ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка застосунку "Нотна бібліотека"»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Антон КУЛАЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-44
Антон КУЛАЙ

Керівник: Олесь ДІБРІВНИЙ
доктор філософії (PhD)

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____Ірина ЗАМРІЙ
« ____» _____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Кулаю Антону Сергійовичу

1. Тема кваліфікаційної роботи: «Розробка застосунку "Нотна бібліотека"» керівник кваліфікаційної роботи доктор філософії (PhD) Олесь ДІБРІВНИЙ, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про розробку мобільних додатків із використанням .NET MAUI, принципи побудови архітектури програмних систем за шаблоном MVVM, документація щодо використання хмарних сервісів Firebase (Authentication, Firestore, Storage), а також практичні рекомендації щодо реалізації інтерфейсу користувача, організації нотних бібліотек, пошуку та синхронізації даних у кросплатформених застосунках.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз існуючих методів і технологій мобільної розробки для організації цифрових нотних бібліотек.
2. Проектування мобільного застосунку «Нотна бібліотека» на базі платформи .NET MAUI.
3. Програмна реалізація та опис функціонування мобільного застосунку «Нотна бібліотека».
4. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. ER-діаграма.
6. Діаграма класів.
7. Екранні форми.
8. Демонстрація роботи застосунку.
9. Апробація результатів дослідження.

6. Дата видачі завдання«25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Формування вимог та проектування архітектури застосунку	14.03–25.03.2025	
4	Розробка мобільного застосунку «Нотна бібліотека» на .NET MAUI	26.03–15.04.2025	
5	Реалізація функціоналу пошуку, перегляду та керування нотами	16.04–25.04.2025	
6	Тестування застосунку	26.04–28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Антон КУЛАЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Олесь ДІБРІВНИЙ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 51 стор., 4 табл., 31 рис., 20 джерел.

Мета роботи – спрощення доступу до нотної інформації та підвищення зручності її використання за допомогою мобільного додатку нотної бібліотеки.

Об'єкт дослідження – процес цифрової каталогізації, зберігання та відтворення нотного матеріалу за допомогою мобільних пристроїв.

Предмет дослідження – процес цифрової каталогізації, зберігання та відображення нотного матеріалу за допомогою мобільних пристроїв.

Короткий зміст роботи: У роботі проаналізовано сучасні підходи, інструменти та технології для розробки мобільних застосунків на базі C# та .NET MAUI з використанням хмарного сховища Firebase. Розглянуто архітектуру та життєвий цикл .NET MAUI-застосунку, способи зберігання даних у Firestore, особливості реалізації авторизації через Firebase Authentication, а також методи обробки PDF-файлів нот. Розроблено алгоритм роботи застосунку та програмно реалізовані ключові функціональні можливості, зокрема: пошук нот за назвою, композитором або жанром; перегляд нот у форматі PDF; додавання нот до персональних папок; авторизація та відновлення паролю через електронну пошту; динамічне завантаження та синхронізація даних з Firebase. Проведено функціональне тестування основних компонентів додатку. В роботі використано такі інструменти: бібліотека Syncfusion.Maui.PdfViewer для відображення нот, Firebase Realtime Database та Firestore для зберігання даних, .NET MAUI Community Toolkit для реалізації MVVM-архітектури.

Сферою використання застосунку є індивідуальна робота музикантів та освітній процес у музичних закладах.

КЛЮЧОВІ СЛОВА: .NET MAUI, C#, FIREBASE, FIRESTORE, PDF-НОТИ, МОБІЛЬНИЙ ДОДАТОК, MVVM, СИНХРОНІЗАЦІЯ ДАНИХ, КОРИСТУВАЦЬКІ ПАПКИ, МУЗИЧНА БІБЛІОТЕКА

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення

MAUI – Multi-platform App UI

.NET – платформа розробки від Microsoft

MVVM – Model-View-ViewModel

PDF – Portable Document Format

JSON – JavaScript Object Notation

API – Application Programming Interface

UI – User Interface (Інтерфейс користувача)

ORM – Object-Relational Mapping

REST – Representational State Transfer

HTTP – HyperText Transfer Protocol

EF Core – Entity Framework Core

XAML – eXtensible Application Markup Language

DI – Dependency Injection

ID – Identifier

FAQ – Frequently Asked Questions

VM – ViewModel

URI – Uniform Resource Identifier

OAuth – Open Authorization (протокол авторизації)

SDK – Software Development Kit

SQLite – легка вбудована реляційна база даних

Firebase – платформа для розробки хмарних застосунків від Google

Firestore – хмарна NoSQL-база даних від Firebase

UI/UX – User Interface / User Experience

ВСТУП

Актуальність: Збереження, систематизація та використання нотних матеріалів є важливою складовою музичної освіти, самонавчання та професійної діяльності виконавців. У сучасних умовах цифровізації все більше музикантів переходять від паперових нот до електронного формату, що дозволяє значно підвищити зручність доступу, організації та обміну навчальними матеріалами. Водночас, більшість існуючих рішень або орієнтовані на комерційне використання, або не забезпечують повноцінної інтеграції персональної нотної бібліотеки, функцій фільтрації, структуризації та перегляду нот у зручному форматі. Саме тому розробка мобільного додатку для зберігання нот із можливістю додавання до папок, пошуку за жанром, композитором або назвою, а також перегляду нот у форматі PDF є актуальним напрямком для підтримки навчального процесу, музичної практики та персональної архівації матеріалів.

Мобільний застосунок забезпечує гнучкість, доступність у будь-який момент часу та можливість синхронізації даних між пристроями. Це особливо важливо для студентів музичних навчальних закладів, викладачів, а також музикантів- практиків, які хочуть мати швидкий доступ до власної нотної бібліотеки з будь- якого пристрою. Таким чином, запропонований застосунок сприяє підвищенню ефективності музичного навчання та цифрової організації навчальних ресурсів.

Об'єктом дослідження є процес цифрової каталогізації, зберігання та відображення нотного матеріалу за допомогою мобільних пристроїв.

Предметом дослідження є механізми, засоби та принципи реалізації мобільного застосунку для цифрової каталогізації, зберігання та управління нотним матеріалом.

Метою роботи є спрощення доступу до нотної інформації та підвищення зручності її використання за допомогою мобільного додатку нотної бібліотеки

Досягнення поставленої мети передбачає реалізацію наступних завдань:

1. Провести аналіз сучасних технологій для розробки кросплатформених мобільних додатків, зокрема C# та .NET MAUI.

2. Вивчити інструменти хмарного зберігання та синхронізації даних, зокрема Firebase Firestore і Firebase Authentication.

3. Розробити архітектуру мобільного додатку з використанням шаблону MVVM.

4. Реалізувати функціонал авторизації, відновлення паролю та зберігання інформації про користувача.

5. Здійснити інтеграцію з хмарною базою даних для зберігання нот, папок та метаданих.

6. Реалізувати функцію перегляду нот у форматі PDF за допомогою зовнішнього компоненту (Syncfusion PDF Viewer).

7. Забезпечити можливість пошуку нот за назвою, композитором, жанром і категорією.

8. Надати користувачу можливість створення персональних папок, додавання та видалення нот.

9. Провести тестування працездатності основних функцій додатку та перевірити синхронізацію між пристроями.

10. Підготувати користувацький інтерфейс, що відповідає сучасним вимогам зручності, адаптивності та простоти.

Практичне значення результатів: Розроблений мобільний застосунок призначений для організації, збереження та перегляду нотних матеріалів користувача у зручному цифровому форматі. Він дозволяє створювати персональні папки, додавати до них ноти, переглядати їх у форматі PDF, а також здійснювати пошук за назвою, композитором або жанром. Для реалізації функціоналу застосунку використовувалися мова програмування C#, кросплатформений фреймворк .NET MAUI, хмарний сервіс Firebase (Authentication, Firestore, Storage) та середовище розробки Visual Studio 2022.

Практична значущість результатів полягає у створенні зручного та функціонального інструменту для студентів, викладачів музичних навчальних закладів та музикантів, який дозволяє швидко отримувати доступ до нотної бібліотеки, впорядковувати навчальні матеріали та зберігати їх в одному місці з можливістю доступу з будь-якого пристрою. Застосунок сприяє підвищенню ефективності музичного навчання та самостійної роботи.

1 АНАЛІЗ СИСТЕМ ДЛЯ РОБОТИ З НОТАМИ

1.1 Опис предметної галузі – нотні бібліотеки

Нотні бібліотеки як складова музичної культури мають тривалу історію й відіграють ключову роль у збереженні, поширенні та виконанні музичних творів. Починаючи з епохи середньовіччя, коли нотна грамота записувалась вручну на пергаменті, до сьогодення, коли зберігання партитур здійснюється у вигляді цифрових файлів, людство постійно вдосконалювало способи організації та доступу до музичних джерел. У своїй класичній формі нотна бібліотека є зібранням паперових видань, систематизованих за алфавітом, жанром, автором або складом виконавців. Такі бібліотеки широко представлені у вищих навчальних закладах мистецького спрямування, в оркестрах, філармоніях, а також у приватних колекціях виконавців і педагогів.

З розвитком комп’ютерних технологій з’явилася потреба трансформувати традиційні нотні фонди у цифровий формат. Цей процес пов’язаний не лише з прагненням зменшити фізичний простір для зберігання матеріалів, а й з бажанням покращити доступність нот, спростити їхню навігацію, забезпечити підтримку інтерактивних елементів і мультимедійного супроводу [1], [2]. Таким чином, сучасна нотна бібліотека – це не просто набір відсканованих партитур, а складний програмний комплекс, що виконує функції архівування, пошуку, відтворення та взаємодії з користувачем.

Однією з характерних особливостей предметної галузі є її поєднання мистецького змісту з цифровими технологіями, що вимагає від розробника не лише інженерних навичок, а й розуміння специфіки роботи з музичними матеріалами. Наприклад, функціональність пошуку нот за назвою твору, жанром або ім’ям композитора є базовою, але водночас необхідною складовою для зручного доступу до великої нотної колекції. Особливу увагу у проєкті приділено підтримці формату PDF, який, хоча й не призначений для машинного аналізу нотного тексту,

залишається найпоширенішим і найзручнішим способом графічного представлення нот. Саме тому застосунок орієнтовано на стабільне та якісне відображення нот у PDF-форматі з можливістю збереження, перегляду та структуризації нотних файлів у персональних папках користувача. Це забезпечує комфортне використання застосунку як у навчальному процесі, так і в повсякденній роботі музикантів.

Створення нотних бібліотек для мобільних пристроїв породжує нові завдання, пов’язані з оптимізацією інтерфейсу, забезпеченням читабельності нотного стану на різних розмірах екранів, синхронізацією між пристроями, а також збереженням персоналізованих налаштувань. Користувач, який працює з нотами на планшеті, очікує, що система дозволить не лише знаходити та переглядати потрібні твори за назвою або ім’ям композитора , але й завантажувати твори на пристрій, зберегти улюблені твори у спеціальні добірки або папки а також ділитися ними з колегами або учнями. Таким чином, нотна бібліотека стає не лише сховищем, а й інструментом навчання, виступів та творчої взаємодії.

Предметна область охоплює також різноманітні категорії користувачів, кожна з яких має свої специфічні потреби. Студенти музичних шкіл, академій та консерваторій зазвичай шукають матеріали для підготовки до іспитів, викладачі – для проведення занять, акомпаніатори – для виступів, а аматори – для самостійного вивчення. Отже, гнучкість та масштабованість цифрової системи є критичними факторами її успішного впровадження.

У рамках даної дипломної роботи предметна область розглядається як поєднання трьох ключових компонентів: це організація нотного контенту у структуровану бібліотеку з можливістю розширеного пошуку; це забезпечення інтуїтивної, комфортної та ефективної взаємодії користувача з цим контентом за допомогою сучасного інтерфейсу; і, нарешті, це технічна реалізація функцій, що підвищують цінність додатку у порівнянні з існуючими аналогами. Саме така комплексна модель предметної галузі дозволяє сформувати цілісне бачення майбутнього продукту та сформулювати обґрунтовані вимоги до його функціоналу, дизайну й архітектури.

1.2 Огляд існуючих програмних продуктів

У сучасному цифровому середовищі існує низка програмних рішень, призначених для роботи з нотними матеріалами. Ці продукти варіюються від простих переглядачів до комплексних систем для створення, редагування та організації нот. У цьому розділі розглянемо найбільш відомі та функціональні додатки, зокрема ті, що мають українське походження або орієнтовані на українську аудиторію.

На рисунку 1.1 зображено мобільний застосунок Ukrainian Live Classic, створений з метою популяризації української класичної музики.

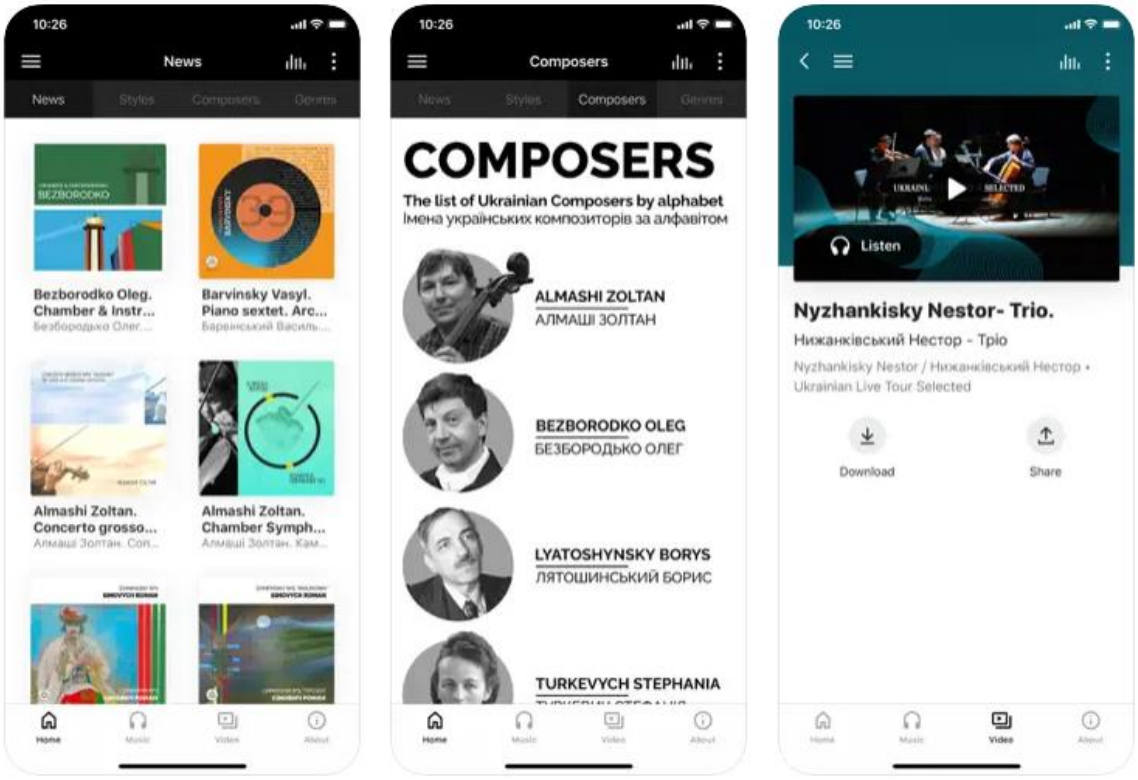


Рис.1.1 Екрани мобільного застосунку Ukrainian Live Classic

Він дозволяє прослуховувати аудіозаписи професійних виконань творів українських композиторів, переглядати відеоматеріали концертів, читати історичні довідки, коментарі музикознавців, а також зберігати вподобані твори у власну добірку. Особливістю додатку є те, що весь контент адаптований саме до

українського культурного контексту: користувач отримує доступ до творів, які не представлені на міжнародних платформах. Застосунок має сучасний інтерфейс, підтримує швидкий пошук за автором чи жанром, а також дозволяє налаштовувати списки відтворення. На відміну від більшості комерційних сервісів, Ukrainian Live Classic є безкоштовним і створений на основі культурної ініціативи [20].

Ukrainian Scores – це онлайн-архів нот українських композиторів, розроблений як доповнення до Ukrainian Live Classic, головна сторінка якого зображена на рисунку 1.2.

Рис.1.2 Головна сторінка Ukrainian Scores

Сервіс не є мобільним застосунком у вузькому розумінні, проте має адаптований веб-інтерфейс для мобільних пристроїв. Його основна цінність – відкритий доступ до понад трьох тисяч нотних видань, багато з яких не представлені в інших базах даних. Користувачі можуть шукати твори за автором, роком написання або жанром, а також завантажувати партитури у форматі PDF [4]. Система також дозволяє подавати індивідуальні запити на отримання нот, що надає можливість виконавцям працювати з унікальним українським репертуаром. Весь контент проходить перевірку на авторські права та надається лише у рамках чинного законодавства.

На рисунку 1.3 зображено MuseScore – один із найвідоміших і найпотужніших додатків для роботи з нотами у світі

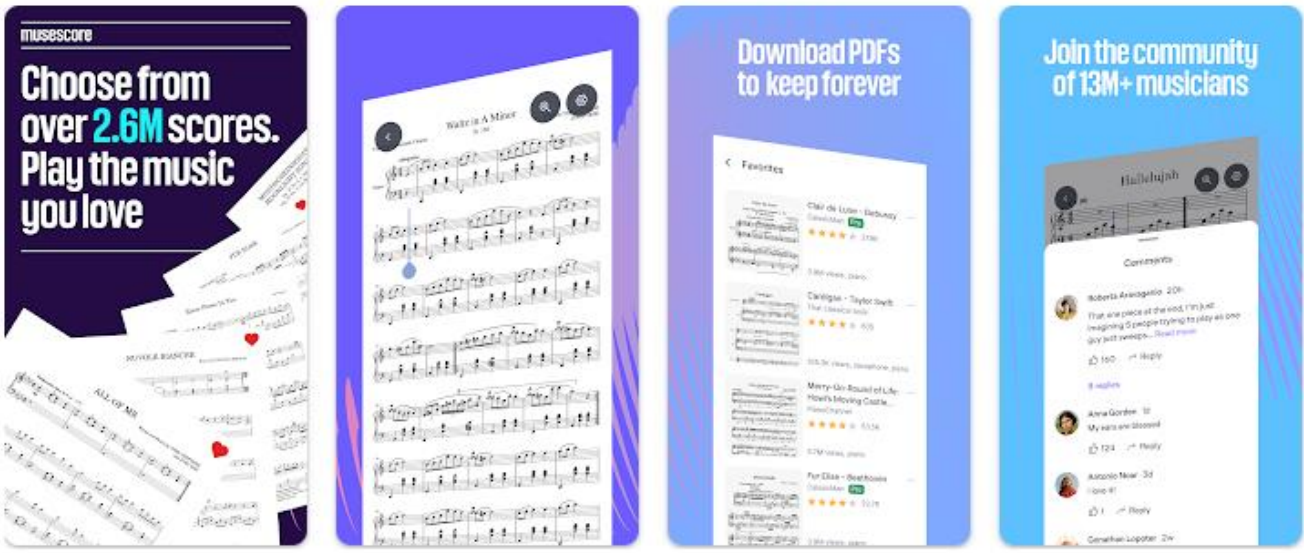


Рис.1.3 Екрани мобільного застосунку MuseScore

Це повноцінний нотний редактор з відкритим кодом, який дозволяє створювати, редагувати, імпортувати та експортувати ноти у форматах MusicXML, MIDI, PDF тощо. MuseScore має величезну онлайн-бібліотеку з мільйонами партитур [3], які можна зберігати, ділитися ними з іншими користувачами або інтегрувати у власні добірки. Інтерфейс програми доступний на різних мовах, у тому числі українською, а також містить довідкові матеріали, підказки, інтерактивні приклади. У версіях для Android та iOS доступна функція прослуховування нот, підсвічування тактів під час програвання та синхронізація між пристроями через обліковий запис MuseScore. Цей додаток – універсальний інструмент як для професійних композиторів, так і для учнів та викладачів.

Flat: Music Score & Tab Editor – сучасний хмарний нотний редактор, який підтримує роботу в режимі реального часу. Особливістю Flat є спрощений інтерфейс, який дозволяє створювати ноти навіть без попередньої музичної підготовки. Додаток ідеально підходить для співпраці: кілька користувачів можуть редагувати один файл одночасно. Крім нотної грамоти, Flat підтримує гітарні

табулатури, барабанні партії, акордові символи. Екрани цього додатку зображено на рисунку 1.4.

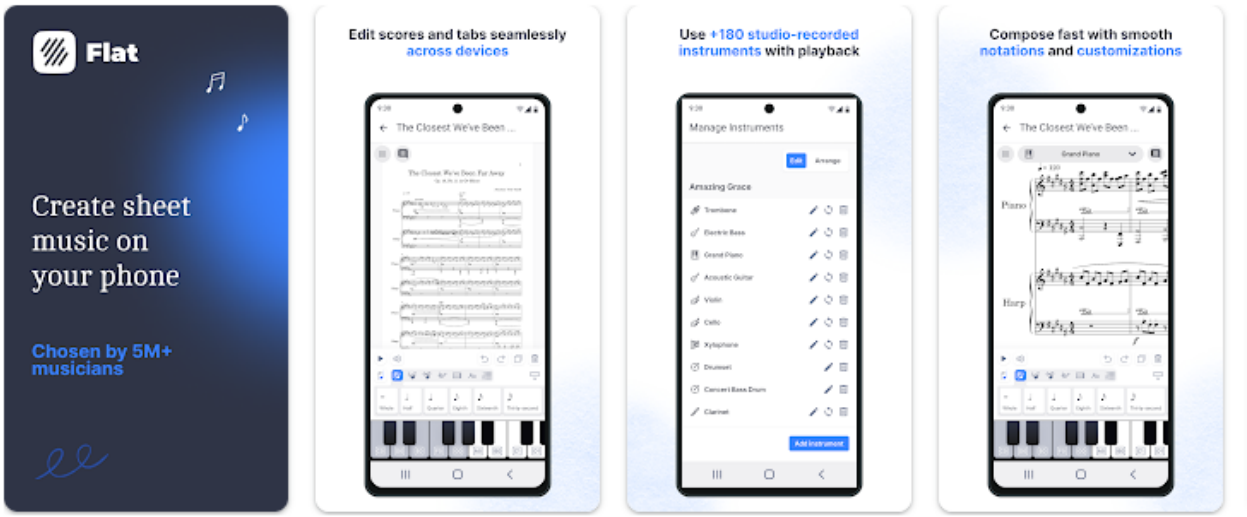


Рис.1.4 Екрани мобільного застосунку Flat: Music Score & Tab Editor

Усі зміни зберігаються в реальному часі, а завершені композиції можна експортувати у форматах MusicXML або MIDI. У додатку передбачено можливість інтеграції з Google Classroom, що робить його особливо привабливим для шкіл, студій і університетів. Flat – це приклад того, як нотний сервіс може працювати у вигляді SaaS-платформи з широкими можливостями [5].

Score Creator – мобільний застосунок для Android, орієнтований на створення нот і простих композицій. Екран цього додатку зображено на рисунку1.5.

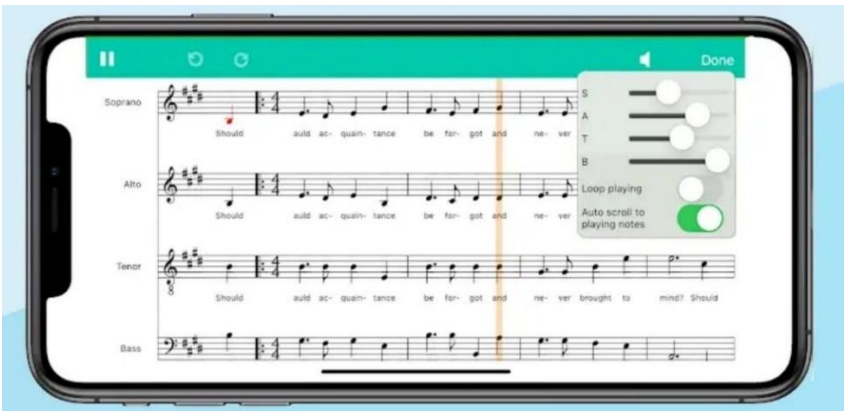


Рис.1.5 Екрани мобільного застосунку Score Creator

Його перевагою є легкість у використанні: користувач може вводити ноти, акорди, тексти пісень та одразу слухати результат. Програма має вбудований синтезатор, який дозволяє прослуховувати кожен фрагмент, а також підтримує створення багатоголосих партитур. Score Creator підходить для музичних ідей, створення шкільних завдань або швидкої чернетки для подальшої роботи на ПК. Незважаючи на обмеженість щодо форматів виводу, додаток має стабільну аудиторію та добрі відгуки від викладачів і музикантів-любителів.

Для кращого розуміння переваг та недоліків даних розглянутих рішень було внесені дані до порівняльної таблиці 1.1 нотних додатків.

Таблиця 1.1

Порівняльна таблиця існуючих нотних додатків

Додаток	Ukrainian Live Classic	Ukrainian Scores	MuseScore	Flat.io	Score Creator	MyLibraryApp
Створення власних папок	-	-	+	+	-	+
Пошук за назвою/жанром/композитором	+	+	+ (в акаунті)	+	-	+
Підтримка кирилиці	+	+	+	+	+	+
Перегляд нот у PDF	+	+	+	+	+	+
Редагування нот	-	-	+	+ (браузерний редактор)	+(мінімалістичний)	-
Аудіо-програвання	+ (аудіозаписи)	-	+	+	+(MIDI)	-
Синхронізація між пристроями	-	-	+	+	-	+
Завантаження нот з пристрою	-	-	+	+	+	+(обмежено)
Експорт нот у PDF/MusicXML	-	-	+ (через акаунт)		+(лише PDF)	+(лише PDF)
Спільний доступ до нот	-	-	+ через акаунт)	+ (публічні посилання)	-	+ (публічні посилання)
Офлайн-доступ	+	+	+(в додатку)	-	+	-
Авторизація користувача			+	+	-	+

Продовження таблиці 1.1

Порівняльна таблиця існуючих нотних додатків

Платформи	Android, iOS	Web	Android, iOS, Web	Android, iOS, Web	-	Android, iOS
Безкоштовність	+	+	+ (більшість функцій платні)	+ (з платними функціями)	-	+

При порівнянні наведених додатків можна відзначити, що українські розробки, такі як Ukrainian Live Classic та Ukrainian Scores, зосереджені на популяризації української класичної музики та наданні доступу до нот українських композиторів. Вони мають культурно-просвітницьку спрямованість і є безкоштовними для користувачів.

Міжнародні додатки, такі як MuseScore, Flat та Score Creator хоча і платні рішення але пропонують ширший спектр функцій для створення та редагування нот, підтримують різні формати файлів та мають великі спільноти користувачів. Вони можуть бути корисними для композиторів, виконавців та викладачів музики, які працюють з різними жанрами та стилями.

Аналіз існуючих програмних продуктів показує, що хоча на ринку доступні різноманітні рішення для перегляду та редагування нот, більшість із них або орієнтовані на професійну композицію, або не забезпечують просту організацію нотної бібліотеки в мобільному форматі. Водночас функції, як-от створення власних папок, пошук за назвою чи композитором, зручний перегляд нот у PDF, а також синхронізація між пристроями, рідко поєднуються в одному додатку.

Це створює передумови для розробки нового застосунку, що поєднує найкращі властивості існуючих рішень і реалізує інтуїтивний, простий у користуванні інструмент для зберігання та перегляду нот, орієнтований на потреби студентів, викладачів і музикантів-практиків.

1.3 Визначення вимог до застосунку

Враховуючи проведений аналіз предметної галузі, а також огляд наявних аналогів, у цьому розділі сформульовано комплекс функціональних, нефункціональних, інтерфейсних та технічних вимог до мобільного застосунку «Нотна бібліотека».

Перш за все, при формуванні функціональних вимог враховувалася потреба користувачів мати швидкий і зручний доступ до нотного матеріалу. Основною функцією є реалізація пошуку нот за назвою твору, ім’ям композитора та жанровою належністю. Такий підхід забезпечує базову навігацію по нотному архіву та дозволяє легко знаходити необхідні матеріали для навчання, репетицій чи виступів.

Крім пошуку, важливу роль відіграють функції персоналізації. Застосунок дозволяє створювати власні папки, до яких можна додавати потрібні нотні матеріали, зокрема вручну вводити назву твору та, за бажанням, ім’я композитора. Ноти, що зберігаються у персональних папках, можна переглядати, відкривати та видаляти. Це дає змогу структурувати нотну бібліотеку під індивідуальні потреби користувача.

Особливу увагу приділено зручному перегляду нот у форматі PDF. Завдяки інтеграції з PDF-переглядачем користувач може швидко відкривати ноти у зручному форматі без сторонніх програм. Також реалізовано можливість завантаження нот на пристрій або їх поширення через інші застосунки. Підтримка хмарного збереження через Firebase Firestore дозволяє синхронізувати дані між різними пристроями та зберігати персональні добірки нот у хмарі [6].

Окремим функціональним модулем є розділ «Налаштування», який дозволяє перемикати тему оформлення (світла/темна), переглядати акаунт користувача, вийти з системи або повністю видалити акаунт. Також реалізовано перехід до розділів «Підтримка» та «FAQ», а для відновлення доступу передбачено надсилання листа на електронну пошту.

У межах нефункціональних вимог визначено необхідність стабільної роботи застосунку на пристроях із різними розмірами екранів. Інтерфейс реалізовано з урахуванням адаптивної верстки та масштабування елементів, що забезпечує коректну роботу як на смартфонах, так і на планшетах. Забезпечено підтримку темної теми, що покращує комфорт роботи в умовах низької освітленості.

З технічного погляду застосунок побудовано на основі фреймворку .NET MAUI, що забезпечує кросплатформенну підтримку Android та iOS при єдиній кодовій базі. Для збереження нот та пов’язаних метаданих використано сервіси Firebase: Firestore для структурування даних, Firebase Storage для зберігання PDF-файлів та Firebase Authentication для організації авторизованого доступу користувачів і синхронізації бібліотеки.

Інтерфейсні вимоги передбачають просту, інтуїтивну навігацію, чітке структурування сторінок («Моя бібліотека», «Пошук», «Налаштування») та мінімалістичний дизайн. На головних екранах забезпечено швидкий доступ до функцій пошуку, створення нових папок, перегляду нот, а також керування обліковим записом.

1.4 Цільова аудиторія

Чинник, що визначає успішність програмного продукту, є правильне розуміння та аналіз цільової аудиторії. У випадку мобільного застосунку «Нотна бібліотека» користувачем виступає не лише умовний «музикант», а широкий спектр осіб, пов’язаних з музичною практикою, освітою, творчістю, а також організацією навчального процесу або концертної діяльності. Визначення портретів користувачів дозволяє сформувати вимоги до інтерфейсу, доступності, гнучкості функціональності та організації зберігання даних.

Умовно всю цільову аудиторію проєкту можна поділити на кілька основних категорій, кожна з яких має свої характерні потреби, очікування та сценарії використання. До першої групи належать учні та студенти мистецьких навчальних закладів: шкіл, училищ, консерваторій, факультетів музичного мистецтва. Для цієї

категорії важливими є функції збереження власної бібліотеки нот, створення тематичних добірок, а також гнучкий пошук за жанром чи композитором. Часто ці користувачі навчаються у різних умовах – вдома, у транспорті, у залі – і потребують офлайн-доступу до нот, швидкого відкриття PDF-файлів і зручного інтерфейсу без перевантаження зайвими елементами.

Другу категорію представляють викладачі музичних дисциплін, які використовують нотний матеріал не лише як засіб виконавства, а й як навчальний інструмент. Їм необхідно мати змогу швидко шукати й сортувати твори за жанром, композитором або тематикою. Додатково корисною є можливість створення добірок для занять або уроків, а також зручна навігація у структурі нотної бібліотеки. Зважаючи на обмежений час заняття, викладачам важливо, щоби доступ до нот був швидким, а перемикання між сторінками – інтуїтивним.

До третьої категорії відносяться виконавці – професійні музиканти, концертуючі піаністи, скрипалі, вокалісти, диригенти, акомпаніатори. Їхня робота часто передбачає потребу у швидкому доступі до нотного матеріалу, зокрема з мобільних пристроїв. Для цієї групи критично важливими є стабільність роботи додатку, збереження нот у хмарі, перегляд у форматі PDF та можливість створення власних структурованих бібліотек для репетицій або виступів.

Четверту групу становлять аматори, любителі музики, самоучки, які навчаються грі на інструменті або просто хочуть мати доступ до репертуару для особистого задоволення. Ця категорія часто має нижчий рівень підготовки, тому очікує простоти інтерфейсу, зрозумілого дизайну, а також можливості переглядати ноти у зручному форматі без зайвих дій. Бажаною є також можливість зберігати власну добірку творів без складної реєстрації чи налаштувань.

Окремо варто виділити організаторів подій, методистів, викладачів, відповідальних за підготовку та зберігання навчального або концертного нотного матеріалу. У контексті цифрової бібліотеки ці користувачі можуть використовувати систему як інструмент збереження нот у хмарному середовищі, впорядкування за темами або жанрами, а також швидкого доступу до потрібного репертуару.

2 ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Середовище розробки

У процесі створення сучасних програмних продуктів головну роль відіграє вибір середовища розробки. Від цього вибору залежить не лише зручність написання коду, а й ефективність налагодження, підтримка обраних мов програмування, інтеграція з системами контролю версій, а також можливості розгортання на цільових платформах. З огляду на мету даногоо проєкту, якою є розробка кросплатформенного мобільного застосунку для взаємодії з нотними матеріалами, було прийнято рішення використати середовище розробки Microsoft Visual Studio 2022.

Visual Studio – це одна з інтегрованих серед розробницьких середовищ (IDE), яка поєднує широкий спектр інструментів для повноцінного життєвого циклу застосунків: від написання коду і його тестування до побудови, налагодження та розгортання. У межах даного проєкту було встановлено редакцію Visual Studio 2022 Community Edition, яка є безкоштовною та водночас повнофункціональною для потреб індивідуальної розробки, навчання та дослідницьких робіт. Головна сторінка VS зображена на рисунку 2.1.

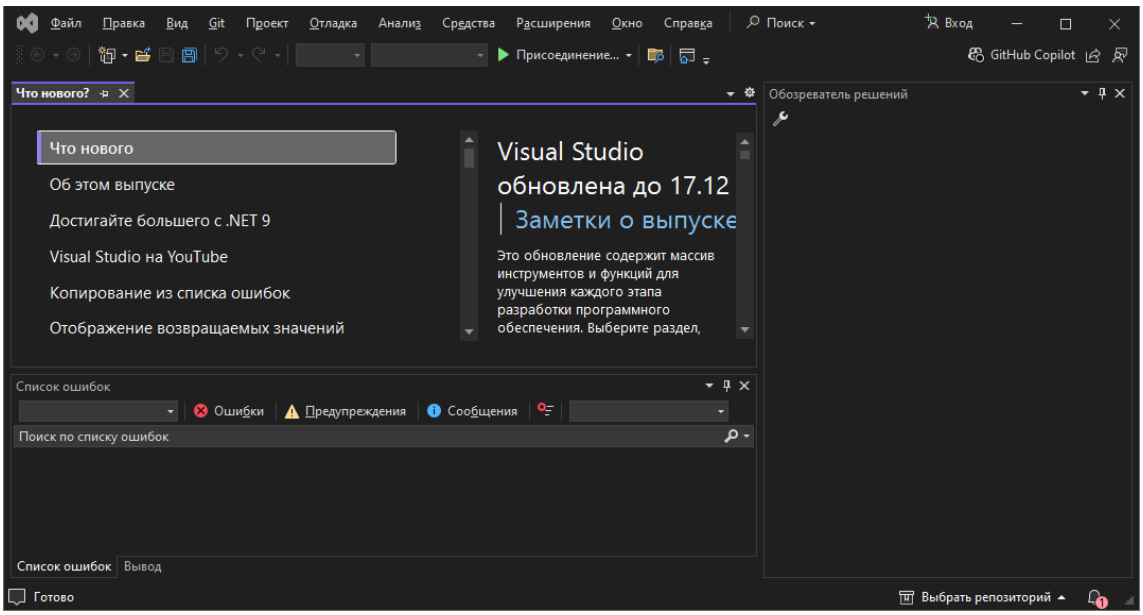


Рис. 2.1 Інтерфейс середовища Visual Studio 2022 Community Edition

Середовище Visual Studio надає підтримку найсучасніших мов програмування та фреймворків, зокрема .NET 8, який є основною технологічною платформою обраного проєкту. Важливо зазначити, що для розробки на платформі .NET Multi-platform App UI (MAUI), яка використовується у проєкті як головна технологія для створення кросплатформенного мобільного інтерфейсу, необхідним є встановлення відповідної робочої навантаження (workload), а саме .NET MAUI workload, що дозволяє створювати застосунки під Android, Windows, iOS та macOS [7], [8].

Процес налаштування середовища включав встановлення останньої версії .NET SDK (версія 8.0) та компонентів MAUI. Крім того, було додатково інтегровано бібліотеки CommunityToolkit.Maui для реалізації інтерфейсних компонентів, Firebase SDK для авторизації користувачів і збереження нот у хмарі, а також Syncfusion.Maui.PdfViewer для відображення нот у форматі PDF.

Суттєвою перевагою середовища Visual Studio є підтримка гарячого перезавантаження (hot reload), що дозволяє розробнику змінювати XAML-розмітку або C#-код без необхідності перезапуску всього застосунку. Такий підхід значно знижує час ітерацій при розробці інтерфейсу користувача та логіки взаємодії з ним.

Іншим важливим аспектом є можливість налаштування середовища під особисті потреби. Visual Studio підтримує численні розширення (extensions), які можуть бути використані для перевірки якості коду, інтеграції з GitHub та моніторингу залежностей. Завдяки цьому було забезпечено організовану структуру проєкту та ефективну роботу з репозиторієм застосунку.

Для збереження інформації про нотні файли, метадані композиторів та структури папок використовувалися сервіси Firebase Firestore і Firebase Storage. На відміну від традиційного підходу з локальною базою даних (наприклад, SQLite чи EF Core), використання Firebase дозволило реалізувати синхронізацію між пристроями та хмарне зберігання даних без необхідності налаштування локальних моделей.

Не менш важливою є підтримка розробки багатоплатформених застосунків. У ході створення додатку проводилося тестування в середовищі Android-емулятора

та на реальному Android-пристрої, що дозволило перевірити коректність адаптивного дизайну, взаємодії з сенсорними елементами інтерфейсу та відображення PDF-файлів нот. Тестування на Windows також дозволило переконатися у стабільній роботі застосунку на різних платформах.

Окрему увагу варто приділити середовищу налагодження і тестування (debugging), яке у Visual Studio підтримує одночасно декілька цільових платформ. У межах даного проєкту були використані конфігурації Debug та Release з платформо-залежним розгортанням (Android, Windows), а також були задіяні засоби трасування, моніторингу помилок і продуктивності, що дозволили досягти стабільної роботи застосунку на обраних пристроях.

Варто також зазначити, що Visual Studio надає повну підтримку роботи з XAML-маркуванням, яке використовується для опису інтерфейсів у .NET MAUI. Комбінування декларативного підходу (XAML) із мовою C# для програмної логіки дозволяє гнучко налаштовувати всі елементи взаємодії користувача із застосунком. Такий підхід є ефективним для розробки мобільного інтерфейсу, орієнтованого на швидкий доступ до нот, їх структуризацію та перегляд [18].

2.2 Мови та технології

Під час створення програмного забезпечення надзвичайно важливим є вибір мов програмування та технологій, які безпосередньо впливають на архітектурні рішення, продуктивність, масштабованість і зручність підтримки розробленої системи. У контексті даного проєкту було прийнято рішення про використання мови **C#** як основної для реалізації логіки, а також мови розмітки **XAML** для побудови інтерфейсу користувача. Обидві мови є складовими екосистеми .NET і забезпечують повноцінну підтримку в межах MAUI-проєктів.

Мова **C#** – це сучасна об’єктно-орієнтована мова програмування, розроблена компанією Microsoft, яка поєднує продуктивність та безпечність роботи з пам’яттю, багатий синтаксис та розвинену підтримку парадигм функціонального і реактивного програмування. Завдяки стабільній інтеграції з платформою .NET, C#

є природним вибором для розробки кросплатформених застосунків, де потрібна висока ефективність, масштабованість та стабільність [19].

У межах цього проєкту C# використовується для реалізації логіки авторизації, навігації між сторінками, взаємодії з Firebase, обробки REST-запитів, роботи з PDF-файлами та локального зберігання ключових параметрів користувача. Архітектура застосунку побудована на основі патерну MVVM, реалізованого через механізми INotifyPropertyChanged, Command, ObservableCollection, а також асинхронного програмування з використанням async/await.

Візуальна частина додатку реалізована за допомогою XAML **(**eXtensible Application Markup Language**)** — декларативної мови розмітки для побудови інтерфейсів користувача у .NET. У проєкті XAML використовується для створення структури сторінок, форм введення, списків, стилізованих кнопок, елементів навігації та відображення нот. Це дозволяє розділити логіку поведінки (написану на C#) і візуальну частину, що відповідає принципам чистої архітектури [9].

Основу проєкту становить .NET MAUI (Multi-platform App UI) — сучасний кросплатформенний фреймворк для розробки застосунків з єдиною кодовою базою. Він дозволяє створювати застосунки для Android, iOS та Windows без необхідності дублювання коду для кожної платформи. У проєкті використано можливості MAUI для побудови адаптивного інтерфейсу, підтримки темної теми, сенсорної взаємодії та кросплатформеного розгортання.

2.3 Фреймворки та бібліотеки

Одним із ключових чинників успішної реалізації функціонального та стабільного застосунку є вибір відповідних фреймворків і бібліотек. У цьому проєкті було використано низку перевірених рішень, що розширюють можливості базового фреймворку .NET MAUI, забезпечують зручну роботу з інтерфейсом, PDF-документами та хмарними сервісами.

Для побудови інтерфейсних компонентів використано бібліотеку CommunityToolkit**.**Maui — набір розширень, що додає зручні елементи керування, поведінки, конвертери та механізми, які полегшують реалізацію патерну MVVM. Завдяки цій бібліотеці в проєкті реалізовано зручну обробку подій, відображення списків, а також покращено взаємодію з користувачем.

Для перегляду нот у форматі PDF було використано компонент Syncfusion.Maui.PdfViewer [10]. Він дозволяє інтегрувати багатосторінковий PDF-переглядач без використання сторонніх застосунків. Завдяки цій бібліотеці реалізовано зручну навігацію між сторінками нот, масштабування та підтримку вертикального прокручування, що є важливим для зручності під час музичної практики. Ключовим компонентом застосунку є інтеграція з Firebase. Зокрема:

- Firebase Authentication — використовується для реєстрації, авторизації та відновлення паролю користувача;
- Firebase Firestore — для зберігання структури нотної бібліотеки, даних про папки та метаданих нот;
- Firebase Storage — для зберігання PDF-файлів нот, доступних через посилання.

Для виконання HTTP-запитів і роботи з REST API використовуються стандартні засоби .NET — HttpClient, у поєднанні з Newtonsoft**.**Json (Json.NET) для серіалізації та десеріалізації об'єктів [11]. Для збереження і швидкого доступу до ключових параметрів (ідентифікатор користувача, токен доступу) застосовується API Preferences.

Оскільки робота з нотами у проєкті реалізована на основі PDF-файлів, у поточній версії не використовуються бібліотеки для імпорту MusicXML або синхронізованого відтворення музики. Проте архітектура проєкту дозволяє потенційно розширити функціонал у майбутньому для підтримки додаткових форматів або інтеграції з MIDI-синтезаторами.

У таблиці 2.1 наведено стислий огляд основних бібліотек і фреймворків, що використовуються в дипломному проєкті, із зазначенням їхніх версій та функціонального призначення.

Таблиця 2.1

Таблиця стислого огляд основних бібліотек і фреймворків

№	Назва бібліотеки / фреймворку	Версія	Призначення
1	.NET MAUI	8.0	Основний кросплатформений фреймворк для створення мобільного застосунку
2	C#	12.0 (з .NET 8)	Мова програмування для реалізації логіки, MVVM-архітектури, роботи з API
3	XAML	-	Декларативна мова розмітки для створення UI в .NET MAUI
4	CommunityToolkit.Maui	5.0.0	Набір UI-компонентів, поведінок, MVVM-утиліт, конвертерів
5	Syncfusion.Maui.PdfViewer	29.2.5	Відображення PDF-файлів нот у застосунку
6	Firebase Authentication	REST API	Авторизація, реєстрація та відновлення паролю користувача
7	Firebase Firestore	REST API	Збереження структури бібліотеки, папок, метаданих нот
8	Firebase Storage	REST API	Хмарне зберігання PDF-файлів нот
9	Newtonsoft.Json	13.0.3	Серіалізація / десеріалізація JSON-даних при роботі з Firebase

Продовження таблиці 2.1

Таблиця стислого огляд основних бібліотек і фреймворків

10	System.Net.Http (HttpClient)	.NET built-in	Виконання запитів до Firebase REST API
11	Microsoft.Maui.Storage. Preferences	.NET built-in	Збереження локальних параметрів (token, email, localId тощо)

Зазначені інструменти було обрано на основі їхньої підтримки .NET MAUI, відповідності сучасним стандартам мобільної розробки, а також можливості гнучкого масштабування й розширення функціоналу в майбутньому. Їх комплексне використання дозволило реалізувати всі ключові вимоги до застосунку, зокрема: зручний пошук нот, структурування нотної бібліотеки за допомогою папок, перегляд нот у форматі PDF, авторизацію користувача, синхронізацію даних через Firebase та кросплатформену роботу на Android і Windows.

Така архітектура забезпечує не лише реалізацію основного функціоналу, але й високу стабільність, адаптивність інтерфейсу та можливість подальшої модифікації проєкту — наприклад, для додавання нових форматів нот або інтеграції з іншими сервісами.

Таким чином, набір фреймворків і бібліотек, використаний у проєкті, був підібраний з урахуванням вимог кросплатформеності, хмарного збереження, персоналізації та зручності користування. Їх інтеграція відбулася відповідно до принципів чистої архітектури й забезпечила високий рівень надійності, масштабованості й продуктивності застосунку.

3 ПРОЄКТУВАННЯ СИСТЕМИ

3.1 Архітектура застосунку

Архітектура програмного забезпечення визначає структуру, логіку взаємодії компонентів і загальні принципи функціонування системи. У випадку створення мобільного застосунку, який працює з нотними файлами у форматі PDF, забезпечує синхронізацію між пристроями, авторизацію користувача та організацію особистої бібліотеки, архітектурні рішення повинні поєднувати модульність, масштабованість і стабільність. Тому основою застосунку було обрано багаторівневу архітектуру, орієнтовану на шаблон MVVM **(**Model–View–ViewModel), доповнену логічною сегментацією за функціональними блоками.

Обрана архітектура відповідає стандартам побудови сучасних кросплатформенних застосунків, де критично важливими є принципи поділу відповідальності, інверсії залежностей, реактивного оновлення інтерфейсу та масштабованості. Реалізація проєкту виконана засобами .NET MAUI, що дає змогу поєднати нативну продуктивність із високим рівнем абстракції над платформо-залежними елементами [12], [13].

У загальному вигляді застосунок складається з таких основних рівнів:

- View (XAML-сторінки інтерфейсу),
- ViewModel (логіка поведінки сторінок),
- Models (класи предметної області),
- Services (взаємодія з Firebase),
- Helpers **/ Resources** (допоміжні утиліти, стилі, зображення).

Такий підхід забезпечує ізольовану розробку й тестування окремих модулів, а також дозволяє легко додавати нові функції без зміни основної структури [17].

На структурному рівні проєкт організовано за папками, кожна з яких відповідає за свою категорію логіки:

- Pages/ – усі XAML-сторінки з розміткою інтерфейсу (вхід, пошук, бібліотека, перегляд нот тощо);
- ViewModels/ – класи, що реалізують логіку поведінки UI, прив’язку даних, управління станами сторінок;
- Services/ – сервіси, що відповідають за авторизацію, обробку запитів до Firebase, завантаження нот;
- Models/ – класи предметної області: ноти, папки, користувач;
- Resources/ – стилі, зображення, піктограми, файли локалізації;
- Helpers/ – конвертери, утиліти, константи, що використовуються у декількох компонентах.

Патерн MVVM є фундаментальним у .NET MAUI, оскільки дозволяє чітко розділити інтерфейс користувача (View), логіку взаємодії (ViewModel) та структуру даних (Model). Такий підхід спрощує тестування, повторне використання компонентів і забезпечує ефективну прив’язку інтерфейсу до даних за допомогою механізму Binding.

ViewModel у проєкті реалізують оновлення стану за допомогою інтерфейсу INotifyPropertyChanged, використовують асинхронні команди з CommunityToolkit.MVVM та підтримують логіку навігації, пошуку, фільтрації та збереження нот.

Компоненти застосунку взаємодіють наступним чином:

- Pages взаємодіють із ViewModels через Binding у XAML;
- ViewModels звертаються до Services для виконання запитів до Firebase (авторизація, завантаження нот, створення папок);
- Services формують REST-запити до Firebase API та обробляють відповіді;
- Models описують структуру сутностей (нота, композитор, папка);
- Helpers використовуються для конвертації значень, форматування тексту тощо.

З урахуванням поставлених функціональних вимог — таких як підтримка темної теми, перегляд нот у PDF, створення папок, синхронізація через Firebase — було реалізовано окремі сервіси:

- FirebaseAuthService – авторизація, реєстрація, відновлення паролю;
- FirebaseFirestoreService – збереження/отримання папок, нот, метаданих із Firestore (включно з URL-посиланням на PDF у Firebase Storage).

Кожен сервіс побудований із використанням HttpClient, Json.NET та локального збереження даних через Preferences.

До ключових переваг архітектурного рішення належать:

- гнучкість – можливість додавати нові сторінки або сервіси без зміни всієї логіки;
- тестованість – ViewModels і Services можна тестувати окремо від UI;
- модульність – логіка розділена на ізольовані частини (інтерфейс, логіка, дані);
- масштабованість – структура легко розширюється для підтримки нових функцій;
- реактивність – інтерфейс оновлюється автоматично при зміні стану.

Побудова архітектури проєкту відповідала принципам **SRP (**Single Responsibility**)**, DRY **(**Don’t Repeat Yourself**)**, SOLID та MVVM, що дозволило створити стабільну основу з потенціалом для подальшого розвитку — наприклад, додавання нових форматів файлів, інтеграція з Google Drive або офлайн-збереження нот [14].

3.2 Формалізація вимог до програмного забезпечення

Формалізація вимог до програмного забезпечення – це етап, на якому розробник переходить від описових характеристик і загального бачення продукту до чітко структурованих специфікацій, які можуть бути інтерпретовані як програмістами, так і засобами моделювання та тестування. Для мобільного застосунку, що призначений для роботи з нотними файлами, взаємодії з

користувачем, синхронізації з хмарним сховищем і забезпечення інтуїтивного користувацького досвіду, моделювання вимог дозволяє досягти узгодженості між ідеєю та реалізацією.

Основою для побудови моделі вимог стали функціональні та нефункціональні вимоги. Вони охоплюють такі аспекти, як пошук нот, підтримка категорій, керування бібліотекою, перегляд нот, авторизація, синхронізація, інтерфейс користувача.

У ході моделювання вимог було створено логічну класифікацію функціональностей, які згруповано за логікою користувацьких сценаріїв. Такий підхід дозволяє представити вимоги у вигляді функціональних блоків, що формують цілісну структуру системи.

Однією з груп є блок **пошуку нот**, який передбачає можливість здійснення запитів за кількома параметрами, зокрема за назвою твору, його жанровою належністю, ім’ям композитора. Кожен із цих параметрів визначає власну логіку фільтрації, але в сукупності вони формують систему зручного пошуку, що орієнтована на музичні запити користувача. Вимога забезпечення швидкої та гнучкої навігації нотним фондом була визначальною для моделювання цього функціонального сегменту.

Наступною складовою є блок **авторизації та синхронізації**. Система забезпечує доступ до персоніфікованого середовища користувача через електронну пошту та пароль. Усі дані — створені папки, прив’язані до акаунта ноти — зберігаються у Firebase і можуть бути синхронізовані між пристроями. Також реалізована функція відновлення паролю через електронну пошту.

Ще однією групою є функції керування нотною бібліотекою. Застосунок дозволяє користувачеві створювати персональні папки, додавати до них ноти з бази даних, а також переглядати й видаляти вміст. При додаванні нот користувач може ввести назву твору та, за бажанням, ім’я композитора. Кожна папка пов’язана з акаунтом користувача та синхронізується з хмарою. Вміст нот зберігається у форматі PDF, і для їх перегляду реалізований вбудований PDF-переглядач.

Щодо функцій імпорту та експорту, у поточній реалізації застосунку передбачено можливість перегляду нот у форматі PDF та завантаження файлу на пристрій. Експорт здійснюється через стандартну функцію "Поділитися", яка використовує інтерфейси спільного доступу системи. Імпорт нот із MusicXML та підтримка повноцінного редагування не реалізовані, але структура системи дозволяє розширити функціонал у майбутньому.

Вимоги до інтерфейсу охоплюють сторінки: налаштувань, пошуку, бібліотеки, перегляду нот, FAQ, підтримки та акаунту. Інтерфейс підтримує зміну теми (світла/темна), адаптивний дизайн, базову локалізацію, а також навігацію за допомогою нижньої панелі вкладок. Кожен елемент інтерфейсу спроєктовано відповідно до принципів доступності та простоти, що важливо для широкого кола користувачів.

У табл. 3.1 наведено узагальнення ключових функціональних блоків, їх логічне призначення та пов’язані з ними модулі.

Таблиця 3.1

Таблиця структури функціональних вимог і відповідність архітектурним модулям

№	Функціональний блок	Призначення	Відповідні модулі / класи
1	Пошук нот	Пошук нот за назвою, композитором, жанром	SearchPage.xaml, SearchViewModel.cs, FirebaseFirestoreService.cs
2	Перегляд нот	Відображення PDF-документів нот	NoteViewerPage.xaml, Syncfusion.Maui.PdfViewer
3	Авторизація	Реєстрація, вхід, вихід, відновлення паролю	LoginPage.xaml, FirebaseAuthService.cs
4	Синхронізація	Збереження даних користувача у Firebase Firestore	FirebaseFirestoreService.cs, Preferences

Продовження таблиці 3.1

Таблиця структури функціональних вимог і відповідність архітектурним модулям

5	Бібліотека користувача	Створення папок, додавання нот, перегляд вмісту	MyLibraryPage.xaml, NotesPage.xaml, FolderNotesPage.xaml, NotesViewModel.cs
6	FAQ / Підтримка	Надання довідки та можливість звернення до розробника	FaqPage.xaml, SupportPage.xaml, FaqViewModel.cs
7	Налаштування	Зміна теми, вихід з акаунту, видалення акаунту	SettingsPage.xaml, AccountPage.xaml, SettingsViewModel.cs
8	Поділитися/ Завантажити	Завантаження PDF-файлу абонадсилання його через сторонні застосунки	NotesViewModel.cs (методи DownloadAsync, ShareAsync)
9	Вибір папки	Вибір існуючої папки користувача перед додаванням ноти	NotesViewModel.cs, FirebaseFirestoreService.cs

На рисунку 3.1 подано спрощену діаграму варіантів використання, яка відображає основні сценарії взаємодії користувача з системою на верхньому рівні.

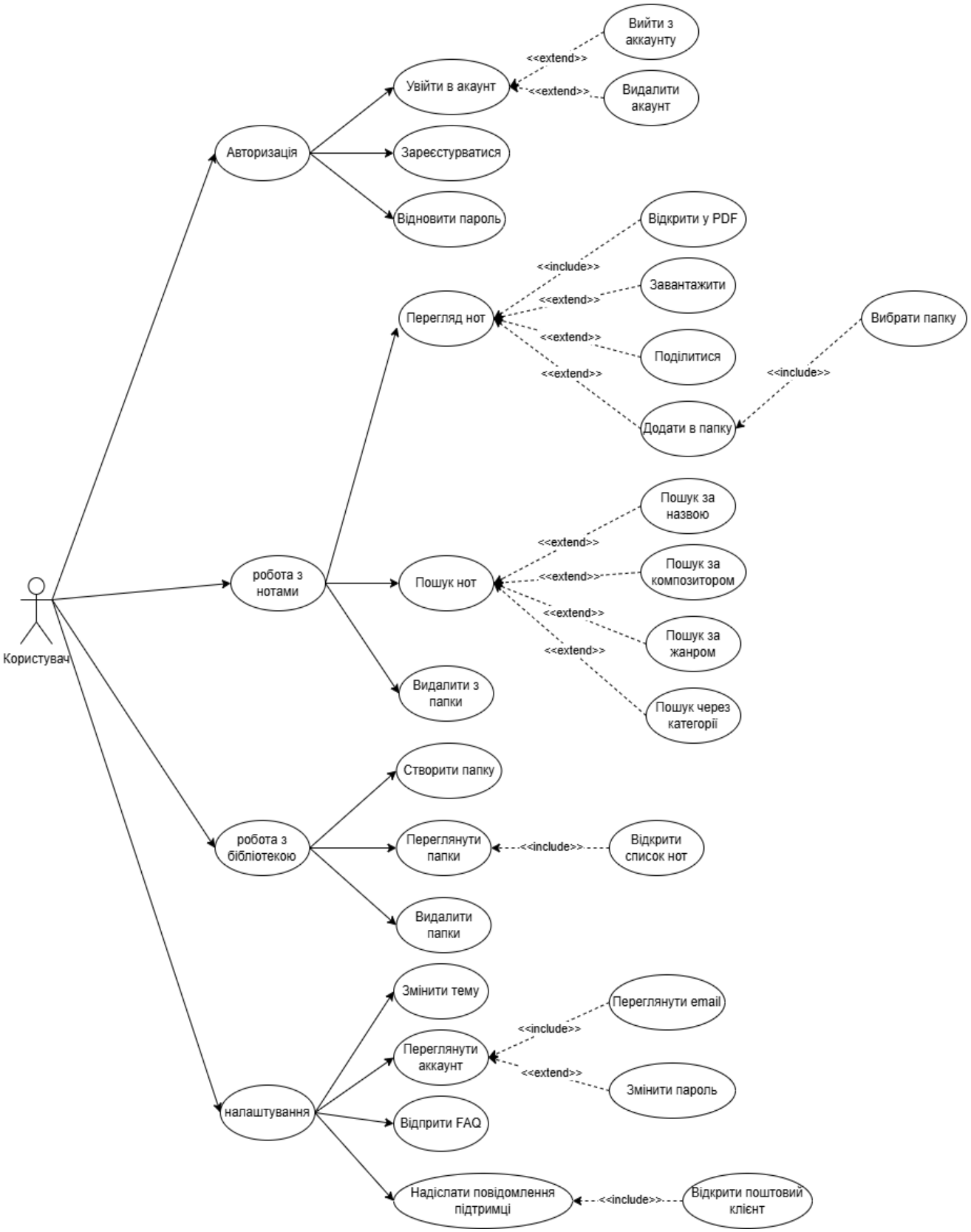


Рис.3.1 Діаграма варіантів використання

Таким чином, модель вимог до застосунку охоплює як функціональні, так і нефункціональні аспекти системи. Вона сформована з урахуванням практичних потреб користувача, сучасних вимог до мобільного інтерфейсу та гнучкої архітектури. Усі вимоги структуруються навколо конкретних сценаріїв, що реалізуються через відповідні модулі та компоненти в архітектурі системи. Це забезпечує узгодженість між очікуваннями користувача, бізнес-логікою і технічною реалізацією проєкту [15].

3.3 Проєктування бази даних

Розробка програмного забезпечення, що працює з нотними матеріалами, синхронізується між пристроями та зберігає індивідуальні бібліотеки користувачів, передбачає створення гнучкої системи збереження даних. У рамках даного застосунку головним завданням стало забезпечення швидкого доступу до нот, збереження структури користувацьких папок, підтримка авторизації та синхронізації між пристроями через хмарну платформу Firebase.

Для реалізації зберігання даних було використано Firebase Firestore як основну базу даних, а також Firebase Storage для розміщення PDF-файлів нот [16]. Такий вибір дозволив забезпечити:

- просту синхронізацію між пристроями,
- зберігання персональних даних користувача у структурі users/{userId}/folders/{folderName}/notes,
- масштабованість без потреби у власній серверній інфраструктурі.

Основні сутності системи:

У Firestore структура організована за такими ключовими компонентами:

- Колекція materials – глобальний репозиторій нот. Кожен документ містить поля:
  - title: назва твору,
  - composer: ім’я композитора,

- genre: жанрова належність,
- category: категорія (наприклад, "solfege", "scales"),
- pdfUrl: посилання на PDF-файл у Firebase Storage.

- Колекція users/{userId}/folders – індивідуальні папки користувача. Кожен документ у folders має поле:
  - name: назва папки.

У межах кожної папки:

- підколекція notes містить нотні документи, додані вручну користувачем. Кожен документ включає:
  - title, composer (введені користувачем),
  - pdfUrl (отримане з глобальної бази).

Ця структура дозволяє гнучко працювати з персональною бібліотекою, ізоляцією даних і збереженням потрібних нот без дублювання у загальній базі.

Синхронізація та кешування

Після авторизації користувача в додатку:

- дані про його папки (folders) і вміст (notes) отримуються з Firestore;
- доступ до файлів здійснюється через URL, що відкривається у вбудованому PDF-переглядачі;
- за потреби нота може бути завантажена на пристрій — файл зберігається локально через FileSystem.AppDataDirectory.

Окрема локальна база (наприклад, SQLite) **не використовується**, оскільки дані централізовано зберігаються у Firebase, а кешування відбувається на рівні файлів.

Ізоляція даних

Кожен користувач ідентифікується у Firestore за унікальним localId, отриманим під час авторизації. Дані кожного користувача зберігаються окремо в шляху users/{localId}/, що забезпечує:

- ізоляцію бібліотеки,
- можливість редагування лише власних нот,

- розширення функціоналу у майбутньому (наприклад, спільне використання нот).

Структура відповідає вимогам до:

- Персоналізації — кожен користувач має власні папки та ноти;
- Масштабованості — Firebase дозволяє обслуговувати велику кількість користувачів;
- Синхронізації — дані доступні з будь-якого пристрою після авторизації;
- Простоти обробки — використання REST-запитів через HttpClient і JSON-десеріалізацію через Newtonsoft.Json.

В таблиці 3.2 розбажено основні сутності в структурі зберігання даних.

Таблиця 3.2

Основні сутності в структурі зберігання даних NoteLibraryApp

№	Сутність	Призначення	Ключові поля / атрибути	Розташування в Firestore
1	materials	Глобальний репозиторій нот для пошуку	title, composer, genre, category, pdfUrl	materials
2	users	Коренева колекція з даними користувача	— (службова, містить вкладені колекції)	users/{userId}/
3	folders	Папки користувача для структурування бібліотеки	name	users/{userId}/folders
4	notes (у папці)	Ноти, додані користувачем у власну папку	title, composer (опційно), pdfUrl	users/{userId}/folders/{folderName}/notes
5	Firebase Storage (файли)	Фізичне зберігання PDF-файлів нот	— (файл .pdf, доступний через URL)	Зовнішнє: gs://<project-id>.appspot.com/
6	Preferences (локально на пристрої)	Локальне зберігання параметрів авторизації та теми	local_id, id_token, user_email, is_dark_theme	Microsoft.Maui.Storage.Preferences

На рисунку 3.2 зображено логічну ER-діаграму, яка відображає основні сутності та їхні зв’язки у структурі бази даних.

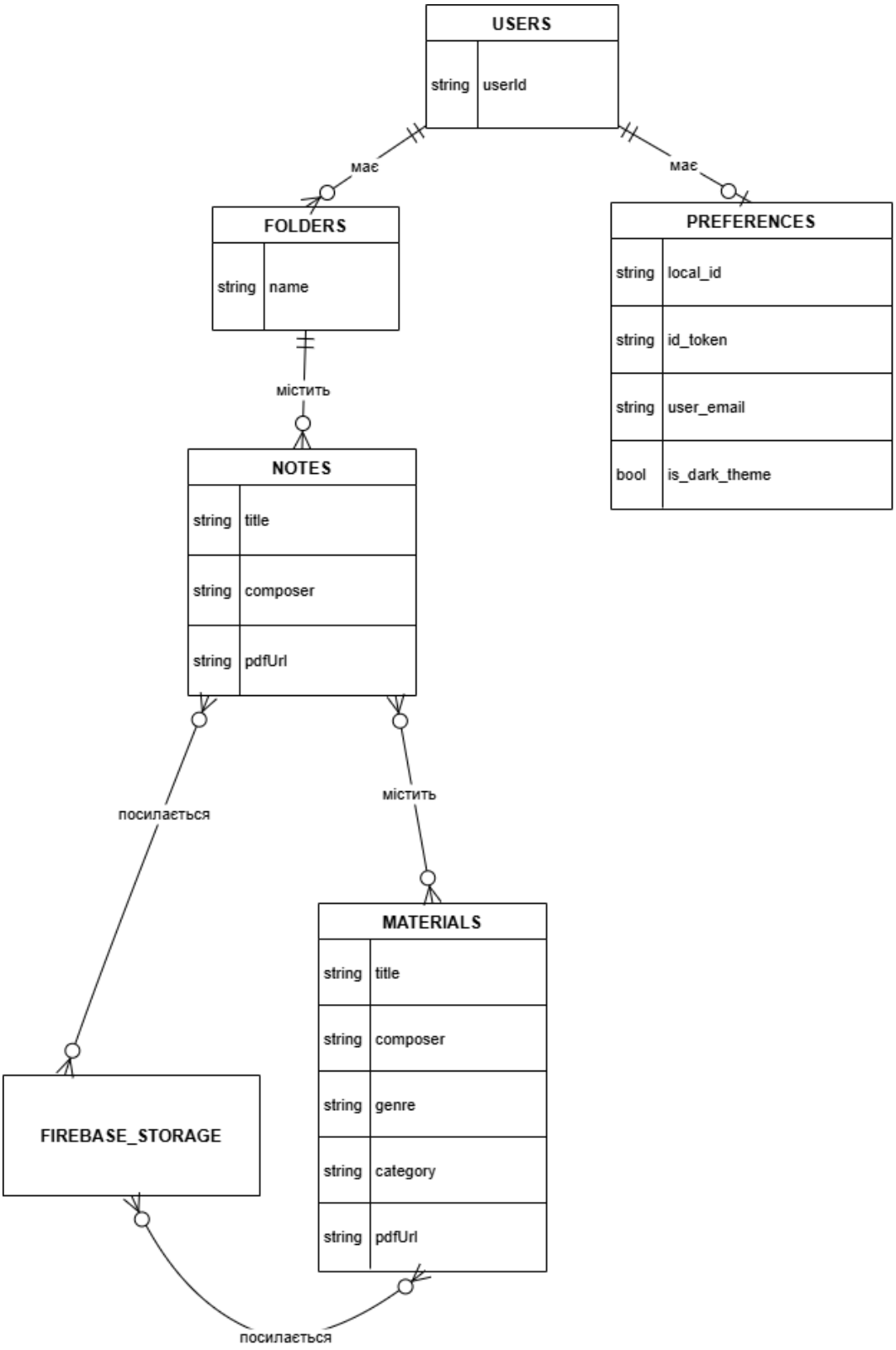


Рис. 3.2 ER-діаграма логічної структури зберігання даних

У результаті проєктування та реалізації структури збереження даних у мобільному застосунку NoteLibraryApp було використано хмарну платформу **Firebase**, зокрема Firestore для зберігання метаданих нот і папок, та Firebase Storage для збереження PDF-файлів. Такий підхід забезпечив централізовану синхронізацію між пристроями, масштабованість, а також простоту реалізації без потреби у власній серверній інфраструктурі. Завдяки структурі users/{userId}/folders/{folderName}/notes, кожен користувач має ізольований доступ до власної бібліотеки нот, що дозволяє ефективно реалізувати функції додавання, збереження та видалення нот у папках. Збереження файлів на пристрої доповнює систему офлайн-доступом.

Обрана архітектура зберігання є гнучкою та легко масштабованою, а використання REST-запитів та стандартних інструментів .NET дозволяє швидко адаптувати її до майбутніх змін. Таким чином, структура повністю відповідає функціональним вимогам застосунку і створює надійну основу для подальшого розвитку проєкту.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Діаграма класів

Діаграма класів є центральним елементом об’єктно-орієнтованого проєктування, оскільки дозволяє відобразити основні програмні сутності, зв’язки між ними, типи взаємодії та ієрархію залежностей. У межах архітектури, реалізованої з використанням .NET MAUI у поєднанні з MVVM-підходом, було спроєктовано чітку багаторівневу систему класів, яка розділяє логіку на окремі шари: моделі, представлення **(**ViewModel**)**, сервіси та допоміжні компоненти.

Кожен клас реалізований відповідно до принципу єдиної відповідальності **(**SRP**)**, що забезпечує модульність, гнучкість розширення та зручність супроводу. Уся структура застосунку умовно поділена на кілька логічних блоків.

Моделі (Models)

У шарі моделей (Models/) розміщено класи предметної області:

- NoteModel – описує ноту (title, composer, genre, category, pdfUrl);
- FolderModel – зберігає назву папки;
- FaqModel – модель питання-відповіді для FAQ-сторінки.

Кожна модель є простою DTO-структурою з властивостями, які відповідають полям документів у Firebase Firestore.

Представлення (ViewModels)

Кожна сторінка має свій відповідний ViewModel-клас:

- LoginViewModel – логіка входу, реєстрації, відновлення паролю;
- SearchViewModel – реалізує пошук нот за текстовим запитом або категоріями (жанр, композитор);
- NotesViewModel – відповідає за логіку перегляду, завантаження, додавання до папки, поширення;
- MyLibraryViewModel – логіка створення та відкриття папок;
- FaqViewModel – завантаження списку FAQ на основі мови;

- SettingsViewModel – зміна теми, вихід із акаунту, навігація до сторінок;
- AccountViewModel – керування акаунтом: email, видалення, вихід.

Кожен ViewModel реалізує інтерфейс INotifyPropertyChanged та використовує команди Command або AsyncCommand для взаємодії з інтерфейсом у стилі MVVM.

Сервіси (Services)

Усі взаємодії із зовнішніми API, Firebase та локальним сховищем реалізовано через сервіси:

- FirebaseAuthService – авторизація, реєстрація, відновлення паролю через REST API;
- FirebaseFirestoreService – отримання нот, жанрів, композиторів, збереження в папки;
- Preferences – локальне збереження токенів, email, localId;
- HttpClient – завантаження PDF-файлів за прямим URL;
- Microsoft.Maui.Storage.FileSystem – збереження нот локально;
- Microsoft.Maui.ApplicationModel.DataTransfer – реалізація функції "Поділитися".

Зовнішній доступ до Firebase Storage реалізується лише через pdfUrl — окремого FirebaseStorageService не створювалося.

Допоміжні компоненти (Helpers / Converters)

- InverseBoolConverter – логічний конвертер для відображення/приховування компонентів у XAML;
- LocalizationHelper – перемикач між мовами для FAQ (реалізовано у ViewModel або ресурсах);
- В окремих ViewModels використовується форматування назв, обробка рядків, повідомлення та ін.

Узагальнення

Завдяки патерну MVVM і розділенню на логічні блоки, кожен клас у застосунку виконує чітко визначену роль. Це дозволяє:

- легко масштабувати проект (наприклад, додати нову сторінку або сервіс),
- зручно реалізовувати функціональні оновлення,
- тестувати окремі компоненти без зміни решти структури.

На рисунку 4.1 буде подано повну UML-діаграму класів, що демонструє ключові сутності, зв’язки та методи.

Ця діаграма дозволяє наочно оцінити архітектурну структуру мобільного застосунку NoteLibraryApp, виявити логічні залежності між класами, а також переконатися у коректному розмежуванні відповідальностей між окремими компонентами системи. Завдяки графічному представленню зв’язків можна швидко зрозуміти взаємодію між ViewModel, сервісами та моделями даних, що особливо корисно в умовах командної розробки або при передачі проєкту іншим фахівцям.

Діаграма класів також може бути використана як основа для побудови автоматизованих тестів, планування майбутнього рефакторингу та виявлення точок розширення функціоналу. Її структура дозволяє оцінити масштабованість застосунку, виявити вузли повторного використання логіки та зони потенційної оптимізації.

Використання архітектурного шаблону MVVM (Model–View–ViewModel) у поєднанні з чіткою сервісною організацією забезпечує ізольовану реалізацію кожного функціонального блоку. Класи ViewModel відповідають лише за керування інтерфейсом, моделі — за структуру даних, а сервіси — за взаємодію з Firebase. Це не лише спрощує супровід коду, а й забезпечує його повторне використання, гнучкість змін і високу якість програмної реалізації відповідно до принципів об’єктно-орієнтованого програмування [17].

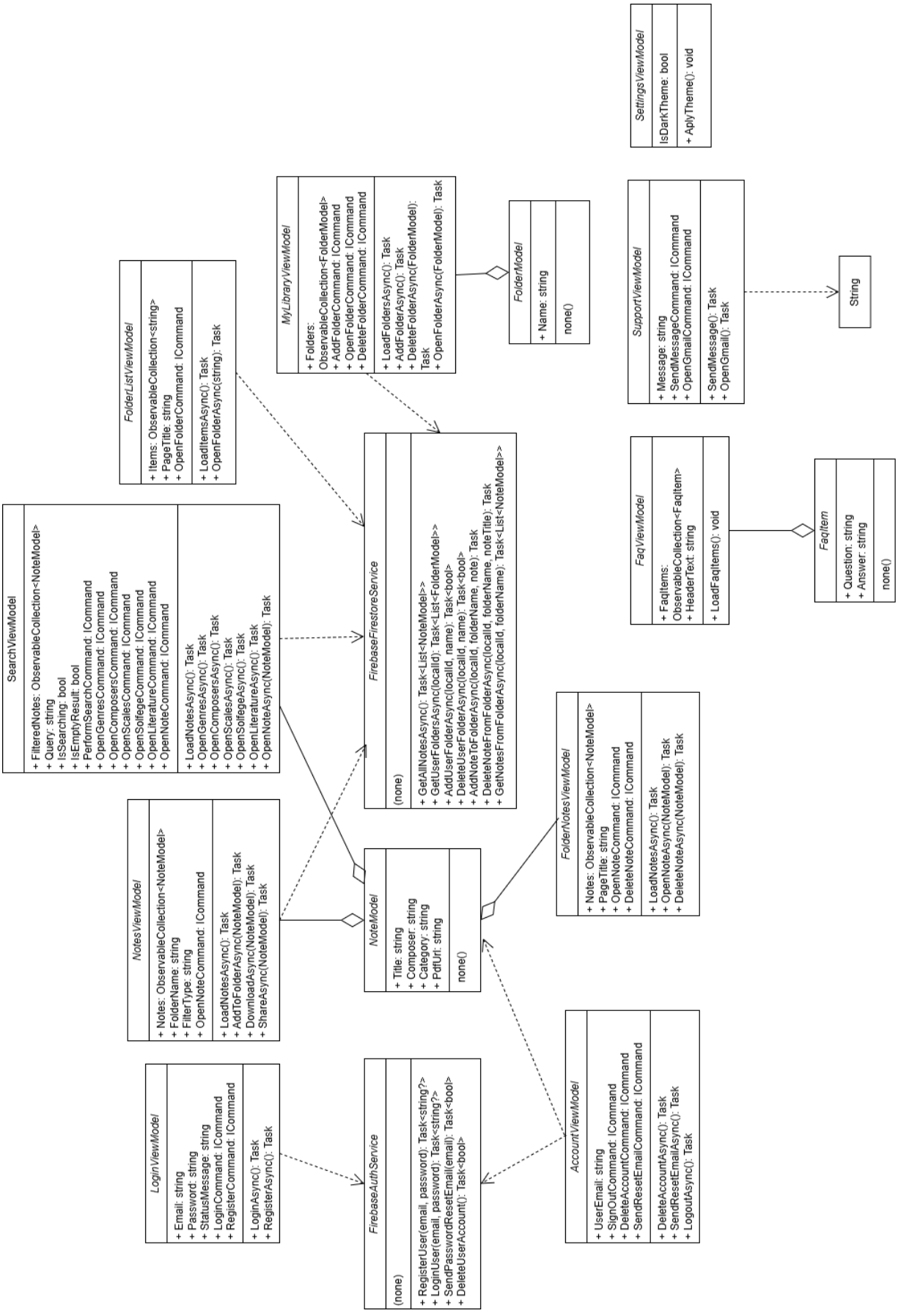


Рис 4.1 UML-діаграма класів архітектури застосунку

4.2 Реалізація функцій

Програмна реалізація функціональних можливостей мобільного застосунку NoteLibraryApp виконана у відповідності до патерну MVVM (Model–View–ViewModel) із чітким розділенням логіки, інтерфейсу користувача та обробки даних. Основними мовами розробки є C# та XAML у середовищі .NET MAUI. Усі функціональні блоки реалізовано у вигляді окремих модулів – ViewModel та сервісів – що забезпечує гнучкість, повторне використання і зручне тестування.

Нижче описано ключові реалізовані функції:

1. Авторизація користувача

Функціонал авторизації реалізований у класі LoginViewModel за допомогою сервісу FirebaseAuthService. Підтримується реєстрація та вхід користувача за email і паролем. Після успішної авторизації зберігаються токени та ідентифікатор користувача для подальшої роботи із базою Firebase.

```
private async Task LoginAsync()
{
    StatusMessage = "Вхід...";
    var token = await _authService.LoginUser(Email, Password);
    if (token != null)
    {
        Preferences.Set("user_email", Email);

        StatusMessage = "Успішний вхід!";
        Application.Current.MainPage = new AppShell();
    }
    else
    {
        StatusMessage = "Помилка входу. Перевірте дані.";
    }
}
```

Рис. 4.2 Функціонал авторизації

2.Пошук нот

У класі SearchViewModel реалізовано логіку пошуку нот за назвою твору або іменем композитора. При введенні запиту користувача система фільтрує список нот, отриманих із Firebase, і виводить відповідні результати.

```
private void FilterNotes()
{
    FilteredNotes.Clear();

    var queryLower = Query?.ToLower() ?? "";
    IsSearching = !string.IsNullOrWhiteSpace(queryLower);

    if (!IsSearching)
    {
        IsEmptyResult = false;
        return;
    }

    var results = _allNotes
        .Where(note =>
            note.Title.ToLower().Contains(queryLower) ||
            note.Composer.ToLower().Contains(queryLower))
        .ToList();

    IsEmptyResult = results.Count == 0;

    foreach (var note in results)
        FilteredNotes.Add(note);
}
```

Рис. 4.3 Функціонал пошуку нот за назвою твору

3. Додавання нот до папки

Ця функція реалізована у класі NotesViewModel. Користувач вводить назву твору (обов'язково) та композитора (необов'язково), після чого обирає папку зі списку. Дані зберігаються у Firestore у відповідну вкладену колекцію notes.

```
public async Task AddToFolderAsync(NoteModel note)
{
    var localId = Preferences.Get("user_localId", "");
    if (string.IsNullOrWhiteSpace(localId))
    {
        await Application.Current.MainPage.DisplayAlert("Помилка", "Не знайдено користувача", "OK");
        return;
    }

    string title = await Application.Current.MainPage.DisplayPromptAsync(
        "Назва твору", "Введіть назву твору:");

    if (string.IsNullOrWhiteSpace(title))
    {
        await Application.Current.MainPage.DisplayAlert("Помилка", "Назву твору не введено", "OK");
        return;
    }

    string composer = await Application.Current.MainPage.DisplayPromptAsync(
        "Композитор", "Введіть ім'я композитора (за бажанням):");

    var service = new FirebaseFirestoreService();
    var folders = await service.GetUserFoldersAsync(localId);

    if (folders == null || folders.Count == 0)
    {
        await Application.Current.MainPage.DisplayAlert("Папки не знайдено", "У вас немає жодної створеної папки", "OK");
        return;
    }
```

Рис. 4.4 Функціонал додавання нот до папки (частина 1)

```
    string[] folderNames = folders.Select(f => f.Name).ToArray();
    string selectedFolder = await Application.Current.MainPage.DisplayActionSheet(
        "Оберіть папку", "Скасувати", null, folderNames);

    if (string.IsNullOrWhiteSpace(selectedFolder) || selectedFolder == "Скасувати")
        return;

    var updatedNote = new NoteModel
    {
        Title = title,
        Composer = composer,
        PdfUrl = note.PdfUrl
    };

    try
    {
        await service.AddNoteToFolderAsync(localId, selectedFolder, updatedNote);
        await Application.Current.MainPage.DisplayAlert("Успіх", $"Ноту додано до папки «{selectedFolder}»", "OK");
    }
    catch (Exception ex)
    {
        await Application.Current.MainPage.DisplayAlert("Помилка", ex.Message, "OK");
    }
}
```

Рис. 4.4 Функціонал додавання нот до папки (частина 2)

4. Перегляд нот

Для перегляду нот у форматі PDF використовується бібліотека Syncfusion.Maui.PdfViewer. Реалізація розташована у NoteViewerPage.

```
private async Task LoadPdfAsync(string url)
{
    try
    {
        using var httpClient = new HttpClient();
        var pdfBytes = await httpClient.GetByteArrayAsync(url);

        string fileName = $"note_{Guid.NewGuid()}.pdf";
        string localPath = Path.Combine(FileSystem.CacheDirectory, fileName);
        File.WriteAllBytes(localPath, pdfBytes);

        var stream = File.OpenRead(localPath);
        PdfViewer.LoadDocument(stream);
    }
    catch (Exception ex)
    {
        await DisplayAlert("Помилка", $"Не вдалося завантажити PDF: {ex.Message}", "OK");
    }
}
```

Рис. 4.5 Функціонал перегляду нот у PDF

5. Створення, перегляд і видалення папок

У MyLibraryViewModel реалізовано повний цикл керування користувацькими папками:

- створення через AddFolderAsync() Рисунок 4.6,
- перегляд LoadFoldersAsync() Рисунок 4.7,
- видалення DeleteFolderAsync() Рисунок 4.8.

```
private async Task AddFolder()
{
    string name = await Application.Current.MainPage.DisplayPromptAsync("Нова папка", "Введіть назву нової папки:");
    if (!string.IsNullOrWhiteSpace(name))
    {
        var folder = new FolderModel { Name = name };
        await _firestoreService.AddUserFolderAsync(_userId, folder.Name);
        Folders.Add(folder);
    }
}
```

Рис. 4.6 Функціонал створення папок

```
private async Task LoadFoldersAsync()
{
    Folders.Clear();
    var folders = await _firestoreService.GetUserFoldersAsync(_userId);
    foreach (var folder in folders)
        Folders.Add(folder);
}
```

Рис. 4.6 Функціонал перегляду папок

```
public async Task DeleteFolder(FolderModel folder)
{
    if (Folders.Contains(folder))
    {
        Folders.Remove(folder);
        await _firestoreService.DeleteUserFolderAsync(_userId, folder.Name);
    }
}
```

Рис. 4.7 Функціонал видалення папок

6. Видалення нот із папки

Функція реалізована у FolderNotesViewModel. Дія підтверджується діалогом перед виконанням:

```
private async Task DeleteNoteAsync(NoteModel note)
{
    bool confirm = await Application.Current.MainPage.DisplayAlert("Видалити", $"Видалити «{note.Title}» з папки?", "Так", "Ні");
    if (!confirm) return;

    var service = new FirebaseFirestoreService();
    await service.DeleteNoteFromFolderAsync(localId, folderName, note.Title);
    Notes.Remove(note);
}
```

Рис. 4.8 Функціонал видалення нот із папки

7. Видалення акаунту

Функціонал видалення акаунту реалізовано у сервісі FirebaseAuthService. Після підтвердження дії користувачем, відправляється POST-запит до API Firebase, який видаляє обліковий запис на сервері. Метод використовує токен користувача,

отриманий під час входу.

```
public async Task<bool> DeleteUserAccount()
{
    string idToken = Preferences.Get("id_token", "");
    if (string.IsNullOrEmpty(idToken)) return false;

    var payload = new { idToken };
    var content = new StringContent(JsonConvert.SerializeObject(payload), Encoding.UTF8, "application/json");

    var response = await new HttpClient().PostAsync(
        $"https://identitytoolkit.googleapis.com/v1/accounts:delete?key={ApiKey}", content);

    return response.IsSuccessStatusCode;
}
```

Рис. 4.9 Функціонал видалення акаунту

Кнопка видалення акаунту розміщена у розділі "Налаштування акаунту", після чого користувача повертає на екран авторизації.

8. Відновлення паролю

Відновлення паролю здійснюється шляхом надсилання спеціального листа на електронну пошту користувача. Для цього у сервісі FirebaseAuthService реалізовано метод SendPasswordResetEmail:

```
public async Task<bool> SendPasswordResetEmail(string email)
{
    var client = new HttpClient();
    var requestUri = $"https://identitytoolkit.googleapis.com/v1/accounts:sendOobCode?key={ApiKey}";

    var payload = new
    {
        requestType = "PASSWORD_RESET",
        email = email
    };

    var content = new StringContent(JsonConvert.SerializeObject(payload), Encoding.UTF8, "application/json");

    var response = await client.PostAsync(requestUri, content);
    return response.IsSuccessStatusCode;
}
```

Рис. 4.10 Функціонал відновлення паролю

Після введення email на сторінці входу й натискання кнопки "Забув пароль?", користувач отримує лист для відновлення паролю на вказану адресу.

10. Зворотній зв’язок (відкриття email клієнта)

Функціонал реалізовано у SupportViewModel за допомогою Launcher.Default.OpenAsync(). При натисканні кнопки "Зв’язатися", застосунок відкриває поштову програму з автоматично заповненими полями email, темою та тілом листа.

```
OpenGmailCommand = new Command(async () =>
{
    string supportEmail = "notelibraryhelp@gmail.com";
    var uri = new Uri($"mailto:{supportEmail}?subject=Підтримка&body=");
    await Launcher.Default.OpenAsync(uri);
});
```

Рис 4.11 Функціонал зворотнього зв’язку

Цей підхід дозволяє користувачеві швидко перейти до створення листа зворотного зв’язку без необхідності копіювати адресу вручну.

4.3 Тестування

Після завершення реалізації функціональних модулів мобільного застосунку NoteLibraryApp було проведено комплексне тестування з метою перевірки працездатності, стабільності та відповідності функціональним вимогам. Метою тестування було виявлення помилок, перевірка взаємодії між компонентами та підтвердження правильного виконання ключових сценаріїв користувача.

Тестування охоплювало ручну перевірку повної поведінки системи в інтерактивному режимі. Було змодельовано реальні ситуації: реєстрація, авторизація, перегляд нотної бібліотеки, пошук нот, завантаження та відкриття файлів, додавання нот до папок, видалення, а також функціонал зворотного зв’язку та налаштування теми. Особлива увага приділялася перевірці стабільності при відсутності інтернету, порожніх даних або некоректних введеннях.

Тестування авторизації

Було перевірено екран авторизації, включно з валідацією правильності email

і паролю. Система коректно обробляє помилки при неправильному вводі, а також успішно виконує реєстрацію та вхід користувача.

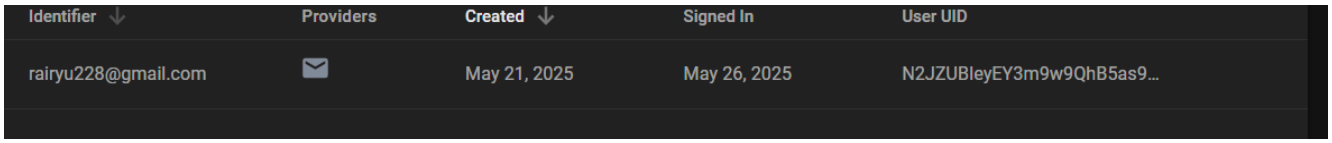


Рис. 4.12 Екранні форми Firebase Authentication до авторизації

Рис. 4.13 Екранні форми Firebase Authentication після авторизації

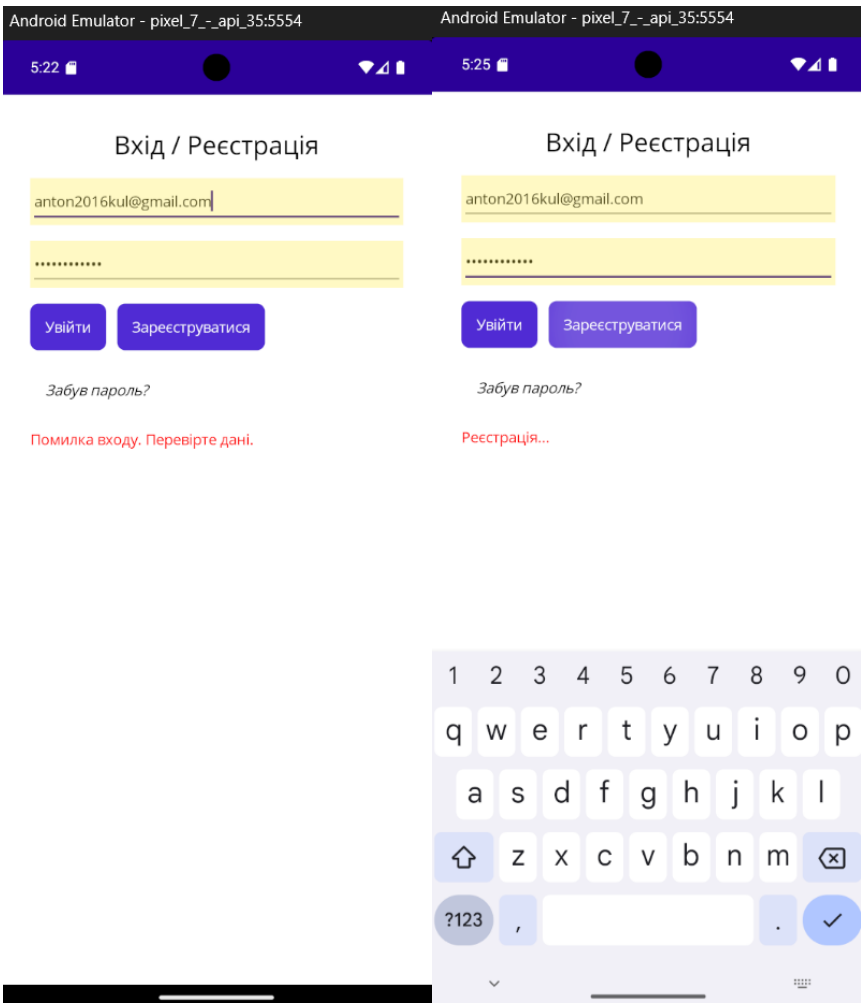


Рис. 4.14 Екранні форми сторінки авторизації

Тестування навігації між сторінками

Після входу користувач потрапляє на головну сторінку з нижньою панеллю навігації: Бібліотека, Пошук, Налаштування. Було перевірено перемикання між сторінками, збереження стану інтерфейсу та доступ до всіх функцій.

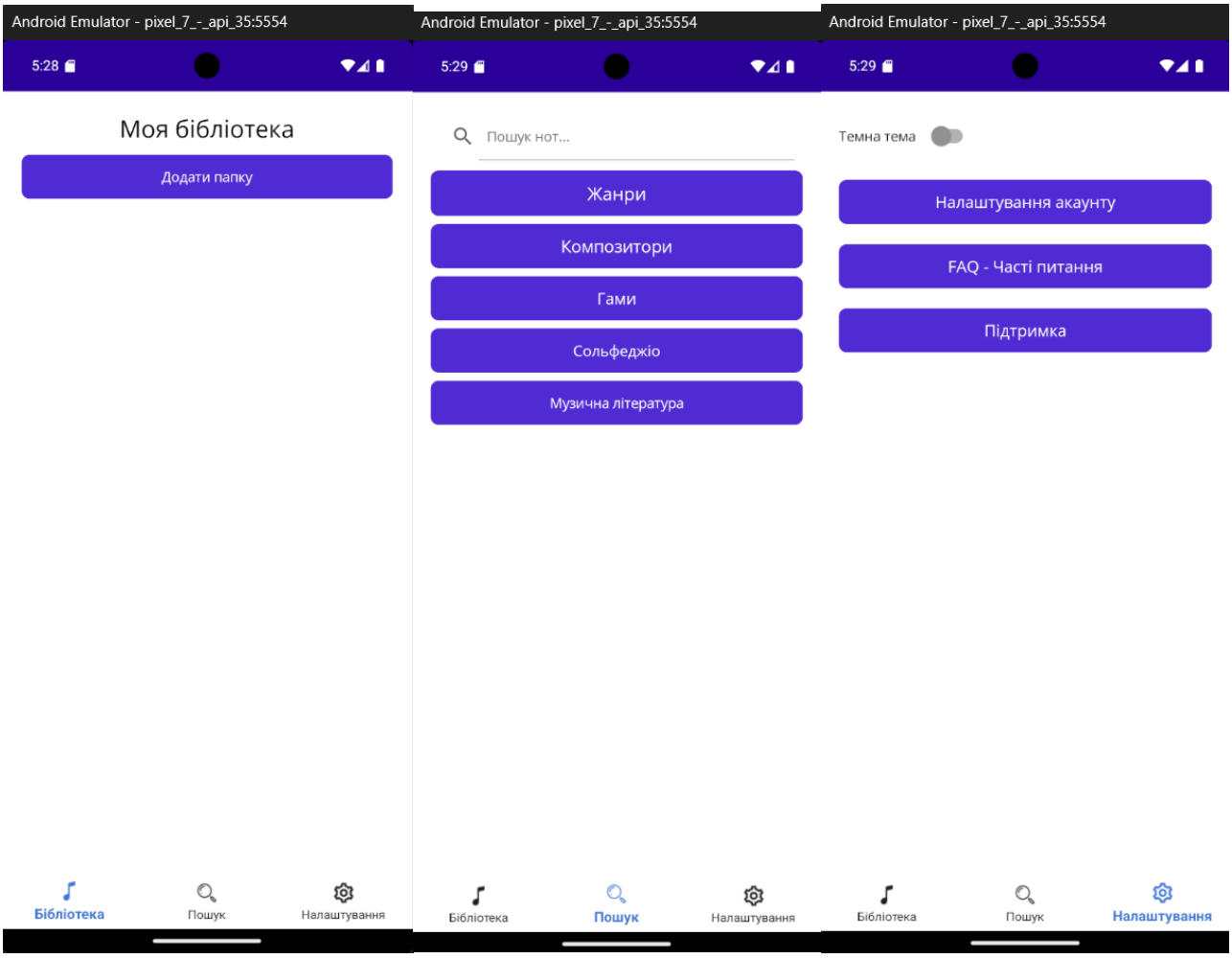


Рис. 4.15 Екранні форми сторінок основних навігаційних сторінок

Тестування бібліотеки та папок

У вкладці Бібліотека перевірено: створення нових папок, перегляд нот у вибраній папці, відкриття та видалення нот подвійним натисканням.

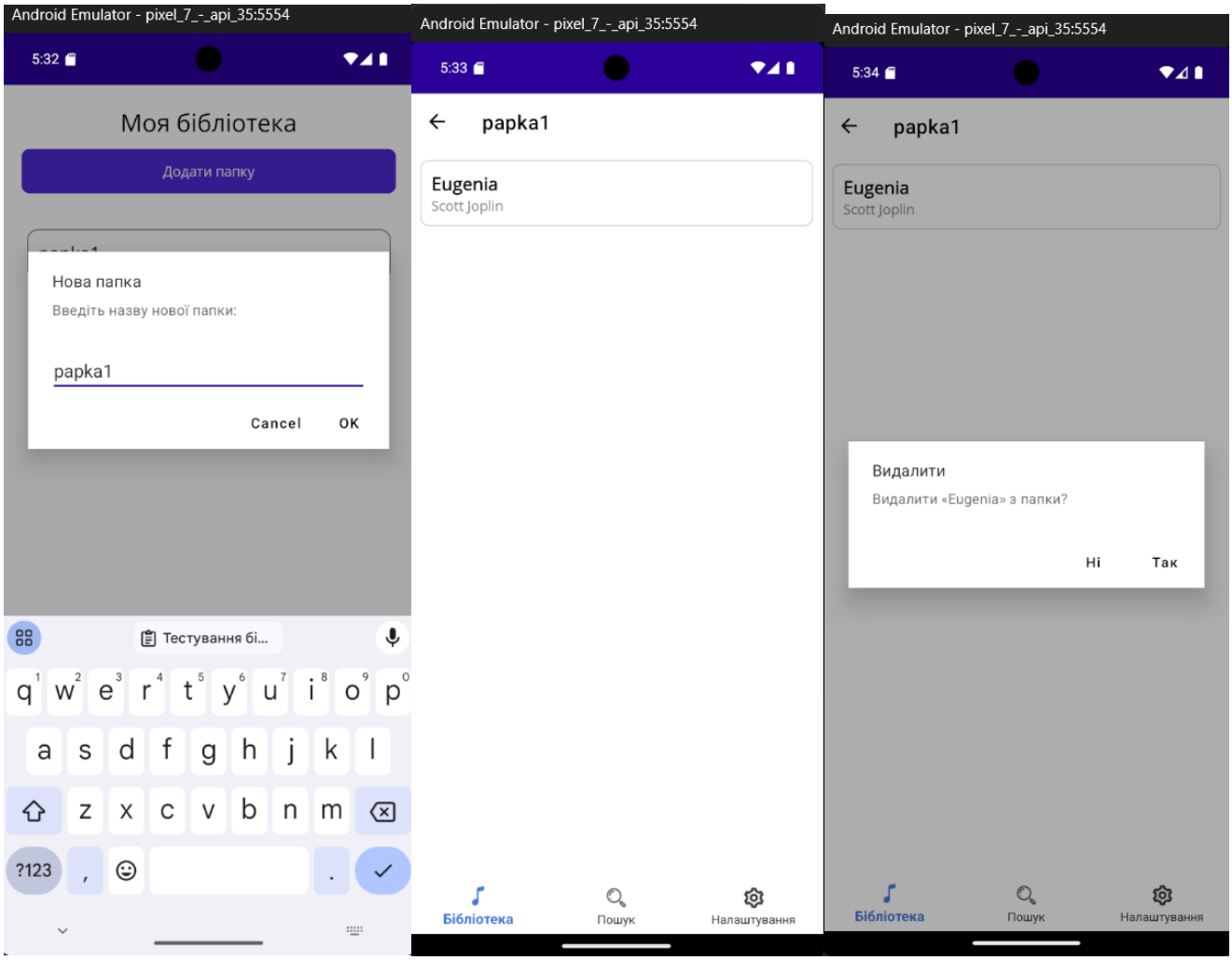


Рис 4.16 Екранні форми сторінки бібліотеки

Тестування функції пошуку

Модуль пошуку перевірено на обробку запитів із неповними або неправильними словами, а також на пошук повних назв нот. Результати фільтрації були релевантними.

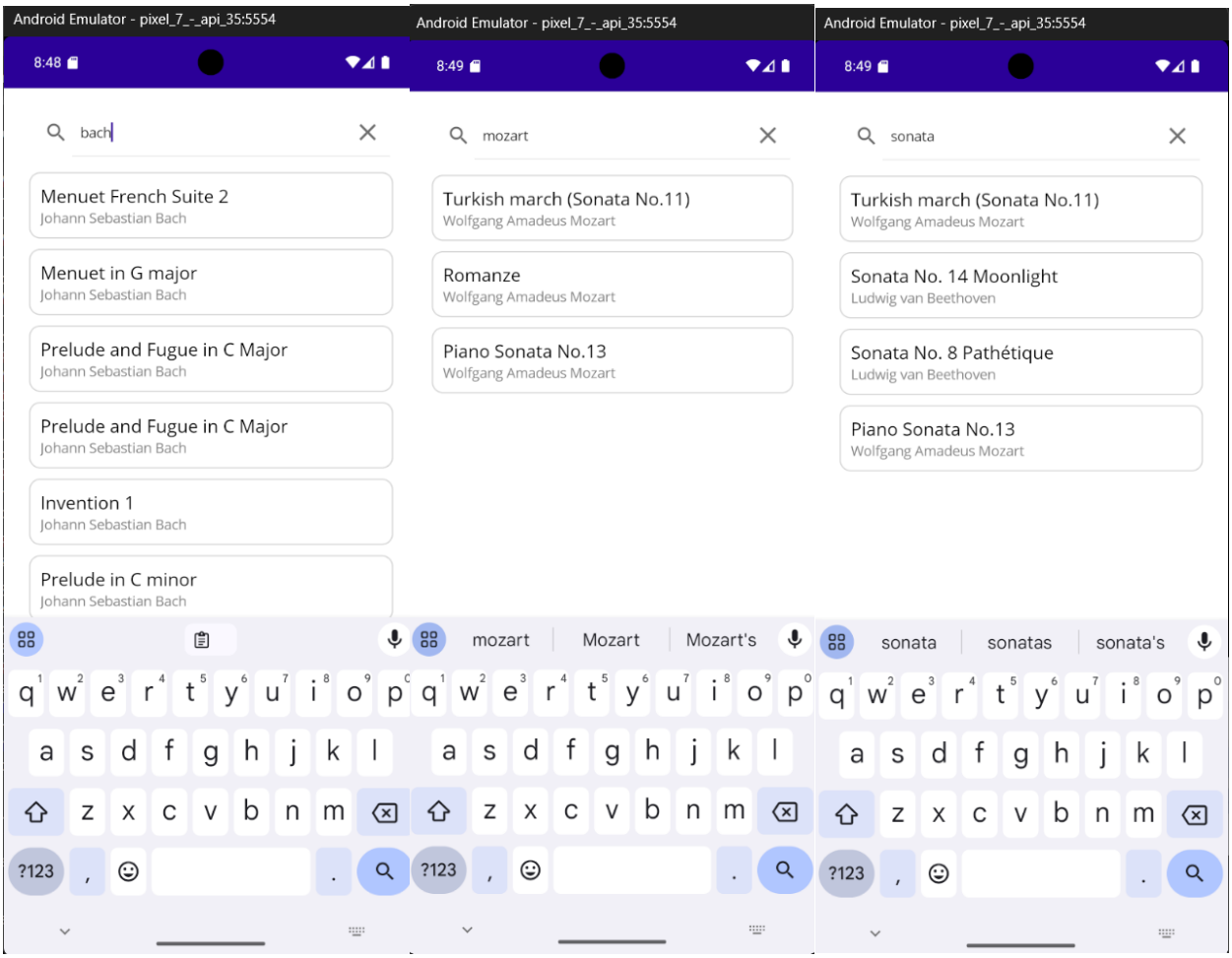


Рис. 4.17 Екранна форма реалізації пошуку нот за запитом з результатами

Пошук нот за допомогою категорій

Крім основного пошуку за назвою або композитором, застосунок підтримує пошук за допомогою кнопок-папок: Жанри, Композитори, Гами, Сольфеджіо, Музична література. Для кнопок Жанри та Композитори реалізовано перехід на сторінку з унікальними значеннями з бази, після чого користувач може обрати конкретну категорію для перегляду нот.

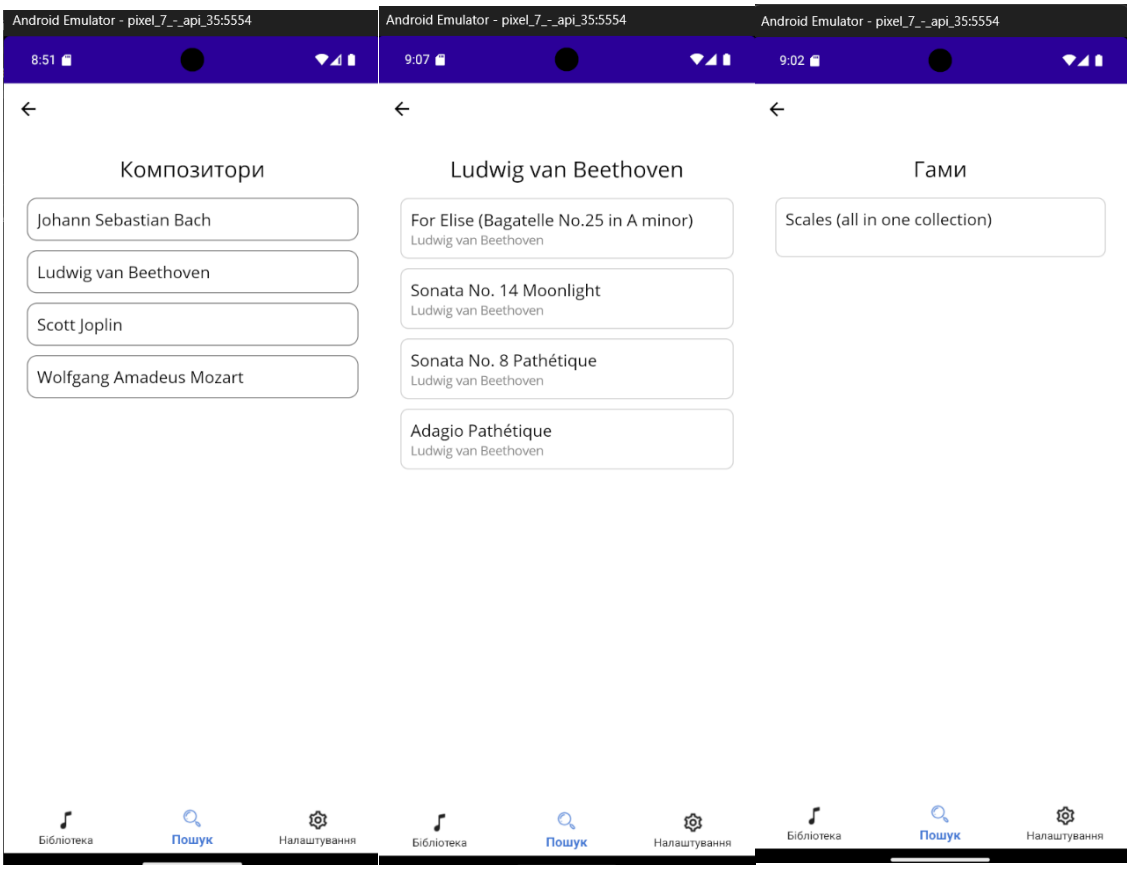


Рис.4.18 Сторінка з унікальними жанрами

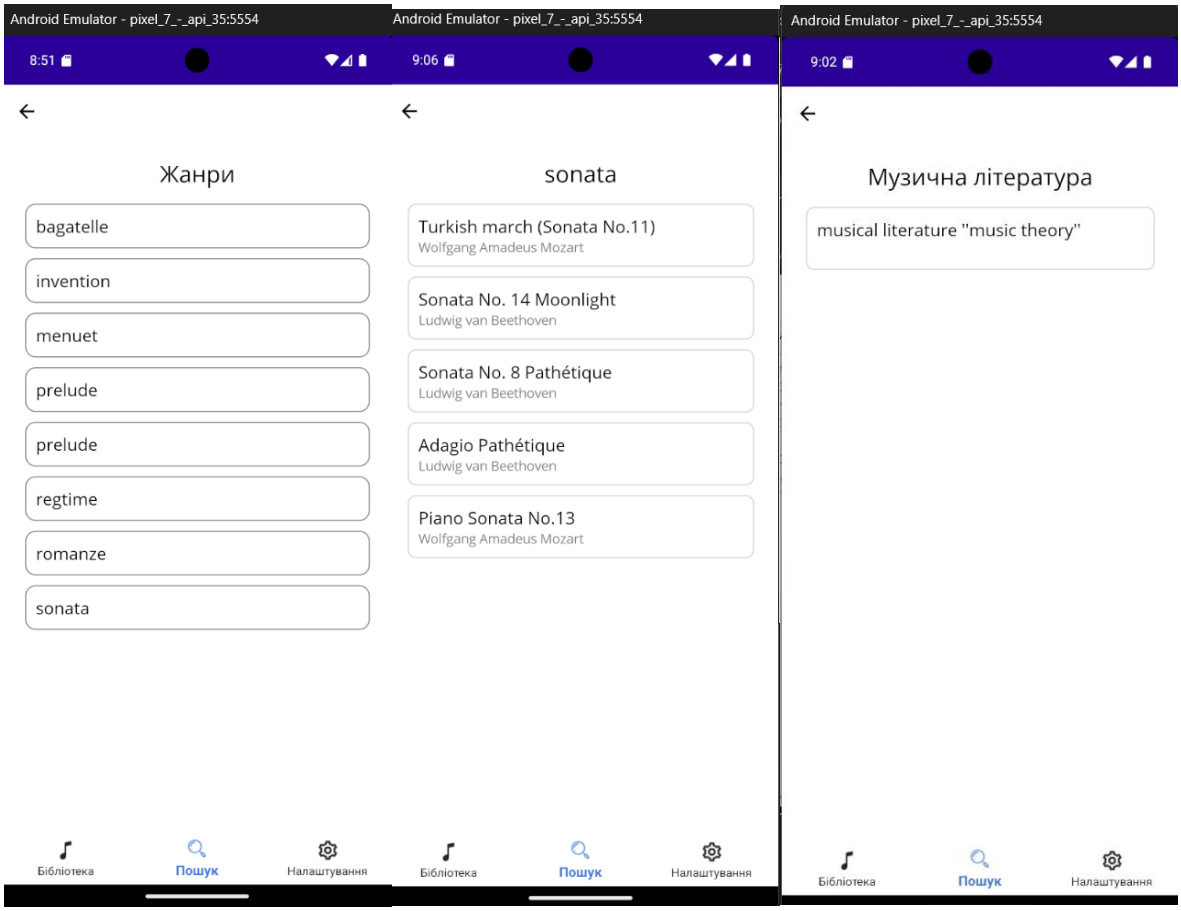


Рис. 4.19 Сторінка з унікальними композиторами

Перегляд нот та додавання нот до папки

Перегляд PDF-ноти здійснюється через Syncfusion PdfViewer. Було протестовано:

- завантаження документа з Firebase;
- перегляд у повноекранному режимі;
- доступ до кнопок "Додати до папки", "Поділитися", "Завантажити".

Також протестовано додання користувачем нот до однієї з особистих папок. Вибір папки здійснюється через DisplayActionSheet.

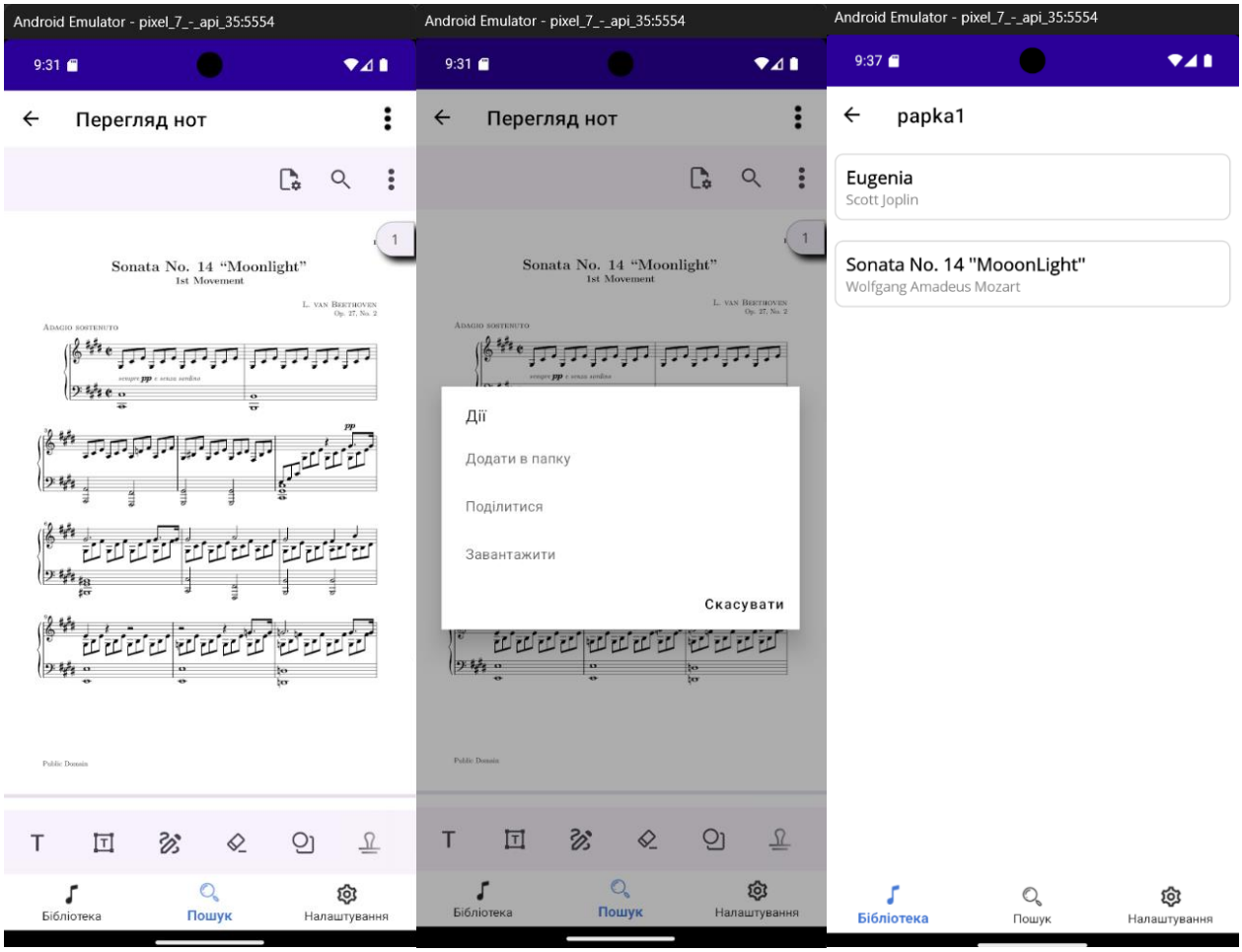


Рис. 4.20 Екранна форма реалізації додання нот до папки

Тестування налаштувань і зворотного зв’язку

На сторінці «Налаштування» було перевірено основні функції персоналізації та допомоги користувачу:

- Перемикання теми інтерфейсу (світла/темна): при зміні положення

перемикача одразу змінюється вигляд додатку. Після перезапуску застосунку зберігається вибрана тема завдяки збереженню параметру в Preferences.

- Перехід до сторінок "Акаунт", "FAQ" та "Підтримка": кожна кнопка правильно обробляє навігацію. Дані на сторінках відображаються коректно.
- Сторінка підтримки (зворотний зв’язок): перевірено два основні сценарії:
  - введення тексту в поле повідомлення і натискання кнопки «Надіслати» (з’являється сповіщення про успішне надсилання);
  - натискання кнопки відкриття поштового клієнта (здійснюється перехід у Gmail з готовим шаблоном листа).
- Обробка порожнього поля: при спробі надіслати порожнє повідомлення система показує повідомлення про помилку.

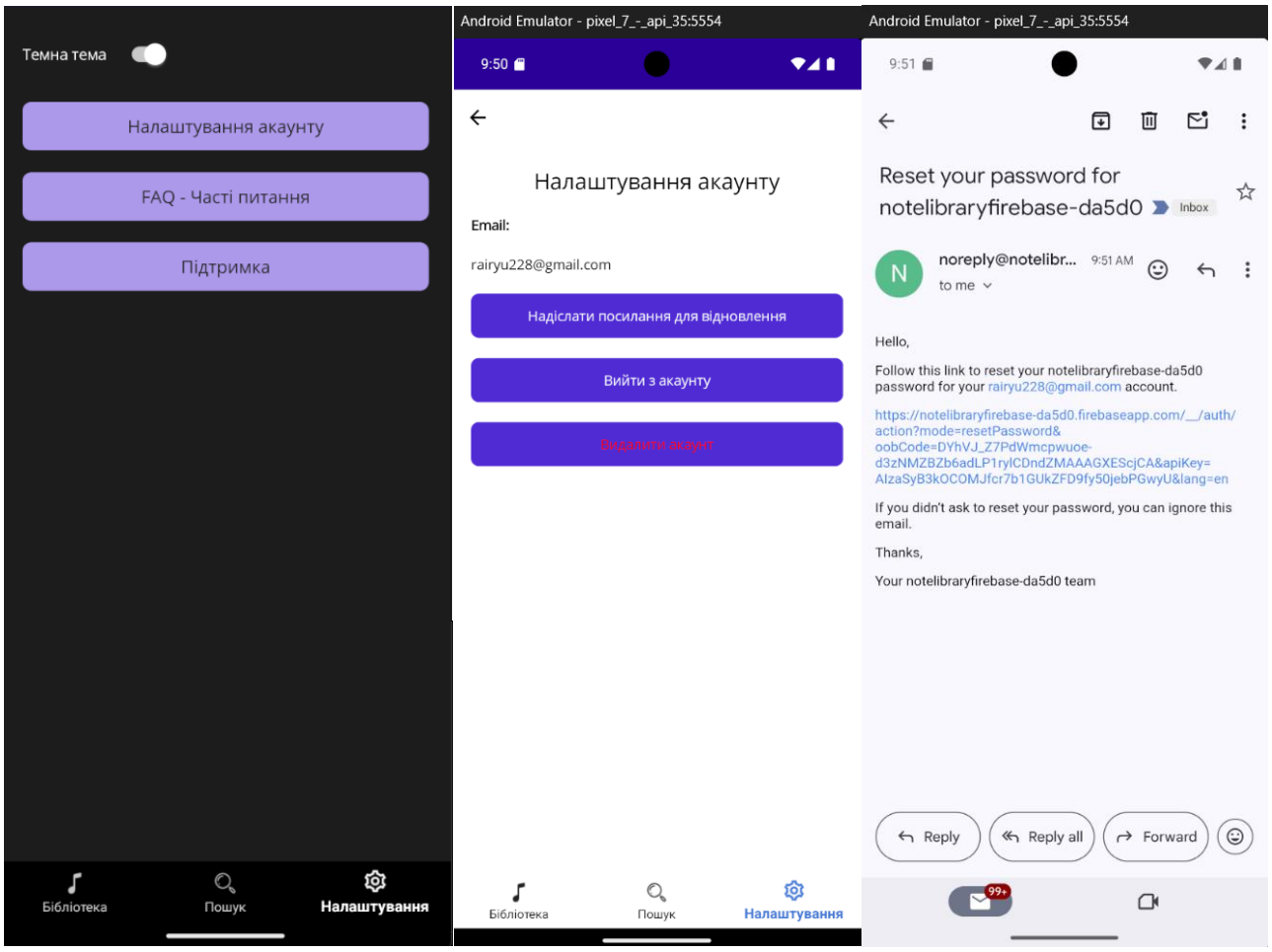


Рис. 4.21 Екранні форми реалізації сторінки налаштування аккаунту

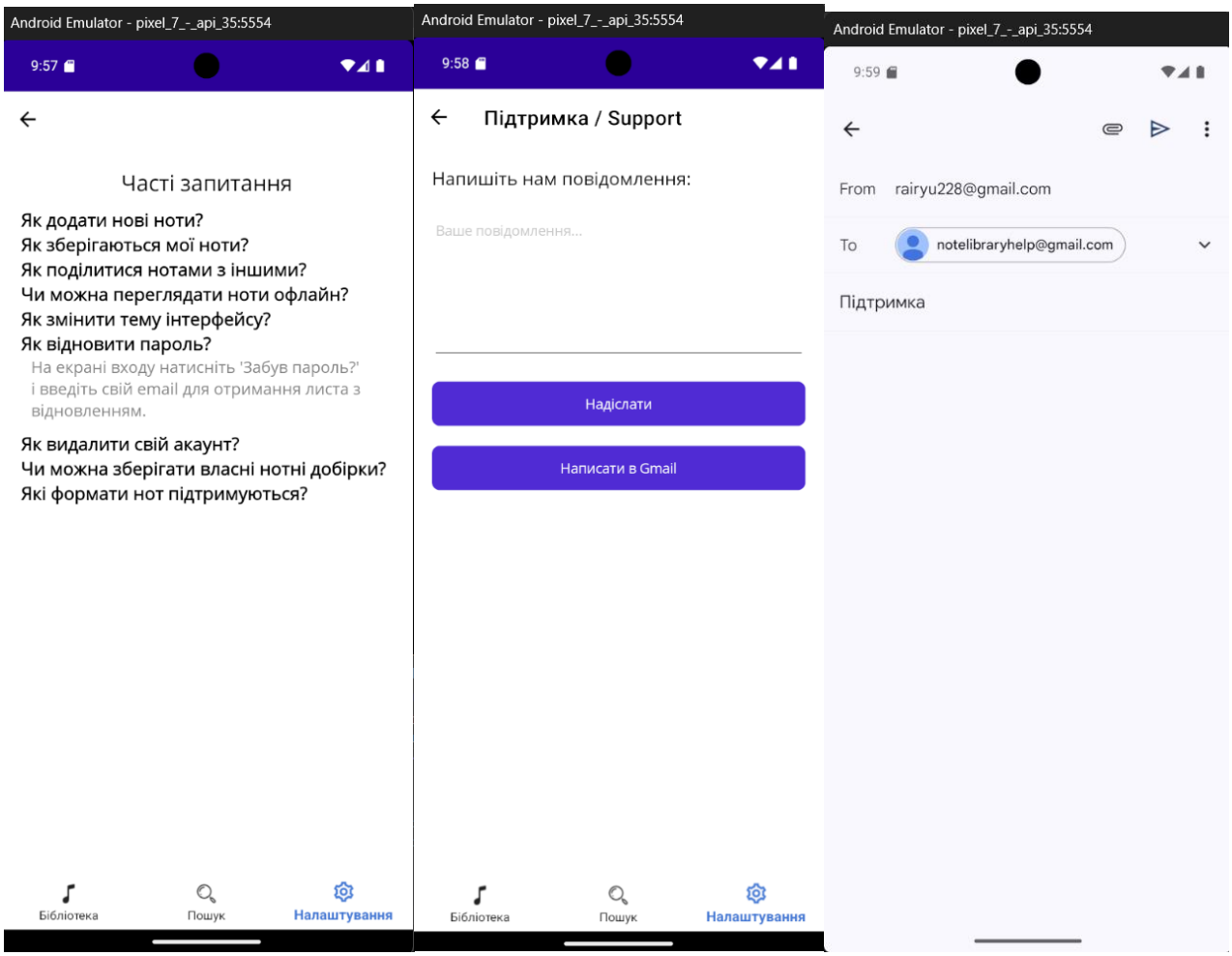


Рис. 4.22 Екранні форми сторінки FAQ та Підтримки

ВИСНОВКИ

У результаті виконання дипломного проєкту було повністю реалізовано цикл розробки мобільного застосунку Нотна бібліотека, призначеного для зручного пошуку, перегляду, завантаження та збереження нотних творів з хмарного середовища. Основна мета полягала в створенні інтуїтивного, функціонального інструменту для користувачів, які працюють із нотами на мобільних пристроях — як для навчання, так і для творчої діяльності.

У ході дослідження було проаналізовано існуючі програмні продукти на ринку, визначено їхні недоліки та обґрунтовано необхідність створення нового рішення. Основними критеріями стали простота інтерфейсу, синхронізація нотної бібліотеки, можливість збереження власних добірок нот, підтримка формату PDF, а також доступність без підключення до мережі.

Застосунок створено на базі сучасної платформи .NET MAUI, що дозволило забезпечити кросплатформенність (Android/Windows) з єдиною кодовою базою. Архітектура побудована за патерном MVVM, що забезпечує чіткий поділ між UI, логікою та моделями даних. Для зберігання нот і метаданих використано хмарну базу Firebase Firestore, а для завантаження та перегляду PDF-файлів — **Firebase** Storage у поєднанні з інтегрованим переглядачем Syncfusion.

Основні реалізовані функції включають:

- авторизацію користувача (реєстрація, вхід, вихід, відновлення паролю, видалення акаунту);
- пошук нот за назвою, композитором, жанром та категорією;
- перегляд нот у форматі PDF з можливістю відкриття, завантаження, поширення та додавання до особистих папок;
- створення**,** перегляд і видалення папок з нотами, які синхронізуються з акаунтом користувача;
- персональні налаштування, зокрема перемикання теми (світла/темна), перегляд акаунту;
- FAQ та підтримку, реалізовані у вигляді відповідних сторінок із

зручним доступом до поширених запитань та електронної пошти служби підтримки.

Завдяки використанню Firebase REST API та збереженню стану користувача локально, було реалізовано повноцінну синхронізацію нотної бібліотеки між пристроями, що забезпечує збереження добірок нот в акаунті користувача.

Функції перегляду нот реалізовані через вбудовану сторінку з переглядачем PDF, а на сторінці нот реалізовано контекстне меню з діями (додати до папки, поділитися, завантажити). Вміст нот у папках оновлюється динамічно, а видалення нот реалізовано подвійним натисканням.

У процесі тестування було перевірено всі функції, включно з обробкою нештатних ситуацій: відсутність інтернету, відсутність нот у бібліотеці, спроба дії без авторизації тощо. Застосунок продемонстрував **високу стабільність**, швидкодію та інтуїтивність інтерфейсу, що відповідає сучасним вимогам до мобільного ПЗ.

Таким чином, мобільний застосунок реалізує повний набір необхідних функцій для роботи з нотами на мобільних пристроях. Його архітектура є гнучкою, масштабованою і відкритою до розширення — як функціонального, так і інтерфейсного — у майбутніх версіях. Застосунок повністю відповідає поставленим вимогам і може бути рекомендований до використання як у навчальних закладах, так і у творчій діяльності.

Робота пройшла апробацію на двох наукових конференціях, за результатами апробації опубліковано тези доповідей:

1. Кулай А.С., Дібрівний О.А. Використання баз даних в мобільній розробці // Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ», 24.04.25, м. Київ, Збірник тез К.:ДУІКТ, 2025. С. 457-469.
2. Кулай А.С., Дібрівний О.А. Інформаційні технології в сучасній музичній освіті: досвід розробки нотної бібліотеки на .Net maui // Матеріали VI Міжнародної науково-технічної конференції «Сучасний стан та перспективи розвитку IoT», 15.04.2025, м. Київ, Україна (подано до друку).

ПЕРЕЛІК ПОСИЛАНЬ

1. Pressman R.S., Maxim B.R. Software Engineering: A Practitioner’s Approach. 9th ed. New York: McGraw-Hill Education, 2020. 910 p.

2. Sommerville I. Software Engineering. 10th ed. Harlow: Pearson Education Limited, 2016. 19 p.

3. MuseScore. URL: https://musescore.org (дата звернення: 17.05.2025).

4. Ukrainian Scores. URL: https://ukrainianscores.org (дата звернення: 17.05.1025).

5. Flat.io – Music Score Editor. URL: https://flat.io (дата звернення: 17.05.1025).

6. Firebase Documentation. URL: https://firebase.google.com/docs (дата звернення: 17.05.2025).

7. Allen G., Holub A. Developing Cross-Platform Apps with .NET MAUI. Berkeley: Apress, 2020. 388 p.

8. Microsoft Docs: .NET MAUI. URL: https://learn.microsoft.com/en-us/dotnet/maui (дата звернення: 17.05.2025).

9. Probert S. XAML in .NET MAUI: Build Native Apps with C#. Independently published, 2022. 256 p.

10. Syncfusion MAUI PDF Viewer. URL: https://www.syncfusion.com/maui-controls/pdf-viewer (дата звернення: 17.05.2025).

11. Newtonsoft.Json GitHub Repository. URL: https://github.com/JamesNK/Newtonsoft.Json (дата звернення: 17.05.2025).

12. Freeman E., Robson E. Head First Design Patterns. 2nd ed. Sebastopol: O’Reilly Media, 2020. 694 p.

13. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2002. 560 p.

14. Kruchten P. The Rational Unified Process: An Introduction. 3rd ed. Boston: Addison-Wesley, 2004. 336 p.

15. Laganiere R. Hands-On Design Patterns with C# and .NET Core. 2nd ed. Birmingham: Packt Publishing, 2021. 482 p.

16. Denny J. Firebase Essentials: Developing Cloud-Connected Android & iOS Apps with Firebase. Birmingham: Packt Publishing, 2022. 302 p.

17. Martin R.C. Clean Code: A Handbook of Agile Software Craftsmanship. Upper Saddle River: Prentice Hall, 2009. 464 p.

18. Lhotka R. Introducing .NET MAUI: Build Apps for Android, iOS, macOS, and Windows with .NET 6. Berkeley: Apress, 2021. 290 p.

19. Liberty J. Programming C# 8.0: Build Cloud, Web, and Desktop Applications. Sebastopol: O'Reilly Media, 2020. 720 p.

20. Ukrainian Live Classic. URL: https://ukrainianlive.org (дата звернення: 17.05.2025).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка застосунку "Нотна бібліотека"

Виконав студент 4 курсу

Групи ПД-44
Кулай Антон Сергійович

Керівник роботи

доктор філософії (PhD), доцент кафедри ІПЗ Дібрівний Олесь Андрійович

Київ – 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи - спрощення доступу до нотної інформації та підвищення зручності її використання за допомогою мобільного додатку нотної бібліотеки

Об’єк дослідження - процес цифрової каталогізації, зберігання та відображення нотного матеріалу за допомогою мобільних пристроїв.

Предмет дослідження - механізми, засоби та принципи реалізації мобільного застосунку для цифрової каталогізації, зберігання та управління нотним матеріалом.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. провести аналіз предметної галузі та існуючих програмних рішень для роботи з нотами;
2. визначити функціональні та нефункціональні вимоги до майбутнього застосунку;
3. обґрунтувати вибір середовища розробки, технологій і інструментів;
4. спроєктувати архітектуру системи, структуру бази даних та інтерфейс користувача;
5. реалізувати ключові функції пошуку: сортування, відображення нот, зберігання, управління бібліотекою;
6. провести тестування реалізованих функцій на цільових пристроях.

3

АНАЛІЗ АНАЛОГІВ

Додаток	Ukrainian Live Classic	Ukrainian Scores	MuseScore	Flat.io	Score Creator	MyLibraryApp
Створення власних папок			+	+	-	+
Пошук за назвою/жанром/композитором	+	+	+ (в акаунті)	+	-	+
Підтримка кирилиці	+	+	+	+	+	+
Перегляд нот у PDF	+	+	+	+	+	+
Синхронізація між пристроями	-	-	+	+	-	+
Завантаження нот з пристрою	-	-	+	+	+	+(обмежено)
Експорт нот у PDF/MusicXML	-	-	+ (через акаунт)		+(лише PDF)	+(лише PDF)
Спільний доступ до нот	-	-	+ через акаунт)	+ (публічні посилання)	-	+ (публічні посилання)
Офлайн-доступ	+	+	+(в додатку)	-	+	-
Авторизація користувача			+	+	-	+
Платформи	Android, iOS	Web	Android, iOS, Web	Android, iOS, Web	-	Android, iOS
Безкоштовність	+	+	+ (більшість функцій платні)	+ (з платними функціями)	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

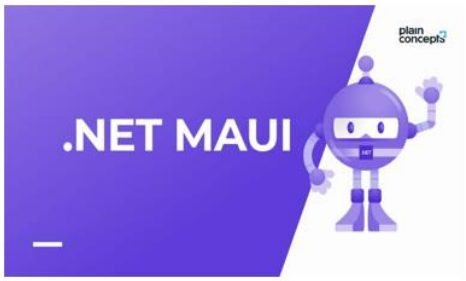
1. Управління обліковим записом користувача – можливість перегляду та редагування даних, зміни пароля, видалення акаунту.
2. Перегляд та організація нот – доступ до персональної бібліотеки користувача з можливістю створення папок, додавання нот до папок, перегляду нот у PDF-форматі, а також видалення нот із папки.
3. Функціонал пошуку та фільтрації – реалізація пошуку нот за назвою, жанром і композитором. Можливість відкривати ноти з категорій «Гами», «Сольфеджіо», «Музична література».
4. Функція завантаження нот – можливість завантажити ноту на пристрій у форматі PDF для офлайн-доступу.
5. Функція поширення нот – можливість поділитися нотами з іншими користувачами через стандартні засоби поширення.
6. Налаштування користувача – перемикання між світлою і темною темами інтерфейсу; доступ до розділів «FAQ» та, «Підтримка».

Нефункціональні вимоги:

1. **Кросплатформеність** – додаток має підтримувати роботу на мобільних пристроях з операційними системами Android (8.0+) та iOS (13.0+).
2. **Адаптивність** – інтерфейс додатку автоматично підлаштовується під розміри різних екранів мобільних пристроїв (на дисплеях 6,3 - 10,95 дюймів).
3. **Продуктивність** – відкриття нот у PDF-форматі повинне відбуватись не більше трьох секунд
4. **Сумісність** з Firebase – для зберігання нот, синхронізації даних та автентифікації користувачів використовується хмарна платформа Firebase.
5. **Безпека**
   Доступ до персональних даних здійснюється лише після автентифікації користувача через email/пароль. Відновлення пароля реалізоване через офіційний API Firebase.

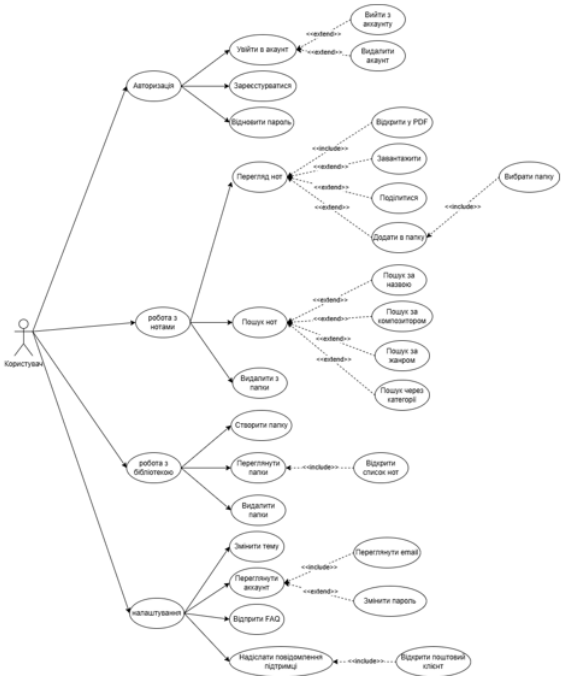
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



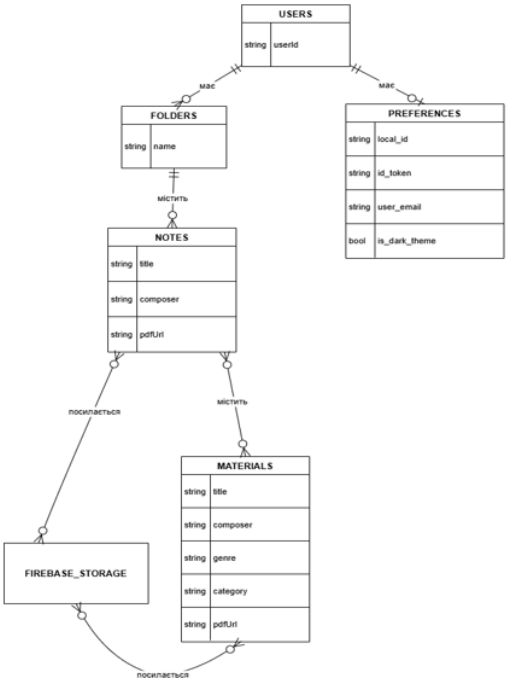
6

Діаграма варіантів використання



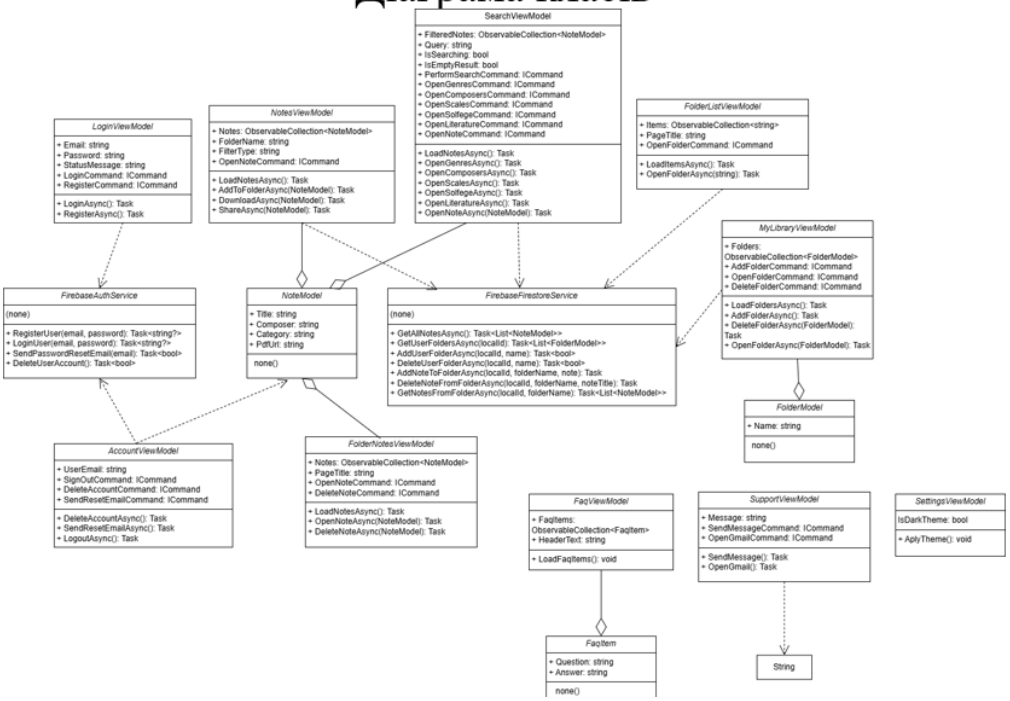
7

ER-діаграма



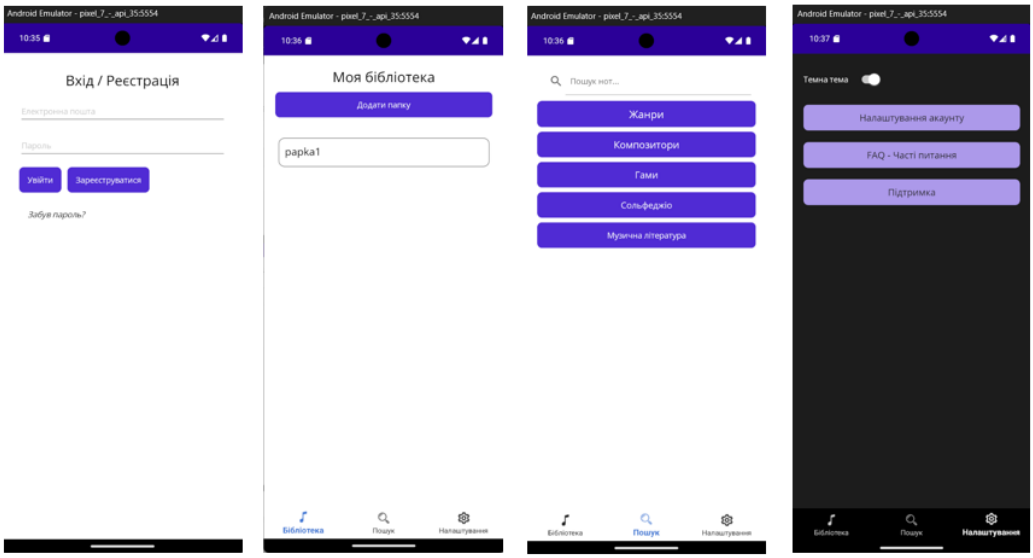
8

Діаграма класів



9

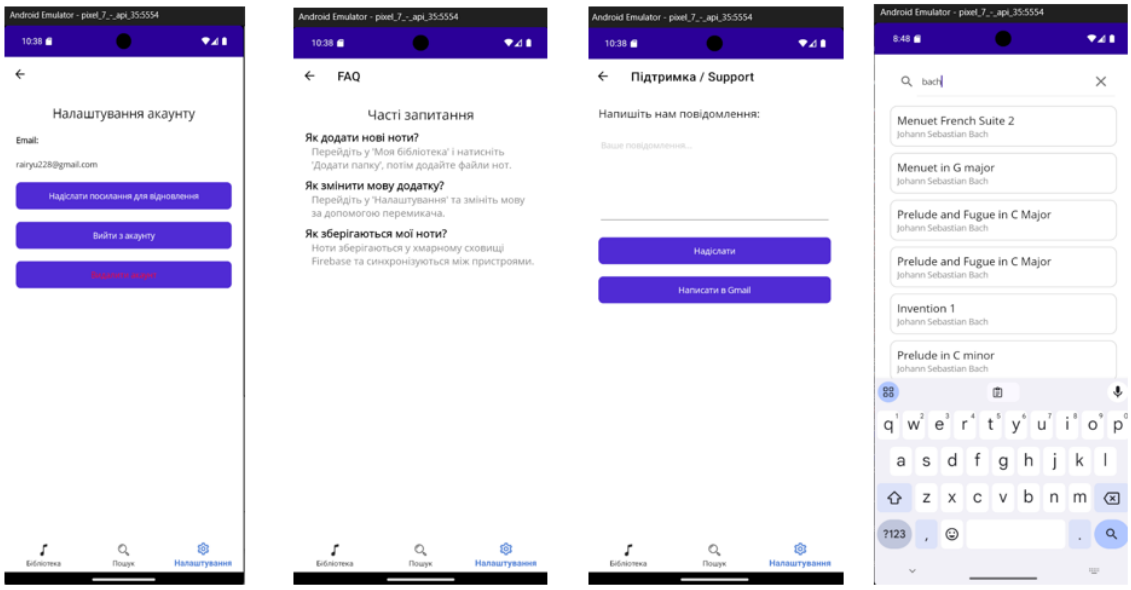
Екранні форми



Головні сторінки та сторінка авторизації

10

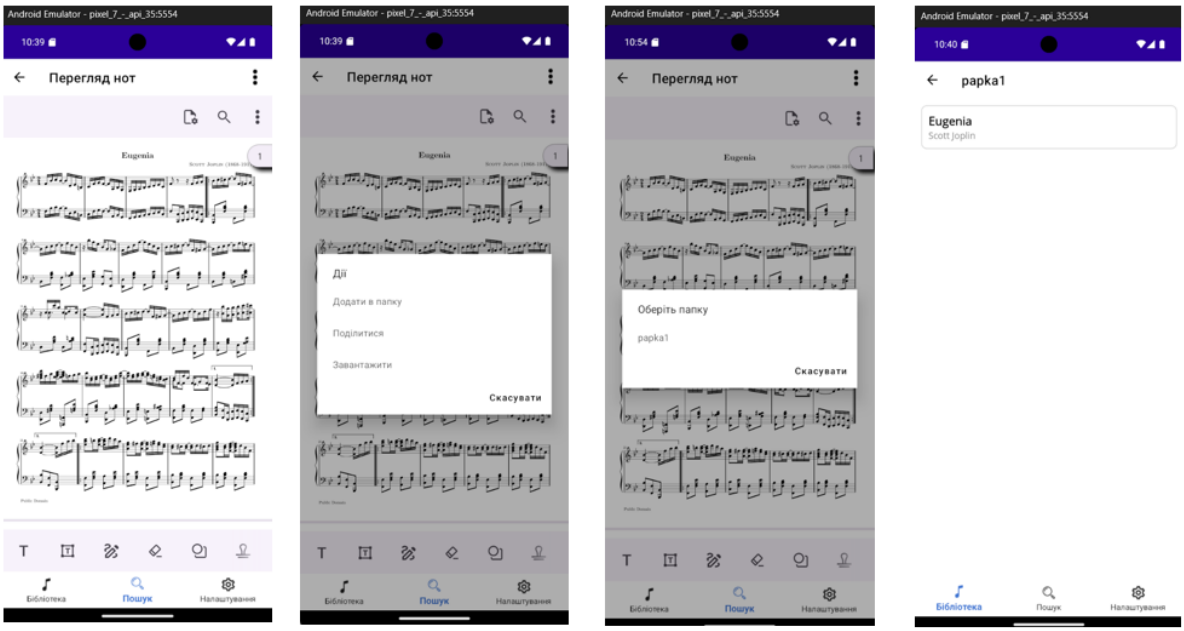
Екранні форми



Сторінки “Налаштування”, “FAQ”, “Підтримка” та “Пошук”

11

Екранні форми



Додання нот до папки

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Кулай А.С., Дібрівний О.А. Використання баз даних в мобільній розробці. Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ» Збірник тез. 24.04.25 м. Київ. К.:ДУІКТ, С. 457-469.
2. Кулай А.С., Дібрівний О.А. Інформаційні технології в сучасній музичній освіті: досвід розробки нотної бібліотеки на .Net maui. Матеріали Міжнародної науково-технічної конференції «Сучасний стан та перспективи розвитку IoT» Подано до друку

14

ВИСНОВКИ

1. Проведено огляд предметної галузі нотних бібліотек, аналіз цільової аудиторії та існуючих програмних продуктів, зокрема Ukrainian Live Classic, Ukrainian Scores, MuseScore, Flat.io та Score Creator; сформульовано вимоги до мобільного застосунку для зберігання та перегляду нот.
2. Проаналізовано засоби реалізації мобільного застосунку та обґрунтовано вибір інструментів розробки: мови C# і XAML, фреймворку .NET MAUI, середовища Visual Studio, а також сервісів Firebase (Authentication, Firestore, Storage), бібліотек Syncfusion PDF Viewer, CommunityToolkit.Maui та Newtonsoft.Json.
3. Запроєктовано архітектуру застосунку за шаблоном MVVM із використанням UML-діаграм, включно з діаграмою класів, компонентів і варіантів використання; спроєктовано базу даних на основі Firebase Firestore та визначено логіку збереження нот, папок і користувацьких метаданих.
4. Реалізовано мобільний застосунок “Нотна бібліотека”, що забезпечує пошук нот за назвою, жанром або композитором, перегляд нот у форматі PDF, створення персональних папок, збереження та синхронізацію даних між пристроями, авторизацію користувачів і керування акаунтом.
5. Проведено тестування застосунку, яке підтвердило працездатність функціональних модулів, стабільність взаємодії з Firebase та зручність користування інтерфейсом.

15

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Код, що реалізує функцію авторизації користувача:

```
namespace NoteLibraryApp.ViewModels
{
   public class LoginViewModel : INotifyPropertyChanged
   {
      public event PropertyChangedEventHandler PropertyChanged;

      private string _email;
      public string Email
      {
         get => _email;
         set { _email = value; OnPropertyChanged(nameof(Email)); }
      }

      private string _password;
      public string Password
      {
         get => _password;
         set { _password = value; OnPropertyChanged(nameof(Password)); }
      }

      private string _statusMessage;
      public string StatusMessage
      {
         get => _statusMessage;
         set { _statusMessage = value; OnPropertyChanged(nameof(StatusMessage)); }
      }

      public ICommand LoginCommand { get; }
      public ICommand RegisterCommand { get; }

      private readonly FirebaseAuthService _authService;

      public LoginViewModel()
      {
         _authService = new FirebaseAuthService();
         LoginCommand = new Command(async () => await LoginAsync());
         RegisterCommand = new Command(async () => await RegisterAsync());
      }

      private async Task LoginAsync()
      {
         StatusMessage = "Вхід...";
         var token = await _authService.LoginUser(Email, Password);
         if (token != null)
         {
            Preferences.Set("user_email", Email);

            StatusMessage = "Успішний вхід!";
            Application.Current.MainPage = new AppShell();
         }
         else
         {
            StatusMessage = "Помилка входу. Перевірте дані.";
         }
      }

      private async Task RegisterAsync()
      {
         StatusMessage = "Реєстрація...";
         var token = await _authService.RegisterUser(Email, Password);
         if (token != null)
         {
            Preferences.Set("user_email", Email);


            StatusMessage = "Успішна реєстрація!";
            Application.Current.MainPage = new AppShell();
         }
         else
         {
            StatusMessage = "Помилка реєстрації. Email може бути зайнятий.";
         }
      }

      private void OnPropertyChanged(string propertyName)
         => PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
   }
}
```

Код що реалізує пошук нот (включно з пошуком через папки)

```
namespace NoteLibraryApp.ViewModels
{
   public class NotesViewModel : INotifyPropertyChanged
   {
      public event PropertyChangedEventHandler PropertyChanged;

      public ObservableCollection<NoteModel> Notes { get; set; } = new();

      private string _folderName;
      public string FolderName
      {
         get => _folderName;
         set
         {
            _folderName = value;
            OnPropertyChanged();
         }
      }

      private string _filterType;
      public string FilterType
      {
         get => _filterType;
         set
         {
            _filterType = value;
            OnPropertyChanged();
         }
      }

      public ICommand OpenNoteCommand { get; }

      public NotesViewModel(string folderName, string filterType)
      {
         FolderName = folderName;
         FilterType = filterType;

         OpenNoteCommand = new Command<NoteModel>(async (note) =>
         {
            if (!string.IsNullOrWhiteSpace(note?.PdfUrl))
            {
               var encodedUrl = Uri.EscapeDataString(note.PdfUrl);
               await Shell.Current.GoToAsync($"noteviewer?pdfUrl={encodedUrl}");
            }
         });

         _ = LoadNotesAsync();
      }

      private async Task LoadNotesAsync()
      {
         Notes.Clear();

         var service = new FirebaseFirestoreService();
         var allNotes = await service.GetAllNotesAsync();

         IEnumerable<NoteModel> filtered;

         if (FilterType == "genre")
            filtered = allNotes.Where(n => n.Genre == FolderName);
         else if (FilterType == "composer")
            filtered = allNotes.Where(n => n.Composer == FolderName);
         else if (FilterType == "category")
         {
            string categoryKey = FolderName switch
            {
               "Гами" => "scales",
               "Сольфеджіо" => "solfege",
               "Музична література" => "literature",
               _ => ""
            };

            filtered = allNotes.Where(n => n.Category == categoryKey);
         }
         else
            filtered = Enumerable.Empty<NoteModel>();

         foreach (var note in filtered)
            Notes.Add(note);
      }

      public async Task AddToFolderAsync(NoteModel note)
      {
         var localId = Preferences.Get("user_localId", "");
         if (string.IsNullOrWhiteSpace(localId))
         {
            await Application.Current.MainPage.DisplayAlert("Помилка", "Не знайдено користувача", "OK");
            return;
         }

         string title = await Application.Current.MainPage.DisplayPromptAsyn
```

```
c(
            "Назва твору", "Введіть назву твору:");

        if (string.IsNullOrWhiteSpace(title))
        {
            await
Application.Current.MainPage.DisplayAlert("Помилка", "Назву твору не введено", "OK");
            return;
        }

        string composer = await
Application.Current.MainPage.DisplayPromptAsync(
            "Композитор", "Введіть ім’я композитора (за бажанням):");

        var service = new
FirebaseFirestoreService();
        var folders = await
service.GetUserFoldersAsync(localId);

        if (folders == null || folders.Count == 0)
        {
            await
Application.Current.MainPage.DisplayAlert("Папки не знайдено", "У вас немає жодної створеної папки", "OK");
            return;
        }

        string[] folderNames = folders.Select(f =>
f.Name).ToArray();
        string selectedFolder = await
Application.Current.MainPage.DisplayActionSheet(
            "Оберіть папку", "Скасувати", null,
folderNames);

        if
(string.IsNullOrWhiteSpace(selectedFolder) ||
selectedFolder == "Скасувати")
            return;

        var updatedNote = new NoteModel
        {
            Title = title,
            Composer = composer,
            PdfUrl = note.PdfUrl
        };

        try
        {
            await
service.AddNoteToFolderAsync(localId,
selectedFolder, updatedNote);
            await
Application.Current.MainPage.DisplayAlert("Успіх", $"Ноту додано до папки «{selectedFolder}»",
"OK");
        }
        catch (Exception ex)
        {
            await
Application.Current.MainPage.DisplayAlert("Помилка", ex.Message, "OK");
        }
    }


    public async Task
DownloadAsync(NoteModel note)
    {
        try
        {
            if
(string.IsNullOrWhiteSpace(note.PdfUrl))
            {
                await
Application.Current.MainPage.DisplayAlert("Помилка", "URL файлу відсутній.", "OK");
                return;
            }

            using var httpClient = new HttpClient();
            var pdfBytes = await
httpClient.GetByteArrayAsync(note.PdfUrl);

            string fileName =
$"{note.Title}.pdf".Replace(" ", "_");
            string filePath =
Path.Combine(FileSystem.AppDataDirectory,
fileName);
            File.WriteAllBytes(filePath, pdfBytes);

            await
Application.Current.MainPage.DisplayAlert("Успіх", $"Файл збережено у:\n{filePath}", "OK");
        }
        catch (Exception ex)
        {
            await
Application.Current.MainPage.DisplayAlert("Помилка", $"Не вдалося завантажити файл:
{ex.Message}", "OK");
        }
    }


    public async Task ShareAsync(NoteModel
note)
    {
        try
        {
```

```
            if
(string.IsNullOrWhiteSpace(note.PdfUrl))
            {
                await
Application.Current.MainPage.DisplayAlert("Поми
лка", "URL файлу відсутній.", "OK");
                return;
            }

            using var httpClient = new HttpClient();
            var pdfBytes = await
httpClient.GetByteArrayAsync(note.PdfUrl);

            string fileName =
$"{note.Title}.pdf".Replace(" ", "_");
            string filePath =
Path.Combine(FileSystem.CacheDirectory,
fileName);
            File.WriteAllBytes(filePath, pdfBytes);

            await Share.RequestAsync(new
ShareFileRequest
Робота з власною бібліотекою
public class MyLibraryViewModel :
INotifyPropertyChanged { public
ObservableCollection Folders { get; set; } = new();
public ICommand OpenFolderCommand { get; }
public ICommand AddFolderCommand { get; }
public ICommand DeleteFolderCommand { get; }
private readonly FirebaseFirestoreService
_firestoreService;
private readonly string _userId;

public event PropertyChangedEventHandler
PropertyChanged;
private void
OnPropertyChanged([CallerMemberName] string
name = null) =>
  PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(name));

public MyLibraryViewModel()
{
  _firestoreService = new
FirebaseFirestoreService();
  _userId = Preferences.Get("user_localId", "");

  AddFolderCommand = new Command(async ()
=> await AddFolder());
  OpenFolderCommand = new
Command<FolderModel>(async (folder) =>
  {
    if (folder != null)
      await
Shell.Current.GoToAsync($"foldernotes?folderNa
me={folder.Name}");
  });
            {
                Title = "Поділитися нотами",
                File = new ShareFile(filePath)
            });
        }
        catch (Exception ex)
        {
            await
Application.Current.MainPage.DisplayAlert("Поми
лка", $"Не вдалося поділитися файлом:
{ex.Message}", "OK");
        }
    }

    private void
OnPropertyChanged([CallerMemberName] string
propertyName = null) =>
        PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
  }
}



  DeleteFolderCommand = new
Command<FolderModel>(async (folder) =>
  {
    if (folder != null)
      await DeleteFolder(folder);
  });

  _ = LoadFoldersAsync();
}

private async Task LoadFoldersAsync()
{
  Folders.Clear();
  var folders = await
_firestoreService.GetUserFoldersAsync(_userId);
  foreach (var folder in folders)
    Folders.Add(folder);
}

private async Task AddFolder()
{
  string name = await
Application.Current.MainPage.DisplayPromptAsyn
c("Нова папка", "Введіть назву нової папки:");
  if (!string.IsNullOrWhiteSpace(name))
  {
    var folder = new FolderModel { Name = name
};
    await
_firestoreService.AddUserFolderAsync(_userId,
folder.Name);
    Folders.Add(folder);
  }
}
```

```
public async Task DeleteFolder(FolderModel
folder)
{
  if (Folders.Contains(folder))
  {
     Folders.Remove(folder);
     await
_firestoreService.DeleteUserFolderAsync(_userId,
folder.Name);
  }
}

}
```

Код що реалізує перегляд нот у власній папці:

```
public class FolderNotesViewModel :
BindableObject
{
   public ObservableCollection<NoteModel> Notes
{ get; set; } = new();

   public ICommand OpenNoteCommand { get; }
   public ICommand DeleteNoteCommand { get; }

   private string folderName;
   private string localId;

   public string PageTitle => folderName;

   public FolderNotesViewModel()
   {
      OpenNoteCommand = new
Command<NoteModel>(async (note) => await
OpenNoteAsync(note));
      DeleteNoteCommand = new
Command<NoteModel>(async (note) => await
DeleteNoteAsync(note));
   }

   public async Task InitializeAsync(string folder)
   {
      folderName = folder;
      localId = Preferences.Get("user_localId", "");

      var service = new FirebaseFirestoreService();
      var notes = await
service.GetNotesFromFolderAsync(localId,
folderName);

      Notes.Clear();
      foreach (var note in notes)
         Notes.Add(note);

      OnPropertyChanged(nameof(PageTitle));
   }

   private async Task OpenNoteAsync(NoteModel
note)
   {
      var url = Uri.EscapeDataString(note.PdfUrl);
      var title = Uri.EscapeDataString(note.Title ??
"");
      var composer =
Uri.EscapeDataString(note.Composer ?? "");

      await
Shell.Current.GoToAsync($"noteviewer?pdfUrl={u
rl}&title={title}&composer={composer}");
   }

   private async Task DeleteNoteAsync(NoteModel
note)
   {
      bool confirm = await
Application.Current.MainPage.DisplayAlert("Вида
лити", $"Видалити «{note.Title}» з папки?",
"Так", "Ні");
      if (!confirm) return;

      var service = new FirebaseFirestoreService();
      await
service.DeleteNoteFromFolderAsync(localId,
folderName, note.Title);
      Notes.Remove(note);
   }
}
```

Код що реалізує перегляд нот у PDF :

```
[QueryProperty(nameof(PdfUrl), "pdfUrl")]
[QueryProperty(nameof(Title), "title")]
[QueryProperty(nameof(Composer), "composer")]
[QueryProperty(nameof(Genre), "genre")]
[QueryProperty(nameof(Category), "category")]

public partial class NoteViewerPage : ContentPage
{
   private NoteModel CurrentNote = new
NoteModel();

   public string PdfUrl
   {
```

```
        set
        {
            if (!string.IsNullOrWhiteSpace(value))
            {
                CurrentNote ??= new NoteModel();
                CurrentNote.PdfUrl = value;
                _ = LoadPdfAsync(value);
            }
        }
    }

    public string Title
    {
        set => (CurrentNote ??= new
NoteModel()).Title = value;
    }

    public string Composer
    {
        set => (CurrentNote ??= new
NoteModel()).Composer = value;
    }

    public string Genre
    {
        set => (CurrentNote ??= new
NoteModel()).Genre = value;
    }

    public string Category
    {
        set => (CurrentNote ??= new
NoteModel()).Category = value;
    }

    public NoteViewerPage()
    {
        InitializeComponent();
    }

    private async Task LoadPdfAsync(string url)
    {
        try
        {
            using var httpClient = new HttpClient();
            var pdfBytes = await
httpClient.GetByteArrayAsync(url);

            string fileName =
$"note_{Guid.NewGuid()}.pdf";
            string localPath =
Path.Combine(FileSystem.CacheDirectory,
fileName);
            File.WriteAllBytes(localPath, pdfBytes);

            var stream = File.OpenRead(localPath);
            PdfViewer.LoadDocument(stream);
        }
        catch (Exception ex)
        {
            await DisplayAlert("Помилка", $"Не
вдалося завантажити PDF: {ex.Message}",
"OK");
        }
    }



    private async void
OnMoreOptionsClicked(object sender, EventArgs
e)
    {
        if (CurrentNote == null)
        {
            await DisplayAlert("Помилка", "Нота не
завантажена", "OK");
            return;
        }

        string action = await
DisplayActionSheet("Дії", "Скасувати", null,
            "Додати в папку", "Поділитися",
"Завантажити");

        var vm = new NotesViewModel("", "");

        switch (action)
        {
            case "Додати в папку":
                await
vm.AddToFolderAsync(CurrentNote);
                break;
            case "Поділитися":
                await vm.ShareAsync(CurrentNote);
                break;
            case "Завантажити":
                await vm.DownloadAsync(CurrentNote);
                break;
        }
    }
}
```