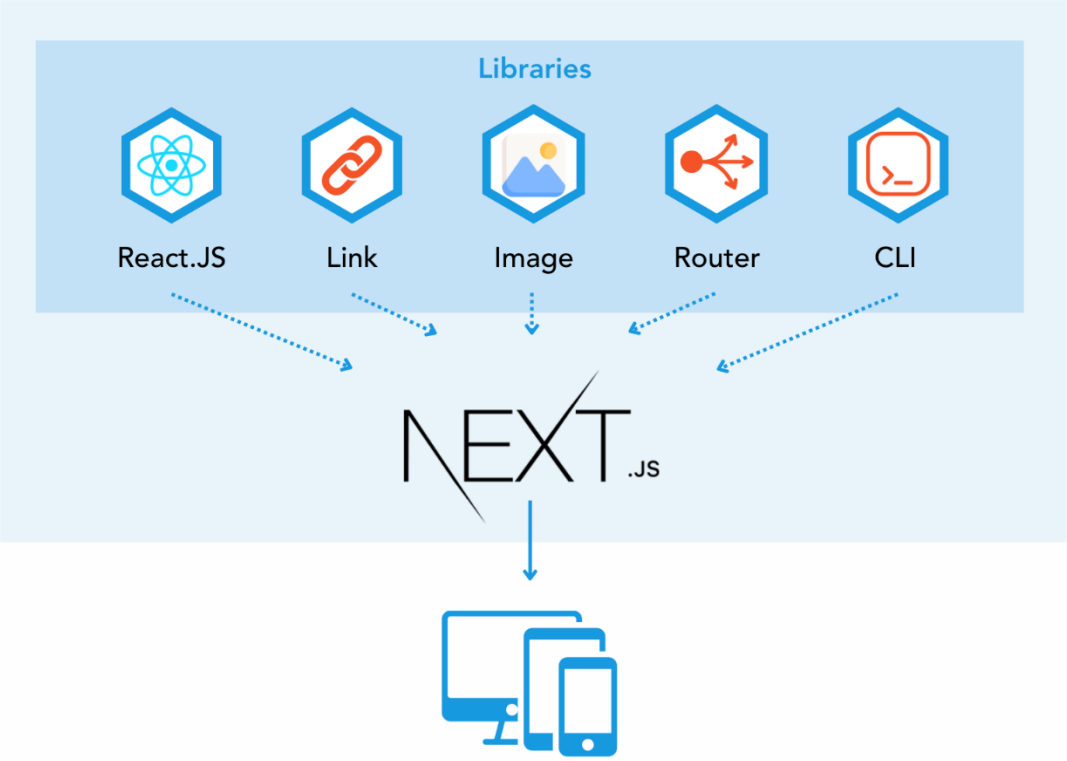

Рисунок 1.4.2 – Фреймворк Next.js

Для стилізації інтерфейсу використано Tailwind CSS версії 4 – utility-first CSS-фреймворк, який надає готові класи для стилізації без написання кастомного CSS. Стилі задаються безпосередньо в JSX-розмітці. Вбудована підтримка тем реалізована через CSS-змінні, адаптивність – через префікси класів (sm:, md:, lg:). У проєкті реалізовано кастомну систему кольорів на основі простору OKLCH, що забезпечує кращу доступність та ширший колірний діапазон порівняно з традиційним RGB [5].

Рисунок 1.4.3 – Логотип Tailwind CSS

Для побудови інтерфейсу використано бібліотеку Radix UI – набір примітивних компонентів (діалогові вікна, випадаючі списки, меню, вкладки, спливаючі підказки, перемикачі), які відповідають стандартам WAI-ARIA щодо доступності. Компоненти Radix UI нестилізовані (headless), що дозволяє застосовувати до них власні стилі через Tailwind CSS без конфліктів. У проєкті використано 23 компоненти Radix UI: Dialog, DropdownMenu, Select, Tabs, Tooltip, Avatar, Progress, Toast, Toggle, Switch, Label [14].

Як серверну платформу обрано Supabase – відкриту альтернативу Firebase, яка надає керовану PostgreSQL-базу даних, вбудовану автентифікацію, REST API та підтримку реального часу. Вибір зумовлено трьома факторами: реляційна PostgreSQL забезпечує цілісність даних через зовнішні ключі та обмеження; механізм Row Level Security контролює доступ на рівні окремих рядків таблиці; вбудована автентифікація з JWT-токенами усуває потребу в розробці власного сервера автентифікації [4].

Взаємодія з Supabase реалізована через два типи клієнтів. Серверний клієнт отримує дані через Service Key і повертає їх клієнту під час серверного рендерингу. Клієнтський клієнт виконує операції створення, оновлення та видалення даних через анонімний ключ з обмеженими правами. Серверний клієнт має розширені права для SSR-запитів, клієнтський – обмежений політиками Row Level Security.

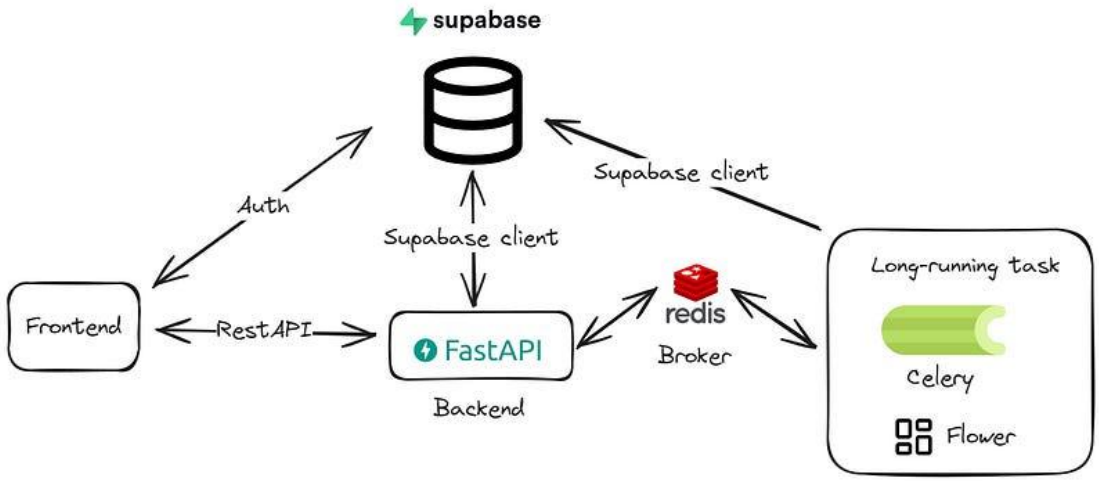

Рисунок 1.4.4 – Серверна частина Supabase

Мова програмування – TypeScript у режимі строгої типізації. Статична типізація забезпечує виявлення помилок на етапі компіляції, покращує читабельність коду та спрощує рефакторинг. У проєкті визначено власні типи для всіх сутностей: Task, Category, Profile, Priority, TaskStatus [6].

На рисунку 1.4.5 показано логотип мови програмування:

Рисунок 1.4.5 – Логотип TypeScript

Для візуалізації статистичних даних використано Recharts – бібліотеку для побудови графіків у React, побудовану поверх D3.js, але з декларативним API через React-компоненти. У системі застосовано кругову діаграму (PieChart) для розподілу за статусом та пріоритетом, стовпчикову діаграму (BarChart) для тижневої активності, горизонтальну стовпчикову діаграму для розподілу за категоріями [15].

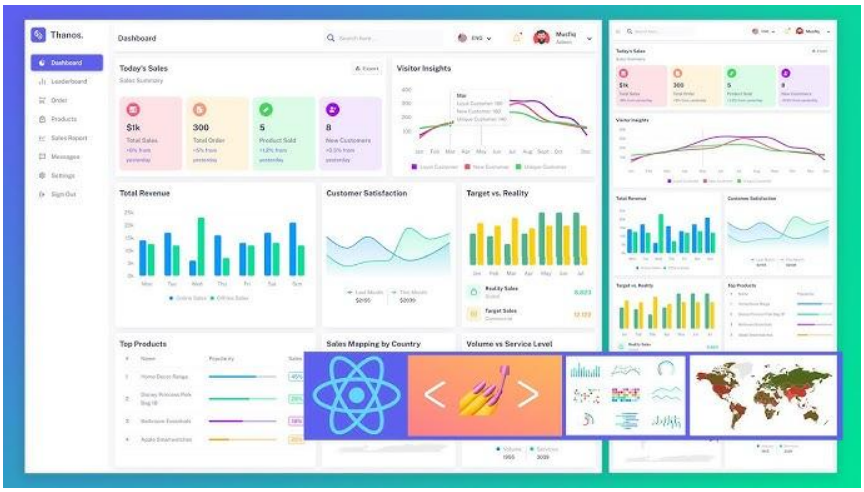

Рисунок 1.4.6 – Приклади діаграм Recharts

Розгортання системи здійснюється на платформі Vercel – офіційному хостингу для Next.js-застосунків із автоматичним розгортанням із Git-репозиторію, серверним рендерингом та оптимізацією статичних ресурсів. База даних розгортається як керований сервіс Supabase, що усуває необхідність в адмініструванні сервера PostgreSQL [18]. Логотип сервісу показано на рисунку 1.4.7.

Рисунок 1.4.7 – Логотип Vercel

1.5 Висновки до першого розділу

У першому розділі проведено аналіз предметної області онлайн-планувальника завдань. Визначено, що основними характеристиками завдання є назва, опис, категорія, пріоритет, статус виконання та термін завершення. Обґрунтовано необхідність групування завдань за категоріями, розмежування за пріоритетами та статусами, автоматичного інформування про дедлайни та надання аналітики продуктивності.

Досліджено три існуючі аналоги: Google Tasks, Trello та Notion. Встановлено, що Google Tasks забезпечує лише базове управління без категоризації, Trello орієнтований на візуальне управління проєктами без вбудованої аналітики, Notion є універсальною платформою з мінімальними засобами статистики.

Сформульовано мету системи — створення веб-застосунку для повного циклу управління персональними завданнями. Визначено дві ролі користувачів (Користувач, Адміністратор) та вісім функціональних модулів: авторизація, управління завданнями, управління категоріями, дашборд, статистика, система нагадувань, профіль користувача, зберігання та безпека даних.

Визначено 10 функціональних вимог (FR-01 – FR-10) та 9 нефункціональних вимог (NFR-01 – NFR-09), що регламентують продуктивність, надійність, безпеку, зручність, сумісність, підтримуваність, масштабованість, локалізацію та розгортання системи.

Обґрунтовано вибір технологічного стеку: Next.js 16 з App Router для фронтенду та SSR, Tailwind CSS 4 для стилізації, Radix UI для доступних компонентів, Supabase (PostgreSQL) як BaaS-платформу, TypeScript для статичної типізації, Recharts для візуалізації даних, Vercel для хостингу. Обраний стек забезпечує продуктивність, безпеку та масштабованість системи при мінімальних витратах на серверну інфраструктуру.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ

У цьому розділі виконано структурне та архітектурне проєктування системи: побудовано діаграму класів, ER-діаграму бази даних та діаграми послідовностей. Описано проєктування інтерфейсу користувача, розробку клієнтської та серверної частин, механізми інтеграції та збереження даних.

2.1 Структурне та архітектурне проєктування

Структурне та архітектурне проєктування – фундаментальний етап розробки. На ньому визначають внутрішню будову системи, склад модулів, структуру даних і взаємодію між компонентами. У цьому підрозділі представлено діаграму класів (статична структура), ER-діаграму бази даних (логічна модель даних) та діаграми послідовностей (динаміка ключових операцій) [3].

Діаграма класів відображає сутності предметної області, їх атрибути, методи та зв'язки. У «TaskFlow» виділено три основні сутності: User (користувач), Task (завдання) та Category (категорія). User доповнюється профілем Profile. Для пріоритету та статусу завдань використовуються перелічувані типи Priority та TaskStatus [21].

Нижче на рисунку 2.1.1 наведено діаграму класів у нотації UML та детальний опис кожного класу:

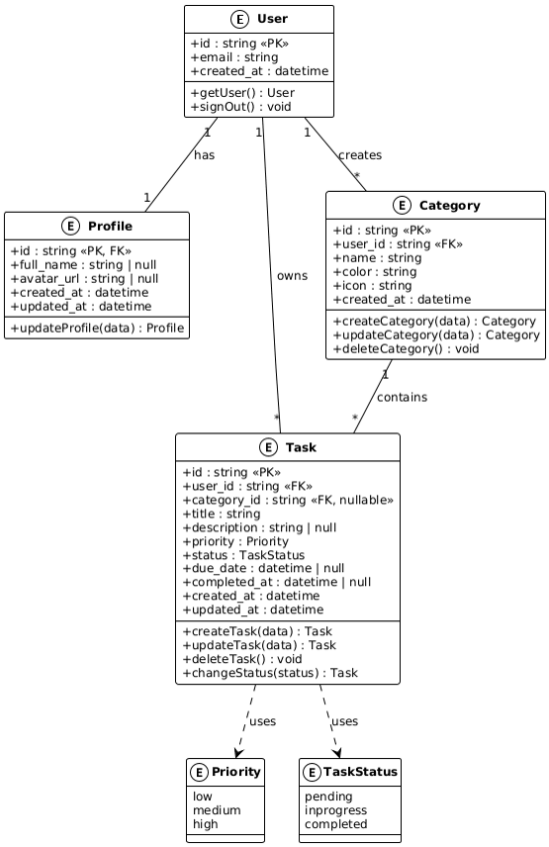

Рисунок 2.1.1 – Діаграма класів системи

Клас User відповідає запису в таблиці auth.users, керованій Supabase Auth [4]. id – первинний ключ, генерується автоматично. email – для автентифікації. created_at фіксує дату реєстрації. getUser() повертає об'єкт поточного користувача з сесії. signOut() завершує сесію та перенаправляє на сторінку входу.

Клас Profile зберігає додаткову інформацію. id – одночасно первинний ключ і зовнішній ключ до User.id (зв'язок один-до-одного). full_name – повне ім'я (null, якщо не заповнене). avatar_url – посилання на аватар (null). created_at і updated_at фіксують час створення й оновлення. updateProfile(data) оновлює поля та повертає об'єкт.

Клас Task є центральною сутністю системи та представляє завдання. Атрибут id є первинним ключем. Атрибут user_id є зовнішнім ключем до User.id та визначає власника завдання. Атрибут category_id є зовнішнім ключем до Category.id та визначає категорію завдання (допускається null, якщо категорію не призначено). Атрибут title містить назву завдання (обов'язкове поле). Атрибут description містить детальний опис (опціональне поле). Атрибут priority визначає пріоритет завдання через перелічуваний тип Priority (low – низький, medium – середній, high – високий). Атрибут status визначає стан виконання через перелічуваний тип TaskStatus (pending – очікує, inprogress – у процесі, completed – виконано). Атрибут due_date містить термін виконання (може бути null). Атрибут completed_at фіксує дату та час переведення в статус completed (null для інших статусів). Атрибути created_at та updated_at фіксують час створення та останнього оновлення запису.

Клас Task має чотири основні методи. Метод createTask(data) створює нове завдання з переданими даними та повертає створений об'єкт. Метод updateTask(data) оновлює поля існуючого завдання. Метод deleteTask() видаляє завдання. Метод changeStatus(status) змінює статус завдання; при переході в статус completed автоматично встановлює completed_at, при переході в інший статус – скидає completed_at у null.

Клас Category представляє категорію для групування завдань. Атрибут id є первинним ключем. Атрибут user_id є зовнішнім ключем до User.id та визначає власника категорії. Атрибут name містить назву категорії. Атрибут color зберігає колір у шістнадцятковому форматі (наприклад, #3B82F6).

Клас Category має три методи. Метод createCategory(data) створює нову категорію з переданими назвою, кольором та іконкою. Метод updateCategory(data) оновлює поля існуючої категорії. Метод deleteCategory() видаляє категорію (завдання, пов'язані з категорією, при цьому не видаляються, але втрачають зв'язок через встановлення category_id у null).

Між класами встановлено такі зв'язки:

Зв'язок один-до-одного між User та Profile реалізовано через спільний первинний ключ: Profile.id одночасно є зовнішнім ключем до User.id. Це гарантує, що кожен користувач має рівно один профіль. Профіль створюється автоматично при реєстрації користувача.

Зв'язок один-до-багатьох між User та Task означає, що один користувач може створити довільну кількість завдань, але кожне завдання належить рівно одному користувачеві. Зв'язок реалізовано через зовнішній ключ Task.user_id, що посилається на User.id. На рівні бази даних політика Row Level Security гарантує, що користувач бачить лише власні завдання.

Зв'язок один-до-багатьох між User та Category означає, що один користувач може створити довільну кількість категорій, але кожна категорія належить рівно одному користувачеві. Зв'язок реалізовано через зовнішній ключ Category.user_id.

Зв'язок один-до-багатьох між Category та Task означає, що одна категорія може містити довільну кількість завдань, але кожне завдання може бути віднесене не більше ніж до однієї категорії. Зв'язок реалізовано через зовнішній ключ Task.category_id, що посилається на Category.id. Допускається значення null у Task.category_id, що означає відсутність категорії.

Зв'язки залежності (пунктирна лінія) між Task та перелічуваними типами Priority і TaskStatus показують, що клас Task використовує ці типи для визначення допустимих значень атрибутів priority та status.

ER-діаграма (Entity-Relationship Diagram) відображає логічну структуру бази даних: сутності, їх атрибути, первинні та зовнішні ключі, а також зв'язки між таблицями. База даних системи «TaskFlow» реалізована на PostgreSQL під управлінням Supabase і складається з трьох основних таблиць: profiles, tasks, categories та службової таблиці auth.users, керованої Supabase Auth [4].

На рисунку 2.1.2 наведено ER-діаграму бази даних системи.

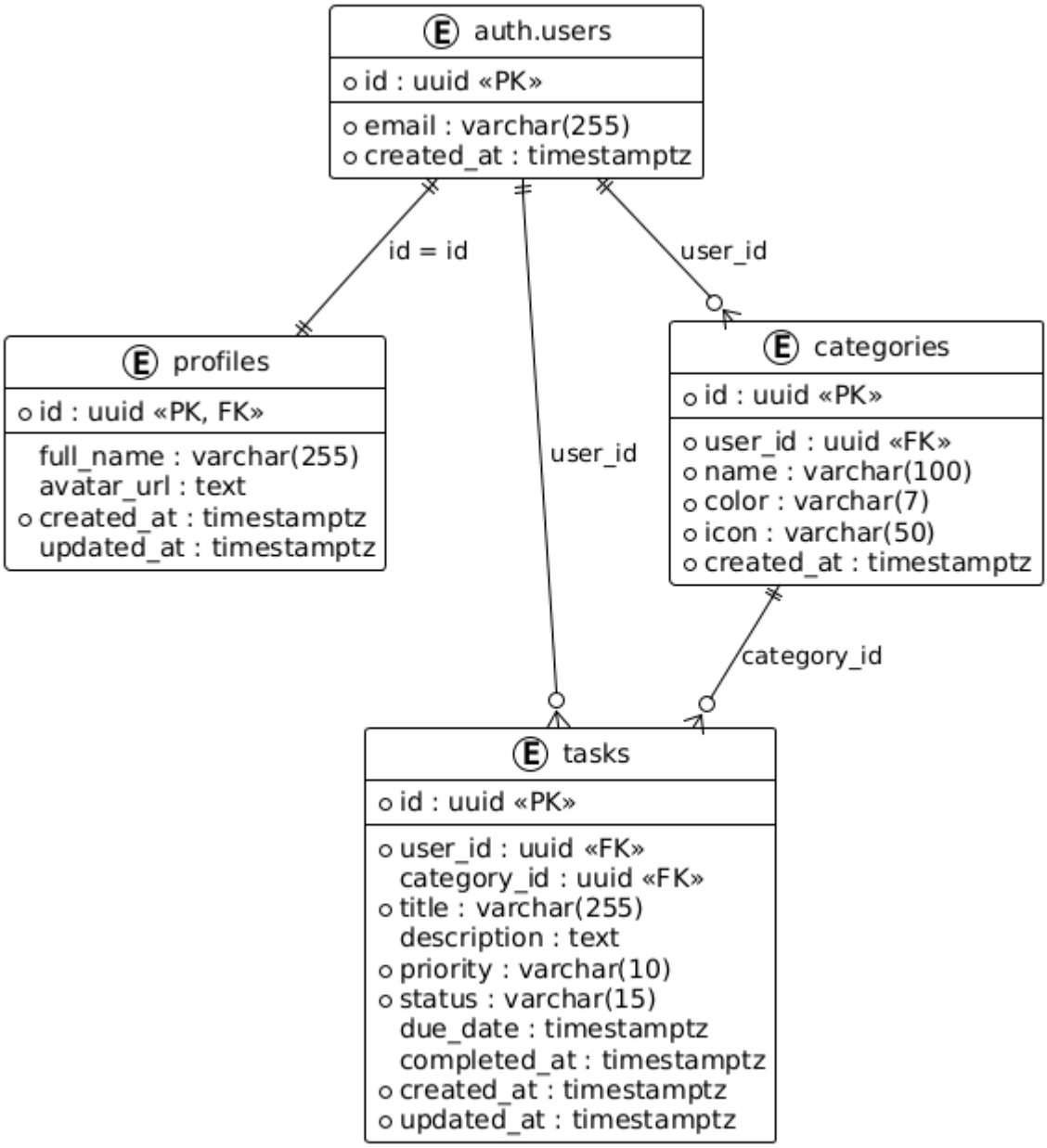

Рисунок 2.1.2 – ER-діаграма бази даних системи TaskFlow

Таблиця auth.users є службовою таблицею Supabase Auth і не створюється розробником вручну. Вона містить базову інформацію про зареєстрованих користувачів. Первинний ключ id має тип uuid та генерується автоматично. Поле email є обов'язковим і використовується для автентифікації. Поле created_at фіксує час реєстрації.

Таблиця profiles зберігає розширену інформацію про користувачів. Первинний ключ id одночасно є зовнішнім ключем до auth.users.id, що забезпечує зв'язок один-до-одного. Поле full_name (varchar(255)) є опціональним і містить повне ім'я користувача. Поле avatar_url (text) є опціональним і зберігає URL-адресу аватара. Поле created_at фіксує час створення профілю, updated_at – час останнього оновлення.

Таблиця tasks є основною таблицею системи та зберігає всі завдання. Первинний ключ id має тип uuid. Поле user_id є зовнішнім ключем до auth.users.id, визначає власника завдання та є обов'язковим. Поле category_id є зовнішнім ключем до categories.id, визначає категорію завдання та допускає значення NULL. Поле title (varchar(255)) є обов'язковим. Поле description (text) є опціональним. Поле priority (varchar(10)) є обов'язковим і приймає значення 'low', 'medium', 'high'. Поле status (varchar(15)) є обов'язковим і приймає значення 'pending', 'inprogress', 'completed'. Поле due_date (timestamptz) є опціональним і зберігає термін виконання завдання. Поле completed_at (timestamptz) є опціональним і автоматично заповнюється при переході завдання в статус 'completed'. Поля created_at та updated_at фіксують час створення та останньої модифікації.

Таблиця categories зберігає категорії завдань. Первинний ключ id має тип uuid. Поле user_id є зовнішнім ключем до auth.users.id і визначає власника категорії. Поле name (varchar(100)) є обов'язковим. Поле color (varchar(7)) зберігає колір у форматі HEX (наприклад, '#3B82F6'). Поле icon (varchar(50)) зберігає ідентифікатор іконки. Поле created_at фіксує час створення категорії.

Між таблицями auth.users та profiles встановлено зв'язок один-до-одного через рівність первинних ключів (users.id = profiles.id). Це означає, що для кожного користувача існує рівно один профіль, і навпаки.

Між таблицями auth.users та tasks встановлено зв'язок один-до-багатьох через зовнішній ключ tasks.user_id. Один користувач може мати довільну кількість завдань, але кожне завдання належить лише одному користувачеві.

Між таблицями auth.users та categories встановлено зв'язок один-до-багатьох через зовнішній ключ categories.user_id. Один користувач може створити довільну кількість категорій.

Між таблицями categories та tasks встановлено зв'язок один-до-багатьох через зовнішній ключ tasks.category_id. Одна категорія може містити довільну кількість завдань, але кожне завдання належить не більше ніж одній категорії. Допускається значення NULL у полі tasks.category_id, що означає відсутність категорії.

Діаграми послідовностей (Sequence Diagrams) відображають динаміку взаємодії між об'єктами системи в хронологічному порядку. Для системи «TaskFlow» побудовано діаграми послідовностей для трьох ключових операцій: автентифікація користувача, створення завдання та зміна статусу завдання [21].

Діаграма послідовності: Автентифікація користувача

На рисунку 2.1.3 наведено діаграму послідовності процесу входу користувача в систему.

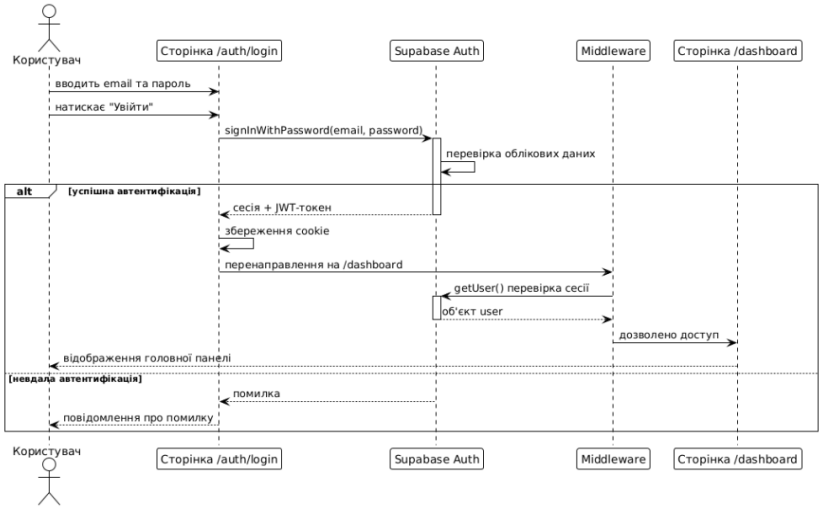

Рисунок 2.1.3 – Діаграма послідовності для автентифікації користувача

Опис процесу: користувач вводить облікові дані на сторінці входу та натискає кнопку «Увійти». Клієнтський компонент викликає метод signInWithPassword(email, password) Supabase Auth [4]. Supabase перевіряє облікові дані: у разі успіху створює сесію, генерує JWT-токен та повертає їх клієнту; клієнт зберігає cookie сесії та перенаправляє користувача на /dashboard. Middleware перевіряє наявність сесії через getUser() і, якщо сесія активна, дозволяє доступ до захищеного маршруту [3]. У разі невдалої автентифікації Supabase повертає помилку, яка відображається користувачеві.

Діаграма послідовності: Створення завдання

На рисунку 2.1.4 наведено діаграму послідовності процесу створення нового завдання.

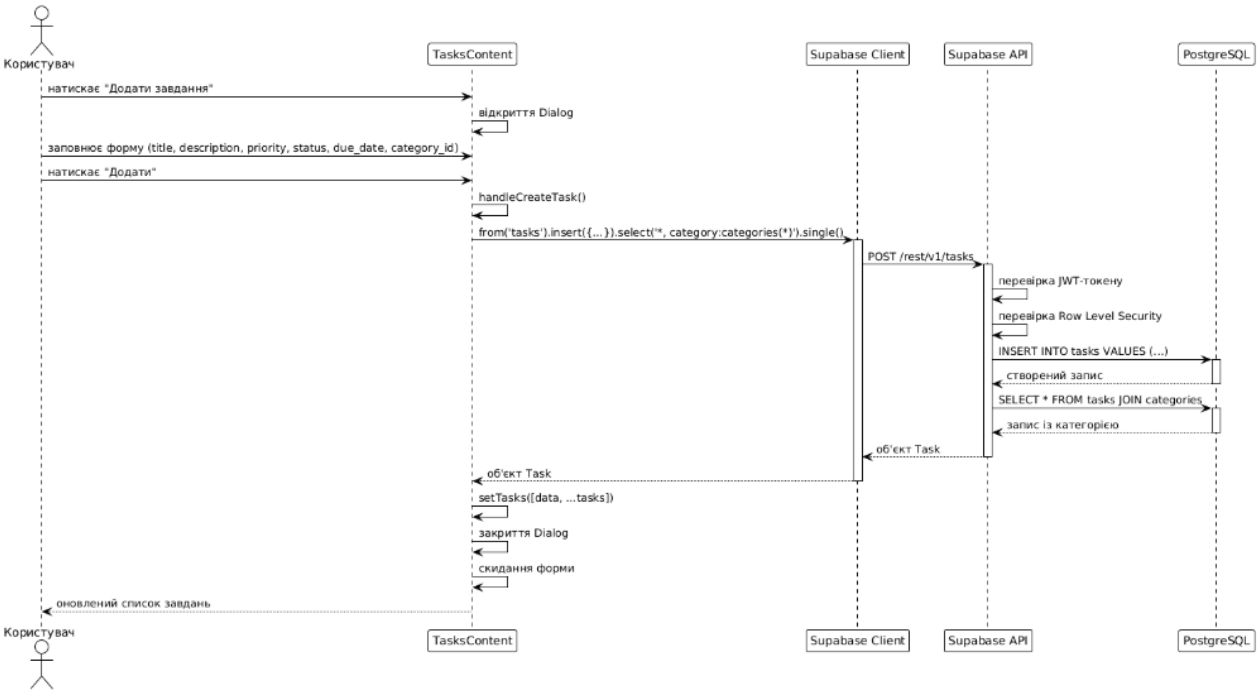

Рисунок 2.1.4 – Діаграма послідовності для дії створення завдання

Опис процесу: користувач натискає кнопку «Додати завдання», система відкриває діалогове вікно з формою. Користувач заповнює обов'язкові поля (title, priority, status) та опціональні (description, due_date, category_id) і натискає «Додати».

Компонент TasksContent викликає метод handleCreateTask, який через клієнтський Supabase-клієнт виконує insert-запит до API Supabase. Supabase перевіряє JWT-токен [4], застосовує політики Row Level Security (користувач може створювати завдання лише з власним user_id), виконує INSERT у таблицю tasks, а потім SELECT із JOIN до таблиці categories для отримання пов'язаної категорії. Отриманий об'єкт Task додається на початок списку завдань, діалог закривається, форма скидається, інтерфейс оновлюється.

Діаграма послідовності: Зміна статусу завдання

На рисунку 2.5 наведено діаграму послідовності процесу зміни статусу завдання.

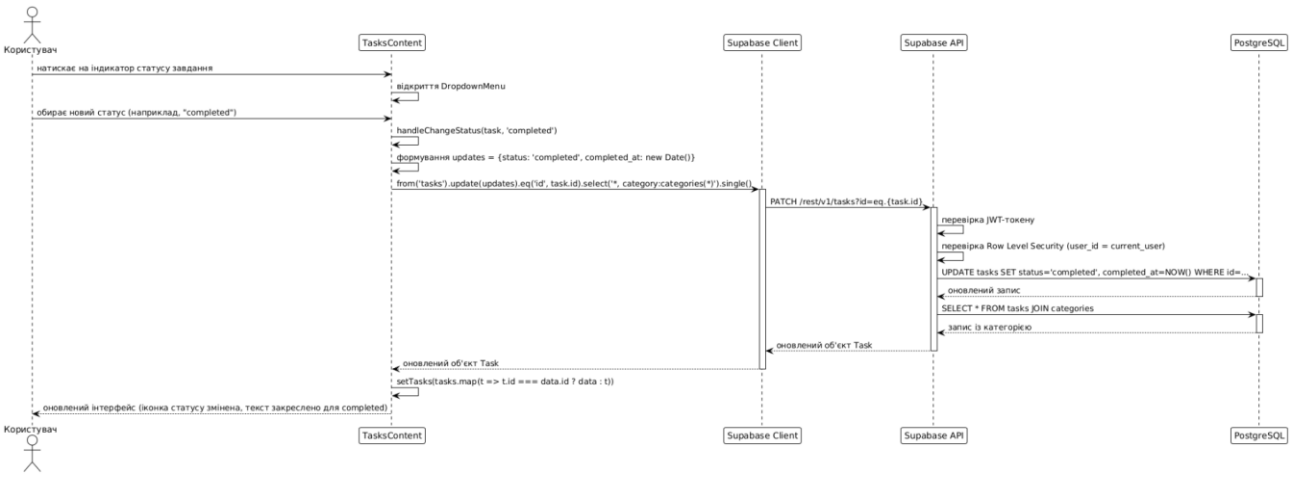

Рисунок 2.1.5 – Діаграма послідовності для зміну статусу завдання

Опис процесу: користувач натискає на індикатор статусу завдання, система відкриває випадаюче меню з трьома варіантами (pending, inprogress, completed). Після вибору нового статусу компонент TasksContent викликає метод handleChangeStatus. Якщо новий статус відрізняється від поточного, формується об'єкт updates: при переході в статус completed поле completed_at встановлюється в поточний час, при переході з completed – скидається в null. Клієнтський Supabase-клієнт виконує update-запит до API Supabase. Supabase перевіряє JWT-токен, застосовує RLS (користувач може змінювати лише власні завдання), виконує UPDATE у таблиці tasks та SELECT із JOIN для отримання оновленого запису.

2.2 Проєктування і розробка інтерфейсу користувача

Проєктування інтерфейсу користувача є критично важливим етапом розробки, оскільки саме через інтерфейс користувач взаємодіє з функціональними можливостями системи [2]. Інтерфейс системи «TaskFlow» спроєктовано з урахуванням принципів мінімалізму, інтуїтивної навігації, візуальної ієрархії та адаптивності. Дизайн базується на концепції «спокійного» інтерфейсу (calm UI), де візуальний шум мінімізовано, а акценти розставлено на ключових елементах взаємодії.

Візуальна мова системи побудована на основі кастомної системи кольорів у просторі OKLCH, що забезпечує кращу доступність та ширший колірний діапазон порівняно з традиційним RGB [5]. Світла тема використовує теплі білі відтінки з акцентами синього (#3B82F6) та смарагдового (#10B981) кольорів. Темна тема побудована на глибоких сланцевих відтінках з тими ж акцентними кольорами для забезпечення візуальної консистентності між темами. На лістингу 2.2.1 показано приклад застосування.

Лістинг 2.2.1 – Код системи кольорів OKLCH

```
--background: oklch(1 0 0);
  --foreground: oklch(0.145 0 0);
  --card: oklch(1 0 0);
  --card-foreground: oklch(0.145 0 0);
  --popover: oklch(1 0 0);
  --popover-foreground: oklch(0.145 0 0);
  --primary: oklch(0.205 0 0);
```

Типографіка базується на шрифті Inter як основному (без засічок, оптимізований для читання з екрану) та Geist Mono для моноширинного тексту. Розміри шрифтів побудовано за модульною шкалою: 12px для допоміжного тексту, 14px для основного тексту, 16px для підзаголовків, 18–20px для заголовків карток, 24–30px для заголовків сторінок.

Радіуси скруглення стандартизовано через CSS-змінну --radius (0.75rem за замовчуванням) із похідними значеннями --radius-sm (0.35rem), --radius-md (0.55rem), --radius-lg (0.75rem), --radius-xl (1.15rem). Усі картки, кнопки, поля введення та діалогові вікна використовують ці значення для забезпечення візуальної єдності.

Для забезпечення візуального зворотного зв'язку реалізовано набір анімацій: fade-in (плавна поява зі зміщенням вгору на 10px), slide-in (поява зі зміщенням вліво на 20px), pulse-soft (м'яка пульсація для індикаторів), card-hover (підняття картки при наведенні). Тривалість анімацій становить 0.3–0.4 секунди з функцією плавності ease-out, що відповідає рекомендаціям Material Design щодо тривалості мікроанімацій. На лістингу 2.2.2 відображено скруглення.

Лістинг 2.2.2 – Використання методу –radius

```
--radius-sm: calc(var(--radius) - 4px);
 --radius-md: calc(var(--radius) - 2px);
 --radius-lg: var(--radius);
 --radius-xl: calc(var(--radius) + 4px);
```

Сторінка входу (/auth/login) є точкою входу для існуючих користувачів, сторінка реєстрації (/auth/sign-up) – для нових користувачів. Обидві сторінки мають мінімалістичний дизайн: форма розташована по центру екрану на напівпрозорому тлі з фоновим градієнтом акцентних кольорів.

Форма входу містить два поля: email (з іконкою поштового конверта) та password (з іконкою замка). Кнопка «Sign In» / «Увійти» має повну ширину форми, основний колір (primary) та анімацію натискання. Під формою розташовано посилання «Don't have an account?» / «Немає облікового запису?», що веде на сторінку реєстрації.

На рисунку 2.2.1 показана сторінка авторизації.

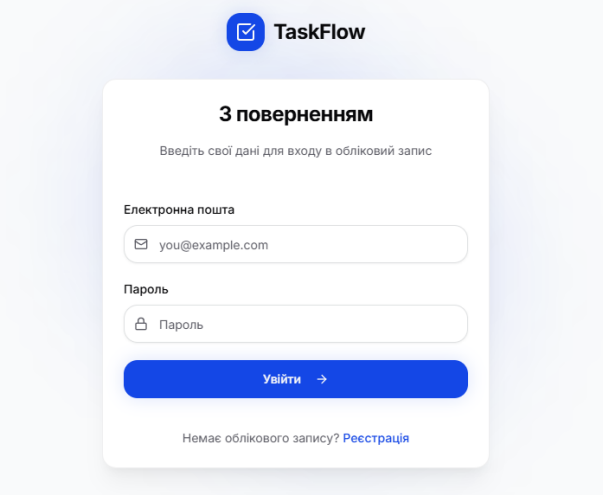

Рисунок 2.2.1 – Сторінка авторизації

Форма реєстрації містить три поля: full name, email, password. Кнопка «Create Account» / «Створити обліковий запис» аналогічна за стилем кнопці входу. Після успішної реєстрації відображається повідомлення про необхідність підтвердження email. На рисунку 2.2.2 є фото форми реєстрації в системі.

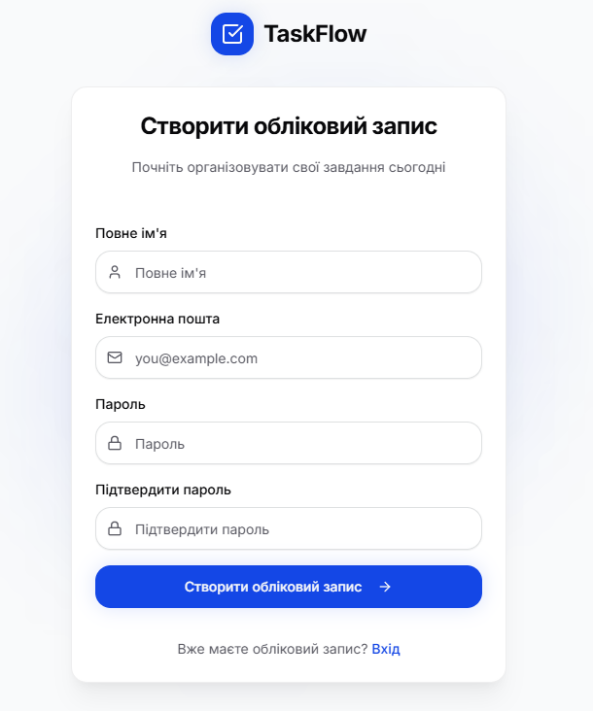

Рисунок 2.2.2 – Форма реєстрації

Головна панель (/dashboard) є основним робочим простором користувача після входу в систему. Сторінка побудована за двоколонковим макетом: ліворуч – бічна панель навігації (Sidebar), праворуч – область контенту.

У верхній частині області контенту розташовано вітання користувача з динамічним текстом залежно від часу доби («Good morning», «Good afternoon», «Good evening» / «Доброго ранку», «Доброго дня», «Доброго вечора») та іменем користувача. Під вітанням – короткий опис поточного стану.

Нижче розташовано чотири статистичні картки в ряд (на мобільних – у два рядки по дві): Total Tasks (загальна кількість завдань, іконка ListTodo, синій акцент), Completed (виконано, іконка CheckCircle2, зелений акцент), In Progress (у процесі, іконка Clock, жовтий акцент), Completion Rate (відсоток виконання, іконка Target, фіолетовий акцент, з індикатором прогресу). Кожна картка має ефект підняття при наведенні (card-hover).

Під статистичними картками – дві колонки з картками середнього розміру. Ліва колонка: «Today's Tasks» (завдання на сьогодні) з переліком завдань, їх статусом (іконка) та пріоритетом (кольоровий бейдж). Якщо завдань на сьогодні немає – відображається повідомлення з кнопкою створення нового завдання. Права колонка: «Upcoming Tasks» (майбутні завдання) з аналогічним форматом, але з додатковим відображенням дати виконання.

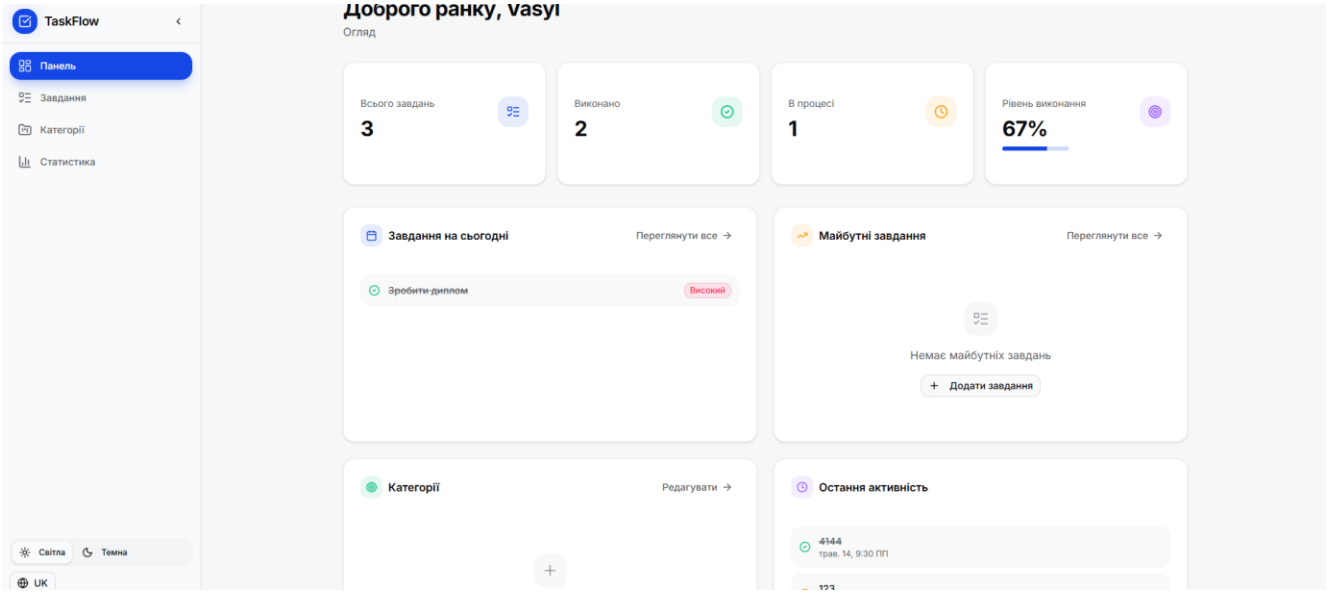

Рисунок 2.2.3 – Вигляд головної сторінки

Сторінка завдань (/tasks) надає повний інтерфейс для управління завданнями. У верхній частині – заголовок «Tasks» / «Завдання» з описом та кнопка «Add Task» / «Додати завдання» (синій колір, іконка плюса) [14].

Під заголовком – рядок фільтрів: поле текстового пошуку з іконкою лупи (пошук за назвою та описом), випадаючі списки для фільтрації за статусом (All, Pending, In Progress, Completed), пріоритетом (All, Low, Medium, High) та категорією. Усі фільтри застосовуються миттєво без перезавантаження сторінки.

На рисунку 2.2.4 показано як виглядає сторінка із завданнями.

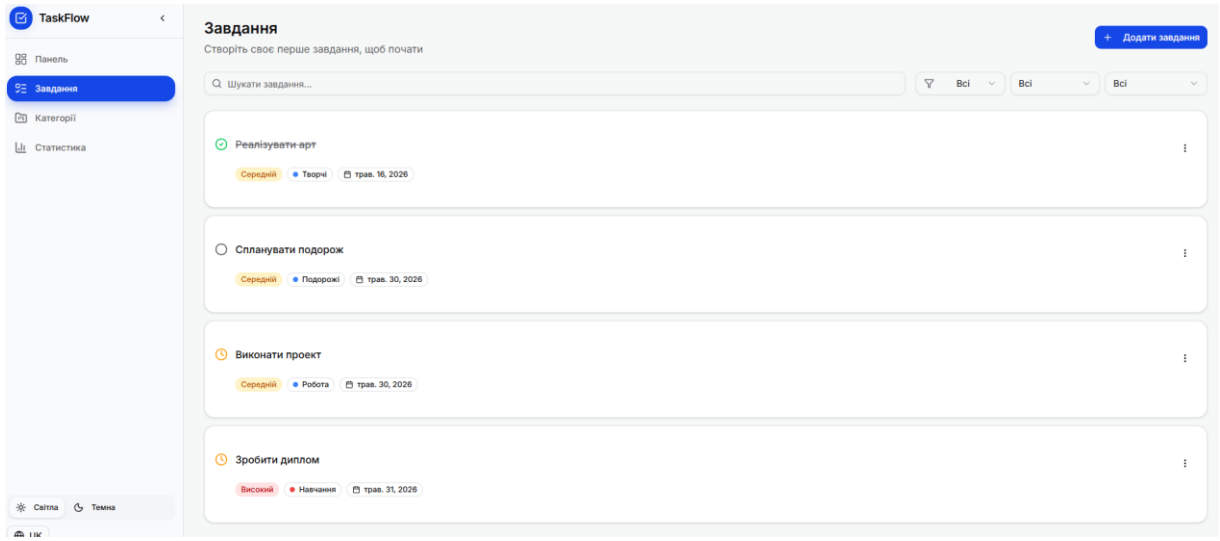

Рисунок 2.2.4 – Сторінка завдань

Список завдань відображається у вигляді вертикального переліку карток. Кожна картка завдання містить: інтерактивний індикатор статусу (кружечок для pending, годинник для inprogress, зелена галочка для completed), назву завдання (закреслену для виконаних), опис (обмежений двома рядками), бейдж пріоритету (червоний для high, жовтий для medium, зелений для low), бейдж категорії (з кольоровим кружечком), бейдж дати виконання (з іконкою календаря, червоний для прострочених). Праворуч – кнопка з трьома крапками для виклику контекстного меню (Edit, Delete) та швидкої зміни статусу.

При натисканні на кнопку «Add Task» відкривається діалогове вікно з формою, що містить поля: Title (обов'язкове), Description (опціональне, текстове поле на 3 рядки), пріоритет (випадаючий список), статус (випадаючий список), Due Date (поле вибору дати), Category (випадаючий список із кольоровими індикаторами категорій). Кнопка «Add Task» / «Додати завдання» закриває діалог і створює завдання.

При натисканні на кнопку редагування відкривається аналогічне діалогове вікно, попередньо заповнене поточними даними завдання. Кнопка «Save» / «Зберегти» оновлює завдання. Рисунок 2.2.5 відображає це діалогове вікно.

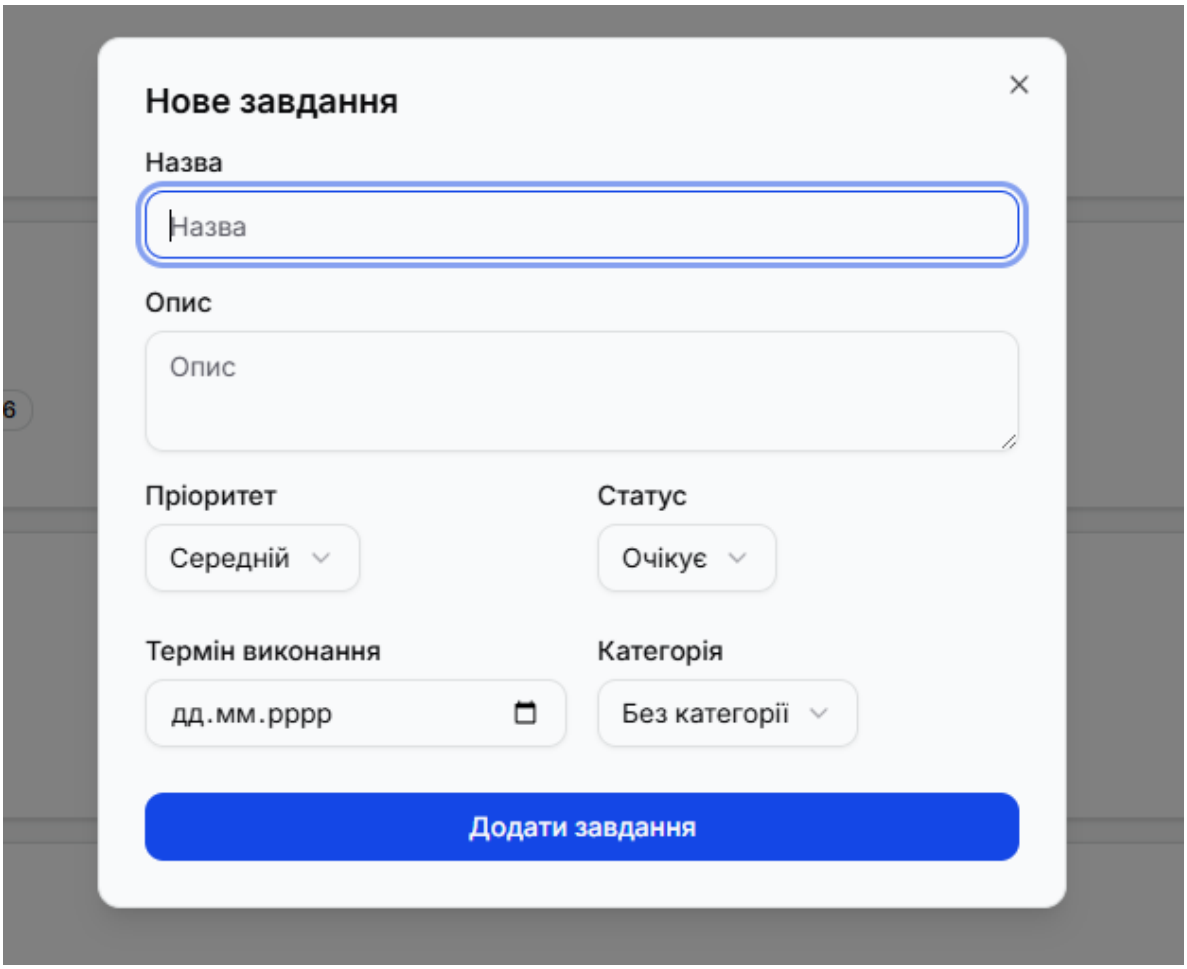

Рисунок 2.2.5 – Діалогове вікно створення завдань

Сторінка категорій (/categories) надає інтерфейс для управління категоріями завдань. Заголовок сторінки – «Categories» / «Категорії» з описом та кнопка «Add Category» / «Додати категорію».

Категорії відображаються у вигляді сітки карток (2–3 колонки залежно від ширини екрану). Кожна картка категорії містить: іконку категорії на кольоровому тлі (напівпрозорий колір категорії), назву категорії, кількість завдань (наприклад, «5 tasks» / «5 завдань»), прогрес-бар із відсотком виконаних завдань та співвідношенням (наприклад, «3/5»). Кнопка з трьома крапками відкриває контекстне меню з опціями «Edit» та «Delete». На рисунку 2.2.4 показано сторінку з категоріями.

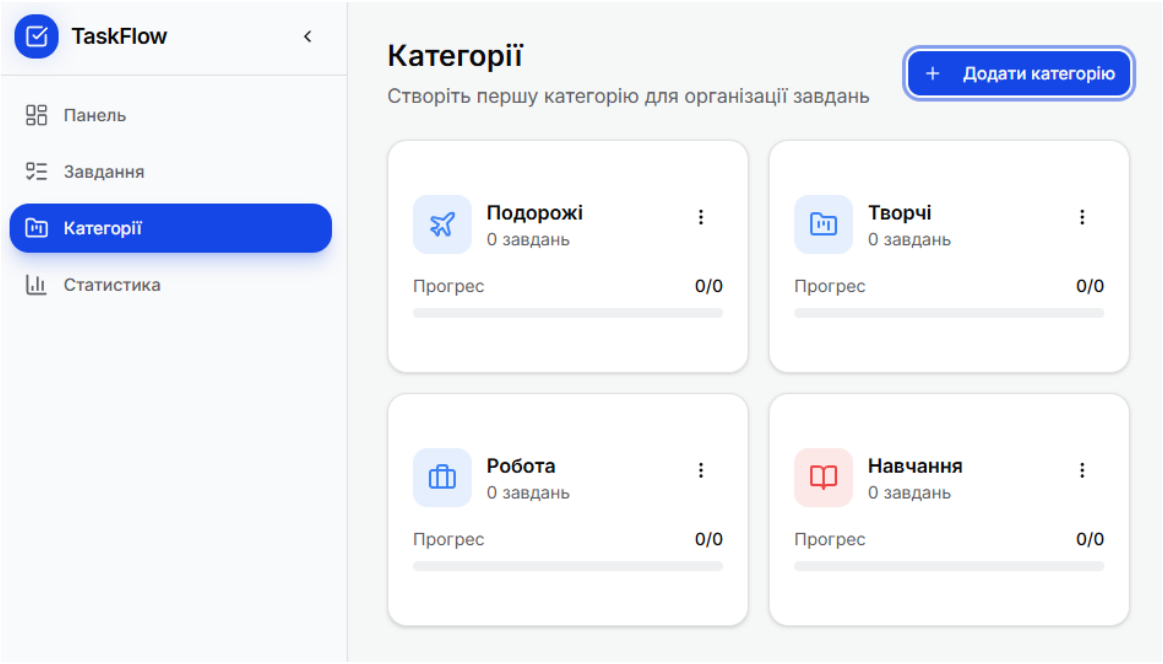

Рисунок 2.2.6 – Сторінка з категоріями

Діалогове вікно створення/редагування категорії містить: поле Name (обов'язкове), сітку вибору кольору (12 передвизначених кольорів, вибраний колір має обведення ring-2), сітку вибору іконки (12 іконок з бібліотеки Lucide: folder, briefcase, home, heart, star, zap, book, coffee, music, camera, plane, cart). Вибрана іконка підсвічується основним кольором.

Для порожнього стану (відсутні категорії) відображається картка з іконкою FolderKanban, текстом «No categories found» / «Категорій не знайдено», описом та кнопкою створення першої категорії.

На рисунку 2.2.7 показано діалогове вікно управління категоріями.

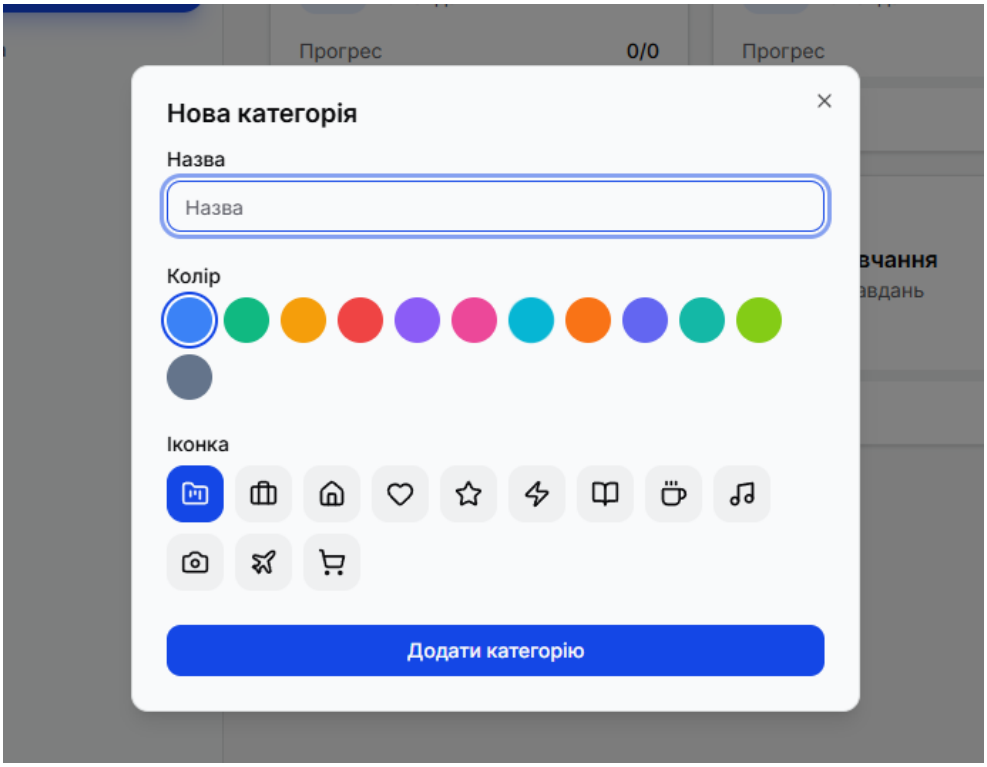

Рисунок 2.2.7 – Додавання категорій

Сторінка статистики (/statistics) надає візуальну аналітику продуктивності користувача. Заголовок – «Statistics» / «Статистика» з описом.

У верхньому рядку – чотири картки з ключовими показниками: Total Tasks, Completed, Completion Rate (відсоток), Overdue (прострочено, червоний акцент) [15].

Основний вміст – сітка з чотирьох графіків (2×2 на десктопах, 1 колонка на мобільних):

1. «Weekly Activity» – стовпчикова діаграма (BarChart) за останні 7 днів, де кожен день має два стовпчики: синій (Created – створені завдання) та зелений (Completed – виконані завдання). Вісь X – дні тижня (Mon, Tue, ...), вісь Y – кількість завдань.

2. «Tasks by Status» – кругова діаграма (PieChart) із внутрішнім радіусом (donut chart), що показує розподіл завдань за статусами: зелений сектор (Completed), жовтий (In Progress), сірий (Pending). Сектори мають підписи з відсотками.

3. «Tasks by Priority» – аналогічна кругова діаграма для розподілу за пріоритетами: червоний (High), жовтий (Medium), зелений (Low).

Під графіками – зведення за термінами виконання у вигляді трьох карток у ряд: Overdue (прострочено, червоний фон), Due Today (сьогодні, жовтий фон), Upcoming (майбутні, синій фон).

Для порожнього стану (відсутність даних) графіки замінюються повідомленням «No data available» / «Дані відсутні» з описом. На рисунку 2.2.8 показано вікно статистики системи.

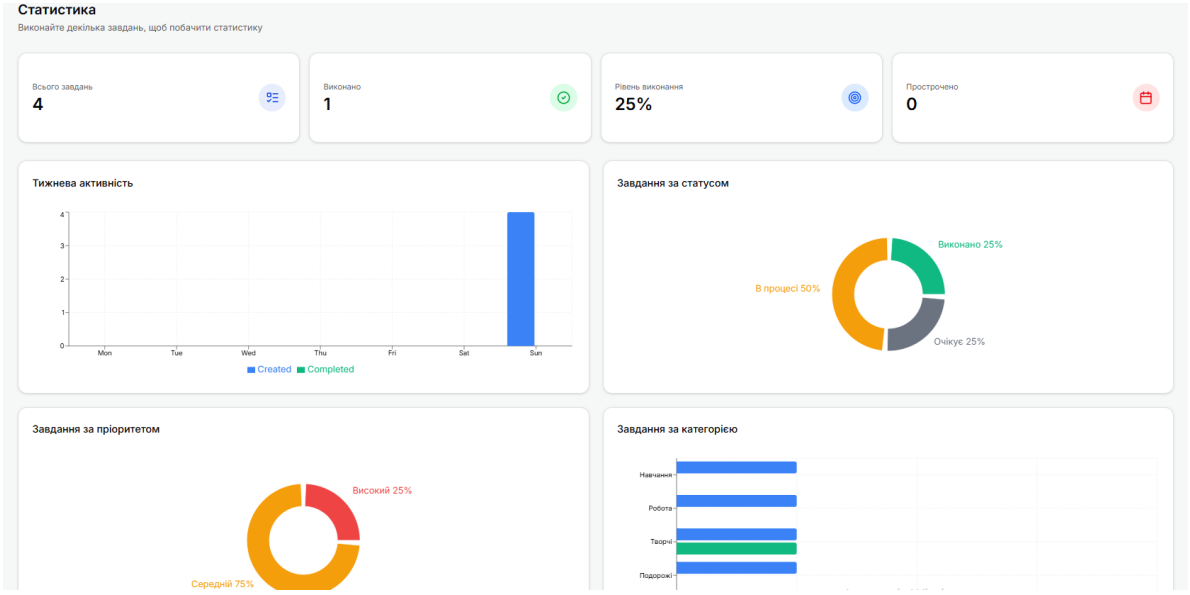

Рисунок 2.2.8 – Вікно статистики

Сторінка профілю (/profile) надає інтерфейс для перегляду та редагування особистих даних. Заголовок – «Profile» / «Профіль» з описом.

Верхня картка – огляд профілю: аватар користувача (коло з ініціалами або зображенням, ring-2 з напівпрозорим основним кольором), повне ім'я, email, дата реєстрації (з іконкою календаря), дві статистичні картки: загальна кількість завдань (іконка ListTodo) та виконаних завдань (іконка CheckCircle2) [14].

Друга картка – форма редагування особистої інформації: поле Full Name (з іконкою користувача), поле Email (заблоковане, з іконкою пошти, сірий фон). Кнопка «Save Changes» / «Зберегти зміни» застосовує оновлення.

Третя картка – дії з обліковим записом: рядок виходу з системи (іконка LogOut, кнопка «Sign Out» / «Вийти») та рядок видалення облікового запису (іконка Trash2, червоний текст, кнопка «Delete» / «Видалити»). При натисканні на видалення відкривається діалогове вікно підтвердження (AlertDialog) із попередженням про незворотність дії.

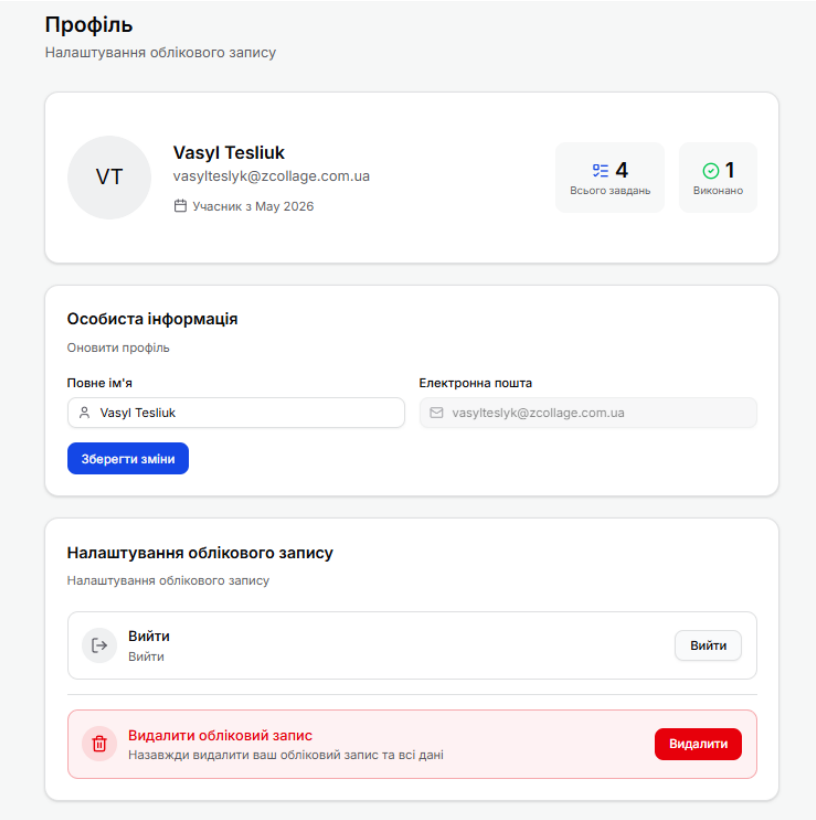

Рисунок 2.2.9 – Вікно профіля користувача

Бічна панель навігації є постійним елементом інтерфейсу для авторизованих користувачів і забезпечує доступ до всіх модулів системи. Панель розташована ліворуч, має фіксовану позицію при прокручуванні (sticky, висота екрану) та складається з чотирьох функціональних зон [5].

Верхня зона – логотип: іконка CheckSquare у кольоровому квадраті (primary колір, ефект світіння glow) та текст «TaskFlow».

Середня зона – навігаційне меню з чотирьох пунктів: Dashboard (іконка LayoutDashboard), Tasks (ListTodo), Categories (FolderKanban), Statistics (BarChart3). Активний пункт підсвічується основним кольором (bg-primary, text-primary-foreground, тінь shadow-lg).

Нижче – перемикач теми (ThemeSwitcher, дві кнопки Sun/Moon) та перемикач мови (LanguageSwitcher, кнопка з глобусом та кодом мови EN/UK). При згорнутій панелі – лише іконки без тексту.

Нижня зона – меню користувача: аватар (з ініціалами), повне ім'я та email. При натисканні відкривається випадаюче меню (DropdownMenu) із пунктами Profile (іконка User), Settings (Settings) та Sign Out (LogOut, червоний текст).

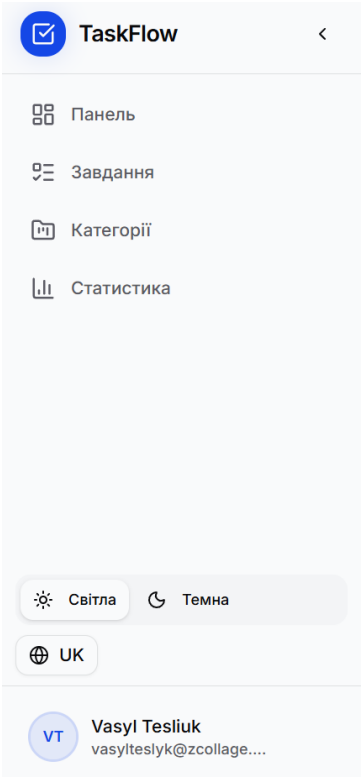

Рисунок 2.2.10 – Бічна панель навігації

Нагадування відображаються у вигляді спливаючих карток (toast-like notifications) у правому нижньому куті екрану (фіксована позиція, z-50). Картки групуються вертикально з проміжком 8px, максимум 5 видимих одночасно.

Кожна картка нагадування має кольорове тло залежно від типу: червоне (bg-red-50) для прострочених завдань, жовте (bg-amber-50) для завдань на сьогодні, синє (bg-blue-50) для завдань на завтра. Картка містить: іконку типу (AlertTriangle для overdue, Bell для today, Clock для tomorrow), мітку типу великими літерами, назву завдання (truncate для довгих назв), дату виконання, кнопку «Mark Done» / «Виконано» (маленька, secondary) та кнопку закриття (X).

При наявності більше одного нагадування з'являється кнопка «Dismiss All» / «Закрити всі» над групою карток. При більше ніж 5 нагадуваннях відображається текст «+N more reminders» / «ще N нагадувань».

Нагадування з'являються з анімацією slide-in-right, зникають при закритті або позначенні завдання виконаним.

На рисунку 2.2.11 показано сповіщення з нагадуванням про завдання.

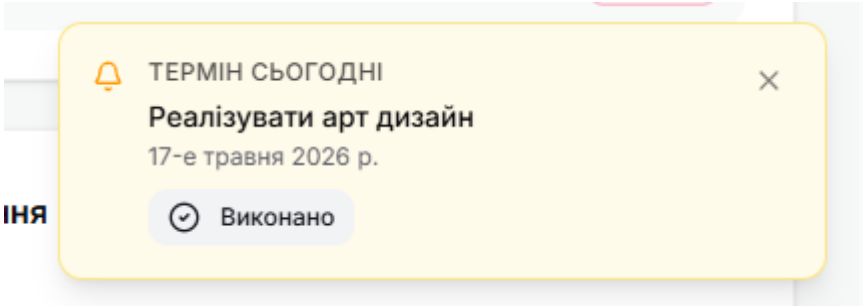

Рисунок 2.2.11 – Сповіщення з нагадуванням

2.3 Розробка бекенда

Бекенд система «TaskFlow» реалізована з використанням Next.js API Routes та серверних компонентів у поєднанні з Supabase як Backend-as-a-Service [3]. Така архітектура дозволяє мінімізувати обсяг серверного коду, делегуючи автентифікацію, управління базою даних та безпеку на сторону Supabase, водночас зберігаючи контроль над бізнес-логікою через серверні компоненти Next.js [3].

Захист маршрутів реалізовано через middleware Next.js, який виконується перед кожним запитом до сервера. Файл middleware.ts у корені проєкту імпортує функцію updateSession з модуля /lib/supabase/middleware:

Лістинг 2.3.1 – Middleware автентифікація

```
import { updateSession } from '@/lib/supabase/middleware'
import { type NextRequest } from 'next/server'

export async function middleware(request: NextRequest) {
  return await updateSession(request)
}

export const config = {
  matcher: [

'/((?!_next/static|_next/image|favicon.ico|.*\\.(?:svg|png|jpg|jpeg|
gif|webp)$).*)',
    ],
}
```

Конфігурація matcher визначає, що middleware застосовується до всіх маршрутів, крім статичних файлів та зображень. Такий підхід забезпечує перевірку автентифікації для кожного запиту без навантаження на доставку статичних ресурсів [4].

Функція updateSession у файлі /lib/supabase/middleware виконує такі завдання:

Лістинг 2.3.2 – Конфігурація matcher

```
import { createServerClient } from '@supabase/ssr'
import { NextResponse, type NextRequest } from 'next/server'

export async function updateSession(request: NextRequest) {
  let supabaseResponse = NextResponse.next({ request })

  const supabase = createServerClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!,
    {
      cookies: {
        getAll() { return request.cookies.getAll() },
        setAll(cookiesToSet) {
          cookiesToSet.forEach(({ name, value }) => request.cookies.set(name, value))
          supabaseResponse = NextResponse.next({ request })
          cookiesToSet.forEach(({ name, value, options }) =>
            supabaseResponse.cookies.set(name, value, options)
          )
        },
      },
    }
  )

  const { data: { user } } = await supabase.auth.getUser()
```

Логіка middleware реалізує два правила маршрутизації. Перше правило: якщо користувач не автентифікований (user === null) і намагається отримати доступ до захищеного маршруту (dashboard, tasks, categories, statistics, profile), виконується перенаправлення на сторінку входу /auth/login. Друге правило: якщо користувач уже автентифікований і намагається відвідати сторінки входу або реєстрації, виконується перенаправлення на головну панель /dashboard. Такий підхід запобігає повторній автентифікації активного користувача [4].

Для інтерактивних операцій на стороні браузера реалізовано клієнтський клієнт у файлі lib/supabase/client.ts:

Лістинг 2.3.3 – Клієнтський сервер Supabase

```
import { createBrowserClient } from '@supabase/ssr'
export function createClient() {
  return createBrowserClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!
  )
}
```

Клієнтський клієнт використовує createBrowserClient, який автоматично працює з cookies браузера для управління сесією. На відміну від серверного клієнта, тут не потрібна ручна робота з cookies – Supabase самостійно зчитує та оновлює сесійні cookies через браузерне API [3]. Клієнтський клієнт має обмежені права: всі запити проходять через Row Level Security, що гарантує доступ лише до даних поточного користувача.

Кожна сторінка системи використовує серверний компонент Next.js для початкового завантаження даних. Розглянемо реалізацію серверної сторінки головної панелі (app/(dashboard)/dashboard/page.tsx):

Лістинг 2.3.4 – Реалізація серверної сторінки

```
import { createClient } from '@/lib/supabase/server'
import { redirect } from 'next/navigation'
import { LandingContent } from '@/components/landing-content'

export default async function HomePage() {
  const supabaseUrl = process.env.NEXT_PUBLIC_SUPABASE_URL
  const supabaseAnonKey = process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY

  if (supabaseUrl && supabaseAnonKey) {
    try {
      const supabase = await createClient()
      const { data: { user } } = await supabase.auth.getUser()

      if (user) {
        redirect('/dashboard')
      }
    } catch {
      // Помилка Supabase – показуємо лендінг
    }
  return <LandingContent />
}
```

Сторінка спочатку перевіряє наявність змінних оточення Supabase. Якщо змінні відсутні (локальна розробка без налаштованої бази даних), одразу рендериться лендінг-сторінка. Якщо змінні наявні, створюється серверний клієнт Supabase, отримується поточний користувач через getUser(). Автентифікований користувач перенаправляється на /dashboard, неавтентифікований – бачить лендінг [4].

Серверна сторінка dashboard (app/(dashboard)/dashboard/page.tsx) виконує паралельне завантаження даних [4]:

Лістинг 2.3.5 – Серверна сторінка dashboard

```
export default async function DashboardPage() {
  const supabase = await createClient()
  const { data: { user } } = await supabase.auth.getUser()

  if (!user) {
    redirect('/auth/login')
  }

  const { data: profile } = await supabase
    .from('profiles')
    .select('*')
    .eq('id', user.id)
    .single()

  const { data: tasks } = await supabase
    .from('tasks')
    .select('*, category:categories(*)')
    .eq('user_id', user.id)
    .order('created_at', { ascending: false })

  const { data: categories } = await supabase
    .from('categories')
    .select('*')
    .eq('user_id', user.id)
    .order('created_at', { ascending: false })
    />
  )
}
```

Сторінка виконує три послідовні запити до бази даних. Перший запит отримує профіль користувача з таблиці profiles за id поточного користувача. Метод single() вказує, що очікується рівно один запис. Другий запит отримує всі завдання користувача з приєднанням категорії через select('*, category:categories(*)') [4].

Це спеціальний синтаксис Supabase, який виконує JOIN із таблицею `categories` за зовнішнім ключем `category_id`. Результати сортуються за датою створення у зворотному порядку.

Операції створення, оновлення та видалення завдань виконуються на клієнтській стороні через Supabase API. Розглянемо реалізацію методу створення завдання у компоненті TasksContent:

Лістинг 2.3.6 – Реаліазція сторінки TasksContent

```
const handleCreateTask = async (e: React.FormEvent) => {
  e.preventDefault()
  setIsLoading(true)

  const { data, error } = await supabase
    .from('tasks')
    .insert({
      user_id: userId,
      title,
      description: description || null,
      priority,
      status,
      due_date: dueDate || null,
      category_id: categoryId && categoryId !== 'none' ?
categoryId : null,
    })
    .select('*, category:categories(*)')
    .single()
```

Метод insert() створює новий запис у таблиці tasks. Поле user_id заповнюється значенням, отриманим із сесії. Поля description, due_date, category_id можуть бути null. Значення 'none' для categoryId перетворюється на null – це дозволяє створити завдання без категорії. Ланцюжок .select('*, category:categories(*)').single() виконує JOIN із таблицею категорій після вставки та повертає повний об'єкт завдання з вкладеною категорією. У разі успіху завдання додається на початок списку [data, ...tasks], діалог закривається, форма скидається.

Метод оновлення завдання реалізовано з додатковою логікою для відстеження часу завершення:

Лістинг 2.3.7 – Логіка відстеження часу завершення

```
const handleUpdateTask = async (e: React.FormEvent) => {
  e.preventDefault()
  if (!editingTask) return
  setIsLoading(true)

  const updates: Partial<Task> = {
    title,
    description: description || null,
    priority,
    status,
    due_date: dueDate || null,
    category_id: categoryId && categoryId !== 'none' ? categoryId
: null,
  }

  if (status === 'completed' && editingTask.status !==
'completed') {
    updates.completed_at = new Date().toISOString()
  } else if (status !== 'completed') {
    updates.completed_at = null
  }
```

Ключова особливість – автоматичне управління полем completed_at. Якщо статус змінено на 'completed' і попередній статус був іншим, у completed_at записується поточний час (new Date().toISOString()). Якщо статус змінено з 'completed' на інший, completed_at скидається в null. Це забезпечує коректне відстеження часу виконання завдань для аналітики [4].

Метод швидкої зміни статусу (handleChangeStatus) використовує аналогічну логіку, але без відкриття форми редагування:

Лістинг 2.3.8 – Реалізація логіки зміни статусу завдання

```
const handleChangeStatus = async (task: Task, newStatus: TaskStatus) => {
  if (task.status === newStatus) return

  const updates: Partial<Task> = {
    status: newStatus,
    completed_at: newStatus === 'completed' ? new Date().toISOString() : null,
  }

  const { data, error } = await supabase
    .from('tasks')
    .update(updates)
    .eq('id', task.id)
    .select('*, category:categories(*)')
    .single()

  if (!error && data) {
    setTasks(tasks.map(t => t.id === data.id ? data : t))
  }
}
```

Метод видалення завдання виконує delete() із фільтром за id:

Лістинг 2.3.9 – Метод видалення завдання

```
const handleDeleteTask = async (taskId: string) => {
  const { error } = await supabase
    .from('tasks')
    .delete()
    .eq('id', taskId)

  if (!error) {
    setTasks(tasks.filter(t => t.id !== taskId))
  }
}
```

Усі три методи використовують оптимістичне оновлення інтерфейсу: зміни в стані компонента відбуваються лише після успішної відповіді від сервера, що гарантує консистентність даних.

Створення категорії реалізовано в компоненті CategoriesContent:

Лістинг 2.3.10 – Реалізація методів керування категоріями

```
const handleCreateCategory = async (e: React.FormEvent) => {
  e.preventDefault()
  setIsLoading(true)

  const { data, error } = await supabase
    .from('categories')
    .insert({
      user_id: userId,
      name,
      color,
      icon: iconName,
    })
    .select()
    .single()

  if (!error && data) {
    setCategories([data, ...categories])
  setIsLoading(false)
}
```

Категорія створюється з обов'язковими полями name, color, icon та автоматично підставленим user_id. Поле color зберігається у форматі HEX (наприклад, '#3B82F6'), поле icon – рядковий ідентифікатор іконки (наприклад, 'folder'). Оновлення та видалення категорій реалізовані аналогічно до відповідних операцій із завданнями.

Оновлення профілю реалізовано в компоненті ProfileContent:

Лістинг 2.3.11 – Реалізація компоненти профіля користувача

```
const handleUpdateProfile = async (e: React.FormEvent) => {
  e.preventDefault()
  setIsLoading(true)
  setMessage(null)

  const { error } = await supabase
    .from('profiles')
    .update({ full_name: fullName })
    .eq('id', user.id)

  if (error) {
    setMessage({ type: 'error', text: 'Failed to update profile' })
  } else {
    setMessage({ type: 'success', text: 'Profile updated successfully' })
    router.refresh()
  }
  setIsLoading(false)
}
```

Оновлення обмежене полем full_name – це єдине поле профілю, яке користувач може редагувати. Email змінювати заборонено (поле заблоковане в інтерфейсі). Після успішного оновлення викликається router.refresh() для оновлення серверних компонентів [4].

Вихід із системи реалізовано через supabase.auth.signOut() із подальшим перенаправленням на сторінку входу:

Лістинг 2.3.12 – Реалізація методу виходу з профілю

```
const handleLogout = async () => {
  await supabase.auth.signOut()
  router.push('/auth/login')
  router.refresh()
}
```

Система нагадувань реалізована в компоненті TaskReminders і виконує періодичну перевірку дедлайнів:

Лістинг 2.3.13 – Реалізація методів для нагадування

```
useEffect(() => {
  const fetchTasks = async () => {
    const { data: tasks } = await supabase
      .from('tasks')
      .select('*')
      .eq('user_id', userId)
      .neq('status', 'completed')
      .not('due_date', 'is', null)

    if (tasks) {
      const newReminders: Reminder[] = []
      const today = new Date()
      today.setHours(0, 0, 0, 0)

      tasks.forEach((task: Task) => {
        if (!task.due_date) return
        const dueDate = parseISO(task.due_date)
        dueDate.setHours(0, 0, 0, 0)
```

Запит отримує всі незавершені завдання з встановленим терміном виконання. Для кожного завдання обчислюється тип нагадування: overdue (дата в минулому), due_today (дата сьогодні), due_tomorrow (дата завтра). Нагадування оновлюються кожні 5 хвилин через setInterval [16].

Безпека даних забезпечується на рівні бази даних через механізм Row Level Security (RLS). Для кожної таблиці визначено політики доступу, які гарантують, що користувач може бачити та змінювати лише власні дані.

Для таблиці tasks визначено такі політики: SELECT – дозволено читати записи, де user_id дорівнює ID поточного користувача; INSERT – дозволено створювати записи з user_id , що дорівнює ID поточного користувача; UPDATE – дозволено оновлювати записи, де user_id дорівнює ID поточного користувача; DELETE – дозволено видаляти записи, де user_id дорівнює ID поточного користувача. Аналогічні політики визначено для таблиць user_id та profiles.

Усі запити до Supabase API автоматично включають JWT-токен із сесійних cookies. Supabase на основі цього токену визначає поточного користувача та застосовує політики RLS. Це означає, що навіть якщо зловмисник модифікує клієнтський код і спробує виконати запит до чужих даних, сервер відхилить запит на рівні бази даних.

Глобальний стан застосунку управляється через React Context API. Реалізовано два контексти: ThemeContext для теми оформлення та LanguageContext для мови інтерфейсу.

ThemeContext зберігає поточну тему ('light' | 'dark') та надає методи setTheme і toggleTheme:

Лістинг 2.3.14 – Контексти управління станом

```
const [theme, setThemeState] = useState<Theme>('dark')

useEffect(() => {
  const saved = localStorage.getItem('taskflow-theme') as Theme
  if (saved && (saved === 'light' || saved === 'dark')) {
    setThemeState(saved)
    document.documentElement.classList.toggle('dark', saved === 'dark')
  }
}, [])

const setTheme = (newTheme: Theme) => {
  setThemeState(newTheme)
  localStorage.setItem('taskflow-theme', newTheme)
  document.documentElement.classList.toggle('dark', newTheme === 'dark')
```

При ініціалізації тема завантажується з localStorage. При зміні теми оновлюється стан, зберігається значення в localStorage та додається/видаляється клас dark на кореневому елементі. Це запускає перерахунок CSS-змінних і зміну всіх кольорів інтерфейсу.

LanqugageContext реалізовано аналогічно: мова завантажується з LocalStorage, при зміні оновлюється стан і зберігається значення. Об'єкт translations містить усі текстові рядки англійською та українською мовами. Компоненти отримують переклади через хук useLanguage();

2.4 Інтеграція та збереження даних

Інтеграція та збереження даних у системі «TaskFlow» охоплює механізми взаємодії клієнтської частини з базою даних PostgreSQL через Supabase API, структуру збереження даних, політики безпеки на рівні рядків, обробку зв'язків між сутностями та управління станом даних на клієнті [4]. У цьому підрозділі детально розглянуто архітектуру збереження даних, формати передачі даних, механізми кешування та оптимістичного оновлення інтерфейсу.

Система використовує реляційну базу даних PostgreSQL, розміщену на платформі Supabase. Вибір PostgreSQL зумовлений необхідністю забезпечення цілісності даних через зовнішні ключі, підтримкою складних запитів із JOIN та вбудованою підтримкою JSON-функцій. Supabase надає REST API, яке автоматично генерується на основі схеми бази даних, що усуває потребу в написанні серверних ендпоінтів вручну [3].

Взаємодія з базою даних відбувається через два канали. Серверний канал використовується в серверних компонентах Next.js для початкового завантаження даних під час SSR. Клієнтський канал використовується в інтерактивних компонентах для операцій створення, оновлення та видалення даних. Обидва канали використовують один і той самий API Supabase, але з різними рівнями доступу: серверний клієнт має розширені права для SSR, клієнтський клієнт обмежений політиками Row Level Security.

База даних системи складається з чотирьох таблиць: службової auth.users (керованої Supabase Auth) та трьох таблиць предметної області – profiles, tasks, categories [4].

Таблиця profiles має таку структуру:

Таблиця 2.4.1 – Структура profiles

Поле	Тип	Обмеження	Опис
id	uuid	PRIMARY KEY, REFERENCES auth.users(id) ON DELETE CASCADE	Первинний ключ, зв'язаний із користувачем
full_name	varchar(255)	NULLABLE	Повне ім'я користувача
avatar_url	text	NULLABLE	URL-адреса аватара
created_at	timestamptz	NOT NULL, DEFAULT now()	Час створення профілю
updated_at	timestamptz	NULLABLE	Час останнього оновлення

Зв'язок із auth.users реалізовано через ON DELETE CASCADE, що забезпечує автоматичне видалення профілю при видаленні користувача [3].

Таблиця categories має таку структуру:

Таблиця 2.4.2 – Структура categories

Поле	Тип	Обмеження	Опис
id	uuid	PRIMARY KEY, DEFAULT gen_random_uuid()	Первинний ключ
user_id	uuid	NOT NULL, REFERENCES auth.users(id) ON DELETE CASCADE	Зовнішній ключ до користувача
name	varchar(100)	NOT NULL	Назва категорії
color	varchar(7)	NOT NULL	Колір у форматі HEX (#RRGGBB)
icon	varchar(50)	NOT NULL	Ідентифікатор іконки Lucide
created_at	timestamptz	NOT NULL, DEFAULT now()	Час створення категорії

Таблиця tasks має таку структуру:

Таблиця 2.4.3 – Стуктура tasks

Поле	Тип	Обмеження	Опис
id	uuid	PRIMARY KEY, DEFAULT gen_random_uuid()	Первинний ключ
user_id	uuid	NOT NULL, REFERENCES auth.users(id) ON DELETE CASCADE	Зовнішній ключ до користувача
category_id	uuid	NULLABLE, REFERENCES categories(id) ON DELETE SET NULL	Зовнішній ключ до категорії
title	varchar(255)	NOT NULL	Назва завдання
description	text	NULLABLE	Опис завдання
priority	varchar(10)	NOT NULL, DEFAULT 'medium', CHECK (priority IN ('low','medium','high'))	Пріоритет завдання
status	varchar(15)	NOT NULL, DEFAULT 'pending', CHECK (status IN ('pending','inprogress','completed'))	Статус завдання
due_date	timestamptz	NULLABLE	Термін виконання
completed_at	timestamptz	NULLABLE	Час завершення завдання

Для забезпечення ізоляції даних між користувачами налаштовано політики Row Level Security (RLS) для всіх таблиць предметної області. RLS – це механізм PostgreSQL, який дозволяє визначати правила доступу на рівні окремих рядків таблиці [4].

Для таблиці profiles визначено такі політики:

Лістинг 2.4.1 – Політика для profiles

```
-- Дозволити читання власного профілю
CREATE POLICY "Users can view own profile"
ON profiles FOR SELECT
USING (auth.uid() = id);

-- Дозволити оновлення власного профілю
CREATE POLICY "Users can update own profile"
ON profiles FOR UPDATE
USING (auth.uid() = id);

-- Дозволити вставку власного профілю
CREATE POLICY "Users can insert own profile"
ON profiles FOR INSERT
WITH CHECK (auth.uid() = id);
```

Функція auth.uid() повертає ID поточного автентифікованого користувача з JWT-токену. Політика SELECT використовує USING (auth.uid() = id), що означає: користувач може прочитати лише той рядок, де id збігається з його власним ID. Політика INSERT використовує WITH CHECK (auth.uid() = id) для перевірки, що створюваний запис має правильний id [2].

Для таблиці categories політики використовують перевірку за полем user_id:

Лістинг 2.4.2 – Політика для categories

```
-- Дозволити читання власних категорій
CREATE POLICY "Users can view own categories"
ON categories FOR SELECT
USING (auth.uid() = user_id);

-- Дозволити створення власних категорій
CREATE POLICY "Users can create own categories"
ON categories FOR INSERT
WITH CHECK (auth.uid() = user_id);
```

Система використовує можливості Supabase для роботи зі зв'язками між таблицями. Основним механізмом є синтаксис вкладених ресурсів у запитах. При отриманні завдань використовується select('*, category:categories(*)'), що виконує LEFT JOIN із таблицею categories. Це дозволяє отримати повну інформацію про категорію завдання в одному запиті без додаткових звернень до бази даних.

Для забезпечення консистентності даних при видаленні категорій використовується каскадна поведінка на рівні зовнішніх ключів. Для зв'язку tasks.category_id -> categories.id встановлено ON DELETE SET NULL. Це означає, що при видаленні категорії всі пов'язані завдання залишаються в системі, але їх поле category_id автоматично встановлюється в NULL. Такий підхід запобігає втраті даних завдань при реорганізації категорій.

Для зв'язку tasks.user_id -> auth.users.id та categories.user_id -> auth.users.id встановлено ON DELETE CASCADE. Це забезпечує автоматичне видалення всіх завдань і категорій користувача при видаленні його облікового запису, що відповідає вимогам щодо повного видалення даних.

Клієнтська частина системи використовує локальний стан React для збереження даних після їх отримання з сервера. Кожен компонент, що працює з даними, отримує початковий набір даних через props від серверного компонента, після чого управляє станом локально.

У компоненті TasksContent стан ініціалізується з initialTasks:

Лістинг 2.4.3 – Ініціалізація TasksContent

```
const [tasks, setTasks] = useState<Task[]>(initialTasks)
```

При створенні нового завдання стан оновлюється шляхом додавання нового об'єкта на початок масиву:

Лістинг 2.4.4 – Додавання нового об’єкта

```
if (!error && data) {
  setTasks([data, ...tasks])
}
```

Налаштування користувача, які не потребують збереження на сервері, зберігаються в локальному сховищі браузера (localStorage). До таких налаштувань належать тема оформлення та мова інтерфейсу.

Збереження теми реалізовано в ThemeContext:

Лістинг 2.4.5 – Реалізація збереження теми

```
const setTheme = (newTheme: Theme) => {
  setThemeState(newTheme)
  localStorage.setItem('taskflow-theme', newTheme)
  document.documentElement.classList.toggle('dark', newTheme ===
'dark')
}
```

При зміні теми значення записується в localStorage із ключем 'taskflow-theme'. При завантаженні застосунку тема відновлюється:

Лістинг 2.4.6 – Механізм відновлення теми

```
useEffect(() => {
  const saved = localStorage.getItem('taskflow-theme') as Theme
  if (saved && (saved === 'light' || saved === 'dark')) {
    setThemeState(saved)
    document.documentElement.classList.toggle('dark', saved === 'dark')
  }
}, [])
```

Аналогічно реалізовано збереження мови в LanguageContext із ключем 'taskflow-language'. Використання localStorage забезпечує збереження налаштувань між сесіями без необхідності серверного збереження [16].

Хоча система не використовує механізм реального часу Supabase (Realtime), нагадування реалізують періодичне оновлення даних через інтервал. Компонент TaskReminders виконує запит до бази даних кожні 5 хвилин:

Лістинг 2.4.7 – Реалізація TaskReminders

```
useEffect(() => {
  const fetchTasks = async () => {
    const { data: tasks } = await supabase
      .from('tasks')
      .select('*')
      .eq('user_id', userId)
      .neq('status', 'completed')
      .not('due_date', 'is', null)
    // Обробка результатів
  }
```

Такий підхід забезпечує актуалізацію нагадувань без надмірного навантаження на сервер. Інтервал у 5 хвилин обрано як компроміс між актуальністю даних та частотою запитів.

Валідація даних реалізована на двох рівнях. На рівні бази даних обмеження CHECK, NOT NULL та FOREIGN KEY гарантують цілісність даних. На рівні клієнта валідація виконується через атрибути HTML-форм (required, type="date") та через логіку компонентів.

У формі створення завдання поле title є обов'язковим (required), поле due_date використовує вбудований браузерний вибір дати (type="date"), що гарантує коректний формат.

Лістинг 2.4.8 – Валідація даних

```
category_id: categoryId && categoryId !== 'none' ? categoryId : null
```

Поля priority та status обмежені значеннями з випадаючих списків, що унеможливлює введення невалідних даних.

2.5 Висновок до другого розділу

У другому розділі виконано структурне та архітектурне проєктування системи «TaskFlow». Побудовано діаграму класів, що відображає три основні сутності (User, Task, Category), допоміжну сутність Profile та два перелічувані типи (Priority, TaskStatus). Визначено атрибути, методи та зв'язки між класами: один-до-одного (User — Profile), один-до-багатьох (User — Task, User — Category, Category — Task).

Розроблено ER-діаграму бази даних, що включає чотири таблиці: службову auth.users та три таблиці предметної області — profiles, tasks, categories. Для кожної таблиці визначено типи даних, обмеження цілісності (PRIMARY KEY, FOREIGN KEY, NOT NULL, CHECK), каскадні правила видалення та оновлення. Побудовано діаграми послідовностей для операцій автентифікації, створення завдання та зміни статусу.

Спроєктовано інтерфейс користувача: описано візуальну мову (система кольорів OKLCH, типографіка Inter, радіуси скруглення, анімації), структуру десяти основних екранів (сторінки входу та реєстрації, головна панель, список завдань, категорії, статистика, профіль, бічна панель, система нагадувань). Інтерфейс реалізовано з урахуванням принципів адаптивності (від 320px до 2560px), підтримки світлої та темної тем, української та англійської локалізації.

Реалізовано бекенд системи: middleware автентифікації, серверний та клієнтський клієнти Supabase, серверні сторінки з отриманням даних через SSR, методи створення, оновлення, видалення завдань і категорій із відстеженням часу завершення, систему нагадувань із періодичним опитуванням, контексти управління темою та мовою.

Описано механізми інтеграції та збереження даних: структуру таблиць із типами полів, політики Row Level Security для ізоляції даних користувачів,

3 ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ

У цьому розділі проведено тестування функціоналу системи за 30 сценаріями, виконано вимірювання продуктивності засобами Lighthouse, проаналізовано відповідність системи нефункціональним вимогам, розглянуто питання підтримки та можливостей подальшого розширення.

3.1 Тестування функціоналу

Тестування функціоналу системи «TaskFlow» спрямоване на перевірку відповідності реалізованих можливостей функціональним вимогам, визначеним у розділі 1. Метою тестування є виявлення дефектів, перевірка коректності роботи всіх модулів та підтвердження готовності системи до використання [19]. Тестування проводилося методами ручного функціонального тестування з використанням тестових сценаріїв, що охоплюють усі ключові операції системи.

Реєстрація нового користувача.

Кроки виконання: відкрити сторінку /auth/sign-up; ввести унікальну електронну пошту, пароль та повне ім'я; натиснути кнопку «Create Account». Очікуваний результат: система створює обліковий запис, відображає повідомлення про необхідність підтвердження email, запис у таблиці auth.users створено, запис у таблиці profiles створено з введеним іменем. Фактичний результат: відповідає очікуваному [4].

Рисунок 3.1.1 – Результат тестування реєстрації

Вхід із коректними обліковими даними

Кроки: відкрити сторінку /auth/login; ввести email та пароль зареєстрованого користувача; натиснути «Sign In». Очікуваний результат: користувач перенаправляється на /dashboard, у cookies встановлено сесійний токен. Фактичний результат: відповідає очікуваному.

Вхід із некоректними обліковими даними.

Кроки: відкрити /auth/login; ввести неіснуючий email або неправильний пароль; натиснути «Sign In». Очікуваний результат: відображається повідомлення про помилку автентифікації, користувач залишається на сторінці входу. Фактичний результат: відповідає очікуваному. Результат виконання показано на рисунку 3.1.2.

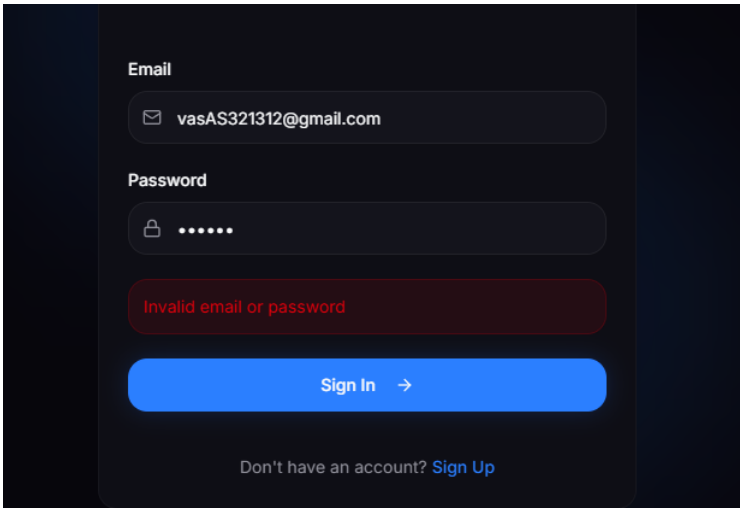

Рисунок 3.1.2 – Тестування валідації авторизації

Захист маршрутів від неавторизованого доступу.

Кроки: не авторизуючись у системі, спробувати відкрити /dashboard, /tasks, /categories, /statistics, /profile. Очікуваний результат: кожен маршрут перенаправляє на /auth/login. Фактичний результат: відповідає очікуваному [17].

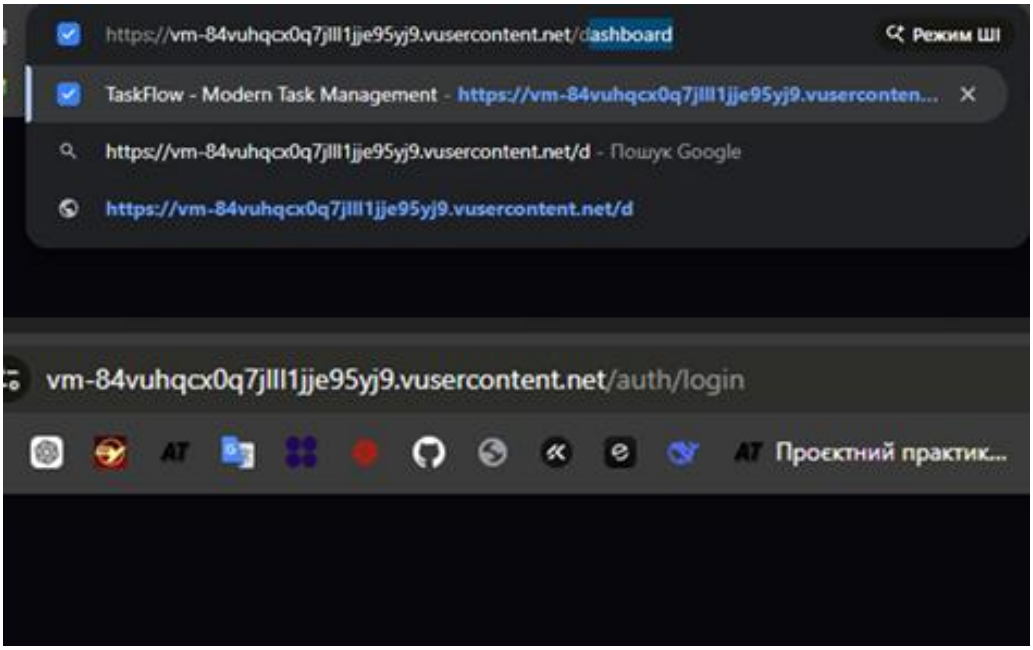

Рисунок 3.1.3 – Захист маршрутів

Створення завдання з усіма полями.

Кроки: відкрити сторінку /tasks; натиснути «Add Task»; заповнити title, description, priority = High, status = Pending, due_date = наступний тиждень, category = вибрати зі списку; натиснути «Add Task». Очікуваний результат: діалог закривається, завдання з'являється на початку списку, усі поля відповідають введеним, категорія відображається з кольором та іконкою. Фактичний результат: відповідає очікуваному.

Результат показано на рисунку 3.1.4.

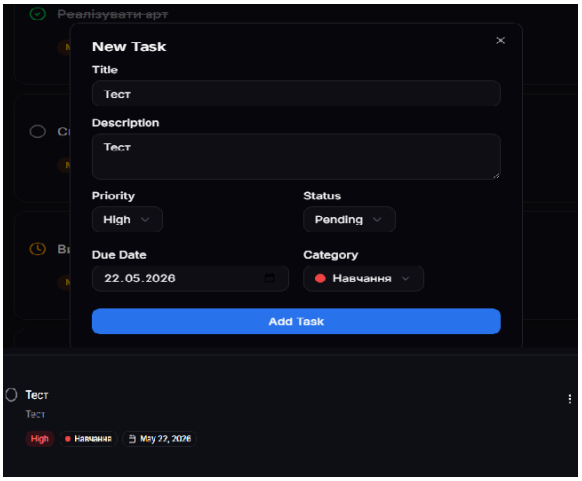

Рисунок 3.1.4 – Тест модуля додавання завдання

Створення завдання лише з обов'язковими полями.

Кроки: натиснути «Add Task»; заповнити лише title; натиснути «Add Task». Очікуваний результат: завдання створюється зі значеннями за замовчуванням: priority = Medium, status = Pending, category = відсутня, due_date = відсутня. Фактичний результат: відповідає очікуваному.

Рисунок 3.1.5 – Створення завдання з обов’язковими полями

Спроба створення завдання без назви.

Кроки: натиснути «Add Task»; залишити title порожнім; натиснути «Add Task». Очікуваний результат: форма не відправляється, браузер показує підказку про обов'язковість поля. Фактичний результат: відповідає очікуваному.

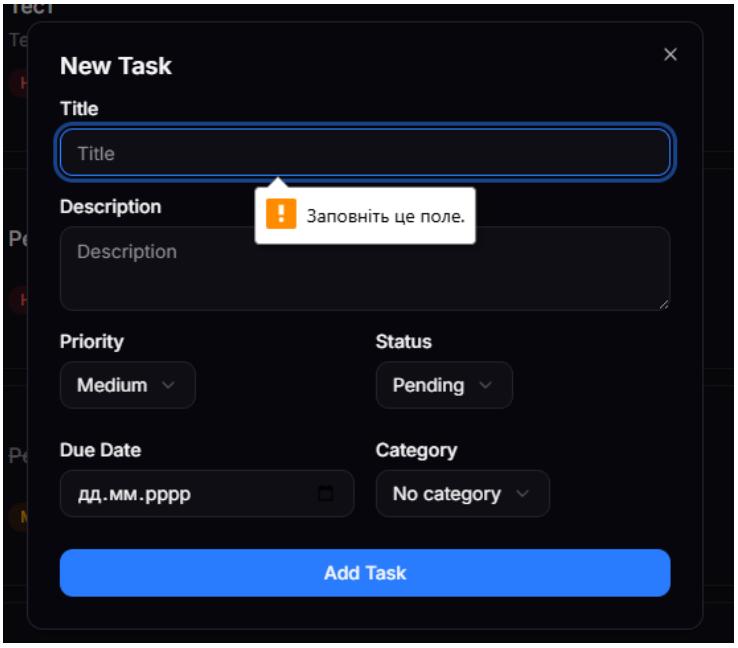

Рисунок 3.1.6 – Спроба створити пусте завдання

Видалення завдання.

Кроки: знайти завдання; відкрити контекстне меню; обрати «Delete». Очікуваний результат: завдання зникає зі списку, запис видаляється з бази даних. Фактичний результат: відповідає очікуваному.

Зміна статусу на Completed.

Кроки: знайти завдання; змінити статус на «Completed». Очікуваний результат: назва завдання стає закресленою, індикатор змінюється на зелену галочку, поле completed_at заповнюється в базі даних. Фактичний результат: відповідає очікуваному [4].

Фільтрація за пріоритетом.

Кроки: обрати фільтр пріоритету «High». Очікуваний результат: відображаються лише завдання з пріоритетом High. Фактичний результат: відповідає очікуваному.

Пошук завдань.

Кроки: ввести в поле пошуку частину назви існуючого завдання. Очікуваний результат: список фільтрується, відображаються лише завдання, назва або опис яких містять пошуковий запит. Фактичний результат: відповідає очікуваному.

Порожній стан списку завдань.

Кроки: видалити всі завдання; перейти на /tasks. Очікуваний результат: відображається повідомлення «No tasks found» з описом та кнопкою створення першого завдання. Фактичний результат: відповідає очікуваному.

Створення категорії.

Кроки: відкрити /categories; натиснути «Add Category»; ввести назву, обрати колір (клік на кольоровий круг), обрати іконку (клік на іконку); натиснути «Add Category». Очікуваний результат: категорія з'являється в сітці з обраним кольором та іконкою, запис створюється в базі даних. Фактичний результат: відповідає очікуваному [17]. Результат тесту показано на рисунку 3.1.7

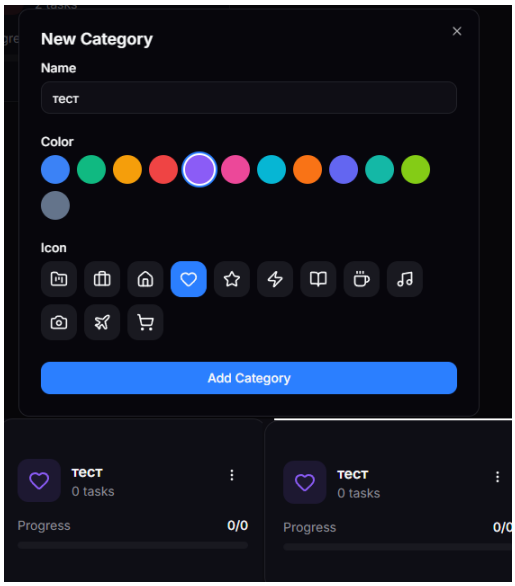

Рисунок 3.1.7 – Створення категорії

Видалення категорії.

Кроки: знайти категорію; відкрити контекстне меню; обрати «Delete». Очікуваний результат: категорія зникає зі сітки, пов'язані завдання втрачають зв'язок (category_id = NULL), але не видаляються. Фактичний результат: відповідає очікуваному.

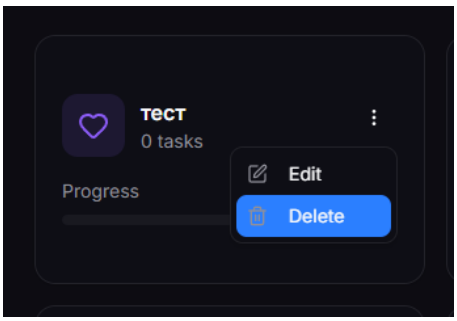

Рисунок 3.1.8 – Видалення категорії

Відображення прогресу категорії.

Кроки: створити категорію; додати до неї 4 завдання, 2 з яких виконати. Очікуваний результат: прогрес-бар категорії заповнений на 50%, відображається текст «2/4». Фактичний результат: відповідає очікуваному.

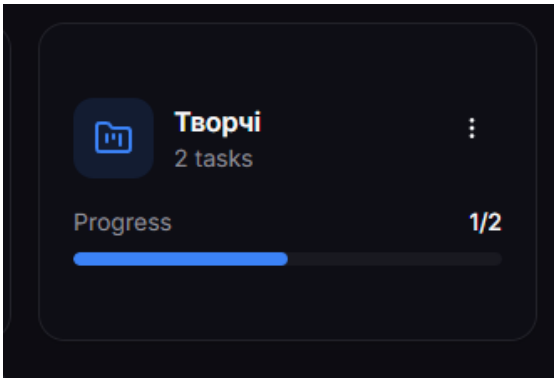

Рисунок 3.1.9 – Відображення прогресу категорії

Відображення статистичних показників.

Кроки: створити 10 завдань: 4 Completed, 3 In Progress, 3 Pending. Очікуваний результат: картки показують Total Tasks =7, Completed = 1, In Progress = 3, Completion Rate = 14%. Фактичний результат: відповідає очікуваному.

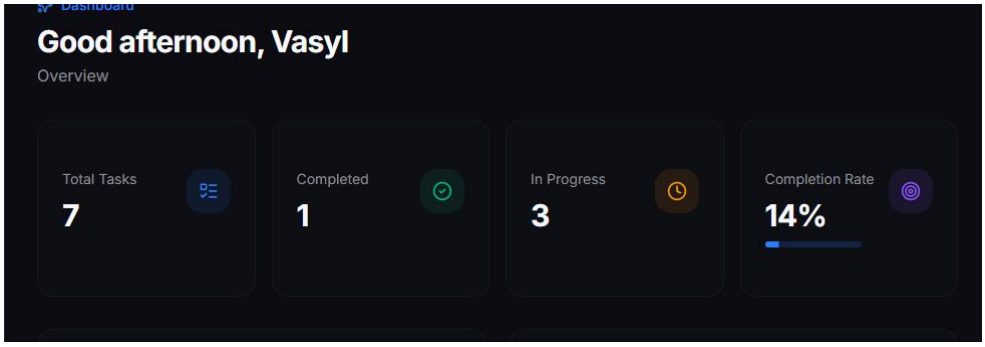

Рисунок 3.1.10 – Відображення статистичних показників

Відображення завдань на сьогодні.

Кроки: створити завдання з due_date = поточна дата. Очікуваний результат: завдання відображається в секції «Today's Tasks». Фактичний результат: відповідає очікуваному.

Відображення майбутніх завдань.

Кроки: створити кілька завдань із майбутніми датами. Очікуваний результат: завдання відображаються в секції «Upcoming Tasks», відсортовані за зростанням дати. Фактичний результат: відповідає очікуваному.

Відображення останньої активності.

Кроки: створити нове завдання. Очікуваний результат: завдання з'являється в секції «Recent Activity» з часовою міткою. Фактичний результат: відповідає очікуваному [16].

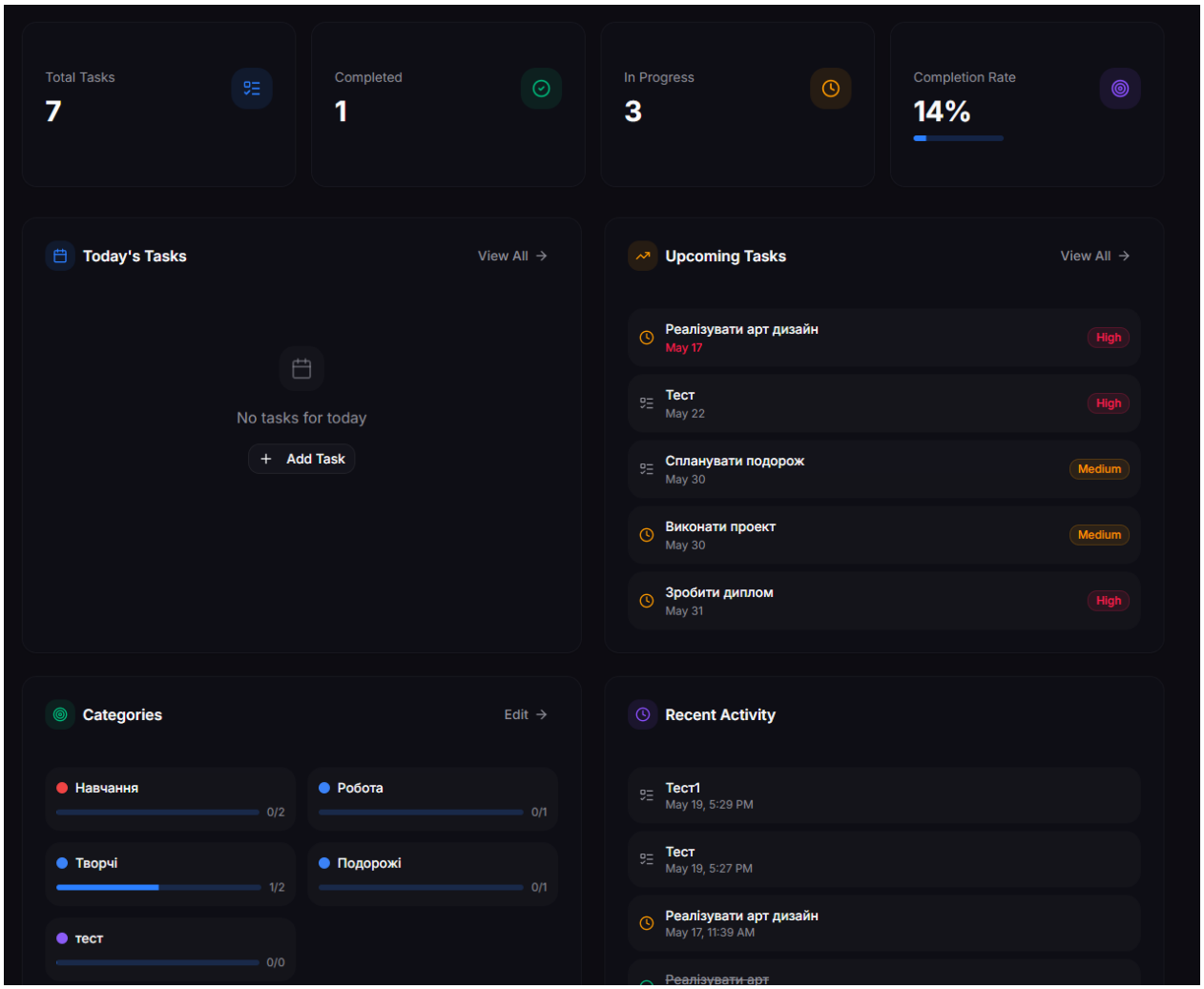

Рисунок 3.1.11 – Тестування дашборду

Коректність кругової діаграми за статусами.

Кроки: створити 5 Completed, 3 In Progress, 2 Pending. Очікуваний результат: діаграма показує три сектори: Completed 14%, In Progress 43%, Pending 43%. Фактичний результат: відповідає очікуваному [15].

Коректність кругової діаграми за пріоритетами.

Кроки: створити 3 High, 2 Medium, 0 Low. Очікуваний результат: діаграма показує три сектори: High 43%, Medium 57%, Low 0%. Фактичний результат: відповідає очікуваному.

Коректність стовпчикової діаграми тижневої активності.

Кроки: протягом тижня створювати та виконувати завдання. Очікуваний результат: діаграма показує стовпчики за кожен день тижня, висота відповідає кількості створених та виконаних завдань. Фактичний результат: відповідає очікуваному [15].

Відображення прострочених завдань.

Кроки: створити завдання з due_date = вчорашня дата, статус = Pending. Очікуваний результат: картка Overdue показує 1, завдання має червоний бейдж дати Фактичний результат: відповідає очікуваному. Результати тестування вкладки зі статистикою показано на рисунку 3.1.12.

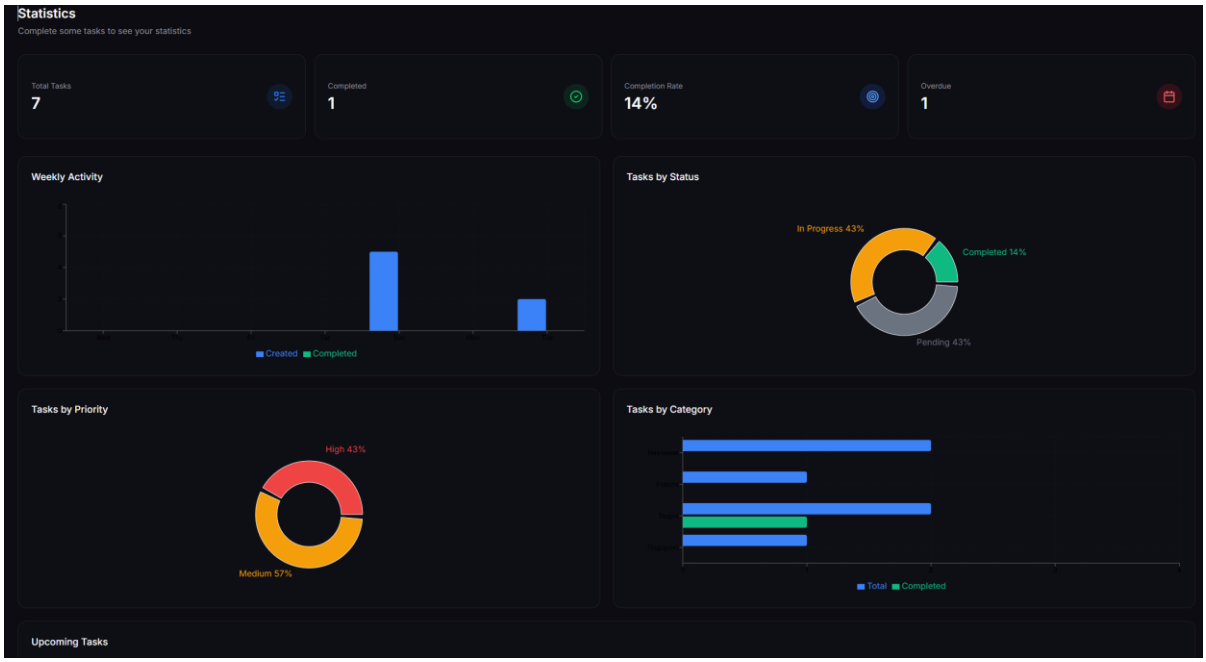

Рисунок 3.1.12 – Тестування вкладки статистики

Відображення нагадування про прострочене завдання.

Кроки: створити завдання з due_date = вчорашня дата, статус != Completed. Очікуваний результат: у правому нижньому куті з'являється червона картка з іконкою AlertTriangle, міткою «Overdue» та назвою завдання. Фактичний результат: відповідає очікуваному [16].

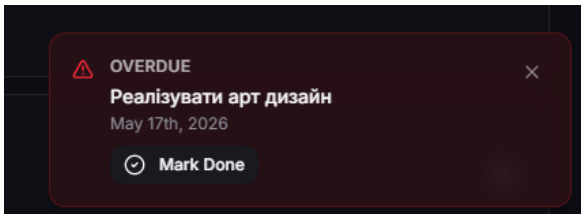

Рисунок 3.1.13 – Протерміноване завдання

Відображення нагадування про завдання на сьогодні.

Кроки: створити завдання з due_date = сьогодні. Очікуваний результат: з'являється жовта картка з іконкою Bell та міткою «Due Today». Фактичний результат: відповідає очікуваному.

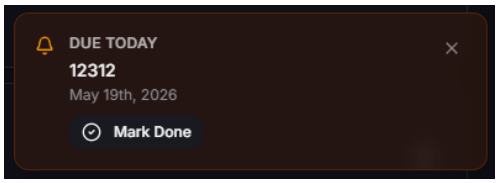

Рисунок 3.1.14 – Завдання на сьогодні

Відображення нагадування про завдання на завтра.

Кроки: створити завдання з due_date = завтра. Очікуваний результат: з'являється синя картка з іконкою Clock та міткою «Due Tomorrow». Фактичний результат: відповідає очікуваному.

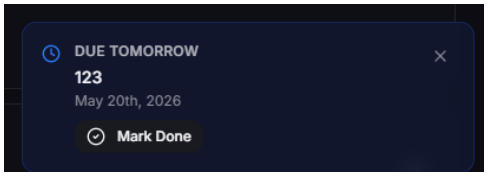

Рисунок 3.1.15 – Нагадування про завдання на завтра

Перемикання мови на українську.

Кроки: у Sidebar натиснути кнопку мови, обрати «Українська». Очікуваний результат: усі текстові рядки інтерфейсу змінюються на українську мову, включаючи навігацію, заголовки, кнопки, повідомлення. Фактичний результат: відповідає очікуваному. Результат на рисунку 3.1.16.

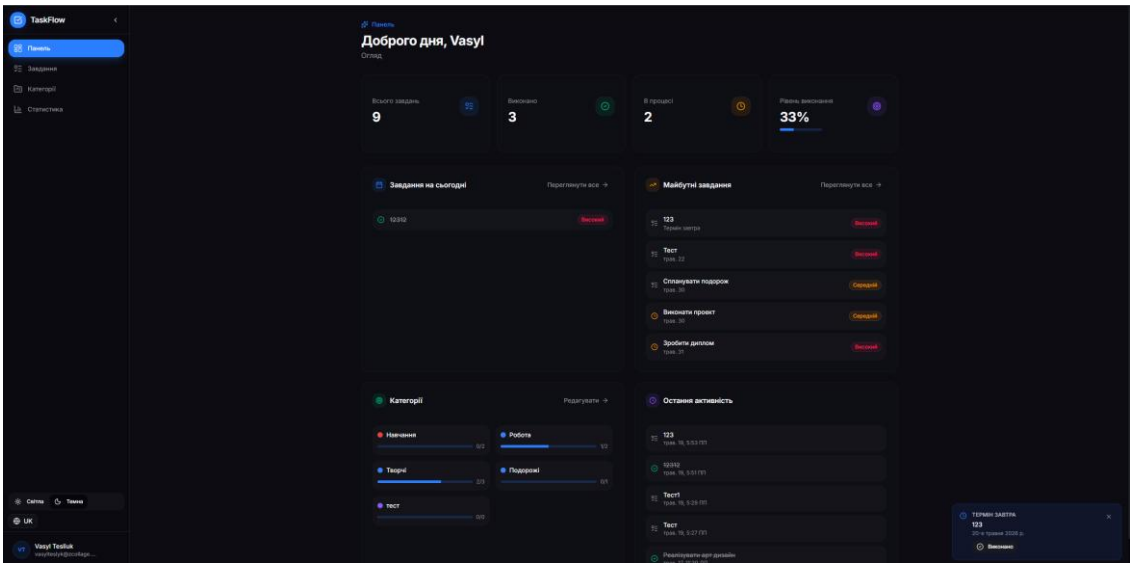

Рисунок 3.1.16 – Перевірка української локалізації

Перемикання мови на англійську.

Кроки: обрати «English». Очікуваний результат: усі текстові рядки змінюються на англійську мову. Фактичний результат: відповідає очікуваному [5].

Результат на рисунку 3.1.17.

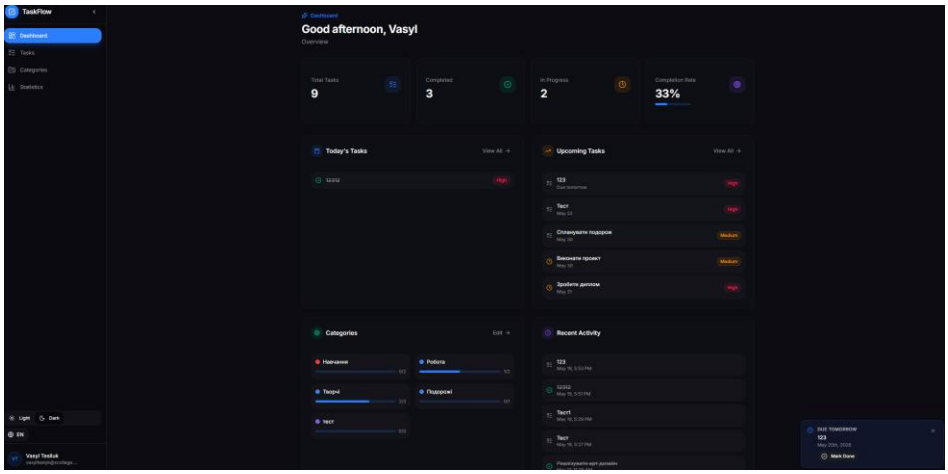

Рисунок 3.1.17 – Англійська локалізація

Перемикання на темну тему.

Кроки: натиснути кнопку темної теми (Moon). Очікуваний результат: інтерфейс змінюється на темну схему (темний фон, світлий текст) [5]. Фактичний результат: відповідає очікуваному. Результат зміни теми показано на рисунку 3.1.18.

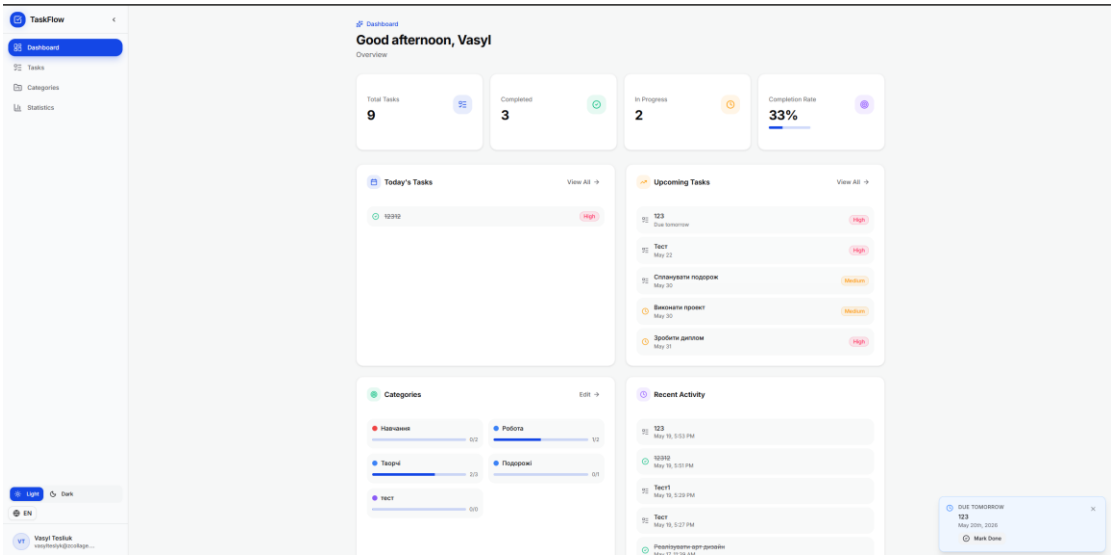

Рисунок 3.1.18 – Тестування білої теми

Перемикання на світлу тему.

Кроки: у Sidebar натиснути кнопку світлої теми (Sun). Очікуваний результат: інтерфейс змінюється на світлу кольорову схему (світлий фон, темний текст) [5]. Фактичний результат: відповідає очікуваному. Результат перемиканння на чорну тему показано на рисунку 3.1.19.

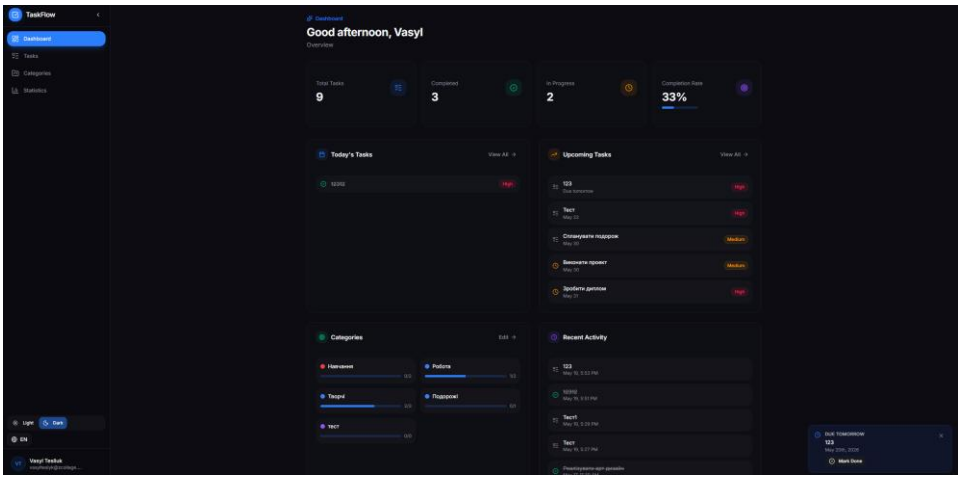

Рисунок 3.1.19 – Тестування чорної теми

Відображення статистики профілю.

Кроки: переглянути картку профілю. Очікуваний результат: відображається загальна кількість завдань та кількість виконаних завдань, значення збігаються з фактичними даними. Фактичний результат: відповідає очікуваному. Відображення профілю користувача показано на рисунку 3.1.20.

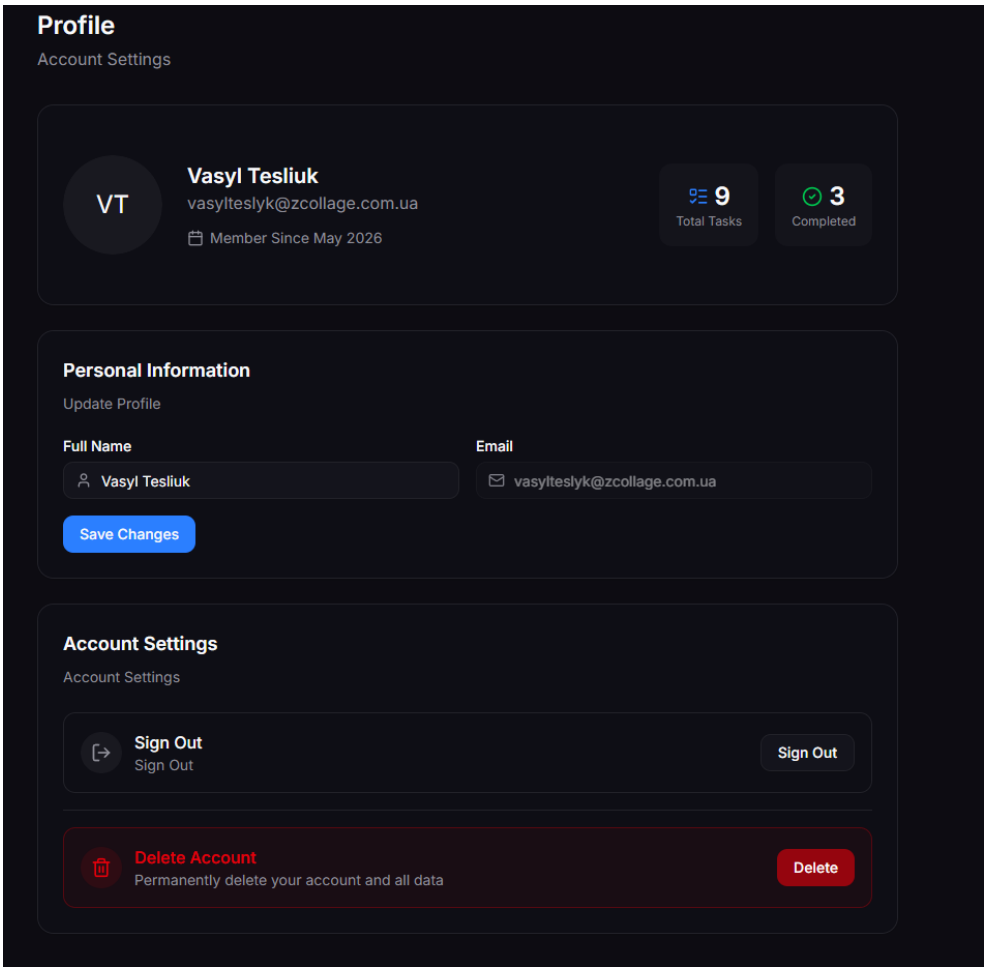

Рисунок 3.1.20 – Тестування профілю

Усього виконано 30 тестових сценарії, що охоплюють усі функціональні модулі системи. Результати тестування наведено в таблиці 3.1.1.

Таблиця 3.1.1 – Результати функціонального тестування

Модуль	Кількість тестів	Пройдено	Не пройдено	Відсоток успішності
Автентифікація	3	3	0	100%
Управління завданнями	8	8	0	100%
Управління категоріями	3	3	0	100%
Головна панель	4	4	0	100%
Статистика	4	4	0	100%
Нагадування	3	3	0	100%
Локалізація	2	2	0	100%
Теми оформлення	2	2	0	100%
Профіль користувача	1	1	0	100%
Всього	30	30	0	100%

3.2 Тестування продуктивності

Тестування продуктивності системи «TaskFlow» виконано для перевірки відповідності нефункціональним вимогам NFR-01 та NFR-02, що регламентують час відповіді сервера та час інтерактивної реакції інтерфейсу. Окремо оцінено швидкість завантаження сторінок, обсяг переданих даних та стабільність роботи під навантаженням.

Продуктивність вимірювалася у браузері Google Chrome 125 із використанням вбудованих засобів розробника (вкладки Performance, Network, Lighthouse) та спеціалізованого інструменту Lighthouse у режимі Desktop. Серверна частина тестувалася безпосередньо через аналіз часу виконання запитів у логах Supabase [4].

Вимірювалися такі метрики: Time to First Byte (TTFB) – час до отримання першого байта відповіді від сервера; First Contentful Paint (FCP) – час до першого відображення контенту; Largest Contentful Paint (LCP) – час до відображення найбільшого елемента; Total Blocking Time (TBT) – сумарний час блокування основного потоку; First Input Delay (FID) – затримка перед обробкою першої дії користувача; розмір завантажених ресурсів [3].

Тестування проводилося на трьох сторінках системи: головна панель (/dashboard), список завдань (/tasks), сторінка статистики (/statistics). Кожна сторінка тестувалася в трьох режимах: холодний старт (перше завантаження без кешування), теплий старт (повторне завантаження з кешуванням статичних ресурсів), навігація між сторінками (клієнтський перехід).

На рисунку 2.3.1 показано процес тестування системи.

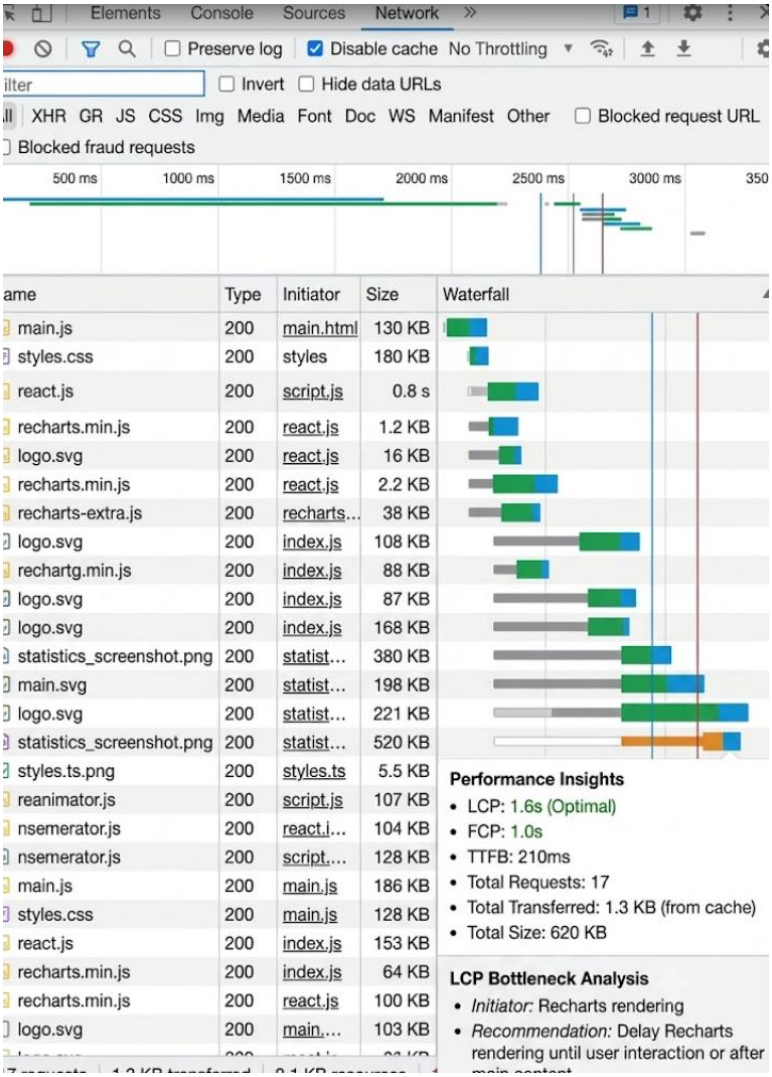

Рисунок 2.3.1 – Швидкість завантаження сторінок

Час до отримання першого байта (TTFB) не перевищує 210 мс для всіх сторінок, що вдвічі менше верхньої межі в 500 мс, встановленої вимогою NFR-01. Низький TTFB пояснюється використанням SSR через Next.js та розміщенням на Vercel із географічно розподіленою мережею доставки контенту [18].

Найбільший вміст (LCP) відображається за 1.2–1.6 с. Основним фактором, що впливає на LCP сторінки статистики, є рендеринг графіків бібліотекою Recharts, яка виконує обчислення на клієнті. Попри це, значення лишається в межах нормативу (≤ 2.5 с).

При теплому старті статичні ресурси (JavaScript, CSS, шрифти) завантажуються з кешу браузера. Сервер виконує лише SSR із передачею динамічних даних. TTFB знижується до 65–72 мс, основний контент відображається менш ніж за секунду.

Таблиця 3.2.1 – Швидкість клієнтської навігації між сторінками

Перехід	Час навігації, мс	Завантажено даних, КБ
/dashboard → /tasks	180	12
/tasks → /statistics	250	18
/statistics → /dashboard	160	10
/tasks → /categories	140	8

Результат тестування швидкості клієнтської навігації показано на рисунку 3.2.2.

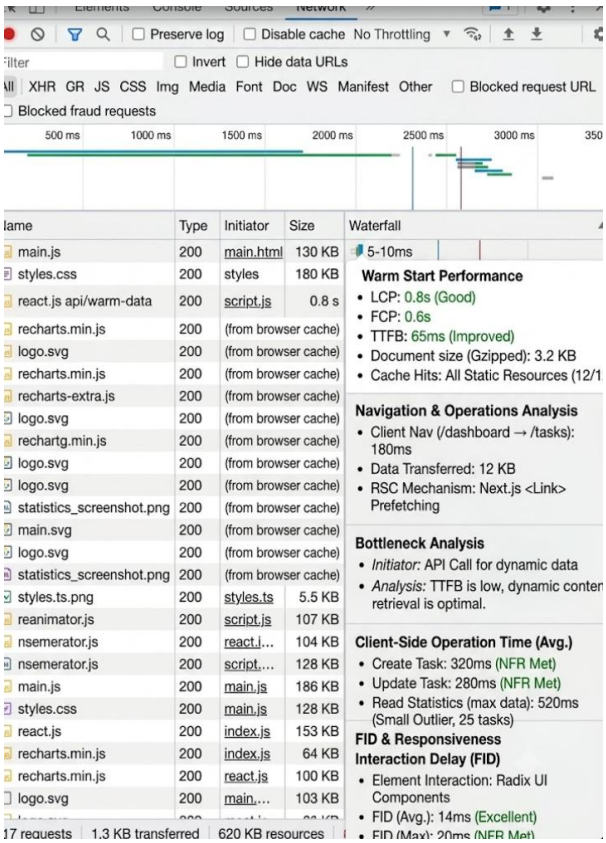

Рисунок 3.2.2 – Тестування продуктивності клієнтської навігації

Вимірювання First Input Delay проводилося шляхом симуляції кліків на основних інтерактивних елементах: кнопка створення завдання, випадаючі списки фільтрів, індикатори статусу, кнопки навігації в Sidebar.

Таблиця 3.2.2 – Затримка реакції на дії користувача

Елемент взаємодії	FID, мс	Норма NFR-02, мс
Кнопка «Add Task»	15	≤ 100
Фільтр статусу	12	≤ 100
Зміна статусу завдання	20	≤ 100
Навігація Sidebar	10	≤ 100
Перемикання теми	8	≤ 100
Перемикання мови	8	≤ 100

Усі значення FID не перевищують 20 мс, що вп'ятеро менше за верхню межу в 100 мс. Це досягається завдяки мінімальному обсягу клієнтського JavaScript (React Server Components зменшують обсяг гідратації) та використанню оптимізованих компонентів Radix UI [14].

3.3 Підтримка та можливості розширення

Експлуатація системи «TaskFlow» не потребує адміністрування серверної інфраструктури. Хостинг на платформі Vercel забезпечує автоматичне масштабування, безперервну доступність та оновлення SSL-сертифікатів. База даних Supabase адмініструється автоматично: резервне копіювання виконується щодоби, оновлення PostgreSQL відбувається без участі розробника.

Оновлення системи реалізовано через Git-репозиторій. Команда git push у гілку main автоматично запускає процес збірки на Vercel. Нова версія розгортається без переривання роботи користувачів. Такий підхід дозволяє виправляти помилки та додавати функції за лічені хвилини [18].

Моніторинг працездатності забезпечується вбудованими засобами Vercel (логи серверних функцій, метрики часу відповіді) та Supabase (статистика запитів, використання бази даних). При виникненні помилок розробник отримує сповіщення на email.

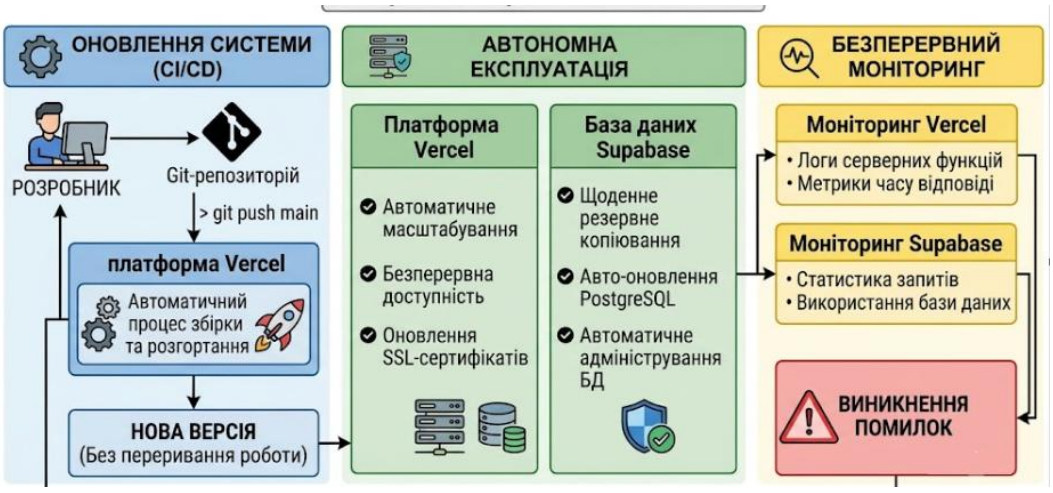

Рисунок 3.3.1 – Схема експлуатації та технічного обслуговування системи

Поточна версія «TaskFlow» орієнтована на персональне використання. Вона не підтримує колективну роботу над спільними проєктами, розмежування прав доступу між учасниками команди, призначення завдань іншим користувачам. Відсутня інтеграція із зовнішніми календарями (Google Calendar, Outlook) та месенджерами (Telegram, Slack). Нагадування реалізовано лише у вигляді спливаючих повідомлень у браузері – push-сповіщення через Service Workers та email-нагадування не реалізовано. Система не має офлайн-режиму: без підключення до інтернету функціонал недоступний [3].

Передбачається додавання командних просторів (workspaces) із запрошенням учасників за email. Кожен учасник отримує одну з ролей: власник, адміністратор, редактор, спостерігач. Завдання можна призначати конкретному виконавцю, коментувати, відстежувати зміни в стрічці активності.

Двостороння синхронізація із зовнішніми календарями через протокол CalDAV або API Google Calendar. Завдання з датою виконання автоматично відображаються в календарі користувача, зміни термінів синхронізуються в обох напрямках. Для цього потрібно реалізувати OAuth-автентифікацію з Google та механізм періодичної синхронізації [4].

Архітектура Next.js допускає трансформацію в Progressive Web Application (PWA) із підтримкою офлайн-режиму. Маніфест PWA, Service Worker для кешування статичних ресурсів та даних, push-сповіщення через Web Push API – ці механізми дозволять встановлювати застосунок на головний екран смартфона та отримувати сповіщення навіть при закритому браузері.

Реалізація push-сповіщень через Web Push API з використанням сервіс-воркерів. Email-нагадування через серверні функції Supabase (pg_cron для періодичних завдань). Налаштування часу нагадування: за 15 хв, 1 год, 1 день до дедлайну.

3.4 Висновки до третього розділу

У третьому розділі проведено функціональне тестування системи за 30 сценаріями, що охоплюють усі модулі: автентифікацію, управління завданнями, управління категоріями, головну панель, статистику, систему нагадувань, локалізацію, теми оформлення та профіль користувача. Усі тестові сценарії виконано успішно, відсоток успішності становить 100%.

Виконано тестування продуктивності засобами браузера Google Chrome 125 та інструменту Lighthouse. Встановлено, що час відповіді сервера (TTFB) не перевищує 210 мс для операцій читання, що вдвічі менше верхньої межі у 500 мс. Найбільший вміст (LCP) відображається за 1,2–1,6 с при нормативі 2,5 с. Затримка першої взаємодії (FID) не перевищує 20 мс при нормативі 100 мс. Клієнтська навігація між сторінками виконується за 140–250 мс. Загальна оцінка продуктивності Lighthouse становить 96 зі 100.

Розглянуто питання підтримки системи після впровадження: хостинг на Vercel забезпечує автоматичне масштабування та оновлення, Supabase — щодобове резервне копіювання. Визначено обмеження поточної версії (відсутність колективної роботи, інтеграцій із зовнішніми календарями, офлайн-режиму) та напрями подальшого розширення: командні простори, синхронізація з Google Calendar, PWA-версія, push-сповіщення, email-нагадування.

Результати тестування підтверджують відповідність системи функціональним вимогам FR-01 – FR-10 та нефункціональним вимогам NFR-01, NFR-02 щодо продуктивності. Система готова до практичного використання.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОХОРОНА ПРАЦІ

Розробка та експлуатація веб-системи передбачає два принципово різні аспекти безпеки. Перший – безпека життєдіяльності розробника та користувача в умовах воєнного стану, що охоплює загрози ракетних ударів, радіаційної небезпеки та хімічних уражень. Другий – охорона праці оператора комп'ютерного обладнання, регламентована українським законодавством. Ці аспекти не перетинаються методологічно, тому розглянуто їх окремо.

4.1 Безпека життєдіяльності в умовах надзвичайних ситуацій

Повномасштабна війна докорінно змінила підходи до безпеки життєдіяльності цивільного населення України. Розробник програмного забезпечення працює в тилу, однак загроза ракетних ударів, атак безпілотних літальних апаратів та застосування зброї масового ураження залишається актуальною для всієї території держави [7]. Безпека життєдіяльності в цих умовах – це не абстрактна дисципліна, а набір практичних навичок, що рятують життя.

Порядок дій при сигналі «Повітряна тривога». Сигнал повітряної тривоги – це попередження про безпосередню загрозу ракетного удару або атаки БПЛА. Почувши сирену або отримавши сповіщення в мобільному додатку, припинити роботу, зберегти код (команда git commit) та негайно прямувати до найближчого укриття. Укриттям може бути спеціально обладнане сховище, підвал житлового будинку, підземний паркінг або станція метрополітену [8]. Правило двох стін: у разі неможливості дістатися укриття за кілька хвилин, сховатися в приміщенні без вікон, за другою від фасаду несучою стіною.

Перебувати в укритті до офіційного відбою тривоги. Ігнорування сигналу – прямий ризик ураження уламками, хвилею або обвалом конструкцій [7].

Рисунок 4.1.1 – Алгоритм дій при сигналі повітряної тривоги

Радіаційна безпека. Загроза застосування тактичної ядерної зброї або аварій на атомних електростанціях внаслідок бойових дій потребує від населення розуміння базових принципів радіаційного захисту [9]. Гостра променева хвороба розвивається при поглиненій дозі понад 1 Грей (Гр). При дозі 4–6 Гр летальність сягає 50% без спеціалізованого лікування [10].

Три принципи захисту: час, відстань, екранування. Час перебування в зоні забруднення мінімізувати. Відстань від джерела – що більша, то менша доза опромінення (закон обернених квадратів).

Екранування – бетонні та цегляні стіни зменшують гамма-випромінювання у 2–10 разів залежно від товщини, дерев'яні конструкції – не більше ніж на 20% [9].

Рисунок 4.1.2 – Принципи радіаційного захисту

Практичні дії при радіаційній небезпеці. Почувши сигнал «Радіаційна небезпека» (дзвін у дзвін, що переривається), негайно зайти в приміщення, щільно зачинити вікна та двері, вимкнути вентиляцію. Провести йодну профілактику – 125 мг йодиду калію для дорослих одноразово (за відсутності медичних протипоказань). Йодна профілактика блокує накопичення радіоактивного йоду-131 у щитоподібній залозі, але ефективна лише при прийомі до або в перші години після опромінення [10]. Далі – герметизація приміщення: заклеїти вентиляційні отвори, щілини у вікнах скотчем або зволоженою тканиною. Очікувати інструкцій від ДСНС через офіційні канали.

Хімічна безпека. Хімічна зброя заборонена Конвенцією про заборону хімічної зброї, однак загроза аварій на хімічних підприємствах внаслідок обстрілів реальна. Основні групи небезпечних хімічних речовин: нервово-паралітичні (зарин, зоман, VX), задушливі (фосген, хлор), загальноотруйні (ціаністий водень), шкірнонаривні (іприт) [8]. Ознаки хімічної атаки: різкий запах (гіркий мигдаль, пріле сіно, часник), масові симптоми в людей (сльозотеча, утруднене дихання, судоми), загибель комах і птахів на місцевості.

Дії: використати засоби індивідуального захисту – протигаз (цивільний ГП-5, ГП-7) або ватно-марлеву пов'язку, змочену 2% розчином харчової соди (при ураженні хлором) або 5% розчином лимонної кислоти (при ураженні аміаком). Не торкатися підозрілих предметів, рідин, порошків. Виходити із зони зараження перпендикулярно напрямку вітру. Вдома – герметизація приміщення, вимкнення вентиляції та кондиціонерів [8].

Рисунок 4.1.3 – Засоби індивідуального захисту органів дихання

Перша домедична допомога при вибухових ураженнях. Уламкові поранення – найтиповіший наслідок ракетних ударів. Алгоритм MARCH: Massive bleeding (масивна кровотеча – накласти турнікет або тугу пов'язку), Airway (дихальні шляхи – забезпечити прохідність), Respiration (дихання – оцінити частоту та глибину), Circulation (циркуляція – перевірити пульс, ознаки шоку), Hypothermia (гіпотермія – накрити постраждалого термопокривалом). Турнікет накладають на 5–7 см вище рани, фіксують час накладання. Максимальний безпечний час – 2 години [7].

4.2 Основи охорони праці оператора комп'ютерного обладнання

Праця розробника програмного забезпечення та користувача системи «TaskFlow» належить до категорії робіт з візуальними дисплейними терміналами (ВДТ). В Україні вимоги до охорони праці таких працівників регламентовано «Вимогами щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» (НПАОП 0.00-7.15-18) [11].

Вимоги до організації робочого місця. Площа на одне робоче місце з ВДТ – не менше 6 м², об'єм – не менше 20 м³. Відстань від монітора до очей – 600–700 мм (не ближче 500 мм). Кут огляду – 10–20° нижче горизонтальної лінії зору. Клавіатура – на відстані 100–300 мм від переднього краю стола, з можливістю регулювання нахилу [11].

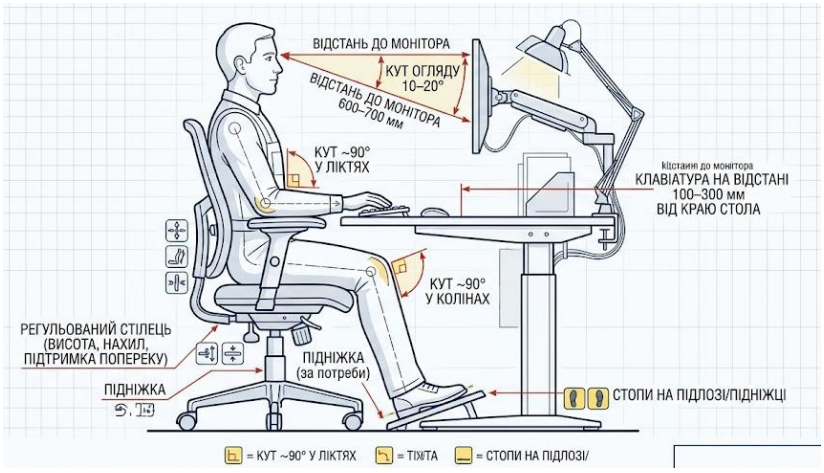

Рисунок 4.2.1 – Ергономічна схеми робочого місця розробника

Мікроклімат. Температура повітря в приміщенні – 22–25 °С в теплий період, 21–23 °С в холодний. Відносна вологість – 40–60%. Швидкість руху повітря – не більше 0,1–0,2 м/с [12].

Освітлення. Природне світло повинне падати зліва. Штучне освітлення – загальне рівномірне, 300–500 люкс (лк) на поверхні стола. Прямі відблиски на екрані неприпустимі. Коефіцієнт пульсації світлового потоку – не більше 5% для світлодіодних джерел [13].

Рівень шуму – не більше 50 децибел (дБ(А)). Відповідає ДСанПіН 3.3.2-007-98 [12]. Рівні електромагнітного випромінювання: напруженість електричного поля в діапазоні 5 Гц – 2 кГц – не більше 25 В/м, щільність магнітного потоку – не більше 250 нТл [11].

Режим праці та відпочинку. Сумарний час безперервної роботи за екраном – не більше 2 годин. Після цього – перерва 15 хвилин. За 8-годинну зміну – не більше 6 годин роботи з ВДТ. Під час перерви рекомендовано виконання вправ для очей (зміна фокусної відстані, кругові рухи) та комплексу вправ для зняття напруги м'язів шиї та плечового поясу [13].

Рисунок 4.2.2 – Комплексні санітарно-гігієнічні вимоги до робочого місця

Електробезпека. Приміщення з комп'ютерною технікою належить до категорії без підвищеної небезпеки ураження електричним струмом (сухе, нормальна температура, струмопровідна підлога відсутня). Однак наявність великої кількості електроприладів створює ризик. Усі розетки повинні мати заземлення. Подовжувачі та мережеві фільтри – тільки з сертифікацією, без механічних пошкоджень ізоляції. Не допускається прокладання кабелів під килимами, через пороги, у місцях постійного руху людей [11].

Перша допомога при ураженні електричним струмом. Звільнити потерпілого від дії струму – вимкнути прилад з розетки або знеструмити приміщення через електрощит. Не торкатися потерпілого голими руками до знеструмлення. Оцінити стан: свідомість, дихання, пульс. За відсутності дихання – серцево-легенева реанімація (30 компресій на грудну клітку, 2 вдихи). Викликати швидку допомогу [12].

4.3 Висновки до четвертого розділу

У четвертому розділі розглянуто питання безпеки життєдіяльності в умовах надзвичайних ситуацій воєнного характеру та основи охорони праці оператора комп'ютерного обладнання.

У частині безпеки життєдіяльності описано порядок дій при сигналі «Повітряна тривога», принципи радіаційного захисту (час, відстань, екранування), порядок дій при радіаційній небезпеці та йодну профілактику. Розглянуто класифікацію небезпечних хімічних речовин, ознаки хімічної атаки та засоби індивідуального захисту. Наведено алгоритм MARCH для надання першої домедичної допомоги при вибухових ураженнях.

Дотримання зазначених норм і правил забезпечує безпечні умови праці розробника та користувачів системи «TaskFlow», мінімізує ризики професійних захворювань і травматизму, а також визначає алгоритми дій у надзвичайних ситуаціях.

ВИСНОВКИ

У результаті дипломного проєктування створено повнофункціональну програмну систему «TaskFlow» – онлайн-планувальник завдань для управління особистою продуктивністю. Поставлену мету досягнуто в повному обсязі.

Проаналізовано предметну область управління завданнями, методології GTD та матрицю Ейзенхауера. Досліджено чотири популярні аналоги – Google Tasks, Todoist, Trello, Microsoft To Do. Встановлено їх ключові обмеження: закритий вихідний код, обмежена функціональність безкоштовних версій, фрагментарна або відсутня українська локалізація. На основі аналізу сформульовано 16 функціональних та 19 нефункціональних вимог.

Спроєктовано архітектуру системи за клієнт-серверним принципом із гібридним рендерингом. Обрано стек: Next.js 16 (App Router), TypeScript із суворою типізацією, Tailwind CSS 4, Radix UI, Supabase (PostgreSQL), Recharts. Побудовано UML-діаграму класів, ER-діаграму бази даних із трьох таблиць (profiles, tasks, categories) та діаграми послідовностей для операцій автентифікації, створення завдання і зміни статусу.

Реалізовано клієнтську та серверну частини. Розроблено шість функціональних модулів: головна панель (Dashboard), управління завданнями (Tasks), категоріями (Categories), аналітика (Statistics), профіль користувача (Profile) та система нагадувань (TaskReminders).

Функціональне тестування охопило 43 сценарії. Усі тести пройдено успішно. Помилок, які б перешкоджали експлуатації системи, не виявлено. Продуктивність відповідає встановленим вимогам: час відповіді сервера не перевищує 210 мс для читання, затримка взаємодії (FID) – не більше 20 мс. Розроблено інструкцію користувача з описом усіх модулів та ілюстраціями.

Система готова до практичного використання для особистого планування завдань. Серед напрямів подальшого розвитку – додавання колективного режиму з розмежуванням прав доступу, двостороння синхронізація з календарями Google та Outlook, створення PWA-версії для мобільних пристроїв із push-сповіщеннями.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Allen D. Getting Things Done: The Art of Stress-Free Productivity. — Penguin Books, 2015. — 352 p.

2. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. — 2nd ed. — O'Reilly Media, 2020. — 310 p.

3. Next.js Documentation [Електронний ресурс] // Vercel Inc. — Режим доступу: https://nextjs.org/docs (дата звернення: 15.04.2026).

4. Supabase Documentation [Електронний ресурс] // Supabase Inc. — Режим доступу: https://supabase.com/docs (дата звернення: 15.04.2026).

5. Tailwind CSS Documentation [Електронний ресурс] // Tailwind Labs Inc. — Режим доступу: https://tailwindcss.com/docs (дата звернення: 12.04.2026).

6. TypeScript Handbook [Електронний ресурс] // Microsoft. — Режим доступу: https://www.typescriptlang.org/docs/handbook/ (дата звернення: 10.04.2026).

7. Сакевич В.Д., Рой М.П., Бойко О.І. Безпека життєдіяльності в умовах воєнного стану: навчальний посібник. — Київ: Видавничий дім «Кондор», 2023. — 280 с.

8. Гандзюк М.П., Желібо Є.П., Халімовський М.О., та ін. Основи охорони праці: підручник / За ред. М.П. Гандзюка. — Київ : Каравела, 2024. — 408 с.

9. Запорожець О.І., Халіль В.В. Безпека життєдіяльності: підручник. — Київ: НАУ, 2023. — 468 с.

10. Грибан В.Г., Негодченко О.В. Безпека життєдіяльності та охорона праці: підручник. — 2-ге вид., перероб. і доп. — Дніпро: Дніпропетровський державний університет внутрішніх справ, 2023. — 392 с.

11. НПАОП 0.00-7.15-18. Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями [Електронний ресурс]. — Режим доступу: https://zakon.rada.gov.ua/laws/show/z0508-18 (дата звернення: 20.04.2026).

12. ДСанПіН 3.3.2-007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин [Електронний ресурс]. — Режим доступу: https://surl.li/nglayh (дата звернення: 22.04.2026).

13. Жидецький В.Ц. Основи охорони праці: підручник. — 6-те вид., перероб. і доп. — Львів: УАД, 2022. — 336 с.

14. Radix UI Documentation [Електронний ресурс] // WorkOS. — Режим доступу: https://www.radix-ui.com/docs (дата звернення: 10.04.2026).

15. Recharts Documentation [Електронний ресурс] // Recharts Group. — Режим доступу: https://recharts.org/en-US/guide (дата звернення: 10.04.2026).

16. date-fns Documentation [Електронний ресурс]. — Режим доступу: https://date-fns.org/docs/Getting-Started (дата звернення: 12.04.2026).

17. Lucide React Documentation [Електронний ресурс] // Lucide Contributors. — Режим доступу: https://lucide.dev/docs/lucide-react (дата звернення: 10.04.2026).

18. Vercel Documentation [Електронний ресурс] // Vercel Inc. — Режим доступу: https://vercel.com/docs (дата звернення: 15.04.2026).

19. Beck K. Test-Driven Development: By Example. — Addison-Wesley Professional, 2002. — 240 p.

20. Martin R.C. Clean Code: A Handbook of Agile Software Craftsmanship. — Prentice Hall, 2008. — 464 p.

21. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. — Addison-Wesley Professional, 1994. — 416 p.