БАКАЛАВРСЬКА РОБОТА

КРБ.ІСТ-15.00.00.000 ПЗ

Група ІСТ-22-1

Максим РУДИК

2026

Міністерство освіти і науки України

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інформаційно-телекомунікаційних технологій та систем

Рудик Максим Сергійович

(прізвище, ім'я, по батькові)

УДК 004.415.2:004.8

(індекс)

БАКАЛАВРСЬКА РОБОТА

Розроблення інформаційної системи управління завданнями з інтелектуальним модулем генерації та аналізу

(назва роботи)

Інформаційні системи та технології

(назва освітньої програми)

126 − Інформаційні системи та технології

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня Рудик М. С.

(підпис, ініціали та прізвище здобувача)

Науковий керівник к.т.н., доцент Штаєр Л. О.

(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
В. о. завідувача кафедри

Заміховська О. Л.

(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу
(повне найменування закладу вищої освіти)

Факультет інформаційних технологій
Кафедра інформаційно-телекомунікаційних технологій та систем
Освітній рівень бакалавр
Спеціальність 126 − Інформаційні системи та технології
(шифр і назва)

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри ІТТС
Заміховська О. Л.
«____» _____2026 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Рудику Максиму Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення інформаційної системи управління завданнями з інтелектуальним модулем генерації та аналізу
керівник роботи, доц. каф. ІТТС Штаєр Л. О.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від " 07 " *квітня* 2026 року № 163/7
2. Строк подання студентом роботи
3. Вихідні дані до роботи Кросплатформна інформаційна система на базі React Native із серверною частиною Ruby on Rails 8 та AI-мікросервісом на Python FastAPI; локальне зберігання нотаток у форматі Markdown; наскрізне шифрування резервних копій на стороні клієнта; локальне виконання мовних моделей без залежності від зовнішніх AI API; JWT-автентифікація з відкликанням токенів; розмежування доступу через Pundit; підтримка підписки через Stripe.
4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)
аналіз предметної області та існуючих систем управління завданнями; проєктування інформаційної системи; програмна реалізація системи та інтелектуальний модуль.
5. Перелік презентаційного матеріалу (з точним зазначенням обов'язкових презентацій)
варіанти використання інформаційної системи управління завданнями; діаграма компонентів інформаційної системи управління завданнями; діаграма використання інформаційної системи обліку завдань; програмні вікна клієнтського додатку інформаційної системи управління завданнями; програмні вікна клієнтського додатку інформаційної системи управління завданнями
6. Дата видачі завдання: 07.04.2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Термін виконання етапів	Примітка
1.	Аналіз предметної області та огляд існуючих систем управління завданнями	01.04.2026 - 08.04.2026 р.	Виконано
2.	Розробка технічного завдання, діаграм прецедентів та Use Case	01.04.2026 - 08.04.2026 р.	Виконано
3.	Проєктування бази даних (PostgreSQL) та ER-діаграми	08.04.2026 - 15.04.2026 р.	Виконано
4.	Програмна реалізація серверної частини (Ruby on Rails API) та JWT-автентифікації	15.04.2026 - 22.04.2026 р.	Виконано
5.	Розробка клієнтського інтерфейсу на базі React Native (файловий провідник, редактор, Kanban)	22.04.2026 - 30.04.2026 р.	Виконано
6.	Реалізація AI-мікросервісу на Python FastAPI з локальними моделями Hugging Face	30.04.2026 - 07.05.2026 р.	Виконано
7.	Реалізація модуля шифрування резервних копій (AES-256-CBC) та системи підписок Stripe	07.05.2026 - 14.05.2026 р.	Виконано
8.	Оформлення пояснювальної записки та графічного матеріалу	21.05.2026 - 28.05.2026 р.	Виконано
9.	Подання роботи на рецензування та підготовка до захисту	28.05.2026 - 04.06.2026 р.	

Студент _____ *Рудик М. С.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ Штаєр Л. О.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Розрахунково-пояснювальна записка: 64 сторінок, 27 малюнків, 3 таблиці, 16 посилань, 2 додатки.

Об'єкт дослідження – процеси управління завданнями та нотатками в сучасних інформаційних системах з підтримкою інтелектуальної обробки даних.

Мета роботи – розроблення інформаційної системи управління завданнями з інтелектуальним модулем генерації та аналізу, що забезпечує ефективну організацію робочих процесів, офлайн-орієнтоване зберігання даних та автоматизовану обробку текстового контенту на основі локально розгорнутих моделей машинного навчання.

У першій частині роботи виконано системний аналіз предметної області, проведено порівняння існуючих програмних рішень та обґрунтовано доцільність розроблення власної системи.

У другій частині здійснено проєктування архітектури системи. Обґрунтовано вибір клієнт-серверної архітектури з окремим AI-мікросервісом. Розроблено ER-діаграму бази даних PostgreSQL, діаграму компонентів та варіанти використання системи. Спроєктовано інтерфейс користувача з підтримкою файлового провідника, Markdown-редактора та Kanban-дошки.

У третій частині описано програмну реалізацію системи. Реалізовано серверну частину на Ruby on Rails 8 у режимі API-only. Розроблено клієнтський застосунок на React Native з підтримкою платформ iOS, Android та Windows у межах єдиної кодової бази. Реалізовано AI-мікросервіс з локальним виконанням моделей Hugging Face (DistilBART для узагальнення тексту, GPT-2 для генерації) без звернення до зовнішніх API. Впроваджено модуль наскрізного шифрування резервних копій на стороні клієнта для генерації ключа.

КЛЮЧОВІ СЛОВА: УПРАВЛІННЯ ЗАВДАННЯМИ, МАШИННЕ НАВЧАННЯ, ВЕЛИКІ МОВНІ МОДЕЛІ, REACT NATIVE, RUBY ON RAILS, FASTAPI, ОФЛАЙН-ОРІЄНТОВАНА АРХІТЕКТУРА, НАСКРІЗНЕ ШИФРУВАННЯ, КРОСПЛАТФОРМНИЙ ЗАСТОСУНОК, ІНТЕЛЕКТУАЛЬНИЙ МОДУЛЬ.

ABSTRACT

Calculation and explanatory note: 64 pages, 27 figures, 3 tables, 16 references, 2 appendices.

Object of research – the processes of task and note management in modern information systems with support for intelligent data processing.

Purpose of the work – development of a task management information system with an intelligent generation and analysis module that ensures effective organization of workflows, offline-oriented data storage, and automated processing of text content based on locally deployed machine learning models.

In the first part of the work, a systematic analysis of the subject area is performed, existing software solutions are compared, and the feasibility of developing a proprietary system is justified.

In the second part, the system architecture is designed. The choice of a client-server architecture with a dedicated AI microservice is substantiated. An ER diagram of the PostgreSQL database, a component diagram, and use cases are developed. The user interface is designed with support for a file explorer, Markdown editor, and Kanban board.

In the third part, the software implementation of the system is described. The server side is implemented on Ruby on Rails 8 in API-only mode. The client application is developed on React Native with support for iOS, Android, and Windows platforms within a single codebase. An AI microservice is implemented with local execution of Hugging Face models (DistilBART for text summarization, GPT-2 for generation) without calls to external APIs. A client-side end-to-end backup encryption module is implemented for key derivation.

KEYWORDS: TASK MANAGEMENT, MACHINE LEARNING, LARGE LANGUAGE MODELS, REACT NATIVE, RUBY ON RAILS, FASTAPI, OFFLINE-FIRST ARCHITECTURE, END-TO-END ENCRYPTION, CROSS-PLATFORM APPLICATION, INTELLIGENT MODULE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД – База даних

ШІ – Штучний інтелект

UI – User Interface (Інтерфейс користувача)

UX – User Experience (Досвід користувача)

API – Application Programming Interface (Інтерфейс програмування застосунків)

REST API – Representational State Transfer Application Programming Interface

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

JWT – JSON Web Token

MVC – Model–View–Controller (Модель–Вигляд–Контролер)

CRUD – Create, Read, Update, Delete

IV – Initialization Vector (Вектор ініціалізації)

CPU – Central Processing Unit (Центральний процесор)

GPU – Graphics Processing Unit (Графічний процесор)

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

OS – Operating System (Операційна система)

LLM – Large Language Model (Велика мовна модель)

UUID – Universally Unique Identifier (Універсальний унікальний ідентифікатор)

ВСТУП

Актуальність. В умовах зростаючого обсягу інформації та динамічності сучасного робочого середовища, ефективне управління завданнями стає ключовим фактором успіху як для окремих фахівців, так і для організацій. Традиційні методи організації та моніторингу задач часто виявляються недостатніми для обробки складних робочих процесів, які вимагають не тільки структурування, а й інтелектуальної підтримки. Розробка інформаційних систем управління завданнями, доповнених інтелектуальними модулями генерації та аналізу, набуває особливої актуальності. Такі системи можуть не лише автоматизувати рутинні операції, а й надавати користувачам розширені можливості, включаючи інтелектуальне планування, прогнозування ризиків, аналіз продуктивності та автоматичну генерацію контенту, що значно підвищує ефективність та швидкість виконання завдань. Ця тема спонукає до дослідження можливостей інтеграції передових технологій штучного інтелекту в системи управління завданнями з метою підвищення їхньої функціональності та адаптивності до індивідуальних потреб користувачів.

Мета дослідження – напрацювання теоретичних аспектів поставленого питання, а також демонстрація розробки інформаційної системи управління завданнями з інтелектуальним модулем генерації та аналізу, що забезпечуватиме ефективне планування та оптимізацію робочих процесів.

Задля досягнення мети було поставлено наступні завдання:

1 дослідити сучасні підходи до управління завданнями та роль штучного інтелекту в цьому процесі;

2 здійснити огляд існуючих програмних рішень для управління завданнями та інтеграції інтелектуальних модулів;

3 обґрунтувати необхідність розробки інформаційної системи управління завданнями з інтелектуальним модулем генерації та аналізу;

4 розробити архітектуру інформаційної системи, що включає мобільний додаток на базі react native та серверну частину на ruby on rails, визначивши протоколи взаємодії між компонентами (rest api);

5 показати особливості розробки інтелектуального модуля для генерації та аналізу завдань на основі штучного інтелекту;

6 оцінити очікувану ефективність та практичне значення розробленої системи.

Об'єкт дослідження – процес управління завданнями в сучасних інформаційних системах.

Предмет дослідження – розробка інформаційної системи управління завданнями, яка базується на архітектурі з мобільним клієнтом та серверною частиною та інтегрує інтелектуальний модуль на основі технологій штучного інтелекту для генерації та аналізу даних.

Методи дослідження. В роботі було використано методи аналізу літературних джерел для обробки теоретичної інформації, порівняння та систематизація виокремлених даних, методи системного аналізу та проєктування архітектури, програмування мобільних та веб-додатків, а також аналізу та обробки даних з використанням алгоритмів штучного інтелекту.

Практичне значення. Результати дослідження мають велике практичне значення для підвищення продуктивності індивідуальних користувачів та команд. Розроблена інформаційна система сприятиме оптимізації процесів планування, моніторингу та виконання завдань, а також покращить якість прийняття рішень за рахунок інтелектуального аналізу. Дипломна робота може бути використана як посібник з розробки подібних інтелектуальних систем управління завданнями.

Структура дипломної роботи обумовлена метою та поставленими завданнями. Робота складається зі вступу, двох розділів з підрозділами та висновками до кожного, загальних висновків, списку використаних джерел та додатків. У дипломній роботі наведено 3 таблиці, 27 рисунків. Список

використаних джерел містить 16 позицій. До роботи долучено 2 додатка на 16 сторінках.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ДЛЯ ПРОГРАМНОГО ДОДАТКУ УПРАВЛІННЯ ЗАВДАННЯМИ

1.1 Встановлення вимог з програм-аналогів

Сучасні інформаційні системи управління завданнями є критично важливим інструментом для оптимізації робочих процесів та підвищення продуктивності. Проте, з постійним зростанням обсягів інформації та складністю задач, виникає потреба в інтеграції інтелектуальних модулів для автоматичної генерації, аналізу та оптимізації завдань. На основі аналізу аналогів можна виділити ключові вимоги до розробки якісної системи, що враховують сучасні тенденції в галузі розробки програмного забезпечення, штучного інтелекту та потреби цільової аудиторії.

Додаток має мати не тільки привабливий та інтуїтивно зрозумілий дизайн, але й надавати потужний функціонал. Розглянемо аналог Notion на рисунку 1.1.

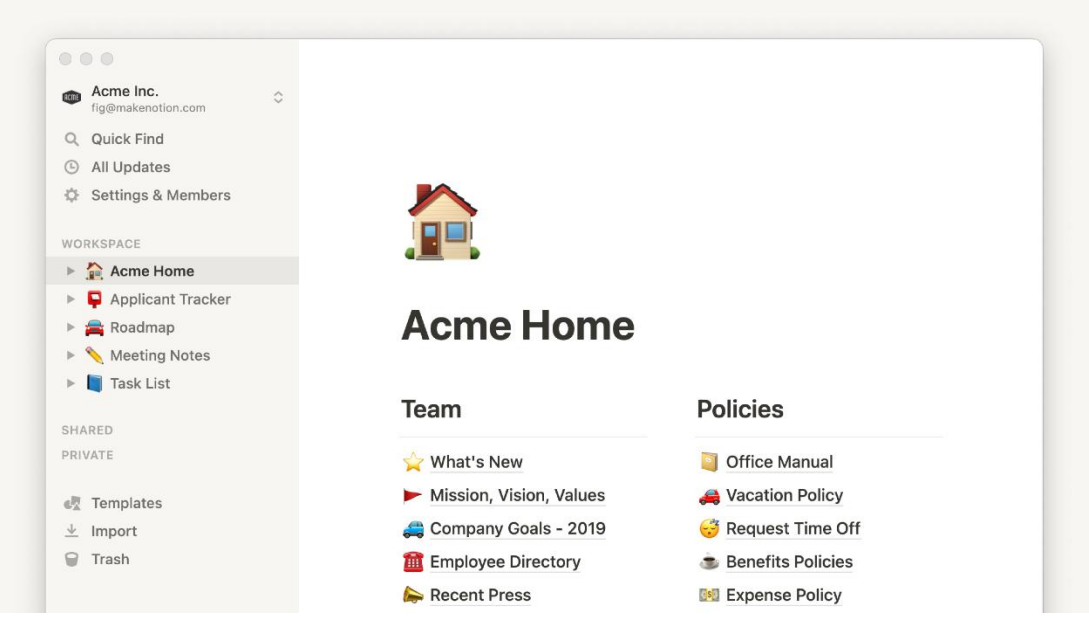

Рисунок 1.1 – Головна сторінка Notion

Notion – це багатофункціональний інструмент для організації роботи, нотаток, управління проєктами та базами даних, який дозволяє користувачам

створювати власні робочі простори, адаптовані до їхніх потреб. Це один із найбільш гнучких та популярних ресурсів для підвищення особистої та командної продуктивності [1].

Розробник: Notion Labs Inc. Архітектура: веб-додаток, десктопні та мобільні клієнти.

Функції:

– система створення нотаток та документів;

– управління проєктами та завданнями (канбан-дошки, списки, календарі);

– бази даних з можливістю налаштування властивостей;

– спільна робота та коментування;

– інтеграції з іншими сервісами;

– система шаблонів.

Переваги:

– висока гнучкість та кастомізація;

– велика спільнота та бібліотека шаблонів;

– кросплатформність.

Недоліки:

– може бути складним для нових користувачів через велику кількість функцій;

– офлайн-режим має обмеження.

Ефективна структура інформаційної системи є критично важливою для зручності користувачів, особливо при інтеграції складних функцій управління завданнями та інтелектуальних модулів. Аналіз аналогів, таких як Jira чи Trello, показує, що чітка ієрархія контенту (проєкти, завдання, підзавдання, теги) полегшує пошук та організацію інформації. Вимоги до структури включають створення основних розділів (наприклад, "Мої завдання", "Проєкти", "Нотатки", "Аналітика"), підтримку тегів для позначення пріоритетів, термінів або категорій завдань, а також інтуїтивну навігацію з

можливістю швидкого переходу між розділами. Приклад структуризації контенту зображено на рисунку 1.2 на прикладі інформаційної системи Asana.

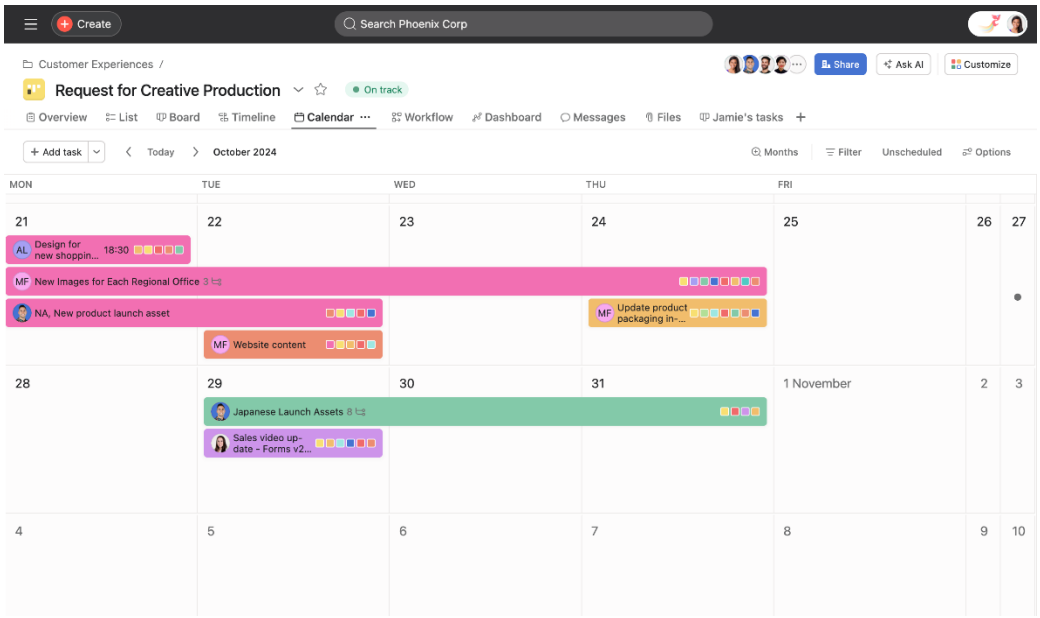

Рисунок 1.2 – Головна сторінка Asana

Asana – це веб-додаток для управління проєктами та командної роботи, який допомагає командам організовувати, відстежувати та управляти своєю роботою. Він забезпечує прозорість, співпрацю та дозволяє ефективно керувати завданнями від початку до кінця [2].

Розробник: Asana, Inc. Архітектура: веб-додаток, десктопні та мобільні клієнти.

Функції:

– створення та призначення завдань;

– відстеження прогресу виконання завдань;

– спільна робота та комунікація в рамках завдань;

– перегляд завдань у різних форматах (список, дошка, календар, діаграма Ганта);

– інтеграція з іншими інструментами (Slack, Google Drive, Microsoft Teams);

– система звітів та аналітики.

Переваги:

– візуалізація робочих процесів;

– ефективна командна співпраця;

– широкий набір функцій для управління проєктами.

Недоліки:

– ціноутворення може бути високим для малих команд;

– деякі функції можуть бути надмірними для простих потреб.

Крім того, критично важливим аспектом є інтеграція інтелектуальних модулів, що використовують технології штучного інтелекту, а також ефективні інструменти для організації та зв'язування інформації. Наприклад, такі інструменти, як Obsidian, демонструють потужні можливості для створення взаємопов'язаних нотаток та баз знань, що може бути надзвичайно корисним для управління завданнями та аналізу даних.

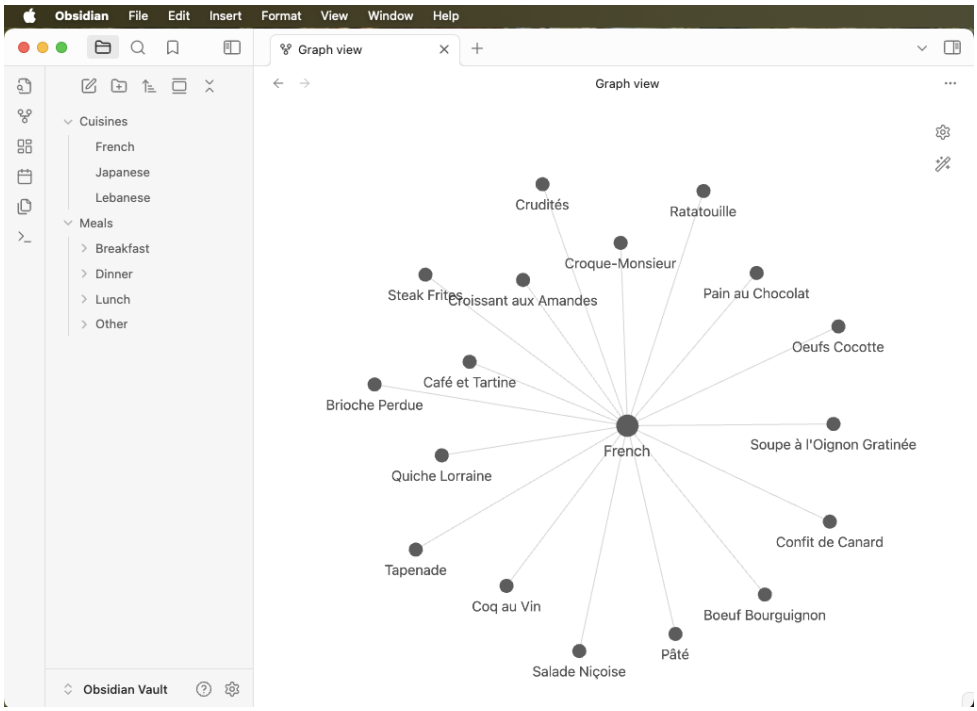

Рисунок 1.3 – Інтерфейс Obsidian

Obsidian – це потужний інструмент для ведення нотаток та побудови бази знань на основі файлів у форматі Markdown. Він дозволяє створювати внутрішні посилання між нотатками, формуючи "графік знань", що допомагає виявляти зв'язки та інтелектуально організовувати інформацію [3].

Розробник: Dynalist Inc. Архітектура: десктопний та мобільний додаток (локальне зберігання файлів).

Функції:

– ведення нотаток у форматі Markdown;

– створення внутрішніх посилань між нотатками (двосторонні посилання);

– графік знань для візуалізації зв'язків;

– система плагінів для розширення функціоналу (управління завданнями, календарі, ШІ-асистенти);

– тегування та пошук нотаток.

Переваги:

– контроль над власними даними (локальне зберігання);

– гнучкість та розширюваність через плагіни;

– потужні можливості для організації знань.

Недоліки:

– потребує певного часу для освоєння;

– не має вбудованих функцій для спільної роботи без сторонніх плагінів.

Також важливо подбати про те, щоб користувачі мали зручний доступ до системи з будь-яких пристроїв, включаючи комп’ютери, ноутбуки, планшети і смартфони. Зростання кількості користувачів, які взаємодіють з інформаційними системами через мобільні пристрої, вимагає адаптивного дизайну. Адаптивний вебдизайн або відгукливий вебдизайн – це дизайн вебсторінок, що забезпечує оптимальне відображення та взаємодію сайту з користувачем незалежно від роздільної здатності та формату пристрою, з якого здійснюється перегляд сторінки. Як зазначають фахівці з проектування інтерфейсів, сучасний мобільний дизайн вимагає глибокого розуміння адаптивних сіток, гнучких зображень та медіа-запитів задля створення консистентного досвіду користувача (UX) на різноманітних екранах [4]. Аналоги, такі як Trello, демонструють успішне використання адаптивних

шаблонів, тоді як застарілі системи, що не оптимізовані для мобільних пристроїв, втрачають аудиторію через незручність (рис 1.4). Вимоги включають створення інтерфейсу, який коректно відображається на всіх типах екранів, із застосуванням сучасних CSS-фреймворків для спрощення стилізації. Інтерфейс має бути мінімалістичним, з акцентом на читабельність тексту та легкий доступ до основних функцій управління завданнями та інтелектуального модуля.

Рисунок 1.4 – Схематичне порівняння хорошого та поганого адаптивного дизайну

Trello – це візуальний інструмент для управління проєктами та завданнями, що використовує систему канбан-дошок. Дозволяє користувачам організовувати завдання у списки та картки, що полегшує відстеження прогресу та співпрацю [5].

Розробник: Atlassian. Архітектура: веб-додаток, десктопні та мобільні клієнти.

Функції:

– канбан-дошки для візуалізації завдань;
– створення карток із завданнями, списками, термінами;
– чек-листи та додавання файлів;
– спільна робота та коментування;

– інтеграції з іншими сервісами.

Переваги:

– інтуїтивно зрозумілий інтерфейс;

– візуалізація робочих процесів;

– можливість швидкого налаштування.

Недоліки:

– обмежений функціонал для великих та складних проєктів;

– може стати неорганізованим при великій кількості карток.

Таблиця 1.1 – Порівняльний аналіз функціональних можливостей аналогів та розроблюваної системи

Функція	**Notion**	**Asana**	**Obsidian**	**Trello**	**Розроблювана система**
Створення та редагування нотаток/завдань	+	+	+	+	+
Підтримка формату Markdown	+	-	+	-	+
Канбан-дошки	+	+	через плагін	+	+
Мобільний клієнт (iOS / Android)	+	+	+	+	+
Desktop-клієнт (Windows)	+	+	+	+	+
Єдина кодова база для всіх платформ	-	-	-	-	+
Офлайн-режим (offline-first)	частково	-	+	частково	+
Синхронізація між пристроями	+	+	через плагін	+	+
Командна співпраця	+	+	-	+	-

Продовження таблиці 1.1

Функція	**Notion**	**Asana**	**Obsidian**	**Trello**	**Розроблювана система**
Теги та пріоритети	+	+	+	+	+
Аналітика та звіти	частково	+	-	-	+
Автоматична генерація тексту (AI)	через інтеграцію	через інтеграцію	через плагін	-	+ (вбудовано)
Автоматичне узагальнення нотаток (AI)	-	-	через плагін	-	+ (вбудовано)
Локальне виконання AI-моделей	-	-	через плагін	-	+
Незалежність від зовнішніх AI API	-	-	-	-	+
Наскрізне шифрування резервних копій	-	-	-	-	+
Zero-knowledge резервне копіювання	-	-	-	-	+
Відкритий вихідний код	-	-	частково	-	+
Безкоштовний базовий доступ	частково	частково	+	частково	+

Аналіз прикладів на ринку дає змогу встановити чіткі вимоги до проєкту, використати позитивні характеристики популярних інформаційних систем управління завданнями та інтелектуальних інструментів, а також уникнути негативних рис, щоб розробити конкурентоспроможну інформаційну систему з інтелектуальним модулем генерації та аналізу.

1.2 Огляд методів, технологій розробки програмного забезпечення

Розробка будь-якої інформаційної системи вимагає ретельного вибору методів і технологій, які забезпечать ефективність, масштабованість і зручність продукту.

1.2.1. Вибір методології розробки. Для створення веб-додатку необхідно обрати методологію розробки, яка забезпечить структурований процес, ефективне управління завданнями та відповідність продукту вимогам. Основні методи, які розглядаються, включають:

– Agile – це методологія гнучкої розробки, яка першочергово сьогодні популярна в IT і дозволяє клієнтам швидше отримувати якісне програмне забезпечення. Застосування гнучких методів мінімізує проектні ризики завдяки розробці у вигляді коротких ітерацій, що називаються спринтами, забезпечуючи високу адаптивність до мінливих вимог предметної області [6];

– Waterfall – це модель, в якій кожен етап розробки, що відповідає стадії життєвого циклу ПЗ, продовжує попередній. Як демонструє класична інженерія програмного забезпечення, каскадний підхід є жорстко структурованим, де перехід до наступної фази відбувається лише після повного завершення та верифікації результатів попередньої [7]. Каскадна модель проста і зрозуміла, але не така практична, як раніше. Однак для проєкту, який розробляється одним розробником, методологія достатньо добре підходить.

1.2.2. Технології серверної частини та бази даних. Для розробки API-сервера передбачається використовувати Ruby on Rails 8, що функціонуватиме в режимі API-only.

Ruby on Rails (Rails) – це фреймворк для розробки веб-додатків, написаний мовою програмування Ruby. Він створений для спрощення та прискорення процесу розробки, надаючи готові рішення та конвенції. Rails дозволяє писати менше коду, досягаючи при цьому більшого, ніж багато інших мов і фреймворків [8]. Планується застосовувати Rails 8.1 у поєднанні з

сервером Puma та Thruster (для HTTP кешування та стиснення), що має забезпечити високу продуктивність та ефективність обробки запитів.

Як основна база даних буде обрана PostgreSQL 17. Це потужна об'єктно-реляційна система керування базами даних (СКБД), яка відома своєю надійністю, масштабованістю та підтримкою складних запитів. PostgreSQL вважається ідеальною для зберігання структурованих даних, що включатимуть інформацію про користувачів, нотатки, завдання та метадані для інтелектуального аналізу.

Для роботи з фоновими завданнями, кешуванням та вебсокетами в Rails 8 планується застосовувати вбудовані рішення: Solid Queue, Solid Cache та Solid Cable. Це дозволить уникнути залежності від зовнішніх систем, таких як Redis, та спростити інфраструктуру, використовуючи лише PostgreSQL як єдину базу даних.

Авторизація буде реалізована за допомогою Devise та devise-jwt для stateless JWT автентифікації, що забезпечить масштабованість та безпеку. Таблиця JwtDenylist буде використана для відкликання токенів при виході користувача. Для політики авторизації передбачається застосовувати Pundit.

Для адміністрування системи планується використовувати Motor Admin, який автоматично генерує зручну адміністративну панель за адресою /admin. Інтеграція зі Stripe дозволить обробляти платні підписки, включаючи вебхуки для автоматичного оновлення ролі користувача до ai_user. Асинхронні завдання, такі як резервне копіювання (через CreateBackupJob), будуть виконуватися за допомогою Solid Queue, що запобігатиме блокуванню основного потоку запитів при обробці великих зашифрованих даних.

1.2.3. Технології клієнтської частини. Клієнтська частина системи буде розроблена на базі React Native 0.81 з використанням TypeScript 5.8. Це дозволить створити єдину кодову базу для розгортання додатку на Windows Desktop, Android та iOS, забезпечуючи справжню кросплатформність.

React Native – це фреймворк для створення мобільних додатків, який дозволяє розробникам використовувати JavaScript та React для створення

інтерфейсів, що працюють на різних платформах. Це значно прискорює розробку та знижує витрати, оскільки не потрібно писати окремий код для кожної операційної системи. Завдяки TypeScript буде забезпечена строга типізація для API-контрактів та компонентів, що підвищить надійність коду.

Як локальне сховище для нотаток передбачається використовувати AsyncStorage, що дозволить додатку працювати в режимі "offline-first" – нотатки зберігатимуться на пристрої перед синхронізацією з сервером. Для візуалізації нотаток планується застосовувати бібліотеку react-native-markdown-display, а для кастомних іконок – react-native-svg.

Особлива увага буде приділена безпеці даних: для клієнтського наскрізного шифрування хмарних резервних копій передбачається використання crypto-js (AES-256-CBC + PBKDF2-SHA256). Ключ шифрування генеруватиметься з електронної пошти та пароля користувача, що дозволить відновлювати дані на будь-якому пристрої без того, щоб сервер мав доступ до незашифрованого тексту (zero-knowledge backups).

Інтерфейс користувача включатиме такі елементи, як файлове дерево у стилі VS Code, канбан-дошки, екран налаштувань, модальні вікна та підтримку темної/світлої теми за допомогою useColorScheme.

1.2.4. Технології інтелектуального модуля (AI Microservice). Інтелектуальний модуль планується реалізувати як окремий мікросервіс на Python з використанням FastAPI.

FastAPI – це сучасний, швидкий (високопродуктивний) веб-фреймворк для побудови API за допомогою Python 3.7+, заснований на стандартних підказках типу Python. Він забезпечує автоматичну валідацію даних, документацію та є асинхронним, що робить його ідеальним для розробки високонавантажених сервісів [9]. У парі з Uvicorn та Pydantic v2 він має забезпечити типізовані схеми запитів/відповідей та валідацію даних.

Для локального виконання висновків моделей машинного навчання передбачається використання бібліотеки Hugging Face transformers:

1 sshleifer/distilbart-cnn-12-6 для узагальнення тексту (summarization);

2 gpt2 для генерації тексту.

Моделі планується завантажувати лише один раз під час запуску мікросервісу за допомогою контексту lifespan FastAPI, що забезпечить їх тепле зберігання в пам'яті та повторне використання при кожному запиті без додаткових затримок. Мікросервіс буде розроблений для роботи на центральному процесорі (CPU-only PyTorch) без потреби у графічному процесорі (GPU).

Ізоляція AI-сервісу дозволить серверу Rails викликати його через HTTP та незалежно масштабувати, що вважається ключовим для оптимізації ресурсів та гнучкості архітектури.

1.2.5. Інфраструктура та DevOps. Для оркестрації всіх сервісів (PostgreSQL, AI та Backend) передбачається використання Docker Compose. Це дозволить легко управляти контейнерами, їхнім мережевим взаємодією та томами для зберігання даних бази даних, Ruby gems та кешу моделей Hugging Face.

Роздільні Dockerfile для розробки та продакшн-середовища (Dockerfile / Dockerfile.dev) мають забезпечити оптимальні конфігурації для кожного етапу. Мережева взаємодія між сервісами відбуватиметься за їхніми іменами (наприклад, Rails викликатиме http://ai:8000).

Система також включатиме виробничі інструменти, такі як Brakeman (статичний аналіз коду), bundler-audit (сканування CVE), RuboCop (стиль коду), RSpec (тестування) та Kamal (для розгортання), що має забезпечити високу якість та безпеку розробки.

Вибраний технологічний стек передбачає поліглотну архітектуру, де Ruby, TypeScript та Python будуть використовуватися відповідно до їхніх найкращих якостей. Це дозволить створити надійну, масштабовану та функціональну інформаційну систему управління завданнями з інтелектуальним модулем, що відповідатиме сучасним вимогам розробки та безпеки.

1.3 Розробка варіантів використання

Для чіткого визначення функціональності додатку необхідно розробити прецеденти використання (use cases), які описують взаємодію користувачів із системою. Прецеденти допомагають ідентифікувати ключові функції, ролі користувачів та сценарії використання, що є основою для подальшого проєктування та реалізації системи.

Описані прецеденти лягли в основу для створення діаграми прецедентів (рис. 2.1), яка візуально представляє всі основні варіанти використання системи та взаємозв'язки між ними та акторами.

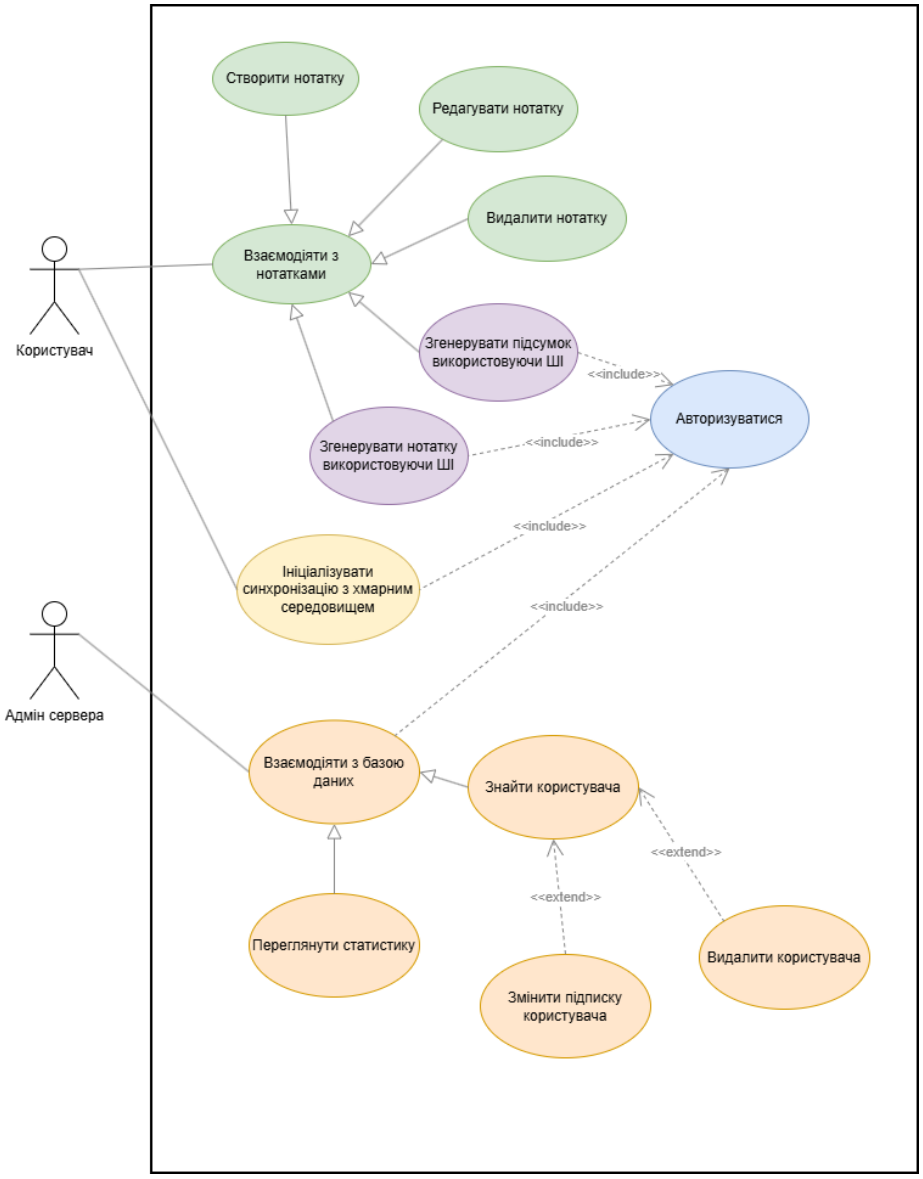

Рисунок 1.5 – Діаграма прецедентів

1.3.1. Перший прецедент. Створення та інтелектуальна обробка нотатки.

Користувач відкриває мобільний додаток для створення нової нотатки. Він вносить текстовий вміст, після чого ініціює функцію автоматичного підбиття підсумків за допомогою вбудованого модуля ШІ. Система обробляє текст на сервері, повертає стислий виклад та синхронізує оновлену нотатку з хмарним сховищем. Користувач отримує підтвердження про успішне збереження та готовність короткого підсумку.

Поверхнева форма.

Коли користувач авторизується у мобільному додатку на базі React Native, він може створити нову нотатку. Після введення тексту він натискає кнопку «Згенерувати підсумок». Запит надсилається на сервер Ruby on Rails, який через API взаємодіє з модулем ШІ. Отриманий результат відображається в інтерфейсі додатка. При натисканні кнопки збереження дані синхронізуються із сервером. Якщо з’єднання з інтернетом відсутнє, нотатка зберігається локально і буде синхронізована пізніше. У разі помилки модуля ШІ система повідомляє користувача про неможливість генерації підсумку, зберігаючи при цьому основний текст нотатки.

Повна форма.

Назва прецеденту: “Створити та підсумувати нотатку”.

Сфера застосування: мобільний додаток та серверна частина системи.

Рівень: ціль користувача.

Головні актори: користувач.

Другорядні актори: сервер Ruby on Rails, модуль штучного інтелекту.

Зацікавлені сторони та їхні інтереси:

– користувач: хоче швидко зафіксувати інформацію та отримати її стислий виклад для економії часу в майбутньому;

– розробник: зацікавлений у стабільній роботі синхронізації та коректній відповіді від ШІ;

– система: повинна забезпечити цілісність даних між мобільним пристроєм та сервером.

Передумови:

– користувач авторизований у мобільному додатку;

– наявне підключення до мережі Інтернет (для функцій ШІ та синхронізації);

– запущений сервер Ruby on Rails та доступ до API модуля ШІ.

Гарантія успіху:

– нотатка збережена в базі даних на сервері;

– згенерований ШІ-підсумок успішно відображений та закріплений за нотаткою;

локальна копія на пристрої відповідає серверній.

Основний сценарій успіху:

1 користувач створює нову нотатку в додатку React Native;

2 користувач вводить або вставляє текст нотатки;

3 користувач обирає функцію «Підсумувати через ШІ»;

4 додаток надсилає запит на сервер Ruby on Rails;

5 сервер перенаправляє запит до модуля ШІ для обробки тексту;

6 сервер отримує результат та відправляє його назад у додаток;

7 додаток відображає підсумок користувачу;

8 користувач підтверджує збереження;

9 система синхронізує дані та повідомляє про успішне завершення операції.

Альтернативні сценарії.

а) Відсутність підключення до мережі:

1 система сповіщає, що функції ШІ та синхронізація наразі недоступні;

2 нотатка зберігається локально;

3 синхронізація відбудеться автоматично після відновлення зв’язку.

б) Помилка модуля ШІ (перевищення лімітів або збій API):

1 система повідомляє про технічну помилку обробки тексту;

2 користувачу пропонується зберегти нотатку без підсумку або спробувати пізніше.

в) Конфлікт версій під час синхронізації:

1 система виявляє, що нотатка була змінена на іншому пристрої;

2 користувачу пропонується обрати версію для збереження або об'єднати вміст.

Спеціальні вимоги:

– час відповіді модуля ШІ не повинен перевищувати 5-7 секунд;

– передача даних між React Native та RoR має бути захищена протоколом HTTPS;

– додаток повинен коректно працювати в офлайн-режимі (збереження тексту);

– інтерфейс має бути адаптивним для різних розмірів екранів мобільних пристроїв.

Список технологій і варіацій даних:

– Frontend: React Native (для кросплатформеності);

– Backend: Ruby on Rails;

– автентифікація: JWT (JSON Web Token) або гем Devise;

– база даних: PostgreSQL (сервер) або AsyncStorage (локально);

– інтеграція ШІ: OpenAI API або внутрішня LLM модель;

– синхронізація: RESTful API запити.

Частота використання: висока (основна функція додатку).

1.3.2. Другий прецедент. Синхронізація та резервне копіювання Простору.

Користувач ініціює процес резервного копіювання своїх нотаток у хмару. Система збирає всі локальні Markdown-файли, що належать до обраного Простору, пакує їх у єдиний JSON та виконує наскрізне шифрування на стороні клієнта. Зашифрований контейнер надсилається на сервер Ruby on

Rails, де він зберігається як бінарний об’єкт, прив’язаний до унікального ключа синхронізації. Користувач отримує повідомлення про успішне створення бекапу.

Поверхнева форма.

Коли користувач натискає кнопку синхронізації в налаштуваннях Простору в додатку React Native для Windows, система автоматично перевіряє цілісність локальних Markdown-файлів. Додаток формує JSON файл, шифрує його за допомогою локального ключа (AES-256) і надсилає на сервер Ruby on Rails через захищене з’єднання. Сервер приймає файл і оновлює запис у базі даних PostgreSQL, посилаючись на новий об’єкт у сховищі Active Storage. У разі відсутності інтернету або помилки авторизації, система повідомляє про неможливість синхронізації та пропонує повторити спробу пізніше.

Повна форма.

Назва прецеденту: «Синхронізація та резервне копіювання Простору».

Сфера застосування: десктопний додаток та серверна частина системи.

Рівень: ціль користувача.

Головні актори: користувач.

Другорядні актори: сервер Ruby on Rails, локальна файлова система Windows.

Зацікавлені сторони та їхні інтереси:

– користувач: хоче мати надійну резервну копію своїх нотаток для відновлення на іншому пристрої та гарантію приватності (адмін сервера не має бачити контент);

– адміністратор сервера: зацікавлений у цілісності збережених архівів та відсутності несанкціонованого доступу до API;

– система: повинна забезпечити швидку архівацію та безпечну передачу даних без втрат.

Передумови:

– користувач авторизований у системі;

– користувач створив або імпортував хоча б одну нотатку у Простір;

– наявне стабільне підключення до мережі Інтернет.

Гарантія успіху:

– локальні файли успішно заархівовані та зашифровані;

– зашифрований контейнер збережений на сервері та доступний за унікальним sync_key;

– користувач бачить актуальну дату останньої синхронізації.

Основний сценарій успіху:

1 користувач обирає Простір для синхронізації та натискає «Sync Now»;

2 додаток зчитує всі .md файли та metadata.json з локальної папки Простору;

3 система формує тимчасовий JSON файл;

4 модуль безпеки шифрує архів за алгоритмом AES-256, використовуючи ключ, що базується на паролі користувача;

5 додаток надсилає POST-запит із зашифрованим файлом на сервер Ruby on Rails, вказуючи sync_key;

6 сервер перевіряє права доступу користувача до даного Простору.

7 сервер зберігає файл у сховищі (Active Storage) та оновлює контрольну суму (checksum) у базі даних;

8 сервер повертає статус «200 OK»;

9 додаток видаляє тимчасовий архів і оновлює статус синхронізації в інтерфейсі.

Альтернативні сценарії.

а) Помилка авторизації (недійсний токен):

1 система повідомляє про завершення сесії;

2 користувачу пропонується знову ввести логін та пароль;

3 після успішного входу процес синхронізації відновлюється.

б) Конфлікт версій (на сервері вже є новіший бекап з іншого пристрою):

1 система виявляє розбіжність у мітках часу;

2 користувачу пропонується: «Завантажити версію з хмари» або «Перезаписати хмарну версію локальною».

в) Недостатньо місця на сервері або обмеження розміру файлу:

1 сервер повертає помилку про перевищення квоти;

2 користувач отримує сповіщення про неможливість завантаження та рекомендацію видалити застарілі дані.

Спеціальні вимоги:

– процес архівації та шифрування не повинен блокувати основний інтерфейс користувача (використання фонових потоків);

– сервер повинен зберігати щонайменше одну попередню версію простору (versioning).

Список технологій і варіацій даних:

– Frontend: React Native;

– Backend: Ruby on Rails, Active Storage;

– автентифікація: JWT (JSON Web Token) або гем Devise;

– шифрування: AES-256 (на стороні клієнта);

– формат даних: ZIP-контейнер із Markdown-файлами;

– база даних: PostgreSQL;

Частота використання: середня (виконується автоматично при виході або вручну користувачем).

За створеними прецедентами створюється діаграма прецедентів.

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених межею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть діаграми прецедентів полягає в тому, що проєктована система подається у вигляді множини сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Проектування прецедентів з використанням уніфікованої мови моделювання UML дозволяє чітко специфікувати межі інформаційної системи та зафіксувати функціональні вимоги замовника ще до початку активної фази кодування [10].

Варіант використання використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система під час діалогу з актором. При цьому нічого не говориться про те, яким чином буде реалізовано взаємодію акторів із системою.

У даному розділі було розглянуто процес розробки можливих прецедентів використання для майбутнього програмного додатку для ведення та структуризації нотаток. Шляхом визначення ключових акторів системи – звичайного користувача та адміністратора серверної частини – та їхніх основних взаємодій із системою, було сформовано чітке уявлення про функціональні вимоги до додатку.

1.4 Постановка завдання на розробку

На основі проведеного аналізу аналогів та огляду сучасних технологій розробки можна сформулювати чітке завдання на створення інформаційної системи управління завданнями з інтелектуальним модулем генерації та аналізу.

Аналіз існуючих рішень – Notion, Asana, Obsidian та Trello – виявив низку спільних обмежень: недостатню інтеграцію інтелектуальних можливостей, складність освоєння для нових користувачів, високу вартість для малих команд або відсутність повноцінної кросплатформності. Огляд технологій підтвердив наявність зрілого стеку (Ruby on Rails, React Native, FastAPI, PostgreSQL), який дозволяє усунути ці недоліки в рамках єдиної системи.

Для реалізації проєкту необхідно виконати наступні завдання:

1 визначення предметної області: «Інформаційна система управління завданнями з інтелектуальним модулем генерації та аналізу»;

2 визначення структури розроблюваної інформаційної системи на основі результатів порівняльного аналізу аналогів;

3 визначення функціональних особливостей системи, включаючи функціонал інтелектуального модуля;

4 проектування архітектури системи: мобільний клієнт на React Native, серверна частина на Ruby on Rails та AI-мікросервіс на FastAPI;

5 розробка системи відповідно до спроектованої архітектури та визначених вимог;

6 тестування працездатності та ефективності системи.

Виконання цих завдань забезпечить створення інформаційної системи, яка поєднує найкращі практики існуючих аналогів із новими інтелектуальними можливостями: автоматичною генерацією підзавдань, аналізом прогресу та персоналізованими рекомендаціями щодо пріоритетності, при збереженні зручності інтерфейсу та безпеки даних.

Висновки до розділу 1

У першому розділі виконано аналіз предметної області та обґрунтовано доцільність розробки інформаційної системи управління завданнями з інтелектуальним модулем генерації та аналізу.

Проведено огляд існуючих програмних рішень – Notion, Asana, Obsidian та Trello. Виділено їхні ключові особливості, переваги та недоліки. Аналіз показав, що, незважаючи на широкі функціональні можливості цих систем, вони мають спільні обмеження: недостатній рівень інтелектуальної автоматизації, складність освоєння, високу вартість для малих команд або обмежену кросплатформність. Це підтвердило необхідність розробки власного рішення, яке усуне виявлені недоліки.

Здійснено огляд підходів і технологій, що застосовуються при розробці подібних застосунків, та обґрунтовано вибір інструментів для кожної складової системи.

Як методологію розробки обрано Waterfall – послідовний підхід, який добре підходить для індивідуальної роботи з чітко визначеними вимогами на початку проєкту.

Серверну частину побудовано на Ruby on Rails у режимі API з PostgreSQL як основною базою даних. Для обробки фонових завдань, кешування та роботи в реальному часі використано вбудовані рішення платформи. Автентифікацію реалізовано через токени, а розмежування прав доступу – через окремий механізм політик.

Клієнтський застосунок розроблено на React Native з TypeScript, що дозволяє підтримувати Windows, Android та iOS в межах єдиної кодової бази. Дані зберігаються локально на пристрої, контент відображається у форматі Markdown, а чутливі дані перед відправкою шифруються безпосередньо на клієнті.

Для роботи зі штучним інтелектом розроблено окремий мікросервіс на Python з використанням FastAPI. Він запускає дві моделі локально, без звернення до зовнішніх API: одну для автоматичного стиснення тексту нотатки, іншу для генерації тексту за запитом користувача. Сервіс розрахований на роботу без відеокарти, що знижує вимоги до інфраструктури.

Усі компоненти системи об'єднані через Docker Compose, що спрощує розгортання і забезпечує однакове середовище виконання на різних машинах. Якість коду і безпека підтримуються статичним аналізом і автоматизованим тестуванням.

Розроблені варіанти використання (прецеденти) чітко визначили функціональність додатка. Два основні прецеденти – «Створення та інтелектуальна обробка нотатки» та «Синхронізація та резервне копіювання Простору» – детально описують взаємодію користувачів із системою, включаючи основні та альтернативні сценарії, зацікавлені сторони, передумови та гарантії успіху. Це дозволило ідентифікувати ключові функції, ролі користувачів та сценарії використання, що є фундаментом для подальшої

реалізації системи. Діаграма прецедентів візуалізувала ці взаємодії, забезпечуючи чітке розуміння функціональних вимог.

На основі результатів аналізу у розділі 1.4 сформульовано постановку завдання на розробку, яка визначає структуру, функціональні особливості та етапи реалізації системи.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОБЛІКУ ЗАВДАНЬ

2.1 Проєктування бази даних

Проєктування бази даних можна визначити як набір процедур або сукупність завдань, що включають різні кроки, необхідні для реалізації бази даних. Хороший дизайн бази даних є дуже важливим – він допомагає отримати потрібну інформацію тоді, коли вона потрібна.

Для реалізації проєкту обрано архітектуру Local-First, де основним місцем зберігання даних є локальне сховище мобільного пристрою (AsyncStorage у React Native). Нотатки створюються та редагуються у форматі Markdown. Серверна частина на Ruby on Rails виконує роль хмарного сховища для зашифрованих резервних копій даних користувача. Структуру бази даних наведено на рисунку 2.1.

Основними компонентами серверної бази даних є наступні таблиці.

Таблиця users забезпечує ідентифікацію та авторизацію користувачів:

– id – унікальний ідентифікатор користувача;

– email – логін користувача;

– encrypted_password – зашифрований хеш пароля;

– role – роль користувача в системі (наприклад, звичайний або AI-користувач);

– stripe_customer_id – ідентифікатор клієнта в платіжній системі Stripe;

– created_at, updated_at – часові мітки створення та оновлення запису.

Таблиця user_backups відповідає за зберігання зашифрованих резервних копій даних конкретного користувача:

– id – унікальний ідентифікатор запису;

– user_id – зовнішній ключ, зв'язок із таблицею users;

– checksum – контрольна сума для перевірки цілісності даних;

– data – зашифровані дані резервної копії у форматі JSONB;

– notes_count – кількість нотаток у резервній копії;

– created_at, updated_at – часові мітки створення та оновлення запису.

Таблиця jwt_denylists забезпечує безпеку автентифікації шляхом відкликання недійсних токенів:

– id – унікальний ідентифікатор запису;

– jti – унікальний ідентифікатор JWT-токена;

– exp – дата та час закінчення дії токена.

Окрему групу складають таблиці адмін-панелі Motor, які забезпечують адміністративний інтерфейс системи. Таблиця queries зберігає збережені SQL-запити адмін-панелі. Таблиця alerts містить налаштування сповіщень, прив'язаних до запитів, а таблиця alert_locks фіксує часові мітки спрацювання сповіщень. Таблиці notes, note_tags та note_tag_tags забезпечують систему внутрішніх нотаток адміністратора з підтримкою тегування. Таблиці tags та taggable_tags реалізують універсальний механізм тегування для будь-яких об'єктів системи. Діаграма бази даних зображена на рисунку 2.1.

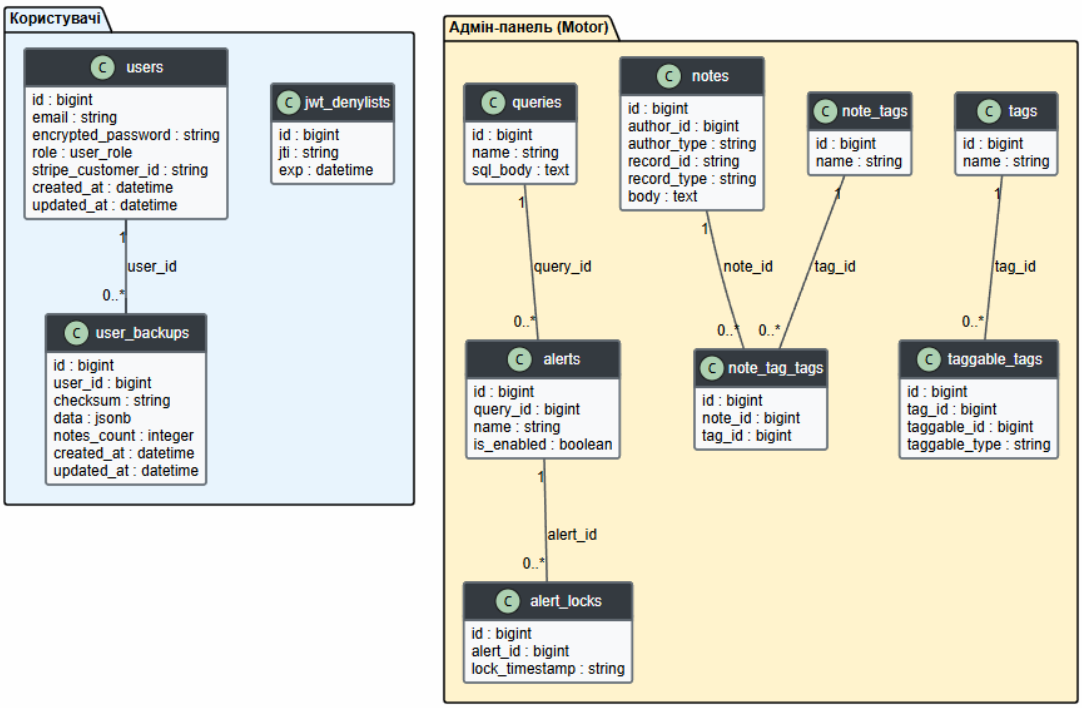

Рисунок 2.1 – Діаграма бази даних

2.2 Проєктування компонентів

Діаграма компонентів розбиває фактичну систему, що розробляється, на різні високі рівні функціональності. Кожен компонент відповідає за одну чітку мету в рамках всієї системи і взаємодіє з іншими важливими елементами тільки в разі необхідності [11]. Архітектура розділена на три основні рівні: клієнтська частина (Mobile Client), серверна частина (Backend Server) та зовнішні сервіси (AI Service).

Система складається з 3 компонентів.

Мобільний клієнт (React Native Application):

– інтерфейс користувача: відповідає за рендеринг Markdown-файлів та інтерфейс управління спейсами;

– Storage Manager: компонент, що здійснює пряму взаємодію з локальною файловою системою пристрою для читання та запису .md файлів;

– Security Module: реалізує логіку наскрізного шифрування, генерацію ключів та шифрування архівів перед синхронізацією;

– Sync Manager: відповідає за архівацію спейсів у ZIP-контейнери та їх передачу на сервер.

Серверна частина (Ruby on Rails Backend):

– API Controller: забезпечує RESTful інтерфейс для комунікації з мобільним клієнтом;

– Auth Service: компонент, що відповідає за реєстрацію та авторизацію користувачів;

– Active Storage Manager: використовує Active Storage для управління зашифрованими архівами (blobs) на сервері.

Зовнішня інфраструктура:

– Database (PostgreSQL): зберігає метадані користувачів та ключі синхронізації спейсів.

– AI Engine: зовнішня мовна модель, що виконує інтелектуальну обробку тексту.

Діаграма компонентів зображена на рисунку 2.2.

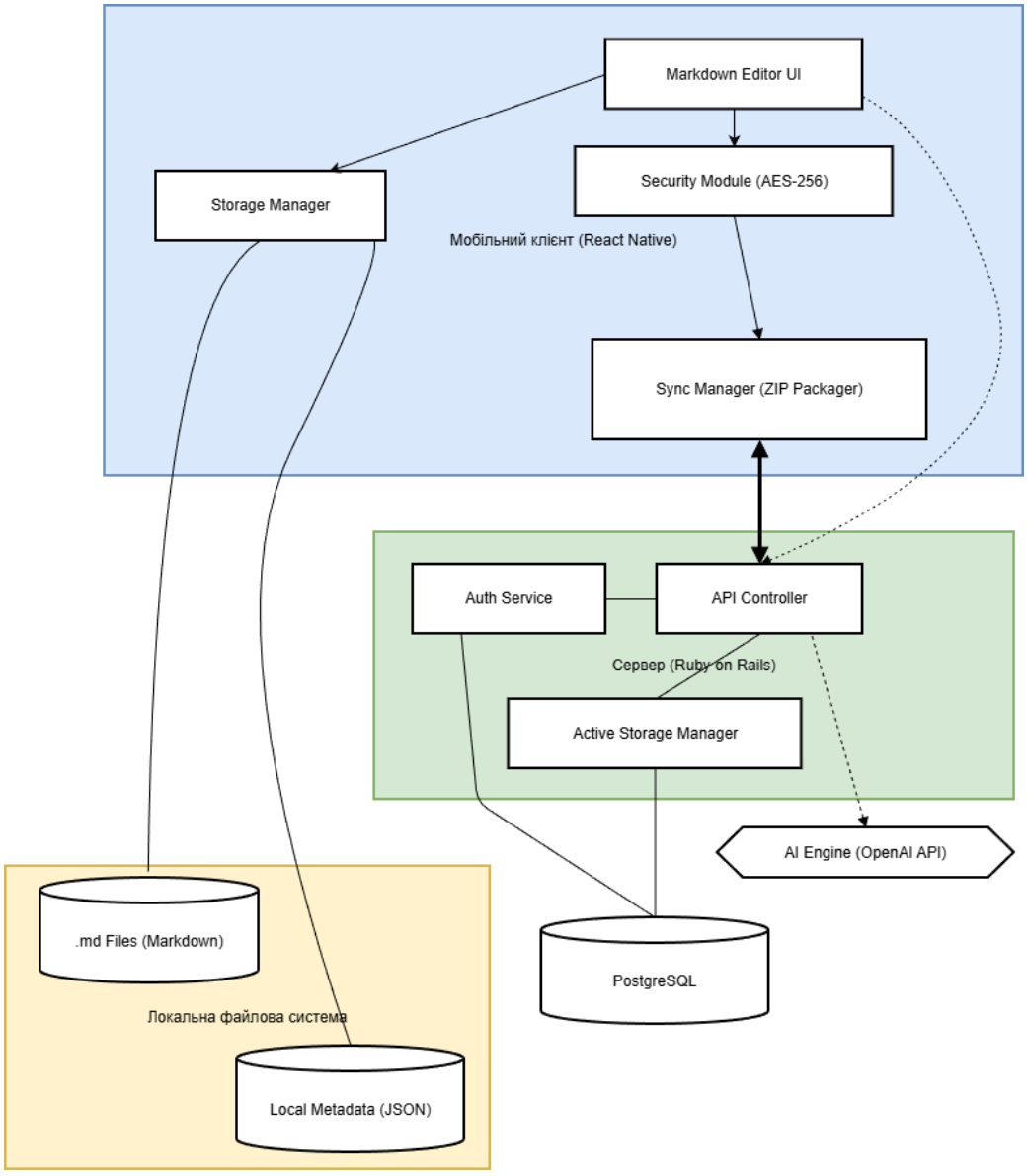

Рисунок 2.2 – Діаграма компонентів

2.3 Вибір архітектури додатку

Враховуючи специфіку проєкту, що поєднує мобільний інтерфейс та віддалений сервер для синхронізації, було обрано клієнт-серверну архітектуру. Серверна частина розробляється на базі фреймворку Ruby on

Rails у режимі API-only, що дозволяє використовувати її як незалежний бекенд для мобільного додатка на React Native.

Такий підхід забезпечує гнучкість розробки, бо мобільний клієнт відповідає за логіку відображення (UI) та роботу з локальними Markdown-файлами, тоді як сервер фокусується на безпечному зберіганні архівних копій та взаємодії зі штучним інтелектом.

Файлова структура наведена в таблиці 2.1.

Таблиця 2.1 – Файлова структура Ruby on Rails додатку

Назва директорії/файлу	**Призначення**
app/	Містить основний код програми.
bin/	Скрипти для запуску, оновлення і налаштування програми.
config/	Файли конфігурації (наприклад, routes.rb, database.yml).
db/	Міграції, схема бази даних (schema.rb), початкові дані (seeds.rb).
lib/	Кастомні бібліотеки, модулі, утиліти.
log/	Логи роботи додатку.
public/	Статичні файли, які віддаються напряму веб-сервером (favicon, error pages).
coverage/	Файли для відстеження покритого тестами коду
spec/	Тестові файли.
tmp/	Тимчасові файли, що використовуються під час виконання.
Gemfile	Список Ruby-гемів (залежностей) для додатку.
Gemfile.lock	Зафіксовані версії гемів, щоб забезпечити однакові залежності.
Rakefile	Опис задач, які можна виконати через rake.
README.md	Документація до проекту.

Ruby on Rails застосунок побудований на MVC шаблоні.

Модель–вигляд–контролер (MVC) – архітектурний шаблон, який використовується під час проєктування та розробки програмного забезпечення. Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Розділення бізнес-логіки та механізмів відображення інформації в межах MVC суттєво підвищує загальну читабельність вихідного коду та спрощує

паралельну роботу над серверними ендпоінтами та клієнтськими екранами [12]. Враховуючи, що серверна частина передбачена як API-Only, то за частину вигляду будуть відповідати серіалайзери.

Також потрібно навести структуру мобільного додатка. Оскільки React Native базується на компонентному підході, його структура фокусується на розділенні інтерфейсу, логіки управління станом та сервісів для роботи з файловою системою та API.

Файлова структура React Native дадатку наведена в таблиці 2.2.

Таблиця 2.2 – Файлова структура React Native додатку

Назва директорії або файлу	**Призначення**
src/	Основна директорія з вихідним кодом JavaScript / TypeScript.
src/components/	Перевикористовувані UI-компоненти для десктопного інтерфейсу.
src/screens/	Основні екрани: "Markdown Editor", "AI Request Modal", "Settings".
src/services/	Модулі для роботи з API, логіка шифрування та синхронізації.
src/storage/	Логіка управління локальною файловою системою Windows (.md файли).
src/navigation/	Налаштування маршрутизації між екранами додатку.
app.json	Метадані додатку для ОС Windows (назва, іконка, версія).
metro.config.js	Налаштування збирача Metro для середовища Windows.
package.json	Перелік JS-залежностей, включаючи react-native-windows.

Таким чином вся система розподілена на дві великі частини: мобільний інтерфейс та сервер.

2.4 Проєктування інтерфейсу користувача

Інтерфейс додатка розроблено з урахуванням специфіки систем, де пріоритетом є раціональне використання робочого простору, легка навігація

між великою кількістю файлів та зручність редагування тексту. Дизайн виконано у стилі сучасного мінімалізму з акцентом на контенті.

Ключові компоненти графічного інтерфейсу (GUI):

1. Бічна панель навігації (Sidebar):

– реалізовано ієрархічну структуру «Просторів» та папок (наприклад, "Personal", "Work"), що дозволяє логічно групувати Markdown-нотатки. Користувач може миттєво перемикатися між файлами.

– у верхній частині панелі розміщені кнопки для швидкого створення нових документів та папок.

– у нижній частині бічної панелі інтегровано календар, що допомагає користувачеві орієнтуватися в датах створення або редагування записів, що важливо для ведення щоденників або планів.

2. Робоча область редактора (Main Editor):

– верхня частина редактора містить панель інструментів для швидкого редагування тексту (жирний шрифт, курсив, заголовки, списки, вставка посилань та коду). Це дозволяє користувачеві працювати з Markdown, не знаючи синтаксису мови розмітки.

– передбачено перемикання між режимами «Edit» (редагування файлу) та «Preview» (візуальне відображення Markdown-файлу).

3. Модуль інтеграції зі штучним інтелектом (AI Actions):

– у правому верхньому куті виділено блок інтелектуальних функцій: «Generate with AI» та «Summarize». Використано яскраві кольори для цих кнопок, що акцентує увагу на ключових можливостях додатка. Функція «Summarize» дозволяє отримати стислий підсумок поточної нотатки, а «Generate with AI» – автоматично доповнити текст на основі введеного запиту.

4. Інформаційна панель (Status Bar):

– Нижня панель додатка відображає статистику: кількість слів, символів та рядків у поточному документі. Також виводиться час останньої модифікації файлу.

Сформований макет інтерфейсу зображено на рисунку 2.3.

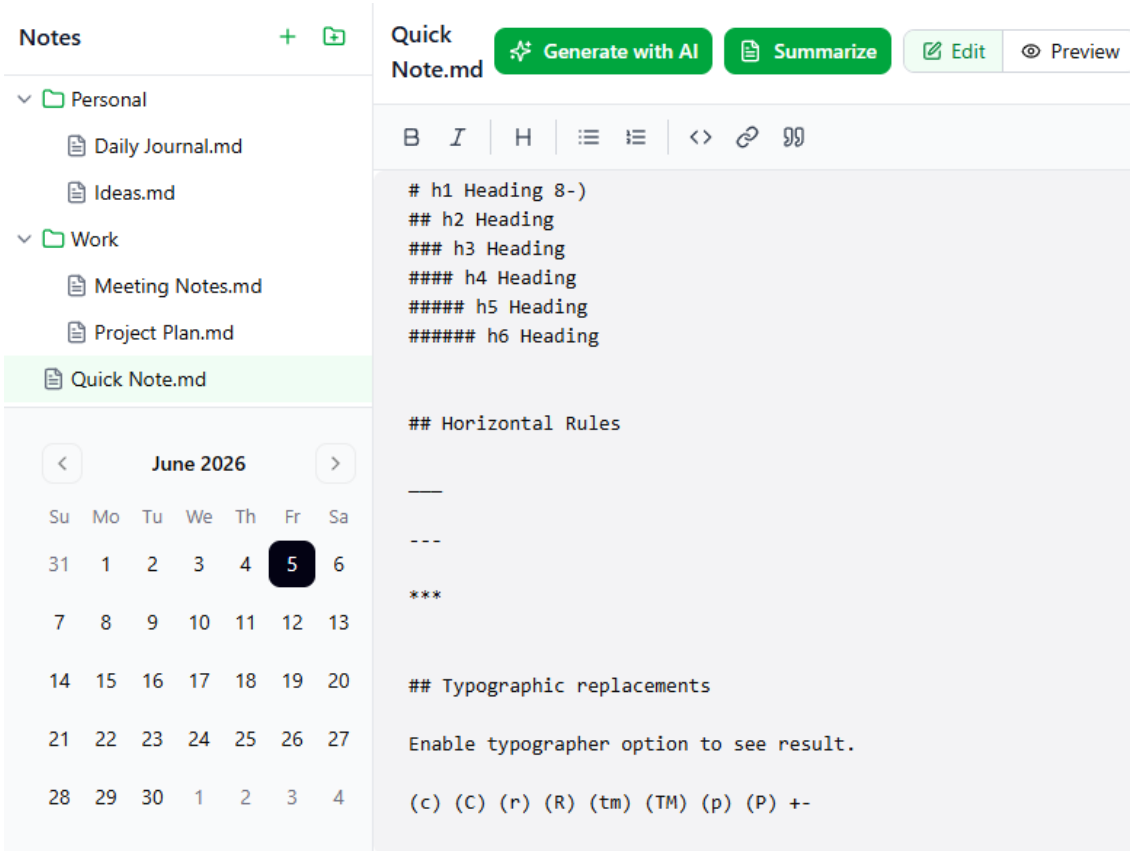

Рисунок 2.3 – Макет інтерфейсу

Висновки до розділу 2

У другому розділі було детально розглянуто процес проєктування інформаційної системи управління завданнями з інтелектуальним модулем генерації та аналізу.

Проєктування бази даних врахувало обрану архітектуру Local-First, де дані першочергово зберігаються локально на пристрої у форматі Markdown. Серверна частина (Ruby on Rails) виступає в ролі хмарного сховища для зашифрованих архівних копій. Були визначені основні таблиці на сервері («User» та «Space») для ідентифікації користувачів та зберігання бекапів, а

також локальна файлова структура на мобільному пристрої, що включає Markdown-файли нотаток та метадані.

Проєктування компонентів системи базується на клієнт-серверній архітектурі, розділеній на три основні рівні: Мобільний клієнт (React Native Application), Серверна частина (Ruby on Rails Backend) та Зовнішня інфраструктура (Database, AI Engine). Кожен компонент має чітко визначені обов'язки, що забезпечує модульність та масштабованість системи. Діаграма компонентів наочно ілюструє взаємодію між цими частинами.

Вибір клієнт-серверної архітектури з Ruby on Rails у режимі API-only та React Native для мобільного клієнта забезпечує гнучкість розробки та ефективний розподіл відповідальності. Детально описано файлову структуру як для Ruby on Rails, так і для React Native додатків, що відображає архітектурні шаблони MVC (для сервера) та компонентний підхід (для клієнта).

Проєктування інтерфейсу користувача було виконано з акцентом на мінімалізм, раціональне використання робочого простору та зручність навігації. Ключові елементи графічного інтерфейсу, такі як бічна панель навігації, робоча область редактора, модуль інтеграції зі штучним інтелектом та інформаційна панель, були детально описані. Макет інтерфейсу візуалізує ці елементи, підкреслюючи їх функціональне призначення та інтуїтивність, що сприятиме комфортній роботі користувачів з великою кількістю нотаток та інтелектуальними функціями

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ФУНКЦІОНАЛУ СИСТЕМИ УПРАВЛІННЯ ЗАВДАННЯМИ ТА ІНТЕЛЕКТУАЛЬНИЙ МОДУЛЬ

3.1 Реалізація логіки додатку

3.1.1. Реалізація автентифікації та авторизації. Реалізація автентифікації та авторизації. Автентифікація та авторизація побудовані як послідовний ланцюг. Спочатку користувач входитв або реєструється, після чого клієнт отримує JWT-токен і зберігає його локально. Кожен захищений запит надсилається з токеном у заголовку Authorization, сервер визначає поточного користувача, а для чутливих функцій – зокрема AI – застосовуються політики доступу Pundit. Побудова безпечної безсесійної архітектури (stateless architecture) з використанням веб-токенів вимагає дотримання суворих криптографічних стандартів шифрування та правильних механізмів верифікації підписів на стороні сервера [13]. Коротко кажучи, автентифікація відповідає за ідентифікацію користувача, а авторизація – за перевірку його прав.

Після успішного входу клієнт отримує два ключові об'єкти: user і token. Токен зберігається в локальному сховищі, користувач також кешується – щоб швидко відновити стан інтерфейсу після перезапуску застосунку без зайвих затримок.

```
const {user, token} = await login(email.trim(), password);
if (token) {
  await saveAuthToken(token);
}
await saveAuthUser(user);
```

Рисунок 3.1 – Використання об’єктів user і token в коді

На старті застосунку виконується перевірка локальної сесії: якщо токен і користувач вже існують, застосунок одразу переходить в авторизований стан без повторного логіну. Це дає швидкий запуск без зайвих екранів входу,

уніфіковану роботу на всіх клієнтських платформах і мінімальну залежність від мережі для базового UX.

Для всіх запитів використовується централізована функція API-клієнта. Якщо запит не позначений як публічний, токен підтягується зі сховища і автоматично додається в заголовок Authorization – жодного дублювання коду в кожному окремому виклику.

```
const headers: Record<string, string> = {
  'Content-Type': 'application/json',
  Accept: 'application/json',
};

if (options.authenticated !== false) {
  const token = await getAuthToken();
  if (token) {
    headers['Authorization'] = `Bearer ${token}`;
  }
}
```

Рисунок 3.2 – Визначення заголовків для запитів до сервера

3.1.2. Реалізація управління нотатками та папками. Реалізація управління нотатками та папками. У підсистемі управління нотатками реалізовано локальну, швидку та офлайн-орієнтовану модель роботи: усі операції з нотатками і папками виконуються на клієнті, а дані зберігаються в AsyncStorage. Використання AsyncStorage разом із парадигмою Offline-First забезпечує безперебійний інтерфейс (UI Response), оскільки користувач уникає очікування мережевих запитів під час виконання базових операцій додавання чи модифікації записів [14]. Це забезпечує миттєвий відгук інтерфейсу, незалежність від мережі та просту синхронізацію з бекендом як окремий етап.Основні компоненти та модулі системи: UI-провідник нотаток і папок FileTree.tsx, редактор однієї нотатки NoteEditor.tsx, сервіс локального збереження fileStorage.ts, оркестрація стану в кореневому компоненті App.tsx і тип сутності нотатки Note.ts.

Кожна нотатка містить ідентифікатор, заголовок, контент, часові мітки, ім’я markdown-файлу, а також опційно папку і статус для Kanban-

представлення. Це дозволяє одній сутності одночасно підтримувати і файлову, і дошкову логіку.

```
export interface Note {
  id: string;
  title: string;
  content: string;
  createdAt: Date;
  updatedAt: Date;
  filename: string;
  folder?: string;
  status?: string;
}
```

Рисунок 3.3 – Інтерфейс об’єкту “Нотатка”

Модуль fileStorage.ts інкапсулює всі CRUD-операції. Розробка збереження даних у реляційних та локальних NoSQL структурах базується на стандартизованих шаблонах проектування доступу до даних, що дає змогу повністю ізолювати низькорівневі виклики збереження від високорівневої бізнес-логіки [15]. saveNote створює або оновлює нотатку, loadNote отримує її за id, getAllNotes повертає весь список відсортований за updatedAt, deleteNote видаляє нотатку за filename, а getAllFolders, createFolder і deleteFolder відповідають за роботу з папками.

Під час створення нової нотатки генерується безпечне ім'я markdown-файлу (slug + timestamp), що зменшує ризик колізій.

```
const filename =
  note.filename || generateFilename(note.title || 'untitled');
const id = note.id || filename.replace('.md', '');


const existingIndex = allNotes.findIndex(n => n.id === noteData.id);
if (existingIndex >= 0) {
  allNotes[existingIndex] = noteData;
} else {
  allNotes.push(noteData);
}
```

Рисунок 3.4 – Фрагмент коду, який відповідає за створення нотатки

Папки зберігаються окремо списком назв. При видаленні папки нотатки не втрачаються: їх поле folder очищається, тобто вони переміщуються в корінь. Це важливий UX-запобіжник від випадкової втрати даних.

```
const updated = allNotes.map(note =>
  note.folder === name ? {...note, folder: undefined} : note,
);
await AsyncStorage.setItem(NOTES_STORAGE_KEY, JSON.stringify(updated));
```

Рисунок 3.5 – Логіка роботи папок

FileTree.tsx відповідає за відображення кореневих нотаток і папок, розгортання і згортання папок, створення нотаток у корені або всередині папки, контекстне меню для видалення та підтвердження небезпечних дій через модальні вікна.

Компонент працює з локальним станом hovered/expanded/creating та підвантажує фактичні дані через getAllNotes і getAllFolders. Після кожної операції викликається load, щоб синхронізувати відображення зі сховищем.

3.1.3. Реалізація хмарного резервного копіювання та відновлення. Підсистема хмарного резервного копіювання побудована як безпечний двоетапний процес. Спочатку локальні дані нотаток готуються на клієнті, за потреби шифруються і відправляються на сервер. Потім сервер асинхронно створює резервну копію, зберігає метадані та payload, а при відновленні повертає останній знімок.

Обробник резервної копії у App.tsx зчитує всі локальні сутності – нотатки, папки, Kanban-колонки – перевіряє наявність ключа шифрування, формує payload і викликає API створення backup, після чого показує користувачу результат через модальне повідомлення. Якщо ключ є, дані шифруються на клієнті; якщо ключа немає (наприклад, старий акаунт), відправляється незашифрований payload як fallback.

```
const handleBackup = useCallback(async () => {
  if (backupStatus === 'saving') {return;}
  setBackupStatus('saving');
  try {
    const [notes, folders, kanbanColumns, encKey] = await Promise.all([
      getAllNotes(),
      getAllFolders(),
      getKanbanColumns(),
      getEncryptionKey(),
    ]);

    if (encKey) {
      const encrypted = encryptPayload(
        {notes, kanban_columns: kanbanColumns, folders},
        encKey,
      );
      await createBackup({encrypted: true, encrypted_data: encrypted, notes_count: notes.length});
    } else {
      await createBackup({encrypted: false, notes, kanban_columns: kanbanColumns, folders});
    }
    showAlert('Backup Saved', 'Your notes have been backed up successfully.');
  } catch (e: any) {
    showAlert('Backup Failed', __DEV__ ? String(e) : 'Could not save backup. Please try again.');
  } finally {
    setBackupStatus('idle');
  }
}, [backupStatus, showAlert]);
```

Рисунок 3.6 – Основна логіка роботи резервної копії

Процес restore у App.tsx викликає endpoint отримання останнього backup, за потреби розшифровує payload локально, зберігає нотатки і колонки у локальне сховище та оновлює UI через refreshKey. Додатково реалізована логіка авто-пропозиції відновлення після логіну: якщо локальних нотаток ще немає, але в хмарі є backup, користувачу показується confirm-діалог для імпорту.

В api.ts реалізовано окремі методи – createBackup, getBackups, getLatestBackup і getBackupById. Усі вони працюють через централізований apiRequest з JWT-заголовком Authorization, тобто резервні копії завжди прив'язані до поточного автентифікованого користувача.

Шифрування у cryptoStorage.ts побудоване так: ключ генерується з пароля та email через PBKDF2-SHA256, payload шифруватиметься AES-256-CBC з новим IV для кожного сеансу. Створення систем із нульовим рівнем розголошення (Zero-Knowledge) вимагає перенесення всіх криптографічних процедур виключно на сторону клієнта, гарантуючи, що сервер зберігає лише зашифровані бінарні структури і не має технічної можливості дешифрувати приватний контент користувача [16].

Контролер backups_controller.rb надає чотири дії: index повертає історію копій з метаданими, latest і show – останню або конкретну копію з даними, create – ставить задачу у чергу. Важливо, що резервна копія не формується синхронно в HTTP-потоці: замість цього викликається CreateBackupJob, який делегує збереження сервісу BackupService.

```ruby
def create
  payload = backup_params.to_h
  CreateBackupJob.perform_later(current_user.id, payload)
  render json: {
    status: { code: 202, message: "Backup scheduled." }
  }, status: :accepted
end
```

Рисунок 3.7 – Метод сервіса, який відповідає за створення резервної копії

BackupService підтримує два формати payload – encrypted та plain, рахує checksum через SHA256 для контролю цілісності, зберігає notes_count, checksum і data, а також виконує prune старих копій, залишаючи лише обмежену історію, ліміт якої визначений у user_backup.rb.

3.1.4. Реалізація інтеграції штучного інтелекту. Інтеграція ШІ реалізована як окрема функціональна підсистема з чітким розподілом відповідальності між клієнтом, основним API-сервером і AI-мікросервісом. Загальний принцип такий: клієнт ініціює AI-операцію в редакторі нотатки, Rails API перевіряє автентифікацію та права доступу, після чого делегує обчислення Python-сервісу, а результат повертається назад у редактор.

У поточній версії реалізовано дві операції. Summarize автоматично створює стисле резюме поточної нотатки, Generate – формує текст за промптом користувача. Обидві дії доступні прямо в редакторі нотаток без переходу на окремі екрани, що зменшує контекстні перемикання і пришвидшує роботу з текстом.

У NoteEditor.tsx кнопки Summarize і Generate запускають асинхронні обробники. Клієнт перевіряє наявність тексту або промпту, викликає

відповідний API-метод, вставляє результат у тіло нотатки та відображає помилки або paywall-сценарій за відсутності прав.

```
try {
  const summary = await summarizeNote(content);
  setContent(c => `${c}\n\n## AI Summary\n\n${summary}\n\n> *Generated by AI*`);
```

Рисунок 3.8 – Додавання коментаря в нотатку, що текст згенерований ШІ

У ai_controller.rb перед фактичним викликом моделі застосовується авторизація через Pundit. Це означає, що навіть валідний JWT не гарантує доступ до AI-функцій без потрібної ролі. Весь вміст файлу наведено в додатку А.

Правила доступу винесені в ai_policy.rb: і summarize?, і generate? дозволені лише тоді, коли user.ai_access? повертає істину.

AiNoteService у ai_note_service.rb формує HTTP-запит до FastAPI, передає потрібні параметри моделі, перетворює помилки зовнішнього сервісу у керовані помилки домену і повертає контролеру лише необхідний текстовий результат.

```
def self.generate(prompt, max_new_tokens: 100, temperature: 0.7)
  post("/generate", { prompt: prompt, max_new_tokens: max_new_tokens, temperature: temperature }) do |body|
    body["generated_text"]
  end
end
```

Рисунок 3.9 – Метод, в якому генерується і виконується запит до ШІ-сервісу

app.py реалізує FastAPI-сервіс, який завантажує моделі при старті застосунку, надає маршрути /summarize та /generate, повертає структуровані відповіді через Pydantic-схеми [19] і обробляє помилки з відповідними HTTP-кодами 422, 503 і 500.

```python
@app.post("/generate", response_model=GenerateResponse)
async def generate_text(request: GenerateRequest):
    if generator is None:
        raise HTTPException(status_code=503, detail="Text-generation model not available")

    try:
        result = generator(
            request.prompt,
            max_new_tokens=request.max_new_tokens,
            temperature=request.temperature,
            do_sample=True,
            pad_token_id=50256,  # GPT-2 EOS token used as pad
        )
        return GenerateResponse(generated_text=result[0]["generated_text"])
    except Exception as exc:
        logger.error("Text generation failed: %s", exc)
        raise HTTPException(status_code=500, detail=str(exc))
```

Рисунок 3.10 – Метод генерації тексту в ШІ-сервісі

В AI-потоці реалізовано кілька захисних механізмів: валідація порожнього вводу на клієнті і сервері, розділення помилок доступу (403) і технічних помилок сервісу, показ paywall-сценарію для користувачів без AI-доступу та індикатори завантаження, що блокують повторні натискання під час активного запиту.

3.1.5. Реалізація системи підписок та управління доступом. Система підписок реалізована як зв'язка трьох частин: клієнт ініціює оформлення доступу до AI, сервер створює checkout-сесію та обробляє події від Stripe [20], а права на AI перевіряються через ролі та політики доступу. Ключові модулі: NoteEditor.tsx, api.ts на клієнті та subscriptions_controller.rb, webhooks_controller.rb, user.rb, ai_policy.rb, ai_controller.rb і routes.rb на сервері.

У моделі користувача реалізована рольова схема з трьох рівнів: user – базовий користувач, ai_user – користувач з підпискою на AI, admin – адміністратор, який також має AI-доступ.

```ruby
def ai_access?
  ai_user? || admin?
end
```

Рисунок 3.11 – Метод для визначення доступу до ШІ-сервісу

Коли користувач натискає Subscribe, клієнт викликає createCheckoutSession у api.ts, який звертається до ендпоінту subscriptions/checkout у subscriptions_controller.rb. Сервер спочатку перевіряє, чи не має користувач вже активного доступу – і якщо так, повертає успішну відповідь без повторного checkout. У development формується mock checkout URL, у production створюється повноцінна Stripe Checkout Session.

Після завершення оплати Stripe надсилає подію на webhook-ендпоінт webhooks/stripe, описаний у routes.rb. Обробник у webhooks_controller.rb спочатку перевіряє підпис Stripe для захисту від фейкових подій, після чого при події checkout.session.completed підвищує роль користувача до ai_user, а при customer.subscription.deleted – знижує назад до user.

Навіть після активації підписки доступ до AI-функцій додатково контролюється в ai_controller.rb. Перед кожною операцією викликається authorize, а політика ai_policy.rb дозволяє дію лише для користувачів з ai_access? = true.

У NoteEditor.tsx при спробі виконати AI-дію без відповідних прав сервер повертає 403, клієнт показує paywall-модальне вікно, а по кнопці Subscribe відкривається checkout URL. Після успішної оплати права активуються через webhook і оновлення ролі. Це дозволяє користувачу пройти шлях від відмови в доступі до оформлення підписки прямо з робочого сценарію, без зайвих переходів.

3.2. Додаткові інструменти та користувацький досвід

3.2.1. Елементи інтерфейсу та взаємодія. Інтерфейс застосунку побудовано за принципом робочого простору: ліворуч користувач керує структурою даних, праворуч працює з вмістом. Такий підхід зменшує кількість переходів між екранами і підтримує безперервний сценарій роботи з нотатками.

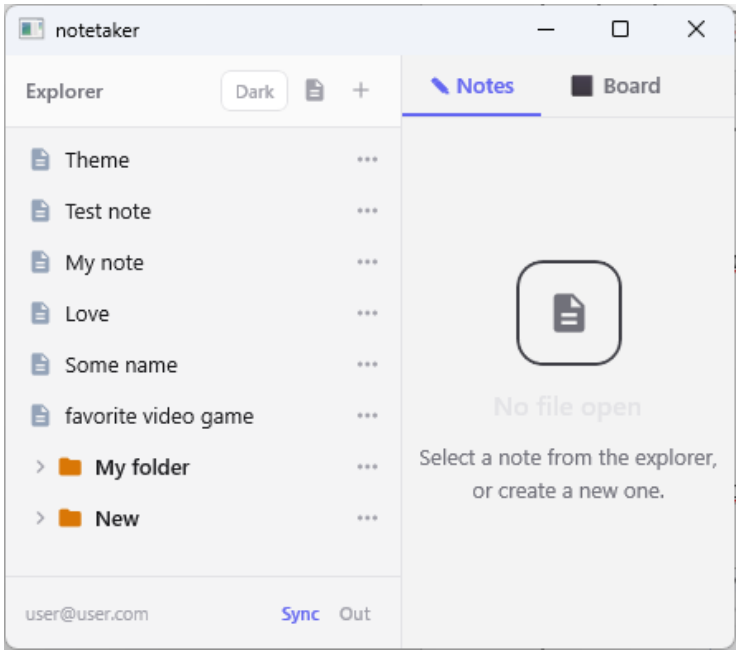

Рисунок 3.12 – Головне меню додатку

Основний контейнер інтерфейсу реалізовано у App.tsx, де поєднуються ліва панель-провідник для навігації по нотатках і папках, центральна область контенту для редактора, welcome-екрана або kanban-дошки, верхні вкладки перегляду Notes / Board, контекстне меню для швидких дій та глобальний модальний шар для підтверджень, помилок і введення значень. Такий layout наслідує desktop-патерн і є інтуїтивним для користувачів редакторів коду та knowledge-інструментів.

FileTree.tsx відповідає за файлово-папкову модель інтерфейсу. Компонент містить заголовок Explorer з кнопками швидкого створення, рядки нотаток із візуальним станом вибору, рядки папок із розгортанням і згортанням, inline-поле для створення нової нотатки або папки та нижню user-панель з діями Sign In, Sync і Out залежно від стану авторизації.

Клік по нотатці відкриває її у редакторі, клік по папці перемикає стан expanded/collapsed, через контекстне меню доступні видалення та створення, а небезпечні дії супроводжуються confirm-діалогом. Інтерфейс миттєво відображає зміни після кожної CRUD-операції.

NoteEditor.tsx реалізує центральний сценарій створення контенту. Редактор включає верхню панель дій з кнопками Back, Preview/Edit і Save,

поле заголовка, поле markdown-контенту, горизонтальний toolbar швидкого форматування та AI-панель із кнопками Summarize і Generate. Перемикання між режимами Edit і Preview не втрачає введений текст, дії форматування працюють з виділеним фрагментом або курсором, а після збереження список у провіднику оновлюється автоматично. За наявності Kanban-колонок доступна зміна статусу нотатки, AI-операції мають індикатори завантаження і керовані стани помилок.

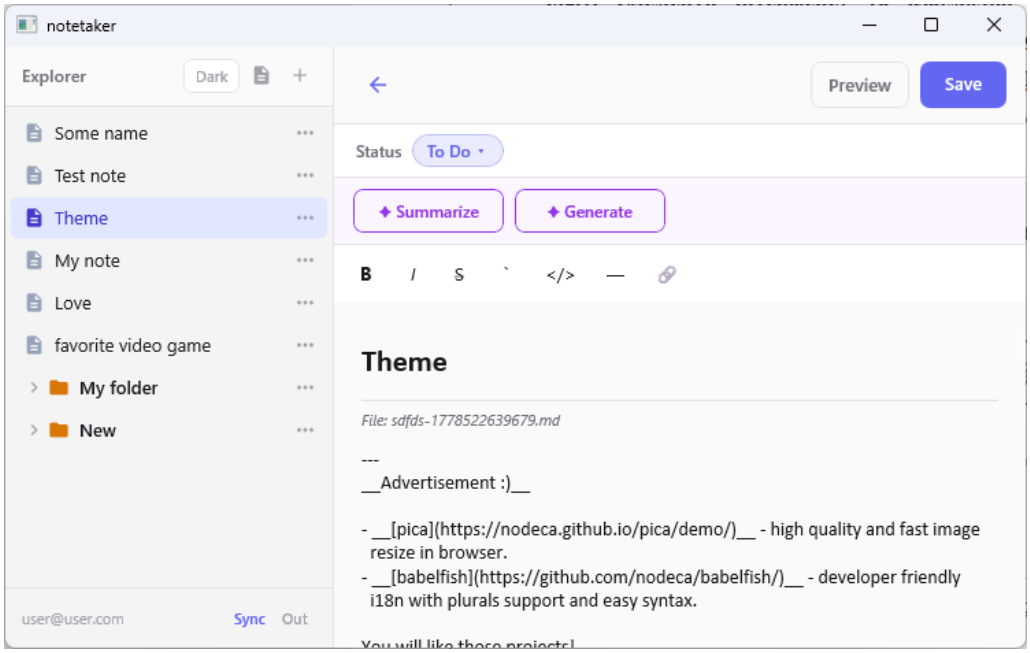

Рисунок 3.13 – Редактор нотатки

KanbanBoard.tsx підтримує альтернативний спосіб роботи з нотатками як із задачами. Дошка складається з горизонтального списку колонок із картками нотаток, які містять заголовок, preview, дату і тег папки. Колонки можна додавати, перейменовувати і видаляти, а картки підтримують drag-and-drop із візуальним overlay під час перетягування. Короткий клік по картці відкриває нотатку в редакторі, довге натискання активує drag-режим, а відпускання в іншій колонці змінює статус нотатки. Усі зміни зберігаються в локальне сховище і відразу відображаються в UI.

LoginScreen.tsx і RegisterScreen.tsx побудовані у card-форматі з валідацією полів і станом loading. Некоректні або порожні поля блокуються повідомленням, успішний вхід переводить у робочий простір, є можливість

тимчасово пропустити авторизацію, а помилки сервера показуються в модальному вікні з дружнім текстом. Компонент, який відповідає за сторінку логіну, наведено в додатку Б.

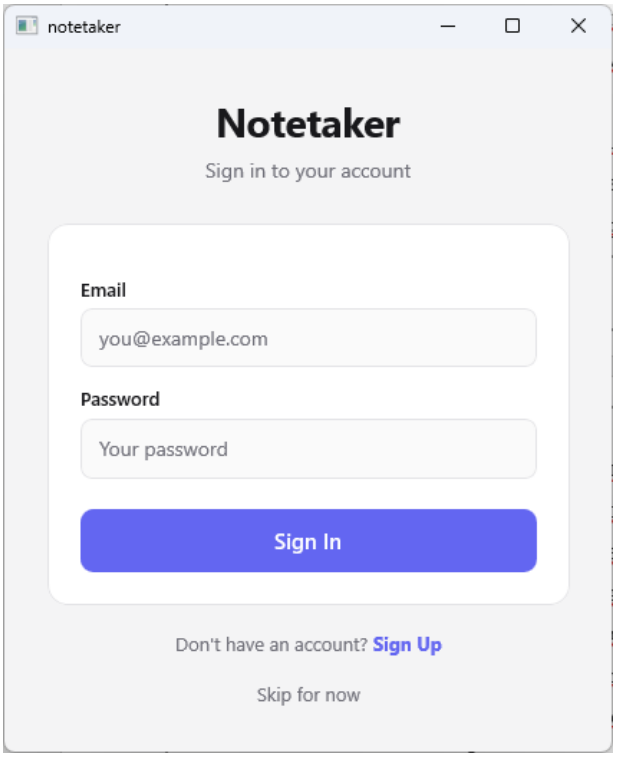

Рисунок 3.14 – Форма авторизації

Загальний механізм виринаючих вікон винесено у AppModal.tsx та useModal.ts. Це забезпечує єдину поведінку для Alert, Confirm, Prompt і Custom-діалогів, консистентний візуальний стиль у всіх частинах застосунку та просте повторне використання в FileTree, NoteEditor, KanbanBoard і auth-екранах. Централізоване керування кнопками, destructive-станами та введенням тексту усуває дублювання логіки по компонентах. Коли користувач хоче, згенерувати текст за допомогою ШІ, він нажимає на відповідну кнопку, після чого з’являється вікно для вводу теми, за якою штучний інтелект згенерує текст.

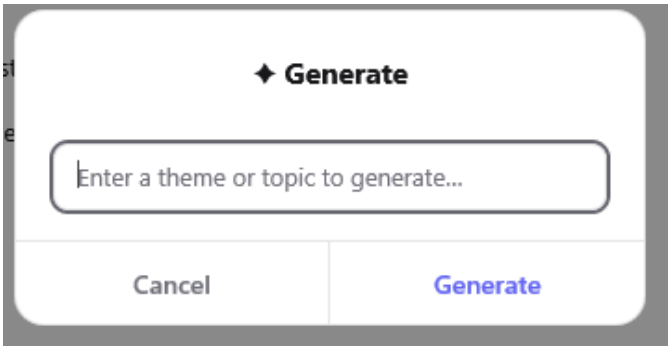

Рисунок 3.15 – Вікно генерації тексту

Після деякого часу текст появиться автоматичну в нотатці, також з підписом, що це згенеровано за домогою ШІ. Приклад згенерованого тексту зображено на рисунку 3.16.

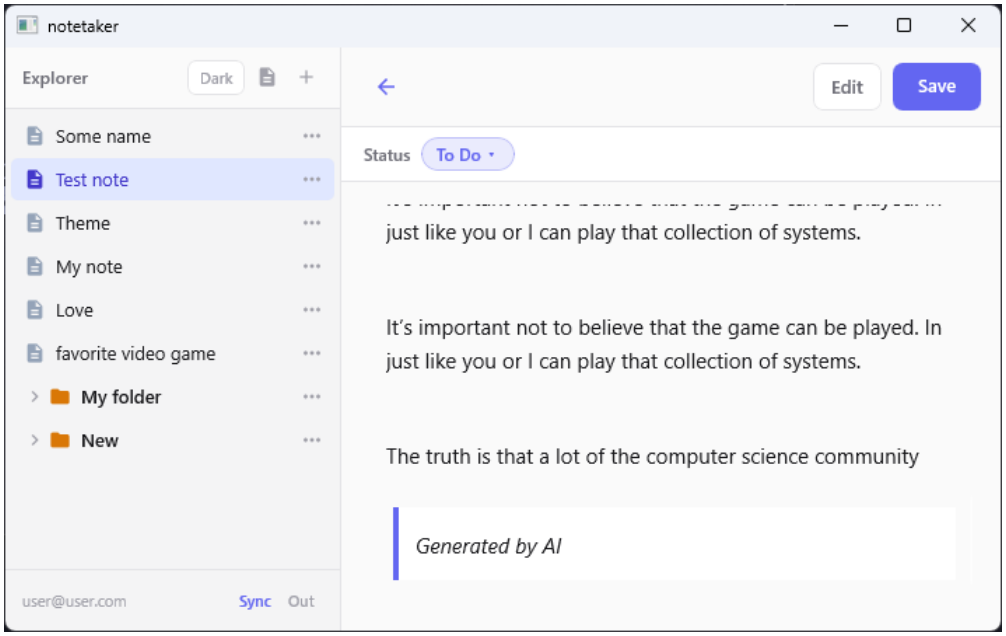

Рисунок 3.16 – Вікно генерації тексту

Інтерфейс підтримує світлу та темну тему через useColorScheme у ключових компонентах – App.tsx, FileTree.tsx, NoteEditor.tsx і KanbanBoard.tsx. Це забезпечує контрастність і читабельність у різних умовах, узгоджені кольорові токени для меж, акцентів, тексту та фонів і цілісне сприйняття інтерфейсу без різких стилістичних розривів.

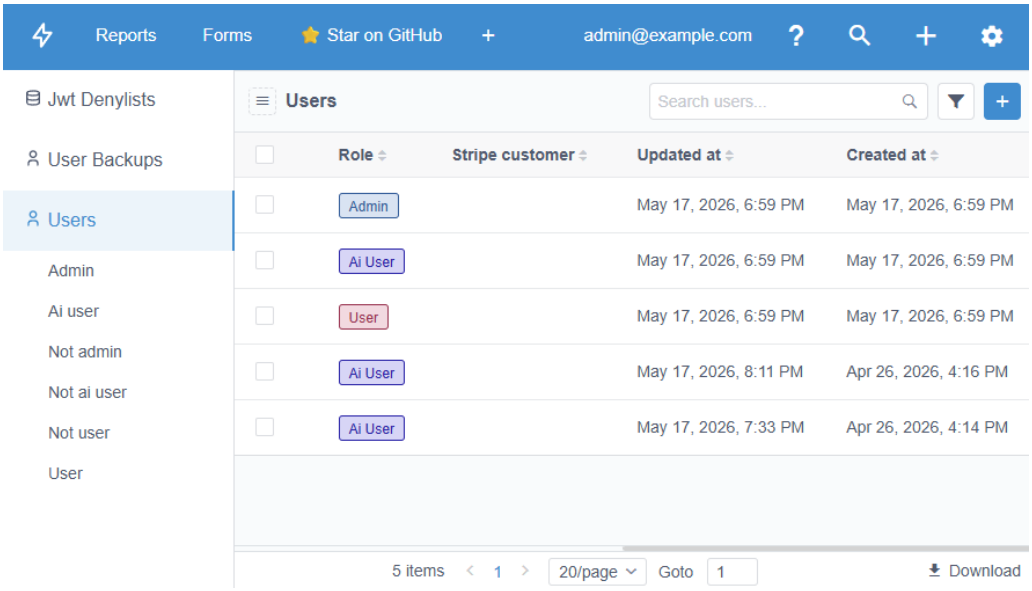

Рисунок 3.17 – Вікно адміна

3.2.2. Забезпечення кросплатформності та тем оформлення. Кросплатформність у застосунку досягається за рахунок єдиної кодової бази React Native з адаптацією поведінки під конкретну ОС у критичних точках. Це дозволяє підтримувати Windows, Android та iOS без дублювання бізнес-логіки, зберігаючи однаковий користувацький сценарій роботи з нотатками, дошкою та синхронізацією.

Архітектурно центр керування інтерфейсом реалізовано у App.tsx, де формується базовий layout і перемикання між основними режимами – редактор, дошка і вітальний екран. Платформна незалежність на рівні даних підтримується через AsyncStorage у fileStorage.ts, завдяки чому операції з нотатками працюють однаково на всіх цільових платформах.

Окремі платформні відмінності враховано явно. Визначення базового URL API залежить від ОС у api.ts: для Android-емулятора використовується 10.0.2.2, для інших платформ – localhost. У текстовому редакторі NoteEditor.tsx є умовна логіка для spellCheck, яка враховує специфіку Windows, а поведінка авторизаційних форм у LoginScreen.tsx та RegisterScreen.tsx адаптується через KeyboardAvoidingView з урахуванням iOS-поведінки.

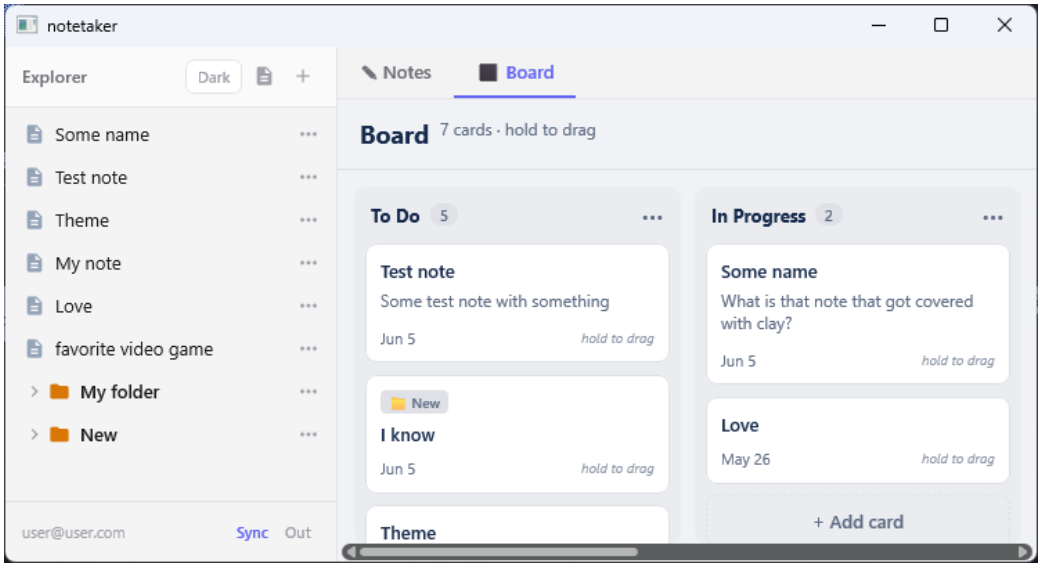

Рисунок 3.18 – Канбан дошка

Підтримка тем оформлення реалізована через useColorScheme() у ключових компонентах: App.tsx, FileTree.tsx, NoteEditor.tsx, KanbanBoard.tsx і AppModal.tsx. Кожен компонент має власний об'єкт кольорів, що динамічно формує палітру для темної та світлої тем. Це забезпечує контрастність тексту і фону, узгоджені акцентні кольори елементів керування, єдиний візуальний стиль для модальних вікон, форм, списків і карток та коректне сприйняття інтерфейсу при системному перемиканні теми.

Важливий практичний ефект такого підходу полягає в тому, що тема не є зовнішньою накладкою, а інтегрована в кожен UI-модуль окремо. Тому застосунок виглядає цілісно в обох режимах і не втрачає читабельність у складних екранах – markdown-редакторі, kanban-дошці та модальних діалогах.

Таким чином, кросплатформність і темізація реалізовані як частина базової архітектури інтерфейсу: єдиний код для логіки, контрольовані платформні винятки та системний темний і світлий рендер у всіх критичних компонентах.

3.2.3. Структура файлів та типизація. Структура проєкту організована за модульним принципом: інтерфейс, бізнес-логіка та серверні відповідальності розділені по окремих директоріях, що спрощує підтримку, тестування і масштабування функціональності. На верхньому рівні

монорепозиторію виділено три головні частини: frontend – клієнтський застосунок на React Native, backend – API-сервер на Rails і ai – окремий мікросервіс на FastAPI. Діаграма взаємодії з системою зображена на рисунку 3.18.

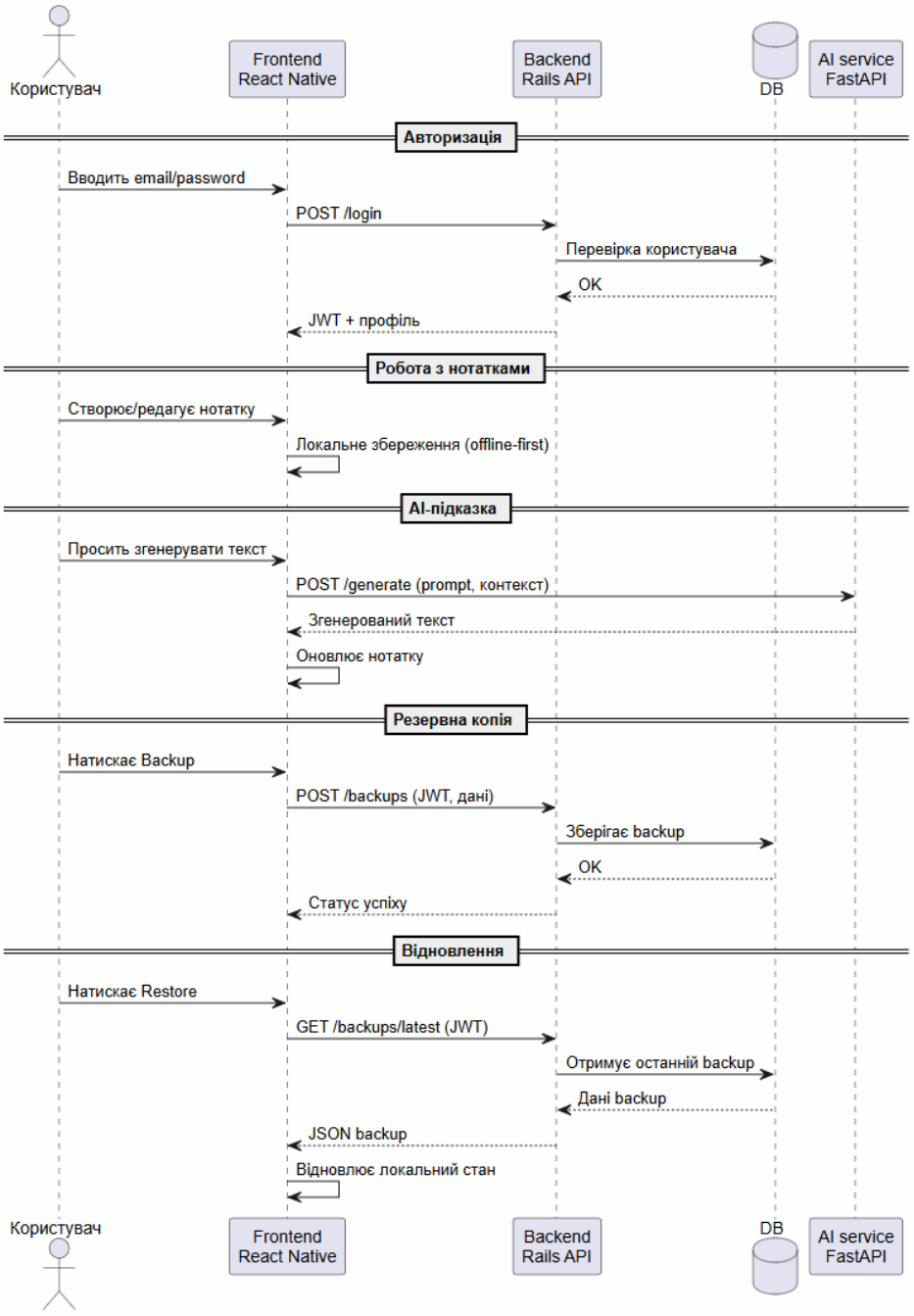

Рисунок 3.19 – Діаграма використання інформаційної системи обліку завдань

У фронтенді структура побудована навколо ролей модулів: компоненти інтерфейсу зосереджені в components, типи доменних сутностей – у types,

утиліти та сервісні шари – у utils, а точкою композиції головного UI є App.tsx. Такий поділ не дозволяє змішувати рендеринг, стан і роботу з API або сховищем в одному місці. Наприклад, UI-поведінка дерева файлів описана в FileTree.tsx, а операції читання і запису нотаток винесені у fileStorage.ts.

У бекенді використано класичну Rails-структуру з поділом за відповідальністю: контролери API у v1, моделі у models, політики авторизації у policies, сервісні класи у services, фонові задачі у jobs і серіалізатори відповіді у serializers. Доступ до AI контролюється в ai_controller.rb через політику ai_policy.rb, а зовнішній виклик до AI-сервісу інкапсульовано в ai_note_service.rb.

Типізація у фронтенді використовується як основний механізм узгодження даних між компонентами та сервісами. Доменна модель нотатки в Note.ts задає єдину форму об'єкта Note для редактора, провідника і kanban. Тип користувача AuthUser визначений у authStorage.ts і використовується у flow логіну і реєстрації. Типи backup-пейлоадів – BackupMeta, BackupData і BackupWithData – описані в api.ts, що робить синхронізацію безпечнішою, а типізовані пропси компонентів у NoteEditor.tsx та KanbanBoard.tsx зменшують ризик помилок інтеграції. Практичний результат такого підходу: невідповідності формату даних виявляються на етапі розробки, а не в runtime.

У Ruby-частині немає compile-time типізації як у TypeScript, проте контракти підтримуються іншими засобами. Серіалізатори user_serializer.rb і backup_serializer.rb фіксують формат відповідей, політики Pundit – зокрема ai_policy.rb – описують правила доступу, контролери на кшталт backups_controller.rb явно валідують вхідні параметри, а Pydantic-схеми у app.py забезпечують строгий контракт request/response на Python-боці.

Висновки до розділу 3

У третьому розділі описано практичну реалізацію застосунку для керування нотатками з підтримкою штучного інтелекту. Розробка охопила

повний стек: клієнтську частину на React Native, серверний API на Ruby on Rails і окремий AI-мікросервіс на FastAPI.

Підсистема автентифікації та авторизації побудована на основі JWT-токенів з локальним кешуванням сесії, що забезпечує швидкий старт застосунку без повторного логіну. Централізований API-клієнт автоматично додає токен до захищених запитів, а політики доступу Pundit контролюють права на рівні окремих функцій, зокрема AI-операцій.

Управління нотатками реалізовано за офлайн-орієнтованою моделлю: усі CRUD-операції виконуються локально через AsyncStorage, що гарантує миттєвий відгук інтерфейсу незалежно від стану мережі. Архітектурне розділення між UI-компонентами і сервісним шаром збереження робить кодову базу передбачуваною і легкою для розширення.

Хмарне резервне копіювання реалізоване з підтримкою end-to-end шифрування: дані шифруються на клієнті за допомогою AES-256-CBC ще до відправки на сервер, тож сервер зберігає лише непрозорий blob і не має доступу до тексту нотаток. Асинхронна обробка через фонові задачі забезпечує стабільний API-відгук навіть при великих обсягах даних.

Інтеграція штучного інтелекту організована як триланковий пайплайн: клієнт ініціює запит, Rails API виконує авторизацію і делегує обчислення FastAPI-сервісу, який повертає результат безпосередньо в редактор нотатки. Доступ до AI-функцій контролюється через рольову модель і Stripe-підписку, а весь шлях від paywall-сценарію до активації прав реалізований без зайвих переходів між екранами.

Інтерфейсна частина побудована за desktop-патерном робочого простору з підтримкою файлової навігації, markdown-редактора і kanban-дошки в єдиному контурі. Кросплатформність досягається єдиною кодовою базою з контрольованими платформними винятками для Android, iOS і Windows. Підтримка світлої та темної теми інтегрована в кожен UI-модуль окремо, що забезпечує цілісне сприйняття інтерфейсу в обох режимах.

Модульна структура монорепозиторію і TypeScript-типізація на клієнті в сукупності знижують кількість інтеграційних помилок і спрощують подальший розвиток функціональності. Серверна частина компенсує відсутність статичної типізації через серіалізатори, Pundit-політики і Pydantic-схеми на боці AI-сервісу.

Таким чином, реалізований застосунок поєднує офлайн-надійність, безпеку даних, керовану монетизацію і зручний UX в єдину архітектурно цілісну систему, придатну до масштабування і подальшого розширення функціональності.

ВИСНОВКИ

У дипломній роботі розроблено інформаційну систему управління нотатками з інтегрованим модулем штучного інтелекту, яка охоплює мобільний клієнт, серверну частину та окремий сервіс для роботи з мовними моделями.

У першому розділі проаналізовано предметну область та розглянуто існуючі програмні рішення – Notion, Asana, Obsidian і Trello. Аналіз показав, що більшість з них або не мають вбудованих інтелектуальних функцій, або залежать від зовнішніх хмарних сервісів для обробки даних, або не забезпечують належного рівня конфіденційності. На основі цього обґрунтовано необхідність розробки власної системи, яка поєднувала б офлайн-орієнтовану модель збереження даних, наскрізне шифрування резервних копій і локальний запуск моделей штучного інтелекту без залежності від зовнішніх API.

У другому розділі виконано проєктування системи. Розроблено варіанти використання, що описують два ключові сценарії – інтелектуальну обробку нотатки та хмарне резервне копіювання. Спроєктовано базу даних з урахуванням архітектури Local-First, де основним місцем зберігання є пристрій користувача, а сервер виконує роль захищеного хмарного сховища. Визначено компонентну структуру системи та обґрунтовано вибір клієнт-серверної архітектури з окремим мікросервісом для штучного інтелекту. Розроблено макет інтерфейсу, орієнтований на мінімалізм і зручність роботи з текстом.

У третьому розділі описано практичну реалізацію всіх підсистем. Автентифікацію побудовано на основі токенів із локальним кешуванням сесії, що забезпечує швидкий старт без повторного входу. Управління нотатками реалізовано локально на пристрої, що гарантує миттєвий відгук інтерфейсу незалежно від наявності мережі. Резервне копіювання виконується з шифруванням на стороні клієнта, тому сервер не має доступу до вмісту

нотаток. Інтеграція штучного інтелекту організована як триланковий пайплайн між клієнтом, серверним API і Python-сервісом, а доступ до AI-функцій контролюється через систему ролей і платну підписку. Інтерфейс підтримує файлову навігацію, markdown-редактор і kanban-дошку в єдиному робочому просторі, працює на Windows, Android та iOS і адаптується до системної теми оформлення.

Практичне значення розробленої системи полягає в тому, що вона забезпечує користувачу повний контроль над власними даними: нотатки зберігаються локально, резервні копії шифруються до відправки на сервер, а мовні моделі запускаються безпосередньо на сервері без передачі тексту стороннім сервісам. Такий підхід є актуальним для користувачів, які цінують конфіденційність і незалежність від хмарних платформ.

Система є архітектурно цілісною і придатною до подальшого розширення – зокрема, додавання повнотекстового пошуку, підтримки спільного редагування, більш потужних мовних моделей або двосторонньої синхронізації між пристроями.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Notion Help Center. *Product Guides & Documentation*. URL: https://www.notion.so/help (дата звернення: 28.05.2026).

2. Asana Product Guide. *How to use Asana & Core Features*. URL: https://asana.com/guide (дата звернення: 28.05.2026).

3. Obsidian Help. *Obsidian Documentation & Knowledge Base*. URL: https://help.obsidian.md (дата звернення: 05.06.2026).

4. Marcotte E. *Responsive Web Design: 10th Anniversary Edition*. New York: A Book Apart, 2021. 158 p.

5. Trello Support. *Trello Guides and Documentation*. URL: https://support.atlassian.com/trello/ (дата звернення: 29.05.2026).

6. Layton M. C., Ostermiller S. J., Kynaston D. *Agile Project Management For Dummies*. 3rd ed. Hoboken: John Wiley & Sons, 2021. 432 p.

7. Sommerville I. *Software Engineering*. 11th ed. Boston: Pearson, 2021. 816 p.

8. Ruby on Rails Guides. *Rails 8.0 Documentation*. URL: https://guides.rubyonrails.org (дата звернення: 05.06.2026).

9. FastAPI Documentation. *FastAPI Framework Reference*. URL: https://fastapi.tiangolo.com (дата звернення: 01.06.2026).

10. Miles R., Hamilton K. *Learning UML 2.0: A Pragmatic Introduction to UML*. 2nd ed. Sebastopol: O'Reilly Media, 2022. 288 p.

11. Fowler M. *Patterns of Enterprise Application Architecture*. 2nd ed. Boston: Addison-Wesley Professional, 2023. 544 p.

12. Martin R. C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Boston: Prentice Hall, 2022. 432 p.

13. Spilka M. *Modern Authentication with JWT and Ruby on Rails*. Berkeley: Apress, 2023. 310 p.

14. React Native Team. *React Native Blueprints: Building Cross-Platform Mobile Apps*. Content Centric / O'Reilly, 2024. 380 p.

15. Kleppmann M. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. 2nd ed. Sebastopol: O'Reilly Media, 2024. 650 p.

16. Ferguson N., Schneier B., Kohno T. *Cryptography Engineering: Design Principles and Practical Applications.* 2nd ed. Indianapolis: John Wiley & Sons, 2025. 416 p.

ДОДАТОК А

ЛІСТИНГ КОДУ СЕРВЕРНОЇ ЧАСТИНИ

СХЕМА БАЗИ ДАНИХ

```
      ActiveRecord::Schema[8.1].define(version: 2026_05_18_000001) do
        # These are extensions that must be enabled in order to support this
database
        enable_extension "pg_catalog.plpgsql"

        # Custom types defined in this database.
        # Note that some types may not work with other database engines. Be careful
if changing database.
        create_enum "user_role", ["user", "ai_user", "admin"]

        create_table "jwt_denylists", force: :cascade do |t|
          t.datetime "exp", null: false
          t.string "jti", null: false
          t.index ["jti"], name: "index_jwt_denylists_on_jti"
        end

        create_table "motor_alert_locks", force: :cascade do |t|
          t.bigint "alert_id", null: false
          t.datetime "created_at", null: false
          t.string "lock_timestamp", null: false
          t.datetime "updated_at", null: false
          t.index          ["alert_id",          "lock_timestamp"],          name:
"index_motor_alert_locks_on_alert_id_and_lock_timestamp", unique: true
          t.index ["alert_id"], name: "index_motor_alert_locks_on_alert_id"
        end

        create_table "motor_alerts", force: :cascade do |t|
          t.bigint "author_id"
          t.string "author_type"
          t.datetime "created_at", null: false
          t.datetime "deleted_at"
          t.text "description"
          t.boolean "is_enabled", default: true, null: false
          t.string "name", null: false
          t.text "preferences", null: false
          t.bigint "query_id", null: false
          t.text "to_emails", null: false
          t.datetime "updated_at", null: false
          t.index ["name"], name: "motor_alerts_name_unique_index", unique: true,
where: "(deleted_at IS NULL)"
          t.index ["query_id"], name: "index_motor_alerts_on_query_id"
          t.index ["updated_at"], name: "index_motor_alerts_on_updated_at"
        end

        create_table "motor_api_configs", force: :cascade do |t|
          t.datetime "created_at", null: false
          t.text "credentials", null: false
          t.datetime "deleted_at"
          t.text "description"
          t.string "name", null: false
          t.text "preferences", null: false
          t.datetime "updated_at", null: false
          t.string "url", null: false
          t.index ["name"], name: "motor_api_configs_name_unique_index", unique:
true, where: "(deleted_at IS NULL)"
        end

        create_table "motor_audits", force: :cascade do |t|
          t.string "action"
```

```
    t.string "associated_id"
    t.string "associated_type"
    t.string "auditable_id"
    t.string "auditable_type"
    t.text "audited_changes"
    t.text "comment"
    t.datetime "created_at"
    t.string "remote_address"
    t.string "request_uuid"
    t.bigint "user_id"
    t.string "user_type"
    t.string "username"
    t.bigint "version", default: 0
    t.index ["associated_type", "associated_id"], name:
"motor_auditable_associated_index"
    t.index ["auditable_type", "auditable_id", "version"], name:
"motor_auditable_index"
    t.index ["created_at"], name: "index_motor_audits_on_created_at"
    t.index ["request_uuid"], name: "index_motor_audits_on_request_uuid"
    t.index ["user_id", "user_type"], name: "motor_auditable_user_index"
  end

  create_table "motor_configs", force: :cascade do |t|
    t.datetime "created_at", null: false
    t.string "key", null: false
    t.datetime "updated_at", null: false
    t.text "value", null: false
    t.index ["key"], name: "index_motor_configs_on_key", unique: true
    t.index ["updated_at"], name: "index_motor_configs_on_updated_at"
  end

  create_table "motor_dashboards", force: :cascade do |t|
    t.bigint "author_id"
    t.string "author_type"
    t.datetime "created_at", null: false
    t.datetime "deleted_at"
    t.text "description"
    t.text "preferences", null: false
    t.string "title", null: false
    t.datetime "updated_at", null: false
    t.index ["title"], name: "motor_dashboards_title_unique_index", unique:
true, where: "(deleted_at IS NULL)"
    t.index ["updated_at"], name: "index_motor_dashboards_on_updated_at"
  end

  create_table "motor_forms", force: :cascade do |t|
    t.string "api_config_name", null: false
    t.text "api_path", null: false
    t.bigint "author_id"
    t.string "author_type"
    t.datetime "created_at", null: false
    t.datetime "deleted_at"
    t.text "description"
    t.string "http_method", null: false
    t.string "name", null: false
    t.text "preferences", null: false
    t.datetime "updated_at", null: false
    t.index ["name"], name: "motor_forms_name_unique_index", unique: true,
where: "(deleted_at IS NULL)"
    t.index ["updated_at"], name: "index_motor_forms_on_updated_at"
  end

  create_table "motor_note_tag_tags", force: :cascade do |t|
    t.bigint "note_id", null: false
    t.bigint "tag_id", null: false
```

```
    t.index ["note_id", "tag_id"], name: "motor_note_tags_note_id_tag_id_index", unique: true
    t.index ["tag_id"], name: "index_motor_note_tag_tags_on_tag_id"
  end

  create_table "motor_note_tags", force: :cascade do |t|
    t.datetime "created_at", null: false
    t.string "name", null: false
    t.datetime "updated_at", null: false
    t.index ["name"], name: "motor_note_tags_name_unique_index", unique: true
  end

  create_table "motor_notes", force: :cascade do |t|
    t.bigint "author_id"
    t.string "author_type"
    t.text "body"
    t.datetime "created_at", null: false
    t.datetime "deleted_at"
    t.string "record_id", null: false
    t.string "record_type", null: false
    t.datetime "updated_at", null: false
    t.index ["author_id", "author_type"], name: "motor_notes_author_id_author_type_index"
    t.index ["record_id", "record_type"], name: "motor_notes_record_id_record_type_index"
  end

  create_table "motor_notifications", force: :cascade do |t|
    t.datetime "created_at", null: false
    t.text "description"
    t.bigint "recipient_id", null: false
    t.string "recipient_type", null: false
    t.string "record_id"
    t.string "record_type"
    t.string "status", null: false
    t.string "title", null: false
    t.datetime "updated_at", null: false
    t.index ["recipient_id", "recipient_type"], name: "motor_notifications_recipient_id_recipient_type_index"
    t.index ["record_id", "record_type"], name: "motor_notifications_record_id_record_type_index"
  end

  create_table "motor_queries", force: :cascade do |t|
    t.bigint "author_id"
    t.string "author_type"
    t.datetime "created_at", null: false
    t.datetime "deleted_at"
    t.text "description"
    t.string "name", null: false
    t.text "preferences", null: false
    t.text "sql_body", null: false
    t.datetime "updated_at", null: false
    t.index ["name"], name: "motor_queries_name_unique_index", unique: true, where: "(deleted_at IS NULL)"
    t.index ["updated_at"], name: "index_motor_queries_on_updated_at"
  end

  create_table "motor_reminders", force: :cascade do |t|
    t.bigint "author_id", null: false
    t.string "author_type", null: false
    t.datetime "created_at", null: false
    t.bigint "recipient_id", null: false
    t.string "recipient_type", null: false
    t.string "record_id"
```

```
    t.string "record_type"
    t.datetime "scheduled_at", null: false
    t.datetime "updated_at", null: false
    t.index ["author_id", "author_type"], name: "motor_reminders_author_id_author_type_index"
    t.index ["recipient_id", "recipient_type"], name: "motor_reminders_recipient_id_recipient_type_index"
    t.index ["record_id", "record_type"], name: "motor_reminders_record_id_record_type_index"
    t.index ["scheduled_at"], name: "index_motor_reminders_on_scheduled_at"
  end

  create_table "motor_resources", force: :cascade do |t|
    t.datetime "created_at", null: false
    t.string "name", null: false
    t.text "preferences", null: false
    t.datetime "updated_at", null: false
    t.index ["name"], name: "index_motor_resources_on_name", unique: true
    t.index ["updated_at"], name: "index_motor_resources_on_updated_at"
  end

  create_table "motor_taggable_tags", force: :cascade do |t|
    t.bigint "tag_id", null: false
    t.bigint "taggable_id", null: false
    t.string "taggable_type", null: false
    t.index ["tag_id"], name: "index_motor_taggable_tags_on_tag_id"
    t.index ["taggable_id", "taggable_type", "tag_id"], name: "motor_polymorphic_association_tag_index", unique: true
  end

  create_table "motor_tags", force: :cascade do |t|
    t.datetime "created_at", null: false
    t.string "name", null: false
    t.datetime "updated_at", null: false
    t.index ["name"], name: "motor_tags_name_unique_index", unique: true
  end

  create_table "user_backups", force: :cascade do |t|
    t.string "checksum", null: false
    t.datetime "created_at", null: false
    t.jsonb "data", default: {}, null: false
    t.integer "notes_count", default: 0, null: false
    t.datetime "updated_at", null: false
    t.bigint "user_id", null: false
    t.index ["user_id", "created_at"], name: "index_user_backups_on_user_id_and_created_at"
    t.index ["user_id"], name: "index_user_backups_on_user_id"
  end

  create_table "users", force: :cascade do |t|
    t.datetime "created_at", null: false
    t.string "email", default: "", null: false
    t.string "encrypted_password", default: "", null: false
    t.datetime "remember_created_at"
    t.datetime "reset_password_sent_at"
    t.string "reset_password_token"
    t.enum "role", default: "user", null: false, enum_type: "user_role"
    t.string "stripe_customer_id"
    t.datetime "updated_at", null: false
    t.index ["email"], name: "index_users_on_email", unique: true
    t.index ["reset_password_token"], name: "index_users_on_reset_password_token", unique: true
    t.index ["stripe_customer_id"], name: "index_users_on_stripe_customer_id", unique: true
  end
```

```
        add_foreign_key "motor_alert_locks", "motor_alerts", column: "alert_id"
        add_foreign_key "motor_alerts", "motor_queries", column: "query_id"
        add_foreign_key   "motor_note_tag_tags",   "motor_note_tags",   column:
"tag_id"
        add_foreign_key "motor_note_tag_tags", "motor_notes", column: "note_id"
        add_foreign_key "motor_taggable_tags", "motor_tags", column: "tag_id"
        add_foreign_key "user_backups", "users"
      end
```

КОНТРОЛЕР ДЛЯ ШІ-ГЕНЕРАЦІЇ

```
module Api

  module V1

    class AiController < ApplicationController

      before_action :authenticate_user!

      rescue_from AiNoteService::AiServiceError, with: :render_ai_error


      # POST /api/v1/ai/summarize

      def summarize

        authorize :ai, :summarize?


        text = params[:text].to_s.strip

        return render_error("text is required", :unprocessable_entity) if
text.empty?


        summary = AiNoteService.summarize(text)

        render json: { summary: summary }, status: :ok

      end


      # POST /api/v1/ai/generate

      def generate

        authorize :ai, :generate?


        prompt = params[:prompt].to_s.strip

        return render_error("prompt is required", :unprocessable_entity) if
prompt.empty?


        generated = AiNoteService.generate(

          prompt,

          max_new_tokens: params.fetch(:max_new_tokens, 100).to_i,

          temperature: params.fetch(:temperature, 0.7).to_f

        )

        render json: { generated_text: generated }, status: :ok
```

```
      end


      private


      def render_ai_error(error)
        render json: { error: error.message }, status: error.http_status
      end


      def render_error(message, status)
        render json: { error: message }, status: status
      end
    end
  end
end
```

МАРШРУТИЗАЦІЯ

```
      Rails.application.routes.draw do
        #    Define    your    application    routes    per    the    DSL    in
https://guides.rubyonrails.org/routing.html

        # Reveal health status on /up that returns 200 if the app boots with no
exceptions, otherwise 500.
        # Can be used by load balancers and uptime monitors to verify that the
app is live.
        get "up" => "rails/health#show", as: :rails_health_check

        # Mock Stripe checkout (development only)
        if Rails.env.development?
          get  "/mock_checkout", to: "mock_checkouts#show"
          post "/mock_checkout", to: "mock_checkouts#create"
        end

        # Admin web sign-in must be defined BEFORE the Motor::Admin mount so Rails
        # matches these routes first, bypassing the authenticate constraint.
        devise_scope :user do
          get        "/admin/sign_in",    to:  "admin/sessions#new",          as:
:new_admin_session
          post       "/admin/sign_in",     to:   "admin/sessions#create",     as:
:admin_session
          delete    "/admin/sign_out",    to:    "admin/sessions#destroy",    as:
:destroy_admin_session
        end

        # Admin panel – only accessible to users with the admin role
        authenticate :user, ->(user) { user.admin? } do
          mount Motor::Admin => "/admin"
        end

        # Authentication
        devise_for :users, path: "api/v1", path_names: {
          sign_in: "login",
          sign_out: "logout",
          registration: "signup"
        }, controllers: {
          sessions: "api/v1/sessions",
```

```
    registrations: "api/v1/registrations"
  }

  namespace :api do
    namespace :v1 do
      get  "current_user",           to: "current_user#show"
      post "ai/summarize",           to: "ai#summarize"
      post "ai/generate",            to: "ai#generate"
      post "subscriptions/checkout", to: "subscriptions#checkout"
      post "webhooks/stripe",        to: "webhooks#stripe"

      # Backups
      resources :backups, only: [ :index, :create, :show ] do
        collection do
          get :latest
        end
      end
    end
  end
end
```

ДОДАТОК Б

ЛІСТИНГ КОДУ ІНТЕРФЕЙСНОЇ ЧАСТИНИ

ДЕРЕВО ФАЙЛІВ

```
      import React, {useState,
useCallback, useEffect, useRef} from
'react';
      import {
        View,
        Text,
        TouchableOpacity,
        TextInput,
        ScrollView,
        StyleSheet,
      } from 'react-native';
      import {Note} from
'../types/Note';
      import {
        getAllNotes,
        getAllFolders,
        createFolder as
storageCreateFolder,
        deleteNote as
storageDeleteNote,
        deleteFolder as
storageDeleteFolder,
        saveNote,
      } from '../utils/fileStorage';
      import AppModal from
'./AppModal';
      import {useModal} from
'../utils/useModal';
      import {FileIcon, FolderIcon,
ChevronIcon, PlusIcon, DotsIcon} from
'./Icons';

      export type CtxItem = {label:
string; onPress: () => void; danger?:
boolean};

      type CreatingState = {type:
'file' | 'folder'; inFolder?:
string};

      interface FileTreeProps {
        selectedNoteId: string |
null;
        onSelectNote: (noteId:
string) => void;
        onNewNoteCreated: (noteId:
string) => void;
        onNoteDeleted: (noteId:
string) => void;
        onShowContextMenu: (x:
number, y: number, items: CtxItem[])
=> void;
        refreshKey: number;
        isDark: boolean;
        onToggleTheme: () => void;
        userEmail?: string;
        onGoToAuth?: () => void;
        onLogout?: () => void;
        onBackup?: () => void;
        backupStatus?: 'idle' |
'saving';
      }

      interface RowProps {
        rowId: string;
        hoveredId: string | null;
        setHoveredId: (id: string |
null) => void;
        onPress: () => void;
        ctxItems: CtxItem[];
        onShowContextMenu: (x:
number, y: number, items: CtxItem[])
=> void;
        style?: any;
        children: React.ReactNode;
        dimColor: string;
        alwaysShowMenu?: boolean;
      }

      const TreeRow:
React.FC<RowProps> = ({
        rowId,
        hoveredId,
        setHoveredId,
        onPress,
        ctxItems,
        onShowContextMenu,
        style,
        children,
        dimColor,
        alwaysShowMenu,
      }) => {
        const menuBtnRef =
useRef<View>(null);
        const hovered = hoveredId ===
rowId;
        const showMenu =
alwaysShowMenu || hovered;

        const openMenu =
useCallback(() => {

menuBtnRef.current?.measure((_fx,
_fy, _w, h, px, py) => {
            onShowContextMenu(px, py
+ h + 2, ctxItems);
          });
        }, [ctxItems,
onShowContextMenu]);

        return (
          <View
            style={style}
            {...({onMouseEnter: () =>
setHoveredId(rowId)} as any)}
            {...({onMouseLeave: () =>
setHoveredId(null)} as any)}>
```

```
            <TouchableOpacity
style={styles.rowMain}
onPress={onPress}
activeOpacity={0.8}>
              {children}
            </TouchableOpacity>
            {showMenu && (
              <TouchableOpacity
                ref={menuBtnRef as
any}

style={styles.menuBtn}
                onPress={openMenu}
                activeOpacity={0.6}>
                <DotsIcon
color={dimColor} size={18} />
              </TouchableOpacity>
            )}
          </View>
        );
      };

      const FileTree:
React.FC<FileTreeProps> = ({
        selectedNoteId,
        onSelectNote,
        onNewNoteCreated,
        onNoteDeleted,
        onShowContextMenu,
        refreshKey,
        isDark,
        onToggleTheme,
        userEmail,
        onGoToAuth,
        onLogout,
        onBackup,
        backupStatus,
      }) => {
        const [notes, setNotes] =
useState<Note[]>([]);
        const [folders, setFolders] =
useState<string[]>([]);
        const [expanded, setExpanded]
= useState<Set<string>>(new Set());
        const [creating, setCreating]
= useState<CreatingState |
null>(null);
        const [inputValue,
setInputValue] = useState('');
        const [hoveredId,
setHoveredId] = useState<string |
null>(null);
        const inputRef =
useRef<TextInput>(null);
        const submittingRef =
useRef(false);
        const headerMenuBtnRef =
useRef<View>(null);
        const {modal, showConfirm} =
useModal();

        const c = {
          bg: isDark ? '#18181B' :
'#F4F4F5',
          rowSelected: isDark ?
'#1E1B4B' : '#E0E7FF',
          rowHover: isDark ?
'#27272A' : '#EBEBEE',
          text: isDark ? '#E4E4E7' :
'#18181B',
          textSelected: isDark ?
'#A5B4FC' : '#4338CA',
          dim: isDark ? '#71717A' :
'#A1A1AA',
          border: isDark ? '#3F3F46'
: '#E4E4E7',
          accent: isDark ? '#818CF8'
: '#6366F1',
          danger: isDark ? '#F87171'
: '#EF4444',
          inputBg: isDark ? '#27272A'
: '#FFFFFF',
          header: isDark ? '#111113'
: '#FAFAFA',
          headerText: isDark ?
'#A1A1AA' : '#71717A',
          folderIcon: isDark ?
'#FCD34D' : '#D97706',
          fileIconColor: isDark ?
'#94A3B8' : '#94A3B8',
        };

        const load =
useCallback(async () => {
          const [allNotes,
allFolders] = await
Promise.all([getAllNotes(),
getAllFolders()]);
          setNotes(allNotes);
          setFolders(allFolders);
        }, []);

        useEffect(() => {
          load();
        }, [load, refreshKey]);

        const startCreating =
useCallback(
          (type: 'file' | 'folder',
inFolder?: string) => {
            if (inFolder !==
undefined && !expanded.has(inFolder))
{
              setExpanded(prev => new
Set([...prev, inFolder]));
            }
            setCreating({type,
inFolder});
            setInputValue('');
            setTimeout(() =>
inputRef.current?.focus(), 50);
          },
          [expanded],
        );

        const cancelCreating =
useCallback(() => {
          if (submittingRef.current)
{return;}
          setCreating(null);
          setInputValue('');
        }, []);
```

```
        const submitCreating =
useCallback(async () => {
          const name =
inputValue.trim();
          if (!name) {
            setCreating(null);
            setInputValue('');
            return;
          }
          submittingRef.current =
true;
          try {
            if (creating?.type ===
'folder') {
              await
storageCreateFolder(name);
              setExpanded(prev => new
Set([...prev, name]));
              await load();
            } else {
              const note = await
saveNote({title: name, content: '',
folder: creating?.inFolder});
              await load();

onNewNoteCreated(note.id);
            }
          } finally {
            submittingRef.current =
false;
            setCreating(null);
            setInputValue('');
          }
        }, [inputValue, creating,
load, onNewNoteCreated]);

        const handleDeleteNote =
useCallback(
          (note: Note) => {
            showConfirm('Delete
Note', `Delete "${note.title}"?`,
async () => {
              await
storageDeleteNote(note.filename);
              await load();
              onNoteDeleted(note.id);
            }, 'Delete');
          },
          [showConfirm, load,
onNoteDeleted],
        );

        const handleDeleteFolder =
useCallback(
          (name: string) => {
            const count =
notes.filter(n => n.folder ===
name).length;
            const msg = count > 0
              ? `Delete "${name}" and
move its ${count} note${count !== 1 ?
's' : ''} to root?`
              : `Delete folder
"${name}"?`;
            showConfirm('Delete
Folder', msg, async () => {
              await
storageDeleteFolder(name);
              await load();
            }, 'Delete');
          },
          [showConfirm, notes, load],
        );

        const toggleFolder =
useCallback((name: string) => {
          setExpanded(prev => {
            const next = new
Set(prev);
            if (next.has(name))
{next.delete(name);} else
{next.add(name);}
            return next;
          });
        }, []);

        const rootNotes =
notes.filter(n => !n.folder);
        const notesInFolder =
(folderName: string) =>
notes.filter(n => n.folder ===
folderName);

        const renderInlineInput =
(inFolder?: string) => {
          const depth = inFolder !==
undefined ? 1 : 0;
          return (
            <View
              key="__input__"
              style={[styles.row,
{paddingLeft: 8 + depth * 16,
backgroundColor: c.inputBg}]}>
              {creating?.type ===
'folder' ? (
                <FolderIcon
color={c.folderIcon} size={16} />
              ) : (
                <FileIcon
color={c.fileIconColor} size={16} />
              )}
              <TextInput
                ref={inputRef}

style={[styles.inlineInput, {color:
c.text, borderColor: c.accent}]}
                value={inputValue}

onChangeText={setInputValue}

onSubmitEditing={submitCreating}

onBlur={cancelCreating}
                blurOnSubmit={false}

placeholder={creating?.type ===
'folder' ? 'folder name…' : 'note
name…'}

placeholderTextColor={c.dim}
```

```
              autoFocus
            />
          </View>
        );
      };

      const renderFileRow = (note: Note, depth: number) => {
        const selected = note.id === selectedNoteId;
        const ctxItems: CtxItem[] = [
          {label: 'Delete', danger: true, onPress: () => handleDeleteNote(note)},
        ];
        let bg = 'transparent';
        if (selected) {bg = c.rowSelected;}
        else if (hoveredId === note.id) {bg = c.rowHover;}
        return (
          <TreeRow
            key={note.id}
            rowId={note.id}
            hoveredId={hoveredId}
            setHoveredId={setHoveredId}
            onPress={() => onSelectNote(note.id)}
            ctxItems={ctxItems}
            onShowContextMenu={onShowContextMenu}
            style={[styles.row, {paddingLeft: 8 + depth * 16, backgroundColor: bg}]}
            dimColor={c.dim}>
            <View style={styles.iconWrap}>
              <FileIcon color={selected ? c.textSelected : c.fileIconColor} size={16} />
            </View>
            <Text
              style={[styles.label, {color: selected ? c.textSelected : c.text}]}
              numberOfLines={1}>
              {note.title}
            </Text>
          </TreeRow>
        );
      };

      const renderFolderRow = (name: string) => {
        const isExpanded = expanded.has(name);
        const children = notesInFolder(name);
        const isCreatingHere = creating !== null && creating.type === 'file' && creating.inFolder === name;
        const folderId = `folder:${name}`;
        const ctxItems: CtxItem[] = [
          {label: 'New File', onPress: () => startCreating('file', name)},
          {label: 'Delete Folder', danger: true, onPress: () => handleDeleteFolder(name)},
        ];
        const bg = hoveredId === folderId ? c.rowHover : 'transparent';
        return (
          <View key={name}>
            <TreeRow
              rowId={folderId}
              hoveredId={hoveredId}
              setHoveredId={setHoveredId}
              onPress={() => toggleFolder(name)}
              ctxItems={ctxItems}
              onShowContextMenu={onShowContextMenu}
              style={[styles.row, {paddingLeft: 4, backgroundColor: bg}]}
              dimColor={c.dim}
              alwaysShowMenu>
              <View style={styles.chevronWrap}>
                <ChevronIcon color={c.dim} size={16} down={isExpanded} />
              </View>
              <View style={styles.iconWrap}>
                <FolderIcon color={c.folderIcon} size={16} open={isExpanded} />
              </View>
              <Text
                style={[styles.label, {color: c.text, fontWeight: '600'}]}
                numberOfLines={1}>
                {name}
              </Text>
            </TreeRow>
            {isExpanded && (
              <>
                {children.map(note => renderFileRow(note, 1))}
                {isCreatingHere && renderInlineInput(name)}
              </>
            )}
          </View>
        );
      };

      const openHeaderMenu = useCallback(() => {
        headerMenuBtnRef.current?.measure((_fx, _fy, _w, h, px, py) => {
```

```
            onShowContextMenu(px, py
+ h + 2, [
              {label: 'New File',
onPress: () =>
startCreating('file')},
              {label: 'New Folder',
onPress: () =>
startCreating('folder')},
            ]);
          });
        }, [onShowContextMenu,
startCreating]);

        return (
          <View
style={[styles.container,
{backgroundColor: c.bg,
borderRightColor: c.border}]}>
            <View
style={[styles.header,
{backgroundColor: c.header,
borderBottomColor: c.border}]}>
              <Text
style={[styles.headerTitle, {color:
c.headerText}]}>Explorer</Text>
              <View
style={styles.headerActions}>
                <TouchableOpacity

onPress={onToggleTheme}

style={[styles.themeBtn,
{borderColor: c.border,
backgroundColor: c.inputBg}]}

activeOpacity={0.7}>
                  <Text
style={[styles.themeBtnText, {color:
c.dim}]}>
                    {isDark ? 'Light'
: 'Dark'}
                  </Text>
                </TouchableOpacity>
                <TouchableOpacity
                  onPress={() =>
startCreating('file')}

style={styles.headerBtn}

activeOpacity={0.6}>
                  <FileIcon
color={c.dim} size={16} />
                </TouchableOpacity>
                <TouchableOpacity

ref={headerMenuBtnRef as any}

onPress={openHeaderMenu}

style={styles.headerBtn}

activeOpacity={0.6}>
                  <PlusIcon
color={c.dim} size={16} />
                </TouchableOpacity>
              </View>
            </View>

            <ScrollView
style={styles.scroll}
showsVerticalScrollIndicator={false}>
              {rootNotes.map(note =>
renderFileRow(note, 0))}
              {creating !== null &&
creating.inFolder === undefined &&
renderInlineInput(undefined)}
              {folders.map(name =>
renderFolderRow(name))}
            </ScrollView>

            {(userEmail ||
onGoToAuth) && (
              <View
style={[styles.userBar,
{borderTopColor: c.border}]}>
                {userEmail ? (
                  <>
                    <Text
style={[styles.userEmail, {color:
c.dim}]} numberOfLines={1}>
                      {userEmail}
                    </Text>
                    {onBackup && (

<TouchableOpacity

onPress={backupStatus === 'idle' ?
onBackup : undefined}

style={styles.barBtn}

activeOpacity={0.65}>
                        <Text
style={[styles.barBtnText, {color:
c.accent}]}>

{backupStatus === 'saving' ? '...' :
'Sync'}
                        </Text>

</TouchableOpacity>
                    )}
                    {onLogout && (

<TouchableOpacity

onPress={onLogout}

style={styles.barBtn}

activeOpacity={0.65}>
                        <Text
style={[styles.barBtnText, {color:
c.dim}]}>Out</Text>

</TouchableOpacity>
                    )}
                  </>
                ) : onGoToAuth ? (
                  <TouchableOpacity
onPress={onGoToAuth}>
```

```
                    <Text
style={[styles.userAction, {color:
c.accent}]}>Sign In</Text>
                  </TouchableOpacity>
                ) : null}
              </View>
            )}

            <AppModal {...modal}
isDark={isDark} />
          </View>
        );
      };

      const styles =
StyleSheet.create({
        container: {
          width: 240,
          borderRightWidth: 1,
          flexDirection: 'column',
        },
        header: {
          paddingHorizontal: 12,
          paddingVertical: 9,
          borderBottomWidth: 1,
          flexDirection: 'row',
          alignItems: 'center',
        },
        headerTitle: {
          flex: 1,
          fontSize: 12,
          fontWeight: '600',
          letterSpacing: 0.3,
        },
        headerActions: {
          flexDirection: 'row',
          alignItems: 'center',
          gap: 4,
        },
        headerBtn: {
          width: 24,
          height: 24,
          borderRadius: 5,
          justifyContent: 'center',
          alignItems: 'center',
        },
        themeBtn: {
          borderWidth: 1,
          borderRadius: 6,
          paddingHorizontal: 8,
          paddingVertical: 4,
        },
        themeBtnText: {
          fontSize: 11,
          fontWeight: '600',
        },
        scroll: {
          flex: 1,
          paddingTop: 4,
        },
        row: {
          flexDirection: 'row',
          alignItems: 'center',
          height: 30,
          paddingRight: 4,
          borderRadius: 5,
          marginHorizontal: 4,
          marginBottom: 1,
        },
        rowMain: {
          flex: 1,
          flexDirection: 'row',
          alignItems: 'center',
          height: '100%',
        },
        chevronWrap: {
          width: 18,
          height: 18,
          justifyContent: 'center',
          alignItems: 'center',
          marginLeft: 4,
        },
        iconWrap: {
          width: 18,
          height: 18,
          justifyContent: 'center',
          alignItems: 'center',
          marginRight: 6,
        },
        label: {
          flex: 1,
          fontSize: 13,
        },
        menuBtn: {
          width: 24,
          height: 24,
          borderRadius: 4,
          justifyContent: 'center',
          alignItems: 'center',
        },
        inlineInput: {
          flex: 1,
          fontSize: 13,
          borderBottomWidth: 1,
          paddingVertical: 0,
          paddingHorizontal: 4,
          height: 22,
          marginRight: 6,
          marginLeft: 6,
          borderRadius: 3,
        },
        userBar: {
          flexDirection: 'row',
          alignItems: 'center',
          justifyContent: 'space-
between',
          paddingHorizontal: 12,
          paddingVertical: 10,
          borderTopWidth: 1,
        },
        userEmail: {
          fontSize: 11,
          flex: 1,
          marginRight: 6,
        },
        userAction: {
          fontSize: 12,
          fontWeight: '600',
        },
        barBtn: {
          paddingHorizontal: 6,
          paddingVertical: 4,
```

```
  },
  barBtnText: {
    fontSize: 11,
    fontWeight: '600',
  },
});

export default FileTree;
```

ЕКРАН ВХОДУ В АККАУНТ

```
import React, {useState} from 'react';
import {
  View,
  Text,
  TextInput,
  TouchableOpacity,
  StyleSheet,
  ActivityIndicator,
  KeyboardAvoidingView,
  Platform,
  ScrollView,
} from 'react-native';
import {useModal} from '../utils/useModal';
import AppModal from './AppModal';
import {login} from '../utils/api';
import {saveAuthToken, saveAuthUser, AuthUser} from '../utils/authStorage';
import {deriveAndSaveKey} from '../utils/cryptoStorage';

interface LoginScreenProps {
  onLoginSuccess: (user: AuthUser) => void;
  onGoToRegister: () => void;
  onSkip?: () => void;
  isDark: boolean;
}

const LoginScreen: React.FC<LoginScreenProps> = ({
  onLoginSuccess,
  onGoToRegister,
  onSkip,
  isDark,
}) => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [loading, setLoading] = useState(false);
  const {modal, showAlert} = useModal();

  const colors = {
    bg: isDark ? '#09090B' : '#F4F4F5',
    card: isDark ? '#18181B' : '#FFFFFF',
    text: isDark ? '#F4F4F5' : '#18181B',
    textSecondary: isDark ? '#A1A1AA' : '#71717A',
    border: isDark ? '#3F3F46' : '#E4E4E7',
    accent: isDark ? '#818CF8' : '#6366F1',
    inputBg: isDark ? '#27272A' : '#FAFAFA',
  };

  const handleLogin = async () => {
    if (!email.trim() || !password.trim()) {
      showAlert('Error', 'Please enter both email and password.');
      return;
    }

    setLoading(true);
    try {
      const {user, token} = await login(email.trim(), password);
      if (token) {
        await saveAuthToken(token);
      }
      await saveAuthUser(user);
      // Derive and persist the encryption key from credentials so backups
      // can be encrypted/decrypted on any device after login.
      await deriveAndSaveKey(password, email.trim());
      onLoginSuccess(user);
    } catch (error: any) {
      showAlert('Login Failed', error.message || 'Invalid email or password.');
    } finally {
      setLoading(false);
    }
  };

  return (
    <KeyboardAvoidingView
      style={[styles.container, {backgroundColor: colors.bg}]}
      behavior={Platform.OS === 'ios' ? 'padding' : undefined}>
      <ScrollView
        contentContainerStyle={styles.scrollContent}
        keyboardShouldPersistTaps="handled">
```

```
            <View
style={styles.header}>
              <Text
style={[styles.title, {color:
colors.text}]}>Notetaker</Text>
              <Text
style={[styles.subtitle, {color:
colors.textSecondary}]}>
                Sign in to your
account
              </Text>
            </View>

            <View
style={[styles.form,
{backgroundColor: colors.card,
borderColor: colors.border}]}>
              <Text
style={[styles.label, {color:
colors.text}]}>Email</Text>
              <TextInput

style={[styles.input, {color:
colors.text, backgroundColor:
colors.inputBg, borderColor:
colors.border}]}

placeholder="you@example.com"

placeholderTextColor={colors.textSeco
ndary}
                value={email}

onChangeText={setEmail}

autoCapitalize="none"

keyboardType="email-address"
                autoCorrect={false}
              />

              <Text
style={[styles.label, {color:
colors.text}]}>Password</Text>
              <TextInput

style={[styles.input, {color:
colors.text, backgroundColor:
colors.inputBg, borderColor:
colors.border}]}
                placeholder="Your
password"

placeholderTextColor={colors.textSeco
ndary}
                value={password}

onChangeText={setPassword}
                secureTextEntry
              />

              <TouchableOpacity

style={[styles.button,
{backgroundColor: colors.accent}]}
onPress={handleLogin}
                disabled={loading}>
                {loading ? (

<ActivityIndicator color="#fff" />
                ) : (
                  <Text
style={styles.buttonText}>Sign
In</Text>
                )}
              </TouchableOpacity>
            </View>

            <TouchableOpacity
onPress={onGoToRegister}
style={styles.switchLink}>
              <Text
style={[styles.switchText, {color:
colors.textSecondary}]}>
                Don't have an
account?{' '}
                <Text
style={{color: colors.accent,
fontWeight: '700'}}>Sign Up</Text>
              </Text>
            </TouchableOpacity>

            {onSkip && (
              <TouchableOpacity
onPress={onSkip}
style={styles.switchLink}>
                <Text
style={[styles.switchText, {color:
colors.textSecondary}]}>
                  Skip for now
                </Text>
              </TouchableOpacity>
            )}
          </ScrollView>
          <AppModal {...modal}
isDark={isDark} />
        </KeyboardAvoidingView>
      );
    };

    const styles =
StyleSheet.create({
      container: {
        flex: 1,
      },
      scrollContent: {
        flexGrow: 1,
        justifyContent: 'center',
        padding: 28,
        maxWidth: 420,
        alignSelf: 'center',
        width: '100%',
      },
      header: {
        alignItems: 'center',
        marginBottom: 28,
      },
      title: {
        fontSize: 28,
        fontWeight: '700',
```

```
    marginBottom: 6,
    letterSpacing: -0.5,
  },
  subtitle: {
    fontSize: 14,
  },
  form: {
    padding: 22,
    borderRadius: 14,
    borderWidth: 1,
  },
  label: {
    fontSize: 13,
    fontWeight: '600',
    marginBottom: 5,
    marginTop: 14,
  },
  input: {
    fontSize: 14,
    paddingVertical: 10,
    paddingHorizontal: 12,
    borderRadius: 8,
    borderWidth: 1,
  },
  button: {
    marginTop: 22,
    paddingVertical: 12,
    borderRadius: 9,
    alignItems: 'center',
  },
  buttonText: {
    color: '#ffffff',
    fontSize: 15,
    fontWeight: '600',
  },
  switchLink: {
    marginTop: 18,
    alignItems: 'center',
  },
  switchText: {
    fontSize: 13,
  },
});

export default LoginScreen;
```

Бібліографічна довідка

Тема роботи: Розроблення інформаційної системи управління завданнями з інтелектуальним модулем генерації та аналізу.

Обсяг бакалаврської роботи 64 сторінок.

Перелік графічного матеріалу:

1. КРБ.ІСТ - 15.00.00.000 С1. Варіанти використання інформаційної системи управління завданнями.
2. КРБ.ІСТ - 15.00.00.000 С1. Діаграма компонентів інформаційної системи управління завданнями.
3. КРБ.ІСТ - 15.00.00.000 С1. Діаграма бази даних інформаційної системи управління завданнями.
4. КРБ.ІСТ - 15.00.00.000 С1. Діаграма використання інформаційної системи обліку завдань.
5. КРБ.ІСТ - 15.00.00.001. Програмні вікна клієнтського додатку інформаційної системи управління завданнями (2 шт.).

Дата закінчення БР

Студент Максим РУДИК