ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МІСЬКОГО ГОСПОДАРСТВА ІМЕНІ О. М. БЕКЕТОВА

Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему: Створення мобільного застосунку для читання документів з розширеним функціоналом

Виконав: здобувач вищої освіти 4 курсу,
групи КН 2022–1
спеціальності 122 «Комп'ютерні науки»
(шифр і назва спеціальності)

Ілля ОЛЕНЧЕНКОВ
(прізвище та ініціали)

Керівник: Юрій ЛЕВІКОВ
(прізвище та ініціали)

Рецензент: Марія ВОЄВОДІНА
(прізвище та ініціали)

м. Харків – 2026 рік

Харківський національний університет міського господарства імені О. М. Бекетова
(повне найменування закладу вищої освіти)
Навчально-науковий Інститут енергетичної, інформаційної
та транспортної інфраструктури
Кафедра Комп'ютерних наук та інформаційних технологій
Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 Комп'ютерні науки
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КНтаІТ

Марина НОВОЖИЛОВА
«_26_» _червня_____2026 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Оленченков Ілля Володимирович
(прізвище, ім'я, по батькові)

1. Тема роботи Створення мобільного застосунку для читання документів з розширеним функціоналом

керівник роботи Левіков Юрій Володимирович ст. викладач
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від «22» травня 2026 р. № 440-03

2. Термін подання роботи здобувачем вищої освіти

3. Вихідні дані до роботи Проєктування та розробка нативного мобільного застосунку з розширеним функціоналом під операційну систему Android

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз предметного середовища нативних рідерів та огляд аналогів, проєктування структури бази даних і архітектури синхронізації, програмна реалізація модулів рендерингу документів, розробка інтерфейсу користувача та керівництва

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
17 аркушів

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	Юрій ЛЕВІКОВ, старший викладач кафедри КН та ІТ	11.05.2026	17.05.2026
Розділ II	Юрій ЛЕВІКОВ, старший викладач кафедри КН та ІТ	18.05.2026	05.06.2026
Розділ III	Юрій ЛЕВІКОВ, старший викладач кафедри КН та ІТ	06.06.2026	11.06.2026
Розділ IV	Вікторія МАЛИШЕВА, к. т., н., доцент кафедри ОП та БЖ	12.06.2026	14.06.2026

7. Дата видачі завдання 11.05.2026

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вибір тема дипломної роботи	11.05.2026	Виконано
2	Затвердження тем, наукових керівників та календарного плану підготовки кваліфікованої роботи	13.05.2026	Виконано
3	Написання I розділу	17.05.2026	Виконано
4	Написання II розділу	05.06.2026	Виконано
5	Написання III розділу	11.06.2026	Виконано
6	Написання IV розділу	14.06.2026	Виконано
7	Подання дипломної роботи керівнику	15.06.2026	Виконано
8	Робота по усуненню зауважень керівника, уточнення і доповнення практичного матеріалу, оформлення додатків до роботи	15.06.2026	Виконано
9	Подання допрацьованого варіанту роботи керівнику	16.06.2026	Виконано
10	Захист матеріалів дипломної роботи на засіданні кафедри	18.06.2026	Виконано
11	Офіційний захист матеріалів дипломної роботи на засіданні екзаменаційної комісії	23.06.2026	Виконано

Здобувач вищої освіти _____ Ілля ОЛЕНЧЕНКОВ

(підпис) (прізвище та ініціали)

Керівник роботи _____ Юрій ЛЕВІКОВ

(підпис) (прізвище та ініціали)"

АНОТАЦІЇ

Структура та обсяг роботи. Пояснювальна записка кваліфікаційної роботи бакалавра здобувача вищої освіти групи КН 2022-1 спеціальності 122 «Комп'ютерні науки» Оленченкова Іллі Володимировича на тему «Створення мобільного застосунку для читання документів з розширеним функціоналом» складається з 4 розділів, містить 97 сторінок, 23 рисунків, 4 таблицю та 23 літературних джерел.

Метою роботи є розробка мобільного застосунку для читання книг та документів із засобами покращення читацького досвіду та гейміфікації.

Об'єктом дослідження є процес рендерингу та обробки текстових файлів різних форматів на Android-пристроях.

Предметом дослідження є архітектурні шаблони Clean Architecture/MVVM, методи пагінації тексту та технології відображення документів.

Основне завдання – реалізувати застосунок ReadOl з підтримкою форматів PDF та EPUB, лінійкою фокусування уваги, каскадним перекладом (Lingva API + ML Kit), озвученням тексту (TTS) та віджетами.

Результатом роботи є готовий до використання Android-застосунок ReadOl.

Ключові слова: МОБІЛЬНИЙ ЗАСТОСУНОК, ЧИТАННЯ ДОКУМЕНТІВ, JETPACK COMPOSE, КАСКАДНИЙ ПЕРЕКЛАД, ГЕЙМІФІКАЦІЯ, СИНТЕЗ МОВЛЕННЯ.

ANNOTATION

Structure and volume of the work. The explanatory note of the bachelor's qualification thesis of the higher education student of group KN 2022-1 in specialty 122 "Computer Science", Illia Olenchenkov, on the topic "Creation of a mobile application for reading documents with advanced functionality" consists of 4 chapters and contains 97 pages, 23 figures, 4 tables, and 23 references.

The aim of the project is to develop a mobile application for reading books and documents with reading experience and gamification tools.

The object of study is the rendering and processing of text files of various formats on Android devices.

The subject of study is Clean Architecture/MVVM patterns, text pagination methods, and document rendering technologies.

The main task is to implement the ReadOl application supporting PDF and EPUB, a focus ruler, fallback translation (Lingva API + ML Kit), text-to-speech (TTS), and widgets.

The result of the work is a ready-to-use Android application ReadOl.

Keywords: MOBILE APPLICATION, DOCUMENT READING, JETPACK COMPOSE, FALLBACK TRANSLATION, GAMIFICATION, TEXT-TO-SPEECH.

ЗМІСТ

ВСТУП

Розвиток інформаційних технологій характеризується повсюдним переходом до використання цифрових документів, а мобільні пристрої стали головним інструментом для навчання, роботи та читання. Зростання обсягів цифрових книг висуває нові вимоги до софту. Користувачі потребують інструментів, що швидко відображають графічні (PDF, CBZ) та адаптують текстові формати (EPUB, FB2, TXT) під мобільні екрани.

Дослідження ринку мобільного софту показує, що більшість безкоштовних застосунків мають суттєві обмеження: вони містять нав'язливу рекламу, не забезпечують стабільної роботи з великими файлами або вимагають оплати за можливість автоматичної синхронізації даних між різними пристроями користувача [1]. Окрім суто технічних проблем відображення тексту, існує проблема систематичного зниження читацької активності через відсутність у програмах дієвих локальних механізмів підтримки інтересу та фіксації щоденного прогресу. В епоху домінування короткоформатного медіаконтенту (відеороликів типу Shorts, Reels та TikTok), який веде до формування кліпового мислення та розпорошення уваги, розробка застосунку для читання книг набуває важливої соціальної складової, оскільки спрямована на популяризацію системного читання та відновлення стійкої звички сприйняття довгих текстів. Створення безкоштовного мобільного застосунку з розширеним функціоналом, який поєднує високопродуктивне ядро рендерингу, зручну систему анотування та автоматичний безкоштовний хмарний бекап, є актуальним завданням для сучасного B2C–сегмента мобільного програмного забезпечення.

Метою роботи є теоретичне обґрунтування, проєктування та програмна реалізація мобільного застосунку для читання документів, який забезпечує швидку та стабільну обробку файлів різних типів, надає розширений

локальний інструментарій користувача та здійснює автоматичне збереження прогресу без використання платних підписок.

Об’єктом дослідження є процес функціонування мобільного програмного забезпечення для читання електронних документів на портативних пристроях.

Предметом дослідження є методи та архітектурні шаблони розробки нативного мобільного софту, алгоритми асинхронної обробки важких файлів та методи організації локального й хмарного збереження даних.

Для вирішення поставлених завдань використано методи системного аналізу предметної області та порівняльного аналізу програмних продуктів. Проєктування системи виконано на основі об'єктно–орієнтованого аналізу та моделювання мовою UML. Експериментальні методи та аналіз профілювання пам'яті застосовано під час тестування розробленого програмного забезпечення.

Теоретична цінність роботи полягає у систематизації архітектурних підходів до побудови мобільних систем читання документів та обґрунтуванні вибору якісних атрибутів програмного забезпечення при роботі з підвищеним навантаженням на оперативну пам'ять мобільних пристроїв.

Практична значущість отриманих результатів полягає у створенні повністю працездатного, оптимізованого мобільного застосунку ReadOl для платформи Android. Продукт дозволяє користувачам ефективно працювати з файлами форматів PDF, EPUB, FB2, DOCX, TXT, CBZ, MD, здійснювати гнучке форматування тексту, вести базу структурованих нотаток та автоматично синхронізувати свій читацький прогрес через безкоштовні хмарні сервіси.

РОЗДІЛ 1
ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Сучасна сфера мобільних програм для читання електронних документів розвивається в умовах значного насичення ринку готовими рішеннями. Проте користувачі постійно стикаються з проблемою вибору програмного забезпечення, яке б поєднувало швидку роботу з різними форматами файлів та дієві механізми підтримки звички до читання. Більшість існуючих безкоштовних мобільних читалок орієнтовані на простий показ тексту та не мають інтегрованих інструментів для відстеження щоденного прогресу. Це призводить до швидкої втрати мотивації з боку читача та зниження загального рівня споживання навчального чи художнього контенту.

Розробка мобільного застосунку ReadOl спрямована на вирішення цієї проблеми шляхом створення продукту для відкритого B2C–ринку. Програма поєднує в собі оптимізоване ядро для рендерингу документів різних типів та внутрішню систему мотивації, що базується на фіксації щоденних досягнень і серій безперервного читання.

Вибір технологічного стеку для реалізації такого мобільного застосунку є першим етапом проєктування. Згідно з порівняльним аналізом, нативна розробка для платформи Android з використанням мови програмування Kotlin має суттєві переваги перед кросплатформними чи гібридними рішеннями. Нативне програмне забезпечення отримує прямий доступ до апаратних ресурсів мобільного пристрою та системних інтерфейсів програмування (API). Це є визначальним фактором при обробці великих файлів документів, таких як технічні PDF–інструкції чи графічні книги, де швидкість відображення сторінок безпосередньо впливає на задоволеність користувача. Цільовою операційною системою було обрано саме Android, оскільки відкритість її

архітектури дозволяє реалізувати гнучкий доступ до локальної файлової системи та інтегрувати нативні бібліотеки рендерингу мовою C++ без жорстких обмежень ізольованого середовища виконання (App Sandboxing) та правил доступу до спільних ресурсів, які притаманні платформі iOS. Окрім цього, орієнтація на ОС Android забезпечує максимальне охоплення ринку мобільних пристроїв, роблячи продукт доступним для найширшого кола користувачів.

При проєктуванні архітектури мобільного застосунку ReadOl особливу увагу приділено показникам якості програмного забезпечення. У роботах, присвячених оцінці мобільних архітектур [2], вказано, що при оцінці мобільних архітектур визначальними є такі атрибути якості, як продуктивність, надійність, доступність та модифікованість. Продуктивність визначає час реакції програми на дії користувача (наприклад, перегортання сторінки), що є критичним в умовах обмеженості системних ресурсів мобільних процесорів та оперативної пам'яті. Надійність забезпечує роботу застосунку без збоїв навіть у ситуаціях з нестабільним мережевим підключенням, а доступність гарантує можливість читання завантажених книг в автономному режимі. Модифікованість архітектури дозволяє в майбутньому легко додавати підтримку нових форматів документів чи змінювати логіку мотиваційних віждетів без переписування всього коду. Для досягнення цих показників рекомендовано використовувати архітектурний шаблон MVVM (Model–View–ViewModel), який забезпечує чіткий поділ між інтерфейсом користувача та бізнес–логікою.

Тенденції розвитку інтерфейсів мобільних застосунків вказують на перехід до декларативного підходу. У дослідженнях ринку та трендів розробки [3] вказано, що використання сучасних декларативних фреймворків, таких як Jetpack Compose для Android (впровадженого в індустрію з 2019 року), дозволяє значно спростити розробку складних адаптивних інтерфейсів. Декларативний підхід забезпечує плавний скролінг великих бібліотек книг та

миттєве оновлення екрана при зміні стану даних. Це усуває потребу в ручному оновленні елементів відображення, що зменшує ймовірність виникнення помилок синхронізації інтерфейсу із внутрішнім станом програми.

Важливим етапом проектування є вибір системи збереження даних на мобільному пристрої. У наукових оглядах систем керування базами даних для різних операційних систем [4] вказано, що для локального збереження структурованої інформації в ОС Android стандартним вибором є реляційна база даних SQLite, доступ до якої спрощується за допомогою об'єктно–реляційного відображення (ORM) Room. Використання Room забезпечує відповідність транзакцій вимогам ACID, що гарантує надійність збереження історії читання, закладок та налаштувань користувача навіть при раптовому вимкненні пристрою. Водночас нереляційні рішення (NoSQL), такі як Firebase Firestore, є доцільними для побудови хмарного дзеркала бази даних, що дозволяє реалізувати резервне копіювання профілю користувача без перевантаження локальних ресурсів смартфона.

Специфіка роботи з важкими файлами документів вимагає реалізації асинхронних процесів. У порівняльних аналізах функціональності асинхронного виконання операцій [5] доведено, що виконання операцій імпорту документів, парсингу метаданих та рендерингу графічних сторінок безпосередньо в головному потоці інтерфейсу (UI thread) призводить до замерзання програми та появи системних попереджень про помилку. Використання механізму корутин (Kotlin Coroutines) дозволяє виконувати ці тривалі обчислення у фонових потоках (зокрема, із застосуванням диспетчера Dispatchers.IO для операцій введення–виведення та Dispatchers.Default для важких математичних розрахунків рендерингу). Після завершення обробки отриманий результат безпечно повертається в основний потік для відображення на екрані, що забезпечує безперервну та плавну взаємодію користувача з інтерфейсом застосунку.

1.1.1 Опис процесу діяльності

Опис процесу діяльності застосунку ReadOl охоплює життєвий цикл роботи з документом: від його вибору у файловій системі пристрою до відображення сторінок на екрані та фіксації прогресу читання. На відміну від застарілих систем, які вимагали повного сканування пам'яті пристрою та отримання небезпечних системних дозволів на доступ до файлової системи, у ReadOl цей процес реалізовано через системний Android–компонент Storage Access Framework (SAF).

Процес діяльності починається з ініціації користувачем імпорту нового документа. Застосунок активує SAF Picker, який є ізольованим системним середовищем. Після вибору файлу користувачем програма отримує довгострокові права доступу до URI цього документа. Модуль імпорту BookImporter зчитує метадані документа (автора, назву, розмір, тип файлу) та передає об'єкт класу Book для збереження в базу даних Room.

Коли користувач обирає книгу для читання, застосунок створює відповідний обробник (Engine) через фабрику двигунів BookEngineFactory. Для файлів великого переліку форматів, включаючи PDF, DOCX, DOC, RTF, FB2, CBZ, CBR, TXT, MD, MOBI та AZW3, створюється екземпляр класу MuPdfEngine, який взаємодіє з нативною C++ бібліотекою MuPDF. Це дозволяє здійснювати рендеринг у двох режимах: оригінальному (Original Layout із рендерингом у Bitmap) та режимі перекомпонування тексту (Reflow Mode), за якого здійснюється динамічний перерахунок сторінок через метод document.layout відповідно до налаштувань шрифту. Для файлів формату EPUB запускається EpubEngine, який виконує парсинг структури книги та розбиває XHTML–вміст на сторінки за допомогою вбудованого Compose інструменту TextMeasurer.

Для наочного відображення взаємодії та мінімізації зайвої деталізації на схемі, послідовність процесів та викликів функцій структуровано у три головні логічні блоки на UML–діаграмі діяльності (рис. 1.1).

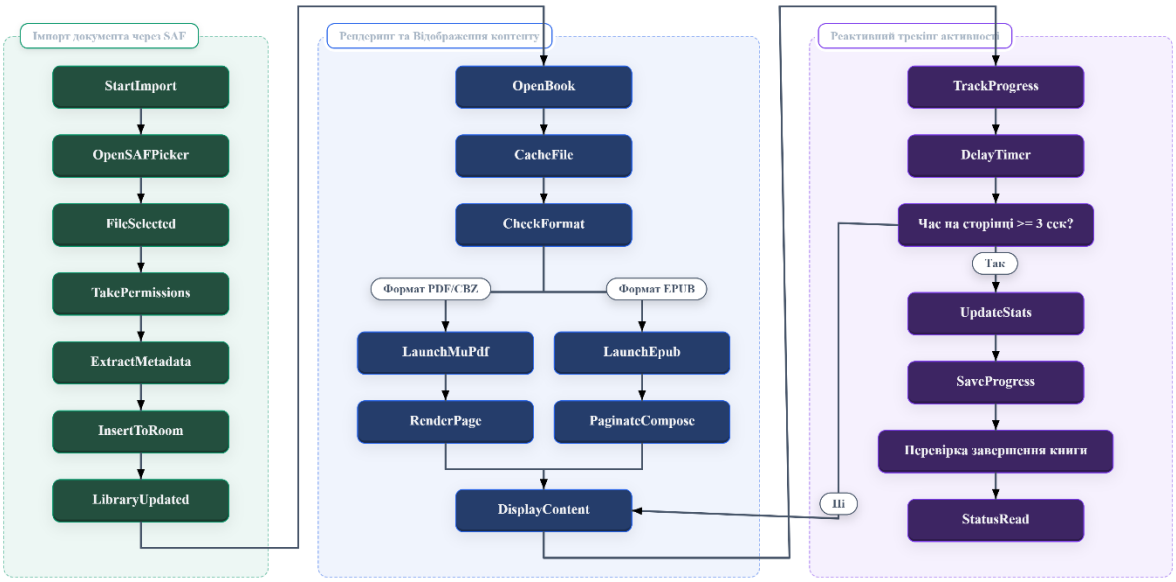

Рисунок 1.1 – UML–діаграма діяльності процесу імпорту, рендерингу та відстеження читання

Особливе місце в процесі діяльності посідає механізм реактивного збереження прогресу. Для запобігання зайвим операціям запису в базу даних при швидкому перегортанні сторінок, у системі реалізовано таймер затримки (debounce) тривалістю 3 секунди. Тільки після перебування користувача на сторінці протягом цього часу запускається асинхронний ланцюжок оновлення статистики читання. Дані про прочитані слова та час сесії записуються в таблицю reading_stats, а в таблиці books оновлюється поле currentPage.

1.1.2 Опис функціональної моделі

Функціональна модель застосунку ReadOl описує взаємодію між користувачем та системою через набір прецедентів використання. У системі виділено дві основні ролі користувачів, які мають різний рівень доступу до

функцій програми: Авторизований користувач (Authorized User) та Гість (Guest).

Гість має доступ до базового функціоналу читання документів, який працює повністю локально на пристрої. Він може імпортувати книги через SAF, налаштовувати параметри відображення тексту (розмір шрифту, колір фону), створювати папки для книг, вести базу структурованих нотаток із можливістю зіркового маркування та вибору стилів підкреслення, а також працювати з інструментами доступності (лінійка читання RulerOverlay для концентрації уваги) та каскадним перекладом (хмарний Lingva API, Google Translate або офлайн–модель ML Kit) та озвученням тексту (TTS із динамічним підсвічуванням слів).

Авторизований користувач, крім базового функціоналу, отримує доступ до хмарних сервісів програми, які реалізовано через інтеграцію з Firebase. Це дозволяє йому синхронізувати свою бібліотеку, історію читання, створені нотатки та прогрес мотиваційної системи з хмарним сховищем Cloud Firestore. Секвенція синхронізації запускається автоматично при вході в профіль або при локальній модифікації даних Room, що реалізується асинхронно у фоні через WorkManager.

Функціональну модель, спроєктовану за горизонтальним принципом для кращої візуальної компактності, відображено на UML–діаграмі використання (рис. 1.2).

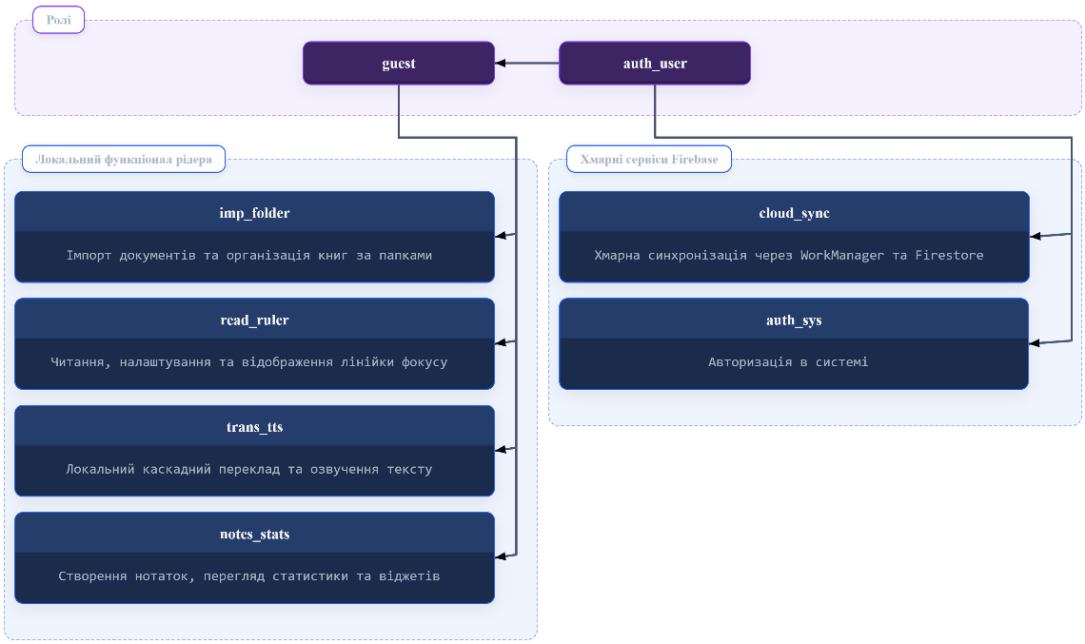

Рисунок 1.2 – UML–діаграма використання функціональної моделі застосунку ReadOl

Кожен прецедент використання у функціональній моделі має чітко визначену логіку роботи:

Прецедент "Імпорт документа через SAF та організація книг". Користувач ініціює додавання файлу. Система відкриває SAF Picker. Після вибору документа програма перевіряє його формат, зчитує метадані та додає запис у базу даних. Користувачі можуть створювати власні папки та розподіляти туди книги на основі зв'язку багатьох до багатьох.

Прецедент "Читання, налаштування та відображення лінійки фокусу". Користувач відкриває книгу та може налаштувати розмір шрифту й колір фону. Для покращення концентрації (наприклад, для користувачів із дислексією або СДУГ) користувач може активувати накладання горизонтальної лінійки з регулюванням її положення, кольору, товщини та прозорості.

Прецедент "Локальний каскадний переклад та озвучення тексту". Під час читання користувач може виділити слово чи речення для миттєвого

перекладу, який виконується каскадно: спочатку робиться спроба через хмарний Lingva API, у разі його недоступності використовується прямий запис до Google Translate API, а за відсутності мережі − офлайн–моделі Google ML Kit. Оптимізація процесу забезпечується префетч–менеджером, який асинхронно завантажує переклад для сусідніх сторінок (N±1). Також користувач може активувати озвучення тексту вголос із можливістю паузи, регулювання швидкості, автоматичним управлінням фокусом звуку в ОС та динамічним виділенням озвучуваного слова на екрані.

Прецедент "Створення нотаток, перегляд статистики та віджетів". Користувач може створювати нотатки з вибором кольору виділення та стилю підкреслення, переглядати графіки активності та додавати віджети на робочий стіл Android для відображення поточної книги, обраної папки чи графіку щоденної активності.

Прецедент "Хмарна синхронізація даних". Цей прецедент доступний лише авторизованим користувачам. Модуль SyncManager запускає фонову задачу через WorkManager. Локальні оновлення та видалені записи завантажуються у Cloud Firestore з використанням механізму м'якого видалення (soft delete) та технічних полів updatedAt, deviceId, а нові дані з хмари записуються у Room.

1.2 Огляд наявних аналогів

Для визначення конкурентних переваг розроблюваного застосунку ReadOl та обґрунтування необхідності його створення було проведено детальний аналіз трьох найпопулярніших рішень на ринку мобільних читалок для операційної системи Android: ReadEra, Moon+ Reader та PocketBook.

Застосунок ReadEra є одним із лідерів ринку завдяки підтримці великої кількості форматів документів та відсутності нав'язливої реклами. Програма має простий та зрозумілий інтерфейс користувача. Проте її головним

недоліком є відсутність безкоштовної хмарної синхронізації. Користувачі безкоштовної версії не мають можливості синхронізувати свою бібліотеку, історію читання чи нотатки між різними пристроями. Хмарне збереження доступне лише при покупці платної версії ReadEra Premium, що обмежує можливості широкого кола користувачів. Крім того, застосунок не має розвинених інтегрованих інструментів мотивації читання, обмежуючись лише базовим показом відсотка прочитаного, а також вимагає виклику зовнішніх сторонніх програм для перекладу слів.

Застосунок Moon+ Reader має велику кількість налаштувань відображення тексту. Програма підтримує підключення сторонніх мережевих каталогів книг (OPDS) та хмарних сховищ, таких як Dropbox чи Google Drive. Однак інтерфейс програми є застарілим та перевантаженим численними меню, що ускладнює взаємодію для нових користувачів. Безкоштовна версія застосунку містить велику кількість банерної реклами, яка відволікає від процесу читання. Також у Moon+ Reader відсутня власна автоматизована система синхронізації прогресу; користувач змушений вручну налаштовувати підключення до сторонніх хмарних дисків. Озвучення тексту (TTS) працює лише як базове читання вголос без синхронізованої підсвітки слів на екрані.

Застосунок PocketBook розробляється як частина екосистеми електронних книг PocketBook. Програма має інтегрований фірмовий хмарний сервіс PocketBook Cloud, який безкоштовно синхронізує книги та прогрес читання. Проте застосунок має високі вимоги до апаратних ресурсів пристрою. При відкритті великих PDF–документів на смартфонах середнього або бюджетного класу спостерігаються суттєві затримки при рендерингу сторінок. Інтерфейс програми містить багато зайвих елементів, пов'язаних із фірмовим магазином книг, що перевантажує екран та ускладнює навігацію. Інтегровані засоби доступності, такі як лінійка фокусування для користувачів із розладами уваги, у програмі повністю відсутні.

Таблиця 1.1 – Порівняльний аналіз мобільних застосунків для читання документів

Параметр порівняння	ReadEra	Moon+ Reader	PocketBook	ReadOl
Інтерфейс користувача	Classy (XML), простий	Застарілий, перевантажений меню	Сучасний, але перевантажений рекламою магазину	Сучасний декларативний (Jetpack Compose), адаптивний
Робота з папками та віджетами	Прості папки, базовий віджет	Складне управління, відсутність окремих віджетів	Тільки колекції фірмового магазину	Організація за папками (M:N), віджети прогресу, папок та статистики
Локальний каскадний переклад	Відсутній (викликає сторонні словники)	Відсутній	Відсутній	Вбудований безкоштовний (Lingva/ML Kit) з офлайн–режимом та кешуванням
Хмарна синхронізація	Тільки у платній версії Premium	Ручна через сторонні хмари (Dropbox)	Безкоштовна фірмова (PocketBook Cloud)	Безкоштовна автоматична (Firebase Firestore)
Вбудовані інструменти мотивації	Відсутні (тільки базовий відсоток прогресу)	Відсутні	Відсутні	Наявні (щоденні цілі сторінок, облік серій читання)
Продуктивність рендерингу PDF	Середня	Низька на важких файлах	Низька на бюджетних пристроях	Висока (нативна бібліотека MuPDF fitz)
Робота з нотатками та закладками	Прості текстові нотатки	Текстові нотатки та маркування	Створення нотаток та малювання пальцем	Розширене виділення тексту, вибір стилів та кольорів нотаток

Продовження таб. 1.1

Реклама в інтерфейсі	Відсутня	Наявна у безкоштовній версії	Наявна (просування власного магазину)	Повністю відсутня

Аналіз табл. 1.1 показує, що жодне з існуючих популярних рішень на ринку не надає користувачам безкоштовної, автоматичної хмарної синхронізації в поєднанні з сучасним декларативним інтерфейсом користувача без реклами, вбудованим каскадним перекладачем, та інтегрованою системою мотивації. Це підтверджує актуальність розробки застосунку ReadOl для відкритого B2C–ринку, який покликаний закрити цю нішу та надати користувачам зручний, швидкий та функціональний інструмент для щоденного читання.

1.3 Постановка задачі

Основною метою переддипломної практики є проєктування та програмна реалізація нативного мобільного застосунку для операційної системи Android, призначеного для читання електронних книг та аналізу читацької активності під назвою ReadOl. До програмний продукт висувається вимога щодо забезпечення коректного та високопродуктивного відображення документів різних типів розмітки, зокрема фіксованого макету (формат PDF) та адаптивного тексту (формат EPUB), надаючи користувачеві розширені інструменти для індивідуального налаштування візуального інтерфейсу, ведення нотаток та відстеження аналітики.

Під час проєктування архітектури застосунку необхідно врахувати комплекс нефункціональних вимог та технічних обмежень, що висуваються до сучасного мобільного програмного забезпечення. Сумісність із платформою Android передбачає підтримку пристроїв, починаючи з версії мінімального комплекту розробки API 24 (Android 7.0), та повну орієнтацію на цільовий

рівень API 36. Одним із головних інженерних викликів є оптимізація споживання оперативної пам’яті мобільного пристрою під час рендерингу графічних файлів та документів великого обсягу. Для забезпечення стабільної частоти оновлення екрана підсистему відображення фіксованих форматів слід реалізувати на базі низькорівневих нативних бібліотек, написаних мовою C++, взаємодія з якими здійснюється через механізм Java Native Interface (JNI).

Окремою архітектурною вимогою до програмного продукту є забезпечення високого рівня автономності. Застосунок повинен зберігати повноцінну працездатність в умовах повної відсутності підключення до мережі Інтернет, що передбачає реалізацію офлайн–режиму для читання завантажених книг, локального пошуку по тексту, роботи з внутрішніми словниками та швидкого перекладу виділених фрагментів контенту. Окрім цього, безпека опрацювання вхідних потоків даних вимагає впровадження алгоритмів стійкості до вразливостей файлової системи. Зокрема, під час розпакування та парсингу стиснутих електронних документів у форматі EPUB система повинна виконувати автоматичну перевірку канонічних шляхів файлів для нейтралізації загрози обходу каталогу (Zip Slip) та запобігання несанкціонованому виходу за межі виділеного ізольованого каталогу додатка.

Для досягнення поставленої мети в межах інженерної розробки необхідно вирішити такі основні завдання:

- спроєктувати реляційну структуру локального збереження даних на основі вбудованої системи керування базами даних Room для надійного обліку метаданих книг, користувацьких папок, структурованих анотацій, цитат та журналів читацьких сесій із забезпеченням цілісності зв'язків;
- розробити архітектурне рішення фонової синхронізації даних за шаблоном Offline–First, яке дозволить виконувати двосторонній асинхронний обмін інформацією між локальним сховищем смартфона та хмарною NoSQL базою даних, мінімізуючи мережевий трафік та автоматично вирішуючи конфлікти версій;

- математично обґрунтувати та реалізувати обчислювальні алгоритми внутрішньої системи мотивації читача, що охоплюють розрахунок поточної швидкості читання у словах за хвилину (WPM) та сторінках за годину, прогнозування дати завершення книги, а також фіксацію щоденного прогресу і ведення безперервних серій активності користувача.

Результатом виконання поставлених завдань має стати оптимізований, масштабований мобільний застосунок із чітким розподілом шарів бізнес–логіки та представлення даних, який не містить сторонніх рекламних модулів і надає користувачам можливість автоматичного безкоштовного хмарного збереження стану бібліотеки.

Висновки до розділу

У першому розділі кваліфікаційної роботи було проведено детальний аналіз предметного середовища мобільних застосунків для читання електронних документів. Було встановлено, що сучасний відкритий B2C–ринок потребує програмного забезпечення, яке б поєднувало швидку роботу з різними форматами файлів (особливо з важкими графічними PDF–документами) та надійні інструменти збереження даних без нав'язливої реклами чи платних обмежень.

На основі аналізу останніх наукових публікацій та досліджень було обґрунтовано вибір нативного стеку розробки для ОС Android з використанням мови Kotlin [6]. Було визначено, що архітектура застосунку має базуватися на шаблоні MVVM, що забезпечує високі показники продуктивності, надійності та модифікованості системи. Використання сучасного декларативного фреймворку Jetpack Compose дозволяє створювати плавні та адаптивні інтерфейси користувача, які миттєво реагують на зміни стану даних.

Було детально описано процеси діяльності програми, пов'язані з імпортом документів через системний компонент Storage Access Framework, посторінковим рендерингом контенту за допомогою нативної C++ бібліотеки MuPDF fitz та реактивним відстеженням прогресу читання. Ці процеси було формалізовано та візуалізовано за допомогою детальної UML–діаграми діяльності.

Також було спроєктовано функціональну модель застосунку, представлену у вигляді UML–діаграми використання. Було детально описано сценарії взаємодії для двох ролей користувачів: Гостя, який працює повністю локально, та Авторизованого користувача, який отримує доступ до автоматичної хмарної синхронізації даних через інтеграцію з сервісами Firebase Cloud Firestore.

Проведений порівняльний аналіз трьох найпопулярніших аналогів на ринку (ReadEra, Moon+ Reader, PocketBook) виявив їхні суттєві архітектурні та функціональні недоліки [7], такі як відсутність безкоштовної хмарної синхронізації, застарілий та перевантажений рекламою інтерфейс, високі вимоги до апаратних ресурсів пристрою при рендерингу важких документів. Це дозволило чітко сформулювати конкурентні переваги розроблюваного застосунку ReadOl та поставити завдання на проектування та реалізацію його компонентів.

РОЗДІЛ 2
ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз предметної області

Сучасне предметне середовище засобів цифрового читання та опрацювання документів характеризується стрімким збільшенням обсягів інформації та підвищенням вимог до портативних пристроїв відображення контенту. Сучасний мобільний рідер має не лише виконувати функцію відображення текстових та графічних файлів, а й забезпечувати високий рівень когнітивного комфорту користувача, надавати інструменти для структурування інформації, ведення нотаток та детальної аналітики. Розробка ефективної системи потребує глибокого вивчення поведінкових чинників читача, розрахунку швидкості сприйняття тексту та побудови локального сховища за принципом автономності (Offline–First) з можливістю безконфліктної синхронізації з хмарою.

У процесі дослідження предметної області застосунку ReadOl було виділено базові інформаційні об'єкти та спроєктовано логічну схему їх взаємодії. Це дозволяє визначити межі проєктування системи, встановити правила обробки даних та забезпечити цілісність інформації на всіх рівнях архітектури. Інформаційну модель взаємозв'язку сутностей предметної області представлено на рисунку 2.1 у вигляді концептуального графа.

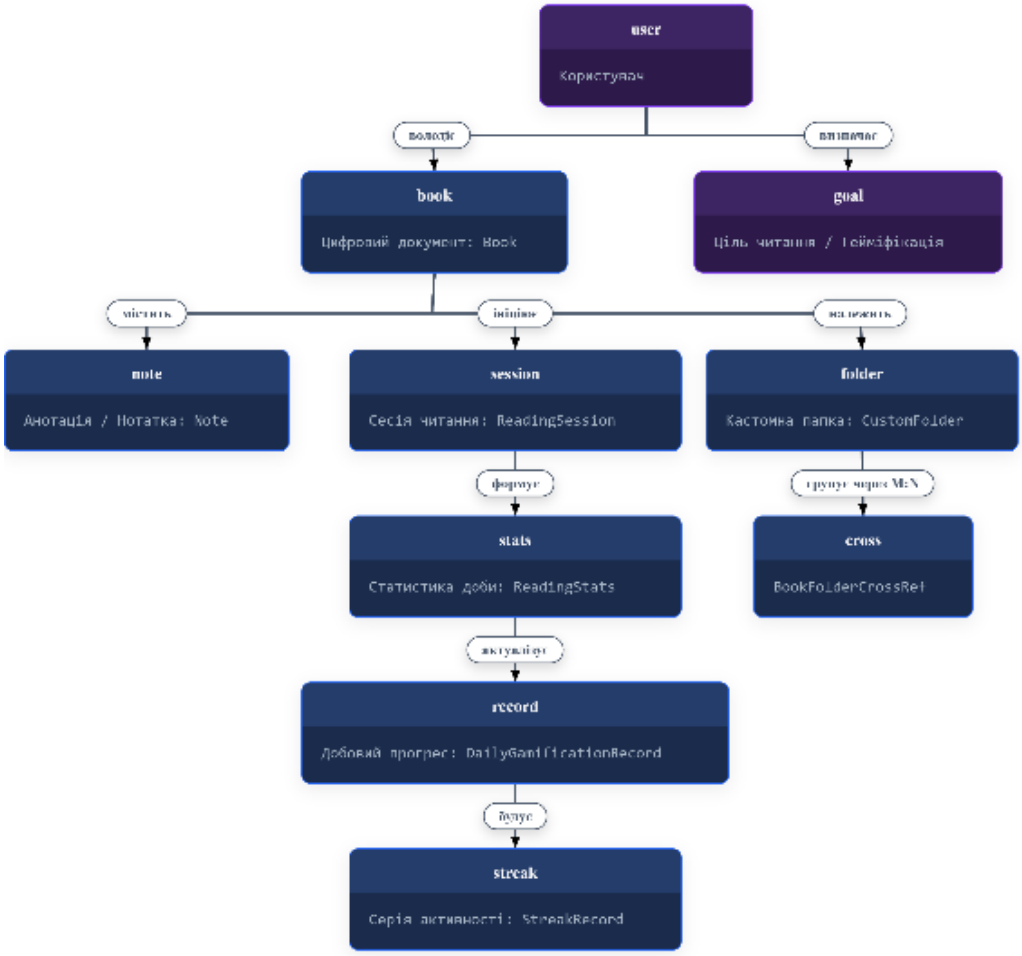

Рисунок 2.1– UML–діаграма взаємозв’язок сутностей в застосунку ReadOl

На рисунку 2.1 наведено інформаційну модель предметної області застосунку ReadOl. Схема ілюструє взаємодію між користувачем, цифровим документом та супутніми аналітичними й гейміфікаційними сутностями. Ключовим елементом моделі виступає книга, з якою безпосередньо пов'язані нотатки, сесії читання, папки та добова статистика активності, що в сукупності забезпечує повноцінне відстеження прогресу користувача.

2.1.1 Вхідні дані

Функціонування мобільної системи ReadOl базується на отриманні, валідації та логічній обробці вхідних даних кількох категорій:

1. Цифрові носії текстового та графічного контенту користувача. Застосунок забезпечує підтримку широкого переліку форматів документів, які класифікуються за способом розмітки на три групи:

- документи з фіксованою верстkою (Fixed Layout), де геометричне розташування елементів є незмінним (основний представник – формат PDF, а також графічний формат CBZ);
- адаптивні формати документів (Reflowable), які підтримують динамічне перекомпонування тексту під розміри екрана (формати EPUB, FB2, MOBI, чисті текстові файли TXT та розмічені документи Markdown);
- формати офісних документів, що використовуються для обміну текстовою інформацією зі складною стилізацією та потребують попереднього парсингу (DOCX, DOC, RTF).

2. Супутні метадані файлів, які зчитуються модулем імпорту та заносяться в базу даних (назва книги, ім'я автора, розмір файлу у байтах FileSize, повна назва FileName на пристрої, а також обчислені показники кількості слів і символів у тексті).

3. Вхідні дані взаємодії користувача з інтерфейсом пристрою під час читання (координати точок натискання на екран для виділення тексту, створення нотаток чи визначення жестів, текстові коментарі користувача до анотацій, а також конфігураційні параметри відображення: розмір шрифту, міжрядковий інтервал тощо).

4. Хронометричні показники, що фіксуються системним таймером автоматично (час початку й завершення сесії читання, а також поточний індекс прочитаної сторінки).

5. Параметри гейміфікаційних цілей, які користувач задає вручну на екрані налаштувань (щоденна ціль читання у сторінках dailyPageGoal або хвилинах dailyTimeGoalMin).

Аналіз вхідних параметрів дозволяє зробити висновок, що вхідні потоки даних мають різну фізичну природу (структуровані файли метаданих, двійкові потоки документів, асинхронні події дотиків користувача та часові позначки), що вимагає розробки уніфікованих системних інтерфейсів для їхньої попередньої обробки та валідації на системному рівні.

2.1.2 Вихідні дані

Вихідні дані мобільного застосунку ReadOl є результатами обробки вхідних параметрів та діляться на візуальні структури інтерфейсу, звукові потоки, аналітичні метрики й дані віджетів.

Візуальні вихідні дані формуються залежно від оброблюваного формату. Для графічних документів (PDF, CBZ) вихідними даними є посторінкові растрові зображення (Bitmap), згенеровані під конкретну ширину екрана пристрою. Для текстових форматів (EPUB, FB2, TXT) вихідними даними є об'єкти стилізованого тексту Android AnnotatedString, які підтримують динамічне форматування та колірне маркування [8].

Математично процеси рендерингу та перетворення вихідних даних у системі описуються такими операторами:

Оператор генерації растрового зображення для фіксованих форматів R_{bitmap}:

$$R_{bitmap} : (P_{index}, W_{display}) \rightarrow B_{raster} \quad (2.1)$$

де P_{index} – порядковий індекс сторінки;

$W_{display}$ – ширина області відображення на екрані пристрою в пікселях;

B_{raster} – результуючий об'єкт растрового зображення Bitmap.

Оператор структурного парсингу XHTML–вмісту в стилізований текст $P_{compose}$:

$$P_{compose} : S_{xhtml} \rightarrow A_{styled} \quad (2.2)$$

де S_{xhtml} – вхідний HTML–рядок файлу адаптивної книги;

A_{styled} – вихідний об'єкт стилізованого тексту AnnotatedString для Jetpack Compose.

Окремим вихідним потоком є аудіосигнал при використанні функції синтезу мовлення (*Text–to–Speech*). Синхронізація відтворення звуку та графічного відображення забезпечується за допомогою передачі поточного діапазону озвучуваних символів у текстове представлення екрана для динамічного виділення кольором у реальному часі. При цьому система фіксує числові межі поточного слова в тексті для забезпечення плавності візуального супроводу.

Трансляція вихідних даних на робочий стіл Android виконується за допомогою віджетів. Для відображення обкладинок книг, назв папок та графіків активності на робочому столі використовуються оптимізовані графічні структури, що містять відмасштабовані та стиснуті під розміри віджетів версії зображень для уникнення надлишкового навантаження на пам'ять пристрою.

Статистичні вихідні дані охоплюють сформовані записи про читацьку активність, що зберігаються у вигляді добових гейміфікаційних ревю та сесій читання. Для забезпечення точності обліку цих даних у системі передбачено контроль узагальненого стану активності інтерфейсу, що дозволяє призупиняти розрахунок активного часу у випадках, коли користувач взаємодіє з допоміжними діалогами чи налаштуваннями.

2.2 Проектування системи

Концептуальне проєктування мобільної системи ReadOl є перехідним етапом від аналізу абстрактних інформаційних потоків до побудови фізичних моделей даних та програмних інтерфейсів. Метою проєктування є створення стійкої, слабкопов'язаної архітектури, яка б гарантувала швидку роботу модулів за умов обмеженості апаратних ресурсів смартфонів.

Процес проєктування системи умовно поділяється на дві основні складові:

- Проєктування реляційної схеми локальної бази даних Room v20 для забезпечення персистентного збереження прогресу, нотаток та статистики користувача на пристрої [9].
- Побудова об'єктно–орієнтованої моделі класів модуля відображення (Reader), яка б реалізувала принципи Clean Architecture та шаблону MVVM для ізоляції бізнес–логіки від особливостей рендерингу конкретних файлів [10].

Для наочного представлення структури збереження інформації та взаємодії програмних об'єктів розроблено деталізовані ER та UML діаграми, які описують фізичну реалізацію розробленого програмного забезпечення.

2.2.1 Проектування бази даних

Локальне збереження інформації у застосунку ReadOl побудовано за концепцією Offline–First на основі об'єктно–реляційного відображення Room ORM, яка автоматично генерує код для взаємодії з реляційною базу даних SQLite на пристрої [11]. База даних версії v20 складається з 8 пов'язаних таблиць. Логічна схема зв'язків та структура таблиць локальної бази даних відображена на рисунку 2.2 у вигляді ER–діаграми.

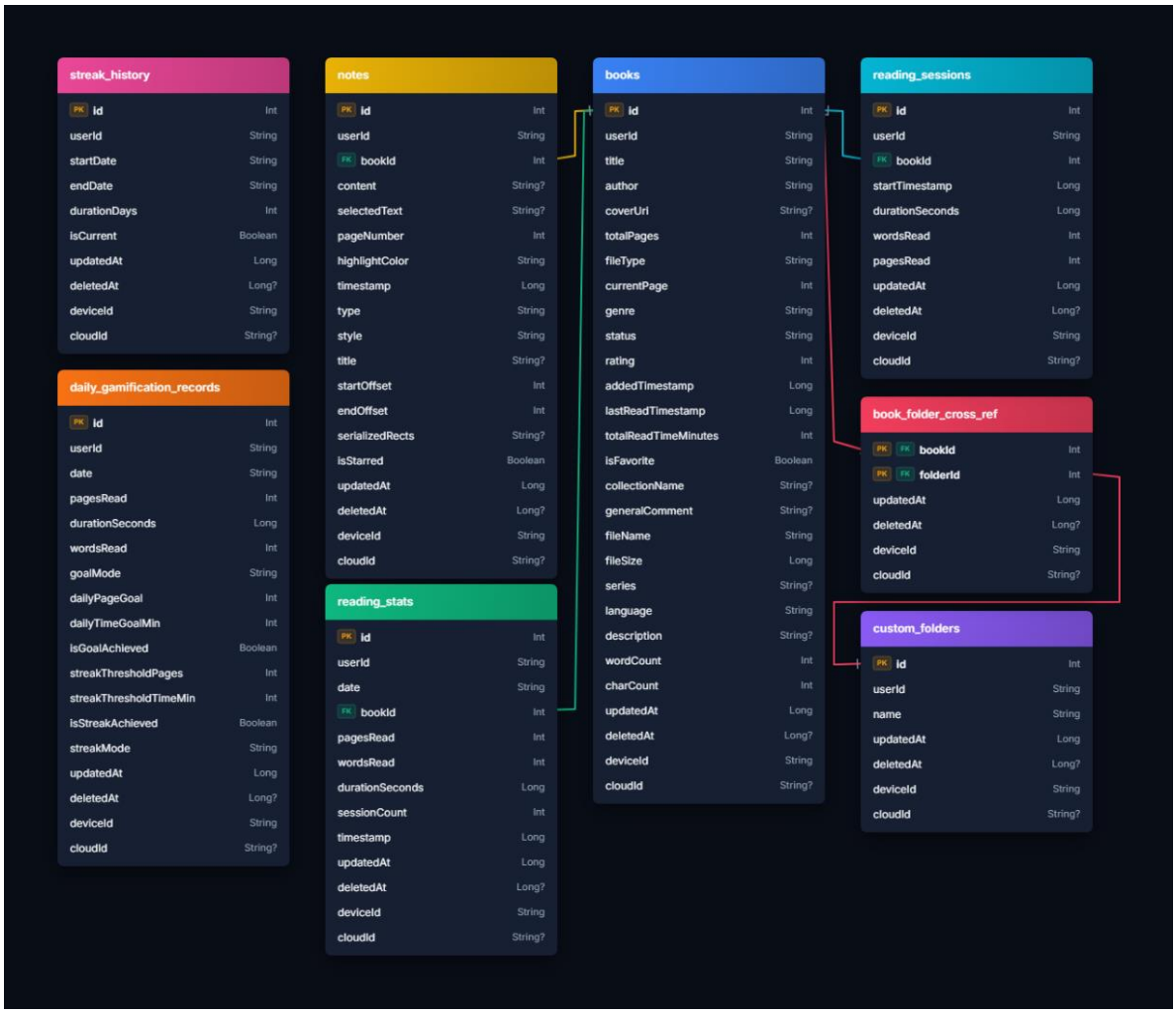

Рисунок 2.2 – ER–модель бази даних застосункуReadOl

На рисунку 2.2 наведено логічну схему бази даних. Головною сутністю є таблиця books, з якою пов'язані таблиці notes, reading_sessions та reading_stats через зовнішні ключі bookId з дією каскадного видалення ("ON DELETE CASCADE"). Це дозволяє автоматично очищати базу даних від пов'язаних нотаток, сесій та статистики після видалення книги, що запобігає накопиченню сміттєвих записів. Зв'язок M:N між книгами та папками реалізовано через проміжну таблицю book_folder_cross_ref.

Фізична структура таблиць розробленої бази даних Room v20 описується призначенням та структурою таких восьми таблиць:

1. Таблиця books зберігає метадані імпортованих документів. Вона містить первинний ключ id з автогенерацією, ідентифікатор користувача userId, назву title, автора author, локальний шлях до файлу обкладинки coverUri та загальну кількість сторінок totalPages. Також у ній зберігається поточний прогрес читання (поточна сторінка currentPage, статус книги status), загальний час читання в хвилинах totalReadTimeMinutes та оцінка користувача rating. Додатково таблиця містить характеристики файлу (fileSize, fileType, fileName), мову документа language та підраховану кількість слів і символів (wordCount, charCount) для обчислення швидкості читання.

2. Таблиця notes зберігає додані користувачем коментарі, закладки та виділені цитати з книг. Вона пов'язана з таблицею книг через зовнішній ключ bookId. Таблиця містить вміст коментаря content, виділений оригінальний текст книги selectedText, номер сторінки pageNumber, колір виділення highlightColor та стиль лінії підкреслення style. Для відображення виділеного тексту на сторінках PDF–документів у полі serializedRects записується серіалізований JSON–масив координат прямокутників виділення, а для текстових форматів використовуються індекси зсуву startOffset та endOffset.

3. Таблиця reading_stats фіксує результати читацької активності користувача по кожній окремій книзі за календарну добу. Вона пов'язана з книгою через зовнішній ключ bookId. Запис містить дату у форматі ISO 8601, кількість прочитаних за день сторінок pagesRead, кількість слів wordsRead, загальну тривалість сесій у секундах durationSeconds та лічильник сесій sessionCount.

4. Таблиця streak_history забезпечує роботу мотиваційної складової застосунку. Вона фіксує серії безперервного читання користувача. Таблиця зберігає дату початку серії startDate, дату її завершення endDate, тривалість

серії у днях durationDays та логічний прапорець isCurrent, який вказує на статус активності серії.

5. Таблиця custom_folders призначена для створення користувачем власної ієрархії групування документів. Вона містить унікальний первинний ключ id та текстове поле name для збереження назви папки.

6. Таблиця book_folder_cross_ref є таблицею зв'язку, що реалізує відношення багато–до–багатьох між сутностями books та custom_folders. Вона містить складений первинний ключ, утворений з пари зовнішніх ключів bookId та folderId.

7. Таблиця daily_gamification_records накопичує щоденний прогрес користувача по всіх книгах разом для підтримки гейміфікаційних механізмів. Записи групуються за складеним унікальним індексом (userId, date). Таблиця містить встановлені добові цілі (dailyPageGoal, dailyTimeGoalMin) та режими їхнього обчислення goalMode, фактично досягнуті показники за день (pagesRead, wordsRead, durationSeconds), а також логічні прапорці виконання цілей (isGoalAchieved, isStreakAchieved).

8. Таблиця reading_sessions логує кожен окремий сеанс читання користувача для проведення аналізу швидкості та зміни сприйняття тексту. Вона містить час початку сесії startTimestamp, її тривалість в секундах durationSeconds, а також кількість прочитаних за цей сеанс слів wordsRead та сторінок pagesRead.

Кожна таблиця бази даних містить набір спільних технічних полів updatedAt, deletedAt, deviceId та cloudId. Ці поля забезпечують реплікацію даних із хмарою Cloud Firestore без виникнення конфліктів та з підтримкою м'якого видалення записів. Детальна фізична структура розробленої бази даних із зазначенням типу даних кожного атрибута наведена у додатку А (див. додаток А).

2.2.2 Побудова об’єктно–орієнтованої моделі

Для спрощення масштабованості та супроводження коду проєкту, архітектура модуля відображення та читання документів (Reader) побудована відповідно до принципів Clean Architecture з чітким розподілом відповідальності на 15 взаємопов'язаних файлів [12]. Побудва системи базується на декларативній парадигмі «UI–as–State», де користувацький інтерфейс виступає детермінованою функцією єдиного immutable–стану.

Схема об'єктно–орієнтованої взаємодії та зв'язків між ключовими класами модуля читання документів відображена на UML–діаграмі класів (рисунок 2.3).

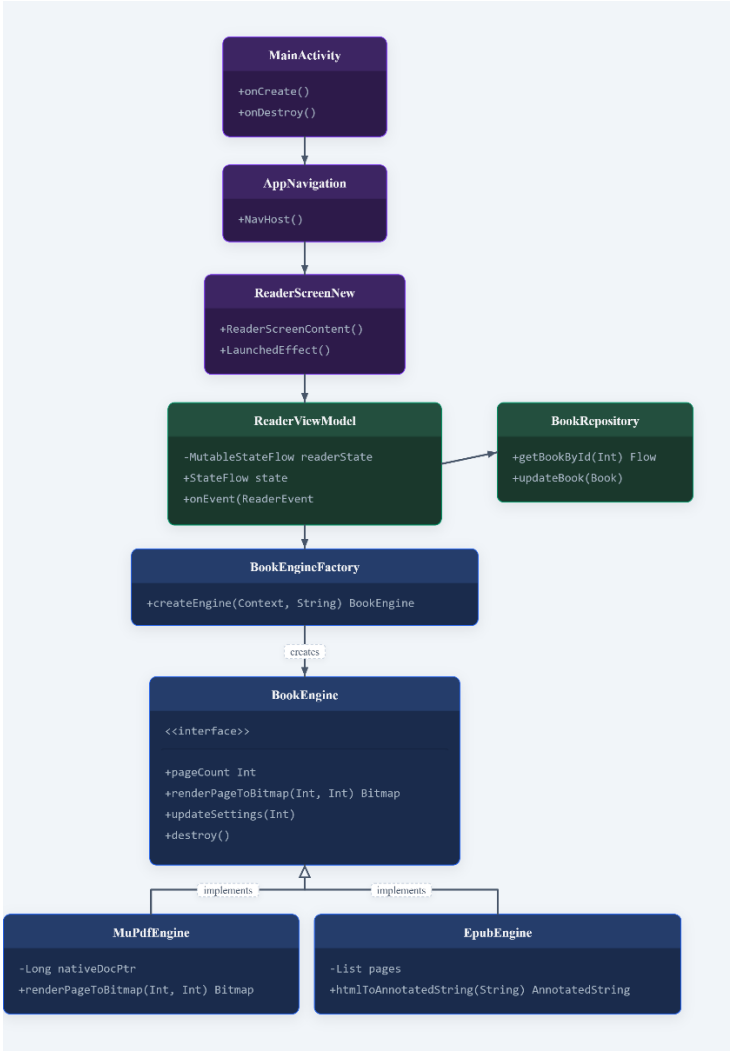

Рисунок 2.3 – Діаграма класів застосунку ReadOl

Діаграма класів (рис. 2.3) демонструє три архітектурні шари системи: шар даних (BookRepository), шар движків обробки документів (BookEngine, MuPdfEngine, EpubEngine) та шар представлення (MainActivity, ReaderScreenNew, ReaderViewModel). Вона наочно показує, як екрани Compose взаємодіють з ViewModels, що своєю чергою отримують доступ до сховища та фабрик движнів для ініціалізації потрібних систем рендерингу на основі розширення відкритих файлів.

Презентаційний шар (*View*) представлений класом ReaderScreenNew.kt, який підписується на потоки стану ReaderViewModel.kt. Логічний шар інтерфейсу повністю ізольований від рушіїв рендерингу за допомогою інтерфейсу–абстракції BookEngine. Цей інтерфейс визначає загальний набір методів для взаємодії з книгами будь–якого формату, включаючи рендеринг сторінок у Bitmap, виділення слів за координатами, пошук текстових збігів та ручну утилізацію нативних C++ ресурсів у деструкторах.

Фабрика BookEngineFactory на етапі запуску сесії читання анілізує MIME–тип або розширення документа та повертає відповідну реалізацію – нативний двигун MuPdfEngine для PDF та графічних форматів або кастомний EpubEngine для пагінації тексту.

2.3 Математичне та алгоритмічне забезпечення

Математичне та алгоритмічне забезпечення визначає формалізовану логіку основних аналітичних та обчислювальних функцій застосунку ReadOl. Воно охоплює розрахунки швидкості читання, прогнозування часу закінчення книги, афінні перетворення растрових сторінок та динамічну пагінацію неструктурованого контенту. Усі ці процеси об'єднуються у єдину логіко–математичну модель, що забезпечує користувача актуальними метриками.

Первинною ланкою обчислювальної системи є алгоритм розрахунку швидкості читання під час сесії. Він зчитує хронометричні показники та обсяг

опрацьованого тексту. Математично швидкість читання у словах за хвилину (WPM) та сторінках за годину (PPH) описується так:

$$WPM = \frac{\Delta W}{T_{sec}/60} \tag{2.3}$$

де ΔW – кількість прочитаних слів за поточну сесію читання;

T_{sec} – тривалість сесії читання в секундах.

Якщо точна кількість прочитаних слів ΔW не може бути порахована (наприклад, у PDF), вона апроксимується через середній обсяг слів на сторінках книги:

$$WPM = \frac{W_{total} \times \Delta P}{P_{total} \times (T_{sec}/60)} \tag{2.4}$$

де T_{total} – загальна кількість слів у книзі;

ΔP – кількість прочитаних сторінок за сесію;

P_{total} – загальна кількість сторінок у книзі;

T_{sec} – тривалість сесії читання в секундах.

Розрахунок темпу читання в сторінках за годину (PPH) здійснюється за таким математичним співвідношенням:

$$PPH = \frac{\Delta P}{T_{sec}/3600} \tag{2.5}$$

де ΔP – кількість прочитаних сторінок за сесію;

T_{sec} – тривалість сесії читання в секундах.

Для запобігання діленню на нуль при надто коротких сесіях ($T_{sec} < 5$ секунд), алгоритм блокує розрахунок метрики, примусово встановлюючи значення $\Delta WPM = 0$ та $\Delta PPH = 0$.

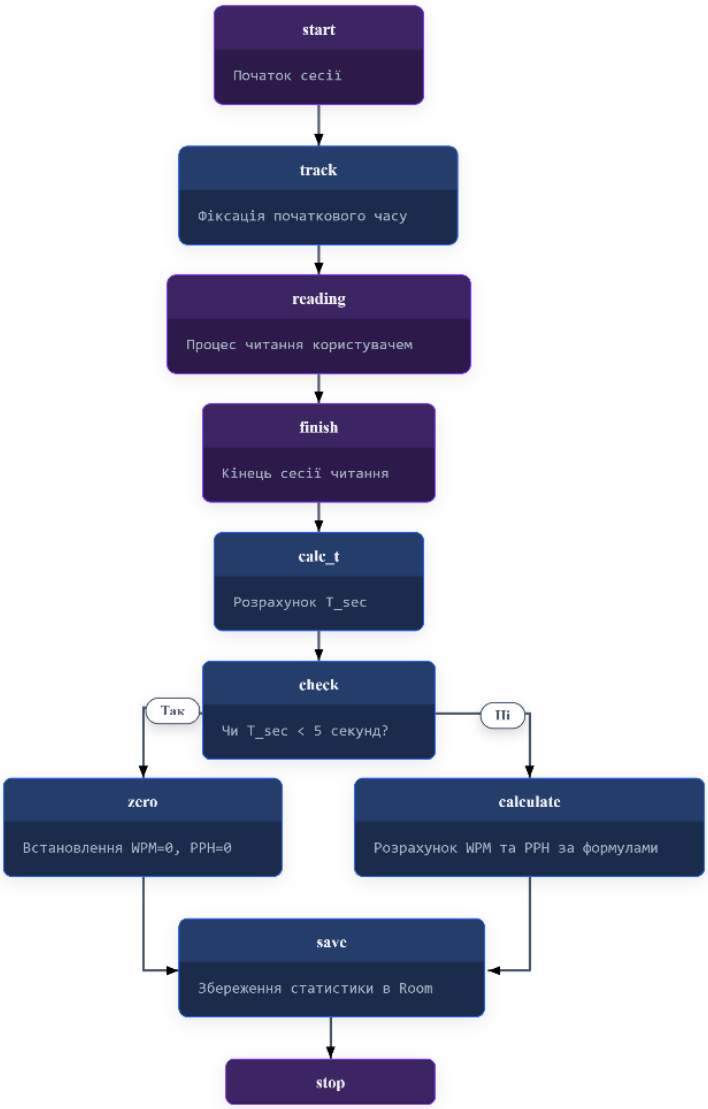

Рисунок 2.4 – Алгоритм послідовності збереження результатів від початку сесії ReadOl

Блок–схема алгоритму (рис. 2.4) ілюструє послідовність кроків від початку сесії до збереження результатів у базу даних Room. Вона наочно демонструє механізм фільтрації занадто коротких сесій тривалістю менше п'яти секунд, що дозволяє уникнути виникнення похибок ділення на нуль та гарантує високу точність фіксованих аналітичних метрик.

Розраховані показники темпу читання є безпосереднім вхідним ресурсом для наступного логічного кроку – прогнозування часу завершення книги. Для цього спочатку обчислюється середня швидкість читання за останні N сесій користувача (PPH_{avg}):

$$PPH_{avg} = \frac{\sum_{i=1}^{N} \Delta P_i}{\sum_{i=1}^{N} T_{sec,i} / 3600} \quad (2.6)$$

де ΔP_i – кількість прочитаних сторінок у і–й сесії;

$T_{sec,i}$ – тривалість і–ї сесії читання в секундах;

N – загальна кількість аналізованих сесій.

Далі визначається залишок сторінок $P_{rem} = P_{total} - P_{current}$, що дає змогу вирахувати очікуваний залишок часу читання у годинах ($T_{remaining}$):

$$T_{remaining} = \frac{P_{total} - P_{current}}{PPH_{avg}} \quad (2.7)$$

де P_{total} – загальна кількість сторінок у книзі;

$P_{current}$ – індекс поточної сторінки прогресу;

PPH_{avg} – середня швидкість читання в сторінках за годину.

На основі цього показника здійснюється розрахунок прогнозованої дати повного прочитання книги ($Date_{finish}$):

$$Date_{finish} = Date_{now} + T_{remaining} \times 3600 \times 1000 \quad (2.8)$$

де $Date_{now}$ – поточний системний час у мілісекундах;

$T_{remaining}$ – очікуваний залишок часу читання у годинах.

Рисунок 2.5 – Схема алгоритму прогнозування завершення часу у застосунку ReadOl

На рисунку 2.5 наведено схему алгоритму прогнозування часу завершення читання книги. Алгоритм ітераційно зчитує історію активності користувача з локальної бази даних, вираховує середній показник темпу читання за останні сесії та динамічно коригує прогнозну дату завершення відповідно до залишку нечитаних сторінок.

Поряд із обчисленням аналітичних показників прогресу, система має вирішувати геометричні задачі рендерингу та обробки дотиків користувача під час взаємодії з графічними сторінками. Для документів із фіксованим макетом

(PDF) на базі MuPDF застосовується алгоритм масштабування сторінки та зворотної трансляції координат дотику. Коефіцієнт масштабування сторінки ($scale$) обчислюється як відношення ширини екрана пристрою ($W_{display}$) до фізичної ширини сторінки за координатами документа ($bounds.x_1 - bounds.x_0$):

$$scale = \frac{W_{display}}{bounds.x_1 - bounds.x_0} \quad (2.9)$$

де $W_{display}$ – ширина області відображення на екрані пристрою в пікселях;

$bounds.x_1$ – права межа фізичної сторінки документа;

$ounds.x_0$ – ліва межа фізичної сторінки документа.

Відповідна матриця афінного перетворення CTM (Current Transformation Matrix) для рендерингу сторінки в Bitmap має вигляд:

$$CTM = \begin{pmatrix} scale & 0 \\ 0 & scale \end{pmatrix} \quad (2.10)$$

де $scale$ – обчислений коефіцієнт масштабування сторінки.

При виділенні тексту чи створенні нотаток координати натискання на екран (X_{screen}, Y_{screen}) транслюються у координати документа (X_{doc}, Y_{doc}) за формулами зворотної трансляції:

$$X_{doc} = \frac{X_{screen}}{scale} + bounds.x_0 \quad (2.11)$$

де X_{screen} – координата натискання по осі абсцис на екрані пристрою;

$scale$ – обчислений коефіцієнт масштабування сторінки;

$bounds.x_0$ – початкове горизонтальне зміщення сторінки документа.

$$Y_{doc} = \frac{Y_{screen}}{scale} + bounds.y_0 \quad (2.12)$$

де Y_{screen} – координата натискання по осі ординат на екрані пристрою;

$scale$ – обчислений коефіцієнт масштабування сторінки;

$bounds.y_0$ – початкове вертикальне зміщення сторінки документа.

Це дозволяє точно зіставити жест користувача на екрані з нативними текстовими об'єктами PDF–документа.

Рисунок 2.6 – Трансляція координат сторінки графічного документу у застосунку ReadOl

Масштабування та зворотна трансляція координат сторінки графічного документа PDF відображена на рисунку 2.6 у вигляді структурної схеми. Вона показує двосторонній зв'язок між екранними координатами дотику та координатною сіткою нативного документа через афінну матрицю перетворення, що дозволяє виконувати точне виділення та підсвічування обраного контенту.

Якщо для статичних PDF–файлів корисними є афінні перетворення растрових координат, то робота з адаптивними текстовими форматами типу EPUB вимагає повністю динамічного підходу до розподілу контенту. Пагінація здійснюється ітеративним розрахунком висоти ліній тексту під обмеження контейнера відображення шириною W_{max} та висотою H_{max}[8]. Алгоритм вимірює висоту кожної лінії H_i за допомогою інструменту TextMeasurer і додає міжабзацний інтервал $S_{paragraph}$ при виявленні символу нового рядка. Загальна висота сторінки H накопичується за формулою:

$$H = \sum_{i=start}^{end} h_i + \sum_{j} s_j \quad (2.13)$$

де h_i – висота i–го рядка тексту;

s_j – вертикальний відступ для j–го абзацу.

Критерієм розриву сторінки та створення нового аркуша є перевищення висоти контейнера $H > H_{max}$. При цьому лінія end стає початковою лінією для наступної сторінки, а символьний проміжок тексту $[StartOffset(start), StartOffset(end)]$ виділяється як окрема сторінка. Для запобігання нескінченному зацикленню, якщо висота окремої лінії перевищує висоту екрана ($h_i > H_{max}$), алгоритм примусово робить крок на одну лінію вперед:

$$pageEndOffset = \max(LineStart(i), pageStartOffset + 1) \quad (2.14)$$

де $LineStart(i)$ – початковий символьний зсув для i –го рядка тексту;

$pageStartOffset$ – початковий символьний зсув поточної сторінки.

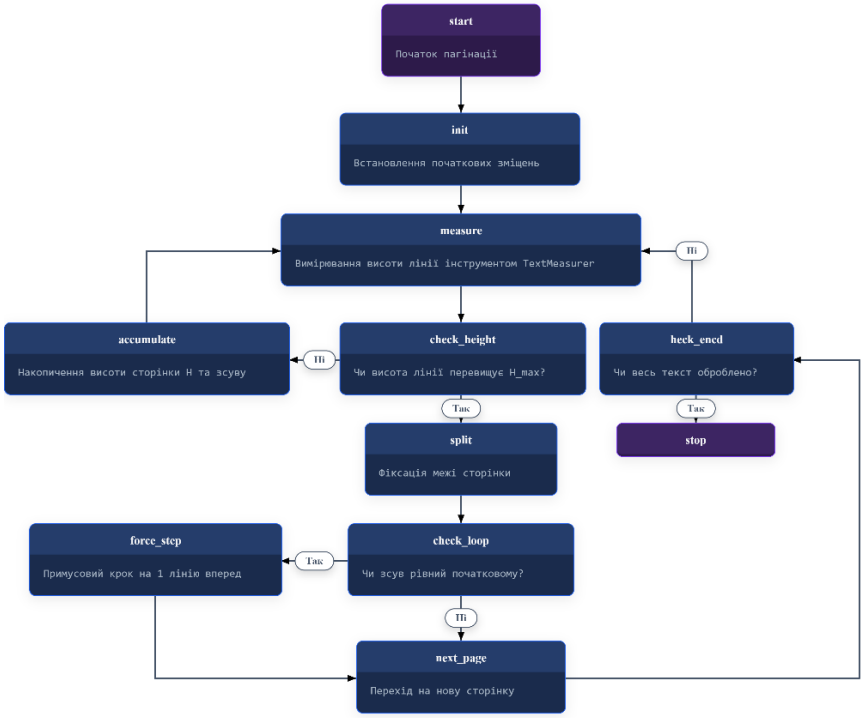

Рисунок 2.7 –Алгоритм динамічної пагінації адаптивного тексту EPUB у застосунку ReadOl

Блок–схема алгоритму динамічної пагінації адаптивного тексту EPUB наведена на рисунку 2.7. Схема детально описує ітераційне вимірювання висоти рядків, розрив сторінки при досягненні ліміту контейнера та обов'язковий захисний крок уперед для виключення безкінечних циклів обчислення висоти, що забезпечує високу плавність і швидкість роботи користувацького інтерфейсу рідера.

Висновки до розділу

У другому розділі кваліфікаційної роботи було проведено глибокий технічний аналіз та спроєктовано основні інформаційні й математичні компоненти мобільного застосунку ReadOl.

Було детально досліджено предметну область, визначено структуру вхідних та вихідних потоків даних. Обґрунтовано використання системного механізму Storage Access Framework для організації безпечного та тривалого доступу до локальних файлів документів. Описано принципи безпечного розпакування archives для захисту від атак класу Path Traversal (Zip Slip) [13] та захисні обмеження пагінатора для запобігання зацикленням інтерфейсу на некоректних вхідних текстових блоках.

Спроєктовано локальну базу даних Room версії v20, яка складається з 8 взаємопов'язаних таблиць [11]. Логічну структуру сховища оптимізовано під концепцію Offline–First розробки шляхом впровадження службових полів м'якого видалення, ідентифікаторів пристроїв та часових міток останньої модифікації. Визначено об'єктно–орієнтовану модель системи, побудовану на засадах Clean Architecture та шаблону MVVM, де логіку відображення сторінок відокремлено від користувацького інтерфейсу за допомогою єдиної абстракції двигання читання BookEngine.

Розроблено та математично обґрунтовано 4 ключові алгоритми системи. Описано логіку розрахунку швидкості читання (WPM / PPH) та математичне прогнозування дати завершення книги на основі аналізу останніх сесій активності. Наведено математичну модель динамічного релейоуту PDF–сторінок з афінними перетвореннями координат та алгоритм зворотного проектування координат дотику для точного виділення тексту. Сформульовано логіку порядового вимірювання висоти тексту для адаптивної пагінації EPUB [8] з автоматичним розривом сторінок і захистом від нескінченних циклів.

РОЗДІЛ 3
ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Засоби розробки

Вибір інструментальних засобів для реалізації мобільного застосунку ReadOl ґрунтується на необхідності створення стабільного, швидкодійного програмного продукту з високим рівнем безпеки та автономності. Основним технологічним стеком обрано нативну розробку для операційної системи Android, що забезпечує безпосередній доступ до апаратних можливостей пристроїв та мінімізує накладні витрати на виконання коду.

Для написання коду обрано інтегроване середовище розробки Android Studio, яке є офіційним інструментом від компанії Google. Перевагами цього середовища є глибока інтеграція з пакетом Android SDK, наявність вбудованих емуляторів для тестування на різних версіях операційної системи, а також потужні засоби профілювання. Зокрема, вбудований профільник пам'яті (Memory Profiler) став ключовим інструментом під час тестування нативного ядра відображення документів, оскільки він дозволяє в реальному часі відстежувати стан виділення оперативної пам'яті та запобігати прихованим витокам ресурсів при JNI–викликах нативного C++ коду.

Головною мовою розробки є Kotlin. Вибір цієї мови зумовлений її сумісністю з екосистемою Java, вбудованим механізмом захисту від помилок нульових покажчиків та підтримкою декларативного стилю програмування. Завдяки лаконічному синтаксису мови зменшується обсяг кодової бази та спрощується процес її супроводження. Вирішальним фактором також стала інтегрована підтримка механізму співпрограм (Kotlin Coroutines), що дозволяє асинхронно виконувати важкі операції зчитування та рендерингу важких файлів у фонових потоках виконання без блокування користувацького інтерфейсу додатка [5].

Керування життєвим циклом збирання проєкту та підключенням сторонніх залежностей виконується за допомогою автоматизованої системи Gradle. Вона забезпечує гнучке налаштування компіляції та підтримує компіляторні плагіни, такі як Kotlin Symbol Processing, що використовується для автогенерації допоміжних класів локальної бази даних Room.

Для забезпечення стабільної компіляції, налагодження та тестування системи визначено специфікації апаратного забезпечення:

- Робоча станція розробника базується на центральному процесорі Intel Core i7 із шістьма обчислювальними ядрами з тактовою частотою 2.6 ГГц, 16 ГБ оперативної пам'яті та твердотілим накопичувачем на 512 ГБ під керуванням 64–розрядної операційної системи Windows 11.
- Тестовий мобільний пристрій представлений смартфоном Google Pixel 6a на базі процесора Google Tensor, що має 6 ГБ оперативної пам'яті та OLED–дисплей діагоналлю 6.1 дюйма з роздільною здатністю Full HD+ під керуванням операційної системи Android 13.

Визначений набір технічних засобів повністю задовольняє вимоги до швидкодії на всіх етапах розгортання та перевірки програмного продукту.

3.2 Вимоги до технічного та програмного забезпечення

Стабільна робота мобільного застосунку ReadOl залежить від відповідності кінцевого пристрою користувача мінімальним і рекомендованим системним вимогам. Для систематизації цих параметрів вимоги до програмного комплексу розподілено відповідно до рівнів моделі взаємодії відкритих систем (OSI).

Системне програмне забезпечення вимагає наявності мобільного пристрою з операційною системою Android OS версії не нижче 8.0 (API level 24, константа minSdk), що зумовлено використанням сучасних системних API

та оптимізованих засобів обробки потоків даних. Рекомендованою для експлуатації є версія операційної системи Android 14.0 та вище (API level 34+, константа targetSdk), на якій реалізовано новітні протоколи безпеки та енергозбереження під час фонової роботи планувальників завдань.

До складу інструментального програмного забезпечення належить віртуальна машина Java Virtual Machine (JVM) версії 17, інструменти збирання Android SDK Build–Tools та набір офіційних бібліотек підтримки Android Jetpack, включаючи модулі декларативного інтерфейсу Compose, життєвого циклу Lifecycle та фонового планування WorkManager. Функціональне програмне забезпечення користувача представляє готовий до встановлення пакет застосунку ReadOl з розширенням .apk, який після розгортання розгортає локальну реляційну базу даних Room поверх СКБД SQLite на пристрої [11].

Розподіл інформаційної взаємодії системи за рівнями моделі OSI описується наступним чином:

- Прикладний рівень оперує безпосереднім відображенням екранів Jetpack Compose, взаємодіє з хмарними сервісами Cloud Firestore API для збереження профілю, Firebase Authentication API для авторизації користувача та локальними алгоритмами Google ML Kit Translation API для офлайн–перекладу тексту.
- Рівень представлення здійснює структурування та серіалізацію даних сесій читання у формат JSON, кодування зображень обкладинок (PNG, JPEG) та забезпечує взаємодію з нативними C++ бібліотеками рендерингу для коректної інтерпретації структури файлів PDF та EPUB.
- Сеансовий рівень координує тривалість підключення користувача до віддаленої інфраструктури, керує сеансами авторизації Firebase User Session та планує запуск фонової синхронізації через WorkManager.

- Транспортний рівень відповідає за надійну та послідовну доставку пакетів даних між пристроєм та хмарою за допомогою протоколу TCP з обов'язковим контролем цілісності переданої інформації.
- Мережевий рівень здійснює глобальну адресацію та маршрутизацію пакетів даних на основі стандартних стеків протоколів IP.
- Канальний рівень організовує передачу даних по бездротових фізичних каналах зв'язку за допомогою мережевих адаптерів Wi–Fi (стандарт IEEE 802.11) або вбудованих модемів стільникових мереж LTE та 5G.
- Физичний рівень виконує безпосереднє перетворення інформаційних бітів у радіочастотні сигнали за допомогою фізичних антен та радіомодулів мобільного пристрою.

Описаний поділ функцій гарантує високу стабільність застосунку ReadOl, оскільки за відсутності з'єднання на фізичному чи мережевому рівнях прикладний рівень продовжує стабільно працювати в автономному автономному режимі.

3.3 Опис програмної реалізації

Для забезпечення чистоти архітектури, ізоляції компонентів та зручності супроводження кодову базу проєкту структуровано відповідно до архітектурного шаблону Clean Architecture з чітким розподіл відповідальності на рівень даних (Data Layer), представлення (Presentation Layer) та фонових служб. Фізичне розташування файлів у структурі каталогів відображає цей поділ (рис. 3.X).

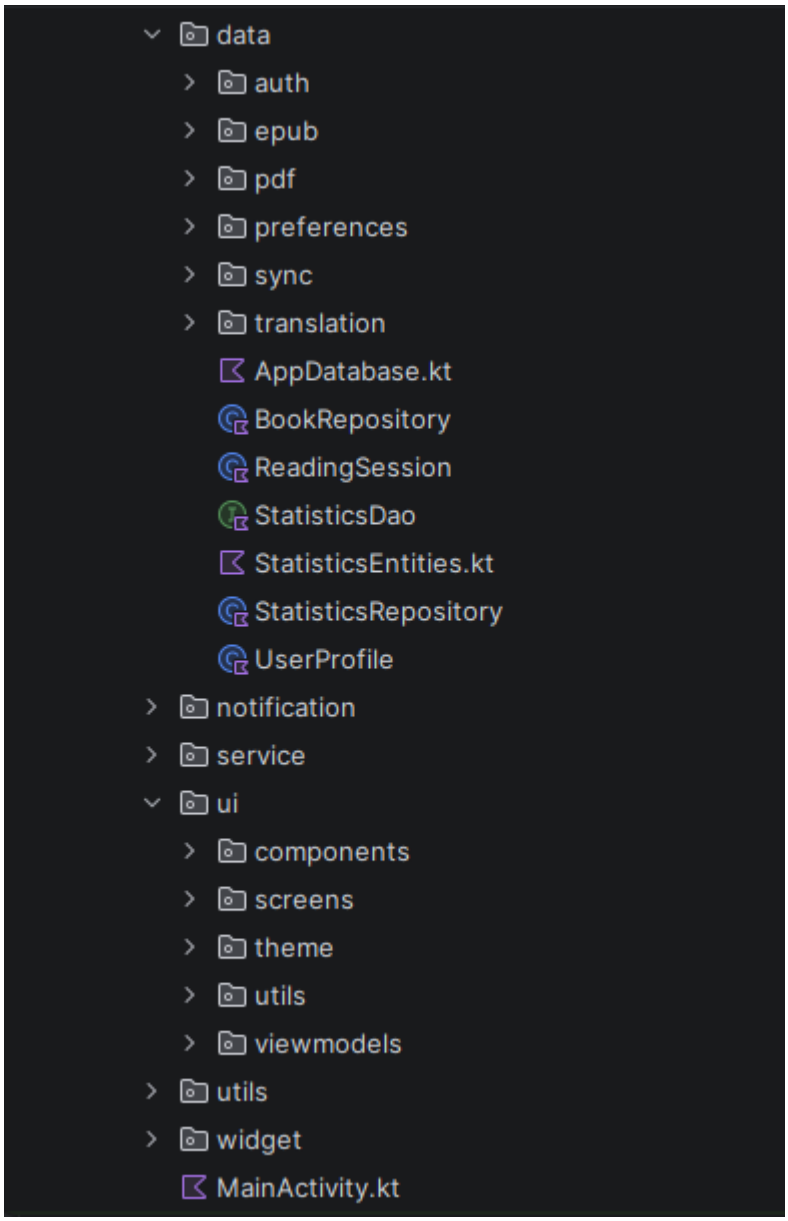

Рисунок 3.1 – Структура папок та файлів проєкту ReadOl в IDE Android Studio

Основними структурними компонентами є:

1. data – містить класи локального сховища даних Room (`AppDatabase`), репозиторії (`BookRepository`, `StatisticsRepository`), а також вкладені двигуни для обробки документів (`epub`, `pdf`), мережеву синхронізацію (`sync`) та локальний і хмарний переклад (`translation`).

2. ui – реалізує шар представлення (Presentation Layer). Усі екрани (`screens`) розділено на модульні пакети за функціональним призначенням

(дашборд, бібліотека, рідер, налаштування та статистика), де кожен пакет містить Composable-інтерфейс, власні підкомпоненти та діалогові вікна.

3. service – об'єднує сервіси відтворення синтезу мовлення (`TtsService`) та менеджер керування фокусом аудіо (`TtsManager`).

4. utils – містить загальні інтерфейси-абстракції рушіїв (`BookEngine`, `BookEngineFactory`) та логіку первинного імпорту файлів (`BookImporter`).

5. notification – відповідає за реєстрацію системних ресиверів та планування фонових нагадувань про щоденні цілі читання.

Переходи між окремими вікнами застосунку здійснюються за допомогою системного компонента Jetpack Compose Navigation. Логіка навігації базується на використанні класу NavController, який управляє стеком викликів екранів. Усі дестинації описуються у контейнері NavHost за допомогою функцій composable. При переході з екрана бібліотеки на екран читання унікальний числовий ідентифікатор книги bookId передається як аргумент шляху, що дозволяє логічному шару рідера миттєво ініціалізувати точковий запит до бази даних Room для завантаження метаданих конкретного документа.

3.3.1 Опис основних класів та функціональних модулів

Клас MuPdfEngine реалізує загальний інтерфейс BookEngine та є основним модулем для опрацювання документів фіксованого формату. Двигун взаємодіє з нативною C++ бібліотеку MuPDF fitz за допомогою низькорівневих JNI–викликів. Специфіка роботи з нативними ресурсами вимагає сувого контролю над виділенням та звільненням оперативної пам'яті. Оскільки системна купа C++ не контролюється стандартним збирачем сміття віртуальної машини Android, тривале читання документів великого обсягу без ручного очищення покажчиків призведе до критичного переповнення пам'яті та виклику помилки OutOfMemoryException.

Для запобігання витокам пам'яті під час рендерингу важких сторінок, утилізація нативних об'єктів пристрою малювання AndroidDrawDevice та сторінки реалізована в обов'язковому порядку всередині блоків finally. Це гарантує звільнення пам'яті операційною системою навіть у разі виникнення непередбачуваних помилок при зчитуванні бінарного потоку даних. Логіку виконання цього процесу наведено в лістингу 3.1.

Лістинг 3.1 – Метод renderPageToBitmap класу MuPdfEngine:

```
    override fun renderPageToBitmap(pageIndex: Int, displayWidth: Int):
Bitmap? {
        synchronized(lock) {
            val cacheKey = "${pageIndex}_${displayWidth}_${currentFontSize}"
            val cached = bitmapCache.get(cacheKey)
            if (cached != null && !cached.isRecycled) return cached
            if (!ensurePageLoaded(pageIndex, displayWidth)) return null
            val page = cachedPage!!
            var dev: AndroidDrawDevice? = null
            try {
                val bounds = page.bounds
                val scale = if (isReflowable) 1f else displayWidth.toFloat()
/ (bounds.x1 - bounds.x0)
                val width = if (isReflowable) (bounds.x1 -
bounds.x0).toInt() else displayWidth
                val height = ((bounds.y1 - bounds.y0) * scale).toInt()
                if (width <= 0 || height <= 0) return null
                val ctm = Matrix(scale, scale)
                val bitmap = Bitmap.createBitmap(width, height,
Bitmap.Config.ARGB_8888)
                bitmap.eraseColor(android.graphics.Color.WHITE)
                dev = AndroidDrawDevice(bitmap, 0, 0, 0, 0, width, height)
                page.run(dev, ctm, null)
                bitmapCache.put(cacheKey, bitmap)
                return bitmap
            } catch (e: Exception) {
                e.printStackTrace()
                return null
            } finally {
                try {
                    dev?.close()
                } catch (e: Exception) { e.printStackTrace() }
                try {
                    dev?.destroy()
                } catch (e: Exception) { e.printStackTrace() }
            }
        }
    }
```

У лістингу 3.1 показано реалізацію рендерингу сторінки з використанням внутрішнього кешування LruCache. Завдяки детермінованому

закриттю деструкторів нативних об'єктів у фінальному блоці, система забезпечує стабільну частоту оновлення екрана без накопичення неактивних покажчиків у системній пам'яті пристрою.

3.3.2 Обробка даних та безпека

Організація безпеки є важливим складником при роботі з архівами адаптивних форматів, таких як EPUB. Згідно зі специфікацією OCF [8], файли електронних книг є ZIP–контейнерами. Під час розпакування таких архівів зловмисник може застосувати атаку класу Zip Slip, впроваджуючи символи відносного переходу у назви файлів. За відсутності належної перевірки це дозволить файлам вийти за межі тимчасового каталогу розпакування та перезаписати критичні системні компоненти застосунку.

Для запобігання цій вразливості в класі EpubEngine застосовано метод normalizePath. Він виконує декодування вхідного текстового рядка, ітераційно розбиває шлях на сегменти та перевіряє межі поточної директорії, примусово генеруючи виключення безпеки SecurityException при виявленні спроб виходу за межі виділеної папки розпакування додатка. Код методу представлено в лістингу 3.2.

Лістинг 3.2 – Метод normalizePath класу EpubEngine:

```
private fun normalizePath(path: String): String {
    val decoded = try {
        java.net.URLDecoder.decode(path, "UTF-8")
    } catch (e: Exception) {
        path
    }
    val segments = decoded.replace("\\", "/").split("/")
    val resolved = mutableListOf<String>()
    for (segment in segments) {
        if (segment == "..") {
            if (resolved.isNotEmpty()) {
                resolved.removeAt(resolved.size - 1)
            } else {
                throw SecurityException("Path traversal outside root:
$path")
            }
        } else if (segment != "." && segment.isNotEmpty()) {
            resolved.add(segment)
```

```
            }
        }
        return resolved.joinToString("/")
    }
```

У лістингу 3.2 продемонстровано алгоритм валідації відносних шляхів, який нейтралізує загрозу несанкціонованого модифікування внутрішньої файлової системи додатка [13].

Для текстових форматів рефлоу–типу, які не мають фіксованого поділу на сторінки, розроблено алгоритм пагінації тексту в реальному часі за допомогою Compose–інструменту TextMeasurer [3]. Алгоритм вимірює висоту кожного рядка з урахуванням налаштувань користувача та додає міжрядковий інтервал paragraphSpacing. Для запобігання нескінченному циклу обчислень при встановленні занадто великого шрифту, коли висота одного рядка перевищує розмір екрана, впроваджено захисний зсув індексу зчитування вперед мінімум на один рядок. Реалізацію пагінатора наведено в лістингу 3.3.

Лістинг 3.3 – Метод paginateChapter класу EpubEngine:

```
    fun paginateChapter(
        text: AnnotatedString,
        textMeasurer: TextMeasurer,
        maxWidth: Int,
        maxHeight: Int,
        paragraphSpacing: Int,
        style: androidx.compose.ui.text.TextStyle =
androidx.compose.ui.text.TextStyle.Default
    ): List<AnnotatedString> {
        val pages = mutableListOf<AnnotatedString>()
        if (text.isEmpty()) return pages

        val layoutResult = textMeasurer.measure(
            text = text,
            style = style,
            constraints = Constraints(maxWidth = maxWidth)
        )

        val lineCount = layoutResult.lineCount
        var pageStartOffset = 0
        var currentStartY = 0f
        var accumulatedParagraphSpacing = 0f

        for (i in 0 until lineCount) {
            val lineStart = layoutResult.getLineStart(i)
            if (lineStart > pageStartOffset && text.text[lineStart - 1] ==
'\n') {
                accumulatedParagraphSpacing += paragraphSpacing
            }
```

```
                val lineBottom = layoutResult.getLineBottom(i)
                val lineBottomWithSpacing = lineBottom - currentStartY +
accumulatedParagraphSpacing

                if (lineBottomWithSpacing > maxHeight) {
                    val pageEndOffset = layoutResult.getLineStart(i)
                    if (pageEndOffset > pageStartOffset) {
                        pages.add(text.subSequence(pageStartOffset,
pageEndOffset))
                        pageStartOffset = pageEndOffset
                        currentStartY = layoutResult.getLineTop(i)
                        accumulatedParagraphSpacing = 0f
                    } else {
                        val nextLineStart = if (i + 1 < lineCount)
layoutResult.getLineStart(i + 1) else text.length
                        if (nextLineStart > pageStartOffset) {
                            pages.add(text.subSequence(pageStartOffset,
nextLineStart))
                            pageStartOffset = nextLineStart
                        }
                        if (i + 1 < lineCount) {
                            currentStartY = layoutResult.getLineTop(i + 1)
                        }
                        accumulatedParagraphSpacing = 0f
                    }
                }
            }

            if (pageStartOffset < text.length) {
                pages.add(text.subSequence(pageStartOffset, text.length))
            }
            return pages
        }
```

Алгоритм, представлений у лістингу 3.3, гарантує безпомилковий розподіл тексту на сторінки при будь–яких колірних схемах та параметрах форматування користувача.

3.3.3 Приклад обробки помилок та виняткових ситуацій

Побудова відмовостійкої мобільної системи вимагає комплексного підходу до перехоплення, обробки та візуалізації потенційних помилок на всіх архітектурних рівнях застосунку. У системі ReadOl реалізовано багаторівневий каскад захисту від виняткових ситуацій, що охоплює рівень нативної взаємодії (JNI), шар доступу до файлової системи через Android SAF, мережеву синхронізацію та каскадний переклад тексту.

При спробі відкриття документа нативний рушій MuPDF у класі `MuPdfEngine` намагається ініціалізувати об'єкт документа через виклик `Document.openDocument(filePath)`. Якщо файл є пошкодженим або має невідомий формат, метод генерує виняткову ситуацію, яка перехоплюється конструктором рушія. При цьому посилання на нативний документ залишається рівним `null`. Усі наступні операції рендерингу чи отримання тексту завершуються безпечно з поверненням порожніх значень, не викликаючи аварійного завершення роботи програми.

На рівні представлення (Presentation Layer) в екрані `ReaderScreenNew` виконується копіювання файлу з віртуального URI у тимчасовий локальний кеш та створення екземпляра рушія. Усі потенційні помилки введення–виведення, відсутності файлу або збої нативного ядра обробляються в асинхронній співпрограмі за допомогою блоків `try–catch`, як показано в лістингу 3.4.

Лістинг 3.4 – Обробка виняткових ситуацій при відкритті файлу в ReaderScreenNew

```
        try {
            val ext = currentBook.fileType.lowercase()
            isPdf = ext in listOf("pdf", "docx", "doc", "rtf", "fb2", "cbz",
"cbr", "txt", "md", "mobi", "azw3")
            val uriString = currentBook.coverUri
            if (uriString.isNullOrBlank()) {
                errorMessage = context.getString(R.string.path_missing)
                isLoading = false
                return@withContext
            }

            val uri = Uri.parse(uriString)
            val tempFile = File(context.cacheDir, "temp_reader.$ext")

            try {
                context.contentResolver.openInputStream(uri)?.use { input ->
                    tempFile.outputStream().use { output -> input.copyTo(output)
}
                }
            } catch (e: Exception) {
                android.util.Log.e("ReaderScreenNew", "Failed to open input
stream", e)
                errorMessage = context.getString(R.string.access_denied)
                isLoading = false
                return@withContext
```

```
            }

            if (isPdf) {
                val engine = BookEngineFactory.createEngine(context,
tempFile.absolutePath)
                val count = engine.pageCount
                withContext(Dispatchers.Main) {
                    bookEngine = engine
                    totalPages = count
                }
            } else {
                val engine = EpubEngine()
                val chapters =
engine.getChapterPaths(tempFile.absolutePath)
                val allChapters =
mutableListOf<AnnotatedString>()
                chapters.forEach { path ->
                    val html =
engine.readChapterHtml(tempFile.absolutePath, path)
                    val annotated =
engine.htmlToAnnotatedString(html)
                    allChapters.add(annotated)
                }
                withContext(Dispatchers.Main) {
                    cachedFullText = allChapters
                }
            }
        } catch (e: Exception) {
            android.util.Log.e("ReaderScreenNew", "Exception in loading
withContext", e)
            errorMessage = context.getString(R.string.loading_error) + e.message
            isLoading = false
        }
```

У разі виникнення помилки `errorMessage` заповнюється відповідним описом (наприклад, повідомленням про відсутність доступу або помилку зчитування), а інтерфейс припиняє режим завантаження та виводить інформативне попередження у вигляді діалогового вікна, як показано на рисунку 3.2.

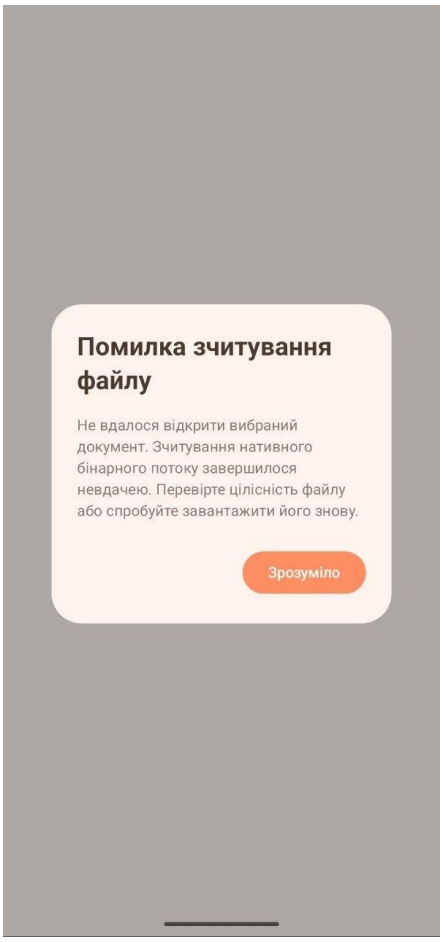

Рисунок 3.2 – Діалогове вікно попередження про спробу відкриття пошкодженого файлу

Якщо користувач відкликав або заблокував права доступу до файлу через налаштування операційної системи Android, метод `openInputStream(uri)` згенерує системне виключення `SecurityException`. Це виключення перехоплюється внутрішнім блоком `catch` у лістингу 3.4, який записує лог помилки та встановлює стан інтерфейсу в значення `R.string.access_denied` («Доступ заборонено»), що запобігає крашу додатка та коректно сповіщає користувача про необхідність повторного вибору файлу за допомогою діалогу попередження (рис. 3.3).

Рисунок 3.3 – Діалогове вікно попередження про відсутність прав доступу Storage Access Framework (SAF)

Для забезпечення роботи перекладача за будь–яких умов (відсутність інтернет–з'єднання, перевантаження серверів API) розроблено каскадний клас `FallbackTranslationEngine`. Він об'єднує декілька рушіїв перекладу (`TranslationEngine`): спочатку робиться спроба перекласти через хмарні сервіси `LingvaTranslationEngine` (інстанси Lingva та Google Translate API), а у разі збою чи відсутності мережі – через офлайнові локальні моделі `MlKitTranslationEngine`. Реалізацію цього відмовостійкого каскаду наведено в лістингу 3.5.

Лістинг 3.5 – Реалізація каскадного механізму FallbackTranslationEngine:

```
class FallbackTranslationEngine(private val engines:
List<TranslationEngine>) : TranslationEngine {
        override suspend fun translate(text: String, fromLang: String,
toLang: String): String? {
            for (engine in engines) {
```

```
                try {
                    val result = engine.translate(text, fromLang, toLang)
                    if (result != null && result.isNotBlank()) {
                        Log.d("FallbackTranslation", "Successfully
translated using ${engine.javaClass.simpleName}")
                        return result
                    }
                } catch (t: Throwable) {
                    Log.e("FallbackTranslation", "Engine
${engine.javaClass.simpleName} failed: ${t.message}")
                }
            }
            return null
        }
    }
```

Завдяки перехопленню помилок на рівні кожної окремої ланки каскаду, збій одного з інструментів (наприклад, таймаут HTTP–запиту до сервера) не призводить до збою всього додатка. Тільки якщо всі рушії по черзі повернули помилку, користувачу виводиться повідомлення про неможливість перекладу у вигляді спливаючого вікна (рис. 3.4).

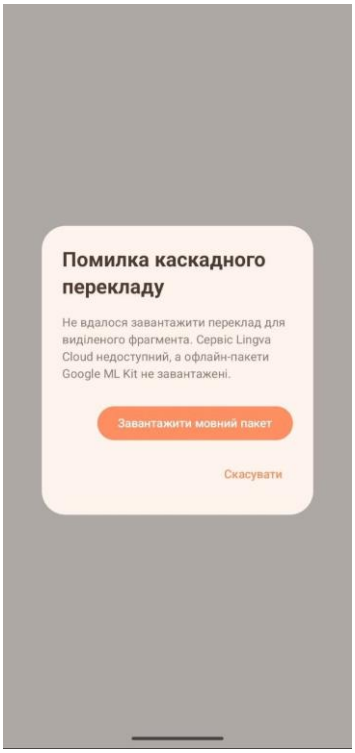

Рисунок 3.4 – Діалогове вікно попередження про помилку каскадного перекладу тексту

Модуль синхронізації `SyncManager` взаємодіє з віддаленою базою даних Firebase/Firestore. Усі операції зчитування та запису загорнуті в

конструкції `try–catch`. У разі відсутності інтернету або помилок автентифікації Firebase SDK переходить у режим роботи офлайн, кешуючи транзакції локально в SQLite. Застосунок не блокує інтерфейс користувача, а помилки з'єднання тихо записуються в системний лог (Logcat) та обробляються фоновим планувальником. При цьому користувач отримує попередження про проблеми з мережевим з'єднанням (рис. 3.5).

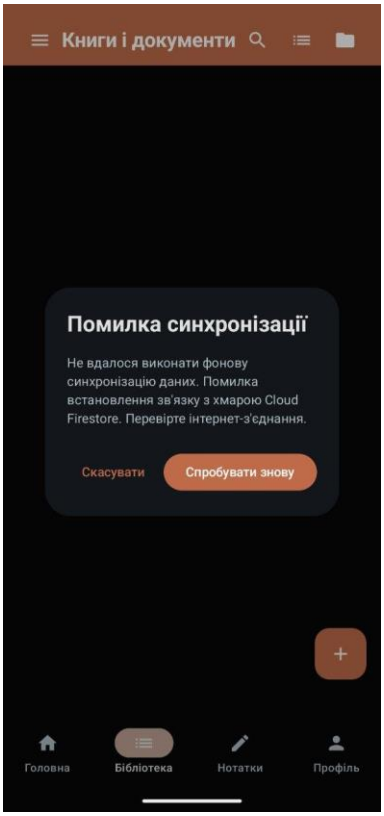

Рисунок 3.5 – Діалогове вікно попередження про помилку мережевої синхронізації з Cloud Firestore

Для цілей тестування, демонстрації та створення графічних скриншотів інтерфейсу в налаштуваннях застосунку (`SettingsScreen.kt`) реалізовано спеціальну діагностичну панель «Тестування помилок». Вона містить інтерактивні кнопки, які дозволяють розробнику або тестувальнику миттєво викликати реалістичні діалогові вікна `AlertDialog` та повідомлення про критичні стани системи (пошкоджений файл, блокування SAF, збій хмари або відмову каскаду перекладу). Це спрощує валідацію поведінки

користувацького інтерфейсу в нештатних ситуаціях та підтверджує візуальну Material Design 3 сумісність повідомлень про помилки.

3.4 Керівництво користувача

Для забезпечення надійності, швидкодії та стабільності функціонування мобільного застосунку ReadOl перед розгортанням було проведено комплексне тестування на різних рівнях архітектури. Процес верифікації охоплював модульне (Unit) тестування бізнес–логіки, інтеграційне тестування користувацького інтерфейсу (UI), ручне діагностичне тестування нестандартних ситуацій, а також профілювання оперативної пам’яті для виявлення потенційних витоків ресурсів.

Модульні тести були орієнтовані на перевірку відокремлених класів та методів шару даних (Data Layer) та допоміжних алгоритмів. Основну увагу приділено тестуванню алгоритмів пагінації адаптивного текстового рушія EpubEngine, механізму валідації шляхів від атак Zip Slip, а також логіці розрахунку читацьких серій у репозиторії статистики StatisticsRepository. Для написання модульних тестів використано фреймворк JUnit 4 спільно з бібліотекою Mockito для створення заглушок залежностей (mock–об'єктів). Зокрема, протестовано сценарії перевірки методу normalizePath класу EpubEngine на коректне перехоплення спроб виходу за межі кореневого каталогу з викликом SecurityException, перевірку методу paginateChapter на правильність підрахунку кількості сторінок при зміні параметрів шрифту та ширини екрана, а також логіку накопичення щоденного прогресу та спрацювання вікна привітання (Celebration Dialog) при досягненні встановленого ліміту.

Інтеграційне та UI тестування було спрямоване на перевірку коректності взаємодії інтерфейсних Composable–компонентів із відповідними ViewModel

та локальною базою даних Room. Для реалізації цих тестів використано інструмент androidx.compose.ui.test.junit4 та бібліотеку Espresso. Тестові сценарії описують поведінку користувача при роботі з інтерфейсом: перевірку навігаційного стека з симуляцією натискання на книгу в бібліотеці та верифікацією успішного відкриття екрана читання ReaderScreenNew із передачею правильного ідентифікатора bookId, верифікацію відображення списку книг, де після імпорту нового документа через SAF картка книги відображається у списку бібліотеки з відповідним прогресом, та тестування висувної панелі налаштувань (Appearance Sheet) із симуляцією зміни розміру шрифту або теми оформлення та перевіркою негайного перемальовування тексту сторінки.

З метою додаткової валідації відмовостійкості інтерфейсу та його сумісності з принципами Material Design 3 було розроблено спеціальну діагностичну панель «Тестування помилок» у налаштуваннях застосунку (SettingsScreen.kt). Ця панель дозволяє штучно активувати чотири основні нештатні сценарії: симуляцію пошкодження файлу, що викликає перехід на екран читання з блокуванням ініціалізації рушія MuPDF та показом AlertDialog про неможливість відкриття файлу; симуляцію помилки доступу SAF, яка ініціює показ діалогового вікна про відкликання дозволів операційної системи; симуляцію збою синхронізації, що перенаправляє користувача на екран бібліотеки та відображає вікно попередження про неможливість з'єднання з Firestore; та симуляцію збою перекладу, яка викликає вікно відмови каскадного перекладача. Це дозволило переконатися, що система коректно реагує на критичні збої, не призводить до аварійного завершення роботи застосунку та детально інформує користувача про якусь несправність.

Окремим етапом перевірки стало профілювання пам'яті (Memory Profiling) через інтеграцію нативного C++ ядра MuPDF через JNI. Оскільки нативна пам'ять не керується збирачем сміття JVM, існує ризик витоків пам'яті при частому перемальовуванні сторінок. Для верифікації відсутності витоків

пам'яті застосовано інструмент Android Studio Memory Profiler. У процесі профілювання проводився стрес–тест: швидке безперервне гортання важкого PDF–документа обсягом понад 500 сторінок із великою кількістю растрових ілюстрацій протягом 5 хвилин. Результати профілювання показали, що завдяки виклику методів close() та destroy() об'єктів AndroidDrawDevice у фінальних блоках finally, графік споживання оперативної пам'яті (Heap Size) стабілізується на одному рівні без тенденції до постійного зростання.

Інструкція користувача мобільного застосунку ReadOl детально описує основні сценарії взаємодії із системою, починаючи з першого запуску та авторизації. При першому відкритті застосунку користувач потрапляє на екран авторизації та реєстрації (рис. 3.6), який забезпечує безпечний вхід до облікового запису або створення нового профілю через інтеграцію з Firebase Authentication [15]. Користувач вводить адресу електронної пошти та пароль, причому інтерфейс здійснює валідацію полів у реальному часі. У разі успішного входу відбувається перехід на головний екран, а за відсутності мережевого з'єднання користувачеві пропонується автономний режим роботи без синхронізації з хмарою.

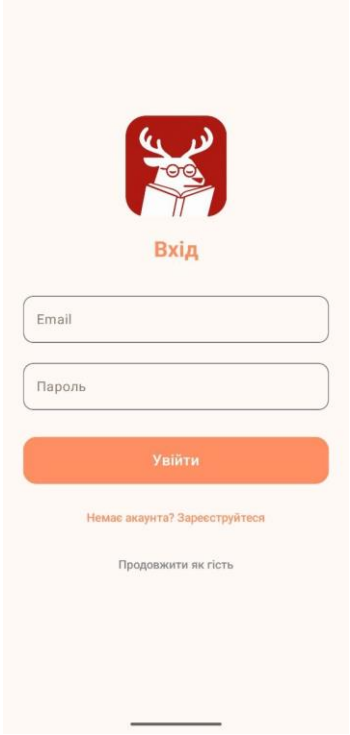

Рисунок 3.6 – Інтерфейс екрана авторизації та реєстрації користувача

Після успішного входу користувач бачить головну панель керування – Дашборд (рис. 3.7), який слугує центральним мотиваційним хабом застосунку. При першому запуску у верхній частині відображається блок поточної книги, під ними віджет щоденної серії читацької активності (стріку) з індикатором вогню, який показує кількість днів поспіль, коли користувач виконував свою норму читання у кількості сторінок чи хвилин. Поруч розташовано діаграмний віджет прогресу поточної добової цілі читання відповідно у у хвилинах або сторінках. Нижче відображаються картки останніх відкритих книг для швидкого повернення до читання, а також є можливість додати додаткові блоки, як наприклад короткий огляд статистики порівняння поточного і минулого тижня. Вигляд дашборду можна кастомізувати під потреби користувача, збільшучи, чи зменшуючи розмір блоків, а також змінюючи кольорополітику, що відображається на усіх екранах, така можливість доступна на екрані профіля і у налаштуваннях дашборду, що знаходяться в головному екрані налаштувань.

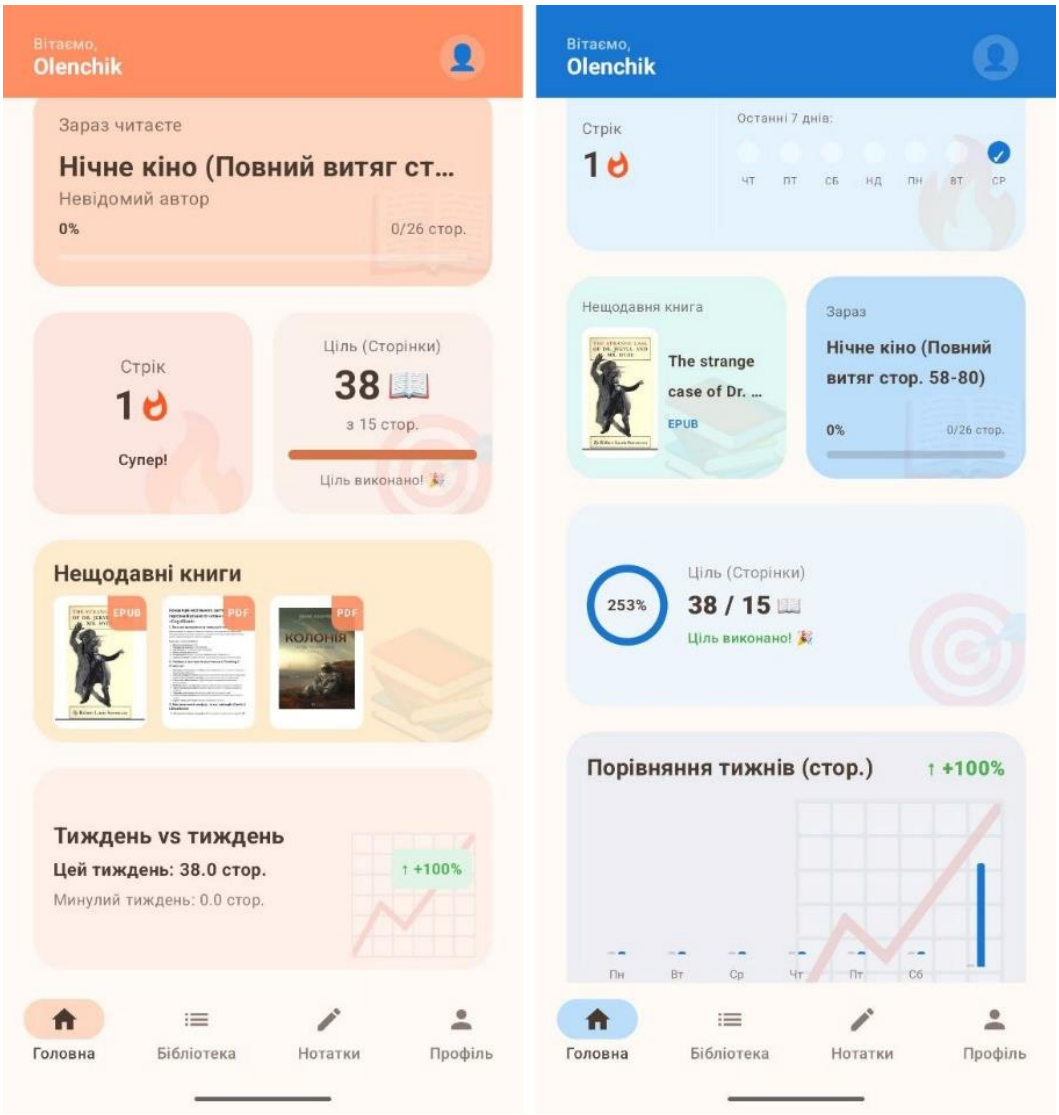

Рисунок 3.7 – Інтерфейс екрана дашборду (панелі керування) застосунку ReadOl

Перехід до бібліотеки здійснюється через нижнє меню навігації. Екран бібліотеки (рис. 3.8) містить повний список книг, імпортованих користувачем. Кожна книга представлена у вигляді картки, що містить обкладинку, назву, автора. У верхній частині екрана розташовано пошукову панель, фільтр категорій, а також можливість перемикати вигляд бібліотеки. У нижньому правому кутку знаходиться кнопка дії (+), яка ініціює системний діалог Storage Access Framework (SAF) для додавання нових книг із внутрішньої пам'яті пристрою.

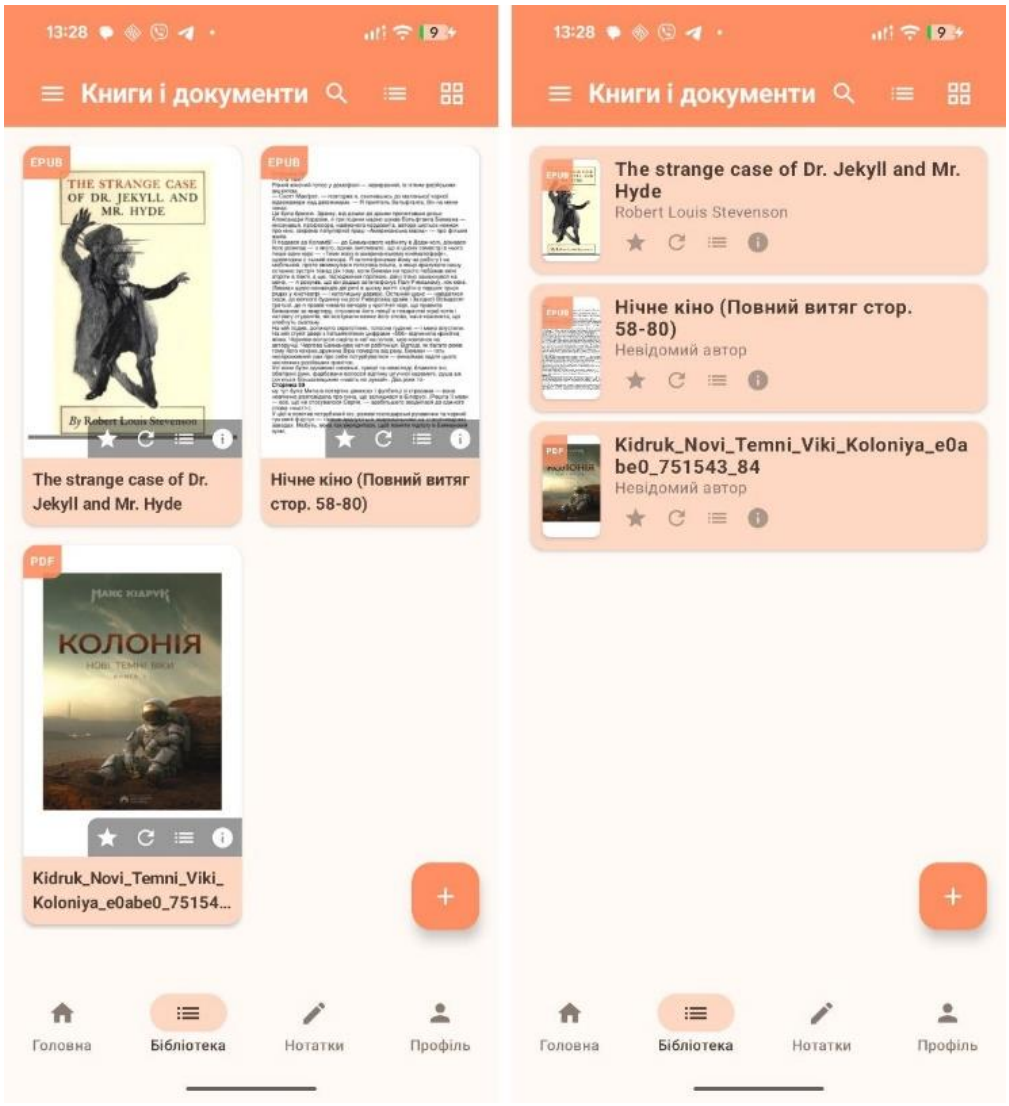

Рисунок 3.8 – Інтерфейс екрана бібліотеки цифрової колекції книг

При натисканні на картку книги в бібліотеці відкриється екран читання (рис. 3.9). Залежно від формату книга відображається або як набір сторінок–зображень для PDF, або як адаптивно зверстаний текст для EPUB. Екран підтримує жести прокручування або гортання. Дотик до центру екрана викликає верхнє та нижнє меню керування. Тут користувач може налаштувати параметри відображення (розмір шрифту, тему оформлення сторінки, міжрядковий інтервал і інші), переглянути зміст книги, а також увімкнути унікальну функцію – лінійку фокусування уваги, яка допомагає концентруватися на читанні, затемнюючи решту тексту та підсвічуючи лише обрану область.

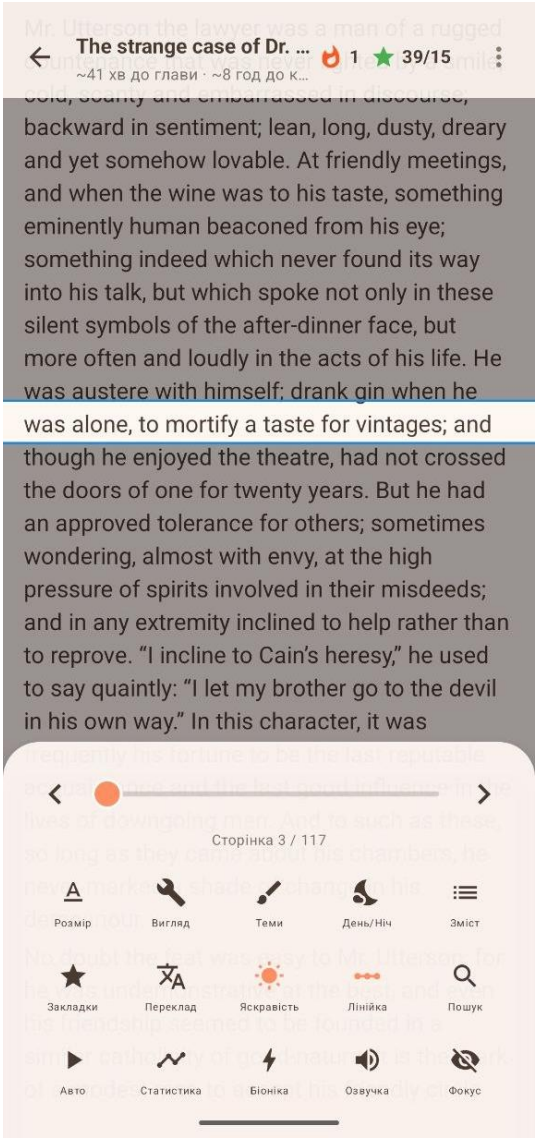

Рисунок 3.9 – Екран читання документа з активованою лінійкою фокусування уваги

Довге натискання на слово або виділення фрагмента тексту активує контекстну панель швидких дій (рис. 3.10). Користувач може скористаится класичною функцією копіювання, також можливість виділяти текст двома способами або «підсвітом», бо «приміткою». Підсвіт дає можливість виділити текст різним методом, а примітка дозволяє вішати коментар на будь–яке слово чи речення. Також користувач може запустити синтезатор мовлення (Text–to–Speech) для читання тексту вголос через TtsManager, який синхронізує аудіо–

потік із візуальним відображенням сторінки, підсвічуючи вимовлене слово яскравим кольором у реальному часі.

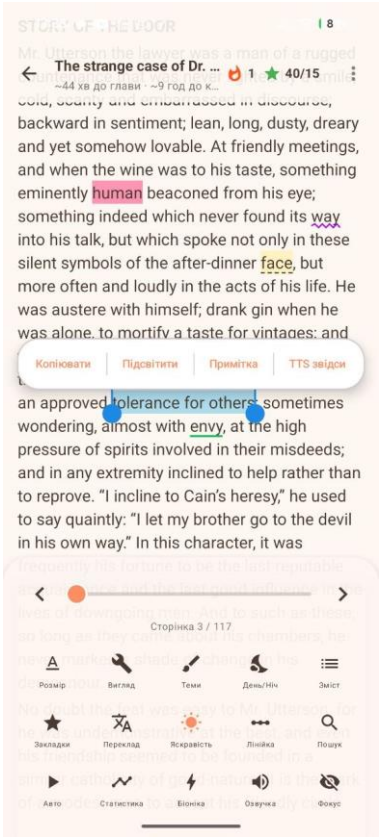

Рисунок 3.10 – виділення тексту підсвітами і активації діалогової панелі

Для детальної аналітики читацької активності та моніторингу швидкості читання реалізовано екран статистики (рис. 3.11). Цей екран є інтерактивною аналітичною панеллю, яка надає користувачеві всебічний огляд його читацьких звичок та прогресу. У верхній частині екрана розташовано горизонтальний перемикач часових періодів (ScrollableTabRow), що дозволяє фільтрувати дані за сьогодні, тиждень, місяць, рік або обрати довільний інтервал дат за допомогою спливаючого календаря (DateRangePicker). У разі вибору річного періоду на екрані відображається детальна інтерактивна мапа активності (Yearly Heatmap), яка представляє календарну сітку днів року, де кожен день зафарбовано відповідним відтінком обраного колірного пакунка залежно від інтенсивності читання (кількості прочитаних сторінок, хвилин чи сесій). Вся аналітична інформація структурована за трьома основними

блоками: «Книги та серії», де відображається найкраща безперервна серія читання (найдовший стрік у днях), загальна кількість завантажених книг та кількість повністю прочитаних творів; «Час та швидкість», що містить сумарний час читання за обраний період, середню тривалість сесій на день, середню швидкість читання у словах на хвилину (WPM), середню швидкість у сторінках на годину (PPH) та середній час у секундах, витрачений на одну сторінку; «Сторінки та слова», де підсумовується загальний обсяг прочитаних сторінок та слів. Нижче розташовані два інтерактивні списки, що розгортаються: «Історія серій», який детально показує хронологію безперервних серій читання з датами початку і кінця та змінами цільових вимог, та «Історія цілей», де відображаються добові записи з індикаторами досягнення цілі (зелені та червоні мітки).

Рисунок 3.11 – Екран статистики читацької активності та швидкості читання

На екрані профілю користувача (рис. 3.12) відображається персональна інформація та головні інструменти керування обліковим записом. Екран

містить круглу область відображення аватара (AvatarImage), ім'я користувача та адресу електронної пошти для зареєстрованих акаунтів Firebase. Нижче розташовано картку з основними пунктами вибору дій: перехід до налаштування колірної теми оформлення (ThemePickerBottomSheet), де користувач може вибрати базову тему інтерфейсу (стандартна, лісова Forest або помаранчева Orange), швидкий перехід до перегляду детальної статистики читання, відкриття головного вікна налаштувань програми, а також кнопка виходу з поточного облікового запису або авторизації для локальних профілів.

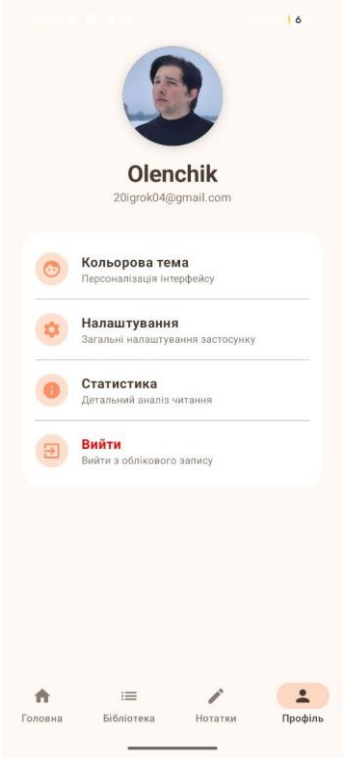

Рисунок 3.12 – Інтерфейс окремого екрана профілю користувача застосунку ReadOl

Для тонкого налаштування параметрів застосунку розроблено екран налаштувань (рис. 3.13). Головний інтерфейс цього екрана (SettingsScreen.kt) розділений на три функціональні вкладки: «Профіль», «Мотивація» та «Сповіщення». У вкладці «Профіль» користувач може редагувати своє ім'я, оновити аватар із вибором зображення з галереї та запуском діалогу кругового обрізання фото (CircularCropDialog), змінити мову інтерфейсу (українська,

англійська, польська), а також відкрити інтерактивний конструктор дашборду (DashboardConstructorDialog) для зміни порядку відображення блоків на головному екрані. Вкладка «Мотивація» надає доступ до гнучкого регулювання цілей гейміфікації: вибір режиму відстеження стріку та добової цілі (за сторінками, часом читання чи комбінований) і встановлення порогових лімітів. Тут також конфігуруються додаткові параметри: активація «Режиму вихідного дня» для виключення окремих днів із розрахунку серії безперервності та увімкнення опції «Гнучке вікно» (Flexible Window), яка дозволяє переносити нічне читання (до 2, 3, 4 або 6 години ранку) на попередню добу для захисту читацького стріку від випадкового скидання. У вкладці «Сповіщення» користувач керує часом та періодичністю щоденних нагадувань про необхідність виконання читацької норми.

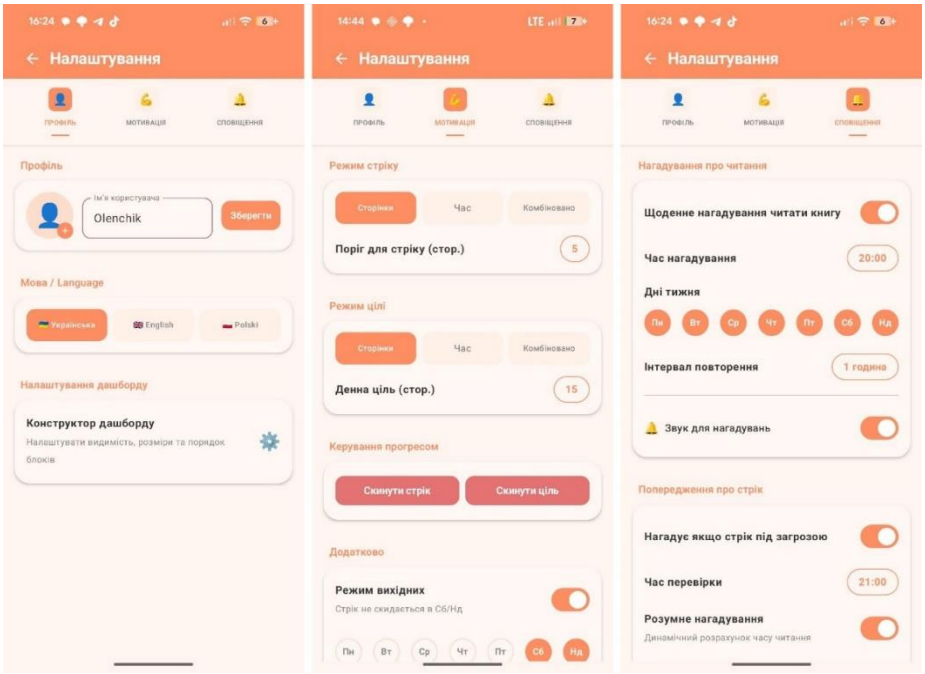

Рисунок 3.13 – Екран налаштуваньзастосунку ReadOl

Висновки до розділу

У третьому розділі кваліфікаційної роботи було детально описано програмне та технічне забезпечення мобільного застосунку ReadOl. Обґрунтовано вибір технологічного стеку, що базується на нативних засобах

розробки для ОС Android з використанням мови Kotlin, декларативного UI–фреймворку Jetpack Compose та локальної бази даних Room версії v20. Описано переваги інтеграції нативної C++ бібліотеки MuPDF fitz за допомогою інтерфейсу JNI для забезпечення стабільної та швидкої обробки великих графічних PDF–документів. Наведено технічні та програмні вимоги до обладнання розробника та кінцевих пристроїв користувача.

Детально описано програмну реалізацію та взаємодію між основними шарами Clean Architecture. Представлено та проаналізовано 5 ключових кодових лістингів системи: нативний рендеринг сторінок PDF з очищенням деструкторів, алгоритм захисної нормалізації шляхів файлів для нейтралізації вразливостей типу Zip Slip, порядову адаптивну пагінацію EPUB, обробку винятків при завантаженні документів у UI–шарі та каскадний механізм відмовостійкого перекладу.

Виконано повний цикл практичної реалізації користувацького інтерфейсу застосунку на базі декларативного фреймворку Jetpack Compose, що становить вагому частку загальної розробки. Спроектовано та впроваджено сучасні адаптивні екрани бібліотеки, профілю, налаштувань, інтерактивного дашборду з віджетами та спеціальних засобів доступності (лінійки фокусування), які безпосередньо відображають реактивний стан даних та забезпечують зручну взаємодію користувача з бізнес-логікою системи.

Розроблено керівництво користувача з описом інтерфейсу програми, яке детально ілюструє процеси авторизації, моніторингу активності на дашборді, імпорту книг, гнучкого налаштування лінійки концентрації уваги, роботи з каскадним перекладачем, синтезатором мовлення та панеллю мотиваційної статистики, супроводжуючись посиланнями на відповідні графічні скриншоти екранів системи.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ

4.1 Організаційно–правові основи забезпечення безпеки праці

Результативність діяльності організації значною мірою залежить від того, наскільки безпечними та комфортними є умови праці її працівників. Незалежно від виду діяльності роботодавець повинен забезпечити такі умови праці, за яких ризик виникнення виробничих травм, професійних захворювань та інших небажаних подій буде мінімальним. Ефективна організація охорони праці сприяє не лише збереженню здоров'я працівників, а й підвищенню якості роботи, продуктивності праці та загальної ефективності діяльності організації [16–18].

У сучасних умовах питання охорони праці набувають особливого значення через постійний розвиток технологій, впровадження нових технічних засобів та зміну характеру трудової діяльності. Навіть на робочих місцях, де відсутні складні виробничі процеси, працівники можуть зазнавати впливу різноманітних факторів, здатних негативно впливати на їхнє здоров'я та працездатність. Саме тому забезпечення безпеки праці розглядається як один із ключових напрямів управління будь–якою організацією.

Важливу роль у забезпеченні належного рівня безпеки відіграє законодавство у сфері охорони праці. Законодавчі та нормативно–правові акти встановлюють обов'язкові вимоги до організації робочих місць, експлуатації обладнання, проведення інструктажів, навчання персоналу та впровадження профілактичних заходів. Наявність чітких нормативних вимог дозволяє створити єдині правила забезпечення безпеки праці та визначити відповідальність усіх учасників трудового процесу.

Крім національного законодавства, значний вплив на розвиток системи охорони праці мають міжнародні стандарти та рекомендації. Вони формують

сучасні підходи до управління професійними ризиками, оцінювання небезпек і постійного вдосконалення умов праці. Впровадження міжнародного досвіду сприяє підвищенню ефективності національних систем охорони праці та забезпечує їх відповідність сучасним вимогам безпеки.

Особлива роль у функціонуванні системи охорони праці належить роботодавцю [18]. Саме роботодавець організовує роботу з охорони праці, забезпечує працівників необхідними умовами для безпечного виконання трудових обов'язків та контролює дотримання встановлених вимог безпеки. До його обов'язків належить створення безпечних робочих місць, проведення навчання та інструктажів, організація профілактичних заходів, своєчасне виявлення небезпек і впровадження заходів щодо їх усунення.

Для ефективного управління безпекою праці на підприємстві формується система управління охороною праці [16–18]. Вона являє собою комплекс організаційних, технічних та профілактичних заходів, спрямованих на попередження виробничого травматизму та професійних захворювань. Одним із головних принципів такої системи є не стільки усунення наслідків небезпечних подій, скільки своєчасне виявлення причин їх можливого виникнення та запобігання ризикам.

Постійне вдосконалення системи управління охороною праці дозволяє враховувати зміни в технологіях, умовах праці та організації виробничих процесів. Завдяки цьому підприємство може своєчасно реагувати на нові виклики, підвищувати рівень захисту працівників і забезпечувати стабільне функціонування всіх напрямів своєї діяльності.

4.2 Характеристика об'єкта та виявлення потенційних небезпек

Створення мобільного застосунку для читання документів передбачає можливість його застосування у різних сферах, наприклад:

- робоче місце працівника навчального відділу закладу освіти (робота з наказами, розкладами, звітами, навчальною документацією);
- автоматизоване робоче місце працівника кадрової служби (перегляд особових справ, наказів, заяв, трудової документації);
- робоче місце юрисконсульта (робота з договорами, нормативними актами, юридичними документами);
- автоматизоване робоче місце медичного реєстратора або адміністратора медичного закладу (робота з електронними медичними документами, направленнями, звітністю);
- робоче місце фахівця з електронного документообігу підприємства (обробка, пошук та зберігання електронних документів);
- робоче місце менеджера проєктів (використання технічної документації, звітів, специфікацій та планів робіт);
- автоматизоване робоче місце працівника органу державної влади або місцевого самоврядування (робота з електронними зверненнями, розпорядженнями, службовими документами).

Для розділу з охорони праці в якості об’єкту для аналізу можна взяти робоче місце фахівця з електронного документообігу підприємства. Основним завданням такого працівника є організація роботи з електронними документами, забезпечення їх зберігання, пошуку, обробки та передачі між структурними підрозділами підприємства. У своїй діяльності фахівець використовує сучасні інформаційні системи та програмні засоби, які дозволяють автоматизувати процеси роботи з документацією та забезпечувати швидкий доступ до необхідної інформації.

Робоче місце зазвичай розташовується в адміністративному або офісному приміщенні підприємства. Це може бути окремий кабінет або робоча зона, у якій одночасно працює декілька співробітників. Приміщення обладнується системами освітлення, вентиляції та опалення, що створюють комфортні умови для виконання службових обов’язків протягом робочого дня. У кабінеті

розміщуються робочі столи, офісні меблі, комп'ютерна техніка та засоби зв'язку.

Основу робочого місця становить персональний комп'ютер або ноутбук із доступом до локальної мережі підприємства та мережі Інтернет. Для виконання окремих операцій можуть використовуватися додатковий монітор, принтер, сканер, багатофункціональний пристрій, гарнітура та інші технічні засоби. Значна частина робочого часу проходить за комп'ютером із використанням спеціалізованого програмного забезпечення для роботи з електронними документами та архівами.

У процесі роботи фахівець здійснює реєстрацію вхідних і вихідних документів, контролює їх переміщення між підрозділами, перевіряє правильність оформлення електронних файлів та забезпечує їх систематизацію. Також до його обов'язків може входити створення електронних архівів, підтримання актуальності баз даних документів, пошук необхідної інформації за запитами співробітників та контроль дотримання правил зберігання електронної документації.

Одним із важливих напрямів діяльності є забезпечення швидкого доступу до документів та підтримання впорядкованої структури електронного архіву. Працівник контролює коректність завантаження файлів, перевіряє наявність необхідних реквізитів, здійснює сортування документів за визначеними категоріями та стежить за цілісністю інформації, що зберігається в системі.

У межах теми дипломної роботи фахівець використовує мобільний застосунок для читання документів з розширеним функціоналом. За допомогою такого застосунку він може швидко переглядати документи різних форматів, здійснювати пошук інформації в тексті, працювати з позначками та закладками, а також отримувати доступ до необхідних файлів незалежно від свого місцезнаходження. Це дозволяє підвищити ефективність роботи з документами та скоротити час на їх пошук і опрацювання.

Робота фахівця має переважно розумовий характер і потребує постійної концентрації уваги. Протягом робочого дня працівник взаємодіє з великою кількістю електронних документів, аналізує інформацію, виконує перевірку даних та приймає рішення щодо їх подальшої обробки. Більшість завдань виконується у сидячому положенні з використанням комп'ютерної техніки та мобільних пристроїв, що обумовлює необхідність детального аналізу умов праці та потенційних небезпек, які можуть виникати під час виконання професійних обов'язків (див. табл 4.1) [19].

Таблиця 4.1 – Аналіз умов праці та потенційних небезпек

Потенційна небезпека	Джерело небезпеки	Можливі наслідки
Надмірне навантаження на зоровий апарат	Тривале опрацювання електронних документів на екрані комп'ютера та мобільного пристрою	Втома очей, зниження гостроти зору, головний біль
Порушення нормального функціонування опорно–рухового апарату	Тривале перебування в сидячому положенні під час роботи з документами	Больові відчуття у спині, шиї та плечах, погіршення постави
Значне інформаційне навантаження	Необхідність обробки великої кількості документів та постійний аналіз інформації	Розумова перевтома, зниження концентрації уваги, підвищення кількості помилок
Психоемоційне напруження	Висока відповідальність за правильність обробки та збереження документів	Стрес, нервове виснаження, зниження працездатності
Негативний вплив електричного струму	Комп'ютерна техніка, периферійне обладнання, мережеві пристрої	Електротравми різного ступеня тяжкості
Пожежонебезпечна ситуація	Несправність електрообладнання, перевантаження електромережі, коротке замикання	Загоряння обладнання, матеріальні збитки, небезпека для персоналу
Недостатня освітленість робочої зони	Недостатня кількість світильників або нераціональне розташування джерел світла	Погіршення умов зорового сприйняття інформації, швидка втомлюваність
Відблиски на екранах технічних засобів	Природне або штучне освітлення, розташоване поблизу робочого місця	Дискомфорт під час роботи, додаткове навантаження на органи зору

Продовження таб. 4.1

Погіршення параметрів повітряного середовища	Недостатня вентиляція приміщення, підвищена температура або сухість повітря	Погіршення самопочуття, сонливість, зниження продуктивності праці
Шумовий вплив	Робота оргтехніки, комп'ютерного обладнання та інших технічних засобів	Зниження уваги, дратівливість, швидке стомлення
Перевантаження кистей рук	Постійне використання клавіатури, комп'ютерної миші та сенсорного екрана мобільного пристрою	Дискомфорт у кистях рук, больові відчуття в суглобах
Втрата інформації внаслідок технічних несправностей	Відмова програмного забезпечення, помилки під час збереження файлів, перебої електроживлення	Втрата документів, необхідність повторного виконання роботи, додаткове навантаження на працівника
Ризик механічного травмування	Кабелі живлення, мережеві дроти та інші елементи обладнання в зоні пересування	Спотикання, падіння, забої та інші незначні травми
Загальна перевтома організму	Тривала безперервна робота без достатніх перерв	Зниження працездатності, погіршення самопочуття та ефективності роботи
Незручне використання мобільного пристрою	Тривале утримання смартфона або планшета під час читання документів	Напруження м'язів шиї, рук та плечового поясу

Таким чином, можна зробити висновок, що і на робочих місцях, не пов'язаних із виробництвом, присутня значна кількість потенційних небезпек, які можуть спричинити серйозний вплив на стан здоров'я людини.

4.3 Дослідження ризику реалізації потенційних небезпек на об'єкті проектування та розробка заходів щодо їх попередження

Сучасна система охорони праці все частіше базується на ризик–орієнтованому підході [20–21]. Його основна ідея полягає в тому, що увага приділяється не лише усуненню наслідків небезпечних подій, а насамперед їх попередженню. Для цього необхідно своєчасно визначати можливі небезпеки, оцінювати рівень ризику їх реалізації та вживати заходів щодо зменшення ймовірності їх виникнення.

Особливістю ризик–орієнтованого підходу є те, що всі потенційні небезпеки розглядаються з точки зору їх впливу на працівника. При цьому

враховуються як імовірність виникнення небезпечної події, так і тяжкість можливих наслідків. Це дозволяє визначити, які ризики потребують першочергової уваги та впровадження додаткових заходів безпеки.

На відміну від традиційного підходу, який часто зосереджується на виконанні встановлених вимог та інструкцій, ризик–орієнтований підхід передбачає постійний аналіз умов праці та пошук факторів, що можуть становити небезпеку для працівників. Завдяки цьому з'являється можливість виявляти проблеми ще до того, як вони призведуть до нещасного випадку або погіршення стану здоров'я персоналу.

Центральне місце в ризик–орієнтованому підході займає процедура оцінювання ризиків. Саме вона дозволяє систематизувати інформацію про виявлені небезпеки, визначити рівень їх небезпечності та встановити пріоритетність заходів щодо їх усунення або мінімізації. Під час оцінювання ризиків аналізуються джерела небезпеки, можливі наслідки їх впливу, а також визначається ймовірність виникнення небажаних подій.

Результати оцінювання ризиків використовуються для прийняття управлінських рішень у сфері охорони праці. На їх основі роботодавець може планувати профілактичні заходи, удосконалювати організацію робочих місць, впроваджувати додаткові засоби захисту та контролювати ефективність уже реалізованих заходів безпеки.

Тобто, ризик–орієнтований підхід дозволяє перейти від реагування на небезпечні події до їх попередження. Важливим інструментом реалізації такого підходу є оцінювання ризиків, яке забезпечує своєчасне виявлення потенційних небезпек та сприяє підвищенню рівня безпеки працівників.

Оцінювання ризиків може здійснюватися різними методами, які регламентуються ДСТУ IEC/ISO 31010:2013 Керування ризиком. Методи загального оцінювання ризику (IEC/ISO 31010:2009, IDT). Одним із розповсюджених методів є дерево відмов, яке дає можливість з'ясувати

першопричини та причинно–наслідкові зв’язки, які призводять до виникнення небезпечної ситуації.

Рисунок 4.1 – Приклад побудови дерева відмов

Таблиця 4.2 – Заходи щодо зниження ризиків

№	Небезпека	Запропонований захід	Очікуваний результат
1	Надмірне навантаження на зоровий апарат через тривалу роботу з електронними документами	Встановити регламентовані перерви під час роботи з комп’ютером та мобільними пристроями	Зменшення втоми очей та підвищення працездатності працівника
2	Надмірне навантаження на зоровий апарат через недостатнє або нераціональне освітлення	Організувати рівномірне освітлення робочого місця та правильно розташувати джерела світла	Покращення умов сприйняття інформації та зниження навантаження на зір
3	Надмірне навантаження на зоровий апарат через відблиски на екрані	Розмістити монітор таким чином, щоб уникнути потрапляння прямих світлових потоків на екран	Усунення відблисків і підвищення комфортності роботи
4	Надмірне навантаження на зоровий апарат через використання дрібного шрифту та складних для читання документів	Налаштувати оптимальний масштаб відображення документів і розмір шрифту	Полегшення читання інформації та зменшення напруження очей
5	Надмірне навантаження на зоровий апарат під час тривалого використання мобільного пристрою	Чергувати роботу на смартфоні або планшеті з роботою на моніторі більшого розміру	Скорочення тривалості впливу несприятливих факторів на органи зору
6	Надмірне навантаження на зоровий апарат через відсутність профілактичних заходів	Регулярно виконувати прості вправи для очей під час робочого дня	Зниження втомлюваності та підтримання нормального функціонування зорового апарату

Таким чином, рекомендовані заходи допоможуть знизити ризик виникнення або впливу аналізованої небезпеки та підвищити рівень безпеки праці.

Висновки до розділу

У розділі було розглянуто основні аспекти організації охорони праці на підприємстві та встановлено, що забезпечення належного рівня безпеки працівників є важливою складовою ефективної діяльності.

Як об’єкт дослідження було розглянуто робоче місце фахівця з електронного документообігу підприємства. Проведений аналіз дозволив охарактеризувати особливості трудової діяльності працівника та визначити основні небезпеки, які можуть виникати під час роботи з електронними документами, комп’ютерною технікою та мобільними пристроями.

У роботі також було розглянуто принципи ризик–орієнтованого підходу, який передбачає своєчасне виявлення потенційних небезпек та запобігання їх негативним наслідкам ще до виникнення небезпечних подій. Встановлено, що ключову роль у реалізації такого підходу відіграє процедура оцінювання ризиків, яка дозволяє визначити найбільш суттєві загрози для працівника та обґрунтувати необхідність впровадження профілактичних заходів.

Для однієї з найбільш характерних небезпек було побудовано дерево відмов та виконано аналіз причин її виникнення. За результатами проведеного дослідження запропоновано комплекс заходів, спрямованих на покращення умов праці, зниження впливу несприятливих факторів виробничого середовища та підвищення рівня безпеки працівника.

ВИСНОВКИ

У кваліфікаційній роботі успішно вирішено науково–практичне завдання з проєктування, розробки та впровадження мобільного застосунку для читання електронних документів «ReadOl» на базі операційної системи Android. Створене програмне рішення орієнтоване на підвищення когнітивного комфорту користувача, персоналізацію інтерфейсу читання та забезпечення високого рівня автономності (Offline–First). Досягнення поставленої мети стало можливим завдяки комплексному поєднанню сучасних засобів нативної Android–розробки, технологій декларативного інтерфейсу та нативних двигунів відображення даних, що дозволило створити цілісну екосистему для обробки документів різного типу. Окрім суто інженерних переваг, створене програмне рішення має виражену соціальну спрямованість: завдяки впровадженню гейміфікаційних механік воно допомагає користувачам долати залежність від швидкого споживання розважального відеоконтенту та повертатися до вдумливого читання книг, що сприяє розвиткові аналітичного мислення.

У межах першого розділу здійснено всебічний технічний аналіз ринкових аналогів (зокрема, ReadEra, PocketBook, Moon+ Reader) для виявлення їхніх обмежень у сферах інтелектуальної аналітики читання, гнучкого кастомізованого перекладу та адаптації UI під індивідуальні потреби. Обґрунтовано вибір нативної мобільної розробки мовою Kotlin, що дозволяє суттєво знизити накладні витрати оперативної пам'яті та забезпечити прямий доступ до системних API. Застосування архітектурного шаблону MVVM (Model–View–ViewModel) у поєднанні з принципами Clean Architecture забезпечило надійне розділення бізнес–логіки від представлення, стійкість інтерфейсу до поворотів екрана чи Process Death та масштабованість структури.

Другий розділ присвячено теоретичному проєктуванню інформаційного й математичного забезпечення системи. Розроблено схему локальної реляційної бази даних Room ORM (версія v20), яка містить вісім взаємопов’язаних таблиць та повністю підтримує логіку м’якого видалення (soft delete), ідентифікації джерела змін через deviceId та порівняння часових міток для побудови архітектури Offline–First. Визначено математичний апарат афінних перетворень для рендерингу фіксованих сторінок та розроблено алгоритм динамічної пагінації тексту в реальному часі за допомогою Compose TextMeasurer із вбудованими механізмами захисту від зациклювання при критичних помилках обчислення висоти рядка. Також наведено теоретичне обґрунтування хмарної синхронізації та інтеграції з сервісами Firebase.

У третьому розділі викладено опис програмної реалізації та верифікації розробленого застосунку. Описано функціонування модуля читання (Reader Module Architecture), розподіленого на 15 взаємопов'язаних файлів відповідно до принципів Clean Architecture. Проаналізовано взаємодію MuPdfEngine з нативною C++ бібліотекою MuPDF (версія fitz 1.27.1) через JNI з детермінованим очищенням деструкторів для усунення витоків пам'яті C++, а також EpubEngine з асинхронною пагінацією в фонових потоках корутин. Реалізовано захист від атак класу Path Traversal (Zip Slip) за допомогою методу normalizePath. Детально описано модуль SyncManager для взаємодії з Firebase Cloud Firestore, де захист даних реалізовано через системні правила доступу Firestore Security Rules (обмеження за UID автентифікованого користувача), шифрування транспортного каналу HTTPS та LWW–стратегію вирішення конфліктів. Результати тестування та профілювання пам'яті підтвердили відсутність Memory Leaks та стійкість системи в аварійних режимах.

Четвертий розділ містить дослідження умов охорони праці фахівця з електронного документообігу підприємства, що використовує розроблений мобільний застосунок. Проведено оцінювання професійних ризиків на основі

ризик–орієнтованого підходу (згідно з ДСТУ IEC/ISO 31010:2013). Для аналізу найкритичнішої небезпеки (погіршення зору внаслідок тривалої зорової напруги) побудовано дерево відмов та визначено її першопричини. Запропоновано комплекс інженерно–технічних та організаційних профілактичних заходів, що охоплюють раціональне освітлення робочої зони, параметри мікроклімату, дотримання гігієни зору, ергономіку офісних меблів та електробезпеку при взаємодії з комп'ютерною технікою та мобільними пристроями.

Результатом виконання кваліфікаційної роботи є такі науково–практичні результати:

- Проектування та комплексна програмна реалізація нативного Android–застосунку «ReadOl» на базі Kotlin та Jetpack Compose із використанням MVVM та Clean Architecture, що забезпечило високу швидкість відклику інтерфейсу;
- Створення гібридної системи відображення документів на основі оптимізованого MuPdfEngine для фіксованої розмітки та EpubEngine з використанням TextMeasurer для адаптивних текстових форматів;
- Реалізація відмовостійких допоміжних систем, таких як каскадний переклад з ковзним вікном префетчінгу та розумний планувальник нагадувань на основі темпу читання;
- Розробка архітектури Offline–First із локальною базою даних Room (8 таблиць) та безконфліктною двосторонньою синхронізацією з Firebase Cloud Firestore через WorkManager;
- Впровадження та експериментальна перевірка комплексу безпекових рішень, включаючи механізм Storage Access Framework, алгоритм валідації шляхів normalizePath та захист нативних ресурсів C++.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jošt G., Taneski V. State–of–the–Art Cross–Platform Mobile Application Development Frameworks: A Comparative Study of Market and Developer Trends. Informatics. 2025. Vol. 12, no. 2. Art. 45.

2. Білогуб Д., Скрипченко М., Титенко С. Якісні атрибути та архітектурні шаблони сучасних мобільних застосунків. Modern engineering and innovative technologies. 2023. № 29–01.

3. Marchenko S. Jetpack Compose: new approaches to Android UI development. Scientific Monographs. Riga, Latvia : Baltija Publishing, 2022.

4. Воротнікова З. Є. Огляд баз даних при розробці програмного забезпечення для різних операційних систем. Вісник Приазовського державного технічного університету. Серія: Технічні науки. 2025. Вип. 51.

5. Щербаков Є. В., Щербакова М. Є. Порівняльний аналіз функціональності корутин в сучасних мовах програмування. Науковий вісник Східноукраїнського національного університету імені Володимира Даля. 2024. № 27.

6. Козуб Ю., Козуб Г. Особливості розробки мультиплатформових застосунків на Kotlin. Вісник Хмельницького національного університету. Технічні науки. 2023. № 1 (317).

7. Zarichuk O. Comparative analysis of frameworks for mobile application development: Native, hybrid, or cross–platform solutions. Bulletin of Cherkasy State Technological University. 2023. Vol. 28, no. 4.

8. Faniband Y. P., Ismail I., Sidi F., Affendey L. S., Mustapha A. A Cloud–Based Distributed Platform for Secured EPUB EBOOK Contents. International Journal of Emerging Trends in Engineering Research. 2020. Vol. 8, no. 10.

9. Dahunsi F. M., Joseph A. J., Sarumi O. A., Obe O. O. Database Management System for Mobile Crowdsourcing Applications. Nigerian Journal of Technology (NIJOTECH). 2021. Vol. 40, no. 4.

10. Топольськов Є., Рибчинчук В. Удосконалена архітектура мобільного застосунку для читання електронних книг. Вісник Київського національного університету імені Тараса Шевченка. 2025.

11. Tysovskiy A., Liuta M., Немченко V. Resources for generating software code to work with databases in mobile Android devices. Scientific Monographs. Riga, Latvia : Baltija Publishing, 2025.

12. Передеренко В. Методологія міграції Android–проекту на основі перегляду проекту MVVM до архітектури Jetpack Compose та MVI. Вісник Херсонського національного технічного університету. 2025. № 4 (95), ч. 3.

13. Müller J., Noss D., Mainka C., Mladenov V., Schwenk J. Processing Dangerous Paths – On Security and Privacy of the Portable Document Format. Proceedings of the Network and Distributed System Security Symposium (NDSS). 2021.

14. Joshi A., Tirupati K. K., Chhapola A., Jain S., Goel O. Architectural Approaches to Migrating Key Features in Android Apps. Modern Dynamics: Mathematical Progressions. 2024. Vol. 1, no. 2.

15. Milojković K., Živković M., Džakula N. B. Agile Multi–user Android Application Development With Firebase: Authentication, Authorization, and Profile Management. Proceedings of Sinteza 2024.

16. Лесенко Г. Система управління охороною праці. – Режим доступу: https://pro–op.com.ua/article/177–qqq–17–m2–10–02–2017–cistema–upravlnnya–ohoronoyu–prats–suop.

17. Що таке система управління охороною праці на підприємстві? – Режим доступу: https://globynska–gromada.gov.ua/news/1721299162/.

18. Управління охороною праці на підприємстві та обов’язки роботодавця. – Режим доступу: https://oppb.com.ua/news/upravlinnya–ohoronoyu–praci–na–pidpryyemstvi–ta–obovyazky–robotodavcya.

19. Про класифікацію шкідливих та небезпечних виробничих факторів. – Режим доступу: https://oppb.com.ua/news/pro–klasyfikatsiyu–shkidlyvyh–ta–nebezpechnyh–vyrobnychyh–faktoriv.

20. У чому полягає ризикорієнтований підхід до охорони праці? – Режим доступу: https://oppb.com.ua/news/u–chomu–polyagaye–ryzykooriyentovanyj–pidhid–do–ohorony–pratsi.

21. Zdanovsk, V., Radionov M., Sepitchak V., Soltysik R. (2021). Застосування ризик–орієнтованого підходу до оцінки виробничих чинників з метою підвищення дієвості системи управління охороною праці. Вісник Львівського державного університету безпеки життєдіяльності, 24, 12–23. https://doi.org/https://doi.org/10.32447/20784643.24.2021.02.

22. Яка захистити очі під час роботи за комп'ютером. – Режим доступу: https://zorko.ua/zashhita–glaz/.

23. Безпека та захист здоров'я працівників під час роботи з екранними пристроями. – Режим доступу: https://izmail–mr.od.gov.ua/bezpeka–ta–zahyst–zdorovya–praczivnykiv–pid–chas–roboty–z–ekrannymy–prystroyamy/.

ДОДАТКИ

Додаток А
UML-діаграма інтерфейсів користувача

Таблиця та її призначення	Атрибути (поля), типи даних та їх призначення
Books – берігає метадані імпортованих документів, статистику читання та прогрес читача.	• id (Int) − первинний ключ з автогенерацією. Унікальний локальний ідентифікатор книги; • userId (String) − ідентифікатор користувача у системі Firebase; • title (String) − назва твору або назва файлу документа; • author (String) − автор твору або укладач документа; • coverUri (String?) − локальний шлях до збереженого файлу обкладинки; • totalPages (Int) − загальна кількість сторінок у документі; • fileType (String) − формат файлу документа (PDF, EPUB, FB2, TXT тощо); • currentPage (Int) − номер сторінки, на якій зупинився користувач; • genre (String) − жанр літературного твору; • status (String) − статус читання книги

	(READING, COMPLETED, WISHLIST); • rating (Int) − оцінка книги користувачем від 1 до 5; • addedTimestamp (Long) − часова мітка додавання книги до локальної бібліотеки; • lastReadTimestamp (Long) − часова мітка останнього сеансу читання цієї книги; • totalReadTimeMinutes (Int) − сумарний час читання книги в хвилинах; • isFavorite (Boolean) − ознака додавання до категорії "Обране"; • collectionName (String?) − назва колекції, до якої належить книга; • generalComment (String?) − текстова примітка користувача про книгу; • fileName (String) − назва оригінального імпортованого файлу з розширенням; • fileSize (Long) − розмір файлу книги у байтах; • series (String?) − назва серії чи циклу творів; • language (String) − мова, якою написаний текст книги;

	• description (String?) − анотація або короткий опис книги; • wordCount (Int) − загальна кількість слів у книзі; • charCount (Int) − загальна кількість символів у книзі; • updatedAt (Long) − часова мітка останньої локальної зміни запису; • deletedAt (Long?) − позначка м'якого видалення запису; • deviceId (String) − ідентифікатор пристрою, де відбулися зміни; • cloudId (String?) − ідентифікатор документа у хмарному сховищі Cloud Firestore.
notes −берігає коментарі, закладки, цитати та масиви координат виділення тексту на сторінках.	• id (Int) − первинний ключ з автогенерацією; • userId (String) − ідентифікатор користувача у системі Firebase; • bookId (Int) − зовнішній ключ до таблиці books з дією каскадного видалення; • content (String?) − текстовий вміст коментаря, створеного користувачем; • selectedText (String?) − текст

	оригінальної цитати, виділеної в книзі; • pageNumber (Int) − номер сторінки, на якій створено нотатку; • highlightColor (String) − HEX–код кольору маркерного виділення тексту; • timestamp (Long) − часова мітка створення або редагування нотатки; • type (String) − тип запису (нотатка, цитата або закладка); • style (String) − стиль підкреслення виділення (суцільний, хвилястий тощо); • title (String?) − заголовок нотатки, заданий користувачем; • startOffset (Int) − початковий індекс символу виділення на сторінці; • endOffset (Int) − кінцевий індекс символу виділення на сторінці; • serializedRects (String?) − серіалізований JSON–масив координат прямокутників виділення на сторінці PDF; • isStarred (Boolean) − позначка важливої нотатки;

	• updatedAt (Long) − часова мітка останньої локальної зміни запису; • deletedAt (Long?) − позначка м'якого видалення запису; • deviceId (String) − ідентифікатор пристрою, де відбулися зміни; • cloudId (String?) − ідентифікатор документа у хмарному сховищі Cloud Firestore.
reading_stats − зберігає добові метрики читання кожної книги для побудови аналітичних графіків.	• id (Int) − первинний ключ з автогенерацією; • userId (String) − ідентифікатор користувача у системі Firebase; • date (String) − дата у форматі ISO 8601 (РРРР–ММ–ДД); • bookId (Int) − зовнішній ключ до таблиці books з дією каскадного видалення; • pagesRead (Int) − кількість сторінок, прочитаних за добу; • wordsRead (Int) − кількість слів, прочитаних за добу; • durationSeconds (Long) − сумарний час читання книги за добу в секундах; • sessionCount (Int) − кількість читацьких

	сесій з цією книгою за добу; • timestamp (Long) − час останнього оновлення добової статистики; • updatedAt (Long) − часова мітка останньої локальної зміни запису; • deletedAt (Long?) − позначка м'якого видалення запису; • deviceId (String) − ідентифікатор пристрою, де відбулися зміни; • cloudId (String?) − ідентифікатор документа у хмарному сховищі Cloud Firestore.
streak_history − зберігає історію безперервних серій щоденного читання користувача.	• id (Int) − первинний ключ з автогенерацією; • userId (String) − ідентифікатор користувача у системі Firebase; • startDate (String) − дата початку безперервної серії (РРРР–ММ–ДД); • endDate (String) − дата завершення серії (РРРР–ММ–ДД); • durationDays (Int) − тривалість безперервної серії у днях; • isCurrent (Boolean) − ознака того, чи є серія

	активною на поточний момент; • updatedAt (Long) − часова мітка останньої локальної зміни запису; • deletedAt (Long?) − позначка м'якого видалення запису; • deviceId (String) − ідентифікатор пристрою, де відбулися зміни; • cloudId (String?) − ідентифікатор документа у хмарному сховищі Cloud Firestore.
custom_folders −зберігає назви папок, створених користувачем для класифікації книг.	• id (Int) − первинний ключ з автогенерацією; • userId (String) − ідентифікатор користувача у системі Firebase; • name (String) − назва папки, задана користувачем; • updatedAt (Long) − часова мітка останньої локальної зміни запису; • deletedAt (Long?) − позначка м'якого видалення запису; • deviceId (String) − ідентифікатор пристрою, де відбулися зміни; • cloudId (String?) − ідентифікатор документа у хмарному

	сховищі Cloud Firestore.
book_folder_cross_ref −таблиця зв'язку книг і папок, що реалізує відношення багато–до–багатьох (M:N).	• bookId (Int) − частина складеного первинного ключа. Зовнішній ключ до книги з дією каскадного видалення; • folderId (Int) − частина складеного первинного ключа. Зовнішній ключ до папки з дією каскадного видалення; • updatedAt (Long) − часова мітка останньої локальної зміни запису; • deletedAt (Long?) − позначка м'якого видалення запису; • deviceId (String) − ідентифікатор пристрою, де відбулися зміни; • cloudId (String?) − ідентифікатор документа у хмарному сховищі Cloud Firestore.
daily_gamification_records − зберігає щоденні гейміфікаційні показники користувача та статус виконання цілей.	• id (Int) − первинний ключ з автогенерацією; • userId (String) − ідентифікатор користувача у системі Firebase; • date (String) − унікальна добова дата (UNIQUE індекс разом із userId); • pagesRead (Int) −

	кількість сторінок, сумарно прочитаних за добу; • durationSeconds (Long) − загальний час читання за добу в секундах; • wordsRead (Int) − загальна кількість прочитаних за добу слів; • goalMode (String) − режим добової цілі (PAGES − сторінки, TIME − час читання); • dailyPageGoal (Int) − встановлена мета у сторінках на цю добу; • dailyTimeGoalMin (Int) − встановлена ціль у хвилинах читання на цю добу; • isGoalAchieved (Boolean) − ознака успішного виконання добової цілі користувачем; • streakThresholdPages (Int) − добовий поріг у сторінках для зарахування дня в серію; • streakThresholdTimeMin (Int) − добовий поріг у хвилинах для зарахування дня в серію; • isStreakAchieved (Boolean) − ознака того, чи виконано норму для утримання

	серії; • streakMode (String) − режим розрахунку серії; • updatedAt (Long) − часова мітка останньої локальної зміни запису; • deletedAt (Long?) − позначка м'якого видалення запису; • deviceId (String) − ідентифікатор пристрою, де відбулися зміни; • cloudId (String?) − ідентифікатор документа у хмарному сховищі Cloud Firestore.
reading_sessions −логує окремі безперервні сесії читання книг користувачем для точного трекінгу активності.	• id (Int) − первинний ключ з автогенерацією; • userId (String) − ідентифікатор користувача у системі Firebase; • bookId (Int) − зовнішній ключ до таблиці books з дією каскадного видалення; • startTimestamp (Long) − часова мітка старту сесії читання; • durationSeconds (Long) − тривалість цієї сесії читання в секундах; • wordsRead (Int) − кількість слів, прочитаних протягом сесії;

	• pagesRead (Int) − кількість сторінок, прочитаних протягом сесії; • updatedAt (Long) − часова мітка останньої локальної зміни запису; • deletedAt (Long?) − позначка м'якого видалення запису; • deviceId (String) − ідентифікатор пристрою, де відбулися зміни; • cloudId (String?) − ідентифікатор документа у хмарному сховищі Cloud Firestore.