Міністерство освіти і науки України
Дрогобицький державний педагогічний університет імені Івана Франка
Кафедра фізики та інформаційних систем

«До захисту допускаю»
завідувач кафедри фізики
та інформаційних систем,
кандидат фіз-мат. наук, доцент
_____ В. Б. Гольський
« ___ » _____ 2026 р.

Розроблення персонального нотатника на основі штучного інтелекту

Спеціальність 122 Комп’ютерні науки

Випускова робота
на здобуття кваліфікації – бакалавр з комп’ютерних наук

Автор роботи: **Коваленко Назар Ігорович** _____
підпис

Науковий керівник: **канд. пед. наук, доцент**
Гарбич-Мошора Ольга Романівна _____
підпис

Дрогобич 2026

Дрогобицький державний педагогічний університет
імені Івана Франка

Зав. кафедрою

_____ _____
(підпис) (дата)

Завдання

на підготовку кваліфікаційної бакалаврської роботи

1. Тема: **"Розроблення персонального нотатника на основі штучного інтелекту"**
2. Керівник *канд. пед. наук, доцент Гарбич-Мошора О. Р.*
3. Студент *Коваленко Назар Ігорович*
(прізвище, ім'я, по батькові)
4. Перелік питань, що підлягають висвітленню у кваліфікаційній роботі

1. Аналіз предметної області.
2.Аналіз наявних аналогів. Опис технічного завдання для проєктування та програмної реалізації додатку.
3. Вимоги, технології та засоби розробки додатку.
4. Проєктування додатку.
5. Програмна реалізація додатку.

5. Список рекомендованої літератури

1.Документація Google Gemini API - [Електронний ресурс] // Режим доступу до ресурсу: https://ai.google.dev/gemini-api/docs.
2.Документація JavaScript - [Електронний ресурс] // Режим доступу до ресурсу: https://uk.javascript.info.
3.Документація Firebase - [Електронний ресурс] // Режим доступу до ресурсу: https://firebase.google.com/docs.

6. Етапи підготовки роботи

№	Назва етапу	Термін виконання	Термін звіту перед керівником, кафедрою
1.	Аналіз предметної області	Вересень 2025	
2.	Аналіз наявних аналогів. Опис технічного завдання для проєктування та програмної реалізації додатку	Вересень-жовтень 2025	
3	Вимоги, технології та засоби розробки додатку	Жовтень 2025	
4.	Проєктування додатку	Жовтень-грудень 2025	
5.	Програмна реалізація додатку	Листопад-травень 2025-2026	

7. Дата видачі завдання _____
8. Термін подачі роботи керівнику_____
9. З вимогами до виконання кваліфікаційної роботи і завданням ознайомлений _____
(підпис студента)
10. Керівник _____
(підпис)

Захист кваліфікаційної (бакалаврської) роботи

"Розроблення персонального нотатника на основі штучного інтелекту"

Оцінка за стобальною шкалою: _____

Оцінка за національною чотирибальною шкалою: _____

Коротка мотивація захисту: _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____ Голова ЕК _____ _____

Дата *підпис* *прізвище, ім'я*

Секретар ЕК _____ _____

підпис *прізвище, ім'я*

Анотація

Коваленко Н. В. Розроблення персонального нотатника на основі штучного інтелекту. Бакалаврська робота, Дрогобицький державний педагогічний університет імені Івана Франка, Дрогобич 2026.

Робота присвячена розробці вебдодатку для персональних нотаток із вбудованим AI-асистентом. Метою проєкту є створення системи, яка дозволяє не лише зберігати нотатки, але й автоматично обробляти їх за допомогою моделі штучного інтелекту.

Додаток реалізовано як вебсистему з використанням **React** (клієнтська частина) та **Node.js + Express** (серверна частина). Для інтелектуальної обробки тексту інтегровано **Google Gemini API**. Забезпечено повний CRUD-цикл, пошук з підсвіткою, систему тегів, темну тему, закріплення нотаток, автоматичне збереження, а також хмарне зберігання через **Firebase** з автентифікацією через Google.

AI-асистент виконує:

- автоматичне підсумовування тексту;
- визначення категорії (Покупки, Зустріч, Ідеї, Робота, Навчання, Важливе, Особисте, Інше);
- оптимізацію тексту (граматика, стиль);
- автоматичне генерування тегів.

Архітектура передбачає гібридний режим: при наявності API-ключа використовується реальна модель, інакше – локальна імітація на основі NLP-аналізу.

Ключові слова: персональні нотатки, штучний інтелект, Google Gemini, React, Node.js, Firebase, AI-асистент, автоматичне підсумовування, категоризація тексту.

Abstracts.

Kovalenko, N. V. Development of a Personal Note-Taking Application Based on Artificial Intelligence. Bachelor's Thesis, Ivan Franko State Pedagogical University of Drohobych, Drohobych 2026.

This thesis focuses on the development of a web application for personal note-taking with a built-in AI assistant. The goal of the project is to create a system that not only allows users to save notes but also automatically processes them using an artificial intelligence model.

The application is implemented as a web system using React (client-side) and Node.js + Express (server-side). The Google Gemini API is integrated for intelligent text processing. The system provides a full CRUD cycle, highlighted search, a tagging system, a dark theme, note pinning, auto-save, and cloud storage via Firebase with Google authentication.

The AI assistant performs:

• automatic text summarization;

• category identification (Shopping, Meeting, Ideas, Work, Study, Important, Personal, Other);

• text optimization (grammar, style);

• automatic tag generation.

The architecture supports a hybrid mode: if an API key is available, the actual model is used; otherwise, a local simulation based on NLP analysis is employed.

Keywords: personal notes, artificial intelligence, Google Gemini, React, Node.js, Firebase, AI assistant, automatic summarization, text categorization.

ЗМІСТ

ВСТУП

Актуальність теми. В умовах стрімкого розвитку інформаційних технологій та зростання обсягів даних, з якими щоденно працює сучасна людина, питання ефективного управління особистою інформацією набуває особливої актуальності.

Традиційні застосунки для ведення нотаток забезпечують базові функції збереження тексту, проте не вирішують проблему інформаційного перевантаження — користувач змушений самостійно структурувати, класифікувати та аналізувати накопичені записи.

Інтеграція штучного інтелекту в інструменти особистої продуктивності [7] відкриває принципово нові можливості: автоматизацію рутинних задач обробки тексту, інтелектуальну категоризацію та генерацію метаданих без участі користувача.

Незважаючи на наявність комерційних рішень у цій сфері (Notion, Evernote, Google Keep), більшість із них або не мають вбудованого AI, або надають його лише в платних тарифах [3, 6], що обмежує доступність для широкої аудиторії.

Розробка власного застосунку з відкритою AI-інтеграцією на базі сучасного стеку технологій є актуальним науково-практичним завданням.

Об'єкт дослідження: процеси проєктування та розробки вебзастосунку для управління персональними нотатками з інтеграцією штучного інтелекту.

Предмет дослідження: методи та засоби реалізації інтелектуальних функцій обробки тексту на основі великих мовних моделей у контексті застосунків для ведення нотаток.

Мета проєкту: спроєктувати та розробити вебзастосунок для ведення персональних нотаток із хмарним зберіганням даних та інтегрованим AI-асистентом на базі Google Gemini 2.5 Flash.

Для досягнення поставленої мети визначено такі завдання:

- провести аналіз існуючих застосунків для ведення нотаток та виявити їх переваги й недоліки;

- обґрунтувати вибір технологічного стеку для реалізації клієнтської, серверної та хмарної частин застосунку;
- спроєктувати архітектуру системи з урахуванням вимог до безпеки, масштабованості та реактивності інтерфейсу;
- реалізувати клієнтську частину на React із підтримкою темної теми, пошуку, фільтрації та автозбереження;
- розробити серверний рівень на Node.js + Express для безпечного проксіювання запитів до Gemini API;
- інтегрувати Firebase Firestore та Firebase Authentication для хмарного зберігання даних і автентифікації через Google;
- реалізувати чотири AI-функції: підсумовування, категоризацію, оптимізацію тексту та генерування тегів;
- провести тестування застосунку та оцінити якість роботи AI-функцій.

Структура роботи: робота складається з вступу, чотирьох розділів, висновків та списку використаної літератури.

Результати роботи доповідались на XIII Міжнародної науково-практичної конференції студентів та викладачів факультету фізики, математики, економіки та інноваційних технологій “Актуальні проблеми сучасної науки” (м. Дрогобич, 24-25 квітня 2026 р.)

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Призначення застосунку для управління персональними нотатками.

В сучасному інформаційному середовищі персональні нотатки є одним із ключових інструментів організації знань, ідей та завдань. Застосунки для нотаток дозволяють користувачам фіксувати думки в будь-який момент, структурувати інформацію за темами та швидко знаходити потрібні записи. Однак більшість існуючих рішень є пасивними сховищами тексту — вони зберігають інформацію, але не допомагають її осмислити.

Інтеграція штучного інтелекту в застосунки для нотаток відкриває якісно новий рівень взаємодії з контентом. AI-асистент здатен автоматично визначати суть нотатки, пропонувати категорію, генерувати теги та покращувати стилістику тексту — тобто виконувати роботу, яка традиційно потребує свідомих зусиль користувача. Це особливо актуально в умовах зростання інформаційного навантаження на сучасну людину.

Розроблюваний застосунок орієнтований на користувачів, які прагнуть ефективно організовувати особисті записи без зайвих витрат часу на ручну структуризацію. Хмарне зберігання через Firebase Firestore забезпечує доступність нотаток [5] із будь-якого пристрою, а автентифікація через Google Sign-In гарантує безпеку персональних даних.

1.2. Огляд наявних аналогів

Перед розробкою власного рішення було проведено аналіз чотирьох найпоширеніших застосунків для нотаток з метою виявлення їх переваг, недоліків та незайнятих ніш.

1.2.1. Google Keep

Google Keep — це безкоштовний хмарний застосунок від Google, орієнтований на швидке створення коротких нотаток, списків та нагадувань.

Інтерфейс є мінімалістичним, а синхронізація з акаунтом Google відбувається миттєво.

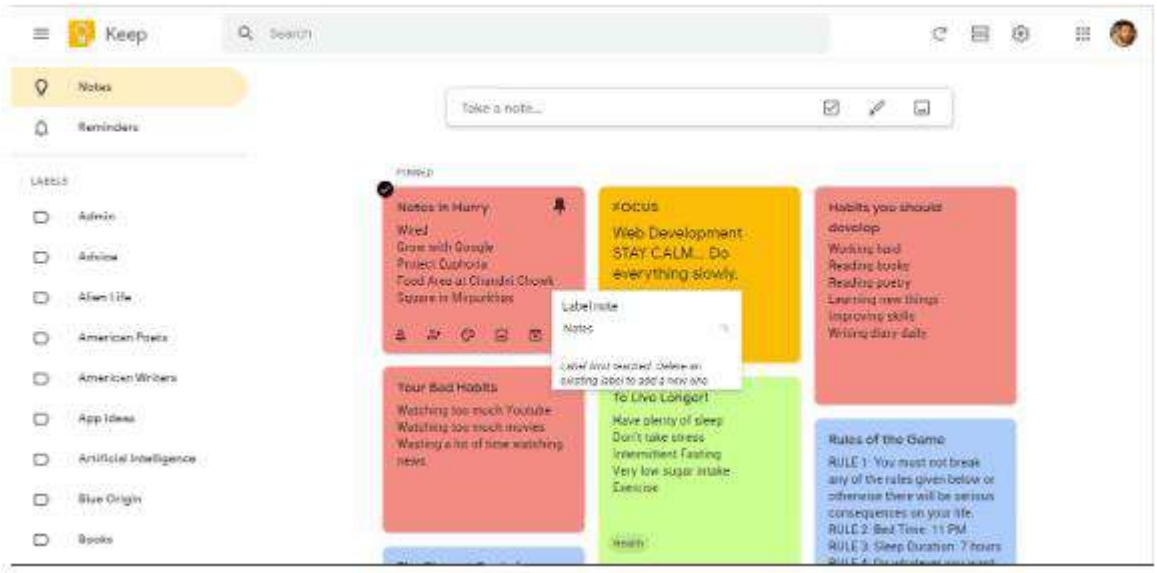

Рис. 1.1 «Інтерфейс Google Keep»

Переваги: простота використання, швидкий доступ, інтеграція з екосистемою Google, підтримка голосових нотаток та зображень.

Недоліки: відсутність будь-яких AI-функцій для обробки тексту, обмежені можливості структурування (немає вкладених структур, слабка система тегів), інтерфейс не підходить для роботи з великими обсягами тексту.

1.2.2. Notion

Notion — це потужна платформа для управління знаннями та проєктами, яка поєднує нотатки, бази даних, таблиці та канбан-дошки в єдиному просторі. Має вбудований AI-асистент (Notion AI).

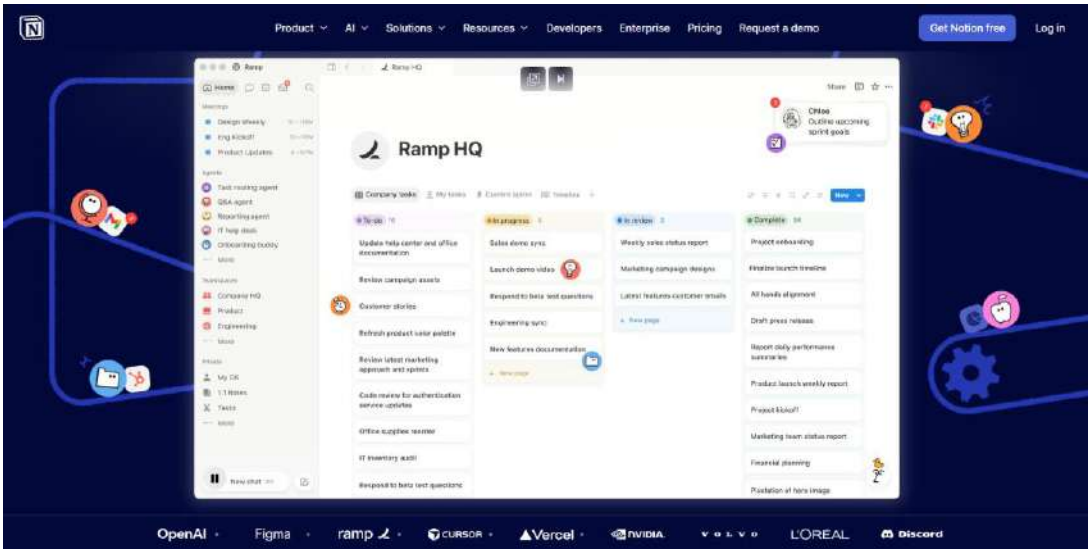

Рис. 1.2 «Головна сторінка Notion»

Переваги: гнучка структура сторінок і баз даних, Notion AI для генерації та редагування тексту, широкі можливості для командної роботи.

Недоліки: Notion AI є платною надбудовою, інтерфейс перевантажений для персонального використання, висока складність налаштування, повільна робота при великій кількості сторінок.

1.2.3. Evernote

Evernote — один із найстаріших і найвідоміших застосунків для нотаток, що підтримує текст, зображення, аудіо та веб-кліпи. Орієнтований на професійних користувачів.

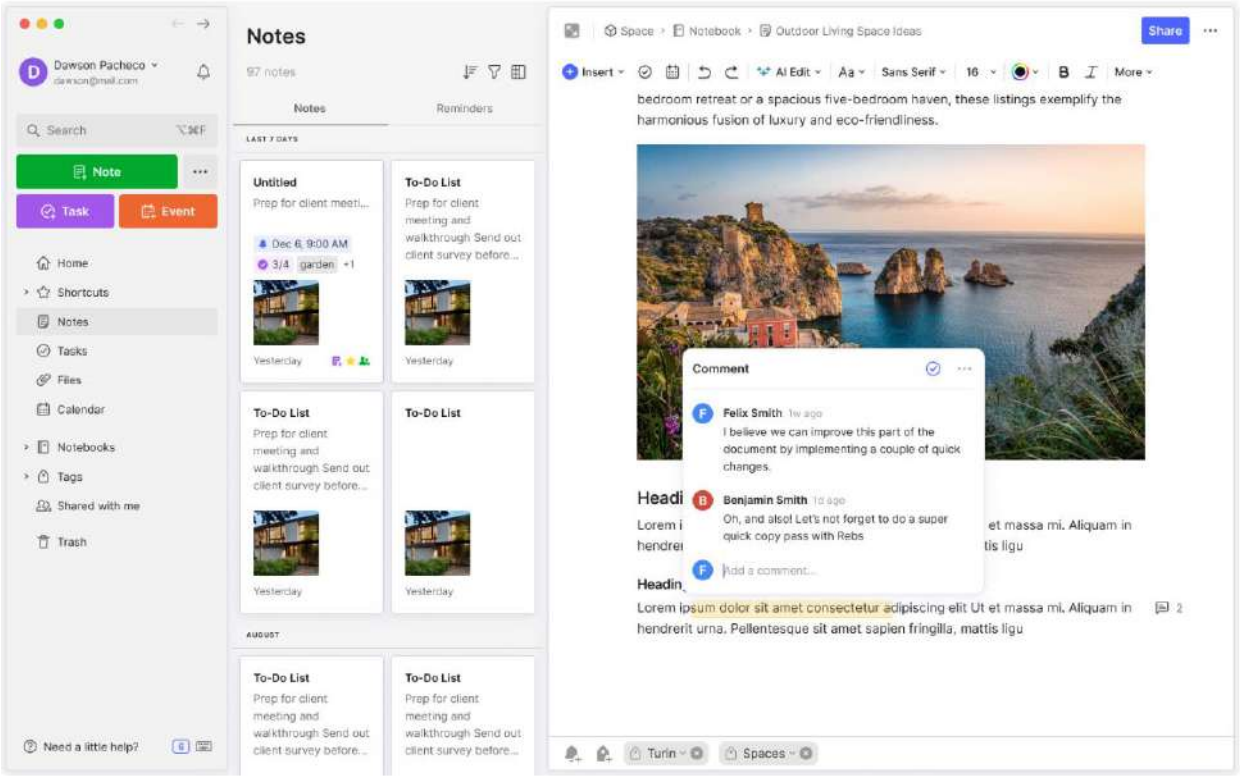

Рис. 1.3 «Інтерфейс Evernote»

Переваги: потужний пошук (зокрема по рукописному тексту та зображеннях), підтримка офлайн-режиму, широкі можливості організації.

Недоліки: більшість функцій доступні лише за підпискою, AI-можливості обмежені та не інтегровані глибоко в робочий процес, інтерфейс застарів у порівнянні з сучасними стандартами дизайну.

1.2.4. Roam Research

Roam Research — інструмент для нелінійного мислення, побудований на концепції двонаправлених посилань між нотатками. Орієнтований на дослідників та людей, що практикують метод «другого мозку».

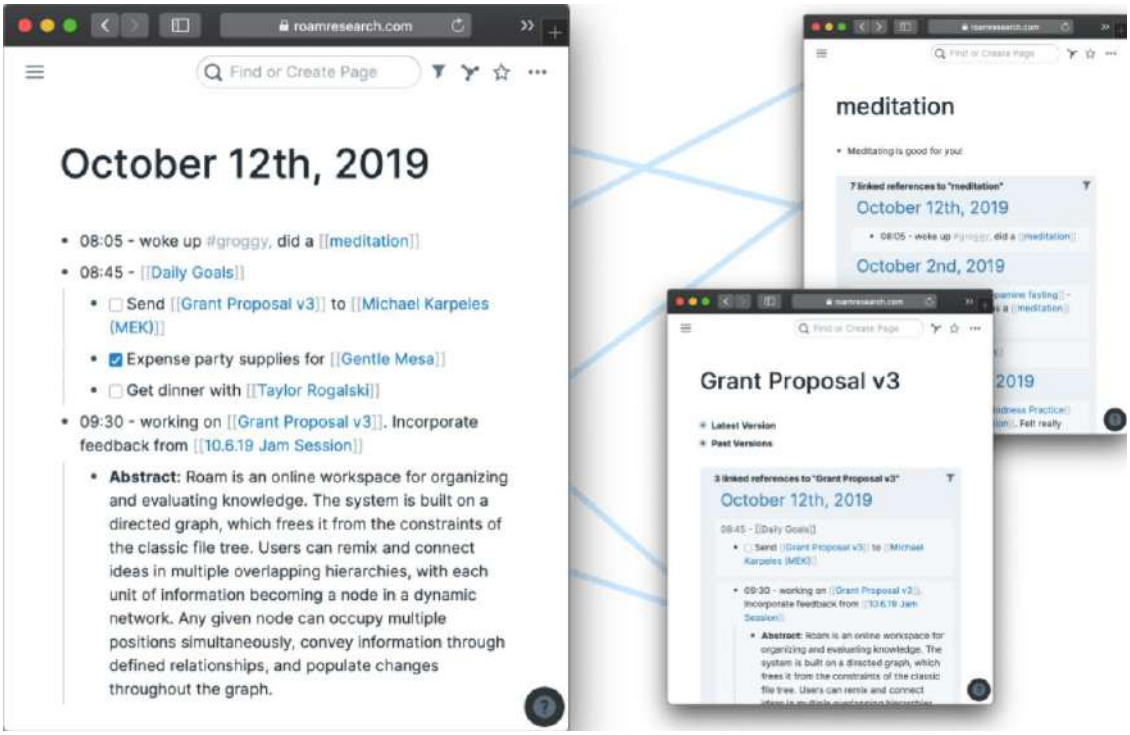

Рис. 1.4 «Головна сторінка Roam Research»

Переваги: унікальна система зв'язків між нотатками, граф знань, підходить для складних інтелектуальних задач.

Недоліки: висока вартість підписки, крута крива навчання, відсутність вбудованих AI-функцій обробки тексту, інтерфейс складний для звичайного користувача.

Таблиця 1.1 — Порівняльна таблиця аналогів

Критерій	Google Keep	Notion	Evernote	Roam Research	Проєкт (розробка)
Безкоштовний доступ	✓	✓ (обмежено)	✓ (обмежено)	–	✓
Хмарна синхронізація	✓	✓	✓	✓	✓
AI-підсумовування	–	(платно)	–	–	✓
AI-категоризація	–	–	–	–	✓
AI-оптимізація тексту	–	(платно)	–	–	✓

Автогенерація тегів	–	–	–	–	✓
Темна тема	✓	✓	✓	✓	✓
Відкрита реалізація	–	–	–	–	✓

Проведений аналіз свідчить, що жоден із розглянутих застосунків не поєднує в собі повністю безкоштовний доступ, глибоку інтеграцію AI-функцій обробки тексту (підсумовування, категоризація, теги, оптимізація) та сучасний мінімалістичний інтерфейс в єдиному рішенні. Саме це визначає актуальність та унікальність розроблюваного застосунку.

2. ВИМОГИ, ТЕХНОЛОГІЇ ТА ЗАСОБИ РОЗРОБКИ

2.1. Вимоги до застосунку

Формування вимог є одним із ключових етапів проєктування програмного забезпечення, оскільки саме вони визначають функціональні можливості системи та критерії її ефективності. Для розроблюваного застосунку персональних нотаток із підтримкою технологій штучного інтелекту було визначено комплекс функціональних і нефункціональних вимог.

Функціональні вимоги описують основні можливості системи та перелік операцій, які можуть виконувати користувачі під час роботи із застосунком. До них належать керування нотатками, автентифікація користувачів, пошук і фільтрація даних, автоматичне збереження змін, голосове введення тексту та використання інтелектуальних функцій обробки інформації.

Нефункціональні вимоги визначають характеристики якості програмного продукту, зокрема продуктивність, безпеку, надійність, адаптивність інтерфейсу та зручність використання. Особлива увага приділяється забезпеченню захисту даних користувачів, стабільності роботи системи в умовах тимчасової недоступності зовнішніх сервісів, а також можливості використання застосунку на різних типах пристроїв.

Сформований перелік вимог став основою для проєктування архітектури системи та вибору технологічних засобів її реалізації.

2.1.1. Функціональні вимоги

Автентифікація та управління обліковим записом. Система забезпечує вхід користувача через Google Sign-In з використанням Firebase Authentication [5]. Після авторизації додаток отримує не лише email та UID, але й аватар та ім'я з профілю Google. У профілі користувача відображається дата реєстрації (береться з user.metadata.creationTime) та динамічна статистика: кількість нотаток, унікальних тегів і загальна кількість слів у всіх записах. Статистика перераховується автоматично при кожній зміні даних.

Базові операції з нотатками. Підтримується повний CRUD-цикл. Усі нотатки зберігаються у Firestore за шляхом users/{userId}/notes, що природньо ізолює дані різних користувачів. Видалення нотатки відбувається не через стандартне вікно alert, а через власне модальне вікно з підтвердженням — це зроблено для кращої інтеграції з дизайном застосунку. Закріплені нотатки візуально відокремлюються від решти та завжди відображаються у верхній частині списку незалежно від сортування.

Автозбереження та копіювання. Автозбереження реалізоване через таймер із затримкою 2 секунди після останньої зміни. Такий підхід дозволяє уникнути надто частих записів у базу даних, коли користувач швидко друкує. Статус збереження відображається у вигляді тексту в правому нижньому кутку редактора: три крапки означають незбережені зміни, піктограма «💾» — процес, галочка «✓» — успішне збереження. Копіювання тексту реалізоване через Clipboard API [8] з тимчасовим спливаючим повідомленням, яке зникає через 2 секунди.

Організація та пошук нотаток. Теги зберігаються у вигляді масиву рядків у кожному документі нотатки. Для фільтрації достатньо перевірити наявність обраного тегу в цьому масиві. Пошук працює одночасно за заголовком та вмістом нотатки. Регулярний вираз, що використовується для пошуку, ескейпить спеціальні символи, тому пошук коректно працює навіть з такими знаками, як дужки або знаки пунктуації. Знайдені слова обгортаються в HTML-тег <mark>, завдяки чому вони підсвічуються жовтим фоном без додаткових стилів.

AI-асистент. Запити до Google Gemini API виконуються не безпосередньо з браузера, а через власний сервер Node.js, що дозволяє приховати API-ключ. Користувач обирає одну з чотирьох функцій: підсумок, категорію, теги або оптимізацію. Сервер формує промпт залежно від обраної функції, надсилає його до моделі gemini-2.5-flash та повертає результат. Якщо API недоступний, активується локальна імітація — вона менш точна, але дозволяє додатку продовжувати роботу.

Інтерфейс та зручність. Темна тема реалізована через глобальний CSS-клас, який додається до кореневого елемента body. Вибір теми зберігається в localStorage, тому після перезавантаження сторінки він не втрачається. Плаваюча кнопка створення нотатки має фіксоване позиціонування (position: fixed), тому вона завжди видна незалежно від прокручування. Усі дані синхронізуються з Firestore у реальному часі, але оновлення списку нотаток відбувається лише після отримання відповіді від сервера, що дозволяє уникнути «миготіння» інтерфейсу.

2.1.2. Нефункціональні вимоги

Адаптивність та сумісність. Інтерфейс побудований з використанням гнучких одиниць вимірювання (проценти, flex, grid), тому він коректно масштабується на різних пристроях. На мобільних пристроях ширина сайдбару та правого блоку зменшується до 100%, а основна область редактора займає весь екран, що робить роботу з текстом більш зручною.

Продуктивність. Запити до Gemini API в середньому займають 1-3 секунди в залежності від швидкості інтернет-з'єднання та довжини тексту. Якщо відповіді немає довше 5 секунд, сервер повертає тайм-аут, а клієнт показує повідомлення про помилку. Локальні операції (створення нотатки, пошук, додавання тегу) не вимагають звернення до сервера, тому виконуються миттєво — це було свідомим рішенням, щоб зробити інтерфейс більш чуйним.

Безпека. Автентифікація повністю делегована Firebase, тому паролі користувачів ніколи не передаються на власний сервер. API-ключ Gemini зберігається у змінних середовища (файл .env) і не потрапляє до репозиторію — це дозволяє уникнути його витоку навіть у разі публікації коду на GitHub. Правила безпеки Firestore налаштовані так, що запит на читання або запис нотатки перевіряє відповідність userId в шляху до документа та request.auth.uid. Жодних інших правил доступу не передбачено, що мінімізує ризик помилок у конфігурації.

Надійність та стабільність. У разі недоступності Gemini API або перевищення ліміту запитів сервер повертає статус 500, а клієнт відображає повідомлення «Помилка AI, спробуйте пізніше». Жодних винятків, що «падають» весь додаток, при цьому не виникає. Локальна імітація AI, хоч і менш точна, дозволяє користувачеві отримати хоча б якийсь результат, якщо основний сервіс тимчасово недоступний.

Портативність. Уся конфігурація застосунку винесена в змінні середовища, тому для запуску на іншому комп'ютері достатньо мати Node.js, npm та встановити залежності. Жодних додаткових налаштувань сервера не потрібно. Застосунок успішно працює на localhost:3000 (фронтенд) та localhost:5000 (бекенд) без необхідності розгортання на хостингу.

2.2. Технології та засоби розробки

Для реалізації проєкту було сформовано технологічний стек, кожен компонент якого обрано з урахуванням специфіки завдань, що вирішуються.

Проєкт включає в собі клінєнську частину (фронтенд) та серверну частину (бекенд) та хмарну базу даних.

React — це бібліотека JavaScript від Meta для побудови користувацьких інтерфейсів на основі компонентного підходу. Використання функціональних компонентів у поєднанні з React Hooks (useState, useEffect, useContext) дозволяє будувати реактивний інтерфейс із чітким управлінням станом без використання класових компонентів [3].

Вибір React обумовлений такими факторами: висока швидкість рендерингу завдяки Virtual DOM, широка екосистема бібліотек, зручне повторне використання компонентів (картки нотаток, панель AI-асистента, сайдбар), а також простота інтеграції з Firebase SDK та зовнішніми API.

Node.js — це середовище виконання JavaScript [4] на стороні сервера на базі рушія V8. Express — мінімалістичний фреймворк для побудови REST API на Node.js. Серверна частина застосунку відповідає за проксування запитів до Google Gemini API, що дозволяє приховати API-ключ від клієнтської частини та централізовано керувати логікою обробки AI-запитів [4].

Вибір Node.js обумовлений єдиною мовою (JavaScript) для фронтенду та бекенду, що спрощує розробку, а також асинхронною моделлю виконання, оптимальною для роботи з зовнішніми API.

Firebase — це платформа від Google для розробки вебта мобільних застосунків, що надає хмарні сервіси без необхідності розгортання власної інфраструктури [5].

У проєкті використовуються три сервіси Firebase:

- **Firebase Authentication** — забезпечує автентифікацію користувачів через провайдер Google Sign-In. Користувач входить у застосунок одним кліком, не створюючи окремого акаунта.
- **Cloud Firestore** — NoSQL хмарна база даних у реальному часі. Нотатки зберігаються як документи у колекціях, прив'язаних до ідентифікатора користувача (UID), що гарантує ізоляцію даних між акаунтами.
- **Google Sign-In** — OAuth 2.0-провайдер, інтегрований у Firebase Auth, що забезпечує безпечну автентифікацію без зберігання паролів.

Вибір Firebase обумовлений відсутністю необхідності налаштовувати власну базу даних, вбудованими правилами безпеки (Firestore Security Rules), безкоштовним тарифом (Spark Plan) для навчальних проєктів та тісною інтеграцією з екосистемою Google.

Google Gemini 2.5 Flash — це велика мовна модель (LLM) від Google, оптимізована для швидкої обробки текстових запитів із збереженням високої якості відповідей. Модель доступна через REST API та використовується в проєкті для реалізації чотирьох AI-функцій: підсумовування, категоризації, генерування тегів та оптимізації тексту нотаток [6].

Вибір Gemini 2.5 Flash обумовлений низькою затримкою відповіді порівняно з повними версіями моделі, безкоштовним рівнем доступу для розробників, підтримкою українськомовних запитів та можливістю гнучкого налаштування промптів під конкретні задачі.

Visual Studio Code — основне середовище розробки проєкту. Використовувались розширення ESLint (аналіз коду), Prettier (форматування), GitLens (робота з Git) та Firebase Explorer для взаємодії з хмарною базою даних безпосередньо з редактора [1].

3. ПРОЄКТУВАННЯ ЗАСТОСУНКУ

3.1. Загальна архітектура додадку

Застосунок побудований за клієнт-серверною архітектурою [10] з розподілом відповідальності між чотирма основними рівнями: клієнтська частина (React), серверна частина (Node.js + Express), хмарна платформа (Firebase) та зовнішній AI-сервіс (Google Gemini API).

Клієнтська частина (React) відповідає за відображення інтерфейсу користувача, управління локальним станом застосунку та взаємодію з Firebase SDK безпосередньо з браузера. Компоненти React організовані за функціональним принципом: окремі компоненти для списку нотаток, картки нотатки, панелі AI-асистента, сайдбару, профілю користувача та форми пошуку.

Серверна частина (Node.js + Express) виступає проміжним шаром між клієнтом та Google Gemini API. Основна функція сервера — приймати AI-запити від клієнта, формувати промпти, надсилати їх до Gemini API та повертати оброблені результати. Такий підхід дозволяє приховати API-ключ Gemini від клієнтської частини та централізовано контролювати логіку AI-функцій.

Firebase забезпечує два критичних сервіси: автентифікацію користувачів (Firebase Auth + Google Sign-In) та хмарне зберігання нотаток (Cloud Firestore). Firebase SDK підключається безпосередньо до React-застосунку, що дозволяє здійснювати читання та запис даних у реальному часі без проміжного сервера для цих операцій.

Google Gemini 2.5 Flash API — зовнішній сервіс штучного інтелекту, до якого звертається виключно серверна частина. Взаємодія відбувається через HTTPS-запити з передачею тексту нотатки та відповідного промпту.

Загальна схема взаємодії компонентів виглядає наступним чином:

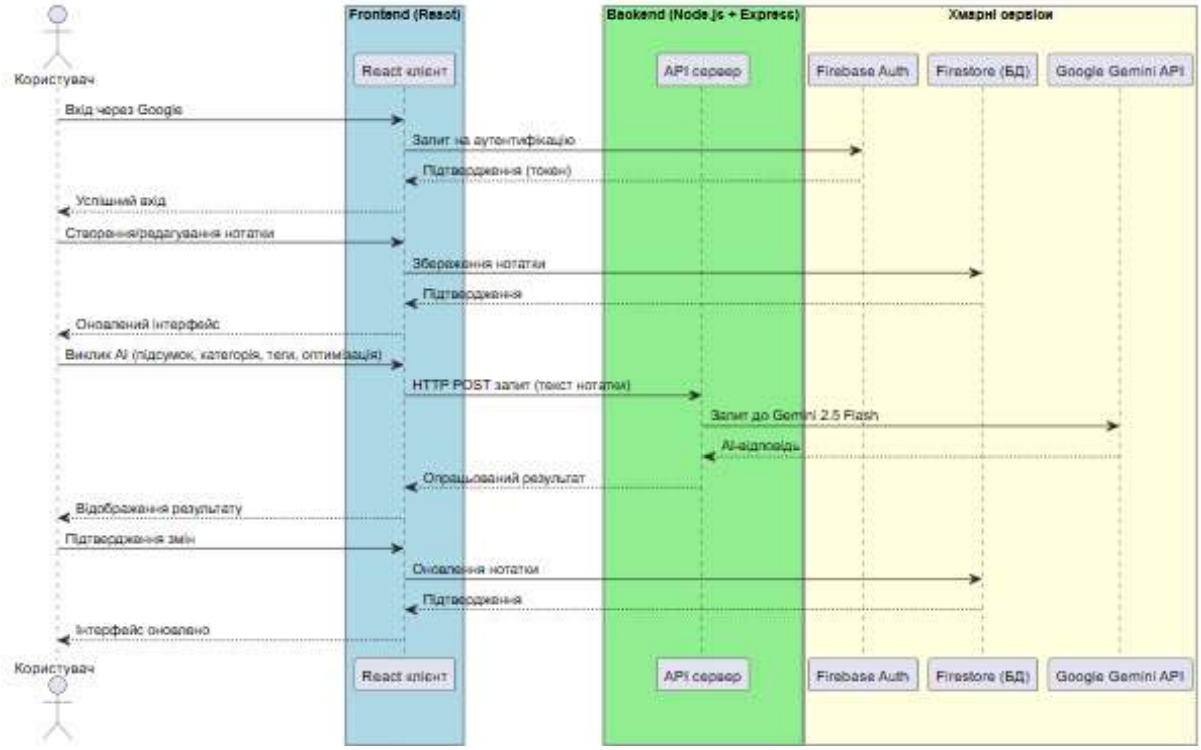

Рис. 3.1 «Схема взаємодії компонентів»

3.2. UML Use Case Diagram

Use Case діаграма описує взаємодію між користувачем та системою на рівні функціональних можливостей. У застосунку передбачено одну роль — **автентифікований користувач** — після входу через Google Sign-In [9].

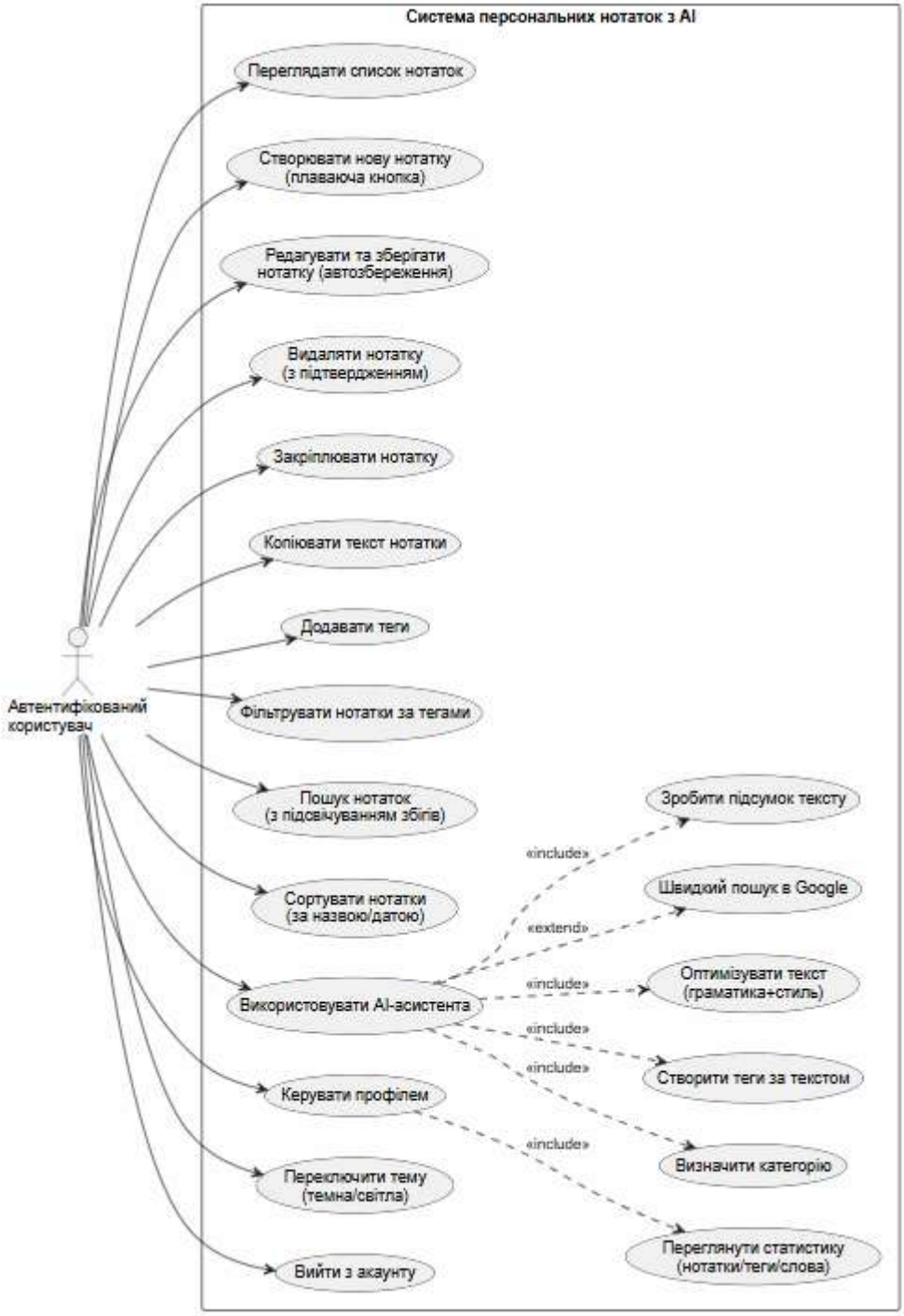

Рис. 3.2 «UML Use Case Diagram»

Актором у даній системі виступає автентифікований користувач — особа, яка пройшла процедуру входу через Google Sign-In. Після успішної автентифікації користувач отримує доступ до всіх функцій системи, описаних нижче. Важливо зазначити, що система не передбачає окремої ролі адміністратора або анонімного доступу — це свідоме архітектурне рішення, оскільки застосунок орієнтований на персональне використання.

Користувач може виконувати наступні дії в системі:

- переглядати список власних нотаток,
- створювати нову нотатку (зокрема через плаваючу кнопку),
- редагувати та зберігати нотатку (з автозбереженням),
- видаляти нотатку з підтвердженням,
- закріплювати нотатку у верхній частині списку,
- копіювати текст нотатки,
- додавати теги та фільтрувати нотатки за ними,
- здійснювати пошук із підсвічуванням збігів,
- викликати панель AI-асистента для обробки тексту нотатки,
- переглядати профіль із статистикою та датою реєстрації,
- перемикати темну та світлу тему інтерфейсу.

3.3. UML Activity Diagram

Activity діаграма моделює послідовність дій при роботі з AI-асистентом — одним із ключових процесів застосунку [9].

Важливо зазначити, що процес AI-обробки є повністю асинхронним. Після надсилання запиту до сервера користувач може продовжувати редагувати нотатку або виконувати інші дії. Коли відповідь від Gemini API надходить, результат з'являється в панелі AI-асистента без перезавантаження сторінки

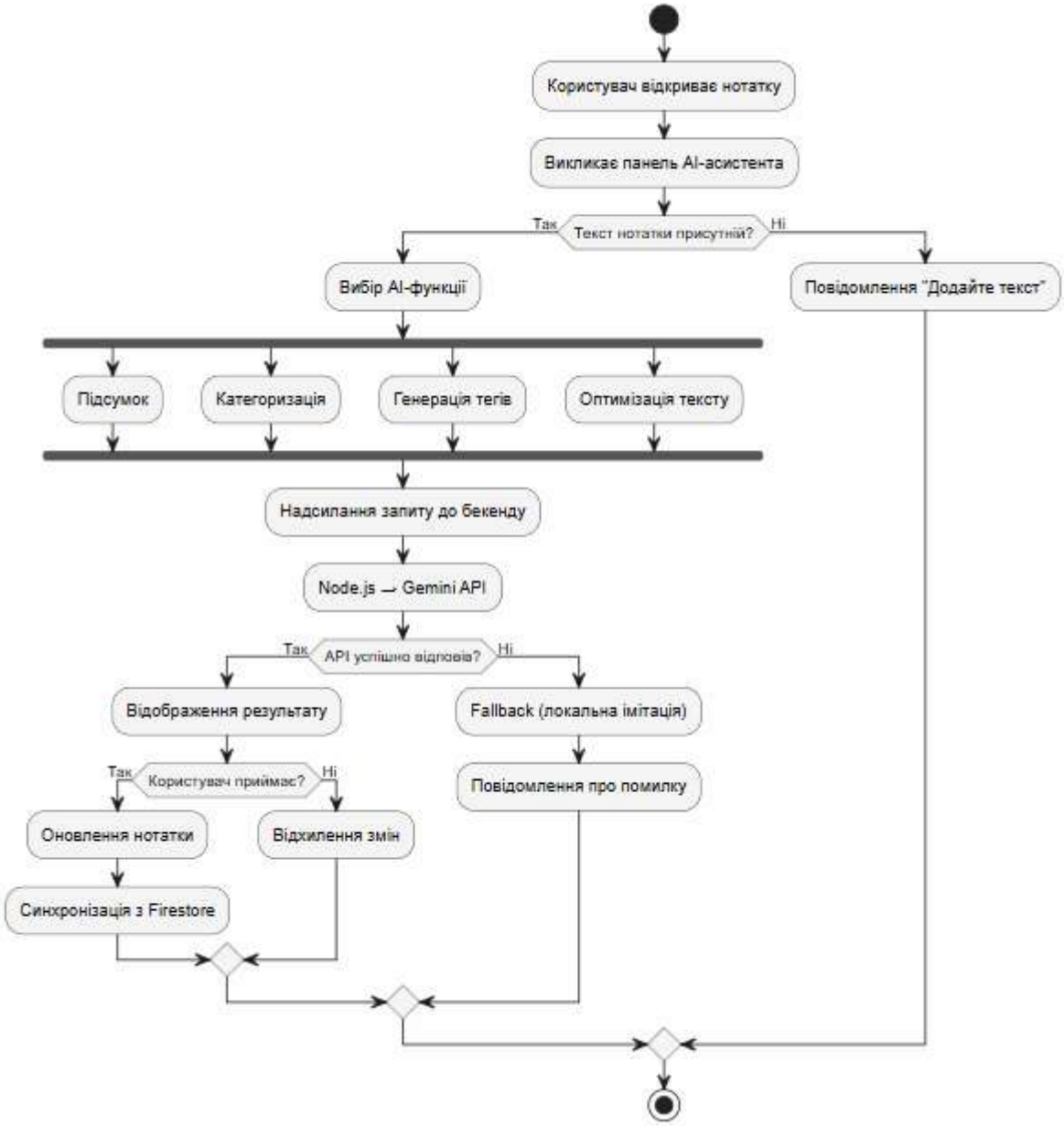

Рис. 3.3 «UML Activity Diagram»

Процес AI-обробки нотатки:

1. Користувач відкриває нотатку та викликає панель AI-асистента.
2. Система перевіряє наявність тексту в нотатці.
3. Якщо текст відсутній — відображається повідомлення з проханням додати вміст.
4. Якщо текст присутній — користувач обирає одну з чотирьох AI-функцій (підсумовування, категоризація, теги, оптимізація).
5. Клієнт надсилає запит на сервер Node.js із текстом нотатки та обраною функцією.

6. Сервер формує промпт та надсилає запит до Gemini API.
7. У разі успішної відповіді — результат відображається в панелі AI-асистента.
8. У разі помилки (недоступність API, перевищення ліміту) — відображається fallback-повідомлення про помилку.
9. Користувач може прийняти результат і застосувати його до нотатки або відхилити.

3.4. UML Sequence Diagram

Sequence діаграма відображає послідовність повідомлень між компонентами системи під час автентифікації та завантаження нотаток [9].

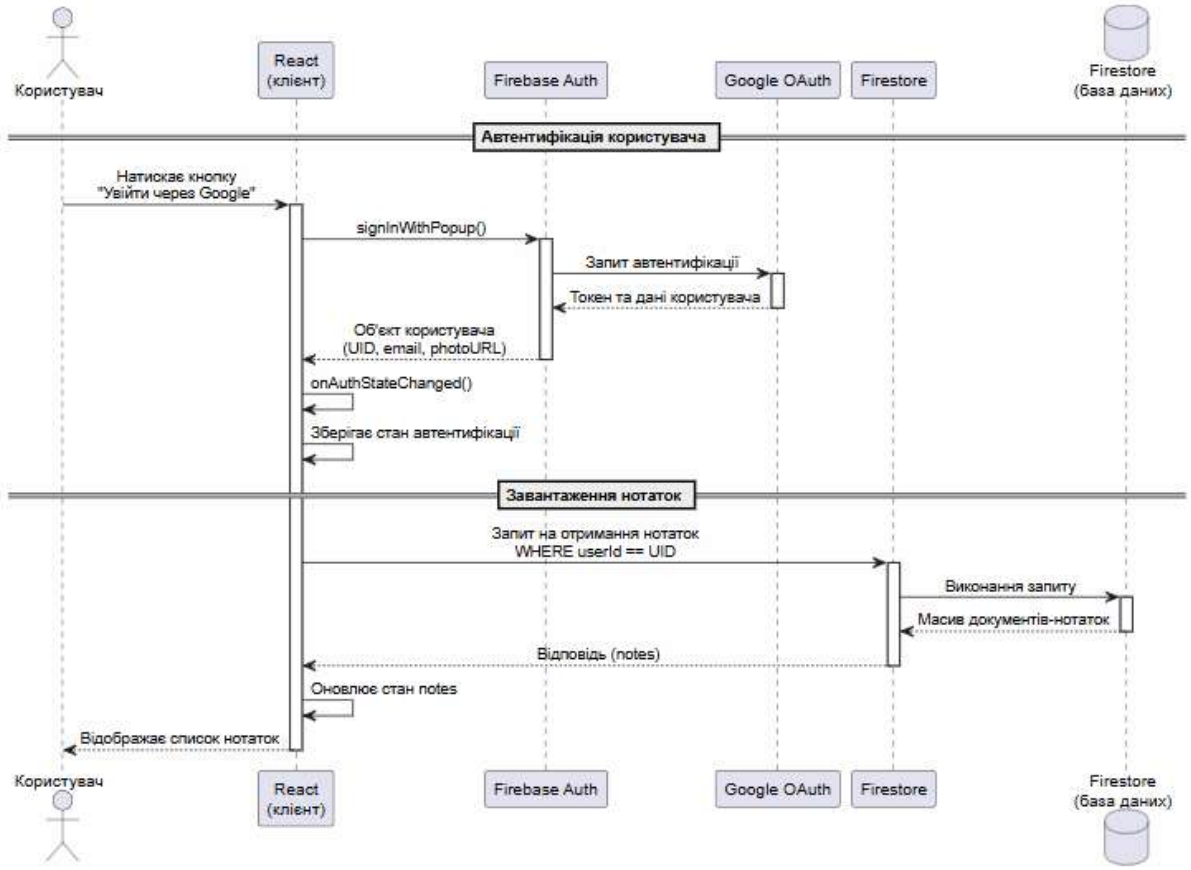

Рис. 3.4 «UML Sequence Diagram»

Сценарій: вхід користувача та завантаження нотаток

1. Користувач натискає кнопку «Увійти через Google».

2. React викликає signInWithPopup з Firebase Auth.
3. Firebase Auth перенаправляє запит до Google OAuth 2.0.
4. Google повертає токен автентифікації до Firebase Auth.
5. Firebase Auth повертає об'єкт користувача (UID, email, аватар) до React.
6. React зберігає стан автентифікації через onAuthStateChanged.
7. React формує запит до Firestore: отримати колекцію нотаток, де userId == UID.
8. Firestore повертає масив документів-нотаток.
9. React оновлює стан та відображає список нотаток у інтерфейсі.

Обрана послідовність забезпечує максимальну безпеку: автентифікація відбувається на стороні Google, паролі ніколи не передаються в застосунок, а Firestore повертає лише ті нотатки, які належать поточному користувачеві завдяки правилам безпеки.

3.5. UML ER (Entity-Relationship(Сутність-зв'язків)) Diagram

База даних побудована на основі Cloud Firestore (NoSQL). Основна структура даних включає дві сутності: **User** та **Note** [9].

Сутність User (зберігається у Firebase Auth):

- uid — унікальний ідентифікатор користувача,
- email — електронна пошта,
- displayName — ім'я користувача,
- photoURL — URL аватара,
- createdAt — дата реєстрації.

Сутність Note (зберігається у колекції Firestore notes):

- id — унікальний ідентифікатор нотатки,
- userId — ідентифікатор власника (зв'язок з User),
- title — заголовок нотатки,
- content — текст нотатки,
- tags — масив тегів,

- category — категорія (може бути визначена AI),
- isPinned — ознака закріплення,
- createdAt — дата створення,
- updatedAt — дата останнього редагування.

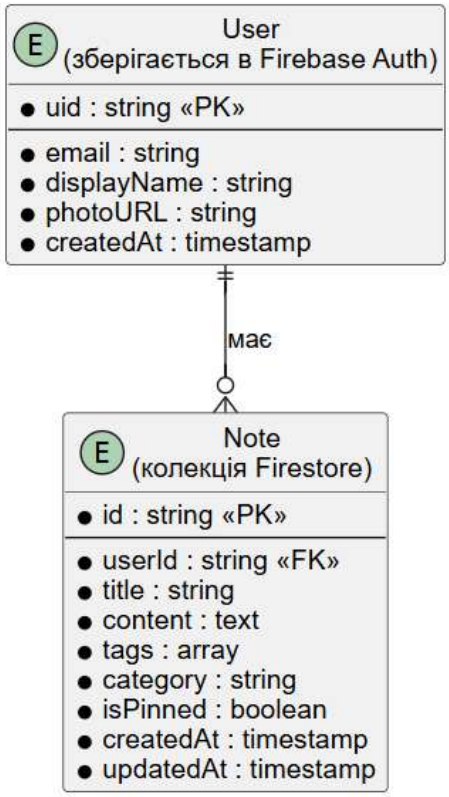

Рис. 3.5 «UML ER Diagram»

Зв'язок між сутностями: один користувач може мати багато нотаток (1:N). Ізоляція даних між користувачами забезпечується правилами безпеки Firestore (Security Rules), які дозволяють читання та запис лише для власника нотатки.

4. ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

У цьому розділі ми детально опишемо, як саме було реалізовано всі ключові частини застосунку: від клієнтської архітектури до інтеграції з Firebase та AI. Пояснимо, чому було обрано ті чи інші підходи, як організовано взаємодію між компонентами й що було зроблено, щоб додаток був зручним і надійним.

4.1. Організація клієнтської частини

Працюючи над фронтендом, ми вирішили використовувати React, оскільки він дає змогу розбити інтерфейс на невеликі, незалежні компоненти. Це дуже зручно, коли проєкт розростається. Уся клієнтська логіка побудована на функціональних компонентах і React Hooks — це сучасний підхід, який робить код чистішим і зрозумілішим.

Загалом додаток складається з трьох великих блоків, які видно на головному екрані:

- **лівий сайдбар** – тут розташовані пошук, фільтри за тегами, сортування та список нотаток;
- **центральна зона** – це редактор, де користувач працює із заголовком, текстом і тегами нотатки;
- **права панель** – тут знаходиться AI-асистент, який виконує підсумовування, категоризацію, оптимізацію та генерує теги.

Така структура не випадкова: вона дозволяє бачити одразу й список, і вміст вибраної нотатки, і пропозиції AI – усе на одному екрані. Це дуже зручно, коли потрібно швидко опрацювати кілька нотаток.

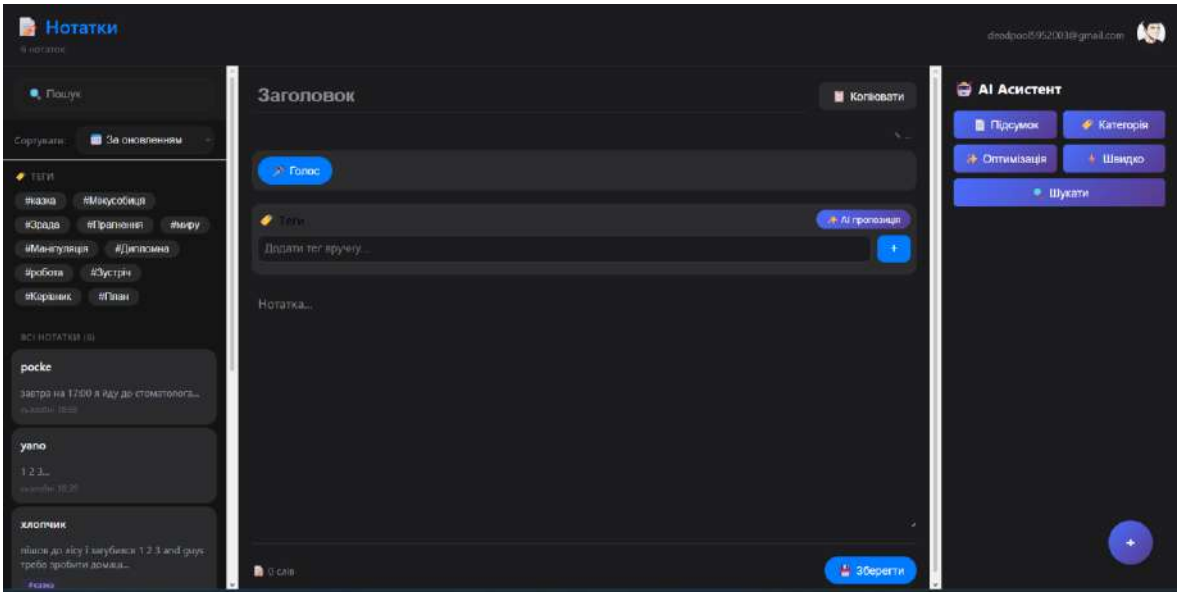

Рис. 4.1 «Головне вікно»

Лістинг 4.1 – Головний компонент App.js (фрагмент)

```
function App() {
  const [notes, setNotes] = useState([]);
  const [darkMode, setDarkMode] = useState(false);

  useEffect(() => {
    const savedTheme = localStorage.getItem('darkMode');
    if (savedTheme) setDarkMode(savedTheme === 'true');
  }, []);

  return (
    <div  className={`app  ${darkMode  ?  'dark-theme'  :
''}`}>
      <NoteList notes={notes} />
      <NoteEditor onSave={saveNote} />
      <AIAssistant content={selectedNote?.content} />
    </div>
  );
}
```

4.2. Робота з Firebase: авторизація та хмарне зберігання

Я не хотів ускладнювати собі життя написанням власного бекенду для зберігання даних і реєстрації. Тому обрав Firebase – готове хмарне рішення від Google [5].

Авторизація зроблена через Google Sign-In. Це буквально дві-три дії: додаєш Firebase до проєкту, підключаєш модуль автентифікації, і все. Кнопка «Увійти через Google» викликає вікно вибору акаунта, після чого додаток отримує унікальний ідентифікатор користувача (UID) та інші дані — ім'я, email, аватар.

Після успішного входу, я відразу завантажую нотатки цього користувача з Firestore. Вони зберігаються за таким шляхом: users/{userId}/notes. Це означає, що кожен користувач бачить лише свої нотатки, навіть фізично вони лежать в одній базі даних – ізоляція досягається за допомогою правил безпеки Firestore.

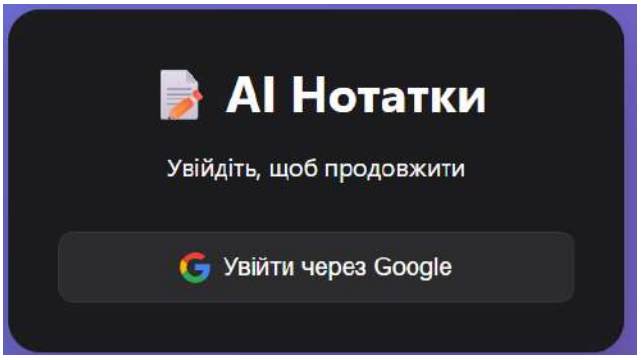

Рис. 4.2 «Сторінка авторизації»

Лістинг 4.2 – Ініціалізація Firebase та вхід через Google

```
import { initializeApp } from "firebase/app";
import { getAuth, GoogleAuthProvider, signInWithPopup } from "firebase/auth";

const firebaseConfig = { ... };
const app = initializeApp(firebaseConfig);
const auth = getAuth(app);
```

```
const provider = new GoogleAuthProvider();

const handleLogin = async () => {
  try {
    const result = await signInWithPopup(auth, provider);
    console.log("Користувач:", result.user);
  } catch (error) {
    console.error("Помилка входу:", error);
  }
};
```

4.3. Основа роботи з нотатками

Усі базові операції з нотатками ми реалізували через Firebase Firestore. Після входу користувача завантажуються його нотатки, які сортуються за датою оновлення (нові зверху) й одразу відображаються в сайдбарі.

Створення нової нотатки починається з натискання великої плаваючої кнопки з плюсом унизу праворуч. Редагування відбувається в центральному редакторі, а видалення — через кнопку з кошиком на картці нотатки. Щоб уникнути випадкового видалення, я додав кастомне модальне вікно з підтвердженням: воно ставить запитання «Ви впевнені?» і чекає на дію.

Автозбереження — одна з найзручніших функцій. Якщо користувач щось змінив, через 2 секунди після останньої зміни дані автоматично надсилаються до Firebase. У нижній частині редактора відображається статус: «...» (ще не збережено), «💾 Збереження...» або «✓ Збережено щойно». Це працює так само, як у Google Docs, і ніхто не втрачає текст через забуту кнопку «Зберегти».

Теги та пошук — це те, що робить додаток зручним для великої кількості нотаток. Користувач може додавати теги вручну або отримати їх від AI. Клік по тегу в сайдбарі миттєво фільтрує список нотаток. Пошук шукає одночасно в

заголовку та вмісті, а знайдені слова підсвічуються жовтим фоном — результати видно одразу.

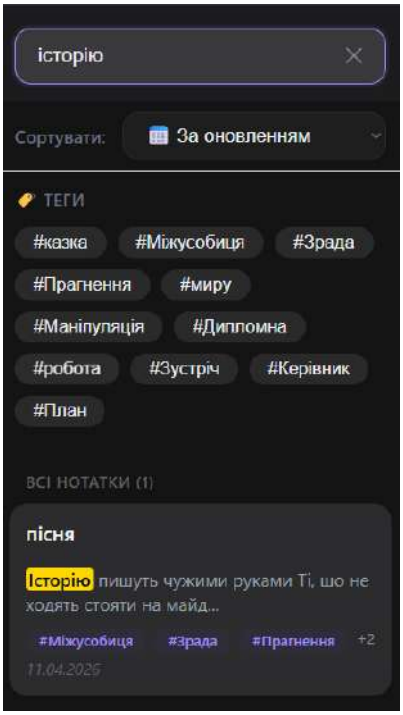

Рис. 4.3 «Теги та пошук»

Лістинг 4.3 – Отримання нотаток з Firestore

```
import { collection, getDocs, addDoc, updateDoc, deleteDoc, doc } from "firebase/firestore";

const fetchNotes = async (userId) => {
  const q = collection(db, `users/${userId}/notes`);
  const snapshot = await getDocs(q);
  const notesData = snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
  setNotes(notesData);
};

const deleteNote = async (noteId) => {
  await deleteDoc(doc(db, `users/${user.uid}/notes`, noteId));
```

```
    setNotes(notes.filter(n => n.id !== noteId));
  };
```

4.4. Інтеграція з Google Gemini API: AI-асистент

Найцікавіша частина проєкту — це AI-асистент. Замість того щоб просто зберігати нотатки, додаток їх аналізує. Для цього я використав Google Gemini 2.5 Flash — сучасну мовну модель [6], яка швидко працює й добре розуміє українську.

Запити до Gemini не можна надсилати безпосередньо з браузера (API-ключ засвітився б у коді), тому я створив простий сервер на Node.js + Express. Він приймає запити від фронтенду, формує промпти, звертається до Gemini й повертає результат.

У правій панелі доступні чотири AI-функції:

- **Підсумок** — скорочує довгу нотатку до кількох речень.
- **Категорія** — визначає тему тексту: Покупки, Зустріч, Ідеї, Робота, Навчання, Важливе, Особисте або Інше.
- **Оптимізація тексту** — виправляє граматику, покращує стиль, робить текст професійнішим.
- **Генерація тегів** — самостійно пропонує теги на основі вмісту нотатки.

Щоб запобігти збоям у роботі застосунку в разі недоступності API, ми передбачили «гібридний режим». Якщо ключ відсутній або сервер не відповідає, активується локальна імітація (вона має менший рівень інтелектуальної обробки, проте працює автономно без інтернету). Таким чином ми забезпечили стабільність роботи системи в будь-яких умовах.

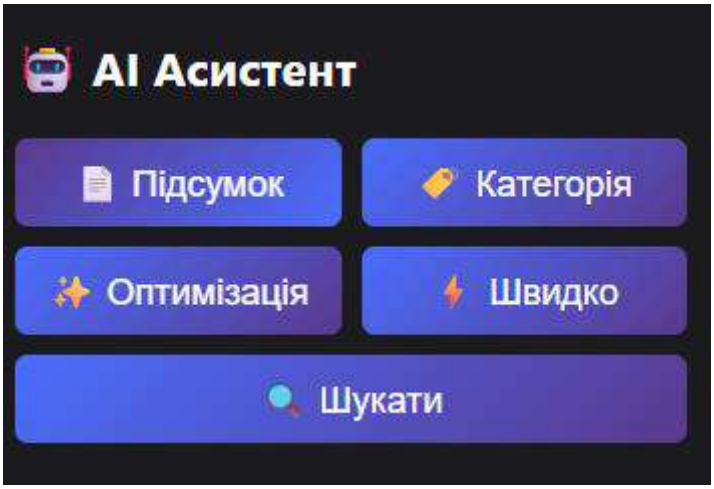

Рис. 4.4 «Панель AI-асистента»

4.5. Копіювання та голосове введення

Дві маленькі, але дуже корисні функції, які значно покращують досвід користування.

Копіювання тексту працює через Clipboard API. Користувач натискає кнопку «📋 Копіювати» — і весь текст нотатки потрапляє в буфер обміну. Після цього з'являється сповіщення «✅ Скопійовано!», яке зникає через секунду.

Голосове введення — це функція для тих, хто не любить друкувати. Вона використовує Web Speech API, вбудований у більшість браузерів (Chrome, Edge, Safari) [8]. Користувач натискає кнопку «🎤 Голос», дозволяє доступ до мікрофона — і можна диктувати текст українською. Система автоматично додає розпізнані слова в редактор.

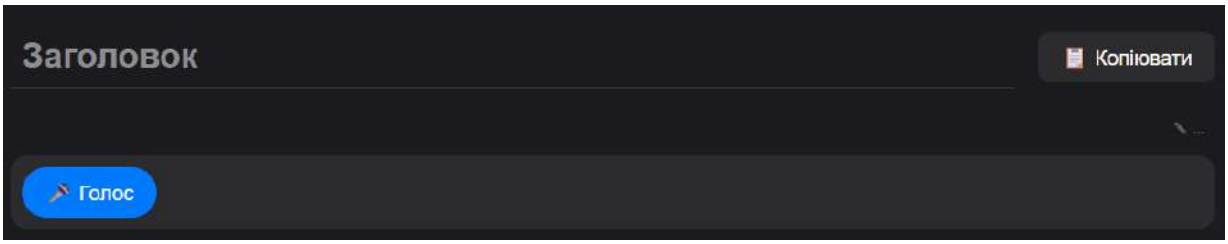

Рис. 4.5 «Голосове введення та копіювання»

Лістинг 4.5 – Голосове введення через Web Speech API

```
const startVoiceInput = () => {
  const recognition = new (window.SpeechRecognition || window.webkitSpeechRecognition)();
  recognition.lang = 'uk-UA';
  recognition.onresult = (event) => {
    const text = event.results[0][0].transcript;
    setContent(prev => prev + ' ' + text);
  };
  recognition.start();
};
```

4.6. Тестування застосунку та якість роботи AI

Після завершення розробки ми провели серію тестів, щоб переконатися у стабільній роботі застосунку та ефективності результатів, які забезпечують AI-функції.

Функціональне тестування охопило всі ключові сценарії: створення нотатки, редагування, пошук, фільтрацію за тегами, автозбереження, перемикання темної теми, а також роботу з Firebase (вхід, вихід, синхронізація). Жоден із цих сценаріїв не викликав критичних помилок — інтерфейс не «вилітав», дані не губилися.

Тестування AI-функцій ми проводили на різних типах текстів: короткі нотатки (одне речення), середні (абзац), довгі (кілька абзаців). Для кожного типу я оцінював якість підсумку, точність визначення категорії та адекватність згенерованих тегів. Найкращі результати Gemini показувала на текстах обсягом 200-500 слів — підсумок виходив змістовним, категорія — точною. Занадто короткі тексти (менше 20 слів) іноді призводили до того, що AI пропонував надто загальну категорію «Інше».

Важливий момент: навіть якщо Gemini API тимчасово недоступний, додаток не падає, а автоматично переходить у локальний режим. Було прийнято рішення не відключивши інтернет під час роботи з AI, у такому разі користувач бачить повідомлення про помилку, але додаток продовжує працювати — нотатки редагуються, зберігаються, видаляються, нічого не блокується.

Тестування продуктивності показало, що час реакції інтерфейсу після дій користувача не перевищує 100 мс (без урахування AI). Запити до Gemini займають 1-3 секунди залежно від швидкості інтернету — це прийнятний результат для асинхронних операцій. Автозбереження не створює надмірного навантаження на Firestore, оскільки запити надсилаються не частіше ніж раз на 2 секунди.

Що стосується безпеки, то всі паролі (точніше, токени) обробляються виключно Firebase, а API-ключ Gemini зберігається лише на сервері й не передається на клієнт. Правила безпеки Firestore налаштовані так, що користувач має доступ тільки до своїх нотаток (request.auth.uid == userId).

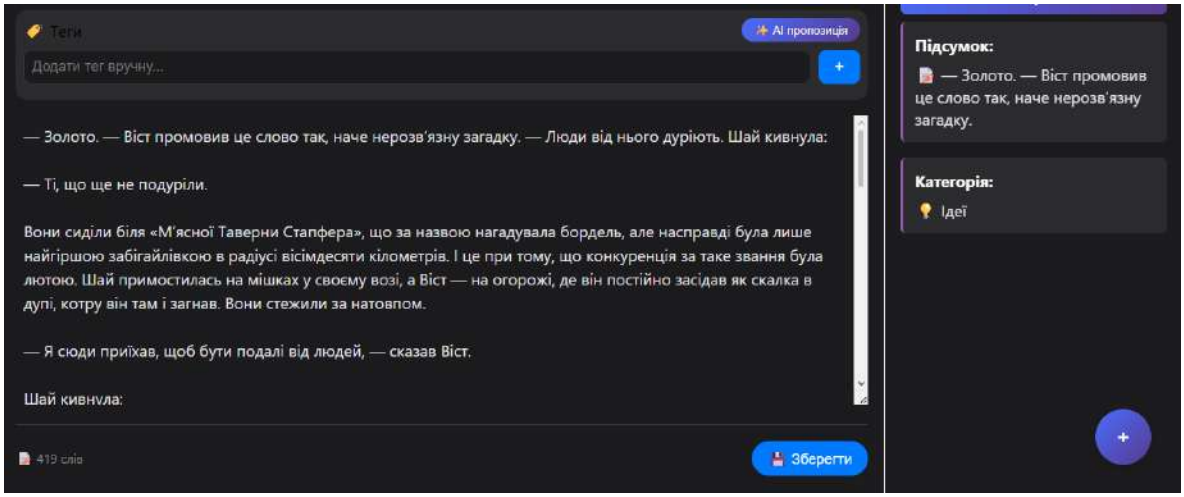

Рис. 4.6 «Приклад роботи AI»

Лістинг 4.6 – Запит до сервера для AI-підсумку

```
const getAISummary = async (text) => {
  const response = await fetch('/api/ai/summarize', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
```

```
    body: JSON.stringify({ content: text })
  });
  const data = await response.json();
  setSummary(data.summary);
};
```

ВИСНОВКИ

У ході виконання кваліфікаційної роботи проведено детальний аналіз предметної області та існуючих застосунків для ведення персональних нотаток. Аналіз показав, що наявні рішення мають ряд обмежень: відсутність вбудованих AI-функцій або їх реалізація виключно в платних тарифах, перевантажений інтерфейс та недостатня гнучкість. Жоден із розглянутих застосунків не поєднує в собі безкоштовний доступ, глибоку інтеграцію AI-функцій та сучасний дизайн, що обґрунтовує актуальність створення власного рішення.

На основі проведеного аналізу сформульовано функціональні та нефункціональні вимоги до системи, обрано відповідний технологічний стек. Архітектура застосунку базується на React SPA, Node.js/Express для проксіювання AI-запитів, Firebase Firestore як хмарній базі даних та Firebase Authentication для входу через Google.

Систему спроєктовано з використанням UML-діаграм (Use Case, Activity, Sequence, ER), що дозволило виявити потенційні проблеми та закласти механізми їх вирішення (автозбереження, гібридний режим AI) [9].

У результаті виконаної роботи розроблено вебзастосунок для персональних нотаток, який включає: автентифікацію через Google, хмарне зберігання нотаток, повний CRUD-цикл, автозбереження, систему тегів, пошук із підсвічуванням, темну тему, копіювання тексту та голосове введення.

AI-асистент на основі Google Gemini 2.5 Flash реалізує чотири функції: підсумовування тексту, визначення категорії, оптимізацію граматики та стилю, автоматичне генерування тегів. Застосовано гібридний підхід: при недоступності API система переходить у локальний режим, що забезпечує безперебійну роботу [6].

Розроблений застосунок є готовим продуктом для ведення персональних записів з інтелектуальною підтримкою. Перспективи розвитку включають додавання зображень до нотаток, резервне копіювання, інтеграцію з календарем та розширення AI-функціоналу.

Результати роботи доповідались на XIII Міжнародної науково-практичної конференції студентів та викладачів факультету фізики, математики, економіки та інноваційних технологій “Актуальні проблеми сучасної науки” (м. Дрогобич, 24-25 квітня 2026 р.)

СПИСОК ВИКОРИСТОВАНОЇ ЛІТЕРАТУРИ

1. Документація Visual Studio Code – [Електронний ресурс] – Режим доступу: - https://code.visualstudio.com/docs
2. Документація JavaScript – [Електронний ресурс] – Режим доступу: - https://uk.javascript.info
3. Документація React – [Електронний ресурс] – Режим доступу: - https://uk.reactjs.org/docs/getting-started.html
4. Вступ до Node.js – [Електронний ресурс] – Режим доступу: - https://nodejs.org/uk/docs/guides/getting-started-guide/
5. Документація Firebase – [Електронний ресурс] – Режим доступу: - https://firebase.google.com/docs
6. Документація Google Gemini API – [Електронний ресурс] – Режим доступу: - https://ai.google.dev/gemini-api/docs
7. Інформація про штучний інтелект –[Електронний ресурс] – Режим доступу: - https://uk.wikipedia.org/wiki/Штучний_інтелект
8. Web Speech API | MDN Web Docs – [Електронний ресурс] – Режим доступу: - https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API
9. Як будувати UML-діаграми – [Електронний ресурс] – Режим доступу: - https://dou.ua/forums/topic/40575/
10. API розробка – [Електронний ресурс] – Режим доступу: - https://restfulapi.net/
11. Н. Коваленко, О. Гарбич-Мошора. Застосування методів штучного інтелекту для автоматизації обробкиперсональних текстових нотаток. Актуальні проблеми сучасної науки: Матеріали XIII Міжнародної науково-практичної конференції студентів та викладачів факультету фізики, математики, економіки та інноваційних технологій / За ред. Юрія Матуріна, Ігоря Столярчука. – Дрогобич: Редакційно-видавничий відділ ДДПУ імені Івана Франка, 2026. – С. 71-74