

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ЗБЕРІГАННЯ
ОСОБИСТИХ НОТАТОК З МОЖЛИВІСТЮ ЇХ АВТОГЕНЕРАЦІЇ
DEVELOPMENT OF A MOBILE APPLICATION FOR STORING
PERSONAL NOTES WITH AN AUTO-GENERATION OPTION

Виконав(ла): студент(ка) 4 курсу, групи КНТ-110
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ХОХЛОВ Н.А.

(ПРІЗВИЩЕ та ініціали)

Керівник ТАБУНЩИК Г.В.

(ПРІЗВИЩЕ та ініціали)

Рецензент ШИТІКОВА О.В.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2024 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ХОХЛОВА Назара Андрійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка мобільного застосунку для зберігання особистих нотаток з можливістю їх автогенерації. Development of a Mobile Application for Storing Personal Notes with an Auto-Generation Option.

керівник проєкту (роботи) к.т.н., професор, ТАБУНЩИК Галина Володимирівна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “24” квітня 2024 року № 168

2. Строк подання студентом проєкту (роботи) 10 червня 2024 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ТАБУНЩИК Г.В., професор		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання “ 15 ” квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз та дослідження предметної області.	1 тиждень	Розділ 1
3	Розробка архітектури програми.	2 тиждень	Розділ 2
4	Основні рішення з реалізації компонентів системи.	3-4 тижні	Розділ 3
5	Експлуатація та тестування програми	5 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
7	Нормоконтроль та рецензування.	7 тиждень	
8	Захист роботи.	8 тиждень	

Студент(ка)

Підпис Назар ХОХЛОВ
(підпис) (Імя ПРИЗВИЩЕ)

Керівник проєкту (роботи)

Галина ТАБУНЩИК
(підпис) (Імя ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи
бакалавра: 138 с., 12 табл., 38 рис., 3 дод., 13 джерел.

МОБІЛЬНИЙ ЗАСТОСУНОК, НОТАТКИ, АВТОГЕНЕРАЦІЯ,
МОВНА МОДЕЛЬ, GPT-3, ANDROID, KOTLIN, FIREBASE.

Об'єкт дослідження – мобільні застосунки для зберігання та обробки особистої інформації.

Предмет дослідження – розробка мобільного застосунка для зберігання особистих нотаток з можливістю їх автогенерації.

Мета роботи – покращити засоби взаємодії з мобільними пристроями за рахунок мобільного додатка, що використовує API для автогенерації текстів.

Матеріали, методи та технічні засоби: мова програмування Kotlin, середовище розробки Android Studio, платформа Firebase, велика мовна модель GPT-3.

Результати: проаналізовано існуючі мобільні застосунки для нотаток, розроблено архітектуру та функціонал мобільного застосунку, реалізовано функціонал автогенерації нотаток на основі GPT-3.

Висновки: створено мобільний застосунок, який дозволяє, створювати, редагувати та зберігати нотатки, синхронізувати нотатки між пристроями користувача, автоматично генерувати нотатки на основі заданих параметрів, шукати нотатки за ключовими словами та іншими параметрами.

Галузь використання – організація особистої інформації, збільшення ефективності роботи, покращення процесу навчання.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 138 pages, 12 tables, 38 figures, 3 appendixes, 13 sources.

MOBILE APPLICATION, NOTES, AUTO GENERATION, MACHINE LEARNING, GPT-3, ANDROID, KOTLIN, FIREBASE.

The object of the research is mobile applications for storing and processing personal information.

The subject of research is the development of a mobile application for storing personal notes with an auto-generation option.

The purpose of the work is to improve the way of interaction with mobile devices using the mobile app structure, which uses a API for auto-generating texts.

Materials, methods and technical means: Kotlin programming language, Android Studio development environment, Firebase platform, large language model GPT-3.

Results: the existing mobile applications for notes were analyzed, the architecture and functionality of the mobile application were developed, and the function of automatic generation of notes based on GPT-3 was implemented.

Conclusions: a mobile application has been created that allows you to create, edit and save notes, synchronize notes between user devices, automatically generate notes based on specified parameters, search for notes by keywords and other parameters.

The field of use is the organization of personal information, increasing work efficiency, improving the learning process.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ.....	9
1 Аналіз предметної області	11
1.1 Проблемна область: мобільні застосунки для нотаток.....	11
1.2 Огляд існуючих рішень.....	12
1.2.1 Застосунок "Google Keep"	12
1.2.2 Застосунок "Evernote"	14
1.2.3 Застосунок "Microsoft OneNote"	16
1.2.4 Застосунок "Notion"	18
1.2.5 Порівняння існуючих аналогів.....	20
1.3 Постановка завдання	23
1.3.1 Функціональність системи	23
1.3.2 Вимоги до інтерфейсу.....	23
1.3.3 Вимоги до даних.....	24
1.3.4 Мінімальні системні вимоги	25
1.3.5 Нефункціональні вимоги до продукту	26
1.4 Висновки за розділом 1	27
2 Розробка архітектури програми	28
2.1 Вибір середовища проєктування.....	28
2.1.1 Вибір платформи для розробки.....	28
2.1.2 Вибір мови програмування.....	30
2.1.3 Вибір середовища розробки	32
2.1.4 Порівняння середовищ розробки.....	37
2.2 Загальна структура застосунку	39

	7
2.3 Висновки за розділом 2.....	41
3 Основні рішення з реалізації компонентів системи.....	43
3.1 Реалізація інтерфейсу користувача.....	43
3.2 Реалізація моделі даних	46
3.3 Інтеграція з Firebase	51
3.4 Реалізація функціоналу автогенерації нотаток.....	54
3.5 Реалізація пошуку та сортування нотаток	55
3.6 Висновки за розділом 3	57
4 Експлуатація та тестування програми.....	58
4.1 Опис застосування.....	58
4.2 Умови виконання програми	59
4.3 Інструкція по експлуатації програми	59
4.3.1 Характеристики програми	59
4.3.2 Перелік файлів, у яких знаходиться програма	61
4.3.3 Звернення до програми	63
4.3.4 Вхідні й вихідні дані	64
4.4 Методика та результати тестування	64
4.4.1 Тестування функціоналу.....	65
4.4.2 Тестування юзабіліті	67
4.4.3 Тестування продуктивності.....	68
Висновки.....	69
Перелік джерел посилання	71
Додаток А Технічне завдання.....	72
Додаток Б Текст програми.....	78
Додаток В Слайди презентації	79

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

- API – Application Programming Interface;
- AAB – Android App Bundle;
- APK – Android Package Kit;
- GPT-3 – Generative Pre-trained Transformer 3;
- IDE – Integrated Development Environment;
- MVC – Model-View-Controller;
- UI – User Interface;
- БД – База даних;
- ОС – Операційна система;
- ПЗ – Програмне забезпечення.

ВСТУП

У сучасному світі мобільні застосунки стали невід'ємною частиною життя, допомагаючи нам в організації особистої інформації, підвищенні продуктивності та вирішенні різноманітних завдань. Одним з найпоширеніших типів застосунків є застосунки для нотаток, які дозволяють зручно фіксувати ідеї, плани, завдання та іншу важливу інформацію.

Збільшення обсягу інформації, з якою ми стикаємося щодня, вимагає ефективних інструментів для її обробки та зберігання. Мобільні застосунки для нотаток дозволяють не тільки зберігати інформацію, але й організовувати її. Деякі застосунки також пропонують можливість синхронізації нотаток між пристроями, що забезпечує доступ до інформації з будь-якого місця.

Однак, більшість існуючих застосунків для нотаток не мають функціоналу автогенерації тексту. Це означає, що користувачеві доводиться вручну вводити весь текст нотатки, що може бути досить нудним та неефективним процесом.

У зв'язку з цим, актуальним є розробка мобільного застосунку для нотаток з можливістю їх автогенерації. Такий застосунок дозволить користувачам не тільки зберігати та організовувати інформацію, але й генерувати новий текстовий контент на основі заданого тексту. Це може бути корисним для різних цілей, таких як:

- створення конспектів лекцій та навчальних матеріалів;
- генерація ідей для проектів та статей;
- створення списків справ та планів;
- підведення підсумків зустрічей та переговорів.

У цій дипломній роботі буде розроблено мобільний застосунок для нотаток з можливістю їх автогенерації на основі великої мовної моделі. Застосунок буде мати зручний інтерфейс, широкий набір функцій та забезпечуватиме надійне зберігання та синхронізацію даних.

Для вирішення поставленого завдання необхідно:

- проаналізувати існуючі мобільні застосунки для нотаток та виявити їх переваги та недоліки;
- спроектувати та розробити архітектуру мобільного застосунку з урахуванням вимог користувачів;
- реалізувати функціонал автогенерації нотаток;
- інтегрувати застосунок з хмарною платформою для зберігання та синхронізації даних;
- провести тестування та оцінити якість роботи застосунку.

Розробка такого мобільного застосунку сприятиме підвищенню ефективності роботи з інформацією, покращенню процесу навчання та розвитку творчих здібностей користувачів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Проблемна область: мобільні застосунки для нотаток

Мобільні застосунки для нотаток стали невід'ємною частиною сучасного життя. Вони дозволяють нам зручно фіксувати ідеї, плани, завдання та іншу важливу інформацію.

Переваги мобільних застосунків для нотаток [1]:

- зручність: мобільні застосунки для нотаток завжди під рукою, що дозволяє швидко та легко робити нотатки в будь-який час і в будь-якому місці;
- організація: мобільні застосунки для нотаток дозволяють організувати нотатки за допомогою тегів, папок та інших функцій;
- синхронізація: деякі мобільні застосунки для нотаток дозволяють синхронізувати нотатки між пристроями, що дає змогу отримати доступ до нотаток з будь-якого місця.

Недоліки мобільних застосунків для нотаток [2]:

- відсутність функціоналу автогенерації: як уже згадувалося вище, більшість мобільних застосунків для нотаток не мають функціоналу автогенерації;
- обмежені можливості форматування: багато мобільних застосунків для нотаток мають обмежені можливості форматування, що може ускладнити читання та розуміння нотаток;
- відсутність інтеграції з іншими сервісами: багато мобільних застосунків для нотаток не інтегруються з іншими сервісами, що може ускладнити роботу з нотатками.

Розробка мобільного застосунку для нотаток з можливістю автогенерації тексту та іншими покращеннями могла б вирішити ці проблеми та зробити застосунки для нотаток більш корисними та зручними для користувачів.

1.2 Огляд існуючих рішень

Існує велика кількість мобільних застосунків для нотаток, але більшість з них не мають функціоналу автогенерації.

Деякі застосунки для нотаток пропонують функції розпізнавання мови, які дозволяють користувачеві диктувати текст нотатки. Однак, ці функції часто неточні та можуть призвести до помилок у тексті.

Інші застосунки для нотаток пропонують функції шаблонів, які дозволяють користувачеві швидко створювати нотатки з певним форматкуванням. Однак, ці шаблони часто негнучкі та не можуть бути легко налаштовані під потреби користувача [3].

1.2.1 Застосунок "Google Keep"

Google Keep - це безкоштовний застосунок для нотаток, розроблений компанією Google. Він доступний на Android, iOS та у веб-версії. Google Keep відрізняється простотою та зручністю використання, що робить його популярним вибором серед користувачів [4].

Основні функції Google Keep:

- створення нотаток різних типів: текстові нотатки, списки, голосові нотатки, фото-нотатки;
- організація нотаток: теги, кольорові мітки, закріплення важливих нотаток;
- спільний доступ до нотаток: можливість ділитися нотатками з іншими користувачами та спільно редагувати їх;
- синхронізація: нотатки автоматично синхронізуються між усіма пристроями користувача;
- нагадування: це можливість встановлювати нагадування для нотаток;

- пошук: зручний пошук нотаток за ключовими словами та тегами.

Переваги Google Keep:

- простота та зручність використання: інтерфейс Google Keep інтуїтивно зрозумілий та простий у використанні;
- доступність: Google Keep доступний на різних платформах та безкоштовний;
- синхронізація: нотатки автоматично синхронізуються між усіма пристроями користувача;
- інтеграція з іншими сервісами Google: Google Keep інтегрується з іншими сервісами Google, такими як Gmail, Google Calendar та Google Drive.

Недоліки Google Keep:

- обмежені можливості форматування: Google Keep має обмежені можливості форматування тексту;
- відсутність функціоналу автогенерації: Google Keep не має функціоналу автогенерації нотаток;
- обмежені можливості для роботи з великими обсягами нотаток: Google Keep не має розширених функцій для організації та управління великими обсягами нотаток.

Висновок: Google Keep - це простий та зручний застосунок для нотаток, який добре підходить для користувачів, яким потрібен базовий функціонал. Однак, він не має функціоналу автогенерації нотаток та розширених можливостей для роботи з великими обсягами інформації.

Приклад вікон застосунку наведено на рисунку 1.1.

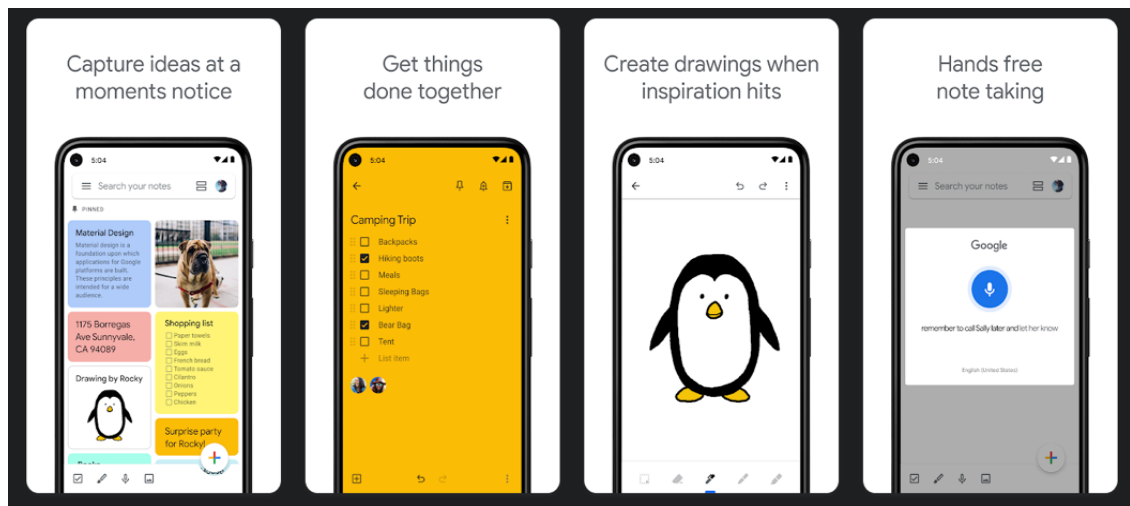


Рисунок 1.1 – Приклад вікон застосунку Google Keep

1.2.2 Застосунок "Evernote"

Evernote - це потужний застосунок для нотаток, який пропонує широкий набір функцій для створення, організації та управління нотатками. Він доступний на різних платформах, включаючи Android, iOS, Windows та macOS. Evernote має як безкоштовну версію, так і платні підписки з розширеним функціоналом [5].

Основні функції Evernote:

- створення нотаток різних типів: текстові нотатки, списки, голосові нотатки, фото-нотатки, рукописні нотатки, веб-кліпіни;
- розширені можливості форматування: Evernote дозволяє формувати текст, додавати таблиці, вставляти зображення та файли;
- організація нотаток: теги, блокноти, стеки блокнотів, пошук;
- спільний доступ до нотаток: можливість ділитися нотатками з іншими користувачами та спільно редагувати їх;
- синхронізація: нотатки автоматично синхронізуються між усіма пристроями користувача;
- інтеграція з іншими сервісами: Evernote інтегрується з різними сервісами, такими як Gmail, Google Drive, Slack та Microsoft Teams;

- шаблони: Evernote пропонує різноманітні шаблони для створення нотаток.

Переваги Evernote:

- потужний функціонал: Evernote пропонує широкий набір функцій для роботи з нотатками;
- розширені можливості форматування: Evernote дозволяє створювати красиві та структуровані нотатки;
- гнучкість: Evernote дозволяє організувати нотатки так, як зручно користувачеві;
- інтеграція з іншими сервісами: Evernote інтегрується з різними сервісами, що розширює його можливості.

Недоліки Evernote:

- складність: інтерфейс Evernote може здатися складним для нових користувачів;
- обмеження безкоштовної версії: безкоштовна версія Evernote має обмеження на обсяг зберігання та функціонал;
- відсутність функціоналу автогенерації: Evernote не має функціоналу автогенерації нотаток.

Висновок: Evernote - це потужний застосунок для нотаток, який добре підходить для користувачів, яким потрібен розширений функціонал та гнучкість. Однак, він може здатися складним для нових користувачів та не має функціоналу автогенерації нотаток.

Приклад вікон застосунку наведено на рисунку 1.2.

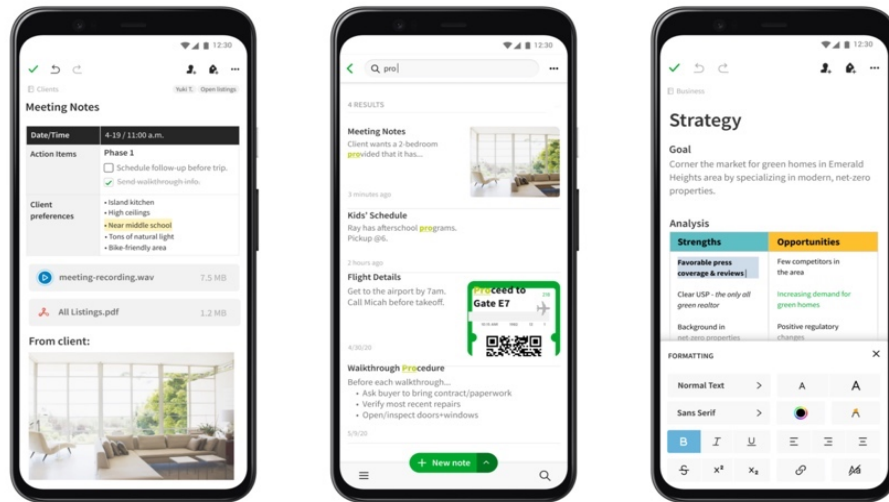


Рисунок 1.2 – Приклад вікон застосунку Evernote

1.2.3 Застосунок "Microsoft OneNote"

Microsoft OneNote - це універсальний застосунок для нотаток, який входить до складу пакету Microsoft Office. Він доступний на різних платформах, включаючи Windows, macOS, Android, iOS та у веб-версії. OneNote пропонує широкий набір функцій для створення, організації та управління нотатками, а також інтегрується з продуктами Microsoft [6].

Основні функції Microsoft OneNote:

- створення нотаток різних типів: текстові нотатки, списки, голосові нотатки, фото-нотатки, рукописні нотатки, веб-кліпінги;
- розширені можливості форматування: OneNote дозволяє формувати текст, додавати таблиці, вставляти зображення та файли, малювати та робити рукописні нотатки;
- організація нотаток: блокноти, розділи, сторінки, теги, пошук;
- спільний доступ до нотаток: можливість ділитися нотатками з іншими користувачами та спільно редагувати їх;
- синхронізація: нотатки автоматично синхронізуються між усіма пристроями користувача;

- інтеграція з іншими продуктами Microsoft: OneNote інтегрується з іншими продуктами Microsoft, такими як Word, Excel, PowerPoint, Outlook;
- шаблони: OneNote пропонує різноманітні шаблони для створення нотаток.

Переваги Microsoft OneNote:

- універсальність: OneNote пропонує широкий набір функцій для роботи з нотатками різних типів;
- розширені можливості форматування: OneNote дозволяє створювати красиві та структуровані нотатки;
- гнучкість: OneNote дозволяє організувати нотатки так, як зручно користувачеві;
- інтеграція з іншими продуктами Microsoft: OneNote інтегрується з іншими продуктами Microsoft, що розширює його можливості;
- доступність: OneNote доступний на різних платформах та безкоштовний для особистого використання.

Недоліки Microsoft OneNote:

- складність: інтерфейс OneNote може здатися складним для нових користувачів;
- обмеження безкоштовної версії: безкоштовна версія OneNote має обмеження на обсяг зберігання та функціонал;
- відсутність функціоналу автогенерації: OneNote не має функціоналу автогенерації нотаток.

Висновок: Microsoft OneNote - це потужний та універсальний застосунок для нотаток, який добре підходить для користувачів, яким потрібен розширений функціонал та інтеграція з іншими продуктами Microsoft. Однак, він може здатися складним для нових користувачів та не має функціоналу автогенерації нотаток.

Приклад вікон застосунку наведено на рисунку 1.3.

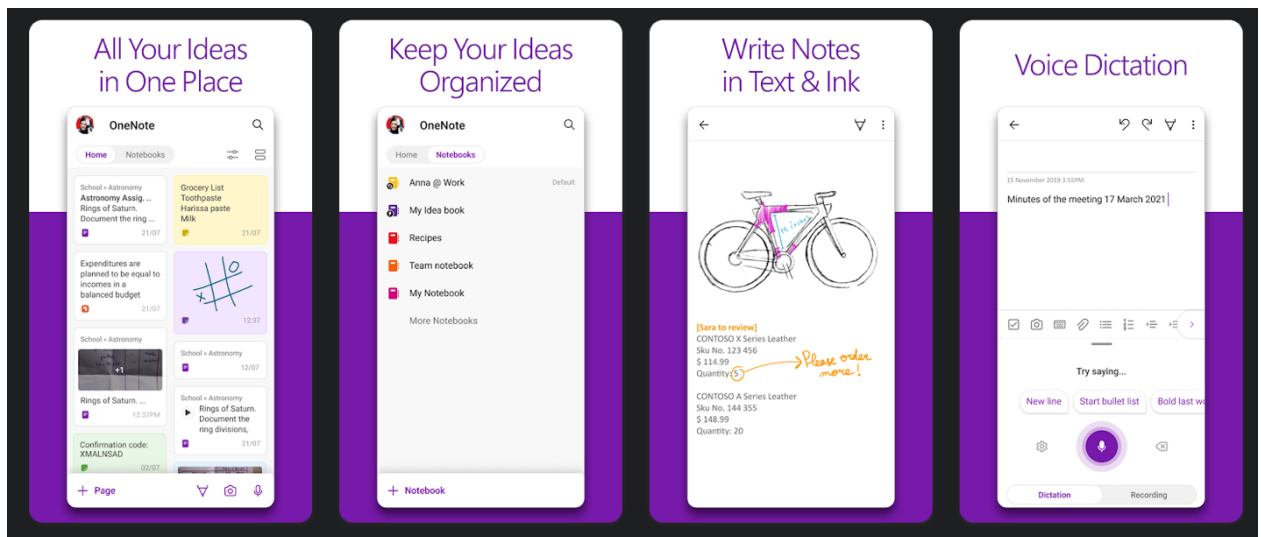


Рисунок 1.3 – Приклад вікон застосунку Microsoft OneNote

1.2.4 Застосунок "Notion"

Notion - це універсальний інструмент для продуктивності, який поєднує в собі функції застосунку для нотаток, менеджера завдань, вікі та бази даних. Він доступний на різних платформах, включаючи веб-версію, десктопні застосунки для Windows та macOS, а також мобільні застосунки для Android та iOS. Notion має як безкоштовну версію, так і платні підписки з розширеним функціоналом.

Основні функції Notion [7]:

- створення сторінок: Notion дозволяє створювати сторінки з різним контентом, включаючи текст, списки, таблиці, календарі, галереї;
- бази даних: Notion дозволяє створювати бази даних з різними типами даних та зв'язками між ними;
- шаблони: Notion пропонує велику бібліотеку шаблонів для різних цілей, таких як управління проектами, ведення нотаток, планування подорожей та інше;
- спільний доступ та співпраця: Notion дозволяє ділитися сторінками та базами даних з іншими користувачами та спільно редагувати;
- інтеграція з іншими сервісами: Notion інтегрується з різними

сервісами, такими як Google Drive, Slack, Trello та іншими;

- Notion AI (beta): Notion нещодавно представив функцію Notion AI, яка знаходиться в бета-версії. Ця функція використовує штучний інтелект для допомоги користувачам у створенні контенту, перекладі, підведенні підсумків, написанні різних видів текстів та іншому.

Переваги Notion:

- універсальність: Notion поєднує в собі функції різних інструментів для продуктивності, що дозволяє користувачам організувати всю свою роботу в одному місці;
- гнучкість: Notion дозволяє користувачам налаштовувати сторінки та бази даних під свої потреби;
- співпраця: Notion дозволяє командам ефективно співпрацювати над проектами;
- Notion AI: нова функція Notion AI має потенціал значно покращити продуктивність користувачів, допомагаючи їм у створенні контенту та виконанні різних завдань.

Недоліки Notion:

- складність: інтерфейс Notion може здатися складним для нових користувачів;
- обмеження безкоштовної версії: безкоштовна версія Notion має обмеження на кількість блоків та функціонал;
- Notion AI (beta): функція Notion AI знаходиться в бета-версії, тому її функціонал може бути обмеженим та нестабільним.

Висновок: Notion - це потужний та універсальний інструмент для продуктивності, який добре підходить для користувачів, яким потрібна гнучкість та можливість організувати всю свою роботу в одному місці. Нова функція Notion AI має потенціал значно покращити продуктивність користувачів, але поки що знаходиться в бета-версії [8].

Приклад вікон застосунку та приклад роботи Notion AI наведено на рисунках 1.4 та 1.5.

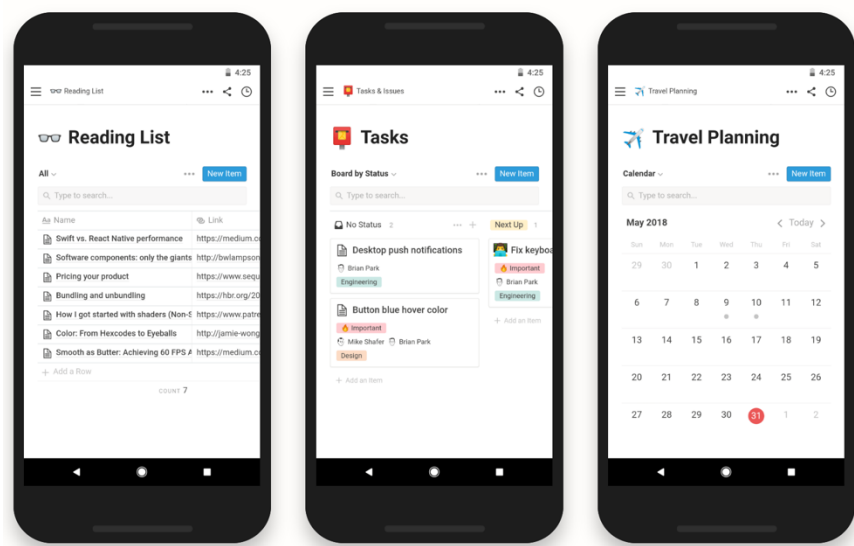


Рисунок 1.4 – Приклад вікон застосунку Notion

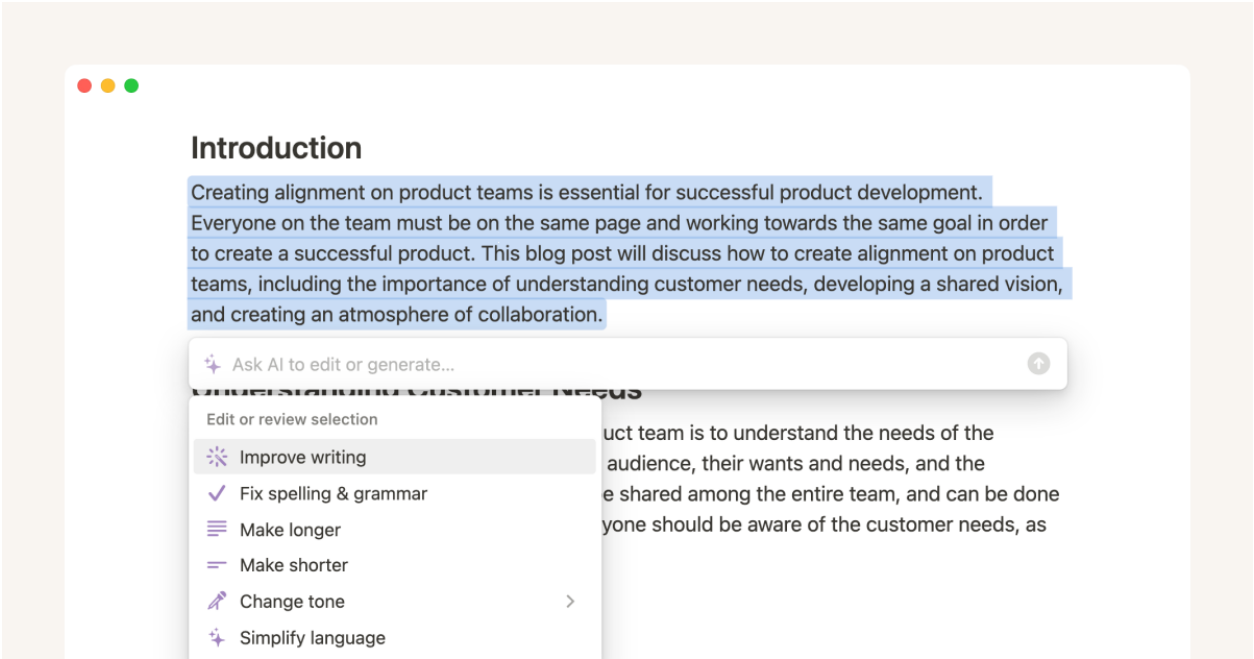


Рисунок 1.5 – Приклад роботи Notion AI

1.2.5 Порівняння існуючих аналогів

Для більш наочного порівняння функціоналу розглянутих застосунків для нотаток, наведемо таблицю 1.1 з основними характеристиками.

Таблиця 1.1 – Порівняння існуючих аналогів

Функція	Google Keep	Evernote	Microsoft OneNote	Notion
1	2	3	4	5
Створення нотаток	Текст, списки, голосові, фото	Текст, списки, голосові, фото, рукописні, веб-кліпінги	Текст, списки, голосові, фото, рукописні, веб-кліпінги	Текст, списки, таблиці, дошки, календарі, галереї, бази даних
Форматування тексту	Обмежене	Розширене	Розширене	Розширене
Організація нотаток	Теги, кольорові мітки, закріплення	Теги, блокноти, стеки блокнотів	Блокноти, розділи, сторінки, теги	Сторінки, бази даних, шаблони
Спільний доступ	Так	Так	Так	Так
Синхронізація	Так	Так	Так	Так
Інтеграція	Google сервіси	Різноманітні сервіси	Microsoft продукти	Різноманітні сервіси
Автогенерація нотаток	Ні	Ні	Ні	Notion AI (beta)
Пошук	Ключові слова, теги	Ключові слова, теги, текст нотаток	Ключові слова, теги, текст нотаток	Ключові слова, теги, текст нотаток, бази даних

Продовження таблиці 1.1

1	2	3	4	5
Нагадування	Так	Так	Так	Так
Експорт/імпорт	Так	Так	Так	Так
Захист паролем	Ні	Так	Так	Так
Безкоштовна версія	Так	Так (з обмеженнями)	Так (з обмеженнями)	Так (з обмеженнями)

Аналіз існуючих застосунків для нотаток виявив, що, хоча вони пропонують широкий спектр функцій, жоден з них не поєднує в собі всі необхідні можливості для ефективної роботи з інформацією.

Основні причини актуальності розробки власного застосунку:

- відсутність комплексного рішення: існуючі застосунки спеціалізуються на окремих аспектах роботи з нотатками, таких як простота використання (Google Keep), розширене форматування (Evernote), інтеграція з екосистемою Microsoft (OneNote) або універсальність (Notion). Однак, жоден з них не поєднує в собі всі ці переваги;

- обмежені можливості автогенерації: Notion AI є єдиним аналогом, що пропонує функцію автогенерації, але вона знаходиться в бета-версії та має обмежений функціонал. Розробка власного застосунку дозволить реалізувати більш гнучкий та адаптивний алгоритм автогенерації, що відповідає потребам;

- можливість індивідуального налаштування: існуючі застосунки мають обмежені можливості налаштування інтерфейсу та функціоналу. Розробка власного застосунку дозволить створити більш персоналізований досвід для користувача.

Таким чином, розробка власного мобільного застосунку для нотаток з можливістю їх автогенерації є актуальною задачею, яка дозволить створити більш ефективний та персоналізований інструмент для роботи з інформацією.

1.3 Постановка завдання

Цей розділ визначає функціональні та нефункціональні вимоги до мобільного застосунку для зберігання нотаток з можливістю їх автогенерації.

1.3.1 Функціональність системи

Створення та редагування нотаток:

- створення нових нотаток з можливістю додавання тексту, зображень, аудіозаписів та інших типів даних;
- редагування існуючих нотаток.

Зберігання та синхронізація нотаток:

- зберігання нотаток в локальній базі даних на пристрої користувача;
- синхронізація нотаток між пристроями користувача через хмарне сховище (Firebase).

Автогенерація нотаток:

- можливість генерувати нотатки на основі заданих параметрів;
- використання великої мовної моделі, для генерації тексту.

Пошук нотаток:

- пошук нотаток за назвою та датою створення.

Додаткові функції:

- можливість захисту нотаток авторизацією.

1.3.2 Вимоги до інтерфейсу

Для забезпечення ергономічності, зручності та зрозумілості інтерфейс розроблюваного мобільного застосунку повинен мати:

- інтуїтивно зрозумілий та простий у використанні інтерфейс;
- сучасний та привабливий дизайн;
- адаптивний дизайн, який підлаштовується під різні розміри екранів;
- зручна навігація між екранами та функціями застосунку.

1.3.3 Вимоги до даних

Мобільний застосунок для нотаток з автогенерацією повинен використовувати хмарний сервіс для зберігання та синхронізації даних. Це означає, що нотатки користувача повинні зберігатися як локально на пристрої, так і в хмарному сховищі.

Основні вимоги до даних:

а) зберігання нотаток:

1) хмарне сховище: нотатки повинні зберігатися у вигляді структурованих даних в хмарному сховищі. Кожна нотатка повинна містити інформацію про текст, час створення, список прикріплених файлів та інше;

2) локальне кешування: для швидкого доступу та роботи в офлайн-режимі, нотатки також повинні кешуватися локально на пристрої користувача;

б) зберігання файлів:

1) хмарне сховище файлів: файли, прикріплені до нотаток, повинні зберігатися в хмарному сховищі. Кожен файл повинен мати унікальний ідентифікатор та бути пов'язаний з відповідною нотаткою;

2) локальне зберігання файлів: для офлайн-доступу, завантажені файли також повинні зберігатися локально на пристрої користувача;

в) аутентифікація користувачів:

1) система аутентифікації: застосунок повинен використовувати систему аутентифікації для ідентифікації користувачів. Це

дозволить безпечно зберігати та синхронізувати дані кожного користувача окремо;

г) безпека даних:

1) контроль доступу: доступ до даних в хмарному сховищі повинен контролюватися, щоб гарантувати, що користувачі мають доступ лише до своїх власних нотаток;

2) шифрування даних: дані, що зберігаються в хмарному сховищі та локально на пристрої, повинні бути зашифровані для забезпечення додаткового рівня безпеки.

Використання хмарного сервісу дозволить забезпечити надійне зберігання, синхронізацію та безпеку даних користувачів, а також спростить розробку та підтримку мобільного застосунку.

1.3.4 Мінімальні системні вимоги

Для забезпечення коректної роботи мобільного застосунку для нотаток з автогенерацією, необхідно дотримуватися наступних мінімальних системних вимог:

а) операційна система:

1) Android 7.0 (Nougat) або вище;

б) апаратне забезпечення:

1) процесор: 1.2 ГГц двоядерний процесор або вище;

2) оперативна пам'ять (RAM): 1 ГБ або більше;

3) внутрішня пам'ять: 50 МБ вільного місця або більше;

4) екран: роздільна здатність 800x480 пікселів або вище;

в) підключення до інтернету:

1) для синхронізації даних та використання автогенерації необхідне стабільне підключення до Інтернету.

Ці мінімальні системні вимоги забезпечать стабільну та ефективну роботу мобільного застосунку для нотаток з автогенерацією.

1.3.5 Нефункціональні вимоги до продукту

Зосереджуючись на нефункціональних аспектах мобільного застосунку для нотаток з автогенерацією, важливо забезпечити позитивний користувацький досвід та ефективність роботи. Ось ключові нефункціональні вимоги, що враховують специфіку цього застосунку:

а) продуктивність:

1) час відгуку: застосунок повинен реагувати на дії користувача швидко та плавно, забезпечуючи комфортне використання;

2) швидкість завантаження: нотатки та файли повинні завантажуватися швидко, як з локального кешу, так і з хмарного сховища, мінімізуючи час очікування користувача;

3) ефективність автогенерації: процес автогенерації нотаток повинен бути оптимізований для швидкого отримання результатів, враховуючи обмежені ресурси мобільних пристроїв;

б) доступність:

1) інтуїтивно зрозумілий інтерфейс: дизайн інтерфейсу повинен бути простим, зрозумілим та зручним для користувачів з різним рівнем технічних знань;

в) зручність використання:

1) інтуїтивна навігація: користувачі повинні легко знаходити потрібні функції та нотатки за допомогою інтуїтивної навігації;

2) зручний пошук: пошук нотаток повинен бути швидким та ефективним, з можливістю використання фільтрів та слів.

Зосередження на цих нефункціональних вимогах дозволить створити мобільний застосунок для нотаток з автогенерацією, який буде зручним, ефективним та доступним для широкої аудиторії користувачів.

1.4 Висновки за розділом 1

У першому розділі було проведено аналіз предметної області мобільних застосунків для нотаток та розглянуто існуючі рішення, такі як Google Keep, Evernote, Microsoft OneNote та Notion. Було виявлено, що, хоча ці застосунки пропонують різноманітні функції для створення, організації та управління нотатками, жоден з них не поєднує в собі всі необхідні можливості для ефективної роботи з інформацією, зокрема, функціонал автогенерації нотаток.

На основі аналізу існуючих рішень та виявлених недоліків, було сформульовано постановку завдання для розробки власного мобільного застосунку для нотаток з можливістю їх автогенерації. Визначено основні функціональні вимоги, такі як створення та редагування нотаток, організація нотаток, зберігання та синхронізація, автогенерація нотаток, пошук нотаток та додаткові функції. Також було сформульовано вимоги до інтерфейсу, мінімальні системні вимоги, вимоги до даних та нефункціональні вимоги, такі як продуктивність, доступність та зручність використання.

Використання платформи для зберігання та синхронізації даних дозволить забезпечити надійність, безпеку та ефективність роботи застосунку. Застосування великої мовної моделі для автогенерації нотаток надасть користувачам унікальні можливості для створення контенту та підвищення продуктивності.

Таким чином, розробка власного мобільного застосунку для нотаток з можливістю їх автогенерації є актуальним та перспективним напрямком, який дозволить створити інноваційний та ефективний інструмент для роботи з інформацією, що відповідає сучасним потребам користувачів.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

У першому розділі було проведено аналіз предметної області мобільних застосунків для нотаток та розглянуто існуючі рішення. На основі проведеного аналізу були сформульовані основні вимоги до розроблюваного мобільного застосунку, включаючи функціональні та нефункціональні вимоги, а також вимоги до даних.

В результаті аналізу можна зробити висновок, що для розробки необхідно обрати платформу та мову програмування для реалізації функціональної частини, середовище розробки для створення інтерфейсу користувача, хмарну платформу для зберігання та синхронізації даних, а також велику мовну модель для реалізації функціоналу автогенерації нотаток.

У цьому розділі буде детально описано обрані технології та архітектуру мобільного застосунку, а також розглянуто процес розробки основних функціональних модулів.

2.1 Вибір середовища проєктування

2.1.1 Вибір платформи для розробки

Для розробки мобільного застосунку для нотаток з автогенерацією було обрано платформу Android. Цей вибір обумовлений наступними факторами:

а) популярність: Android є найпопулярнішою мобільною ОС у світі, з часткою ринку понад 70%. Це означає, що застосунок, розроблений для Android, буде доступний для більшої аудиторії користувачів. Частка ринку різних мобільних ОС показана у табл. 2.1 [9];

Таблиця 2.1 - Частка ринку мобільних ОС

Операційна система	Частка ринку
Android	70.79%
iOS	28.46%
Samsung	0.39%
KaiOS	0.13%
Windows	0.02%
Інші	0.17%

б) відкритість: Android є відкритою платформою, що дозволяє розробникам мати більше свободи та гнучкості при створенні застосунків. Це також означає, що існує велика спільнота розробників Android, що надає підтримку та ресурси;

в) доступність інструментів розробки: Android Studio - це потужне та безкоштовне середовище розробки, яке надає всі необхідні інструменти для створення Android-застосунків;

г) інтеграція з Google сервісами: Android тісно інтегрований з сервісами Google, такими як Firebase, Google Play Services та іншими. Це спрощує розробку та надає додаткові можливості для застосунків;

д) вартість розробки: розробка Android-застосунків, як правило, є більш доступною, ніж розробка iOS-застосунків, завдяки безкоштовним інструментам розробки та відкритій платформі.

Вибір платформи Android дозволяє створити мобільний застосунок для нотаток з автогенерацією, який буде доступний для більшої аудиторії користувачів, матиме більше свободи та гнучкості в розробці, а також буде більш доступним з точки зору вартості розробки [10]. Порівняння платформ Android та iOS наведено у таблиці 2.2.

Таблиця 2.2 - Порівняння Android та iOS

Характеристика	Android	iOS
Частка ринку	70.79%	28.46%
Відкритість	Відкрита	Закрита
Інструменти розробки	Android Studio та інші	Xcode
Інтеграція з Google сервісами	Так	Обмежена
Вартість розробки	Нижча	Вища
Доступність на різних пристроях	Широкий спектр виробників	Тільки на пристроях Apple
Персоналізація	Висока	Обмежена
Апаратні вимоги	Різноманітні, можуть варіюватися від пристрою до пристрою	Стандартізовані, оптимізовані для пристроїв Apple
Контроль над застосунками	Свобода завантаження застосунків з різних джерел	Суворий контроль застосунків через App Store

2.1.2 Вибір мови програмування

Java на даний момент є найпопулярнішою мовою програмування для мобільної розробки на Android. Корпорація Google активно просуває Kotlin як альтернативну мову програмування на заміну Java. Додатки для Android також можуть бути написані іншими мовами [11].

Існують такі мови програмування, які використовуються для розробки мобільних застосунків для Android:

- Java: ця мова є основною для розробки під Android і підтримується Android Studio. Код на Java виконується віртуальною машиною Java (Java Virtual Machine), яка функціонує на пристроях Android і

інтерпретує код.;

- Kotlin: найновіша мова програмування для Android. Вона схожа на Java, але в багатьох аспектах є трохи простішою.;
- C/C++: Android Studio також підтримує цю мову з використанням Java NDK, що є зручним для розробки ігор;
- C#: ця мова є дещо зручнішою для початківців у порівнянні з C або C++. Вона підтримується рядом дуже корисних інструментів, таких як Unity та Xamarin, які чудово підходять для розробки ігор і крос-платформних застосунків.

Проведемо порівняння мов програмування для створення мобільного застосунку та представимо результати у табличній формі (табл. 2.3).

Використаємо наступні критерії:

- швидкість роботи: умовна швидкість роботи технології та мови програмування за оцінками експертів;
- плагіни та розширення: програмні бібліотеки, модулі чи плагіни, які можуть спростити та прискорити процес розробки;
- парадигми програмування: здатність мови підтримувати різні парадигми програмування, що може скоротити час розробки та зробити код більш зрозумілим і легким для підтримки;
- спільнота: кількість активних розробників, які використовують мову програмування, та кількість доступних ресурсів, присвячених цій мові.

Цей аналіз дозволить оцінити різні мови програмування з точки зору їх ефективності, зручності та підтримки розробниками, що допоможе вибрати найбільш підходящу мову для створення мобільного застосунку.

Таблиця 2.3 – Порівняння мов програмування

Критерій	Java	Kotlin	C/C++	C#
1	2	3	4	5
Швидкість роботи	+	±	±	-

Продовження таблиці 2.3

1	2	3	4	5
Плагіни та розширення	+	++	+ (залежить від середовища розробки)	+
Парадигми програмування	±	++	+	±
Спільнота	++	+	±	±

На основі результатів порівняння можна зробити висновок, що на даний момент Kotlin є найкращою альтернативою. Вона має велику спільноту та набір розширень, які спрощують процес розробки. Для створення нашого застосунку на вибрану тему, цілком достатньо обрати мову Kotlin.

2.1.3 Вибір середовища розробки

Android Studio – це інтегроване середовище розробки від Google, яке надає розробникам інструменти для створення застосунків на платформі ОС Android (рис. 2.1). Android Studio можна встановити на Windows, Mac і Linux. Ця середа розробки була створена на базі IntelliJ IDEA. Вона містить макети для створення UI, що зазвичай є початковим етапом роботи над додатком. Android Studio включає інструменти для розробки рішень для смартфонів та планшетів, а також нові технологічні рішення для Android TV, Android Wear, Android Auto, Glass і додаткові контекстуальні модулі [12].

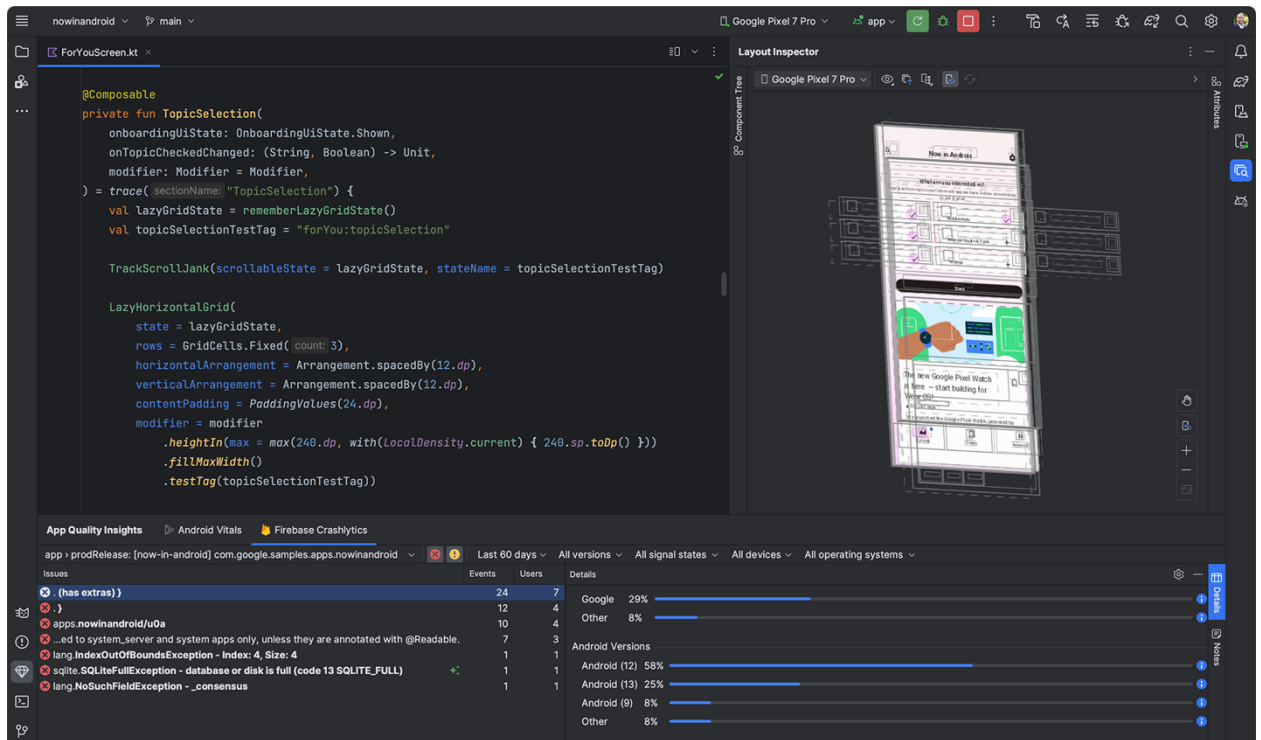


Рисунок 2.1 – Середовище розробки Android Studio

Android Studio призначене як для невеликих команд розробників мобільних застосунків (навіть з однією людиною), так і для великих міжнародних організацій з використанням GIT або інших систем управління версіями.

Переваги:

- мови, які підтримуються: середовище розробки підтримує роботу з декількома мовами програмування, включаючи найпопулярніші - C/C++, Java і Kotlin;
- безкоштовність: Android Studio IDE поширюється безкоштовно;
- рефакторинг: забезпечує зручний і потужний рефакторинг готового коду;
- бібліотека: велика бібліотека з готовими шаблонами і компонентами для розробки ПЗ;
- тестування: широкий набір інструментів для тестування кожного елемента застосунку;
- документація: є багато посібників по використанню Android

Studio для недосвідчених розробників або початківців;

- вбудований емулятор: зручне і швидке тестування в вбудованому Android-емуляторі.

Недоліки:

- системна вимога: для комфортної роботи в середовищі потрібен потужний комп'ютер;

- неможливість написання серверних проектів на мові Java для ПК, Android-пристроїв.

Eclipse Java IDE – це відкрите інтегроване середовище розробки на мові програмування Java. Розробники компанії регулярно вдосконалюють Eclipse, випускаючи оновлення, створюючи нові інструменти та впроваджуючи корисні функції для програмістів.

У цьому середовищі можна підключати додаткові модулі для розширення функціоналу. Наприклад, для розробки проектів на мовах C/C++ необхідно додати модуль C/C++ Development Tools, а для написання програм на Java – Java Development Tools.

Це середовище досить зручне завдяки інтерфейсу і документації, де можна знайти багато корисної інформації. Розробник може налаштувати конфігурацію Eclipse під свої потреби, роблячи середовище максимально зручним для себе. У ньому чудово реалізована функція групової розробки, що буде особливо корисно для команд програмістів, які працюють над одним проектом.

Переваги:

- модулі: Eclipse можна гнучко налаштувати під себе завдяки простій розробці плагінів;

- мови, які підтримуються: середовище розробки підтримує роботу з декількома мовами програмування, до яких відносяться найпопулярніші - C/C++, Java;

- безкоштовність: Eclipse Java IDE поширюється безкоштовно;

- кросплатформеність: у будь-якої програми, написаної на Java, за

замовчуванням є ця перевага. Eclipse можна запустити на будь-якій платформі, на якій встановлена Java;

- системна вимога: можна комфортно працювати в середовищі навіть на слабкому комп'ютері;
- workspace: в Eclipse можна відкрити кілька проєктів в одному workspace, що значною мірою знижує з модульними проєктами або частковим перенесенням коду з одного проєкту в інший;
- документація: Eclipse - зрілий продукт, популярний у всьому світі, тому у нього є велика база документації, в якій з імовірністю 90% знайдеться відповідь на будь-яке питання;
- оформлення інтерфейсу.

Недоліки:

- не інтуїтивний інтерфейс;
- модулі: при наявності великої кількості несумісних плагінів IDE може впасти, і буде потрібна переустановка;
- налагодження: функція налагодження не так досконала, як у інших платних IDE інструментів;
- проблеми з управлінням версіями, зокрема з Git;
- неповна інтеграція з Maven і Gradle.

Xamarin – це фреймворк, що використовує мову C# для розробки кросплатформених застосунків (для iOS, Android, Windows Phone). Xamarin дозволяє створювати спільну логіку додатка за допомогою C# і .NET одночасно для всіх трьох платформ - Android, iOS і Windows Mobile.

Основним інструментом, який компанія Xamarin пропонує для розробки програм для iOS, Android та Windows Phone, є сама Xamarin, яка працює на Windows або Mac OS X. Також розробку можна вести за допомогою розширень у середовищі Microsoft Visual Studio (рис. 2.2).

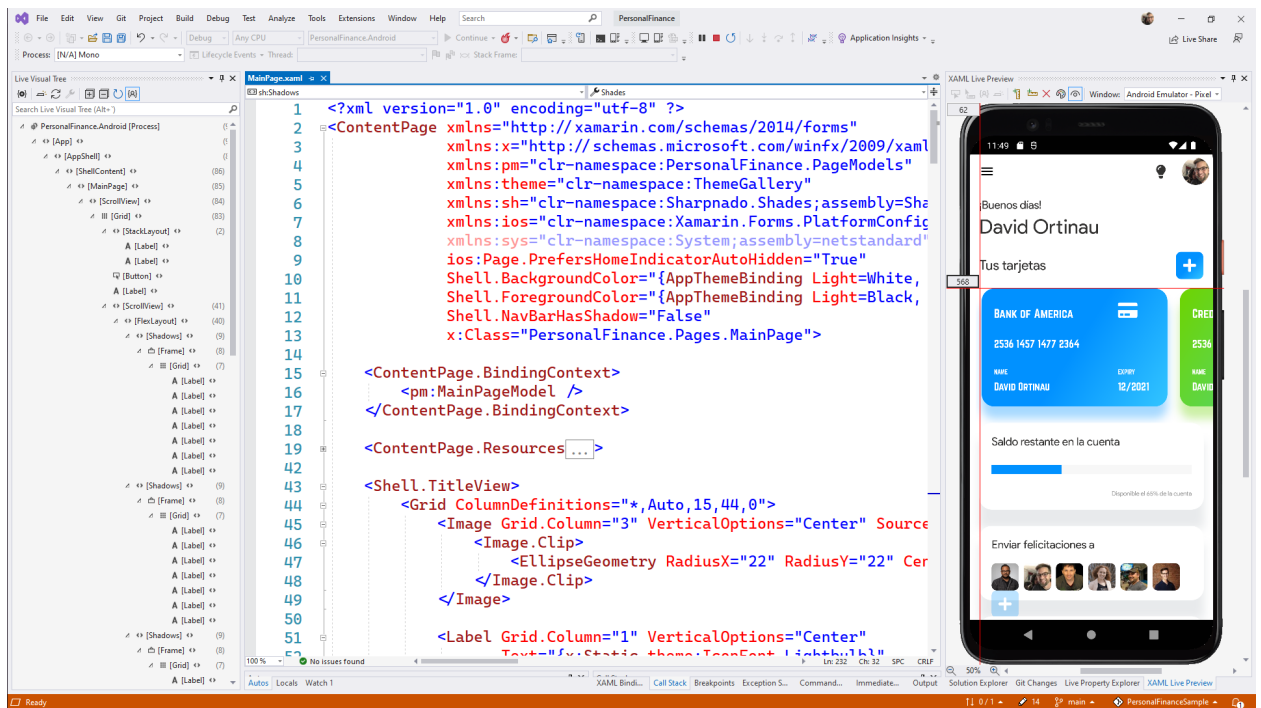


Рисунок 2.2 – Середовище розробки Visual Studio з плагіном Xamarin

Хamarin-застосунки містять у собі виконавче середовище Mono.

Під час запуску Xamarin-застосунку, виконавче середовище Mono транслює скомпільований C# код у команди ОС (iOS, Android або Windows).

Функціонально платформа Xamarin включає кілька субплатформ, зокрема:

- Xamarin.Android: бібліотеки для застосунків на ОС Android;
- Xamarin.Mac: бібліотеки для створення застосунків на Mac OS X;
- Xamarin.iOS: бібліотеки для створення застосунків для iOS;
- компілятор C #;
- .NET Framework;
- інструменти IDE (вбудовані в Visual Studio для Mac OS і Windows).

Переваги:

- кросплатформеність: створюється єдиний код для всіх платформ під час розробки;
- API: Xamarin надає прямий доступ до нативних API кожної платформи;

- документація: для Xamarin доступна відмінно організована документація з практичними прикладами, фрагментами коду і покрокові інструкції;
- сумісність з пристроями: чудово підходить для застосунків, які потребують доступу до даних про місцезнаходження користувача, показань гіроскопа, акселерометра або інших вбудованих датчиків;
- потужне тестування: середовище Test Cloud, що дозволяє емулювати більше 2000 пристроїв;
- оформлення інтерфейсу.

Недоліки:

- розмір застосунків: застосунки, створені за допомогою Xamarin, зазвичай трохи більші за обсягом порівняно з нативними аналогами;
- не підходить для застосунків з високопродуктивною графікою: багатий користувацький інтерфейс або складні анімації з невеликою кількістю спільного коду роблять Xamarin менш підходящим;
- складнощі з інтеграцією.

2.1.4 Порівняння середовищ розробки

У таблиці 2.4 наведено порівняння середовищ розробки Android застосунків. Серед характеристик для порівняння виділімо:

- графічний інтерфейс користувача (GUI): комплексна оцінка зрозумілості та інтуїтивності GUI середовища розробки для користувача;
- продуктивність: навантаження на ОС під час розробки;
- служба підтримки: комплексна оцінка для визначення рівня технічної підтримки середовища, його розробників та спільноти, що використовує середовище;
- підтримка української мови в GUI;
- підсвічування синтаксису;
- рефакторинг;

- автодоповнення коду;
- пошук по проєкту;
- налагодження;
- тестування застосунків;
- мови, які підтримуються: визначає які мови підтримує середовище розробки, що дозволить розширювати функціонал при оновленні застосунку із використанням стороннього коду.

Для розробки мобільного застосунку для нотаток було обрано середовище розробки Android Studio. Це середовище виявилось найбільш зручним і зрозумілим для роботи. Однією з головних причин вибору Android Studio є його підтримка мови програмування Kotlin, яка значно полегшує процес розробки завдяки своїй сучасності та простоті. Android Studio надає всі необхідні інструменти для створення інтерфейсів під Android, включаючи редактор макетів, який дозволяє швидко і зручно проектувати користувацькі інтерфейси. Крім того, середовище підтримує інтеграцію з іншими інструментами розробки, такими як Git для керування версіями, що є важливим для командної роботи.

Таблиця 2.4 – Порівняння середовищ розробки

	Android Studio	Eclipse	Xamarin (Visual Studio)
1	2	3	4
GUI	++	++	++
Продуктивність	+	++	+
Служба підтримки	+	+	+
Українська мова	-	+	+
Підсвічування синтаксису	+	+	+
Рефакторинг	+	+	+

Продовження таблиці 2.4

1	2	3	4
Автодоповнення коду	+	+	+
Пошук по проєкту	+	+	+
Налагодження	+	+	+
Тестування застосунків	+	+	+
Мови, які підтримуються	Java, Kotlin, C++	Java, C++	C#

2.2 Загальна структура застосунку

Для розробки мобільного застосунку для нотаток з автогенерацією було обрано наступні технології:

а) MVC (Model-View-Controller):

Цей архітектурний шаблон складається з:

- Model (Модель): відповідає за дані та логіку застосунку. Включає в себе класи для представлення нотаток, файлів, користувачів та іншої інформації. Також містить логіку для роботи з даними, наприклад, завантаження, збереження, синхронізацію та автогенерацію нотаток;
- View (Вигляд): відповідає за відображення даних користувачеві. Включає в себе екрани застосунку, елементи керування та макети. View отримує дані з Model та відображає їх користувачеві;
- Controller (Контролер): відповідає за обробку дій користувача та оновлення Model та View. Controller отримує події від View, наприклад, натискання кнопок або введення тексту, та оновлює Model відповідним чином. Після оновлення Model, Controller оновлює View, щоб відобразити зміни користувачеві.

Переваги MVC:

- розділення відповідальності: MVC дозволяє розділити логіку застосунку на окремі компоненти, що спрощує розробку, тестування та підтримку;
- гнучкість: MVC дозволяє легко змінювати та розширювати функціонал застосунку без впливу на інші компоненти;
- повторне використання коду: MVC дозволяє повторно використовувати компоненти Model та View в різних частинах застосунку.

На рисунку 2.3 показано схему, що ілюструє взаємодію між компонентами Model, View та Controller та який вигляд має архітектура мобільного застосунку системи на основі предметної області.

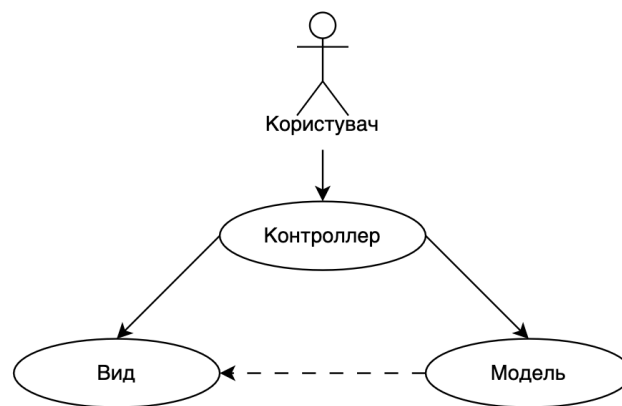


Рисунок 2.3 – Архітектура застосунку MVC

б) Firebase:

Firebase - це хмарна платформа, розроблена компанією Google, яка надає широкий спектр сервісів для розробки мобільних та веб- застосунків.

Переваги Firebase:

- простота використання: Firebase надає простий та зручний API для роботи з даними та іншими сервісами;
- масштабованість: Firebase автоматично масштабується відповідно до потреб застосунку;
- безпека: Firebase забезпечує високий рівень безпеки даних;
- комплексність: Firebase пропонує всі необхідні сервіси для

розробки мобільного застосунку, такі як БД (Firestore), сховище файлів (Storage), аутентифікація (Authentication) та інше.

в) GPT-3:

GPT-3 (Generative Pre-trained Transformer 3) - це велика мовна модель, розроблена компанією OpenAI.

Переваги GPT-3:

- висока якість згенерованого тексту: GPT-3 здатна генерувати текст, який є граматично правильним, зв'язним та відповідає заданим параметрам;
- гнучкість: GPT-3 дозволяє генерувати нотатки різних типів та стилів;
- доступність: GPT-3 доступна через API, що дозволяє легко інтегрувати її в мобільний застосунок.

Вибір MVC, Firebase та GPT-3 дозволяє створити гнучкий, масштабований та функціональний застосунок для нотаток з автогенерацією, який забезпечує надійне зберігання, синхронізацію та безпеку даних, а також надає користувачам унікальні можливості для створення контенту.

Firebase було обрано завдяки його простоті використання, масштабованості, безпеці та комплексності сервісів. GPT-3 було обрано завдяки його високій якості згенерованого тексту, гнучкості та доступності.

2.3 Висновки за розділом 2

У другому розділі було розглянуто архітектуру мобільного застосунку для нотаток з автогенерацією. Було обрано архітектурний шаблон MVC, який дозволяє розділити логіку застосунку на окремі компоненти, забезпечуючи гнучкість, масштабованість та зручність розробки.

Для реалізації мобільного застосунку було обрано наступні технології:

- Kotlin: сучасна мова програмування, що добре підходить для розробки Android- застосунків;

- Android Studio: потужне середовище розробки, яке надає всі необхідні інструменти для створення Android- застосунків;
- Firebase: хмарна платформа, яка надає широкий спектр сервісів, включаючи базу даних (Firestore), сховище файлів (Storage), аутентифікацію (Authentication) та інше;
- GPT-3: велика мовна модель, яка буде використовуватися для автогенерації нотаток.

Вибір цих технологій дозволить створити надійний, ефективний та масштабований мобільний застосунок, який відповідає сучасним вимогам користувачів.

3 ОСНОВНІ РІШЕННЯ З РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Реалізація інтерфейсу користувача

Інтерфейс користувача мобільного застосунку для нотаток з автогенерацією розроблений з урахуванням принципів простоти, зручності та інтуїтивності. Застосунок має наступні екрани:

а) LoginActivity – стартовий екран (рис. 3.1). Екран авторизації користувача:

- 1) поля вводу email та паролю: дозволяють користувачеві ввести свої дані для входу в застосунок;
- 2) кнопка "Log In": авторизує користувача в застосунку;
- 3) кнопка "Sign Up": перенаправляє користувача на екран реєстрації;

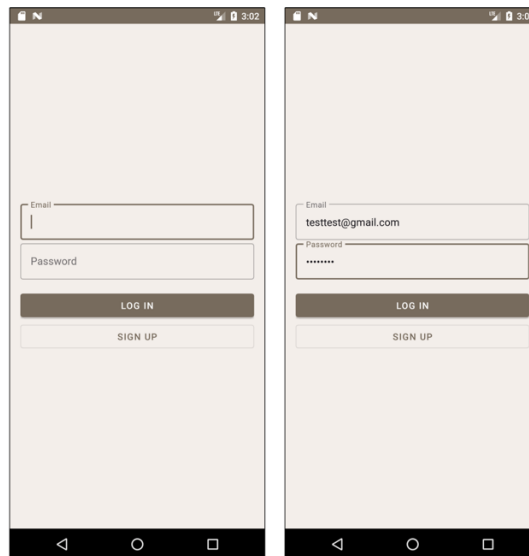


Рисунок 3.1 – Скріншоти екрану авторизації

б) SignupActivity – екран реєстрації користувача (рис 3.2):

- 1) поля вводу email та паролю: дозволяють користувачеві ввести свої дані для реєстрації в застосунку;
- 2) кнопка "Sign Up": реєструє користувача в застосунку;

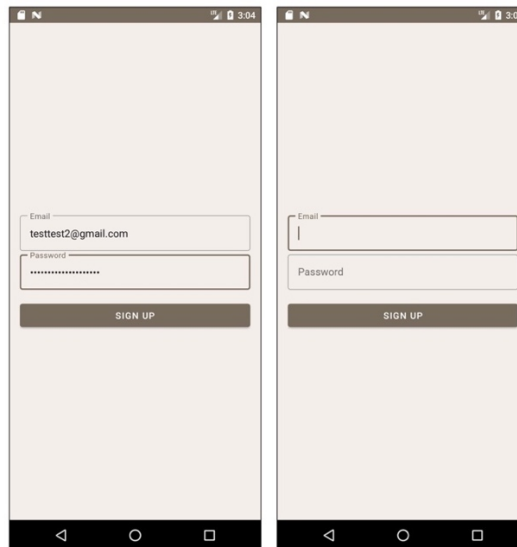


Рисунок 3.2 – Скріншоти екрану реєстрації

в) MainActivity – головний екран (рис. 3.3):

- 1) список нотаток: відображає список всіх нотаток користувача з коротким описом (назва, дата створення);
- 2) пошуковий рядок: дозволяє шукати нотатки за ключовими словами;
- 3) кнопка "Add Note": відкриває екран створення нової нотатки;
- 4) кнопка "Сортування": дозволяє сортувати нотатки за різними параметрами (назва, дата створення, дата останньої зміни);
- 5) кнопка "Налаштування": відкриває екран налаштувань застосунку;

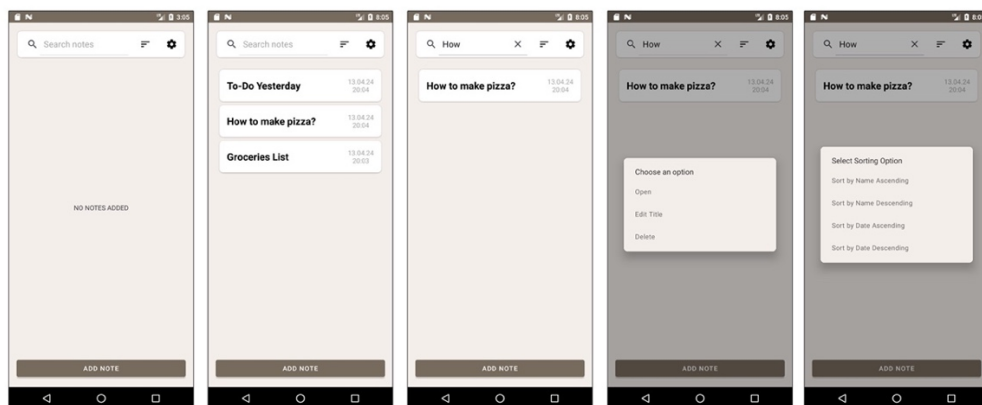


Рисунок 3.3 – Скріншоти головного екрану додатка

г) NoteActivity – екран редагування нотатки (рис. 3.4):

- 1) текстове поле: дозволяє ввести текст нотатки;
- 2) кнопка "Add File": дозволяє прикріпити файл до нотатки;
- 3) список прикріплених файлів: відображає список прикріплених до нотатки файлів;
- 4) кнопка "Згенерувати текст": відкриває діалогове вікно для автогенерації тексту;

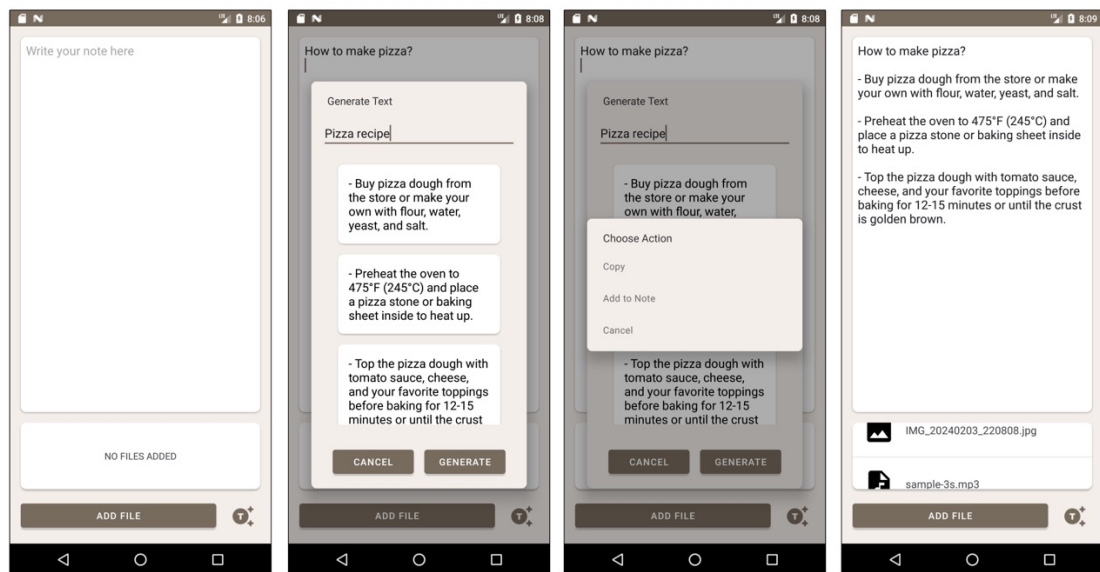


Рисунок 3.4 – Скріншоти екрану редагування нотатки

д) SettingsActivity – екран налаштувань додатка (рис 3.5):

- 1) кнопка "Log Out": дозволяє користувачеві вийти зі свого облікового запису;
- 2) кнопка "Delete Account": дозволяє користувачеві видалити свій обліковий запис;
- 3) кнопка "Clear Cache": дозволяє користувачеві очистити кеш застосунку.

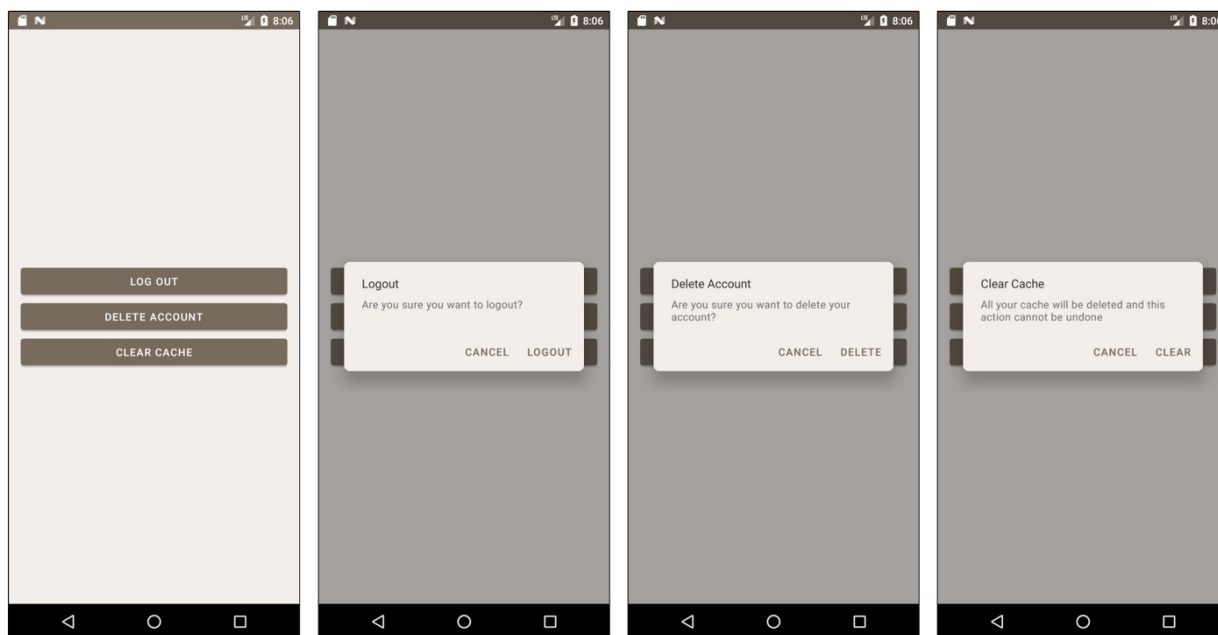


Рисунок 3.5 – Скріншоти екрану налаштувань

Дизайн та стиль інтерфейсу:

- мінімалістичний дизайн: інтерфейс застосунку розроблений в мінімалістичному стилі, щоб не відволікати користувача від основного завдання - роботи з нотатками;
- матеріальний дизайн: застосунок використовує принципи Material Design, що забезпечує сучасний та привабливий вигляд;
- адаптивний дизайн: інтерфейс застосунку адаптується до різних розмірів екранів, що забезпечує зручність використання на різних пристроях.

3.2 Реалізація моделі даних

Модель даних мобільного застосунку для нотаток з автогенерацією складається з наступних класів та компонентів:

- а) `FileItem`. Цей клас представляє файл, прикріплений до нотатки. Дані та методи класу наведені у таблиці 3.1;

Таблиця 3.1 – Опис полів та методів класу FileItem.kt

Поля та методи класу	Опис
public:	
path	Шлях до файлу.
iconResId	Ідентифікатор ресурсу зображення для файлу.

б) Note. Цей клас представляє нотатку користувача. Дані та методи класу наведені у таблиці 3.2;

Таблиця 3.2 – Опис полів та методів класу Note.kt

Поля та методи класу	Опис
public:	
id	Ідентифікатор нотатки.
title	Заголовок нотатки.
text	Текст нотатки.
timestamp	Відмітка часу створення нотатки (в мілісекундах).
files	Список шляхів до файлів, пов'язаних з нотаткою.
toString()	Повертає заголовок нотатки у вигляді рядка.

в) FileManager. Він відповідає за управління файлами, прикріпленими до нотаток. Дані та методи класу наведені у таблиці 3.3;

Таблиця 3.3 – Опис полів та методів класу FileManager.kt

Поля та методи класу	Опис
1	2
private:	
noteManager	Менеджер нотаток, з яким взаємодіє FileManager.
adapter	Адаптер для файлів, з яким взаємодіє FileManager.
noteId	Ідентифікатор нотатки, для FileManager.

Продовження таблиці 3.3

1	2
storageManager	Менеджер для завантаження та видалення файлів.
public:	
isImageFile(file: File)	Перевіряє, чи є файл зображенням.
isVideoFile(file: File)	Перевіряє, чи є файл відео.
isAudioFile(file: File)	Перевіряє, чи є файл аудіо.
getMimeType(file: File)	Отримує MIME-тип файлу.
copyFile(inputStream: InputStream, outputStream: FileOutputStream)	Копіює вміст файлу з одного потоку в інший.
deleteFile(filePath: String)	Видаляє файл за вказаним шляхом.
getImageIdUsingPath(file: File)	Отримує ідентифікатор зображення для файлу на основі його типу.
openFile(filePath: String)	Відкриває файл за вказаним шляхом.
handleSelectedFile(uri: Uri)	Обробляє обраний файл з вказаним URI.
getFileExtension(fileName: String)	Отримує розширення файлу.
getFileNameWithoutExtension(fileName: String)	Отримує назву файлу без розширення.
getFileName(uri: Uri)	Отримує назву файлу за його URI.
openLocalFile(file: File)	Відкриває локальний файл.

г) `FireStorageManager`. Цей клас відповідає за завантаження файлів до та з `Firebase Storage`. Дані та методи класу наведені у таблиці 3.4;

Таблиця 3.4 – Опис полів та методів класу `FireStorageManager.kt`

Поля та методи класу	Опис
private:	
<code>storageRef</code>	Посилання на <code>Firebase Storage</code> , яке використовується для зберігання файлів.
<code>auth</code>	Об'єкт <code>FirebaseAuth</code> для аутентифікації користувача.
<code>storage</code>	Об'єкт <code>FirebaseStorage</code> для роботи з <code>Firebase Storage</code> .
public:	
<code>uploadFileToFirebaseStorage(fileUri: Uri, fileName: String, noteId: String)</code>	Завантажує файл на <code>Firebase Storage</code> .
<code>downloadFileFromFirebaseStorage(file: File, noteId: String)</code>	Завантажує файл з <code>Firebase Storage</code> .
<code>deleteUserUploadFileFromFirebaseStorage(noteId: String, fileName: String)</code>	Видаляє завантажений користувачем файл з <code>Firebase Storage</code> .
<code>deleteAllUserUploadsFromFirebaseStorage()</code>	Видаляє всі файли, завантажені користувачем, з <code>Firebase Storage</code> .

д) `NoteManager`. Цей клас відповідає за управління нотатками, включаючи збереження, завантаження та синхронізацію. Дані та методи класу наведені у таблиці 3.5;

Таблиця 3.5 – Опис полів та методів класу NoteManager.kt

Поля та методи класу	Опис
private:	
auth	Об'єкт FirebaseAuth для аутентифікації користувача.
firebaseFirestore	Об'єкт FirebaseFirestore для роботи з Firebase Firestore.
public:	
updateNoteTitle(noteId: String?, newTitle: String)	Оновлює заголовок нотатки в Firestore.
deleteAllUserNotesFromFirestore(userId: String?)	Видаляє всі нотатки користувача з Firestore.
deleteNoteFromFirestore(noteId: String)	Видаляє нотатку з Firestore.
saveNoteToFirestore(doOnSuccess: (id: String) -> Unit)	Зберігає нову нотатку в Firestore.
retrieveNotes()	Отримує список нотаток з Firestore та оновлює адаптер.
updateNoteInFirestore(noteId: String)	Оновлює нотатку в Firestore.
loadNoteData(noteId: String, fileManager: FileManager)	Завантажує дані про нотатку з Firestore та відображає їх у віджетах.

е) OpenAIChatTask. Цей клас відповідає за взаємодію з GPT-3 API для автогенерації тексту. Дані та методи класу наведені у таблиці 3.6.

Таблиця 3.6 – Опис полів та методів класу OpenAIChatTask.kt

Поля та методи класу	Опис
private:	
userNote	Текстове повідомлення, введене користувачем.
doLayoutFill	Функція, яка заповнює макет із результатом.
public:	
doInBackground(vararg params: Void?): String?	Виконує фоновий процес, відправляючи запит до GPT-3 API з заданими параметрами та отримує згенерований текст.
onPostExecute(result: String?)	Обробляє результат виконання фонового процесу та відображає його користувачеві.

Модель даних мобільного застосунку для нотаток з автогенерацією забезпечує ефективне управління даними, включаючи зберігання, завантаження, синхронізацію та автогенерацію нотаток.

3.3 Інтеграція з Firebase

Firebase - це хмарна платформа, що надає різноманітні сервіси для розробки мобільних та веб- застосунків. У мобільному застосунку для нотаток з автогенерацією Firebase використовується для зберігання та синхронізації даних, зберігання файлів та аутентифікації користувачів.

а) інтеграція з Firebase Firestore (рис. 3.6):

1) створення проекту Firebase: першим кроком є створення проекту Firebase в консолі Firebase;

2) підключення Firebase до Android Studio: далі необхідно підключити Firebase до проекту Android Studio за допомогою Firebase Assistant;

3) створення колекції "user_notes": в Firestore створюється колекція "user_notes" для зберігання нотаток користувачів;

4) створення документів для нотаток: кожна нотатка зберігається як окремий документ в колекції "user_notes". Документ містить поля для назви нотатки, тексту, часу створення, списку прикріплених файлів та іншої інформації;

5) синхронізація даних: Firebase Firestore автоматично синхронізує дані між пристроями користувача та хмарним сховищем;

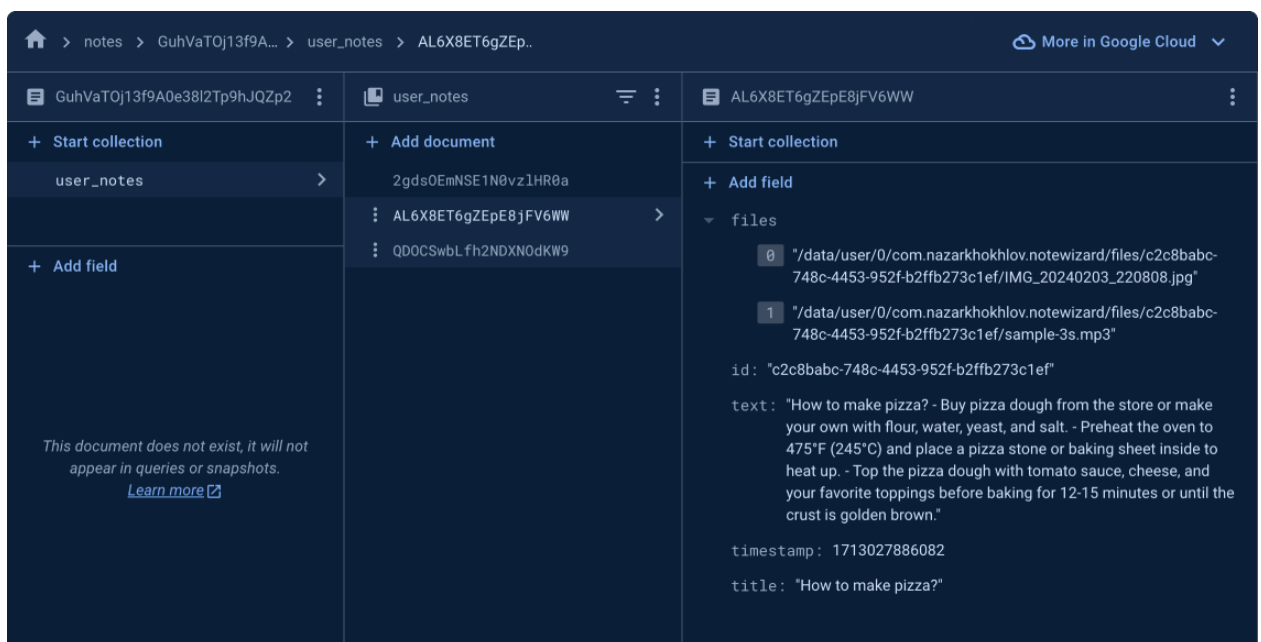


Рисунок 3.6 - Структура колекції "user_notes" в Firestore

б) інтеграція з Firebase Storage (рис. 3.7):

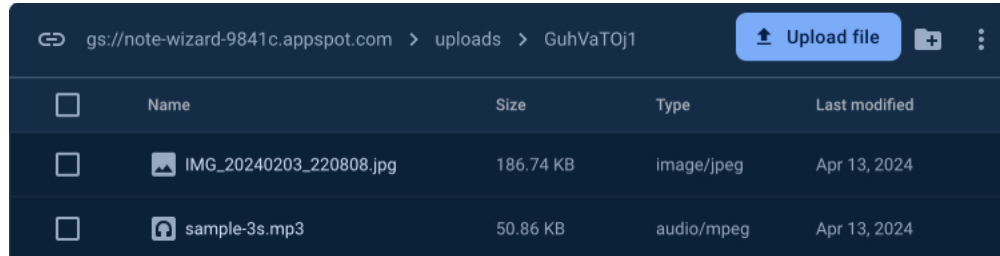
1) створення сховища: в консолі Firebase створюється сховище для зберігання файлів;

2) завантаження файлів: файли, прикріплені до нотаток, завантажуються в Firebase Storage за допомогою Firebase Storage API;

3) зберігання посилань на файли: посилання на файли

зберігаються в документах нотаток в Firestore;

4) завантаження файлів: при відкритті нотатки, прикріплені файли завантажуються з Firebase Storage;



	Name	Size	Type	Last modified
	IMG_20240203_220808.jpg	186.74 KB	image/jpeg	Apr 13, 2024
	sample-3s.mp3	50.86 KB	audio/mpeg	Apr 13, 2024

Рисунок 3.7 - Структура сховища файлів в Firebase Storage

в) інтеграція з Firebase Authentication (рис. 3.8):

1) налаштування аутентифікації: в консолі Firebase налаштовуються методи аутентифікації, такі як email та пароль;

2) реєстрація та вхід користувачів: застосунок використовує Firebase Authentication API для реєстрації та входу користувачів;

3) управління користувачами: Firebase Authentication дозволяє управляти користувачами, наприклад, скидати пароль, видаляти обліковий запис тощо.



Identifier	Providers	Created	Signed in	User UID
testtest2@gmail.com		Mar 18, 2024	Mar 19, 2024	MMzMPsFYXqeC4NmA90wO...
testtest@gmail.com		Feb 3, 2024	Apr 13, 2024	GuhVaTOj13f9A0e38l2Tp9hJ...
a@gmail.com		Jan 28, 2024	Jan 28, 2024	esL62L1yi2RyNnO3LudapBNA...

Рисунок 3.8 - Структура зареєстрованих користувачів

Інтеграція з Firebase дозволяє забезпечити надійне зберігання, синхронізацію та безпеку даних користувачів, а також спрощує розробку та

підтримку мобільного застосунку.

3.4 Реалізація функціоналу автогенерації нотаток

Функціонал автогенерації нотаток в мобільному застосунку реалізований за допомогою інтеграції з GPT-3 API, що дозволяє генерувати текст на основі заданих параметрів.

а) інтеграція з GPT-3 API:

1) отримання API ключа: для використання GPT-3 API необхідно отримати API ключ від OpenAI;

2) використання бібліотеки OkHttp: для взаємодії з GPT-3 API використовується бібліотека OkHttp, яка дозволяє надсилати HTTP-запити та отримувати відповіді;

3) формування запиту: запит до GPT-3 API формується у форматі JSON та містить параметри: "model" - назва моделі GPT-3, яка буде використовуватися для генерації тексту. В цьому випадку - "gpt-3.5-turbo", та "messages" - масив повідомлень, що містить контекст для генерації тексту;

б) алгоритм автогенерації нотаток:

1) користувач вводить текст в поле вводу на екрані створення/редагування нотатки;

2) користувач натискає кнопку "Згенерувати текст";

3) застосунок формує запит до GPT-3 API з текстом, введеним користувачем, як контекст;

4) застосунок надсилає запит до GPT-3 API та отримує згенерований текст;

5) застосунок додає згенерований текст до списку, та відображає користувачу.

Процес від напису тексту та до отримання результатів користувачем відображено на рис. 3.9.

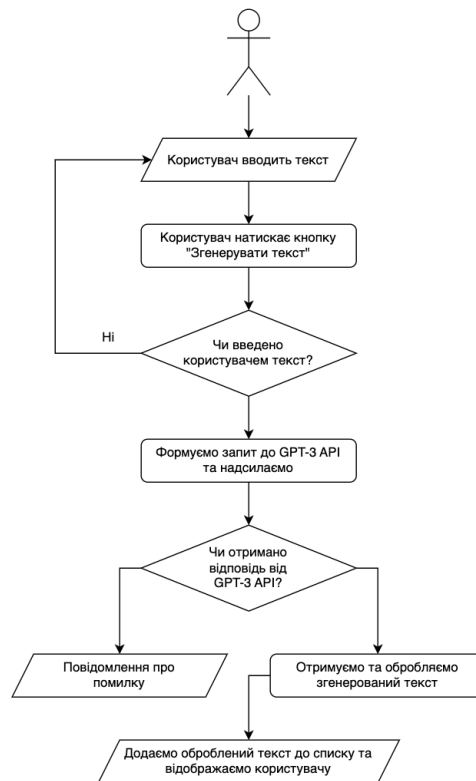


Рисунок 3.9 - Процес автогенерації нотаток, включаючи взаємодію з GPT-3 API

3.5 Реалізація пошуку та сортування нотаток

Для зручної навігації та організації нотаток в мобільному застосунку реалізовано функціонал пошуку та сортування.

а) пошук нотаток за ключовими словами:

1) використання `SearchView`: для пошуку нотаток використовується компонент `SearchView`, який дозволяє користувачеві вводити ключові слова;

2) фільтрація списку нотаток: при введенні ключових слів в `SearchView`, список нотаток фільтрується в реальному часі, відображаючи лише ті нотатки, в яких є введені ключові слова;

3) алгоритм пошуку (рис. 3.10):

— користувач вводить ключові слова в `SearchView`;

- застосунок отримує список всіх нотаток з локального кешу;
- застосунок фільтрує список нотаток, залишаючи лише ті, які містять введені ключові слова в назві;
- застосунок оновлює список нотаток на екрані, відображаючи лише відфільтровані нотатки;

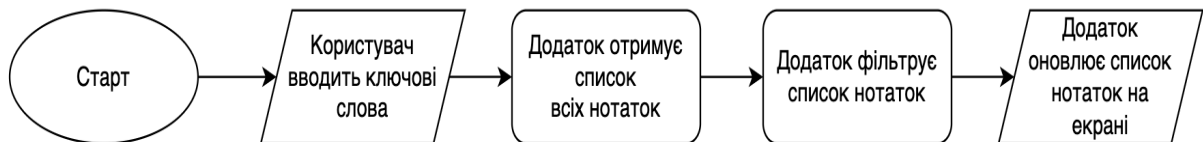


Рисунок 3.10 - Алгоритм пошуку нотатки

б) сортування нотаток (рис. 3.11):

- 1) сортування за назвою: нотатки можна сортувати за назвою в алфавітному порядку (зростання або спадання);
- 2) сортування за датою створення: нотатки можна сортувати за датою створення (від найновіших до найстаріших або навпаки);
- 3) алгоритм сортування:
 - користувач вибирає параметр сортування (назва або дата створення) та порядок сортування (зростання або спадання);
 - застосунок отримує список всіх нотаток з локального кешу;
 - застосунок сортує список нотаток за вибраним параметром та порядком сортування;
 - застосунок оновлює список нотаток на екрані, відображаючи відсортовані нотатки.

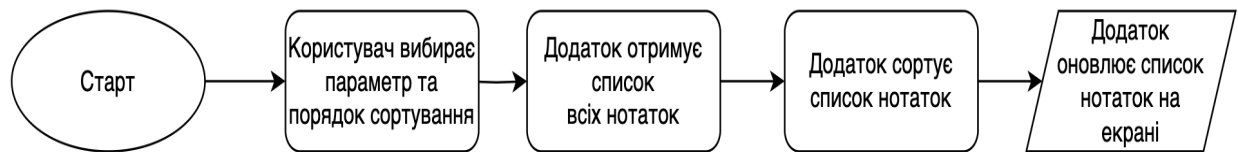


Рисунок 3.11 - Алгоритм сортування нотаток

3.6 Висновки за розділом 3

У третьому розділі було детально описано реалізацію основних компонентів мобільного застосунку для нотаток з автогенерацією. Було розглянуто реалізацію інтерфейсу користувача, моделі даних, інтеграцію з Firebase та функціонал автогенерації нотаток на основі GPT-3.

Розроблено простий та інтуїтивно зрозумілий інтерфейс, що складається з п'яти екранів: LoginActivity, SignupActivity, MainActivity, NoteActivity та SettingsActivity. Інтерфейс адаптований до різних розмірів екранів та використовує принципи Material Design.

Реалізовано модель даних, що складається з класів FileItem, Note, FileManager, FireStorageManager, NoteManager та OpenAIChatTask. Ці класи забезпечують ефективне управління даними, включаючи зберігання, завантаження, синхронізацію та автогенерацію нотаток.

Застосунок інтегровано з Firebase для зберігання та синхронізації даних (Firestore), зберігання файлів (Storage) та аутентифікації користувачів (Authentication). Це забезпечує надійність, безпеку та масштабованість застосунку.

Реалізовано функціонал автогенерації нотаток на основі GPT-3 API. Користувачі можуть генерувати текст на основі заданих параметрів, що розширює можливості для створення контенту.

Реалізовано функціонал пошуку нотаток за ключовими словами та сортування за назвою та датою створення. Це дозволяє користувачам легко знаходити потрібні нотатки та організовувати їх зручним для себе способом.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Опис застосування

Розроблений мобільний застосунок для нотаток з автогенерацією призначений для зручного створення, редагування, зберігання та організації особистих нотаток, а також для генерації нового текстового контенту на основі введеного користувачем запиту.

Основні функції застосунку:

- аутентифікація: користувачі можуть зареєструватися та увійти в застосунок, використовуючи email та пароль;
- створення та редагування нотаток: застосунок дозволяє створювати нові нотатки, редагувати існуючі, додавати текст, прикріплювати файли (зображення, відео, аудіо, документи);
- організація нотаток: користувачі можуть шукати нотатки за ключовими словами, сортувати їх за назвою та датою створення (в порядку зростання та спадання);
- автогенерація нотаток: застосунок використовує GPT-3 API для генерації нового текстового контенту на основі введеного користувачем тексту;
- зберігання та синхронізація: нотатки та файли зберігаються в хмарному сховищі Firebase та синхронізуються між пристроями користувача;
- налаштування: користувачі можуть вийти з облікового запису, видалити обліковий запис та очистити кеш застосунку.

Застосунок розроблений з урахуванням потреб користувачів, які шукають зручний та ефективний інструмент для роботи з нотатками та створення нового контенту.

4.2 Умови виконання програми

Для коректної роботи мобільного застосунку для нотаток з автогенерацією необхідно дотримуватися наступних умов:

а) мінімальні системні вимоги:

1) операційна система: Android 7.0 (Nougat) або вище;

2) апаратне забезпечення:

– процесор: 1.2 ГГц двоядерний процесор або вище;

– оперативна пам'ять (RAM): 1 ГБ або більше;

– внутрішня пам'ять: 100 МБ вільного місця або більше;

– екран: роздільна здатність 800x480 пікселів або вище;

3) підключення до Інтернету: для синхронізації даних з Firebase та використання GPT-3 API необхідне стабільне підключення до Інтернету;

б) обмеження:

1) обмеження GPT-3 API: використання GPT-3 API має обмеження на кількість запитів та обсяг згенерованого тексту;

2) обмеження Firebase: Firebase має обмеження на обсяг зберігання даних та трафік.

Дотримання цих умов забезпечить стабільну та ефективну роботу мобільного застосунку для нотаток з автогенерацією.

4.3 Інструкція по експлуатації програми

4.3.1 Характеристики програми

Мобільний застосунок для нотаток з автогенерацією має наступні характеристики:

а) архітектура:

1) Model-View-Controller (MVC): архітектурний шаблон, що розділяє логіку застосунку на окремі компоненти, забезпечуючи гнучкість, масштабованість та зручність розробки;

б) технології:

1) Kotlin: сучасна мова програмування, що добре підходить для розробки Android- застосунків;

2) Android Studio: потужне середовище розробки, яке надає всі необхідні інструменти для створення Android- застосунків;

3) Firebase: хмарна платформа, яка надає широкий спектр сервісів, включаючи базу даних (Firestore), сховище файлів (Storage), аутентифікацію (Authentication) та інше;

4) GPT-3: велика мовна модель, яка використовується для автогенерації нотаток;

5) OkHttp: бібліотека для взаємодії з GPT-3 API;

в) функціональність:

1) аутентифікація: реєстрація та вхід користувачів за допомогою email та паролю;

2) створення та редагування нотаток: додавання тексту, прикріплення файлів, форматування тексту;

3) організація нотаток: пошук за ключовими словами, сортування за назвою та датою створення;

4) автогенерація нотаток: генерація нового текстового контенту на основі існуючого тексту нотатки та заданих параметрів;

5) зберігання та синхронізація: зберігання нотаток та файлів в хмарному сховищі Firebase та синхронізація між пристроями користувача;

6) налаштування: вихід з облікового запису, видалення облікового запису, очищення кешу;

г) інтерфейс користувача:

1) мінімалістичний дизайн: інтерфейс розроблений в мінімалістичному стилі, щоб не відволікати користувача від основного

завдання - роботи з нотатками;

2) матеріальний дизайн: застосунок використовує принципи Material Design, що забезпечує сучасний та привабливий вигляд;

3) адаптивний дизайн: інтерфейс адаптується до різних розмірів екранів, що забезпечує зручність використання на різних пристроях;

д) безпека:

1) Firebase Authentication: забезпечує безпечну аутентифікацію користувачів;

2) правила безпеки Firestore: контролюють доступ до даних в Firestore;

3) шифрування даних: Firebase автоматично шифрує дані, що зберігаються в Firestore та Storage.

Мобільний застосунок для нотаток з автогенерацією - це надійний, ефективний та зручний інструмент для роботи з інформацією та створення нового контенту.

4.3.2 Перелік файлів, у яких знаходиться програма

Мобільний застосунок для нотаток з автогенерацією складається з наступних основних файлів та папок:

а) файли Kotlin:

1) MainActivity.kt: містить код головного екрану застосунку;

2) LoginActivity.kt: містить код екрану входу;

3) SignupActivity.kt: містить код екрану реєстрації;

4) NoteActivity.kt: містить код екрану створення/редагування

нотатки;

5) SettingsActivity.kt: містить код екрану налаштувань;

6) NoteManager.kt: містить код класу NoteManager, який відповідає за управління нотатками;

7) FileManager.kt: містить код класу FileManager, який

відповідає за управління файлами;

8) `FireStorageManager.kt`: містить код класу `FireStorageManager`, який відповідає за завантаження та завантаження файлів з `Firebase Storage`;

9) `OpenAIChatTask.kt`: містить код класу `OpenAIChatTask`, який відповідає за взаємодію з GPT-3 API;

10) `Note.kt`: містить код класу `Note`, який представляє нотатку;

11) `FileItem.kt`: містить код класу `FileItem`, який представляє файл, прикріплений до нотатки;

12) `Constants.kt`: містить константи та допоміжні функції;

б) файли XML:

1) `activity_main.xml`: макети головного екрану;

2) `activity_login.xml`: макети екрану входу;

3) `activity_signup.xml`: макети екрану реєстрації;

4) `activity_note.xml`: макети екрану редагування нотатки;

5) `activity_settings.xml`: макети екрану налаштувань;

6) `list_item_note.xml`: макети елемента списку нотаток;

7) `list_item_file.xml`: макети елемента списку файлів;

в) інші файли:

1) `AndroidManifest.xml`: файл маніфесту застосунку;

2) `build.gradle (Module: app)`: файл конфігурації Gradle для модуля застосунку;

3) `google-services.json`: файл конфігурації Firebase;

г) Папки:

1) `drawable`: папка з ресурсами зображень;

2) `values`: папка з ресурсами рядків, кольорів та стилів.

Цей перелік файлів та папок представляє основну структуру мобільного застосунку для нотаток з автогенерацією.

4.3.3 Звернення до програми

Оскільки мобільний застосунок для нотаток з автогенерацією наразі не доступний в Google Play Store, для його встановлення та запуску необхідно виконати наступні кроки:

а) отримати APK-файл: отримайте APK-файл застосунку від розробника;

б) дозволити встановлення з невідомих джерел: на своєму Android-пристрої перейдіть до налаштувань безпеки та дозвольте встановлення застосунків з невідомих джерел;

в) встановити застосунок: відкрийте APK-файл та дотримуйтесь інструкцій на екрані для встановлення застосунку;

г) запустити застосунок: після встановлення знайдіть іконку застосунку на головному екрані або в списку застосунків та натисніть на неї, щоб запустити застосунок;

д) увійти або зареєструватися: якщо ви вже маєте обліковий запис, введіть свій email та пароль на екрані входу (LoginActivity). Якщо ви новий користувач, натисніть кнопку "Sign Up" та створіть новий обліковий запис, вказавши свій email та пароль;

е) почати роботу з нотатками: після входу в застосунок ви потрапите на головний екран (MainActivity), де відображається список ваших нотаток. Ви можете створювати нові нотатки, редагувати існуючі, шукати нотатки, сортувати їх та використовувати функцію автогенерації тексту при редагуванні нотатки.

Застосунок не вимагає додаткових налаштувань або конфігурації. Всі дані зберігаються та синхронізуються автоматично за допомогою Firebase.

4.3.4 Вхідні й вихідні дані

Вхідні дані:

- дані користувача: email та пароль для аутентифікації;
- дані нотаток: назва, текст, час створення, список прикріплених файлів;
- дані файлів: вміст файлів, прикріплених до нотаток;
- параметри автогенерації: текст для автогенерації нотатки;
- ключові слова для пошуку: слова або фрази, що використовуються для пошуку нотаток;
- параметри сортування: назва або дата створення, порядок сортування (зростання або спадання).

Вихідні дані:

- список нотаток: відображення списку нотаток з назвою та датою створення;
- вміст нотатки: відображення тексту нотатки, прикріплених файлів та іншої інформації;
- згенерований текст: текст, згенерований GPT-3 API на основі заданих параметрів;
- відфільтрований список нотаток: список нотаток, що відповідають заданим ключовим словам;
- відсортований список нотаток: список нотаток, відсортований за назвою або датою створення.

4.4 Методика та результати тестування

Тестування мобільного застосунку для нотаток з автогенерацією проводилося в декілька етапів, включаючи тестування функціональності, юзабіліті та продуктивності.

4.4.1 Тестування функціоналу

Тестування функціональності розпочалося з перевірки процесу аутентифікації. Користувачі успішно реєструвалися та входили в застосунок, використовуючи email та пароль (рис. 4.1).

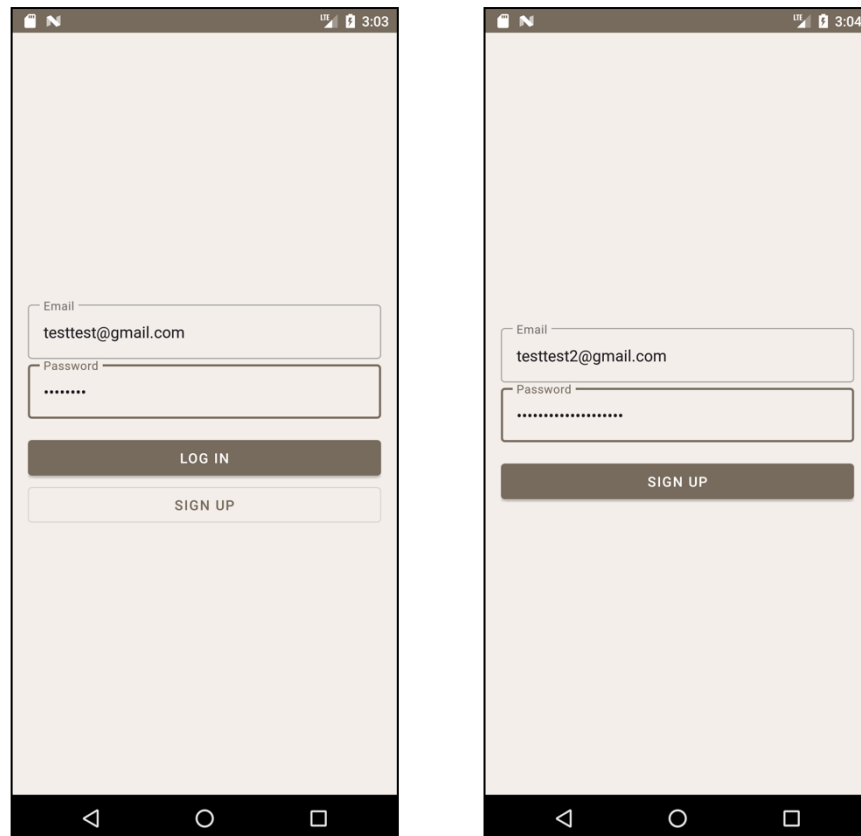


Рисунок 4.1 – Екрани авторизації та реєстрації

Далі було протестовано функціонал створення та редагування нотаток. Користувачі могли додавати текст, змінювати його, прикріплювати файли та зберігати нотатки (рис. 4.2).

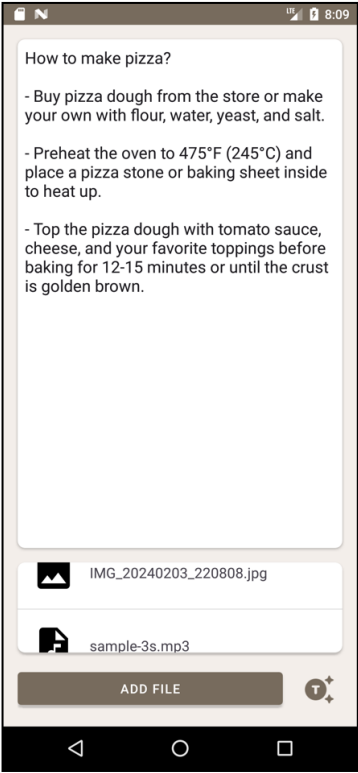


Рисунок 4.2 – Екран створеної та заповненої нотатки

Функціонал пошуку та сортування нотаток також працював коректно. Користувачі могли знаходити потрібні нотатки за ключовими словами та сортувати їх за назвою та датою створення (рис. 4.3).

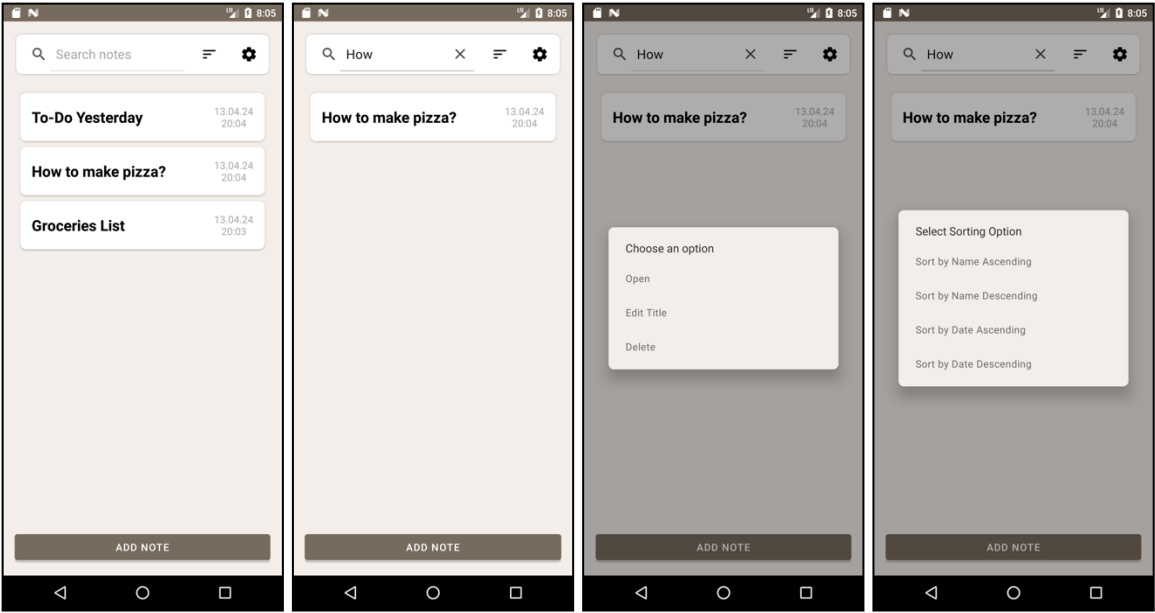


Рисунок 4.3 – Головний екран з демонстрацією функціоналу пошуку та сортування нотаток

Функція автогенерації нотаток успішно генерувала новий текстовий контент на основі існуючого тексту нотатки та заданих параметрів (рис. 4.4).

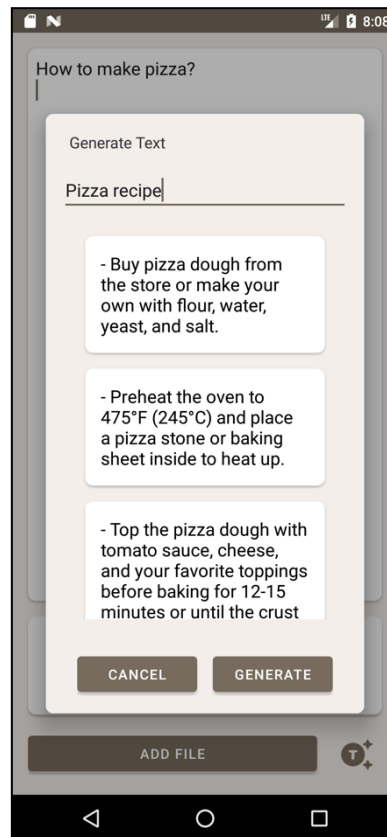


Рисунок 4.4 – Екран редагування нотатки з екраном для автогенерації тексту

4.4.2 Тестування юзабіліті

Для оцінки зручності інтерфейсу було проведено тестування з кількома користувачами. Вони виконували типові завдання, такі як створення нотаток, пошук та сортування, а також використання функції автогенерації. Загалом, користувачі знайшли інтерфейс інтуїтивно зрозумілим та простим у використанні. Було виявлено кілька незначних проблем з дизайном, які були виправлені для покращення загального досвіду користувача.

4.4.3 Тестування продуктивності

Застосунок було протестовано на різних пристроях, щоб оцінити його швидкість та ефективність використання ресурсів. Тестування включало вимірювання часу запуску, завантаження нотаток, автогенерації тексту та використання пам'яті та процесора. Результати показали, що застосунок працює швидко та плавно, навіть при роботі з великою кількістю нотаток. Споживання ресурсів було оптимальним, що забезпечує тривалий час роботи від батареї. Результати тестування продуктивності наведені в таблиці 4.1.

Таблиця 4.1 - Результати тестування продуктивності

Тест	Середній результат
Час запуску	0.9 секунди
Час завантаження нотаток (50 шт.)	0.73 секунди
Час генерації тексту	1.45 секунди
Використання CPU	0%-31% (залежить від навантаження)

ВИСНОВКИ

У дипломній кваліфікаційній роботі бакалавра було розглянуто проблему ефективного зберігання та організації особистої інформації, а також можливості використання технологій штучного інтелекту для покращення процесу створення контенту. Для цього було виконано наступні завдання:

- аналіз предметної області: проведено аналіз існуючих мобільних застосунків для нотаток та виявлено їх переваги та недоліки;
- постановка завдання: сформульовано вимоги до розроблюваного мобільного застосунку, включаючи функціональні та нефункціональні вимоги, а також вимоги до даних;
- розробка архітектури: обрано архітектурний шаблон та визначено основні компоненти системи;
- реалізація функціоналу: розроблено інтерфейс користувача, модель даних, інтеграцію з хмарним сховищем та функціонал автогенерації нотаток на основі великої мовної моделі;
- тестування: проведено тестування функціональності та продуктивності застосунку.

В результаті розробки було створено мобільний застосунок для нотаток з автогенерацією, який має наступні можливості:

- створення та редагування нотаток: додавання тексту, прикріплення файлів;
- організація нотаток: пошук за ключовими словами, сортування за назвою та датою створення;
- автогенерація нотаток: генерація нового текстового контенту на основі заданого тексту;
- зберігання та синхронізація: зберігання нотаток та файлів в хмарному сховищі та синхронізація між пристроями користувача;
- налаштування: вихід з облікового запису, видалення облікового

запису, очищення кешу.

Створений застосунок відповідає всім вимогам, сформульованим у технічному завданні, та надає користувачам зручний та ефективний інструмент для роботи з інформацією та створення нового контенту.

Усі завдання випускної дипломної роботи бакалавра повністю виконано.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. 20 кращих додатків для збереження нотаток [Електрон. ресурс]. – Режим доступу: <https://bizmag.com.ua/dodatky-dlya-notatok/>.
2. Мобільні навчальні додатки [Електрон. ресурс]. – Режим доступу: <https://od.kubg.edu.ua/index.php/journal/article/view/511/431>.
3. ТОП-7 застосунків для створення нотаток [Електрон. ресурс]. – Режим доступу: <https://blog.comfy.ua/ua/top-7-zastosunkiv-dlya-stvorenniya-notatok/>.
4. Google Keep [Electronic resource]. – Access mode: <https://play.google.com/store/apps/details?id=com.google.android.keep>.
5. Evernote - Note Organizer [Electronic resource]. – Access mode: <https://play.google.com/store/apps/details?id=com.evernote>.
6. Microsoft OneNote [Electronic resource]. – Access mode: play.google.com/store/apps/details?id=com.microsoft.office.onenote.
7. Notion – notes, docs, tasks [Electronic resource]. – Access mode: <https://play.google.com/store/apps/details?id=notion.id>.
8. Just ask Notion AI. [Electronic resource]. – Access mode: <https://www.notion.so/product/ai>.
9. Mobile OS Market Share Worldwide [Electronic resource]. – Access mode: <https://gs.statcounter.com/os-market-share/mobile/worldwide>.
10. Що краще iOS або Android [Електрон. ресурс]. – Режим доступу: <https://blog.moyo.ua/ua/shho-krashhe-ios-abo-android-13-parametriv-os-yaki-varto-vrahovuvati-pri-vibori-gadzheta/>.
11. Kotlin або Java [Електрон. ресурс]. – Режим доступу: <https://foxminded.ua/kotlin-abo-java/>.
12. Android Studio [Electronic resource]. – Access mode: <https://developer.android.com/studio>.
13. Xamarin vs. Android Studio [Electronic resource]. – Access mode: <https://medium.com/nerd-for-tech/dab30cb11008>.

ДОДАТОК А
Технічне завдання

Вступ

Метою роботи є розробка мобільного застосунку для зберігання особистих нотаток з можливістю їх автогенерації. Застосунок повинен забезпечувати зручне створення, редагування, зберігання та організацію нотаток, а також генерацію нового текстового контенту на основі існуючих нотаток та заданих параметрів.

A.1 Підстави для розробки

Розробка виконується на основі завдання на дипломну кваліфікаційну роботу бакалавра, яке було визначене за темою "Розробка мобільного застосунку для зберігання особистих нотаток з можливістю їх автогенерації".

A.2 Призначення розробки

Завданням застосунку є надання користувачам зручного та ефективного інструменту для роботи з нотатками та створення нового контенту. Застосунок призначений для широкої аудиторії користувачів, які хочуть покращити свою продуктивність та організувати свою інформацію.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до нефункціональних характеристик

- платформа: Android 7.0 (Nougat) або вище;
- продуктивність: застосунок повинен працювати швидко та ефективно, забезпечуючи комфортне використання;
- масштабованість: застосунок повинен бути здатний обробляти великі обсяги даних та обслуговувати велику кількість користувачів;

- доступність: інтерфейс застосунку повинен бути простим, зрозумілим та зручним для користувачів з різним рівнем технічних знань;
- зручність використання: застосунок повинен мати інтуїтивну навігацію та зручний пошук.

А.3.2 Вимоги до функціональних характеристик

- аутентифікація: реєстрація та вхід користувачів за допомогою email та паролю;
- створення та редагування нотаток: додавання тексту, прикріплення файлів;
- організація нотаток: пошук за ключовими словами, сортування за назвою та датою створення;
- автогенерація нотаток: генерація нового текстового контенту на основі заданого тексту;
- зберігання та синхронізація: зберігання нотаток та файлів в хмарному сховищі Firebase та синхронізація між пристроями користувача;
- налаштування: вихід з облікового запису, видалення облікового запису, очищення кешу.

А.3.3 Вимоги до надійності

Вся інформація про користувачів системи має зберігатися в окремій базі даних на сервері, з функцією періодичного авто-зберігання та резервного копіювання. Дані нотаток та файлів повинні зберігатися надійно в хмарному сховищі та локально на пристрої користувача.

А.3.4 Умови експлуатації

Для встановлення клієнту пристрій повинен мати наступні

характеристики:

- операційна система – Android 7.0 (Nougat) або вище.

A.3.5 Вимоги до складу та параметрів технічних засобів

Для експлуатації програмного забезпечення необхідні такі програмно-технічні засоби:

- пристрій з ОС Android 7.0 (Nougat) або вище;
- процесор: 1.2 ГГц двоядерний процесор або вище;
- оперативна пам'ять (RAM): 1 ГБ або більше;
- внутрішня пам'ять: 50 МБ вільного місця або більше;
- екран: роздільна здатність екрану 800x480 пікселів або вище;
- підключення до інтернету.

A.4 Стадії та етапи розробки

Розробка мобільного застосунку для нотаток з автогенерацією буде здійснюватися в декілька стадій, кожна з яких включає в себе ряд етапів:

а) аналіз вимог та проектування:

- збір та аналіз вимог: визначення функціональних та нефункціональних вимог до застосунку, а також вимог до даних;
- проектування архітектури: вибір архітектурного шаблону, визначення компонентів застосунку та їх взаємодії;
- проектування інтерфейсу користувача: розробка макетів екранів, елементів керування та дизайну інтерфейсу;
- створення технічної документації: документування архітектури, функціоналу та інтерфейсу користувача;

б) розробка:

- реалізація інтерфейсу користувача: створення екранів застосунку, елементів керування та логіки взаємодії з користувачем;

- реалізація моделі даних: створення класів для представлення даних та логіки для роботи з ними;
 - інтеграція зі хмарним сховищем: підключення хмарного сховища до проекту та реалізація функціоналу для зберігання та синхронізації даних, зберігання файлів та аутентифікації користувачів;
 - реалізація функціоналу автогенерації нотаток: інтеграція з API великої мовної моделі та реалізація алгоритму автогенерації тексту;
 - реалізація пошуку та сортування нотаток: реалізація функціоналу пошуку нотаток за ключовими словами та сортування за назвою та датою створення;
 - реалізація додаткових функцій: реалізація реєстрації та авторизації задля захищення нотаток;
- в) тестування:
- інтеграційне тестування: тестування взаємодії між компонентами застосунку;
 - функціональне тестування: тестування функціоналу застосунку з точки зору користувача;
 - тестування продуктивності: тестування швидкості роботи та ефективності використання ресурсів;
- г) розгортання:
- підготовка до розповсюдження: створення AAB-файлу застосунку, підготовка опису та скріншотів для Google Play Store;
 - розповсюдження: публікація застосунку в Google Play Store або розповсюдження AAB або APK-файлу іншими способами;
- д) підтримка:
- виправлення помилок: виявлення та виправлення помилок, що виникають під час використання застосунку;
 - оновлення: розробка та випуск оновлень з новими функціями та виправленнями помилок.

A.5 Порядок контролю та приймання системи

Контроль якості та приймання мобільного застосунку для нотаток з автогенерацією буде здійснюватися на кожному етапі розробки:

- під час проектування: перевірка відповідності функціоналу та дизайну вимогам користувачів;
- під час розробки: тестування кожного модуля та функції на коректність роботи;
- після розробки: комплексне тестування застосунку, включаючи функціональність, продуктивність та безпеку.

Після успішного завершення всіх етапів тестування та виправлення виявлених помилок, застосунок буде вважатися готовим до використання.

ДОДАТОК Б
Текст програми

B.1 LoginActivity.kt

```
package com.nazarkhokhlov.notewizard.activities

import android.content.Intent
import android.os.Bundle
import android.widget.Button
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.google.firebase.Firebase
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.auth
import com.nazarkhokhlov.notewizard.databinding.ActivityLoginBinding

class LoginActivity : AppCompatActivity() {

    private lateinit var auth: FirebaseAuth
    private lateinit var binding: ActivityLoginBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityLoginBinding.inflate(layoutInflater)
        setContentView(binding.root)

        auth = Firebase.auth

        if (auth.currentUser != null) {
            startActivity(Intent(this, MainActivity::class.java))
            finish()
        }
    }
}
```

```

    } else {
        binding.apply {
            val loginButton: Button = btnLogin
            val signUpButton: Button = btnSignUp

            loginButton.setOnClickListener {
                val email = editTextEmail.text.toString()
                val password = editTextPassword.text.toString()

                if (email.isNotEmpty() && password.isNotEmpty()) {
                    auth.signInWithEmailAndPassword(email, password)
                        .addOnCompleteListener(this@loginActivity) { task ->
                            if (task.isSuccessful) {
                                startActivity(Intent(this@loginActivity,
MainActivity::class.java))
                                finish()
                            } else {
                                Toast.makeText(baseContext, "Authentication
failed.", Toast.LENGTH_SHORT).show()
                            }
                        }
                } else {
                    Toast.makeText(baseContext, "Please enter email and
password.", Toast.LENGTH_SHORT).show()
                }
            }

            signUpButton.setOnClickListener {
                startActivity(Intent(this@loginActivity,
SignupActivity::class.java))

```

```

        }
    }
}
}
}

```

B.2 MainActivity.kt

```
package com.nazarkhokhlov.notewizard.activities
```

```

import android.content.Intent
import android.os.Bundle
import android.text.InputType
import android.util.Base64
import android.util.Log
import android.view.View
import android.widget.EditText
import android.widget.FrameLayout
import android.widget.ListView
import android.widget.SearchView
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import com.google.android.material.dialog.MaterialAlertDialogBuilder
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.firestore.FirebaseFirestore
import com.nazarkhokhlov.notewizard.MyApplication.Companion.apiKey
import com.nazarkhokhlov.notewizard.R
import com.nazarkhokhlov.notewizard.adapters.NotesAdapter
import com.nazarkhokhlov.notewizard.databinding.ActivityMainBinding
import com.nazarkhokhlov.notewizard.managers.NoteManager

```

```
import com.nazarkhokhlov.notewizard.models.Note
import com.nazarkhokhlov.notewizard.other.Constants.showToast
import java.nio.charset.StandardCharsets

class MainActivity : AppCompatActivity() {

    private lateinit var auth: FirebaseAuth
    private lateinit var firestore: FirebaseFirestore
    private lateinit var notesAdapter: NotesAdapter
    private lateinit var noteManager: NoteManager
    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        auth = FirebaseAuth.getInstance()
        firestore = FirebaseFirestore.getInstance()

        notesAdapter = NotesAdapter(this, mutableListOf()) {
            isAdapterEmpty(it)
        }
        noteManager = NoteManager(notesAdapter = notesAdapter)

        val listViewNotes: ListView = findViewById(R.id.listViewNotes)
        listViewNotes.adapter = notesAdapter

        noteManager.retrieveNotes()
```

```
binding.btnAddNote.setOnClickListener {
    noteManager.saveNoteToFirestore { id ->
        openNote(-1, id)
    }
}
```

```
binding.btnSort.setOnClickListener {
    showSortingOptionsDialog()
}
```

```
binding.btnSettings.setOnClickListener {
    startActivity(Intent(this, SettingsActivity::class.java))
}
```

```
listViewNotes.setOnItemClickListener { _, _, position, _ ->
    val selectedNote = notesAdapter.getItem(position)
    val intent = Intent(this, NoteActivity::class.java)
    intent.putExtra("noteId", selectedNote?.id)
    startActivity(intent)
}
```

```
listViewNotes.setOnItemLongClickListener { _, _, position, _ ->
    showOptionsDialog(position)
    true
}
```

```
binding.searchView.setOnQueryTextListener(object :
SearchView.OnQueryTextListener {
    override fun onQueryTextSubmit(query: String?): Boolean {
```

```

        return false
    }

    override fun onQueryTextChanged(newText: String?): Boolean {
        notesAdapter.filter.filter(newText)
        return true
    }
})

val apiKeyRef =
    firestore.collection("apiKeys").document("9zwoQ6bVZiFXgpJXk5H5")
    apiKeyRef.get()
        .addOnSuccessListener { document ->
            if (document != null) {
                val encodedApiKey =
                    document.getString("apiKey").toString()
                val decodedBytes = Base64.decode(encodedApiKey,
                    Base64.DEFAULT)
                apiKey = String(decodedBytes, StandardCharsets.UTF_8)
            }
        }
        .addOnFailureListener { exception ->
            showToast("Error getting API key.")
            Log.e("API Key", "Error getting API key: $exception")
        }

    isAdapterEmpty(notesAdapter.isEmpty)
}

private fun isAdapterEmpty(boolean: Boolean) {

```

```

        binding.listViewNotes.visibility = if (!boolean) View.VISIBLE else
        View.GONE
    }

```

```

private fun showSortingOptionsDialog() {
    val options = arrayOf(
        "Sort by Name Ascending",
        "Sort by Name Descending",
        "Sort by Date Ascending",
        "Sort by Date Descending"
    )

    val builder = MaterialAlertDialogBuilder(this,
        R.style.MyMaterialAlertDialog)

    builder.setTitle("Select Sorting Option")
        .setItems(options) { _, which ->
            when (which) {
                0 -> notesAdapter.sortByNameAscending()
                1 -> notesAdapter.sortByNameDescending()
                2 -> notesAdapter.sortByDateAscending()
                3 -> notesAdapter.sortByDateDescending()
            }
        }

    builder.create().show()
}

```

```

private fun openNote(position: Int, createId: String = "") {
    val intent = Intent(this, NoteActivity::class.java)
    val noteId: String = if (position != -1) {
        val selectedNote = notesAdapter.getItem(position)
        selectedNote?.id.toString()
    }
}

```

```

    } else {
        createId
    }
    intent.putExtra("noteId", noteId)
    startActivity(intent)
}

private fun deleteNote(position: Int) {
    val selectedNote = notesAdapter.getItem(position)

    val builder = MaterialAlertDialogBuilder(this,
R.style.MyMaterialAlertDialog)
    builder.setTitle("Delete Note")
        .setMessage("Are you sure you want to delete this note?")
        .setPositiveButton("Yes") { _, _ ->
            if (selectedNote != null) {
                noteManager.deleteNoteFromFirestore(selectedNote.id)
                notesAdapter.notifyDataSetChanged()
                binding.searchView.setQuery("", true)
            }
        }
        .setNegativeButton("No", null)

    builder.create().show()
}

private fun showOptionsDialog(position: Int) {
    val options = arrayOf("Open", "Edit Title", "Delete")
    val builder = MaterialAlertDialogBuilder(this,
R.style.MyMaterialAlertDialog)

```

```

builder.setTitle("Choose an option")
    .setItems(options) { _, which ->
        when (which) {
            0 -> openNote(position)
            1 -> editTitle(position)
            2 -> deleteNote(position)
        }
    }
builder.create().show()
}

```

```

private fun editTitle(position: Int) {
    val selectedNote = notesAdapter.getItem(position)

    val editText = EditText(this)
    editText.inputType = InputType.TYPE_CLASS_TEXT
    editText.setText(selectedNote?.title)
}

```

```

val frameLayout = FrameLayout(this)
frameLayout.layoutParams = FrameLayout.LayoutParams(
    FrameLayout.LayoutParams.MATCH_PARENT,
    FrameLayout.LayoutParams.WRAP_CONTENT
)
frameLayout.setPadding(50, 0, 50, 0)
frameLayout.addView(editText)

```

```

val builder = MaterialAlertDialogBuilder(this,
R.style.MyMaterialAlertDialog)
builder.setTitle("Edit Title")
    .setView(frameLayout)

```

```

        .setPositiveButton("Save") { _, _ ->
            val newTitle = editText.text.toString()
            noteManager.updateNoteTitle(selectedNote?.id, newTitle)
            notesAdapter.notifyDataSetChanged()
            binding.searchView.setQuery("", true)
        }
        .setNegativeButton("Cancel", null)

    builder.create().show()
}
}

```

B.3 NoteActivity.kt

```

package com.nazarkhokhlov.notewizard.activities

import android.app.Activity
import android.content.ClipData
import android.content.ClipboardManager
import android.content.Intent
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.widget.TextView
import androidx.activity.result.contract.ActivityResultContracts
import androidx.appcompat.app.AppCompatActivity
import com.google.android.material.dialog.MaterialAlertDialogBuilder
import com.nazarkhokhlov.notewizard.R
import com.nazarkhokhlov.notewizard.adapters.FilesAdapter
import com.nazarkhokhlov.notewizard.databinding.ActivityNoteBinding

```

```

import
com.nazarkhokhlov.notewizard.databinding.DialogGenerateTextBinding
import com.nazarkhokhlov.notewizard.managers.FileManager
import com.nazarkhokhlov.notewizard.managers.FireStorageManager
import com.nazarkhokhlov.notewizard.managers.NoteManager
import com.nazarkhokhlov.notewizard.network.OpenAIChatTask
import com.nazarkhokhlov.notewizard.other.Constants.showToast
import org.json.JSONException
import org.json.JSONObject

class NoteActivity : AppCompatActivity() {

    private lateinit var binding: ActivityNoteBinding
    private lateinit var dialogGenerateTextBinding:
DialogGenerateTextBinding
    private lateinit var noteManager: NoteManager
    private lateinit var fileManager: FileManager
    private lateinit var adapter: FilesAdapter
    private var noteId: String = ""

    private val filePickerLauncher =

registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {
result ->
    if (result.resultCode == Activity.RESULT_OK) {
        result.data?.data?.let { uri ->
            fileManager.handleSelectedFile(uri)
        }
    }
}
}

```

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityNoteBinding.inflate(layoutInflater)
    setContentView(binding.root)

    noteId = intent.extras?.getString("noteId") ?: ""

    initAdapter()
    initManagers()
    initViews()

    if (noteId.isNotEmpty()) {
        noteManager.loadNoteData(noteId, fileManager)
    }
}

private fun initViews() {
    binding.btnAddFile.setOnClickListener {
        launchFilePicker()
    }

    binding.btnGenerateText.setOnClickListener {
        showGenerateTextDialog()
    }

    binding.listViewAttachedFiles.setOnItemClickListener { _, _,
position, _ ->
        fileManager.openFile(adapter.getItem(position)?.path ?: "")
    }
}

```

```

binding.listViewAttachedFiles.setOnItemLongClickListener { _, _,
position, _ ->
    val fileItem = adapter.getItem(position)

    MaterialAlertDialogBuilder(this, R.style.MyMaterialAlertDialog)
        .setTitle("Options")
        .setItems(arrayOf("Open", "Delete")) { _, which ->
            when (which) {
                0 -> {
                    fileManager.openFile(fileItem?.path ?: "")
                }

                1 -> {
                    adapter.remove(fileItem)
                    noteManager.updateNoteInFirestore(noteId)
                    fileManager.deleteFile(fileItem?.path ?: "")
                }
            }
        }
        .show()

    true
}
}

```

```

private fun showGenerateTextDialog() {
    dialogGenerateTextBinding =
DialogGenerateTextBinding.inflate(layoutInflater)
    val dialog = MaterialAlertDialogBuilder(this,
R.style.MyMaterialAlertDialog)
        .setView(dialogGenerateTextBinding.root)
        .setCancelable(false)
        .setTitle("Generate Text")
        .create()

    dialogGenerateTextBinding.btnCancel.setOnClickListener {
        dialog.dismiss()
    }

    dialogGenerateTextBinding.btnGenerate.setOnClickListener {
        if (dialogGenerateTextBinding.editText.text.isNotEmpty()) {
            it.isEnabled = false
            dialogGenerateTextBinding.btnCancel.isEnabled = false
            dialogGenerateTextBinding.answersProgressBar.visibility =
View.VISIBLE
            dialogGenerateTextBinding.scrollViewAnswers.visibility =
View.GONE

            dialogGenerateTextBinding.linearLayoutAnswers.removeAllViews()

            val inputText =
dialogGenerateTextBinding.editText.text.toString()
            if (inputText.isNotEmpty()) {
                val openAIChatTask = OpenAIChatTask(inputText) { answer

```

->

```

try {
    val jsonObject = JSONObject(answer)
    val answers = jsonObject.getJSONArray("choices")
        .getJSONObject(0)
        .getJSONObject("message")
        .getString("content")
    val answersArray = answers.split("\n")

    for (string in answersArray) {

dialogGenerateTextBinding.answersProgressBar.visibility = View.GONE

        val inflater = LayoutInflater.from(this)
        val answerItemView = inflater.inflate(
            R.layout.answer_item,
            dialogGenerateTextBinding.linearLayoutAnswers,
            false
        )
        val                textViewAnswer                =
answerItemView.findViewById<TextView>(R.id.textViewAnswer)
        textViewAnswer.text = string

dialogGenerateTextBinding.linearLayoutAnswers.addView(answerItemView)

dialogGenerateTextBinding.scrollViewAnswers.visibility = View.VISIBLE
        dialogGenerateTextBinding.btnGenerate.isEnabled    =
true

        dialogGenerateTextBinding.btnCancel.isEnabled = true

        answerItemView.setOnClickListener {

```

```

        showMaterialDialog(string)
    }
}
} catch (e: JSONException) {
    e.printStackTrace()
}
}
openAIChatTask.execute()
}
} else {
    showToast("Text field is empty")
}
}
dialog.show()
}

```

```

private fun showMaterialDialog(text: String) {
    val options = arrayOf("Copy", "Add to Note", "Cancel")

```

```

    MaterialAlertDialogBuilder(this, R.style.MyMaterialAlertDialog)

```

```

        .setTitle("Choose Action")

```

```

        .setItems(options) { dialog, which ->

```

```

            when (which) {

```

```

                0 -> {

```

```

                    val clipboardManager =

```

```

                    getSystemService(CLIPBOARD_SERVICE) as ClipboardManager

```

```

                    val clipData = ClipData.newPlainText("Copied Text", text)

```

```

                    clipboardManager.setPrimaryClip(clipData)

```

```

                    showToast("Text copied to clipboard")

```

```

                }

```

```

1 -> {
    val currentText = binding.memoEditText.text.toString()
    val newText = currentText + "\n" + text
    binding.memoEditText.setText(newText)
    showToast("Text added to note")
}
2 -> {
    dialog.dismiss()
}
}
}
.show()
}

```

```

override fun onPause() {
    noteManager.updateNoteInFirestore(noteId)
    super.onPause()
}

```

```

private fun initManagers() {
    noteManager = NoteManager(adapter, binding = binding)
    fileManager = FileManager(noteManager, adapter, noteId)
}

```

```

private fun initAdapter() {
    adapter = FilesAdapter(this, R.layout.list_item_file, mutableListOf())
}
{
    isAdapterEmpty(it)
}
binding.listViewAttachedFiles.adapter = adapter

```

```
isAdapterEmpty(adapter.isEmpty)
}
```

```
private fun isAdapterEmpty(boolean: Boolean) {
    binding.listViewAttachedFiles.visibility = if (!boolean)
View.VISIBLE else View.GONE
}
```

```
private fun launchFilePicker() {
    val intent = Intent(Intent.ACTION_GET_CONTENT)
    intent.type = "*/*"
    filePickerLauncher.launch(intent)
}
}
```

B.4 SettingsActivity.kt

```
package com.nazarkhokhlov.notewizard.activities

import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import com.google.android.material.dialog.MaterialAlertDialogBuilder
import com.google.firebase.Firebase
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.auth
import com.nazarkhokhlov.notewizard.R
import
com.nazarkhokhlov.notewizard.databinding.ActivitySettingsBinding
import com.nazarkhokhlov.notewizard.managers.FireStorageManager
```

```
import com.nazarkhokhlov.notewizard.managers.NoteManager
import com.nazarkhokhlov.notewizard.other.Constants.appContext
import com.nazarkhokhlov.notewizard.other.Constants.showToast
```

```
class SettingsActivity : AppCompatActivity() {
```

```
    private lateinit var auth: FirebaseAuth
```

```
    private lateinit var binding: ActivitySettingsBinding
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
```

```
        binding = ActivitySettingsBinding.inflate(layoutInflater)
```

```
        setContentView(binding.root)
```

```
        auth = FirebaseAuth
```

```
        binding.btnLogout.setOnClickListener {
```

```
            showLogoutConfirmationDialog()
```

```
        }
```

```
        binding.btnDeleteAccount.setOnClickListener {
```

```
            showDeleteAccountConfirmationDialog()
```

```
        }
```

```
        binding.btnClearCache.setOnClickListener {
```

```
            showClearCacheDialog()
```

```
        }
```

```
    }
```

```
    private fun showClearCacheDialog() {
```

```

MaterialAlertDialogBuilder(this, R.style.MyMaterialAlertDialog)
    .setTitle("Clear Cache")
    .setMessage("All your cache will be deleted and this action cannot
be undone")
    .setPositiveButton("Clear") { _, _ ->
        clearCache()
    }
    .setNegativeButton("Cancel") { dialog, _ ->
        dialog.dismiss()
    }
    .show()
}

```

```

private fun clearCache() {
    val filesDir = appContext.filesDir
    filesDir.deleteRecursively()
    showToast("Successfully deleted cache files.")
}

```

```

private fun showLogoutConfirmationDialog() {
    MaterialAlertDialogBuilder(this, R.style.MyMaterialAlertDialog)
        .setTitle("Logout")
        .setMessage("Are you sure you want to logout?")
        .setPositiveButton("Logout") { _, _ ->
            logoutUser()
        }
        .setNegativeButton("Cancel") { dialog, _ ->
            dialog.dismiss()
        }
        .show()
}

```

```
}
```

```
private fun logoutUser() {
    auth.signOut()
    navigateToLoginScreen()
}
```

```
private fun navigateToLoginScreen() {
    val intent = Intent(this, LoginActivity::class.java)
    intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK or
Intent.FLAG_ACTIVITY_CLEAR_TASK
    startActivity(intent)
    finish()
}
```

```
private fun showDeleteAccountConfirmationDialog() {
    AlertDialog.Builder(this, R.style.MyMaterialAlertDialog)
        .setTitle("Delete Account")
        .setMessage("Are you sure you want to delete your account?")
        .setPositiveButton("Delete") { _, _ ->
            showDeleteAccountWarningDialog()
        }
        .setNegativeButton("Cancel") { dialog, _ ->
            dialog.dismiss()
        }
        .show()
}
```

```
private fun showDeleteAccountWarningDialog() {
    AlertDialog.Builder(this, R.style.MyMaterialAlertDialog)
```

```

        .setTitle("Warning")
        .setMessage("All your data will be deleted and this action cannot be
undone. Are you sure?")
        .setPositiveButton("Delete") { _, _ ->
            deleteAccountAndData()
        }
        .setNegativeButton("Cancel") { dialog, _ ->
            dialog.dismiss()
        }
        .show()
    }

private fun deleteAccountAndData() {
    val user = auth.currentUser
    NoteManager().deleteAllUserNotesFromFirestore(user?.uid)
    FireStorageManager().deleteAllUserUploadsFromFirebaseStorage()
    user?.delete()

    ?.addOnCompleteListener { task ->
        if (task.isSuccessful) {
            showToast("Account deleted successfully.")
            navigateToLoginScreen()
        } else {
            showToast("Failed to delete account. Re-login and try again.")
        }
    }
}
}

```

B.5 SignupActivity.kt

```
package com.nazarkhokhlov.notewizard.activities

import android.content.Intent
import android.os.Bundle
import android.widget.Button
import androidx.appcompat.app.AppCompatActivity
import com.google.firebase.auth.FirebaseAuth
import com.nazarkhokhlov.notewizard.databinding.ActivitySignupBinding
import com.nazarkhokhlov.notewizard.other.Constants.showToast

class SignupActivity : AppCompatActivity() {

    private lateinit var auth: FirebaseAuth
    private lateinit var binding: ActivitySignupBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivitySignupBinding.inflate(layoutInflater)
        setContentView(binding.root)

        auth = FirebaseAuth.getInstance()

        binding.apply {
            val signUpButton: Button = btnSignUp
            signUpButton.setOnClickListener {
                val email = editTextEmail.text.toString()
                val password = editTextPassword.text.toString()

                if (email.isNotEmpty() && password.isNotEmpty()) {
                    auth.createUserWithEmailAndPassword(email, password)
                }
            }
        }
    }
}
```



```

class FilesAdapter(
    context: Context,
    resource: Int,
    objects: MutableList<FileItem>,
    private var onUpdate: (Boolean) -> Unit
):
    ArrayAdapter<FileItem>(context, resource, objects) {

        private val viewsList = mutableListOf<View>()

        override fun getView(position: Int, convertView: View?, parent:
ViewGroup): View {
            val inflater =
context.getSystemService(Context.LAYOUT_INFLATER_SERVICE) as
LayoutInflater
            val rowView = inflater.inflate(R.layout.list_item_file, parent, false)

            val imageViewIcon: ImageView =
rowView.findViewById(R.id.imageViewIcon)
            val textViewFileName: TextView =
rowView.findViewById(R.id.textViewFileName)

            val fileItem = getItem(position)

            imageViewIcon.setImageResource(fileItem?.iconResId ?:
R.drawable.ic_launcher_foreground)

            textViewFileName.text = fileItem?.path?.substringAfterLast("/") ?:
"Unknown File"

```

```

        viewsList.add(rowView)

        return rowView
    }

    fun getAllViews(): List<View> {
        return viewsList.toList()
    }

    override fun notifyDataSetChanged() {
        super.notifyDataSetChanged()
        onUpdate(count == 0)
    }
}

```

B.7 NotesAdapter.kt

```

package com.nazarkhokhlov.notewizard.adapters

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.AdapterView
import android.widget.Filter
import android.widget.Filterable
import android.widget.TextView
import com.nazarkhokhlov.notewizard.R
import com.nazarkhokhlov.notewizard.models.Note

```

```

import java.text.SimpleDateFormat
import java.util.Date
import java.util.Locale

class NotesAdapter(context: Context, private val notesList:
MutableList<Note>, private var onUpdate: (Boolean) -> Unit) :
ArrayAdapter<Note>(context, 0, notesList), Filterable {

    private var filteredNotesList: MutableList<Note> =
notesList.toMutableList()

    override fun getView(position: Int, convertView: View?, parent:
ViewGroup): View {
        var itemView = convertView
        if (itemView == null) {
            itemView =
LayoutInflater.from(context).inflate(R.layout.list_item_note, parent, false)
        }

        val currentNote = getItem(position)

        val titleTextView =
itemView!!.findViewById<TextView>(R.id.textViewTitle)
        val idTextView =
itemView.findViewById<TextView>(R.id.textViewDate)

        titleTextView.text = currentNote?.title

        val timestamp = currentNote?.timestamp ?:
System.currentTimeMillis()

```

```

        val sdf = SimpleDateFormat("dd.MM.yy\nHH:mm",
Locale.getDefault())

        val formattedTimestamp = sdf.format(Date(timestamp))
        idTextView.text = formattedTimestamp

    return itemView
}

override fun getCount(): Int {
    return filteredNotesList.size
}

override fun getItem(position: Int): Note? {
    return filteredNotesList.getOrNull(position)
}

override fun getFilter(): Filter {
    return object : Filter() {
        override fun performFiltering(constraint: CharSequence?):
FilterResults {
            val queryString =
constraint?.toString()?.lowercase(Locale.getDefault())
            val filterResults = FilterResults()
            filterResults.values = if (queryString.isNullOrEmpty()) {
                notesList
            } else {
                notesList.filter {
it.title.lowercase(Locale.getDefault()).contains(queryString) }
            }
            return filterResults
        }
    }
}

```

```
}
```

```
        override fun publishResults(constraint: CharSequence?, results:
FilterResults?) {
            filteredNotesList = results?.values as MutableList<Note>
            notifyDataSetChanged()
        }
    }
}
```

```
fun updateNotes(newNotesList: List<Note>) {
    notesList.clear()
    notesList.addAll(newNotesList)
    filteredNotesList.clear()
    filteredNotesList.addAll(newNotesList)
    notifyDataSetChanged()
}
```

```
fun sortByNameAscending() {
    notesList.sortBy { it.title }
    notifyDataSetChanged()
}
```

```
fun sortByNameDescending() {
    notesList.sortByDescending { it.title }
    notifyDataSetChanged()
}
```

```
fun sortByDateAscending() {
    notesList.sortBy { it.timestamp }
```

```

        notifyDataSetChanged()
    }

```

```

fun sortByDateDescending() {
    notesList.sortByDescending { it.timestamp }
    notifyDataSetChanged()
}

```

```

override fun notifyDataSetChanged() {
    super.notifyDataSetChanged()
    onUpdate(count == 0)
}
}

```

B.8 FileManager.kt

```

package com.nazarkhokhlov.notewizard.managers

import android.content.Intent
import android.net.Uri
import android.provider.OpenableColumns
import android.util.Log
import android.webkit.MimeTypeMap
import android.widget.TextView
import androidx.core.content.FileProvider
import com.nazarkhokhlov.notewizard.R
import com.nazarkhokhlov.notewizard.adapters.FilesAdapter
import com.nazarkhokhlov.notewizard.models.FileItem
import com.nazarkhokhlov.notewizard.other.Constants
import com.nazarkhokhlov.notewizard.other.Constants.showToast

```

```

import java.io.File
import java.io.FileOutputStream
import java.io.InputStream

class FileManager(
    private val noteManager: NoteManager,
    private val adapter: FilesAdapter,
    private val noteId: String
) {

    private val storageManager = FireStorageManager(this)

    private fun isImageFile(file: File): Boolean {
        val imageExtensions = arrayOf("jpg", "jpeg", "png", "gif", "bmp",
"webp")
        return imageExtensions.contains(file.extension.lowercase())
    }

    private fun isVideoFile(file: File): Boolean {
        val videoExtensions = arrayOf("mp4", "avi", "mkv", "mov", "wmv",
"flv", "webm")
        return videoExtensions.contains(file.extension.lowercase())
    }

    private fun isAudioFile(file: File): Boolean {
        val audioExtensions = arrayOf("mp3", "wav", "flac", "aac", "ogg",
"wma", "m4a")
        return audioExtensions.contains(file.extension.lowercase())
    }

```

```

private fun getMimeType(file: File): String {
    val extension = MimeTypeMap.getFileExtensionFromUrl(file.path)
    return
MimeTypeMap.getSingleton().getMimeTypeFromExtension(extension) ?: "*/*"
}

private fun copyFile(inputStream: InputStream, outputStream:
FileOutputStream) {
    inputStream.use { input ->
        outputStream.use { output ->
            input.copyTo(output, bufferSize = 1024)
        }
    }
}

fun deleteFile(filePath: String) {
    val fDelete = File(filePath)
    if (fDelete.exists()) {
        if (fDelete.delete()) {
            showToast("file Deleted :" + filePath)
        } else {
            showToast("file not Deleted :" + filePath)
        }
    }
}

fun getImageIdUsingPath(file: File): Int {
    return when {
        isImageFile(file) -> R.drawable.ic_image
        isVideoFile(file) -> R.drawable.ic_video_file
    }
}

```

```

        isAudioFile(file) -> R.drawable.ic_audio_file
        else -> R.drawable.ic_file
    }
}

```

```

fun openFile(filePath: String) {
    val file = File(filePath)

    if (file.exists()) {
        openLocalFile(file)
    } else {
        storageManager.downloadFileFromFirebaseStorage(file, noteId)
    }
}

```

```

fun handleSelectedFile(uri: Uri) {
    val inputStream: InputStream? =
Constants.appContext.contentResolver.openInputStream(uri)

```

```

    inputStream?.let {
        val fileName = getFileName(uri)
        val fileExtension = getFileExtension(fileName)
        val fileNameWithoutExtension =
getFileNameWithoutExtension(fileName)

```

```

        val existingView = adapter.getAllViews().firstOrNull { view ->
            val textViewFileName =
view.findViewById<TextView>(R.id.textViewFileName)
            textViewFileName?.text?.toString()?.substringAfterLast("/") ==
fileName

```

```

    }

    var modifiedFileName = fileName

    if (existingView != null) {
        val noteDir = File(Constants.appContext.filesDir, noteId)
        val existingFiles = noteDir.listFiles()
        val existingFileNames = existingFiles?.map { it.name }

        var suffix = 2
        while
            (existingFileNames?.contains("$fileNameWithoutExtension($suffix).$fileExtension"
            n") == true) {
            suffix++
        }

        modifiedFileName =
            "${fileNameWithoutExtension}($suffix).$fileExtension"
    }

    val noteDir = File(Constants.appContext.filesDir, noteId)
    if (!noteDir.exists()) {
        noteDir.mkdir()
    }

    val destination = File(noteDir, modifiedFileName)
    copyFile(inputStream, FileOutputStream(destination))
    val selectedFilePath = destination.absolutePath

    val iconResId = getImageIdUsingPath(destination)
    adapter.add(FileItem(selectedFilePath ?: "", iconResId))

```

```

        storageManager.uploadFileToFirebaseStorage(uri,
modifiedFileName, noteId)

```

```

        noteManager.updateNoteInFirestore(noteId)
    }
}

```

```

private fun getFileExtension(fileName: String): String {
    val lastDotIndex = fileName.lastIndexOf('.')
    return if (lastDotIndex == -1) {
        ""
    } else {
        fileName.substring(lastDotIndex + 1)
    }
}

```

```

private fun getFileNameWithoutExtension(fileName: String): String {
    val lastDotIndex = fileName.lastIndexOf('.')
    return if (lastDotIndex == -1) {
        fileName
    } else {
        fileName.substring(0, lastDotIndex)
    }
}

```

```

fun getFileName(uri: Uri): String {
    Constants.appContext.contentResolver.query(uri, null, null, null,
null)?.use { cursor ->
        if (cursor.moveToFirst()) {

```

```

        return
        cursor.getString(cursor.getColumnIndexOrThrow(OpenableColumns.DISPLAY_
NAME))
    }
}

uri.path?.lastIndexOf('/')?.let { cut ->
    return uri.path?.substring(cut + 1) ?: "unknown_file"
}

return "unknown_file"
}

fun openLocalFile(file: File) {
    val uri = FileProvider.getUriForFile(
        Constants.appContext,
        "${Constants.appContext.packageName}.provider",
        file
    )

    val intent = Intent(Intent.ACTION_VIEW)
    intent.setDataAndType(uri, getMimeType(file))
    intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION)

    try {
        Constants.appContext.startActivity(intent)
    } catch (e: Exception) {
        e.printStackTrace()
    }
}

```

```
}
```

B.9 FireStorageManager.kt

```
package com.nazarkhokhlov.notewizard.managers

import android.net.Uri
import android.util.Log
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.storage.FirebaseStorage
import com.google.firebase.storage.StorageReference
import com.nazarkhokhlov.notewizard.other.Constants
import com.nazarkhokhlov.notewizard.other.Constants.showToast
import java.io.File
import java.io.InputStream

class FireStorageManager(private val fileManager: FileManager? = null) {

    private var storageRef: StorageReference
    private var auth: FirebaseAuth = FirebaseAuth.getInstance()
    private var storage: FirebaseStorage = FirebaseStorage.getInstance()

    init {
        storageRef =
storage.reference.child("uploads").child(auth.currentUser?.uid ?: "")
    }

    fun uploadFileToFirebaseStorage(fileUri: Uri, fileName: String, noteId:
String) {
```

```

val fileRef = storageRef.child(noteId).child(fileName)
val inputStream: InputStream =
    Constants.appContext.contentResolver.openInputStream(fileUri)!!

fileRef.putStream(inputStream)
    .addOnSuccessListener {
        showToast("File uploaded to Firebase Storage")
    }
    .addOnFailureListener { e ->
        showToast("Error uploading file to Firebase Storage.")
        Log.e("Firebase Storage", "Error uploading file to Firebase
Storage: $e")
    }
}

fun downloadFileFromFirebaseStorage(file: File, noteId: String) {
    val fileName = file.name

    val fileRef = storageRef.child(noteId).child(fileName)
    val localFile = File(Constants.appContext.filesDir, fileName)

    fileRef.getFile(localFile)
        .addOnSuccessListener {
            fileManager?.openLocalFile(localFile)
        }
        .addOnFailureListener { e ->
            showToast("Error downloading file from Firebase Storage")
            Log.e("Firebase Storage", "Error downloading file from Firebase
Storage: $e")
        }
}

```

```

        }
    }

    fun deleteUserUploadFileFromFirebaseStorage(noteId: String, fileName:
String) {

        storageRef.child(noteId).child(fileName.substringAfterLast('/')).delete()
            .addOnSuccessListener {
                showToast("Successfully deleted file from FireStorage")
            }
    }

    fun deleteAllUserUploadsFromFirebaseStorage() {
        storageRef.listAll().addOnSuccessListener { listResult ->
            listResult.prefixes.forEach { directoryRef ->
                directoryRef.listAll().addOnSuccessListener { list ->
                    list.items.forEach {
                        it.delete()
                    }
                }
            }
        }
        showToast("All user uploads deleted successfully from Firebase
Storage")
    }.addOnFailureListener { e ->
        showToast("Failed to retrieve storage contents")
        Log.e("Firebase Storage", "Failed to retrieve storage contents:
${e.message}")
    }
}
}
}

```

B.10 NoteManager.kt

```
package com.nazarkhokhlov.notewizard.managers

import android.net.Uri
import android.util.Log
import android.widget.AdapterView
import androidx.core.widget.doOnTextChanged
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.firestore.FirebaseFirestore
import com.nazarkhokhlov.notewizard.adapters.FilesAdapter
import com.nazarkhokhlov.notewizard.adapters.NotesAdapter
import com.nazarkhokhlov.notewizard.databinding.ActivityNoteBinding
import com.nazarkhokhlov.notewizard.models.FileItem
import com.nazarkhokhlov.notewizard.models.Note
import com.nazarkhokhlov.notewizard.other.Constants
import com.nazarkhokhlov.notewizard.other.Constants.showToast
import java.io.File
import java.util.UUID

class NoteManager(
    private val adapter: FilesAdapter? = null,
    private val notesAdapter: NotesAdapter? = null,
    private val binding: ActivityNoteBinding? = null
) {

    private var auth: FirebaseAuth = FirebaseAuth.getInstance()
```

```
private val firebaseFirestore = FirebaseFirestore.getInstance()
```

```
fun updateNoteTitle(noteId: String?, newTitle: String) {
    if (noteId != null) {
        firebaseFirestore.collection("notes")
            .document(auth.currentUser?.uid ?: "")
            .collection("user_notes")
            .whereEqualTo("id", noteId)
            .get()
            .addOnSuccessListener { documents ->
                if (!documents.isEmpty) {
                    val documentId = documents.documents[0].id

                    firebaseFirestore.collection("notes")
                        .document(auth.currentUser?.uid ?: "")
                        .collection("user_notes")
                        .document(documentId)
                        .update(
                            "title", newTitle,
                        )
                        .addOnSuccessListener {
                            showToast("Note Title updated successfully")
                            notesAdapter?.notifyDataSetChanged()
                        }
                        .addOnFailureListener { e ->
                            showToast("Error Title updating note")
                            Log.e("Firebase Firestore", "Error Title updating note:
                                $e")
                        }
                    }
            }
    }
}
```

```

    }
    .addOnFailureListener { exception ->
        showToast("Error loading note data")
        Log.e("Firebase Firestore", "Error loading note data:
$exception")
    }
}
}
}

```

```

fun deleteAllUserNotesFromFirestore(userId: String?) {
    firebaseFirestore.collection("notes")
        .document(userId.toString())
        .collection("user_notes")
        .get()
        .addOnSuccessListener { documents ->
            for (document in documents) {
                document.reference.delete()
            }
            showToast("All user notes deleted successfully from Firestore")
        }
        .addOnFailureListener { e ->
            showToast("Error deleting user notes from Firestore")
            Log.e("Firebase Firestore", "Error deleting user notes from
Firestore: $e")
        }
    notesAdapter?.notifyDataSetChanged()
}

```

```

fun deleteNoteFromFirestore(noteId: String) {
    firebaseFirestore.collection("notes")

```

```

        .document(auth.currentUser?.uid ?: "")
        .collection("user_notes")
        .whereEqualTo("id", noteId)
        .get()
        .addOnSuccessListener { documents ->
            if (!documents.isEmpty) {
                val documentId = documents.documents[0].id

                firebaseFirestore.collection("notes")
                    .document(auth.currentUser?.uid ?: "")
                    .collection("user_notes")
                    .document(documentId)
                    .delete()
                    .addOnSuccessListener {
                        showToast("Note deleted successfully")
                        notesAdapter?.notifyDataSetChanged()
                    }
                    .addOnFailureListener { e ->
                        showToast("Error deleting note")
                        Log.e("Firebase Firestore", "Error deleting note: $e")
                    }
            }
        }
        .addOnFailureListener { exception ->
            showToast("Error loading note data")
            Log.e("Firebase Firestore", "Error loading note data:
$exception")
        }
    }
}

```

```

fun saveNoteToFirestore(doOnSuccess: (id: String) -> Unit) {
    val note = Note(
        id = UUID.randomUUID().toString(),
        title = "New Note",
        timestamp = System.currentTimeMillis()
    )

    firebaseFirestore.collection("notes")
        .document(auth.currentUser?.uid ?: "")
        .collection("user_notes")
        .add(note)
        .addOnSuccessListener {
            showToast("Note created successfully")
            doOnSuccess(note.id)
            notesAdapter?.notifyDataSetChanged()
        }
        .addOnFailureListener { e ->
            showToast("Error creating note")
            Log.e("Firebase Firestore", "Error creating note: $e")
        }
}

fun retrieveNotes() {
    firebaseFirestore.collection("notes")
        .document(auth.currentUser?.uid ?: "")
        .collection("user_notes")
        .addSnapshotListener { snapshot, exception ->
            if (exception != null) {
                return@addSnapshotListener
            }
        }
}

```

```

        notesAdapter?.clear()

        val notesList = mutableListOf<Note>()
        snapshot?.documents?.let {
            for (document in it) {
                val note = document.toObject(Note::class.java)
                if (note != null) {
                    notesList.add(note)
                }
            }
        }
        notesAdapter?.updateNotes(notesList)
    }
}

fun updateNoteInFirestore(noteId: String) {
    val memoText = binding?.memoEditText?.text.toString()
    val filePaths = (0 until adapter!!.count).map {
        adapter.getItem(it)?.path!! }

    val updatedNote = Note(
        text = memoText,
        files = filePaths
    )

    firebaseFirestore.collection("notes")
        .document(auth.currentUser?.uid ?: "")
        .collection("user_notes")
        .whereEqualTo("id", noteId)

```

```

.get()
.addOnSuccessListener { documents ->
    if (!documents.isEmpty) {
        val documentId = documents.documents[0].id
        firebaseFirestore.collection("notes")
            .document(auth.currentUser?.uid ?: "")
            .collection("user_notes")
            .document(documentId)
            .update(
                "text", updatedNote.text,
                "files", updatedNote.files
            )
            .addOnSuccessListener {
                showToast("Note updated successfully")
            }
            .addOnFailureListener { e ->
                showToast("Error updating note")
                Log.e("Firebase Firestore", "Error updating note: $e")
            }
    }
}

.addOnFailureListener { exception ->
    showToast("Error loading note data")
    Log.e("Firebase Firestore", "Error loading note data:
$exception")
}

}

fun loadNoteData(noteId: String, fileManager: FileManager) {
    firebaseFirestore.collection("notes")
        .document(auth.currentUser?.uid ?: "")

```

```

.collection("user_notes")
.whereEqualTo("id", noteId)
.get()
.addOnSuccessListener { documents ->
    if (!documents.isEmpty) {
        val note = documents.documents[0].toObject(Note::class.java)
        note?.let {
            binding?.memoEditText?.setText(it.text)
            it.files.forEach { filePath ->
                val fileName =
fileManager.getFileName(Uri.parse(filePath))
                val destination = File(Constants.appContext.filesDir,
fileName)
                adapter?.add(
                    FileItem(
                        filePath,
                        fileManager.getImageIdUsingPath(destination)
                    )
                )
            }
        }
    }
}
.addOnFailureListener { exception ->
    showToast("Error loading note data")
    Log.e("Firebase Firestore", "Error loading note data:
$exception")
}
}
}

```

B.11 FileItem.kt

```
package com.nazarkhokhlov.notewizard.models
```

```
data class FileItem(val path: String, val iconResId: Int)
```

B.12 Note.kt

```
package com.nazarkhokhlov.notewizard.models
```

```
data class Note(
    var id: String = "",
    var title: String = "",
    var text: String = "",
    var timestamp: Long = 0L,
    var files: List<String> = listOf()
) {
    override fun toString(): String {
        return title
    }
}
```

B.13 OpenAIChatTask.kt

```
package com.nazarkhokhlov.notewizard.network
```

```
import android.os.AsyncTask
import android.util.Log
import com.nazarkhokhlov.notewizard.MyApplication.Companion.apiKey
import com.nazarkhokhlov.notewizard.R
```

```

import com.nazarkhokhlov.notewizard.other.Constants.appContext
import com.nazarkhokhlov.notewizard.other.Constants.showToast
import okhttp3.MediaType.Companion.toMediaTypeOrNull
import okhttp3.OkHttpClient
import okhttp3.Request
import okhttp3.RequestBody.Companion.toRequestBody
import org.json.JSONArray
import org.json.JSONObject
import java.io.IOException

```

```

class OpenAIChatTask(private var userNote: String, private var
doLayoutFill: (String) -> Unit) : AsyncTask<Void, Void, String>() {

```

```

    override fun doInBackground(vararg params: Void?): String? {

```

```

        val client = OkHttpClient()
        val url = "https://api.openai.com/v1/chat/completions"
        val json = JSONObject()
        val messagesArray = JSONArray()

```

```

        val systemMessage = JSONObject()
        systemMessage.put("role", "system")
        systemMessage.put("content",
appContext.getString(R.string.chagpt_prompt))
        messagesArray.put(systemMessage)

```

```

        val userMessage = JSONObject()
        userMessage.put("role", "user")
        userMessage.put("content", userNote)
        messagesArray.put(userMessage)

```

```

        json.put("model", "gpt-3.5-turbo")
        json.put("messages", messagesArray)

        val body = json.toString().toRequestBody("application/json".toMediaTypeOrNull())

        val request = Request.Builder()
            .url(url)
            .header("Content-Type", "application/json")
            .header("Authorization", "Bearer $apiKey")
            .post(body)
            .build()

        try {
            val response = client.newCall(request).execute()
            return response.body?.string()
        } catch (e: IOException) {
            Log.e("OpenAIChatTask", "Error making API request: ${e.message}")
            showToast("Error making API request: ${e.message}")
        }
        return null
    }

    override fun onPostExecute(result: String?) {
        super.onPostExecute(result)
        Log.e("OpenAIChatTask", "API response: $result")
        doLayoutFill(result.toString())
    }
}

```

B.14 Constants.kt

```

package com.nazarkhokhlov.notewizard.other

import android.content.Context
import android.widget.Toast

object Constants {
    lateinit var appContext: Context
    fun showToast(message: String) {
        Toast.makeText(appContext, message,
Toast.LENGTH_SHORT).show()
    }
}

```

B.15 MyApplication.kt

```

package com.nazarkhokhlov.notewizard

import android.app.Application
import com.google.firebase.FirebaseApp
import com.nazarkhokhlov.notewizard.other.Constants

class MyApplication : Application() {
    override fun onCreate() {
        super.onCreate()

        Constants.appContext = this
        FirebaseApp.initializeApp(this)
    }
}

```

```
}  
  
companion object {  
    var apiKey: String? = null  
}  
}
```

ДОДАТОК В
Слайди презентації

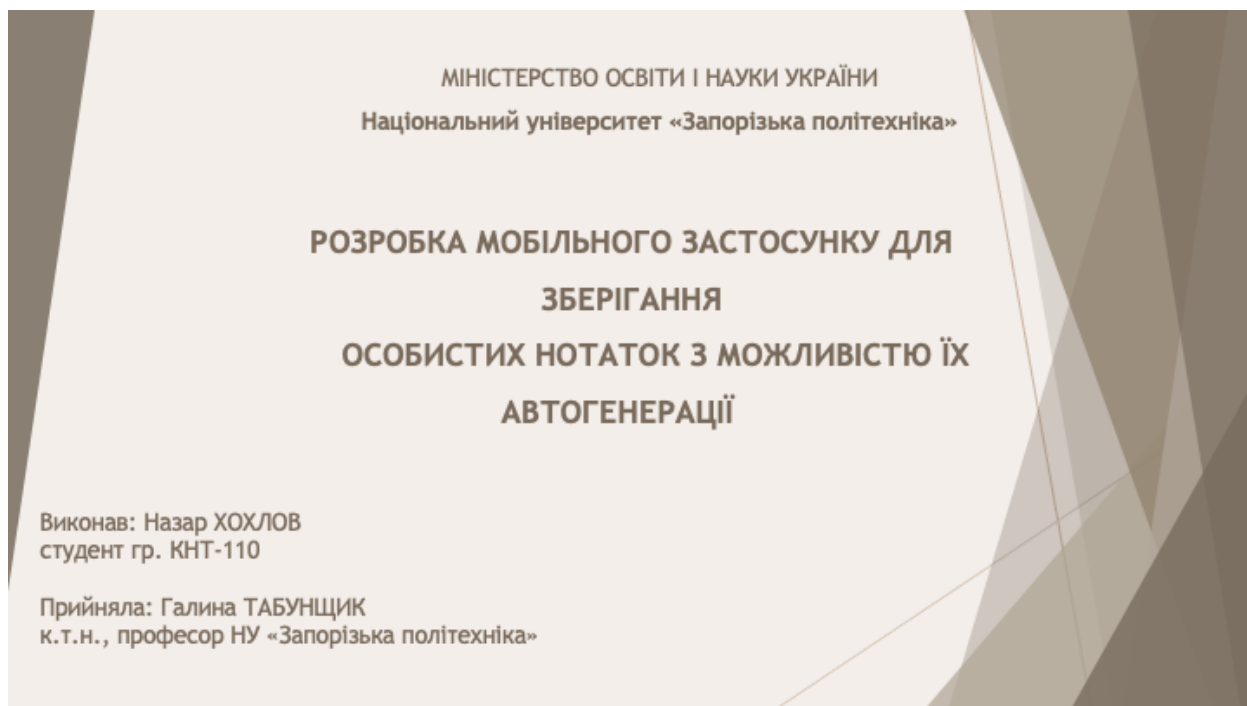


Рисунок В.1 – Слайд 1

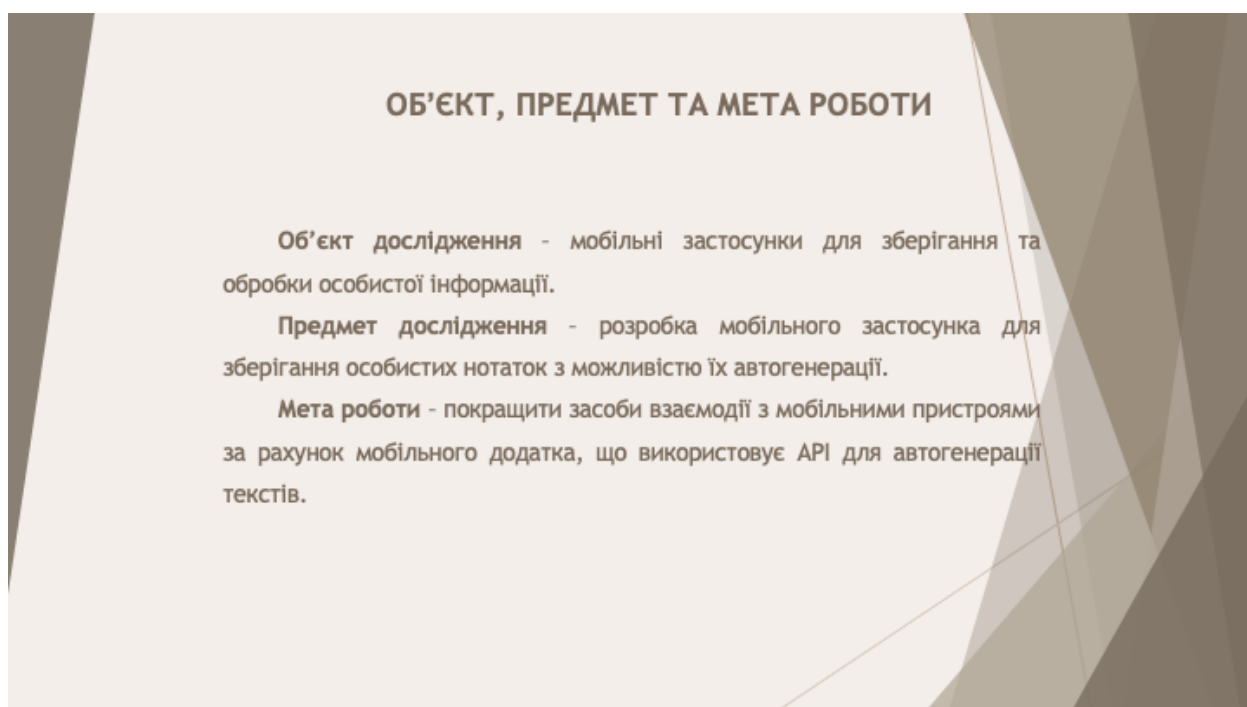


Рисунок В.2 – Слайд 2

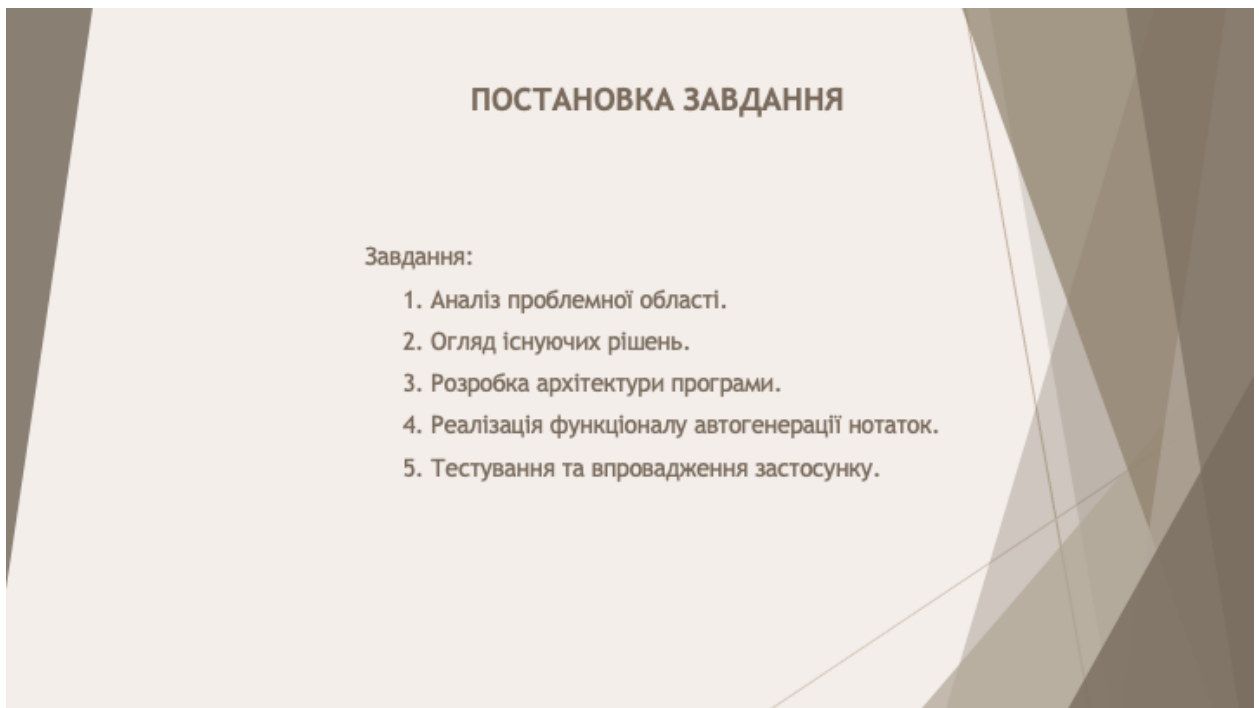


Рисунок В.3 – Слайд 3

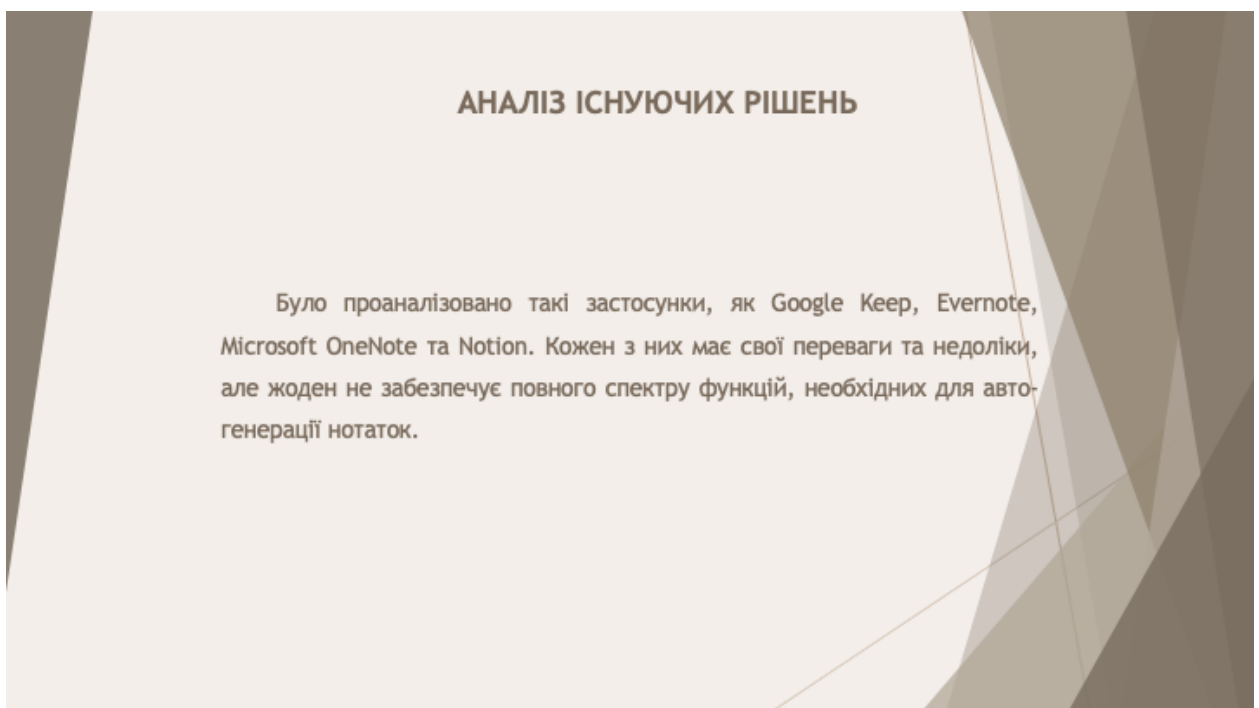


Рисунок В.4 – Слайд 4

ВИБІР ПЛАТФОРМИ ДЛЯ РОЗРОБКИ

Характеристика	Android	iOS
Частка ринку	70.79%	28.46%
Відкритість	Відкрита	Закрита
Інструменти розробки	Android Studio та інші	Xcode
Інтеграція з Google сервісами	Так	Обмежена
Вартість розробки	Нижча	Вища
Доступність на різних пристроях	Широкий спектр виробників	Тільки на пристроях Apple
Персоналізація	Висока	Обмежена
Апаратні вимоги	Різноманітні, можуть варіюватися від пристрою до пристрою	Стандартізовані, оптимізовані для пристроїв Apple
Контроль над застосунками	Свобода завантаження застосунків з різних джерел	Суворий контроль застосунків через App Store

Рисунок В.5 – Слайд 5

ВИБІР МОВИ ПРОГРАМУВАННЯ

Критерій	Java	Kotlin	C/C++	C#
Швидкість роботи	+	±	±	-
Плагіни та розширення	+	++	+ (залежить від середовища розробки)	+
Парадигми програмування	±	++	+	±
Спільнота	++	+	±	±

Рисунок В.6 – Слайд 6

ВИБІР СЕРЕДОВИЩА РОЗРОБКИ

	Android Studio	Eclipse	Xamarin (Visual Studio)
GUI	++	++	++
Продуктивність	+	++	+
Служба підтримки	+	+	+
Українська мова	-	+	+
Підсвічування синтаксису	++	+	+
Рефакторинг	+	+	+
Автодоповнення коду	++	+	+
Пошук по проєкту	+	+	+
Налагодження	+	+	+
Тестування застосунків	+	+	+
Мови, які підтримуються	Java, Kotlin, C++	Java, C++	C#

Рисунок В.7 – Слайд 7

ЗАГАЛЬНА СТРУКТУРА ЗАСТОСУНКУ

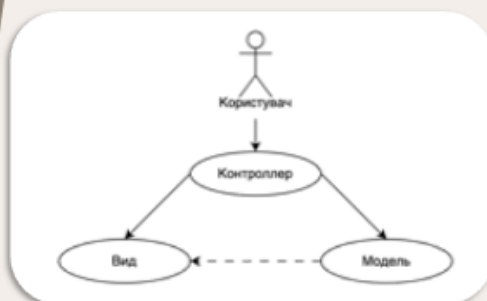


Схема архітектури MVC

Firebase - це хмарна платформа, розроблена компанією Google, яка надає широкий спектр сервісів для розробки мобільних та веб- застосунків.

GPT-3 - це велика мовна модель, розроблена компанією OpenAI.

Рисунок В.8 – Слайд 8

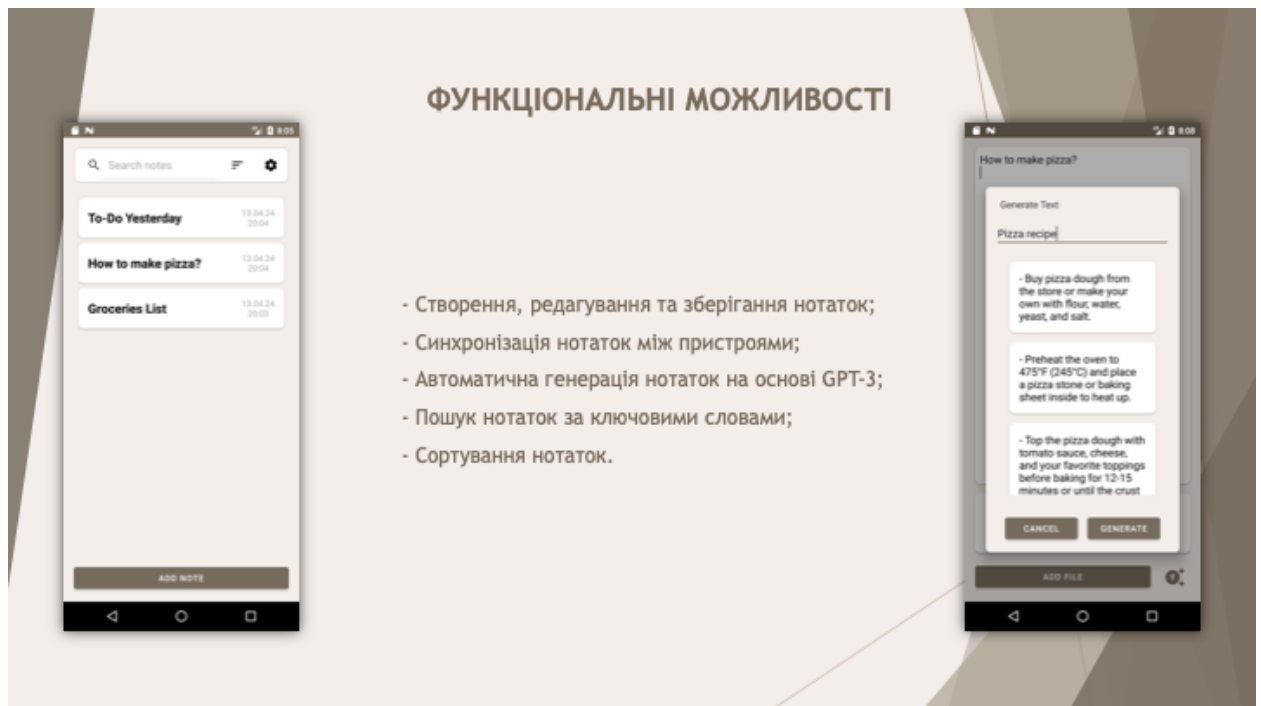


Рисунок В.9 – Слайд 9

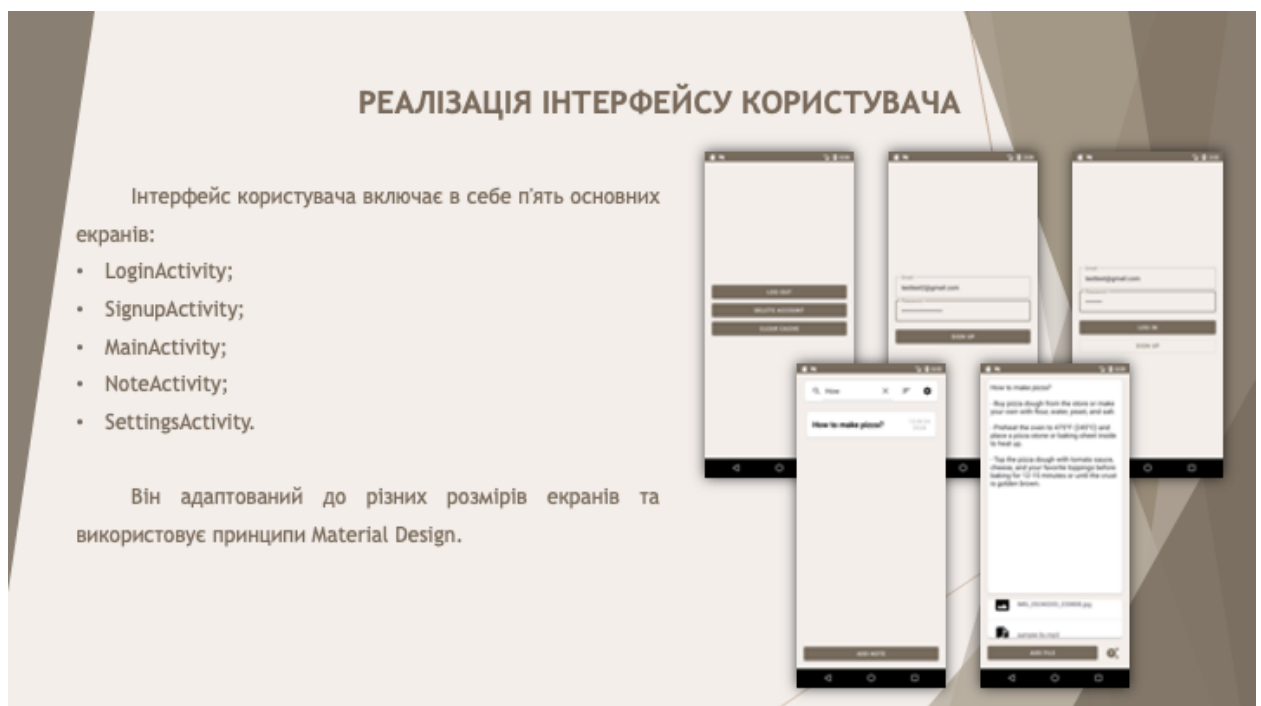


Рисунок В.10 – Слайд 10

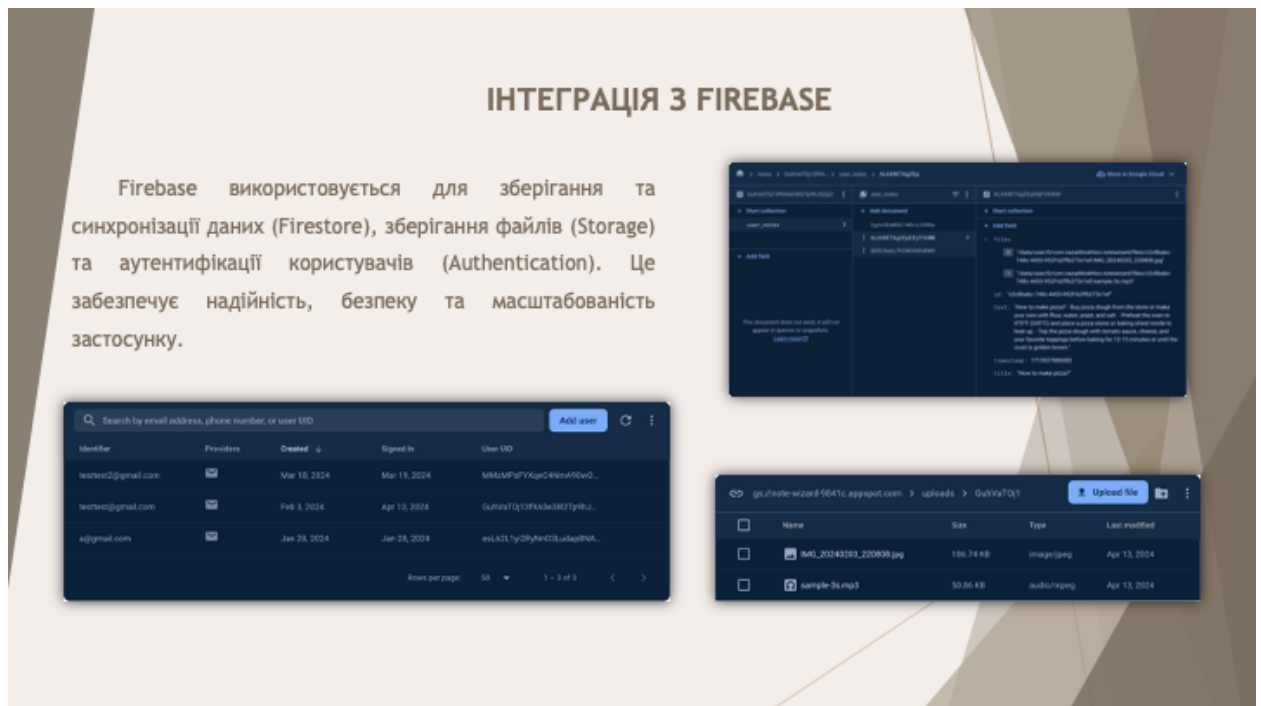


Рисунок В.11 – Слайд 11



Рисунок В.12 – Слайд 12

РЕАЛІЗАЦІЯ ПОШУКУ ТА СОРТУВАННЯ НОТАТОК

Реалізовано функціонал пошуку нотаток за ключовими словами та сортування за назвою та датою створення. Це дозволяє користувачам легко знаходити потрібні нотатки та організовувати їх зручним для себе способом.

Алгоритм пошуку:



Алгоритм сортування:



Рисунок В.13 – Слайд 13

РЕЗУЛЬТАТИ ТА ВИСНОВКИ

У дипломній роботі було виконано наступні завдання:

- розглянуто проблему ефективного зберігання та організації особистої інформації;
- проаналізовано існуючі мобільні застосунки для нотаток та виявлено їх переваги та недоліки;
- виконано проєктування архітектури мобільного застосунку;
- розроблено мобільний застосунок для зберігання особистих нотаток з можливістю їх автогенерації;
- розроблено сценарій та проведено тестування.

В результаті було розроблено мобільний застосунок з наступними можливостями:

- реєстрація користувача;
- авторизація користувача;
- створення та редагування нотаток;
- автогенерація нотаток на основі заданого тексту;
- зберігання нотаток та файлів у хмарному сховищі та їх синхронізація між пристроями користувача;
- пошук нотаток за ключовими словами та сортування за назвою і датою створення;
- управління обліковим записом та кешем застосунку.

Рисунок В.14 – Слайд 14

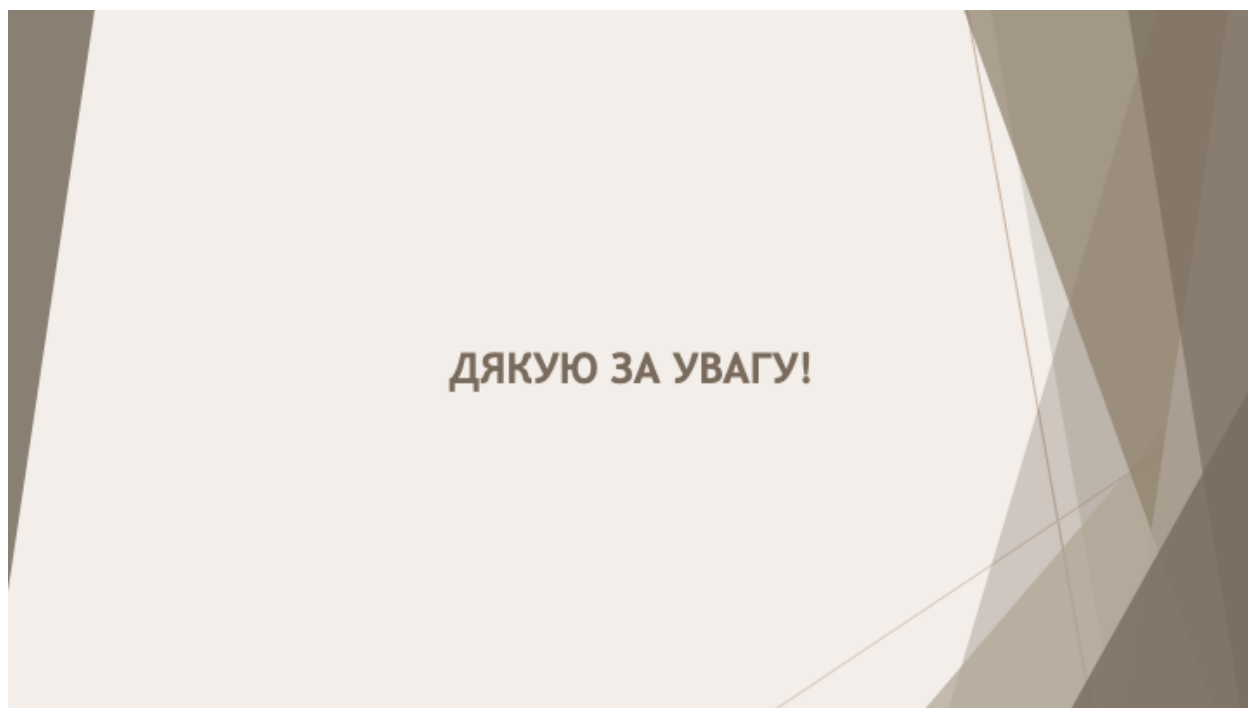


Рисунок В.15 – Слайд 15