НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

В. о. завідувача кафедри

_____ Михайло НОВОТАРСЬКИЙ
(підпис)

«__» _____2025 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: Мобільний застосунок для мультилінгвального читання книжок

Виконала : студентка __4__ курсу, групи _ІО-11_
(шифр групи)

_____Федосєєва Олена Денисівна_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____ст. викл. Васильєва М.Д._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) ___ас. Пономаренко А. М._____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент доцент _____каф. ІПІ, к.т.н. Вечерковська А.С._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2025 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ

В. о. завідувача кафедри

_____ Михайло НОВОТАРСЬКИЙ

(підпис)

«__» _____2025 р.

ЗАВДАННЯ

на бакалаврський дипломний проєкт студента

Федосєєвої Олена Денисівни

1. Тема проєкту *Мобільний застосунок для мультилінгвального читання книжок*

   керівник проєкту Васильєва Марія Давидівна, ст. викл.

   (прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

   затверджені наказом по університету від *«23» травня 2025 р. №1706-с*
2. Термін здачі студентом закінченого проєкту *«7» червня 2025 р.*
3. Вихідні дані до проєкту *технічна документація, теоретичні дані*.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

   Розділ 1. Аналіз існуючих підходів до мультилінгвального читання книг.

   Розділ 2. Проєктування мобільного застосунку для мультилінгвального читання.

Розділ 3. Реалізація мобільного застосунку.

Розділ 4. Огляд та тестування розробленого мобільного застосунку.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень): структурна схема апаратної компоненти системи, структурна схема програмної компоненти системи, блок-схема алгоритму (або UML-діаграма) функціонування програмного продукту, юзкейс-діаграма, діаграма класів, програмна документація, презентаційні матеріали.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	ас. Пономаренко А. М.		

7. Дата видачі завдання «5» лютого 2025 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	10.12.2024-15.12.2024	
2.	*Вивчення та аналіз завдання*	16.12.2024-6.02.2025	
3.	*Розробка архітектури та загальної структури системи*	7.02.2025-28.02.2025	
4.	*Розробка структур окремих підсистем*	1.03.2025-23.03.2025	
5.	*Програмна реалізація системи*	24.03.2025-21.05.2025	
6.	*Оформлення пояснювальної записки*	22.05.2025-29.05.2025	
7.	*Захист програмного продукту*	30.05.2025	

8.	*Передзахист*	6.06.2025	
9.	*Захист*	21.06.2025	

Студент-дипломник _____ Олена ФЕДОСЄЄВА

Керівник проєкту _____ Марія ВАСИЛЬЄВА

АНОТАЦІЯ

Даний дипломний проєкт присвячений розробці мобільного застосунку для мультилінгвального читання книжок, що полегшує вивчення іноземних мов. Реалізовано функціонал паралельного читання текстів на 2-4 мовах з синхронізованим відображенням абзаців, інтерактивним перекладом слів через MyMemory API, створенням персонального словника та модулем тестування знань.

Під час виконання роботи було проаналізовано існуючі рішення, спроектовано архітектуру системи та реалізовано основні модулі. Описано процес обробки текстів, забезпечення безпеки даних та взаємодії з користувачем.

Клієнтську частину розроблено для Android (Kotlin, Java), серверну – на Node.js (PostgreSQL). Безпека забезпечена JWT та bcrypt. Підготовка контенту виконується за допомогою LF_aligner та JSON-структуризації.

Ключові слова: мультилінгвальне читання, вивчення мов, Android, Kotlin, Node.js.

ANNOTATION

This diploma project is dedicated to the development of a mobile application for multilingual book reading, which facilitates the study of foreign languages. The functionality of parallel reading of texts in 2-4 languages with synchronized display of paragraphs, interactive translation of words via MyMemory API, creation of a personal dictionary and a knowledge testing module were implemented.

During the work, existing solutions were analyzed, the system architecture was designed and the main modules were implemented. The process of text processing, ensuring data security and interaction with the user was described.

The client part was developed for Android (Kotlin, Java), the server part was developed on Node.js (PostgreSQL). Security is provided by JWT and bcrypt. Content preparation is performed using LF_aligner and JSON-structuring.

Keywords: multilingual reading, language learning, Android, Kotlin, Node.js.

Довідки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпляра	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Мобільний застосунок для мультилінгвального	4		
			читання книжок			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Мобільний застосунок для мультилінгвального	86		
			читання книжок			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Мобільний застосунок для мультилінгвального	1		
			читання книжок			
			Структурна схема апаратної компоненти			
			системи			
	А4	***ІАЛЦ.467200.005 Д2***	Мобільний застосунок для мультилінгвального	1		
			читання книжок			
			Структурна схема програмної компоненти			
			системи			
	А4	***ІАЛЦ.467200.006 Д3***	Мобільний застосунок для мультилінгвального	9		
			читання книжок			
			Блок-схеми алгоритмів функціонування			
			програмного продукту			
	А4	***ІАЛЦ.467200.007 Д4***	Мобільний застосунок для мультилінгвального	1		
			читання книжок			
			Юзкейс-діаграма			
	А4	***ІАЛЦ.467200.008 Д5***	Мобільний застосунок для мультилінгвального	9		
			читання книжок			
			Діграми класів			
	А4	***ІАЛЦ.467200.009 Д6***	Мобільний застосунок для мультилінгвального	5		
			читання книжок			
			Повний перелік функціональних вимог			
	А4	***ІАЛЦ.467200.010 Д7***	Мобільний застосунок для мультилінгвального	4		
			читання книжок			
			Повний перелік нефункціональних вимог			
	А4	***ІАЛЦ.467200.011 Д8***	Мобільний застосунок для мультилінгвального	1		
			читання книжок			
			Структурна схема бази даних			
	А4	***ІАЛЦ.467200.012 Д9***	Мобільний застосунок для мультилінгвального	50		
			читання книжок			
			Лістинг програмного коду			

					ІАЛЦ.467200.001 ОА
Зм.	Лист.	№ докум.	Підпис	Дата	
Розробив		Федосєєва О.Д.			**Мобільний застосунок для мультилінгвального читання книжок** Опис альбому
Перевірив		Васильєва М.Д.			
Н. Контр.					
Затвердив					

Літ.	Арк.	Акрушів
	1	1

КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Мобільний застосунок для мультилінгвального читання книжок*»

Київ – 2025

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Це технічне завдання поширюється на розробку мобільного застосунку для мультилінгвального читання книжок. Застосунок призначений для полегшення процесу читання літератури іноземними мовами, з можливістю швидкого перекладу незнайомих слів, формування персонального словника та подальшої перевірки знань цих слів.

Областю застосування є користувачі, що вивчають іноземні мови, читають спеціалізовану або художню літературу в оригіналі, та прагнуть ефективно розширювати свій словниковий запас. Застосунок буде корисним як для самостійного навчання, так і в якості допоміжного інструменту в освітньому процесі.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання для виконання бакалаврського проєкту по освітньо-професійній програмі “Комп’ютерні системи та мережі” спеціальності 123 “Комп’ютерна інженерія”, затверджене кафедрою Обчислювальної техніки Національного технічного Університету України “Київський Політехнічний інститут імені Ігоря Сікорського”.

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою розробки є створення мобільного застосунку, що надає користувачам зручний інструмент для читання книг різними мовами з інтегрованою функцією перекладу, створення персоналізованих словників та самоперевірки.

Призначенням розробки є підвищення ефективності вивчення іноземних мов через читання, сприяння активному засвоєнню нової лексики через читання, сприяння активному засвоєнню нової лексики та покращення

розуміння іншомовних текстів. Застосунок має на меті зробити процес читання іноземними мовами більш доступним та менш трудомістким.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами для розробки даного проєкту є офіційна документація Android Developer, мов програмування Kotlin та Java, фреймворку Node.js, системи управління базами даних PostgreSQL. Також використовуються науково-технічна література, статті та публікації в мережі Інтернет, що стосуються розробки мобільних застосунків, методів обробки тексту та інтеграції API для перекладу.

5. ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до розробленого продукту

- Підтримувати аутентифікацію користувачів і збереження даних.
- Інтуїтивно зрозумілий та адаптивний інтерфейс користувача.
- Можливість отримувати доступ до каталогу книг, наданих через серверне сховище з можливістю їх паралельного читання на 2-4 мовах.
- Функція перекладу виділеного слова або фрази безпосередньо під час читання.
- Можливість додавання слів та їх перекладів до персонального словника.
- Інструменти для вивчення слів зі словника.

5.2 Вимоги до програмного забезпечення

- Клієнтська частина: розроблена на Android Studio з використанням Kotlin та Java-бібліотек. Операційна система Android версії 7.0 або вище.
- Серверна частина (backend): розроблена на Node.js.

- База даних: PostgreSQL.
- Для розробки: операційні системи Windows.

5.3 Вимоги до апаратної частини (для користувача)

- Смартфон або планшет під управлінням ОС Android згідно з вимогами до програмного забезпечення.
- Достатній обсяг внутрішньої пам'яті для встановлення застосунку, зберігання книг та словників.
- Доступ до мережі Інтернет для первинного завантаження даних, синхронізації та використання онлайн-перекладача.

6. ЕТАПИ РОЗРОБКИ

Найменування етапів дипломного проєкту	Терміни виконання
Затвердження теми проєкту	10.12.2024-15.12.2024
Вивчення та аналіз завдання	16.12.2024-6.02.2025
Розробка архітектури та загальної структури системи	7.02.2025-28.02.2025
Розробка структур окремих підсистем	1.03.2025-23.03.2025
Програмна реалізація системи	24.03.2025-21.05.2025
Оформлення пояснювальної записки	22.05.2025-29.05.2025
Захист програмного продукту	30.05.2025
Передзахист	6.06.2025
Захист	21.06.2025

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Мобільний застосунок для мультилінгвального читання книжок*»

Київ – 2025

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

МП	Машинний переклад
НММ	(Hidden Markov Models) Приховані Марковські моделі
ФВ-БП	Функціональні вимоги - бізнес-процеси
ФВ-ІК	Функціональні вимоги - інтерфейс користувача
ФВ-ДН	Функціональні вимоги - дані
ФВ-ФП	Функціональні вимоги - функціональна продуктивність
ФВ-ІН	Функціональні вимоги – інтеграція
ФВ-ЗА	Функіональні вимоги - звітность та аналітика
ФВ-НП	Функціональні вимоги - нормативно-правові
НФВ-ЗВ	Нефункціональні вимоги - зручність використання
НФВ-ПД	Нефункціональні вимоги – продуктивність
НФВ-НД	Нефункціональні вимоги – надійність
НФВ-ДП	Нефункціональні вимоги - доступність
ФВ-БЗ	Нефункціональні вимоги – безпека
НФВ-СМ	Нефункціональні вимоги – сумісність
НФВ-СП	Нефункціональні вимоги – супроводжуваність
НФВ-МШ	Нефункціональні вимоги - масштабованість
НФВ-НВ	Нефункціональні вимоги - нормативна відповідність
БД	База даних
СУБД	Система управління бази даних
ОС	Операційна система
PK	(Primary Key) Первинний ключ
FK	(Foreign Key) Вторинний (зовнішній) ключ
UI	(User interface) Інтерфейс коричтувача
UX	(User experience) Досвід користувача

API (Application Programming Interface) Прикладний програмний інтерфейс

SDK (Software development kit) Комплект для розробки програмного забезпечення

OCR (Optical Character Recognition) Оптичне розпізнавання тексту

TMX (Translation Memory eXchange) Обмін пам’яттю перекладу

XML (eXtensible Markup Language) Розширювана мова розмітки

XLS (Excel Spreadsheet) Електронна таблиця Excel

JSON (JavaScript Object Notation) Запис об’єктів JavaScript

URL (Uniform Resource Locator) Уніфікований локатор ресурсів

CRUD (Create Read Update Delete) Створення Читання Оновлення Вилучення

JWT (JSON Web Token) JSON Веб Токен

RSA (Rivest–Shamir–Adleman) Рівест-Шамір-Адлеман

HMAC (Hash-based message authentication code) Код автентифікації повідомлень на основі хешування

SHA-2 (Secure Hash Algorithm Version 2) Безпечний алгоритм хешування версія 2

ВСТУП

У контексті глобалізації та зростаючої ролі міжнародної співпраці, знання іноземних мов набуває особливої актуальності, зокрема, англійської, іспанської та китайської, як найбільш поширених у світі. Для фахівців у сфері технологій володіння іноземними мовами є невід'ємною складовою професійної компетентності, оскільки значна частина передових розробок, документації та професійних комунікацій здійснюється іноземними інформаційними мовами. Більше того, сучасні тенденції до віддаленої роботи та співпраці з міжнародними командами вимагають від IT-фахівців ефективної комунікації з іноземними замовниками та партнерами.

Окрім безпосередньої професійної користі, вивчення іноземних мов позитивно впливає на когнітивні функції людини, покращує пам'ять, концентрацію уваги та сприяє розвитку навичок вирішення проблем. Результати досліджень вказують на те, що володіння кількома мовами може відтермінувати вікові зміни когнітивних функцій, зокрема деменцію.

Процес вивчення мови традиційно охоплює чотири основні навички: аудіювання, говоріння, читання та письмо. Кожна з цих навичок взаємопов'язана з іншими, утворюючи систему, де розвиток однієї навички сприяє покращенню інших. Зокрема, читання, як процес письмового сприйняття інформації, відіграє важливу роль у розширенні словникового запасу, засвоєнні граматичних структур та розвитку навичок розуміння контексту. Регулярна практика читання іноземною мовою сприяє покращенню аудіювання, говоріння та письма.

У цьому контексті, актуальним є розробка інноваційних інструментів, що полегшують процес вивчення іноземних мов, зокрема, через читання. Мультилінгвальні книги, що представляють текст одночасно кількома мовами, є ефективним засобом для покращення розуміння іноземної мови та встановлення асоціацій між мовами. Мобільні застосунки для мультилінгвального читання відкривають нові можливості для інтерактивного

навчання, зокрема, аудіо-супровід, можливість створення нотаток та персоналізованого тренування. Крім того, мобільні застосунки забезпечують зручний доступ до широкого спектру матеріалів, що спрощує пошук та організацію навчального контенту.

Предметом дипломної роботи є програмний засіб для читання книжок із синхронним відображення тексту різними мовами. Об’єктом роботи є сам процес розробки мобільного застосунку для одночасного читання книг кількома мовами.

Метою даної дипломної роботи є розробка мобільного застосунку для мультилінгвального читання книжок, що дозволяє користувачам читати книги кількома мовами одночасно, порівнюючи тексти перекладу та вдосконалюючи власні мовні навички.

Порядок завдань, які необхідно виконати для успішного виконання поставленої мети: дослідити існуючі рішення для мультилінгвального читання, та на основі цього дослідження визначити ключові вимоги до застосунку, які привнесуть щось нове до цієї сфери, або які будуть реалізовані краще; спроектувати архітектуру системи, реалізувати функціонал відображення тексту декількома мовами, пошуку та навігації по тексту, обрати стек технологій для клієнтської та серверної частин і нарешті провести тестування.

Очікується, що розроблений мобільний застосунок забезпечить зручний та ефективний інструмент для вивчення іноземних мов, сприятиме підвищенню мотивації користувачів та покращенню їхніх мовних навичок.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО МУЛЬТИЛІНГВАЛЬНОГО ЧИТАННЯ КНИГ

1.1 Аналіз існуючих мобільних застосунків для мультилінгвального читання

Ідея створення мобільних застосунків, що дозволяють читати книги одночасно кількома мовами, не є новою. У магазинах Google Play Store та Apple App Store можна знайти різноманітні програмні рішення, що реалізують цю концепцію.

Для проведення аналізу було здійснено пошук за запитом “Multilingual readers” у Google Play Store та відібрано перші чотири непроспонсоровані застосунки, основною функцією яких є читання з паралельним перекладом.

1.1.1 Book’s Parallel Translation

Перша версія застосунку була випущена 26 жовтня 2016 року. Програма регулярно оновлюється; станом на момент аналізу, останнє оновлення до версії 3.4 відбулося 19 квітня 2025 року. За час існування застосунок було завантажено понад 1 000 000 разів, він зібрав 26,3 тисячі відгуків та отримав рейтинг 4,8 з 5 зірок [1].

Застосунок інтегрує новітні технології, зокрема, в останній версії додано підтримку моделей штучного інтелекту (ШІ), таких як ChatGPT-4, Google Gemini та Anthropic Claude. Також реалізовано функцію Bionic Reading [1].

Біонічне читання, розроблене швейцарським розробником на ім’я Ренато Касуут, має на меті полегшити читання, проводячи очі через штучні точки фіксації. [2] Зчитувач документує лише виділені початкові літери та дозволяє

мозковому центру завершити слово. Біонічне читання спрямоване на заохочення більш глибокого читання та розуміння письмового вмісту. [3]

Розробники, які використовують інструменти для перетворення тексту у біонічний, можуть налаштувати такі параметри, як кількість літер, виділених жирним шрифтом, і контраст між жирним і звичайним текстом за допомогою Bionic Reading API, що робить його настроюваним параметром для тексту в програмах. Вже існує декілька програмних платформ, які пропонують таку форму читання, такі як Bionic-Reading, Bionify, Speed Reader. [2]

Підтримується ОС Android 5.0 і вище. Застосунок пропонує платну підписку (орієнтовно 60 грн/місяць або 600 грн/рік), яка розблоковує розширений функціонал: доступ до всіх інструментів перекладу, відсутність реклами, зняття лімітів на обсяг перекладу, резервне копіювання (для збереження словника), статистика використання слів, візуальне виділення слів, що часто перекладаються, автоматичний переклад та озвучення.

Основний функціонал:

- *Інтерфейс читання:* На головному екрані надаються рекомендації книг із зазначенням мови оригіналу. При виборі книги користувач переходить у режим читання, де необхідно обрати мову перекладу. Інтерфейс є інтуїтивно зрозумілим.
- *Робота з текстом:* При натисканні на слово з'являється панель (за замовчуванням внизу екрану), яка містить: транскрипцію, можливість прослухати вимову, споріднені слова або інші форми слова (за наявності). Переклад слова надається за допомогою обраного сервісу (ChatGPT, Google, Yandex, Reverso Context, Oxford Dictionaries, Gemini тощо). Додатково пропонуються пояснення, етимологія, визначення, синоніми, переклад етимології та визначень за допомогою ШІ.
- *Переклад абзаців:* Біля кожного абзацу наявна кнопка для його повного перекладу, який відображається безпосередньо під оригінальним текстом і може бути прихований.

- *Озвучення:* Доступна функція озвучення як оригінального тексту абзацу, так і його перекладу.
- *Статистика:* Для кожної книги ведеться статистика: прогрес читання (%), час читання (загальний, сьогоднішній, приблизний залишок), швидкість читання (слів/хв), частота звернень до перекладу слів.
- *Словник:* Можливість додавати слова до особистого словника для подальшого вивчення.

Налаштування:

- Вибір мови перекладу.
- *Налаштування інтерфейсу:* кольорова схема, розмір та тип шрифту, фон, теми, режим читання (вертикальна прокрутка/посторінковий), мова інтерфейсу, яскравість, відображення перекладу абзацу замість оригіналу, роздільники між абзацами, відступи, приховування елементів інтерфейсу (індикатор закладки, кнопка перекладу абзацу).
- *Налаштування вимови:* ввімкнення/вимкнення озвучення абзаців та їх перекладів, підсвічування слів під час озвучення, автоматичне озвучення при натисканні слова, режим "оповідача" (послідовне озвучення абзаців), гучність, швидкість, вибір голосу.
- *Розширені налаштування:* використання онлайн-перекладача для дослівного перекладу, опція читання спочатку перекладу, автоматичне відображення перекладу, біонічне читання, відображення перекладу зі словника в тексті, створення та імпорт резервної копії.

1.1.2 Linga: Books with translations

Перший реліз відбувся 20 травня 2021 році. Застосунок регулярно оновлюється; станом на момент аналізу, останнє оновлення до версії 3.4.5 відбулося 10 квітня 2025 року. За час існування програму було завантажено

понад 500 000 разів, вона зібрала 4 тисячі відгуків та має рейтинг 4,8 з 5 зірок [4].

Застосунок підтримується операційними системами Android 7.0 і вище. Підтримувані формати книг: FB2, EPUB, MOBI або PDF.

Наявні варіанти платної підписки: місячна (орієнтовно 123 грн), річна (1230 грн), безлімітна (2380 грн). Підписка знімає обмеження безкоштовної версії (ліміт на додавання власних книг - 3 на місяць, денний ліміт на переклад 100 речень), відкриває доступ до показу синонімів, споріднених слів, розширює функціонал тренувань та керування словником.

Основний функціонал:

- *Переклад:* Підтримує як повний переклад тексту, так і інструменти контекстного перекладу. Надає можливість перемикатися між різними варіантами перекладу слів.
- *Робота зі словом:* При виділенні слова з'являється панель з опціями: прослухати вимову, перекласти речення цілком, переглянути приклади вживання, визначення, граматичну довідку, додати слово до словника, отримати пояснення від ШІ (напр., ChatGPT). Також доступні синоніми та споріднені слова (у платній версії).
- *Особистий словник:* Функція створення особистого словника; слова можна додавати під час читання або вручну. Можна використовувати запропоновані переклади або додавати власні. Підтримується сортування за категоріями.
- *Тренувальні модулі:* Відмінною рисою Linga є наявність тренувальних модулів для закріплення лексики. Користувач може налаштовувати параметри тренувань, встановлювати щоденні цілі та відстежувати прогрес за допомогою детальної статистики.
- *Бібліотека та мови:* Під час першого запуску застосунок запитує рідну мову (використовується для інтерфейсу), мову для вивчення та рівень володіння нею. Вибір мов для вивчення обмежений (англійська,

німецька, французька, іспанська, італійська, російська). Список доступних рідних мов значно ширший (25 мов, включаючи українську). Пропонується бібліотека книг, відсортованих за рівнем складності, з можливістю пошуку та сортування за різними критеріями. Книги необхідно завантажувати перед читанням.

1.1.3 Duoreader

У порівнянні з застосунками, які описані вище – це простіший застосунок, випущений 15 серпня 2020 року. Останнє оновлення (версія 1.4.0) датується 7 вересня 2023 року. За час існування програму було завантажено понад 10 000 разів. Підтримується ОС Android 5.1 і вище. [5]

Застосунок повністю безкоштовний. Важливою особливістю є відсутність інтеграції зі штучним інтелектом, що може бути перевагою для певної категорії користувачів, які надають перевагу професійним перекладам, простішому функціоналу, або які виступають проти використання ШІ.

Відсутність публічних відгуків, відсутність нових оновлень та ім’я “схсхсхсх” як постачальника натякає на те, що цей проєкт було запущено однією або декількома людьми для некомерційного використання.

Основний функціонал:

- *Контент та мови:* Пропонує обмежений набір книг, для яких доступні офіційні професійні переклади. Додатково доступні статті та новини від ООН. Вибір мов також обмежений: англійська, китайська (ймовірно, мандарин), французька, іспанська, японська, італійська, німецька, арабська, російська. Користувач обирає рідну мову та мову для вивчення з цього списку.
- *Інтерфейс читання:* У горизонтальній орієнтації екран розділяється: одна половина відображає оригінальний текст, інша – переклад.

- *Робота з текстом:* Натискання на речення активує його озвучення та синхронне виділення в обох текстових блоках. При довгому натисканні на слово відображається його переклад.
- *Озвучення:* Є можливість озвучення тексту з налаштуванням голосу та швидкості; можна ввімкнути автоматичне читання або вимкнути озвучення повністю.

Налаштування:

- Дозволяють змінити розмір шрифту та міжрядковий інтервал, приховати переклад або вимкнути функцію словника.

1.1.4 Beelinguapp

Перший реліз застосунку відбувся у 2016 році, і він регулярно оновлюється. Станом на момент аналізу, останнє оновлення до версії 3.218 відбулося 16 квітня 2025 року. Застосунок має понад 5 000 000 завантажень, 77 тисяч відгуків та рейтинг 4,2 з 5 зірок. Підтримується ОС Android 7.0 і вище.

Застосунок позиціонується передусім як інструмент для вивчення мов, де паралельне читання є одним з основних методів навчання.

Пропонується платна підписка (орієнтовно 919.99 грн/рік або 76.67 грн/місяць). Підписка усуває рекламу, надає доступ до всіх мовних курсів, розширених функцій словника ("розумний ШІ словник"), можливості імпорту власних текстів, створення персоналізованого плану навчання та інших преміум-функцій.

Основний функціонал:

- *Налаштування профілю:* При першому запуску користувач визначає рідну мову та мову для вивчення, обирає бажані тематики текстів, рівень володіння мовою та встановлює навчальну мету (кількість історій на тиждень).

- *Мови:* Вибір мов широкий (23 мови), включаючи не тільки найпоширеніші, а й менш розповсюджені (наприклад, філіппінську).
- *Бібліотека:* Містить рекомендовані тексти відповідно до обраних тематик та рівня складності, з можливістю фільтрації. Тексти можна додавати до обраного, читати онлайн або завантажувати.
- *Додаткові матеріали:* До кожного тексту додаються контрольні запитання на розуміння та список ключової лексики.
- *Інтерфейс читання:* За замовчуванням екран розділено горизонтально (оригінал зверху, переклад знизу). Є можливість перемикання на відображення лише одного тексту. Налаштовується розмір шрифту.
- *Озвучення:* Текст супроводжується озвученням (TTS), швидкість якого можна регулювати. Під час відтворення аудіо відповідний фрагмент тексту підсвічується. Натискання на речення виділяє його синхронно в обох мовних версіях.
- *Робота зі словом:* Довге натискання на слово викликає вікно з його перекладом.
- *Вивчення лексики:* Збережені у словнику слова можна вивчати за допомогою інтерактивних карток та вправ (наприклад, заповнення пропусків).

1.2 Методи синхронізації текстів

1.2.1 Джерела перекладу та виклики синхронізації

Аналіз існуючих застосунків (розглянутий у підрозділі 1.1) виявив два основні підходи до отримання перекладеного тексту, що використовуються в системах мультилінгвального читання:

- Використання наперед підготовлених професійних перекладів.

- Застосування машинного перекладу (МП) в режимі реального часу або на етапі підготовки контенту.

Кожен із цих підходів має специфічні особливості та виклики стосовно подальшої синхронізації текстів. Синхронізація професійних перекладів є значно складнішим завданням з точки зору попередньої обробки даних з таких причин [6]:

- *Недослівність перекладу:* Професійні перекладачі рідко вдаються до дослівного перекладу; вони часто виступають інтерпретаторами змісту. Це може проявлятися у тому, що одне речення оригіналу перекладається кількома реченнями, або ж навпаки — кілька речень оригіналу об'єднуються в одне речення перекладу. Перекладачі використовують стилістичні та творчі рішення для оптимальної передачі змісту та атмосфери оригіналу, що призводить до суттєвих розбіжностей у структурі речень та абзаців.
- *Структурні розбіжності:* Можлива відсутність деяких фрагментів оригіналу в перекладі (наприклад, через культурні адаптації) або, навпаки, наявність додаткових пояснень чи коментарів від перекладача, яких немає в оригіналі.
- *Збереження форматування:* Необхідність коректного зіставлення та збереження оригінальної структури абзаців під час вирівнювання додає процесу складності.

Отже, використання професійних перекладів вимагає застосування більш складних алгоритмів вирівнювання та, потенційно, етапу попередньої обробки та нормалізації текстів. Синхронізація тексту, отриманого шляхом МП, зазвичай є простішою, оскільки сучасні системи МП часто прагнуть зберігати відповідність на рівні речень, хоча якість самого перекладу може поступатися професійному.

1.2.2 Поняття синхронізації текстів

Синхронізація текстів, також відома як вирівнювання (text alignment), є процесом встановлення відповідностей між одиницями (сегментами) тексту оригіналу та його перекладу. Такими одиницями можуть бути документи, абзаци, речення або окремі слова. Процес зазвичай включає сегментацію обох текстів та подальше зіставлення відповідних сегментів за допомогою статистичних та/або лінгвістичних алгоритмів [7].

Простий приклад перекладу речення з іспанської на англійську з урахуванням вирівнювання показано на рис. 1.1 i та j позначають індекси іспанської і англійської мови відповідно. При позначення вирівнювання як a, отримаємо, що вирівнювання слів задаються функцією $a: j \rightarrow i$. У наведеній парі речень вирівнювання має вигляд a: {1 → 1, 2 → 3, 3 → 2}. Кількість слів в іспанському та англійському реченні становить відповідно l_f та l_e. Токен 'NULL' додається до іноземного речення, щоб забезпечити можливість врахувати англійські слова, які не мають відповідного слова в іноземному реченні.

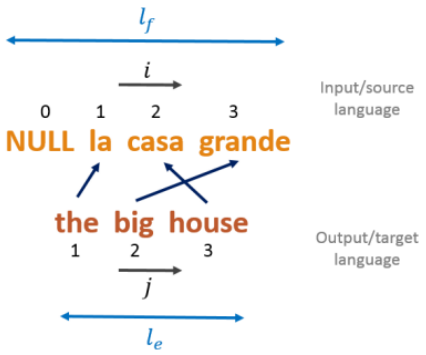


Рис 1.1 – Пара вирівняних речень англійською та іспанською мовами [9]

1.2.3 Моделі IBM

У сфері статистичного перекладу, найбільш відомими є модулі вирівнювання IBM. Ці моделі можна розглядати як послідовність етапів з наростаючою складністю, які використовуються для навчання моделей перекладу та вирівнювання — починаючи з розрахунку ймовірностей

лексичного відповідника і закінчуючи моделями, що враховують порядок слів і їх повторення.

IBM запропонувала п’ять ключових моделей, кожна з яких деталізує процес перекладу, додаючи нові аспекти до попередніх: [8]

- *Модель 1:* Базується виключно на ймовірностях лексичного перекладу (ймовірність того, що слово вихідного тексту є перекладом слова цільового тексту), ігноруючи позиції слів.
- *Модель 2:* Додає залежність від абсолютних позицій слів у реченнях, моделюючи ймовірність вирівнювання слова на позиції *i* зі словом на позиції *j*.
- *Модель 3:* Вводить поняття "плідності" (fertility) – скільки слів у цільовому реченні генерує кожне слово вихідного речення (включаючи нульову плідність).
- *Модель 4:* Враховує відносні позиції вирівняних слів для кращого моделювання локальних переміщень слів та фраз.
- *Модель 5:* Виправляє деякі недоліки попередніх моделей, зокрема, запобігає "дефіцитним" вирівнюванням (коли кілька слів цільового речення намагаються вирівнятися на одну й ту саму позицію вихідного тексту).

Навчання цих моделей зазвичай відбувається ітеративно за допомогою EM-алгоритму (Expectation-Maximization).

1.2.4 HMM модель

Використання Прихованих Марковських Моделей (HMM) для вирівнювання слів є класичним підходом у статистичному машинному перекладі. Основна ідея полягає в тому, щоб розглядати процес вирівнювання як генеративний процес, де кожне слово в цільовому реченні (наприклад,

англійському) або "випромінюється" певним словом (або NULL-токеном) у вихідному реченні (наприклад, іноземному).

Ключова особливість підходу полягає в тому, що ймовірність вирівняти певне слово в цільовому реченні з певним словом у вихідному реченні явно залежить від того, куди було вирівняне попереднє слово цільового речення. Тобто, модель враховує позицію попереднього вирівнювання при визначенні поточного. Це створює ланцюгову залежність у процесі вирівнювання.

Ймовірності перекладу слів, отримані за допомогою HMM-моделі, виявилися співставними з результатами, отриманими за допомогою іншого популярного підходу — моделі суміші вирівнювань (mixture alignment model). Однак, при аналізі саме позицій вирівнювань виявляється, що HMM-модель генерує значно "плавніші" вирівнювання. Це означає, що сусідні слова в одному реченні частіше вирівнюються з сусідніми словами в іншому. Така властивість може бути особливо корисною для мов на кшталт німецької, де складні слова (композити) часто відповідають кільком окремим словам у вихідній мові, що вимагає послідовного вирівнювання. [10]

1.3 Огляд форматів електронних книг та їх підтримка в мобільних застосунках

В електронних книг є велике різноманіття форматів, кожен з яких має свої певні характериситики, які треба брати до уваги, при розробці застосування для читання. Наприклад, треба врахувати рівень інтерактивності та графіки, сумісність формату з різними типами пристроїв.

1.3.1 Формат .txt

Історично формат TXT став першим файловим форматом, який використовували для зберігання електронних документів. Завдяки своїй простоті та зручності, він і досі залишається популярним форматом для

електронних книг. Однією з головних переваг є його універсальність — файли цього типу можна відкривати на будь-якому пристрої.

Втім, формат TXT має й свої обмеження. Він не підтримує оформлення тексту, зображення чи інші мультимедійні елементи, тому не підходить для складних документів або книжок з ілюстраціями.

Працюючи з текстовими файлами у форматі TXT, важливо звертати увагу на кодування символів. Від кодування залежить, як саме текст відображатиметься на різних пристроях. Різні мови можуть вимагати різного кодування, і якщо не врахувати цей момент, текст може відображатися некоректно або спотворено.

1.3.2 Формат .pdf

Одним із найпоширеніших форматів електронних книг сьогодні є PDF (Portable Document Format), розроблений компанією Adobe. Його головна перевага — збереження оригінального вигляду сторінок, тобто текст, зображення та інші елементи відображаються саме так, як задумав автор або дизайнер.

Формат PDF є універсальним — його можна відкрити майже на будь-якому пристрої без потреби у спеціальному програмному забезпеченні. Його також зручно використовувати для друку та розповсюдження. До того ж PDF підтримує базовий захист авторських прав (DRM), що дозволяє видавцям обмежувати несанкціоноване копіювання та поширення. Для цього Adobe створила спеціальну платформу — Adobe Content Server.

Однак у PDF є й недоліки. Формат не підтримує "переобтікання" тексту (reflowable content), тому вміст не адаптується під розміри екрана, що ускладнює читання на смартфонах чи невеликих пристроях. Також PDF-файли часто займають більше місця і завантажуються повільніше, ніж файли у форматах EPUB чи MOBI. До того ж, PDF обмежено підтримує інтерактивні елементи, наприклад, відео або аудіо.

Для покращення досвіду читання, Adobe створила спеціальний додаток — Adobe Digital Editions (ADE), який призначений для перегляду електронних книг.

1.3.3 Формат .epub

Формат EPUB (Electronic Publication) ідеально підходить для читання на екранах різного розміру, оскільки текст у ньому автоматично підлаштовується під розміри шрифтів та дисплея. Це робить його зручним для використання на смартфонах, планшетах та інших пристроях. EPUB особливо добре підходить для книг з великою кількістю зображень — наприклад, коміксів, кулінарних книг або графічних романів.

Формат був розроблений у 2007 році організацією International Digital Publishing Forum (IDPF) як відкритий стандарт для електронних публікацій. Він базується на XML, XHTML та CSS, що дозволяє додавати не лише текст та зображення, а й мультимедійний контент — зокрема, відео чи аудіо. Існують також покращені версії EPUB, які дозволяють відтворювати такі медіа безпосередньо в застосунку для читання, без необхідності відкривати сторонні програми чи браузери. EPUB підтримує зручну навігацію: зміст, закладки, коментарі.

Попри переваги, у EPUB є й певні обмеження. Йому важко обробляти складні макети, як-от у технічних посібниках, або великі обсяги мультимедійного контенту. Захист авторських прав (DRM), реалізований у цьому форматі, зазвичай є достатнім для більшості завдань, однак деяким видавцям може знадобитися більш потужне рішення для захисту вмісту.

1.3.4 Формат .mobi, .azw, .azw3

Формат MOBI (Mobipocket) був розроблений французькою компанією Mobipocket у 2000 році для мобільних пристроїв з обмеженою пам’яттю. Він підтримував текст із форматуванням, зображення, закладки, нотатки та

синхронізацію між пристроями. У 2005 році компанію придбав Amazon і почав використовувати MOBI як основний формат для Kindle.

Хоча MOBI дещо схожий на EPUB, він не є відкритим стандартом і не підтримує вбудовані відео- чи аудіофайли. Проте цей формат добре відображає графіку й таблиці, а його просте форматування зручне для видавців, які прагнуть точного контролю над виглядом книги на різних пристроях.

Щоб розширити можливості MOBI, Amazon створив два власні формати — AZW (2007) та вдосконалений AZW3 (або KF8, 2011). Обидва формати підтримують відео, аудіо, покращене форматування, шрифти та складні макети. AZW3, зокрема, ідеально підходить для технічної літератури або підручників завдяки розширеним типографічним можливостям.

Головна перевага AZW та AZW3 — їхня повна інтеграція з Kindle, що дозволяє зручно синхронізувати дані між усіма пристроями користувача. Однак ці формати є закритими і використовуються переважно в екосистемі Amazon.

1.3.5 Формат .html

HTML часто використовується для створення електронних книг завдяки своїй гнучкості та простоті. Його ключова перевага — можливість вбудовувати мультимедійний контент, як-от зображення, аудіо та відео. Це робить HTML зручним для книг із великою кількістю візуальних матеріалів або для навчальних посібників з інтерактивними елементами. Крім того, HTML-документи можна переглядати на будь-якому пристрої через браузер, що забезпечує широку доступність.

Втім, у порівнянні зі спеціалізованими форматами на кшталт EPUB чи MOBI, HTML має обмеження. Насамперед, складно забезпечити однакове відображення на різних екранах, адже контроль над макетом і стилями обмежений. Також HTML не підтримує деякі важливі функції електронних книг, як-от адаптивне (reflowable) відображення тексту, закладки чи анотації.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі проаналізовано існуючі мобільні застосунки для мультилінгвального читання, методи синхронізації текстів та поширені формати електронних книг. Встановлено, що застосунки значно різняться за функціоналом, підходами до перекладу та використанням технологій. Синхронізація паралельних текстів, особливо професійних перекладів, є нетривіальним завданням, для вирішення якого застосовуються статистичні моделі. Вибір та підтримка форматів е-книг (з урахуванням їх адаптивності та можливостей) є ключовим фактором при розробці таких застосунків.

РОЗДІЛ 2
ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ МУЛЬТИЛІНГВАЛЬНОГО ЧИТАННЯ

2.1 Визначення функціональних та нефункціональних вимог до застосунку

2.1.1 Визначення, особливості та типи функціональних вимог

Функціональні вимоги визначають деталі та інструкції як має працювати і поводитися програмне забезпечення. Функціональні вимоги застосовуються на етапі розробки програмного забезпечення, щоб гарантувати його простоту і працездатність [14]. Вони також визначають властивості і атрибути системи [15].

Існує кілька методів написання функціональних вимог, але найпоширенішим методом є побудова історій користувача та використання форматів історій користувача: як ___, я хочу мати можливість ___ так, щоб ___. [17]

Типи функціональних вимог [16]:

- *Вимоги до бізнес-процесів:* Ці вимоги окреслюють конкретні бізнес-процеси та робочі потоки, які має підтримувати система. Вони деталізують послідовність кроків, необхідні дії та взаємодії для виконання різноманітних завдань у межах системи.
- *Вимоги до інтерфейсу користувача (UI):* Визначають, як має бути спроектований інтерфейс системи та як користувачі будуть з ним взаємодіяти. Сюди входять специфікації щодо макету екранів, навігації, елементів введення (кнопок, полів тощо) та загального візуального оформлення.

- *Вимоги до даних:* Описують елементи даних, їхні структури та взаємозв'язки, які система повинна вміти збирати, зберігати, обробляти та представляти. Вони уточнюють типи даних, правила їх перевірки (валідації), а також будь-які обмеження чи бізнес-правила, що стосуються даних.
- *Вимоги до функціональної продуктивності:* Встановлюють очікувані характеристики продуктивності системи: час відгуку на дії користувача, пропускну здатність (кількість операцій за одиницю часу), ефективність використання ресурсів (пам'ять, процесор) та загальну потужність (максимальне навантаження). Мета – гарантувати, що система зможе впоратися з прогнозованим обсягом роботи та ефективно виконувати свої функції.
- *Вимоги до безпеки:* Окреслюють необхідні заходи та механізми контролю для захисту системи та даних від несанкціонованого доступу, зламів чи неналежного використання. Це включає вимоги до автентифікації (перевірка особи користувача), авторизації (визначення прав доступу), шифрування даних, ведення журналів аудиту (запис важливих подій) та відповідності стандартам безпеки.
- *Вимоги до інтеграції:* Визначають, як система має взаємодіяти та обмінюватися даними з іншими, зовнішніми системами або компонентами. Вони описують необхідні інтерфейси (API), протоколи зв'язку та формати даних для забезпечення безшовної взаємодії та сумісності.
- *Вимоги до звітності та аналітики:* описують можливості системи щодо створення звітів, аналізу даних та візуалізації результатів для отримання цінної інформації (інсайтів). Вони конкретизують, які саме звіти, показники (метрики), діаграми чи інформаційні панелі (дашборди) має надавати система для підтримки прийняття рішень та моніторингу процесів.

- *Нормативно-правові вимоги (відповідальність):* Охоплюють усі юридичні, регуляторні чи галузеві стандарти та норми, яких система повинна дотримуватися. Вони гарантують, що система відповідає законодавчим та нормативним зобов'язанням, наприклад, щодо захисту персональних даних (GDPR), доступності для людей з обмеженими можливостями, чи специфічних правил для певної індустрії.

Важливо усвідомлювати, що функціональні вимоги не існують ізольовано. Вони нерозривно пов'язані з очікуваннями щодо якості (нефункціональними вимогами), стратегічними бізнес-цілями та чинними бізнес-правилами.

Окрім ідентифікації, процес роботи з функціональними вимогами передбачає їх подальший аналіз, який включає пріоритезацію (визначення важливості) та трасування – встановлення відповідності між вимогами та конкретними компонентами програмного забезпечення.

Ключовим етапом є валідація вимог, мета якої – перевірити зібрану специфікацію на повноту, коректність та відсутність суперечностей. Цей процес має підтвердити, чи є наявна інформація достатньою основою для початку розробки продукту, здатного задовольнити бізнес-цілі та потреби користувачів [15].

2.1.2 Визначення, особливості та типи нефункціональних вимог

Нефункціональні вимоги визначають якість роботи системи, а не її конкретні функції. Вони встановлюють стандарти щодо таких аспектів, як зручність використання, надійність та ефективність, що є ключовими для позитивного користувацького досвіду [18].

Нерідко різні нефункціональні вимоги можуть суперечити одна одній або створювати необхідність компромісу. Наприклад, вимога щодо максимального рівня безпеки системи (що може вимагати складних процедур автентифікації чи

шифрування) може вступати в конфлікт з вимогою щодо високої зручності використання (яка прагне до мінімізації перешкод для користувача). Знаходження балансу між такими вимогами є важливим завданням на етапі проєктування [19].

НФВ можуть бути сформульовані двома способами [19]:

- Якісні НФВ: Описують бажані властивості системи в описовій, часто суб'єктивній формі (наприклад, "Система повинна бути інтуїтивно зрозумілою та легкою у використанні"). Їх складніше виміряти об'єктивно.
- Кількісні НФВ: Визначають конкретні, вимірювані показники чи метрики, яким має відповідати система (наприклад, "Час відгуку системи на введення користувача не повинен перевищувати 2 секунди" або "Система повинна обробляти не менше 100 транзакцій на секунду").

Типи нефункціональних вимог [20]:

- *Безпека (Security):* Вимоги до захисту системи та даних від загроз, несанкціонованого доступу; відповідність стандартам безпеки.
- *Ємність (Capacity):* Потреби системи у зберіганні даних (поточні та майбутні); вимоги до обробки обсягів даних.
- *Сумісність (Compatibility):* Вимоги до апаратного забезпечення, підтримуваних операційних систем та їх версій.
- *Надійність (Reliability) та Доступність (Availability):* Очікуваний рівень безвідмовної роботи; вимоги до часу доступності системи для користувачів.
- *Супроводжуваність (Maintainability) та Керованість (Manageability):* Легкість внесення змін, виправлення помилок та адміністрування системи. Може включати відновлюваність (Recoverability).

- *Масштабованість (Scalability):* Здатність системи справлятися зі зростанням навантаження (наприклад, кількість користувачів, даних) без втрати продуктивності.
- *Зручність Використання (Usability):* Простота та ефективність взаємодії користувача з продуктом; якість користувацького досвіду.
- *Продуктивність (Performance):* Швидкодія системи: час відгуку на дії користувача, швидкість виконання операцій.
- *Нормативна відповідність (Regulatory):* Вимоги щодо дотримання галузевих стандартів, законодавчих та регуляторних норм.
- *Вимоги до середовища (Environmental):* Специфікація умов (апаратних, програмних), в яких система має коректно функціонувати.

2.1.3 Ключові функціональні та нефункціональні вимоги до мобільного застосунку

Повні списки функціональних та нефункціональних вимог до мобільного застосунку для мультилінгвального читання детально представлені у Додатках 6 та 7 відповідно. Ці переліки описують цілісний, иаксимально багатофункціональний та високоякісний програмний продукт, призначений для зручного читання, вивчення мов та ефективного керування персональним словником.

Однак реалізція всього функціоналу та досягнення всіх нефункціональних показників є надзвичайно ресурсомісткою задачею. Враховуючи часові обмеження, повна імплементація всіх вимог є неможливою.

Нижче наведено короткий огляд вимог, з розподілом вимог на критично важливі та ті, які є другорядними і які можуть бути розглянуті у випадку достатньої кількості часу, або будуть винесені як ідеї на майбутній розвиток.

Критичні вимоги:

- Керування акаунтом:

- ФВ-БП-01: Реєстрація та автентифікація користувача (Email/пароль).
- ФВ-ДН-01: Зберігання даних користувачів (безпечно, на сервері).
- ФВ-БЗ-01, ФВ-БЗ-02: Хешування паролів, перевірка автентифікації.

- Каталог та вибір Книги:
  - ФВ-БП-02: Перегляд каталогу доступних книг (простий список).
  - ФВ-БП-03: Можливість обрати книгу з каталогу для читання.
  - ФВ-ІК-01: Базовий екран каталогу (список назв, авторів, доступних мов).
  - ФВ-ДН-02: Зберігання каталогу книг (на сервері).
- Читання:
  - ФВ-БП-04: Одночасне відображення тексту обраними мовами із синхронізацією на рівні абзаців.
  - ФВ-ІК-02: Інтерфейс читання з паралельними колонками/розділеним екраном.
- Робота зі словником:
  - ФВ-БП-05: Виділення слова/фрази в одному з текстів.
  - ФВ-БП-06: Додавання виділеного слова/фрази до персонального словника.
  - ФВ-БП-07: Перегляд персонального словника (простий список).
  - ФВ-ІК-05: Візуальний відгук виділення.
  - ФВ-ІК-06: Простий інтерфейс (діалог) для додавання слова.
  - ФВ-ІК-07: Базовий екран словника.
  - ФВ-ДН-04: Зберігання словника на сервері (оригінал, мова, переклад користувача).
- Тестування:
  - ФВ-БП-08: Можливість ініціювати тестування.
  - ФВ-БП-09: Проведення тестування.
  - ФВ-БП-10: Відображення результату.

- ФВ-ІК-08, ФВ-ІК-09: Простий інтерфейс тесту та результатів.

- Інфраструктура та Безпека:
  - ФВ-БЗ-03: Використання HTTPS для зв'язку.
  - ФВ-ІН-01: Реалізація клієнт-серверної взаємодії через API.
- Основні Нефункціональні Аспекти:
  - НФВ-ДП-02: Коректна обробка відсутності мережі (повідомлення користувачу).
  - НФВ-БЗ-01, НФВ-БЗ-02, НФВ-БЗ-03: Реалізація критичних вимог безпеки.
  - НФВ-СМ-01, НФВ-СМ-02: Забезпечення роботи на основній цільовій платформі (Android) та базових розмірах екранів.
  - НФВ-НД-01: Досягнення базової стабільності застосунку.

Вимоги, націлені на майбітній розвиток:

- Офлайн-Режим: ФВ-ДН-08 (кешування), НФВ-ДП-01 (робота офлайн). Вимагає локальної БД та механізмів синхронізації.
- Розширене Читання: Синхронізація на рівні слів, зручний вибір мов (ФВ-ІК-03), жести масштабування (ФВ-ІК-04).
- Розширений Словник: Збереження контексту, посилань на книгу (ФВ-ДН-07), інтеграція з API перекладу (ФВ-ІН-03).
- Розширене Тестування: Різні типи тестів, SRS-алгоритми, детальна історія та статистика (ФВ-ДН-06, ФВ-ЗА-01, ФВ-ЗА-02, ФВ-ЗА-03).
- Додатково: Розширені налаштування (ФВ-ДН-05), зовнішня автентифікація (ФВ-ІН-02), аналітика для адміністратора (ФВ-ЗА-04).
- Юридичні Аспекти: ФВ-НП-* (повне ліцензування, відповідність GDPR). Для MVP передбачається використання контенту з відкритих джерел/суспільного надбання та базова політика конфіденційності.
- Інші НФВ: Досягнення всіх цільових метрик продуктивності (НФВ-ПД), повної доступності (НФВ-ЗВ-05), складного захисту контенту

(НФВ-БЗ-04), масштабованості серверу (НФВ-МШ-02), повної супроводжуваності (НФВ-СП-02) та ін.

2.2 Проєктування архітектури системи

На цьому етапі проєктування визначається загальна структура системи, її основні компоненти, їх взаємодія, а також технології, що будуть використані для реалізації. Архітектура мобільного застосунку для мультилінгвального читання повинна забезпечувати надійність, масштабованість, зручність підтримки та високу продуктивність.

2.2.1 База даних: структура та схеми

База даних (БД) є ключовим компонентом серверної частини, що відповідає за зберігання всієї інформації застосунку. Враховуючи структурований характер даних (користувачі, тексти, слова, переклади, тести) та необхідність забезпечення цілісності даних через зв'язки між сутностями, реляційна модель даних є найбільш доцільним вибором. Як система управління базами даних (СУБД) може бути використана PostgreSQL або MySQL, які є потужними, надійними та добре масштабованими рішеннями з відкритим кодом.

Основні сутності, що будуть зберігатися в системі, включають інформацію про користувачів (Users), каталог доступних книг (Books), бібліотеки користувачів (User_Library), персональні словники (User_Dictionary), а також сутності для зберігання результатів тестування (Test_Scores та Test_Answers). Ключовими атрибутами для сутності Users є ідентифікатор, email та хеш пароля. Для Books – ідентифікатор, назва, автор та посилання на контент. User_Dictionary зберігатиме оригінальне слово, його переклади та зв'язок з користувачем.

Візуальна структура бази даних, що ілюструє основні сутності та зв'язки між ними, представлена у вигляді ER-діаграми у Додатку 8.

Усі зв'язки між основними сутностями, детально відображені на ER-діаграмі є типу «один-до-багатьох»:

- Один користувач (Users) може мати багато записів у своїй бібліотеці (User_library).
- Одна книга (Books) може бути додана до бібліотек багатьох користувачів (через User_library).
- Один користувач (Users) має багато слів у своєму словнику (User_dictionary).
- Один користувач (Users) має багато результатів тестування (Test_scores).
- Один результат тестування (Test_scores) має багато відповідей на питання (Test_answers).
- Одне слово зі словника (User_dictionary) може бути використане у багатьох відповідях тестування (Test_answers).

Запропонована концептуальна модель бази даних, візуалізована у Додатку 8, слугує вихідною точкою для розробки серверної частини застосунку. Важливо зазначити, що відповідно до принципів ітеративної розробки, цей дизайн не є остаточним. У ході імплементації функціоналу та тестування можуть виникнути потреби в уточненні атрибутів, зв'язків або навіть структури таблиць для забезпечення оптимальної продуктивності та відповідності фінальним вимогам.

2.2.2 Серверна та клієнтська частина

Враховуючи потребу в зберіганні даних користувача (словники, прогрес читання, результати тестів) та потенційну необхідність синхронізації між пристроями або доступу до розширених функцій (наприклад, централізованої бази перекладів), доцільно обрати клієнт-серверну архітектуру. [30]

Ця архітектура розділяє функціональність на дві основні частини:

- *Клієнтська частина:* Мобільний застосунок, встановлений на пристрої користувача (смартфон, планшет). Відповідає за весь інтерфейс користувача (UI), обробку взаємодії з користувачем (UX), візуалізацію текстів та іншого контенту, формування запитів до сервера та обробку його відповідей.
- *Серверна частина (Backend):* Відповідає за бізнес-логіку, управління даними (зберігання, оновлення, видалення), автентифікацію та авторизацію користувачів, а також надання API (Application Programming Interface) для взаємодії з клієнтською частиною.

Основні функції клієнтської частини:

- *Представлення даних:* Відображення екранів реєстрації/входу, каталогу книг, інтерфейсу для читання (з паралельним відображенням мов), персонального словника, інтерфейсів тестування та результатів.
- *Взаємодія з контентом:* Завантаження списку книг та їх текстів з сервера, обробка виділення слів/фраз користувачем, ініціювання запитів на додавання слів до словника.
- *Управління словником та тестуванням:* Відображення словника (отриманого з сервера), ініціювання процесу тестування, відображення питань, прийом відповідей користувача та відображення фінальних результатів (отриманих з сервера).
- *Комунікація з сервером:* Формування та надсилання HTTP/S запитів до серверного API для автентифікації, отримання каталогу та текстів, роботи зі словником (збереження, отримання), проведення тестування та отримання результатів.
- *Управління сесією:* Збереження токену доступу після успішного входу для автентифікації наступних запитів.

Основні функції серверної частини (backend):

- *API (Application Programming Interface):* Надання RESTful API для взаємодії з мобільним клієнтом. API визначає ендпоінти (endpoints) для всіх операцій (напр., POST /api/auth/register, POST /api/auth/login, GET /api/texts, POST /api/texts, GET /api/vocabulary тощо). Дані передаються у форматі JSON.
- *Автентифікація та авторизація:* Перевірка облікових даних користувача (логін/пароль), видача токенів доступу (напр., JWT - JSON Web Tokens) для авторизації наступних запитів.
- *Бізнес-логіка:* Обробка запитів від клієнта, валідація даних, реалізація логіки додавання слів, генерації тестів (наприклад, на основі алгоритму Spaced Repetition з використанням familiarity_score та last_reviewed_at), розрахунок результатів.
- *Управління БД:* Взаємодія з базою даних (PostgreSQL/MySQL) для збереження, читання, оновлення та видалення даних користувачів, текстів, словників, тестів.

Взаємодія між клієнтською і серверною частинами відбувається за протоколом HTTP/S. Клієнт надсилає запити до визначених ендпоінтів API сервера. Запити, що вимагають авторизації, містять токен доступу (напр., у заголовку Authorization: Bearer <token>). Сервер обробляє запит, взаємодіє з БД або іншими сервісами, та повертає відповідь клієнту у форматі JSON, що містить або запитані дані, або статус операції (успіх/помилка).

2.3 Вибір стеку технологій для розробки

Для розробки мобільного застосунку розглянемо декілька варіантів стеків технологій.

2.3.1 Native Android Application

Реалізує мобільний додаток, який встановлюється та запускається безпосередньо на пристроях під управлінням операційної системи Android. Розробка ведеться за допомогою Android SDK.

- *Клієнтська частина:* Android: Java або Kotlin. Java була історично першою офіційною мовою для розробки на Android, надаючи об'єктно-орієнтований підхід та можливість роботи на віртуальній машині Java (JVM). Вона має величезну екосистему та зрілу базу коду. Однак, Kotlin, розроблений JetBrains і зараз рекомендований Google, набув величезної популярності завдяки своїй лаконічності, вбудованій безпеці щодо null-значень (що зменшує кількість помилок NullPointerException), підтримці функціональних підходів та корутин для зручної асинхронності. Kotlin повністю сумісний з Java та її бібліотеками, дозволяючи поступово впроваджувати його в існуючі Java-проєкти та ефективно створювати нові, більш надійні та підтримувані Android-додатки. [21]
- *Серверна частина:* Node.js + Express.js. Відповідає за бізнес-логіку, яка не виконується на клієнті (наприклад, автентифікація користувачів, управління каталогом книг), взаємодію з базою даних (збереження та отримання даних про користувачів, книги, словники), а також надання API для клієнтської частини. Express.js спрощує створення веб-сервера та обробку HTTP-запитів для реалізації REST API. [22]
- *База даних:* PostgreSQL. Виступає як постійне сховище даних для серверної частини. Зберігає структуровану інформацію про користувачів, книги, їхній контент, записи словників, результати тестів тощо. Реляційна модель добре підходить для даних із чіткими зв'язками (користувач -> словник, книга -> контент). PostgreSQL відома

своєю надійністю, відповідністю стандартам SQL та можливістю роботи зі складними запитами. [23]

- *Інше:* API — REST. Representational State Transfer (REST) - це архітектурний стиль для створення веб-сервісів. Визначає правила та угоди, за якими клієнтська частина (Android-додаток) та серверна частина (Node.js) обмінюються даними через мережу (зазвичай за допомогою протоколу HTTP). Клієнт надсилає запити (GET, POST, PUT, DELETE) на певні URL (endpoints) сервера, а сервер повертає дані (часто у форматі JSON). [24]

Переваги:

- Висока продуктивність та стабільність.
- Багато готових бібліотек для інтеграції.
- Максимальна адаптація під Android-пристрої.

Недоліки:

- Мобільний додаток доступний тільки для Android.

2.3.2 Native iOS Application

Реалізує мобільний додаток, призначений для роботи на пристроях iPhone та iPad під управлінням iOS. Розробка ведеться за допомогою iOS SDK та інструментів Apple (Xcode).

- *Клієнтська частина:* iOS: Swift. Swift – це сучасна, потужна та інтуїтивно зрозуміла мова програмування, створена Apple у 2014 році як наступник Objective-C для розробки додатків під усі свої платформи (iOS, macOS, watchOS, tvOS). Вона спроектована з акцентом на безпеку, зокрема завдяки використанню опціональних типів для коректної роботи з можливими відсутніми значеннями, та на високу продуктивність, що досягається завдяки компіляції в оптимізований нативний код. Swift має чистий синтаксис, що полегшує читання та

написання коду, підтримує сучасні парадигми програмування та тісно інтегрована з інструментами розробки Apple, такими як Xcode, та фреймворками Cocoa і Cocoa Touch. [26]

- *Серверна частина:* Node.js + NestJS (структурований бекенд). NestJS побудований поверх Express.js (або Fastify), але надає більш високорівневу та структуровану архітектуру, часто натхненну Angular. Він активно використовує TypeScript. NestJS сприяє кращій організації коду, модульності та тестуванню завдяки чіткій структурі (модулі, контролери, сервіси), що може бути корисним для складніших бекендів. [25]
- *База даних:* PostgreSQL.
- *Інше:* API — REST.

Переваги:

- Висока продуктивність на пристроях Apple.
- Гарна інтеграція з iOS-сервісами (iCloud, Siri, Notification Center).

Недоліки:

- Обмеження в аудиторії (тільки пристрої Apple).

2.3.3 Android + окремий сервер на Java

Цей варіант поєднує нативний Android-клієнт із серверною частиною, реалізованою на платформі Java за допомогою популярного фреймворку Spring Boot.

- *Клієнтська частина:* Android: Kotlin.
- *Серверна частина:* Java (Spring Boot Framework). Spring Boot значно спрощує створення автономних веб-додатків на базі Spring Framework, забезпечуючи автоконфігурацію та мінімізуючи необхідність написання шаблонного коду. Spring Boot є стандартом де-факто для

створення ентерпрайз-додатків на Java, надаючи потужні інструменти для безпеки, роботи з даними, створення REST API тощо. [27]

- *База даних:* MySQL або PostgreSQL. MySQL – ще одна дуже популярна реляційна СУБД з відкритим кодом. Для взаємодії з даними MySQL використовує стандартну мову запитів SQL. Система працює за клієнт-серверною моделлю і відома своєю надійністю, швидкістю (особливо для операцій читання) та відносною простотою у використанні та адмініструванні. Завдяки підтримці транзакцій (через механізм зберігання InnoDB) вона забезпечує цілісність даних, що робить її популярним вибором для широкого спектру веб-додатків та серверних рішень. [28]
- *Інше:* API — REST.

Переваги:

- Потужний і масштабований сервер на Java.
- Можливість інтегрувати складну бізнес-логіку на серверній стороні.
- Висока безпека і надійність.

Недоліки:

- Більш складна та тривала розробка серверної частини.
- Потрібно більше ресурсів на підтримку сервера.

2.3.4 Додаткові зовнішні сервіси

Для забезпечення ключових функціональних можливостей мобільного застосунку, таких як переклад слів та фраз, а також зберігання та доставка контенту книг, було проведено аналіз та вибір відповідних зовнішніх сервісів. Розглянемо основні варіанти для API перекладу та хмарних сховищ.

Вибір API для перекладу тексту: [31, 32]

Для інтеграції функції перекладу виділених слів чи фраз у застосунку необхідний надійний API сервіс. Ключовими критеріями вибору є якість перекладу, підтримка необхідних мовних пар, наявність безкоштовного плану для початкового етапу, простота інтеграції та швидкість відповіді сервісу.

- **MyMemory Translated API:** цей сервіс агрегує переклади з великої бази даних, створеної людьми (Translation Memory), та використовує машинні перекладачі, якщо відповідний людський переклад не знайдено. Його перевагами є доступ до великої кількості людських перекладів, що може покращити результат для поширених фраз, та щедрий безкоштовний тариф, зручний для розробки й невеликих проєктів. До недоліків можна віднести потенційно нижчу якість машинного перекладу для складних текстів порівняно з лідерами ринку.
- **Google Cloud Translation API:** один із найпотужніших сервісів машинного перекладу, що використовує сучасні нейронні моделі та підтримує понад 100 мов. Перевагами є висока якість перекладу, широка мовна підтримка, можливість автоматичного визначення мови, надійна інфраструктура та масштабованість. Основним недоліком є висока вартість після вичерпання безкоштовної квоти, що може бути надлишковим для проєктів з обмеженим бюджетом.
- **Microsoft Translator API (Azure Cognitive Services):** сервіс перекладу від Microsoft, що підтримує понад 100 мов, виявлення мови, переклад в реальному часі та інші функції. До переваг належать висока якість перекладу та вигідний безкоштовний тариф (до 2 мільйонів символів/місяць), що підходить для тестування та середніх проєктів. Недоліками є дещо складніша інтеграція, необхідність створення акаунту Azure та обмежена кількість SDK поза середовищем .NET/Azure.

Вибір хмарного сховища для контенту книг: [33]

Для зберігання файлів книг, які будуть доступні користувачам через додаток, необхідне надійне та ефективне хмарне сховище. Основними

критеріями вибору є вартість зберігання, вартість трафіку (особливо вихідного), надійність, масштабованість, простота інтеграції (наприклад, S3-сумісність) та швидкість доступу.

- **Cloudflare R2 Storage:** відносно новий S3-сумісний сервіс об'єктного сховища, головною перевагою якого є відсутність плати за вихідний трафік, що робить його привабливим для додатків з великою кількістю завантажень. S3-сумісний API та інтеграція з CDN Cloudflare спрощують інтеграцію та прискорюють доставку. Недоліком може бути менш розгалужена екосистема порівняно з AWS S3 через новизну сервісу.
- **Amazon S3 (Simple Storage Service):** один із найпоширеніших сервісів об'єктного зберігання, відомий високою надійністю, доступністю та величезною екосистемою інструментів. Він пропонує гнучке керування доступом та чудову документацію. До недоліків належить значна вартість вихідного трафіку при великих обсягах завантажень та складність передбачення витрат через численні тарифікації, що може бути надлишковим для невеликих проєктів.
- **Google Cloud Storage:** об'єктне хмарне сховище від Google з високою продуктивністю та інтеграцією з іншими сервісами Google Cloud. Пропонує гнучкі класи зберігання для оптимізації витрат та сучасну інфраструктуру. Недоліками є плата за вихідний трафік, аналогічно до Amazon S3, та потенційно ускладнене керування правами доступу, що вимагає глибшого розуміння IAM.

2.3.5 Обґрунтування вибору стеку технологій

Після аналізу представлених варіантів стеків технологій (Native Android, Native iOS, Android + Java Server) та враховуючи специфіку даного дипломного проєкту, його часові обмеження та наявні навички розробника, було прийнято рішення зупинитися на Варіанті 1: Native Android Application із серверною

частиною на Node.js та базою даних PostgreSQL, доповненому спеціалізованими API для перекладу та хмарним сховищем.

Основною метою проєкту є створення мобільного застосунку для Android. Розробка нативного додатку за допомогою Android SDK та мови Kotlin дозволяє досягти максимальної продуктивності, найкращого користувацького досвіду, повної інтеграції з функціями операційної системи та апаратними можливостями пристроїв Android. Особисте використання розробником платформи Android також сприяє кращому розумінню потреб користувачів та ефективному тестуванню UI/UX.

Kotlin, як рекомендована Google мова для Android, пропонує значні переваги порівняно з Java, включаючи лаконічність коду, null-безпеку та підтримку корутин, що прискорює процес розробки та підвищує надійність кінцевого продукту.

Особиста знайомість розробника з основами Kotlin для Android та з Node.js/Express.js для серверної розробки суттєво знижує криву навчання та дозволяє швидше перейти до безпосередньої реалізації функціоналу, мінімізуючи ризики, пов'язані з освоєнням повністю нових технологій у рамках обмеженого часу.

Node.js побудований на подієво-орієнтованій, неблокуючій моделі вводу/виводу, що робить його надзвичайно ефективним для обробки великої кількості одночасних з'єднань з відносно невеликими обчислювальними витратами на кожне. Це ідеально підходить для REST API, яке переважно займається передачею даних між клієнтом та базою даних. Використання JavaScript (або TypeScript) на сервері дозволяє потенційно уніфікувати мову в певних частинах екосистеми, а величезна кількість готових модулів в менеджері пакунків npm (включаючи фреймворк Express.js) значно прискорює розробку типових завдань, таких як маршрутизація, автентифікація та обробка запитів.

PostgreSQL є потужною та надійною реляційною СУБД, яка добре підходить для зберігання структурованих даних додатку (користувачі, книги,

словники). Її відповідність стандартам SQL та можливості роботи зі складними запитами забезпечують хорошу основу для потенційного майбутнього розвитку проєкту.

Для реалізації функції перекладу було обрано MyMemory Translated API. Цей вибір зумовлений наявністю щедрого безкоштовного тарифу, що є достатнім для потреб дипломного проєкту, а також доступом до великої бази перекладів, створених саме людьми, що потенційно підвищує якість перекладу для поширених фраз та полегшує інтеграцію.

Для зберігання файлів книг було прийнято рішення використовувати Cloudflare R2 Storage. Ключовою перевагою є відсутність плати за вихідний трафік, що критично важливо для додатку з регулярним завантаженням контенту користувачами. S3-сумісність API та конкурентна ціна на зберігання також сприяли цьому вибору.

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі було детально визначено функціональні та нефункціональні вимоги, що окреслили повний спектр можливостей системи. З огляду на часові обмеження проєкту, проведено пріоритезацію цих вимог, визначивши ядро функціоналу для першої реалізації. Це ядро включає реєстрацію/автентифікацію, перегляд каталогу, обов'язкову функцію паралельного читання кількома мовами (із синхронізацією абзаців), базову роботу зі словником та елементарне тестування. Для прискорення розробки обрано архітектуру без локальної бази даних на клієнті, що передбачає повну залежність від онлайн-доступу.

Розроблено проєкт клієнт-серверної архітектури, включаючи концептуальну структуру реляційної бази даних на сервері та опис взаємодії між клієнтом і сервером через RESTful API (HTTPS, JSON). Створено відповідні структурні та алгоритмічні схеми, що візуалізують запропоновану архітектуру та потоки роботи.

Обґрунтовано вибір технологічного стеку: нативний Android-клієнт на Kotlin, серверна частина на Node.js (з Express.js) та СУБД PostgreSQL. Також було обрано ключові зовнішні сервіси: MyMemory Translated API для забезпечення функції перекладу та Cloudflare R2 для зберігання контенту книг, що дозволяє оптимізувати витрати та забезпечити необхідний функціонал. Цей вибір спрямований на досягнення балансу між продуктивністю на цільовій платформі, швидкістю розробки, економічною доцільністю та надійністю.

Таким чином, другий розділ надає комплексне проєктне рішення для застосунку. Варто зазначити, що представлені проєктні рішення та схеми є основою для розробки, однак у процесі безпосередньої програмної реалізації можуть виникнути потреби в їх коригуванні чи уточненні для вирішення практичних завдань та оптимізації кінцевого продукту.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ

3.1 Підготовка та структурування мультилінгвального контенту для читання

Ефективне відображення та взаємодія з мультилінгвальними текстами вимагає ретельної підготовки та структурування контенту. Цей підрозділ описує процес, починаючи від вирівнювання паралельних текстів до їх представлення у форматі, зручному для обробки мобільним застосунком.

Реалізація функції автоматичного вирівнювання текстів безпосередньо в мобільному застосунку представляє значні технічні складності. Такі алгоритми вимагають значних обчислювальних ресурсів (що може негативно вплинути на продуктивність та час роботи батареї мобільного пристрою), доступу до великих мовних моделей та словників, а також складного налаштування для досягнення прийнятної якості вирівнювання для різних мовних пар та жанрів текстів. Крім того, якість автоматичного вирівнювання сильно залежить від якості вхідних текстів, наявності помилок OCR, відмінностей у форматуванні тощо.

З огляду на ці фактори, було прийнято рішення здійснювати підготовку та вирівнювання контенту офлайн, використовуючи спеціалізовані інструменти, та надавати застосунку вже готовий, структурований контент.

Особлива увага приділялася використанню професійних перекладів книг. Це критично важливо для забезпечення точності та природності тексту різними мовами, що є ключовим для комфортного читання та ефективного вивчення мов. Використання аматорських або неякісних машинних перекладів могло б значно погіршити користувацький досвід.

Для цілей даного дипломного проєкту було відібрано та підготовлено три літературних твори:

- "Колгосп тварин" (Animal Farm) Джорджа Орвелла.
- "Маленький принц" (The Little Prince) Антуана де Сент-Екзюпері, який також містить ілюстрації, що інтегровані в структуру контенту.
- "Химерна історія доктора Джекіла та містера Хайда" (Strange Case of Dr Jekyll and Mr Hyde) Роберта Льюїса Стівенсона.

Ці книги доступні професійними перекладами українською, англійською, французькою та іспанською мовами (за винятком французького перекладу “Джекіла і Хайда”, який знайти не вдалося). Автор дипломної роботи володіє українською та англійською мовами на достатньому рівні для контролю якості вирівнювання, а також має базові знання іспанської та французької мов, що дозволило здійснити вибіркову перевірку вирівняних сегментів. Російськомовні переклади свідомо не використовувалися з огляду на поточну суспільно-політичну ситуацію.

3.1.1 Використання LF_aligner для вирівнювання паралельних текстів

LF_aligner – це утиліта з відкритим кодом, призначена для автоматичного вирівнювання текстів на рівні речень або абзаців. Вона використовує алгоритми, що базуються на довжині сегментів та, опціонально, на словниках або моделях машинного перекладу для знаходження відповідностей між двома або більше паралельними текстами. Інструмент особливо ефективний для текстів, які вже є перекладами один одного. [34]

У даному проєкті LF_aligner використовувався для попередньої обробки вихідних текстів книг. Для кожної книги, що мала бути додана до застосунку, її версії різними мовами подавалися на вхід LF_aligner (рис. 3.1). Результатом роботи інструменту є набір вирівняних пар абзаців, де кожен абзац однією мовою має свого відповідника однією або кількома іншими мовами.

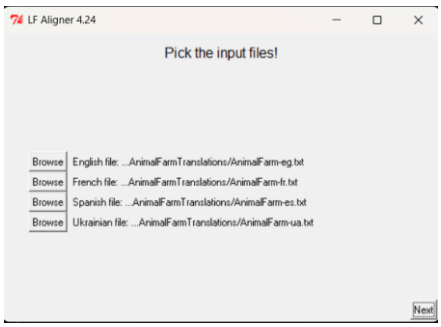
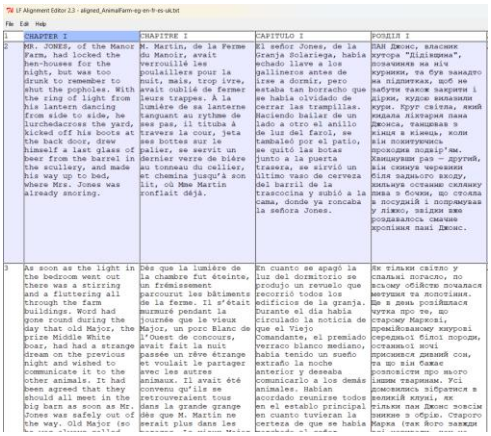


Рис. 3.1 – Завантаження текстів відповідних мов у LF aligner

Після автоматичного вирівнювання проводилася вибіркова ручна перевірка та корекція сегментації (рис. 3.2) для забезпечення максимальної точності.



	CHAPTER I	CHAPITRE I	CAPITULO I	РОЗДІЛ I
2	MR. JONES, of the Manor Farm, had locked the hen-houses for the night, but was too drunk to remember to shut the popholes. With the ring of light from his lantern dancing from side to side, he lurchedacross the yard, kicked off his boots at the back door, drew himself a last glass of beer from the barrel in the scullery, and made his way up to bed, where Mrs. Jones was already snoring.	M. Martin, de la Ferme du Manoir, avait verrouillé les poulaillers pour la nuit, mais, trop ivre, avait oublié de fermer leurs trappes. À la lumière de sa lanterne tanguant au rythme de ses pas, il tituba à travers la cour, jeta ses bottes sur le palier, se servit un dernier verre de bière au tonneau du cellier, et chemina jusqu'à son lit, où Mme Martin ronflait déjà.	El señor Jones, de la Granja Solariega, había echado llave a los gallineros antes de irse a dormir, pero estaba tan borracho que se había olvidado de cerrar las trampillas. Haciendo bailar de un lado a otro el anillo de luz del farol, se tambaleó por el patio, se quitó las botas junto a la puerta trasera, se sirvió un último vaso de cerveza del barril de la trascocina y subió a la cama, donde ya roncaba la señora Jones.	ПАН Джонс, власник хутора "Дідівщина", позачиняв на ніч курники, та був занадто на підпитках, щоб не забути також закрити і дірки, куди вилазили кури. Круг світла, який видала ліхтарик пана Джонса, танцював з кінця в кінець, коли він похитувись проходив подвір'ям. Хвацнувши раз – другий, він скинув черевики біля заднього входу, жильнув останню склянку пива з бочки, що стояла в посудній і попрямував у ліжко, звідки вже розділявсь смачне хропіння пані Джонс.
3	As soon as the light in the bedroom went out there was a stirring and a fluttering all through the farm buildings. Word had gone round during the day that old Major, the prize Middle White boar, had had a strange dream on the previous night and wished to communicate it to the other animals. It had been agreed that they should all meet in the big barn as soon as Mr. Jones was safely out of the way. Old Major (so	Dès que la lumière de la chambre fut éteinte, un frémissement parcourut les bâtiments de la ferme. Il s'était murmuré pendant la journée que le vieux Major, un porc Blanc de l'Ouest de concours, avait fait la nuit passée un rêve étrange et voulait le partager avec les autres animaux. Il avait été convenu qu'ils se retrouveraient tous dans la grande grange dès que M. Martin ne serait plus dans les	En cuanto se apagó la luz del dormitorio se produjo un revuelo que recorrió todos los edificios de la granja. Durante el día había circulado la noticia de que el Viejo Comandante, el premiado verraco blanco mediano, había tenido un sueño extraño la noche anterior y deseaba comunicarlo a los demás animales. Habían acordado reunirse todos en el establo principal en cuanto tuvieran la certeza de que se había	Як тільки світло у спальні погасло, по всьому обійсту почалася метушня та полотіння. Бо з день розійшлася чутка про те, що старому Мажові, премійованому хлуросі середньої білої породи, останньої ночі приснився дивний сон, та що він бажає поспілкстя про нього іншим тваринам. Усі домовлялись зібратися в великій клуні, як тільки пан Джонс зовсім зникне з обрію. Старого Мажора (так його завжди

Рис. 3.2 – Редагування сегментації у LF Aligner

LF_aligner дозволяє експортувати результати у форматі TMX (Translation Memory eXchange), який є XML-структурою, що містить пари вирівняних сегментів (абзаців) для різних мов (рис. 3.3).

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE tmx SYSTEM "tmx14.dtd">
<tmx version="1.4">
  <header
    creationtool="LF Aligner"
    creationtoolversion="4.24"
    datatype="unknown"
    segtype="sentence"
    adminlang="EN-UK"
    srclang="EN-UK"
    o-tmf="TW4Win 2.0 Format"
  >
  </header>
  <body>
<tu creationdate="20250528T133224Z" creationid="LF Aligner 4.24"><prop type="Txt::Note">AnimalFarm-eg</prop>
<tuv xml:lang="EN-UK"><seg>CHAPTER I</seg></tuv>
<tuv xml:lang="FR-FR"><seg>CHAPITRE I</seg></tuv>
<tuv xml:lang="ES"><seg>CAPITULO I</seg></tuv>
<tuv xml:lang="UA"><seg>РОЗДІЛ I</seg></tuv> </tu>

<tu creationdate="20250528T133224Z" creationid="LF Aligner 4.24"><prop type="Txt::Note">AnimalFarm-eg</prop>
<tuv xml:lang="EN-UK"><seg>MR. JONES, of the Manor Farm, had locked the hen-houses for the night, but was too
<tuv xml:lang="FR-FR"><seg>M. Martin, de la Ferme du Manoir, avait verrouillé les poulaillers pour la nuit, ma
<tuv xml:lang="ES"><seg>El señor Jones, de la Granja Solariega, había echado llave a los gallineros antes de i
<tuv xml:lang="UA"><seg>ПАН Джонс, власник хутора &amp;quot;Дідівщина&amp;quot;, позачиняв на ніч курники, та був зана
```

Рис. 3.3 – Згенерований TMX файл сегментації

TMX – це XML-формат, призначений для обміну пам'яттю перекладів. Файл TMX складається з набору "одиниць перекладу" (<tu>). Кожна одиниця перекладу містить один або декілька "варіантів одиниці перекладу" (<tuv>), де кожен <tuv> представляє текст сегмента (зазвичай речення або абзацу) певною мовою. Атрибут xml:lang у тезі <tuv> вказує на мову сегмента (див. рис. 3.3). [36]

Для перетворення TMX-файлу в JSON-структуру книги (описану в п. 3.1.2) використовується програмний підхід, реалізований за допомогою скрипта на Python – convert_tmx_to_json.py, з використанням бібліотеки xml.etree.ElementTree для парсингу XML (рис 3.4).

```
for entry in segments:
    # Перевірка: чи це зображення
    image_detected = any(is_image_filename(entry.get(lang, "")) for lang in languages)

    if image_detected and current_chapter and current_chapter["paragraphs"]:
        # Додаємо поле "image" до останнього абзацу
        for lang in languages:
            if lang in entry and is_image_filename(entry[lang]):
                current_chapter["paragraphs"][-1]["image"] = entry[lang]
                break  # зупинити після першого вдалого
        continue  # не додаємо як окремий абзац

    # Перевірка: чи це заголовок розділу
    is_chapter_title = any(
        re.match(chapter_patterns[lang], entry.get(lang, ""), re.IGNORECASE)
        for lang in languages
    )

    if is_chapter_title:
        if current_chapter:
            chapters.append(current_chapter)
        current_chapter = {
            "title": {lang: entry.get(lang, "") for lang in languages},
            "paragraphs": []
        }
    else:
        if current_chapter:
            paragraph = {lang: entry.get(lang, "") for lang in languages if lang in entry}
            current_chapter["paragraphs"].append(paragraph)

# Add last chapter
if current_chapter:
    chapters.append(current_chapter)
```

Рис. 3.4 – Ідентифікація та структурування розділів, абзаців та зображень

Після цього, створюється кореневий JSON-об'єкт з метаданими книги та масивом chapters.

3.1.2 Перетворення вирівняних текстів у JSON-формат

Після отримання вирівняних абзаців, наступним кроком є їх структурування у формат JSON, який є зручним для передачі, зберігання та

парсингу як на сервері, так і в мобільному застосунку. Було розроблено уніфіковану JSON-структуру для представлення мультилінгвальних книг.

Структура JSON-файлу книги:

Кожна книга представлена JSON-об'єктом, що містить метадані (назва, автор) та масив розділів (chapters). Кожен розділ, у свою чергу, містить назву (також мультилінгвальну) та масив абзаців (paragraphs). Ключовим елементом є об'єкт абзацу, де текст представлений як набір пар "код_мови": "текст_абзацу".

Рис. 3.5 – Приклад структури JSON для книги "Маленький принц"

Така структура дозволяє:

- Легко отримувати текст абзацу для будь-якої з доступних мов за її кодом.
- Відображати назви розділів відповідною мовою.
- Зберігати зв'язок між паралельними абзацами, оскільки вони знаходяться в одному JSON-об'єкті.
- Включати додаткову інформацію, таку як посилання на зображення ("image": "url_to_image.jpg"), що асоціюються з конкретним абзацом.

3.1.3 Механізм завантаження та парсингу контенту в застосунку

JSON-файли з контентом книг зберігаються на R2 Storage від Cloudflare. Інформація про доступні книги (метадані та URL до повного JSON-файлу

книги) отримується з API сервера, через ендпоінти /api/books (для загального каталогу) та /api/library (для персональної бібліотеки користувача). Структура відповіді цих ендпоінтів зображена на рис. 3.6.

Рис 3.6 – Структура відповіді на запит отримання книг

Компонент ReaderActivity.kt відповідає за завантаження та відображення контенту обраної книги. Отримавши URL книги, він асинхронно завантажує JSON-файл. Після успішного завантаження, рядок JSON парситься стандартними засобами Android (JSONObject, JSONArray, рис. 3.7) для вилучення розділів та абзаців. Кожен вирівняний абзац перетворюється на об'єкт ChapterParagraph і передається до адаптера ParagraphPagerAdapter для відображення у ViewPager2 як окрема "сторінка" застосунку.

Рис. 3.7 – Парсинг книги на клієнтській частині

3.2 Реалізація основного функціоналу застосунку для мультилінгвального читання книг

3.2.1 Відображення каталогу книг та персональної бібліотеки користувача

Обидва наступні описані фрагменти використовують патерн MVVM (ViewModel для управління даними та бізнес-логікою) та LiveData для спостереження за змінами даних та оновлення UI.

Загальний каталог книг:

Надає користувачеві можливість переглядати всі доступні в системі книги і реалізований у фрагменті CatalogFragment.kt.

CatalogFragment взаємодіє з CatalogViewModel, який, у свою чергу, запитує список книг з серверного API (ендпоінт /api/books) (рис. 3.8).

```
interface ApiService {

    // --- Book & Library Management ---

    /** Fetches a list of all available books. */
    @GET("api/books")
    suspend fun getBooks(): Response<List<Book>>

    /** Adds a book to the user's personal library. */
    @POST("api/library")
    suspend fun addToLibrary(@Body request: AddToLibraryRequest): Response<AddToLibraryResponse>

    /** Fetches all books in the current user's library. */
    @GET("api/library")
    suspend fun getUserLibrary(): Response<List<Book>>

    /** Removes a book from the user's library by its ID. */
    @DELETE("api/library/{id}")
    suspend fun removeFromLibrary(@Path("id") bookId: Int): Response<RemoveResponse>
```

Рис. 3.8 – Деякі функції запитів до сервера, які використовуються каталогом і особистою бібліотекою

Список книг відображається за допомогою RecyclerView та кастомного адаптера BookAdapter.kt. Кожен елемент списку показує назву, автора, короткий опис та доступні мови книги.

При натисканні на книгу в каталозі, користувачеві пропонується діалогове вікно (showLanguageSelectionDialog) для вибору від 2 до 4 мов, якими він бажає читати цю книгу (рис. 3.9).

```
private fun showLanguageSelectionDialog(book: Book) {
    Log.d(TAG, msg: "Showing language selection dialog for book ID: ${book.id}")
    val availableLangsArray = book.available_languages.toTypedArray()
    val checkedItems = BooleanArray(availableLangsArray.size) // To track selected items

    AlertDialog.Builder(requireContext())
        .setTitle("Select 2-4 Languages") // Title for the dialog
        .setMultiChoiceItems(availableLangsArray, checkedItems) { _, which, isChecked ->
            // Update the checkedItems array when an item's state changes
            checkedItems[which] = isChecked
        }
        .setPositiveButton( text: "Add to Library") { _, _ ->
            val selectedLanguages = availableLangsArray.filterIndexed { index, _ -> checkedItems[index] }
            Log.d(TAG, msg: "Selected languages for book ID ${book.id}: $selectedLanguages")

            if (selectedLanguages.size < 2 || selectedLanguages.size > 4) {
                Log.w(TAG, msg: "Invalid number of languages selected: ${selectedLanguages.size}. Required 2-4.")
                Toast.makeText(requireContext(), text: "Please select between 2 and 4 languages.", Toast.LENGTH_SHORT).show()
            } else {
                viewModel.addToLibrary(book.id, selectedLanguages) // Call ViewModel to add to library
            }
        }
        .setNegativeButton( text: "Cancel") { dialog, _ ->
            Log.d(TAG, msg: "Language selection cancelled for book ID: ${book.id}")
            dialog.dismiss()
        }
        .show()
}
```

Рис. 3.9 – Функція для показу діалогового вікна для вибору мов

Після вибору мов, інформація передається до CatalogViewModel для відправки запиту на сервер (додавання книги до персональної бібліотеки користувача).

Персональна бібліотека користувача:

Відображає книги, які користувач додав до свого списку для читання і реалізована у фрагменті LibraryFragment.kt.

LibraryFragment взаємодіє з LibraryViewModel, який завантажує список книг користувача з серверного API (ендпоінт /api/library) (див. рис. 3.8). Кожна книга в бібліотеці, окрім стандартних метаданих, містить інформацію про раніше обрані користувачем мови (selected_languages).

Список книг бібліотеки також відображається за допомогою RecyclerView та адаптера LibraryAdapter, схожий на BookAdapter, але з додатковими діями (зміна мов і видалення з бібліотеки). Елементи списку показують обрані мови.

Користувач має можливість змінити раніше обрані мови для книги в бібліотеці через діалогове вікно (showLanguageSelectionDialog).

При натисканні на книгу, запускається ReaderActivity.kt, куди передаються r2_url для завантаження контенту, book_id, назва книги та список обраних мов (selected_languages).

Також реалізовано функцію видалення книги з персональної бібліотеки, що супроводжується діалогом підтвердження (confirmAndRemoveBook).

3.2.2 Реалізація інтерфейсу застосунку для читання з паралельним відображенням мов

У п.3.1.3 вже розглядалося як відбувається завантаження та парсинг контенту, тому зараз розглянемо інші аспекти, такі як паралельне відображення контенту, навігація та збереження і відновлення прогресу читання.

Паралельне відображення контенту:

Кожна "сторінка" застосуну, керована ViewPager2 та кастомним адаптером (ParagraphPagerAdapter), динамічно відображає відповідні сегменти тексту (абзаци та, за наявності, назви розділів) для кожної з 2-4 обраних мов (рис. 3.10). Тексти розташовуються паралельно, дозволяючи користувачеві легко їх зіставляти. Якщо у контенті книги присутні ілюстрації, пов'язані з поточним абзацом, вони також відображаються на сторінці

```
// Display chapter title if this is the first paragraph of a chapter
if (currentItem.isFirstInChapter) {
    handleChapterTitle(holder, currentItem.chapterTitleJson)
}

// Check for and handle image
val imageUrl = currentItem.paragraphJson.optString( name: "image",  fallback: "")
if (imageUrl.isNotBlank()) {
    handleImage(holder, imageUrl)
}

// Iterate through selected languages and create TextViews for paragraph content
var paragraphAdded = false
for (lang in selectedLanguages) {
    val paragraphText = currentItem.paragraphJson.optString(lang, context.getString(R.string.translation_unavailable_placeholder, lang.uppercase()))

    // Only add TextView if text is meaningful
    if (paragraphText.isNotBlank() && paragraphText != context.getString(R.string.translation_unavailable_placeholder, lang.uppercase())) {
        paragraphTexts[lang] = paragraphText
        val paragraphTextView = createParagraphTextView(paragraphText, lang, position, holder)
        holder.paragraphContentContainer.addView(paragraphTextView)
        holder.textViewsOnPage[lang] = paragraphTextView // Store for highlighting
        paragraphAdded = true
    }
}
paragraphsTextCache[position] = paragraphTexts // Cache the texts for this position
```

Рис. 3.10 – Динамічне створення TextView для кожної обраної мови

Навігація по книзі:

Для переходу між сторінками реалізовано декілька механізмів: екранні кнопки "Вперед" та "Назад", можливість свайп-жестів, використання SeekBar для швидкого переходу, а також пряме введення номера бажаної сторінки (рис. 3.11). Інтерфейс завжди інформує користувача про поточну сторінку та загальну їх кількість.

```
private fun jumpToPage() {
    try {
        val pageNumber = jumpToPageInput.text.toString().toInt()
        if (pageNumber in 1 ≤ .. ≤ totalPages) {
            val position = pageNumber - 1 // Convert to 0-indexed
            viewPager.currentItem = position
            jumpToPageInput.text.clear()

            // Hide keyboard after jumping
            val imm = getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
            imm.hideSoftInputFromWindow(jumpToPageInput.windowToken,  flags: 0)
        } else {
            Toast.makeText( context: this,  text: "Page number out of range (1-$totalPages)", Toast.LENGTH_SHORT).show()
        }
    } catch (e: NumberFormatException) {
        Toast.makeText( context: this,  text: "Please enter a valid page number", Toast.LENGTH_SHORT).show()
    }
}
```

Рис. 3.11 – Функція реалізація переходу на будь-яку сторінку через введення числа

Збереження та відновлення прогресу читання:

Для забезпечення безперервності читання, поточна позиція користувача (номер сторінки/абзацу та позиція прокрутки всередині нього) автоматично зберігається при зміні сторінки або при повному виході з застосунку. Цей процес координується класом ReadingPositionManager.kt, який зберігає дані як локально (використовуючи ReadingPositionStorage.kt на базі SharedPreferences), так і синхронізує їх з сервером через відповідний API ендпоінт (рис. 3.12).

```
interface ApiService {
    // --- Reading Progress ---

    /** Gets the last read page number for a specific book. */
    @GET("api/library/last-page")
    suspend fun getLastReadPage(@Query("book_id") bookId: String): Response<LastPageResponse>

    /** Updates the last read page number for a book. */
    @POST("api/library/last-page")
    suspend fun updateLastReadPage(@Body request: LastPageRequest): Response<Unit>
```

Рис. 3.12 – Функції запитів до сервера, які використовуються для збереження і відновлення останньої сторінки

При повторному відкритті книги, остання збережена позиція відновлюється, дозволяючи продовжити читання з того ж місця.

3.2.3 Інтерактивна взаємодія з текстом: переклад слів

Для полегшення розуміння іншомовного тексту реалізовано функцію швидкого перекладу слів.

Вибір слова:

При довгому натисканні (long press) на слово в будь-якому з паралельних текстів, система аналізує позицію дотику та текстовий вміст відповідного TextView для точного визначення вибраного слова, враховуючи його межі та ігноруючи пробіли чи розділові знаки.

Отримання перекладу через API MyMemory:

Після ідентифікації слова, спеціалізований клас TranslationManager.kt ініціює запит до зовнішнього API перекладу MyMemory (рис. 3.13). Для цього формується HTTP-запит, що містить вибране слово, мову оригіналу цього слова та мовні коди інших текстів, обраних користувачем для паралельного читання (які стають цільовими мовами для перекладу). Відповідь від API, що містить перекладений текст, обробляється асинхронно.

```
suspend fun getTranslation(word: String, fromLang: String, toLang: String): String? {
    return withContext(Dispatchers.IO) { // Perform network operation on IO dispatcher
        try {
            Log.d(TAG, msg: "Translation requested: '$word' from $fromLang to $toLang")

            // Map local language codes to API-specific codes
            val sourceCode = LANGUAGE_CODES[fromLang.lowercase()] ?: fromLang.lowercase()
            val targetCode = LANGUAGE_CODES[toLang.lowercase()] ?: toLang.lowercase()

            Log.d(TAG, msg: "Using mapped codes: from $sourceCode to $targetCode")

            val translation = makeApiCall(word, sourceCode, targetCode)

            if (translation.isNullOrBlank()) {
                Log.d(TAG, msg: "API translation failed or returned empty.")
            } else {
                Log.d(TAG, msg: "API translation successful: $translation")
            }
            translation // Return the result from the API call
        } catch (e: Exception) {
            // Catch any unexpected errors during the translation process
            Log.e(TAG, msg: "Translation error: ${e.message}", e)
            null // Return null in case of any exception
        }
    }
}
```

Рис. 3.13 – Функція отримання перекладу виділеного слова

Відображення перекладу:

Отримані переклади відображаються користувачеві у рухомому спливаючому вікні. Одночасно для візуального зв'язку оригінальне вибране слово, а також його ймовірні відповідники (знайдені за допомогою алгоритмів стемінгу для врахування різних граматичних форм) у паралельних текстах підсвічуються (рис. 3.14). Перед кожним новим підсвічуванням усі попередні знімаються.

```
private fun highlightOccurrencesInTextView(
    textView: TextView,
    wordToMatch: String,
    langCode: String,
    minWordLength: Int = 2
) {
    if (wordToMatch.isBlank() || wordToMatch.length < minWordLength) return
    val fullText = textView.text
    if (fullText.isEmpty()) return

    val stemFunction = getStemmerForLanguage(langCode)
    val wordToMatchStem = stemFunction(wordToMatch)

    val wordsInText = getWordsFromText(fullText)
    val occurrences = findWordOccurrences(
        fullText,
        wordsInText,
        wordToMatchStem,
        stemFunction,
        minWordLength
    )

    if (occurrences.isNotEmpty()) {
        applyHighlights(textView, occurrences, highlightColor)
    }
}
```

Рис. 3.14 – Функція виділення слова і перекладів

Додавання до словника:

У спливаючому вікні перекладу присутня кнопка "Зберегти до словника". При її натисканні викликається метод, який відправляє запит на серверний API для збереження слова, його мови оригіналу та отриманих перекладів.

3.3 Розробка інструментів для вивчення нових слів

Для активного засвоєння лексики, з якою користувач стикається під час читання, у застосунку реалізовано два основні інструменти: персональний словник та модуль практичного тестування знань.

3.3.1 Реалізація персонального словника користувача

Персональний словник дозволяє користувачам зберігати невідомі слова, їх переклади та мову оригіналу, а також керувати цим списком. Реалізація включає відображення слів, додавання нових, редагування та видалення існуючих, а також фільтрацію та пошук.

Всі операції зі словником синхронізуються з серверною частиною. Взаємодія з сервером для управління даними словника відбувається через відповідні API запити, які показано на рис. 3.15.

```
interface ApiService {
    // --- Dictionary Management ---

    /** Fetches details for a specific word by its ID. */
    @GET("api/dictionary/word/{id}")
    suspend fun getWord(@Path("id") wordId: Int): Response<WordResponse>

    /** Fetches the user's dictionary, optionally filtered by language. */
    @GET("api/dictionary")
    suspend fun getDictionary(
        @Query("lang") languageKey: String? = null,
        @Query("source_lang") sourceLanguage: String? = null
    ): Response<List<DictionaryWord>>

    /** Adds a new word to the user's dictionary. */
    @POST("api/dictionary")
    suspend fun addWord(@Body request: SaveWordRequest): Response<String> // Assuming String is a success message or ID

    /** Deletes a word from the user's dictionary by its ID. */
    @DELETE("api/dictionary/{id}")
    suspend fun deleteWord(@Path("id") wordId: Int): Response<Unit>
```

Рис. 3.15 – Деякі функції запитів до сервера, які використовуються модулем словника

Завантаження та відображення слів (DictionaryFragment.kt):

На екрані словника користувач бачить список всіх доданих ним слів, де для кожного слова вказано його мову оригіналу та набір перекладів іншими мовами.

Додавання нового слова (AddWordDialogFragment.kt):

Слова можуть бути додані до словника двома шляхами:

1. Безпосередньо з застосунку для читання, як описано в п. 3.2.3, при натисканні "Зберегти до словника" після отримання перекладу.
2. Вручну через діалогове вікно, яке викликається з екрану словника. У цьому діалозі користувач вводить слово, обирає мову оригіналу та додає

один або декілька перекладів у динамічно створювані поля (рис. 3.16). При додаванні здійснюється валідація введених даних, наприклад, перевірка на порожні поля або унікальність мов перекладу.

```
private fun addNewTranslationFieldInternal(container: ViewGroup) {
    val usedLanguages = mutableSetOf<String>()
    // Collect currently selected languages in other fields to suggest a new one
    for (i in 0 s until < container.childCount) {
        val view = container.getChildAt(i)
        val languageSpinner = view.findViewById<Spinner>(R.id.spinner_language)
        (languageSpinner.selectedItem?.toString()?.trim()?.lowercase())?.let {
            if (it.isNotEmpty()) usedLanguages.add(it)
        }
    }
    // Suggest the first available language not already in use
    val suggestedLanguage = languageOptions.firstOrNull { it.lowercase() !in usedLanguages } ?: languageOptions.firstOrNull() ?: ""
    Log.d(TAG, msg: "Adding new empty translation field row. Suggested language: '$suggestedLanguage'")
    addTranslationFieldToLayoutInternal(container, suggestedLanguage, translation: "")
}
```

Рис. 3.16 – Функція показу діалогового вікна для додавання слова у словник

Редагування існуючого слова (EditWordDialogFragment.kt):

При натисканні кнопки редагування, відкривається діалогове вікно. Користувач може змінювати, додавати або видаляти переклади для обраного слова (саме слово та мова оригіналу не редагуються). Це дозволяє виправляти помилки або додавати нові варіанти перекладу.

Видалення слова:

Видалення слова ініціюється відповідною кнопкою в списку та підтверджується користувачем через стандартний AlertDialog.

Фільтрація та пошук слів:

Для зручності роботи з великим обсягом лексики реалізовано текстовий пошук по словах та їх перекладах, а також можливість фільтрації списку слів за мовою оригіналу та мовою перекладу (рис. 3.17).

```
private fun filterAndPostWords(originLang: String?, translationLang: String?) {
    Log.d(TAG, msg: "Filtering words with origin: '$originLang', translation: '$translationLang'")
    val filteredList = allFetchedWords.filter { word ->
        val originMatches = originLang?.let {
            word.language.equals(it, ignoreCase = true)
        } ?: true // No origin filter means all origin languages match

        val translationMatches = translationLang?.let {
            word.translations.keys.any { transLang -> transLang.equals(it, ignoreCase = true) }
        } ?: true // No translation filter means all translation languages match

        originMatches && translationMatches
    }
    _displayedItems.value = filteredList
    Log.d(TAG, msg: "Filtered words count: ${filteredList.size}")
}
```

Рис. 3.17 – Функція фільтрація слів словника

3.3.2 Розробка модуля для практичного тестування знань

Для закріплення вивчених слів та оцінки рівня знань користувача, в застосунку реалізовано модуль практичного тестування. Він дозволяє налаштовувати параметри тесту, проходити тестування у форматі "введення слова" та переглядати історію попередніх тестів.

Взаємодія з сервером для отримання питань, надсилання відповідей та збереження результатів тестування відбувається через відповідні API ендпоінти.

```
// --- Practice Test Management ---

/** Starts a new practice test session with specified parameters. */
@POST("api/practice/start-test")
suspend fun startTest(@Body request: StartTestRequest): Response<List<PracticeQuestion>>

/** Submits an answer for the current question in a practice test. */
@POST("api/practice/submit-answer")
suspend fun submitAnswer(@Body request: SubmitAnswerRequest): Response<SubmitAnswerResponse>

/** Finishes the current practice test and submits results. */
@POST("api/practice/finish-test")
suspend fun finishTest(@Body request: FinishTestRequest): Response<FinishTestResponse>

/** Fetches the scores of the last few practice tests for the user. */
@GET("api/practice/scores")
suspend fun getLastTestScores(): Response<List<TestScore>>
```

Рис. 3.18 – Деякі функції запитів до сервера, які використовуються модулем практики

Центр практики (PracticeHubFragment.kt):

У "Хабі практики" користувач може переглянути історію всіх пройдених ним тестів, бачити їх основні результати, а також застосовувати фільтри (наприклад, за датою, результатом, мовною парою) для аналізу свого прогресу. Передбачена можливість видалення окремих записів з історії тестів.

Також тут розташована кнопка "Розпочати нову практику", яка ініціює показ діалогового вікна для налаштування параметрів нового тесту.

Налаштування нового тесту (ConfigureTestDialogFragment.kt):

Перед початком тестування користувач може обрати параметри: мову оригіналу слова, мову, на яку потрібно дати переклад, та кількість питань у тесті. Система динамічно визначає максимальну кількість доступних питань на

основі слів у персональному словнику користувача, що відповідають обраній мовній парі.

Процес проходження тесту (PracticeActivity.kt):

Користувачеві послідовно пред'являються слова мовою оригіналу. Він має ввести відповідний переклад у спеціальне поле. Інтерфейс тестування відображає поточне питання та загальний прогрес. (рис. 3.19).

```
/** Displays the current question text and clears previous input/feedback. */
private fun showQuestion(question: PracticeQuestion) {
    questionText.text = getString(R.string.question_format_translate, question.origin_lang, question.translation_lang, question.word)
    editAnswer.text.clear()
    feedbackText.text = ""
    Log.i( tag: "PracticeActivity",  msg: "showQuestion: Displaying question for word '${question.word}'")
}

/** Updates the progress bar and text to reflect current test progress. */
private fun updateProgressDisplay() {
    val totalQuestions = viewModel.currentQuestionList.size
    val answeredQuestions = viewModel.allAnswers.size

    progressBar.max = if (totalQuestions > 0) totalQuestions else 1
    progressBar.progress = answeredQuestions
    progressText.text = "{answeredQuestions}/{totalQuestions}"
    Log.d( tag: "PracticeActivity",  msg: "updateProgressDisplay: $answeredQuestions/$totalQuestions")
}
```

Рис. 3.19 – Функції відображення питання та оновлення відображення прогресу

Після введення відповіді та її підтвердження, система негайно надає зворотний зв'язок: позначає відповідь як правильну або, у випадку помилки, показує правильний варіант перекладу (рис. 3.20). Залежно від того, чи є ще питання, кнопка дії змінюється на "Наступне" або "Завершити тест".

```
// Observe feedback for the submitted answer and update UI
viewModel.feedback.observe( owner: this) { response ->
    response?.let {
        Log.d( tag: "PracticeActivity",  msg: "feedback observer: Received feedback - Correct: ${it.isCorrect}")
        val feedbackMsg = if (it.isCorrect) {
            val correctText = "Correct"
            "<font color='#2e7d32'>$correctText ✅</font>"
        } else {
            val incorrectText = "Incorrect"
            val correctAnswerPrefix = "Correct answer:"
            "<font color='#c62828'>$incorrectText ❌</font><br/>$correctAnswerPrefix ${it.correctAnswer}"
        }
        feedbackText.text = HtmlCompat.fromHtml(feedbackMsg, HtmlCompat.FROM_HTML_MODE_LEGACY)

        updateProgressDisplay()

        val currentQuestionsCompleted = viewModel.allAnswers.size
        val totalQuestions = viewModel.currentQuestionList.size
        val questionsRemaining = totalQuestions - currentQuestionsCompleted

        Log.d( tag: "PracticeActivity",  msg: "Questions completed: $currentQuestionsCompleted, Total: $totalQuestions, Remaining: $questionsRemaining")

        awaitingSubmission = false
```

Рис. 3.20 – Зворотний зв’язок про правильні відповіді

Відображення результатів тесту (TestSummaryFragment.kt):

Після завершення всіх питань тесту, користувачеві відображається екран з детальними результатами: загальна кількість правильних/неправильних відповідей, відсоток успішності та список усіх питань з відповідями. Ці результати зберігаються на сервері. Залежно від контексту (щойно завершений тест чи перегляд історії), кнопка у фрагменті дозволяє або повернутися до центру практики, або просто закрити вікно результатів.

3.4 Розробка UI/UX для користувачів

Дизайн застосунку прагне до мінімалізму, уникаючи візуального шуму та надлишкових елементів. Основний акцент робиться на контенті – текстах книг та навчальних матеріалах. Використовується обмежена палітра кольорів (білий, чорний, фіолетовий), що сприяє концентрації та не відволікає від читання.

Оскільки деталі UI для окремих функцій були розглянуті раніше, цей підрозділ узагальнює ключові принципи UI/UX, застосовані в усьому продукті.

Структура навігації:

Навігаційна модель застосунку побудована на основі стандартного для Android патерну – нижньої навігаційної панелі (Bottom Navigation Bar), реалізованої в MainActivity. Це забезпечує швидкий доступ до чотирьох основних функціональних розділів: "Каталог книг" (CatalogFragment), "Моя Бібліотека" (LibraryFragment), "Словник" (DictionaryFragment) та "Центр Практики" (PracticeHubFragment). Окремі функціональні блоки, такі як автентифікація (LoginActivity, RegisterActivity), застосунок для читання (ReaderActivity) та проходження тесту (PracticeActivity), реалізовані як окремі Activity. Такий підхід дозволяє максимізувати екранний простір для ключових завдань (читання, тестування) та логічно ізолювати ці процеси.

Механізми зворотного зв'язку:

Для забезпечення прозорості роботи застосунку та постійного інформування користувача про поточний стан системи впроваджено

різноманітні механізми. До них належать візуальні індикатори завантаження даних (ProgressBar, SwipeRefreshLayout), короткі системні повідомлення (Toast, Snackbar) про успішне виконання операцій або виникнення помилок, а також динамічний візуальний відгук на дії користувача, як-от зміна стану інтерактивних елементів або підсвічування тексту в читалці. Окрім того, для порожніх станів списків (наприклад, коли бібліотека або словник ще не містять записів) передбачено відображення відповідних інформативних повідомлень.

Адаптивність інтерфейсу:

Хоча пріоритетною для модуля читання (ReaderActivity) розглядалася горизонтальна орієнтація (для кращого відображення паралельних текстів), а для інших розділів – вертикальна, було докладено зусиль для забезпечення функціональності та візуальної коректності в обох орієнтаціях для більшості екранів. Для ключових операцій, що вимагають введення даних або відображення значного обсягу інформації (налаштування тесту, додавання/редагування слів, результати тесту), використовувалися DialogFragment. Такий підхід забезпечує збереження стану та введених даних при зміні орієнтації екрану, покращуючи користувацький досвід.

При верстці макетів застосовувалися гнучкі контейнери (наприклад, LinearLayout з вагами, ConstraintLayout) та відносні одиниці виміру (dp, sp), що сприяє коректному масштабуванню інтерфейсу на пристроях з різною щільністю та розміром екрану. Списки, реалізовані через RecyclerView, динамічно адаптуються до кількості контенту.

3.5 Забезпечення безпеки та збереження даних користувачів

Забезпечення безпеки та конфіденційності даних користувачів є ключовим пріоритетом при розробці мобільного застосунку. У цьому підрозділі

детально описано механізми, реалізовані для захисту облікових записів, автентифікації, авторизації та передачі даних.

3.5.1 Реєстрація та автентифікація користувачів

Процес реєстрації та автентифікації є першою лінією захисту доступу до даних користувача.

Реєстрація:

Користувач надає email та пароль. Здійснюється клієнтська валідація формату email (рис. 3.21) та серверна перевірка унікальності email. Пароль гешується перед збереженням. Успішна реєстрація веде до екрану входу.

```
// Set up click listener for the registration button
binding.registerButton.setOnClickListener {
    val email = binding.emailEditTextRegister.text.toString().trim()
    val password = binding.passwordEditTextRegister.text.toString().trim()

    // Validate input fields before making an API call
    if (email.isEmpty() || password.isEmpty()) {
        Log.w(TAG, msg: "Registration attempt with empty email or password.")
        Toast.makeText( context: this, "Please enter email and password", Toast.LENGTH_SHORT).show()
        return@setOnClickListener // Exit if validation fails
    }

    if (!Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
        Log.w(TAG, msg: "Registration attempt with invalid email format: $email")
        Toast.makeText( context: this, "Please enter a valid email address", Toast.LENGTH_SHORT).show()
        return@setOnClickListener // Exit if email format is invalid
    }
```

Рис. 3.21 – Валідація електронної пошти

Автентифікація (Вхід):

Для входу в систему користувач вводить свої облікові дані. Сервер перевіряє їх, порівнюючи наданий пароль зі збереженим гешем (рис. 3.22). При успіху генерується JWT для сесії, який клієнт зберігає (рис. 3.23).

```
// Handles user login
const loginUser = async (email, password) => {
  const user = await userModel.getUserByEmail(email);
  // Check user and password
  if (!user || !(await bcrypt.compare(password, user.password_hash))) {
    return { status: 400, body: { message: 'Invalid email or password' } };
  }

  // Generate JWT token
  const token = jwt.sign(
    { id: user.id, email: user.email },
    process.env.JWT_SECRET,
    { expiresIn: '7d' }
  );

  return {
    status: 200,
    body: { message: 'Login successful', token }
  };
};
```

Рис. 3.22 – Функція логіну з генерацією JWT

```kotlin
fun saveToken(context: Context, token: String) {
    // Use applicationContext to avoid potential leaks if a short-lived context is passed.
    with(getSharedPreferences(context.applicationContext).edit()) {
        putString(AUTH_TOKEN_KEY, token)
        putBoolean("isLoggedIn", true) // Also manages a simple logged-in flag.
        apply() // apply() is asynchronous and preferred over commit().
    }
    Log.d( tag: "AuthTokenManager",  msg: "Token saved successfully.")
}
```

Рис. 3.23– Функція збереження JWT

3.5.2 Безпечне зберігання облікових даних

Зберігання паролів у відкритому вигляді є неприпустимою практикою. Для захисту паролів користувачів використовується одностороннє гешування.

Гешування паролів:

Використовується бібліотека bcrypt з параметром cost factor 10 та автоматичною генерацією "солі" для кожного пароля. Це забезпечує надійний односторонній захист від відновлення паролів. [37]

Перевірка пароля при вході:

Сервер безпечно порівнює геш введеного пароля зі збереженим у базі даних за допомогою функції bcrypt.compare.

3.5.3 Автентифікація та авторизація за допомогою JWT

Для керування сесіями користувачів та авторизації запитів до API використовується стандарт JWT (JSON Web Token).

JWT – це компактний, URL-безпечний спосіб представлення заяв (claims) для передачі між двома сторонами. Токен складається з трьох частин, розділених крапками: Заголовок (Header), Корисне навантаження (Payload), Підпис (Signature). [38]

Генерація та використання JWT:

Після успішної автентифікації сервер генерує підписаний JWT, що містить ідентифікатор користувача та термін дії. Клієнт зберігає цей токен і

додає його до заголовка Authorization (як Bearer <token>) усіх наступних запитів до захищених API (рис. 3.24).

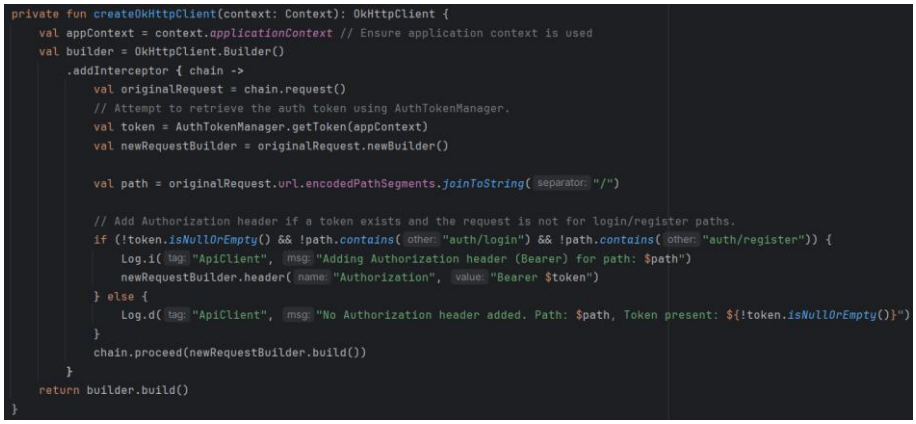


Рис. 3.24 – Створення OkHttpClient з використанням токена

Валідація JWT на сервері:

Спеціальний middleware (authenticate.js) на сервері перевіряє підпис та термін дії кожного JWT за допомогою секретного ключа (рис. 3.25). При успішній валідації доступ до ресурсу надається.

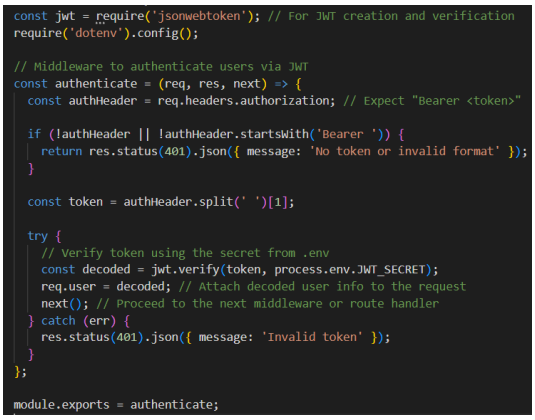


Рис. 3.25 – Валідація JWT

ВИСНОВОК ДО РОЗДІЛУ 3

У третьому розділі дипломного проєкту було успішно реалізовано основний функціонал мобільного застосунку для мультилінгвального читання книжок, що відповідає поставленим вимогам. Ключовим аспектом стала розробка та впровадження ефективного підходу до підготовки контенту з використанням інструменту LF_aligner та JSON-структуризації, що забезпечило точне паралельне відображення текстів.

Було реалізовано модулі каталогу книг, персональної бібліотеки, а також інтерфейс застосунку для читання з можливістю паралельного відображення тексту на 2-4 мовах, навігацією та інтерактивним перекладом слів через API MyMemory з візуальним підсвічуванням. Для підтримки вивчення лексики створено персональний словник з функціями додавання, редагування, видалення та фільтрації слів, а також модуль практичного тестування знань з налаштуванням параметрів та переглядом історії. Також було розроблено інтуїтивно зрозумілий користувацький інтерфейс з належним зворотним зв'язком.

Особливу увагу було приділено забезпеченню безпеки даних користувачів шляхом впровадження механізмів реєстрації та автентифікації, гешування паролів за допомогою bcrypt та використання JWT для авторизації й управління сесіями.

Таким чином, у третьому розділі було продемонстровано повний цикл розробки ключових функціональних блоків мобільного застосунку, від підготовки даних до реалізації користувацьких сценаріїв та забезпечення базових аспектів безпеки. Створений програмний продукт є функціональним прототипом, що закладає міцну основу для подальшого тестування, вдосконалення та можливого розширення функціоналу в майбутньому.

РОЗДІЛ 4
ОГЛЯД ТА ТЕСТУВАННЯ РОЗРОБЛЕНОГО МОБІЛЬНОГО ЗАСТОСУНКУ

4.1 Функціональне тестування основних модулів

Функціональне тестування проводилося для перевірки коректності роботи ключових компонентів та користувацьких сценаріїв мобільного застосунку.

4.1.1 Процес автентифікації та реєстрації користувачів

Модуль автентифікації забезпечує захищений доступ до функціоналу застосунку. Стартовий екран (рис. 4.1) дозволяє користувачеві увійти з існуючими обліковими даними або перейти до створення нового облікового запису.

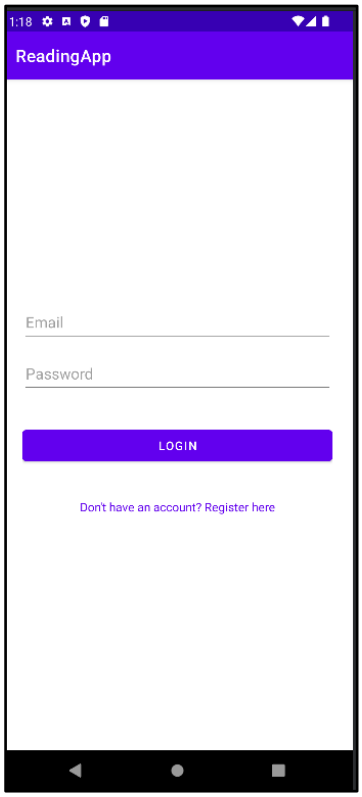


Рис. 4.1 – Стартовий екран входу

Процес реєстрації (рис. 4.2) включає введення електронної пошти та пароля, при цьому здійснюється клієнтська валідація коректності формату введених даних.

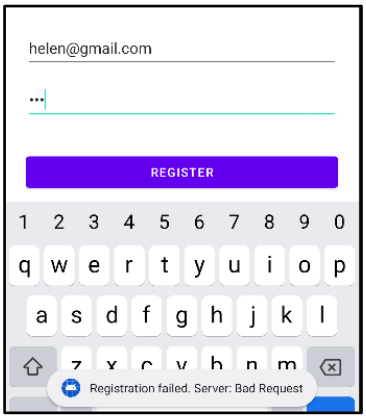


Рис. 4.2 – Екран реєстрації користувача та приклад обробки некоректних даних

Система також перевіряє унікальність email та коректність облікових даних при спробі входу, інформуючи користувача у разі виникнення помилок.

4.1.2 Взаємодія з каталогом книг та персональною бібліотекою

Після успішної автентифікації користувач отримує доступ до загального каталогу книг (рис. 4.3).

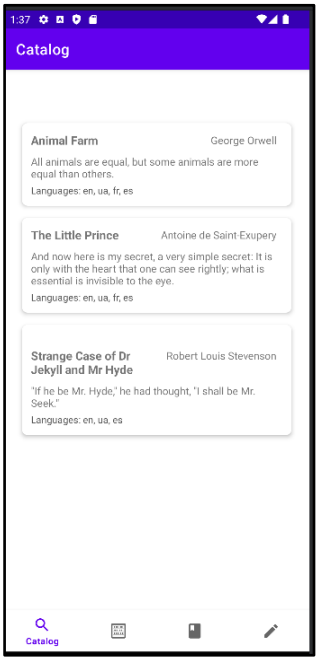


Рис. 4.3 – Відображення загального каталогу книг

При виборі книги з каталогу, користувачеві пропонується обрати від двох до чотирьох мов для паралельного читання через діалогове вікно (рис. 4.4), з відповідною перевіркою кількості обраних мов.

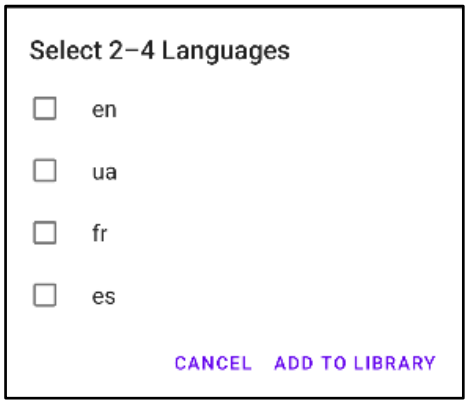


Рис. 4.4 – Діалогове вікно вибору мов для додавання книги до бібліотеки

Успішно додані книги відображаються в персональній бібліотеці користувача (рис. 4.5). З екрану бібліотеки можна керувати книгами: змінювати раніше обрані мови або видаляти книгу з бібліотеки (після підтвердження).

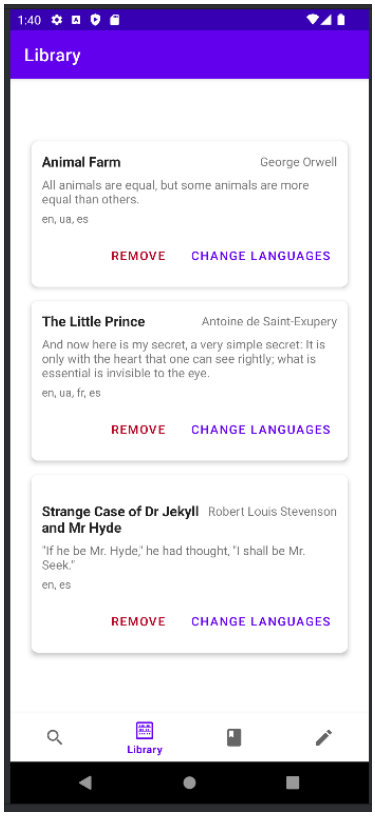


Рис. 4.5 – Екран персональної бібліотеки користувача

4.1.3 Робота з функціоналом застосунку для читання

Застосунок для читання є основним інструментом для взаємодії з мультилінгвальним контентом. Вона дозволяє паралельне відображення тексту обраними мовами, адаптуючись до їх кількості та орієнтації екрану, а також відображає ілюстрації, інтегровані в текст (рис. 4.6, рис. 4.7). Навігація між абзацами (сторінками) здійснюється за допомогою екранних кнопок або свайп-жестів, супроводжуючись індикацією поточної позиції в книзі.

Рис. 4.6 – Інтерфейс застосунку для читання: паралельне відображення тексту книги двома мовами у вертикальній орієнтації із зображенням

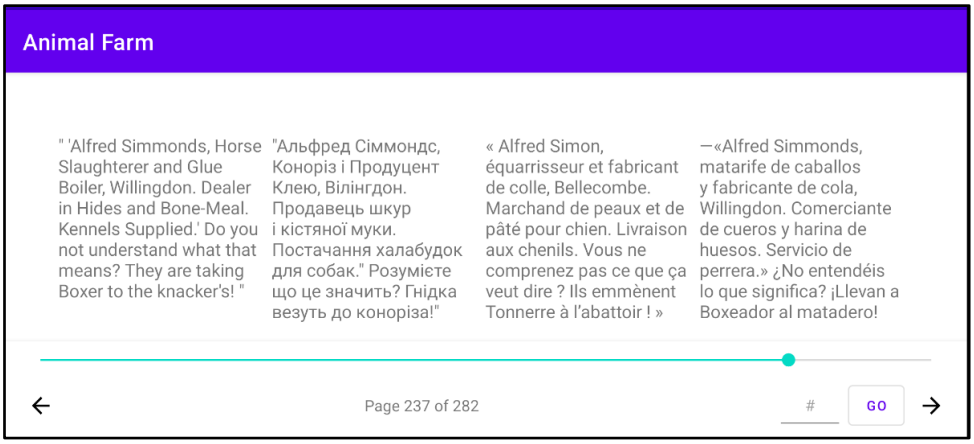


Рис. 4.7 – Інтерфейс застосунку для читання: паралельне відображення тексту книги чотирма мовами у горизонтальній орієнтації із навігацією

Довге натискання на слово активує функцію перекладу: слово виділяється, і у спливаючому вікні відображається його переклад на інші обрані мови, а також підсвічуються відповідники в паралельних текстах (рис. 4.8). З цього ж вікна слово можна додати до персонального словника, про що користувач отримує сповіщення.

Рис. 4.8 – Відображення перекладу та підсвічування слів у текстах

Прогрес читання автоматично зберігається та відновлюється при повторному відкритті книги.

4.1.4 Використання персонального словника

Персональний словник (рис. 4.9) надає користувачеві можливість переглядати та керувати списком збережених слів.

Рис. 4.9 – Загальний вигляд екрану персонального словника зі списком збережених слів

Реалізовано функції редагування перекладів існуючого слова (рис. 4.10) та ручного додавання нових слів з їх перекладами (рис. 4.11). Система включає валідацію введених даних для уникнення помилок, таких як дублювання мов або порожні поля.

Рис. 4.10 – Інтерфейс редагування перекладів у персональному словнику

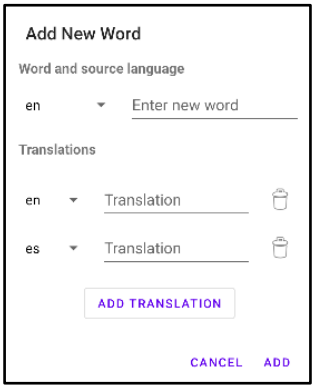


Рис. 4.11 – Форма для ручного додавання нового слова до персонального словника

Для зручності роботи передбачено текстовий пошук (рис. 4.12) та фільтрацію слів за мовами (рис. 4.13).

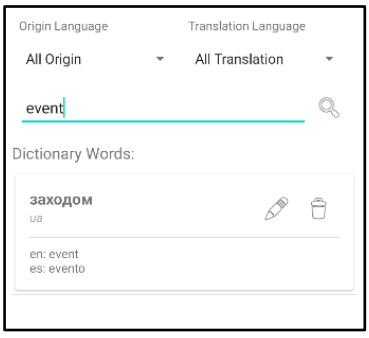


Рис. 4.12 – Результат застосування пошуку слова

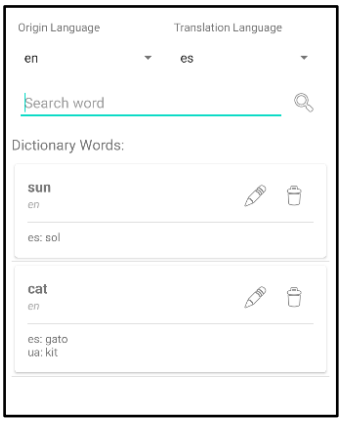


Рис. 4.13 – Результат застосування фільтрації за мовами

Видалення слів зі словника відбувається після підтвердження користувачем.

4.1.5 Проходження практичного тестування знань

Модуль практики дозволяє перевіряти знання вивченої лексики. Процес починається з налаштування параметрів тесту, таких як вибір мов та кількості питань, яка динамічно адаптується до наявних слів у словнику (рис. 4.14).

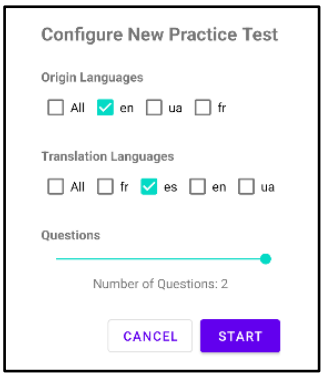


Рис. 4.14 – Налаштування параметрів нового тесту з динамічною кількістю питань

Інтерфейс тестування (рис. 4.15) відображає слово для перекладу, поле для введення відповіді та індикатор прогресу.

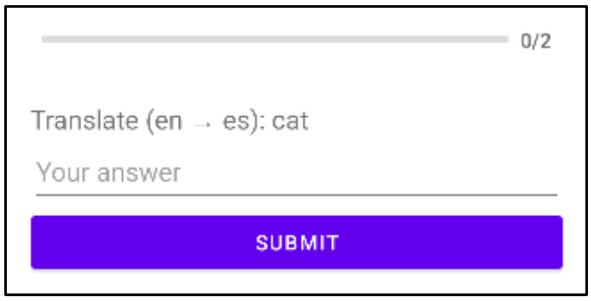


Рис. 4.15 – Інтерфейс проходження тесту

Після введення відповіді система надає миттєвий зворотний зв'язок про її правильність, показуючи вірний варіант у разі помилки (рис. 4.16).

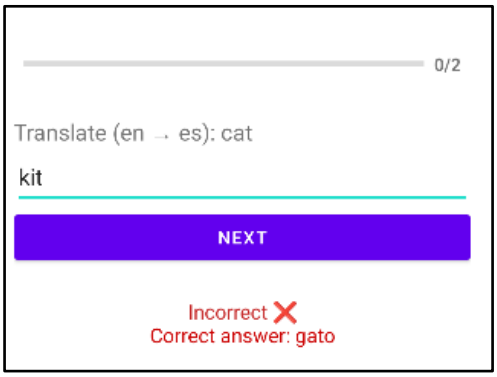


Рис. 4.16 – Зворотний зв'язок на відповідь користувача

По завершенню тесту відображаються детальні результати (рис. 4.17).

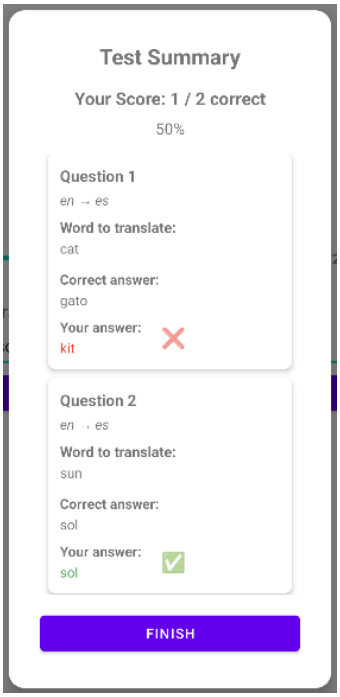


Рис. 4.17 – Відображення результатів після завершення тестування

Історія пройдених тестів доступна в "Хабі практики" (рис. 4.18), де користувач може переглядати попередні результати, застосовувати фільтри (рис. 4.19) та видаляти окремі записи тестів.

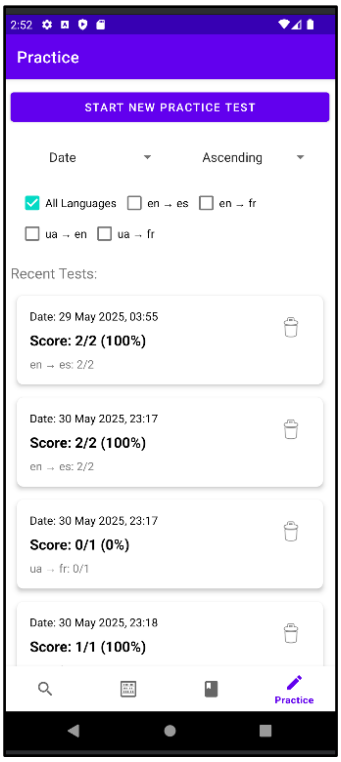


Рис. 4.18 – Екран хабу практики

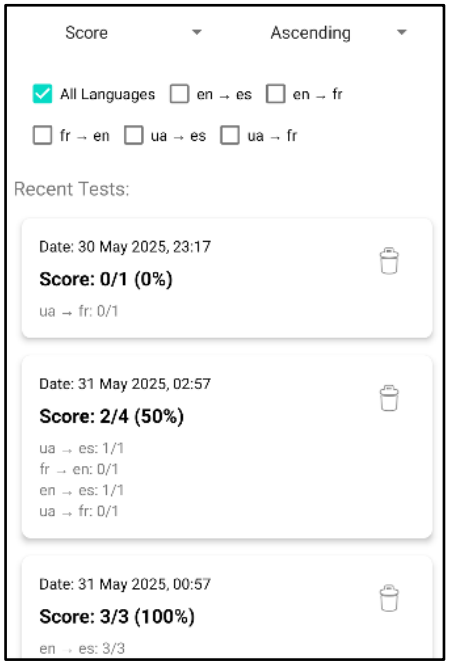


Рис. 4.19 – Застосування фільтрації до історії тестів: за результатом (зростання)

4.2 Оцінка продуктивності застосунку на різних пристроях

Враховуючи, що розробка та первинне тестування мобільного застосунку здійснювалися переважно в середовищі Android Studio з використанням Android Emulator, оцінка продуктивності мала певні особливості.

Реальна продуктивність на фізичних пристроях може відрізнятися через різницю в апаратних характеристиках (тип процесора, швидкість пам'яті, графічний прискорювач), оптимізації виробника ОС та фонові процеси.

Також емулятори зазвичай мають стабільне та швидке підключення до мережі, що може не відображати реальні умови використання мобільного інтернету.

4.2.1 Методологія тестування продуктивності в умовах емуляції

Під час тестування продуктивності увага приділялася наступним ключовим аспектам:

- *Час запуску застосунку:* від ініціації до повної готовності головного екрану.
- *Швидкість завантаження даних:* завантаження списків книг (каталог, бібліотека) та контенту обраної книги для читання.
- *Плавність інтерфейсу:* оцінка плавності навігації між екранами, прокручування списків та гортання сторінок у читалці.
- *Час відгуку:* реакція застосунку на дії користувача (натискання, введення).
- *Використання ресурсів (орієнтовно):* моніторинг споживання CPU та пам'яті за допомогою Android Profiler під час виконання основних сценаріїв.

4.2.2 Конфігурації тестових пристроїв (емулятори)

Застосунок розроблено з minSdk = 26 (Android 8.0 Oreo) та targetSdk = 34 (Android 14), що забезпечує сумісність з широким спектром пристроїв.

Для оцінки роботи застосунку в різних умовах було обрано дві ключові конфігурації віртуальних пристроїв (AVD) в середовищі Android Emulator:

- Пристрій 1 (телефон): API 31 (Android 12), 2048 МБ RAM, 1080x2400px.
- Пристрій 2 (телефон): API 27 (Android 8.1), 1024 МБ RAM, 720x1280px.
- Пристрій 3 (планшет): API 30 (Android 11), 1907 МБ RAM, 2560x1600px.

4.2.3 Результати оцінки продуктивності на емуляторі

Оцінка продуктивності на обраних конфігураціях емуляторів проводилася шляхом виконання типових користувацьких сценаріїв та моніторингу ключових показників за допомогою інструментів Android Studio, зокрема Android Profiler. Нижче наведено узагальнені результати та спостереження.

Запуск та завантаження даних:

На всіх трьох тестових конфігураціях емуляторів час "холодного" запуску застосунку (від моменту натискання іконки до повної готовності головного екрану) був мінімальним, зазвичай не перевищуючи однієї секунди. "Теплий" запуск (коли застосунок вже був у пам'яті) відбувався практично миттєво на всіх конфігураціях. Завантаження списків книг та контенту тестових книг відбувалося швидко, зазвичай менше однієї секунди.

Плавність інтерфейсу та час відгуку:

Навігація між екранами, прокручування списків та гортання сторінок у читалці були плавними на всіх тестових конфігураціях. Реакція на базові дії користувача була миттєвою. Затримки спостерігалися лише під час мережевих операцій (наприклад, отримання перекладу, синхронізація з сервером), що є очікуваним та оброблялося асинхронно.

Використання ресурсів (орієнтовна оцінка за допомогою Android Profiler):

- CPU:

Навантаження на CPU під час читання було стабільним (близько 30% на планшеті). Короткочасні піки спостерігалися під час переходів між екранами та виконання запитів до API (наприклад, до 58% під час запиту перекладу), що є типовим для інтерактивних застосунків (рис. 4.20). У стані очікування навантаження падало практично до нуля.

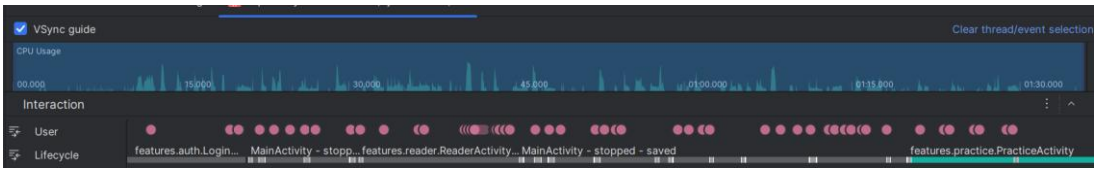


Рис. 4.20 – Графік навантаження CPU пристрою 3 (планшету)

- *Пам'ять (Memory):*

Споживання пам'яті було в межах очікуваного, з піковими значеннями під час активного використання модуля практики (до ~280 МБ виділеної купи на планшеті, до ~126 МБ на менш потужному смартфоні) (рис. 4.21, рис. 4.22). Важливим є ефективне звільнення ресурсів після завершення ресурсоємних операцій, що було підтверджено спостереженнями. На емуляторі з меншим обсягом RAM спостерігалися частіші короткочасні коливання використання пам'яті ("шипи"), що може вказувати на активнішу роботу збирача сміття.

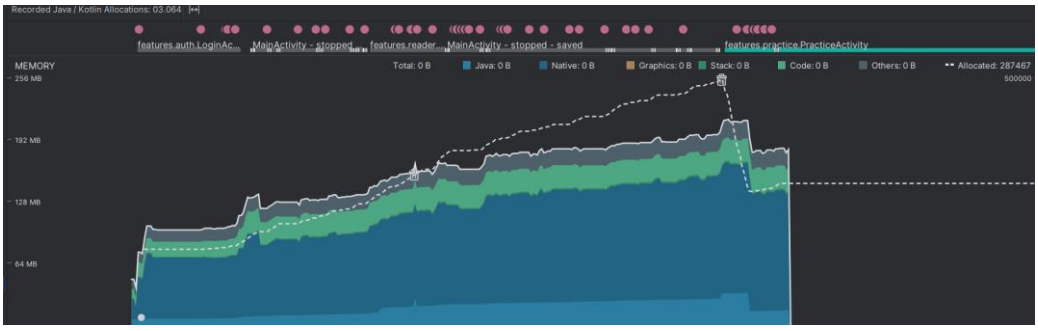


Рис. 4.21 – Графік використання пам’яті пристроєм 3 (планшетом)

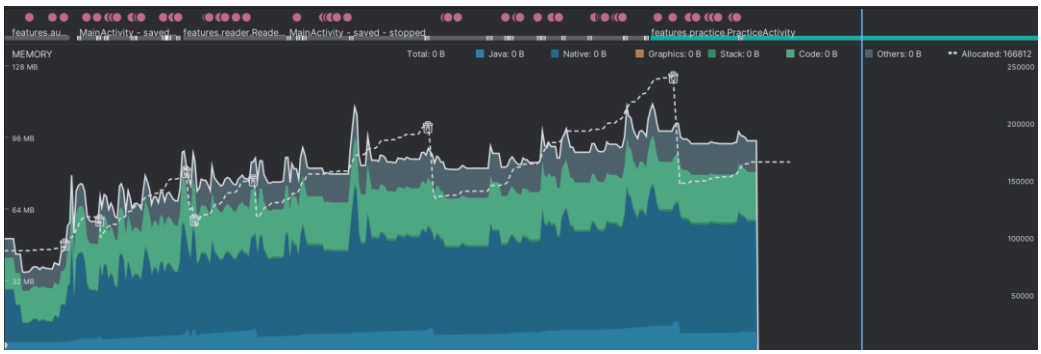


Рис. 4.22 - Графік використання пам’яті пристроєм 2 (телефон)

Загальні спостереження:

На основі тестування в середовищі емуляції, розроблений мобільний застосунок демонструє стабільну та переважно високу продуктивність для виконання ключових функціональних сценаріїв. Оптимізація асинхронних операцій та використання стандартних компонентів Android сприяють плавності роботи. Зафіксовані показники часу відгуку та використання ресурсів є прийнятними для поточного етапу розробки. Хоча результати на емуляторі є орієнтовними, вони дають позитивну попередню оцінку.

4.3 Перспективи розвитку застосунку

Розроблений мобільний застосунок для мультилінгвального читання книжок є функціональним прототипом, що демонструє життєздатність основної концепції та має значний потенціал для подальшого вдосконалення та розширення.

Посилення безпеки та надійності:

Пріоритетним завданням є перехід на протокол HTTPS для шифрування всіх комунікацій. Також доцільно впровадити більш надійні механізми автентифікації, такі як Refresh Tokens, посилення політик паролів, захист від brute-force атак, двофакторна автентифікація (2FA) та можливість OAuth 2.0 авторизації. Для забезпечення стабільної роботи та доступності необхідне розгортання на повноцінному сервері.

Розширення та вдосконалення основного функціоналу:

- *Контент та його представлення:* збільшення кількості доступних книг та мовних версій, додавання обкладинок, системи тегування та розширеної фільтрації для каталогу та бібліотеки.
- *Застосунок для читання:* впровадження навігації по змісту, інтеграція з більш гнучкими API перекладу (з варіантами), можливість додавання контексту до слів у словнику, реалізація нотаток, закладок, налаштувань відображення тексту (шрифти, розміри, теми) та звукового супроводу (Text-to-Speech, синхронізовані аудіодоріжки).

Розвиток інструментів для вивчення слів:

Розширення модуля тестування знань новими типами вправ (наприклад, картки для співставлення), впровадження системи збору детальної статистики прогресу вивчення слів з візуалізацією та можливістю повторного проходження тестів.

Покращення доступності та користувацького досвіду:

Реалізація офлайн-режиму для читання завантажених книг та роботи зі словником/тестами без постійного підключення до Інтернету. Подальше покращення адаптивності інтерфейсу для коректного та зручного відображення на планшетах та пристроях з різними співвідношеннями сторін екрану.

ВИСНОВОК ДО РОЗДІЛУ 4

У четвертому розділі було проведено комплексний огляд та тестування розробленого мобільного застосунку для мультилінгвального читання. Результати функціонального тестування підтвердили коректну працездатність всіх ключових модулів, включаючи автентифікацію, управління каталогом та бібліотекою, основний функціонал застосунку для читання з паралельним відображенням та інтерактивним перекладом, персональний словник з усіма CRUD-операціями, а також модуль практичного тестування знань. Більшість користувацьких сценаріїв виконуються відповідно до очікувань. Оцінка продуктивності на різних конфігураціях емуляторів продемонструвала високу швидкість відгуку, плавність інтерфейсу та ефективне використання ресурсів (CPU та пам'яті) з прийнятними показниками часу запуску та завантаження даних, що свідчить про задовільну оптимізацію на поточному етапі.

Визначено перспективи розвитку застосунку, що включають посилення безпеки (HTTPS, Refresh Tokens, 2FA), розширення функціоналу (обкладинки, теги, навігація по змісту, нотатки, звуковий супровід, різноманітні типи тестів), вдосконалення користувацького досвіду (офлайн-режим, адаптивність) та технічні покращення (розгортання на сервері). Таким чином, проведений огляд та тестування підтвердили працездатність розробленого застосунку та намітили шляхи для його подальшого вдосконалення.

ВИСНОВКИ

У ході виконання даного бакалаврського дипломного проєкту було успішно досягнуто поставлену мету – розроблено мобільний застосунок, що надає користувачам зручний інструмент для мультилінгвального читання книг з інтегрованою функцією перекладу, створення персоналізованих словників та самоперевірки знань.

На першому етапі роботи було проведено аналіз існуючих рішень у сфері мобільних застосунків для мультилінгвального читання, розглянуто методи синхронізації текстів та поширені формати електронних книг. Цей аналіз дозволив виявити ключові особливості та недоліки наявних продуктів, а також визначити потенційні напрямки для створення більш ефективного та зручного інструменту. Було встановлено, що якісна синхронізація паралельних текстів, особливо професійних перекладів, є важливим, але складним завданням.

На основі проведеного аналізу та відповідно до технічного завдання, на другому етапі було сформульовано функціональні та нефункціональні вимоги до розроблюваного застосунку та проведено їх пріоритезацію. Було спроектовано клієнт-серверну архітектуру системи, що включає нативний Android-клієнт, серверну частину на Node.js та базу даних PostgreSQL. Також було обґрунтовано вибір технологічного стеку та зовнішніх сервісів, зокрема MyMemory Translated API для перекладу та Cloudflare R2 для зберігання контенту книг, з акцентом на балансі між продуктивністю, швидкістю розробки та економічною доцільністю. Важливим проєктним рішенням для прискорення розробки стала відмова від локальної бази даних на клієнті на користь повної залежності від онлайн-доступу на поточному етапі.

Третій розділ був присвячений безпосередній реалізації програмного продукту. Ключовим аспектом стала розробка ефективного підходу до підготовки контенту з використанням інструменту LF_aligner для вирівнювання абзаців професійних перекладів та їх структурування у JSON-формат, що

забезпечило точне паралельне відображення. Було реалізовано основний функціонал застосунку: автентифікацію користувачів (з гешуванням паролів bcrypt та використанням JWT), каталог книг, персональну бібліотеку, модуль читання з можливістю паралельного читання на 2-4 мовах, інтерактивний переклад слів із підсвічуванням, персональний словник з CRUD-операціями та фільтрацією, а також модуль практичного тестування знань. Особливу увагу приділено інтуїтивності користувацького інтерфейсу та забезпеченню зворотного зв'язку.

У четвертому розділі було проведено огляд та тестування розробленого застосунку. Функціональне тестування підтвердило коректну роботу всіх основних модулів та відповідність більшості користувацьких сценаріїв очікуванням. Оцінка продуктивності на емуляторах показала високу швидкість відгуку та плавність роботи, а також ефективне звільнення ресурсів після виконання ресурсоємних операцій. Також було визначено широкі перспективи подальшого розвитку продукту, включаючи посилення безпеки, розширення функціоналу та покращення користувацького досвіду.

Таким чином, у результаті виконання дипломного проєкту створено функціональний прототип мобільного застосунку для мультилінгвального читання книжок. Розроблений продукт успішно реалізує поставлені завдання, надаючи користувачам інструмент для ефективного вивчення іноземних мов через читання. Проєкт продемонстрував можливість створення комплексного рішення з використанням сучасних технологій та підходів до обробки тексту. Подальша робота над застосунком може перетворити його на повноцінний та конкурентоспроможний продукт на ринку освітніх технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Smart Book. Google Play Store: веб-сайт. URL: https://play.google.com/store/apps/details?id=com.kursx.smartbook&hl=uk (дата звернення: 15.04.2025)
2. What Is Bionic Reading and Should Students Use It? oxfordlearning: веб-сайт. URL: https://www.oxfordlearning.com/what-is-bionic-reading-and-why-should-you-use-it/ (дата звернення: 15.04.2025)
3. BR Method. Bionic-reading: веб-сайт. URL: https://bionic-reading.com/br-method/ (дата звернення: 15.04.2025)
4. Linga. Google Play Store: веб-сайт. URL: https://play.google.com/store/apps/details?id=io.linga&hl=uk (дата звернення: 15.04.2025)
5. Duoreader. Google Play Store: веб-сайт. URL: https://play.google.com/store/apps/details?id=com.duoreader&hl=uk (дата звернення: 16.04.2025)
6. How to make parallel books for language learning. Part 1. Python and Colab version. Medium: веб-сайт. URL: https://medium.com/@averoo/how-to-make-a-parallel-book-for-language-learning-part-1-python-and-colab-version-cff09e379d8c (дата звернення: 16.04.2025)
7. Alignment. memoqdocs: веб-сайт. URL: https://docs.memoq.com/current/en/Concepts/concepts-alignment.html (дата звернення: 16.04.2025)
8. Arvidson T., Siebecke M. IBM Model 4 Alignment Comparison: Degree Project in Computer Science / KTH Royal Institute of Technology, School of Computer Science and Communication. Stockholm. 2016. 31 p. (дата звернення: 17.04.2025)
9. Rastorgueva E. Neural Hidden Markov Models for Word Alignment: Master's thesis / Department of Engineering

University of Cambridge. Cambridge. 2019. 49 p. (дата звернення: 17.04.2025)

10. Vogel S., Ney H., Tillmann C. HMM-based word alignment in statistical translation // Proceedings of the 16th International Conference on Computational Linguistics (COLING-96). – Copenhagen, Denmark: Center for Sprogteknologi, 1996. – Vol. 2. – p. 836–841. (дата звернення: 18.04.2025)
11. A Guide to Formats. bookize: веб-сайт. URL: https://www.bookize.com/ebooks/a-guide-to-formats/ (дата звернення: 19.04.2025)
12. What is an eBook Format? kitaboo: веб-сайт. URL: https://kitaboo.com/what-is-an-ebook-format/ (дата звернення: 19.04.2025)
13. Guide to ebook formats. ebooks: веб-сайт. URL: https://support.ebooks.com/hc/en-gb/articles/213875366-Guide-to-ebook-formats (дата звернення: 19.04.2025)
14. Functional Requirements Examples (Plus Definition and Types). indeed: веб-сайт. URL: https://www.indeed.com/career-advice/career-development/functional-requirements-examples (дата звернення: 22.04.2025)
15. Functional Requirements. thestory: веб-сайт. URL: https://thestory.is/en/process/development-phase/functional-requirements/ (дата звернення: 22.04.2025)
16. What are Functional Requirements? Definition & Examples. chisellabs: веб-сайт. URL: https://chisellabs.com/glossary/what-are-functional-requirements/ (дата звернення: 22.04.2025)
17. What Are Functional Requirements? Types and Examples. winatalent: веб-сайт. URL: https://winatalent.com/blog/what-are-functional-requirements-types-and-examples/ (дата звернення: 22.04.2025)
18. Functional and Nonfunctional Requirements: Specification and Types. altexsoft: веб-сайт. URL: https://www.altexsoft.com/blog/functional-and-non-

functional-requirements-specification-and-types/ (дата звернення: 24.04.2025)
19. Functional Vs. Nonfunctional Requirements: The 2025 Guide. agilemania: веб-сайт. URL: https://agilemania.com/functional-vs-nonfunctional-requirements (дата звернення: 24.04.2025)
20. Non-functional Requirements: What They Do, Examples, and Best Practices. perforce: веб-сайт. URL: https://www.perforce.com/blog/alm/what-are-non-functional-requirements-examples (дата звернення: 24.04.2025)
21. Kotlin і Java: яку мову краще обирати для вивчення. foxminded: веб-сайт. URL: https://foxminded.ua/kotlin-abo-java/ (дата звернення: 25.04.2025)
22. Express.js Tutorial. geeksforgeeks: веб-сайт. URL: https://www.geeksforgeeks.org/express-js/ (дата звернення: 25.04.2025)
23. What is PostgreSQL? ibm: веб-сайт. URL: https://www.ibm.com/think/topics/postgresql (дата звернення: 25.04.2025)
24. What is REST API? ibm: веб-сайт. URL: https://www.ibm.com/think/topics/rest-apis (дата звернення: 25.04.2025)
25. NestJS: The Perfect Javascript Backend Framework for Structure, Clean Code, and Modularity. medium: веб-сайт. URL: https://medium.com/@ajonesb/nestjs-the-perfect-javascript-backend-framework-for-structure-clean-code-and-modularity-ae1b3a6e1418 (дата звернення: 28.04.2025)
26. Introduction to iOS Development: Getting Started with Swift and Xcode. medium: веб-сайт. URL: https://medium.com/@mellowacademy0507/introduction-to-ios-development-getting-started-with-swift-and-xcode-64b4384bcbb7 (дата звернення: 28.04.2025)
27. What is Java Spring Boot? ibm: веб-сайт. URL: https://www.ibm.com/think/topics/java-spring-boot (дата звернення: 28.04.2025)

28. MySQL: Understanding What It Is and How It's Used. oracle: веб-сайт. URL: https://www.oracle.com/mysql/what-is-mysql/ (дата звернення: 28.04.2025)
29. Client-Server Model. geeksforgeeks: веб-сайт. URL: https://www.geeksforgeeks.org/client-server-model/ (дата звернення: 26.04.2025)
30. Client-Server Architecture Explained with Examples, Diagrams, and Real-World Applications. medium: веб-сайт. URL: https://medium.com/nerd-for-tech/client-server-architecture-explained-with-examples-diagrams-and-real-world-applications-407e9e04e2d1 (дата звернення: 26.04.2025)
31. The 5 best free translation APIs for language translation. smartling: веб-сайт. URL: https://www.smartling.com/blog/free-translation-api (дата звернення: 18.05.2025)
32. 9 Translation APIs to Make Your Application Multilingual. geekflare: веб-сайт. URL: https://geekflare.com/dev/translation-apis/ (дата звернення: 18.05.2025)
33. Cloudflare R2 vs. the Big 3: A Deep Dive into Cost and Technical Efficiency of Cloud Storage. medium: веб-сайт. URL: https://y-consulting.medium.com/cloudflare-r2-vs-the-big-3-a-deep-dive-into-cost-and-technical-efficiency-of-cloud-storage-c1644c61a0d3 (дата звернення: 19.05.2025)
34. LF Aligner. l10nsoftaware: веб-сайт. URL: https://www.l10nsoftware.com/lf-aligner/ (дата звернення: 25.05.2025)
35. LF Aligner | langui.ch /'læŋgwɪtʃ. langui: веб-сайт. URL: https://langui.ch/tool/lf-aligner/ (дата звернення: 25.05.2025)
36. TMX Files and Format. transifex: веб-сайт. URL: https://help.transifex.com/en/articles/6838724-tmx-files-and-format (дата звернення: 27.05.2025)

37. Salt and Hash Passwords with bcryp. Hey Node: веб-сайт. URL: thttps://heynode.com/blog/2020-04/salt-and-hash-passwords-bcrypt/ (дата звернення: 29.05.2025)
38. Introduction to JSON Web Tokens. jwt.io: веб-сайт. URL: https://jwt.io/introduction (дата звернення: 29.05.2025)

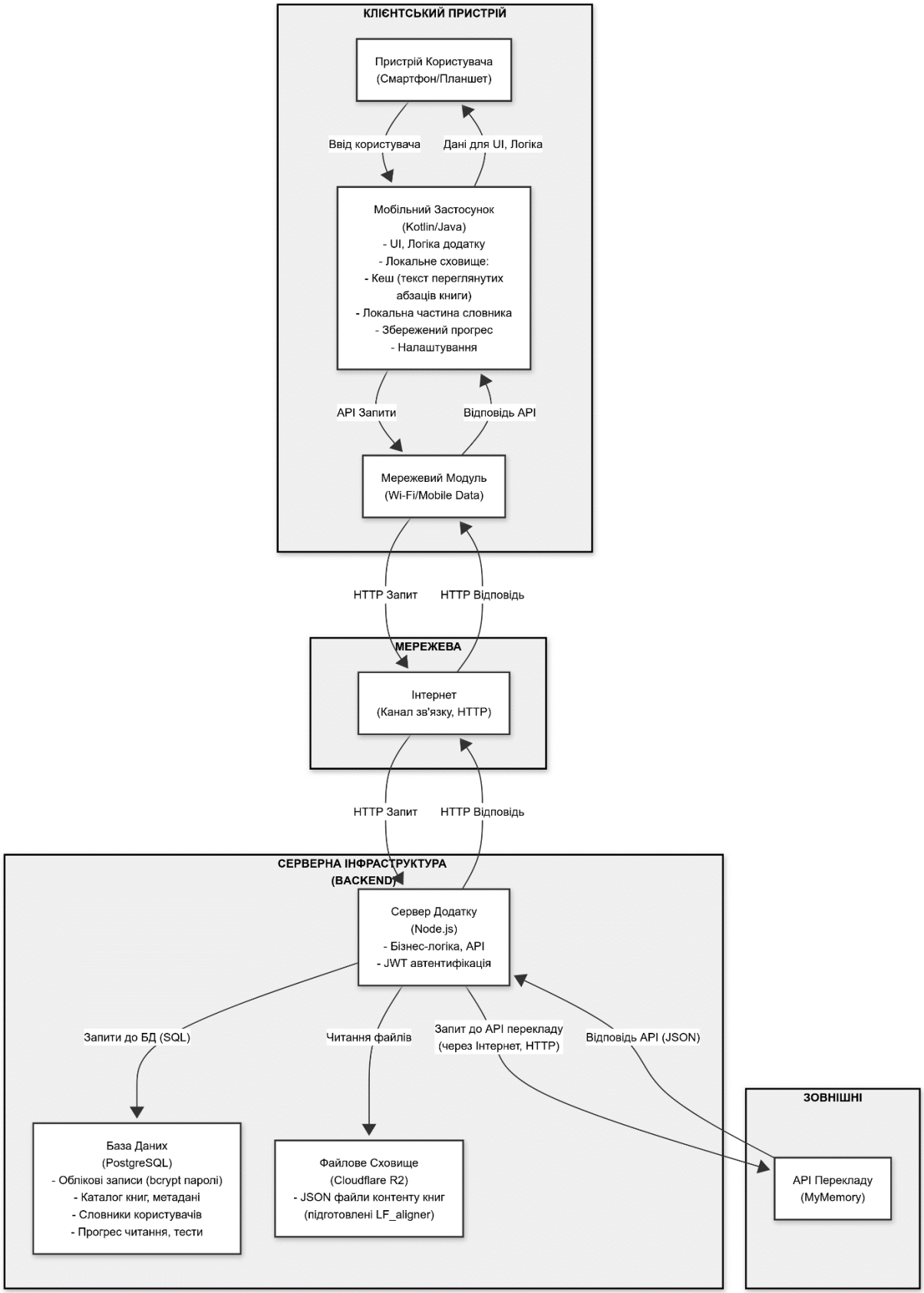
ДОДАТОК 1

Мобільний застосунок для мультилінгвального читання книжок

Структурна схема апаратної компоненти системи
ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.004 Д1	
Зм.	Лист.	№ докум.	Підпис	Дата		
Розробив		Федосєєва О.Д.			**Мобільний застосунок для мультилінгвального читання книжок** Структурна схема апаратної компоненти системи	Літ. / Арк. 1 / Акрушів 1
Перевірив		Васильєва М.Д.				
Н. Контр.						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11
Затвердив						

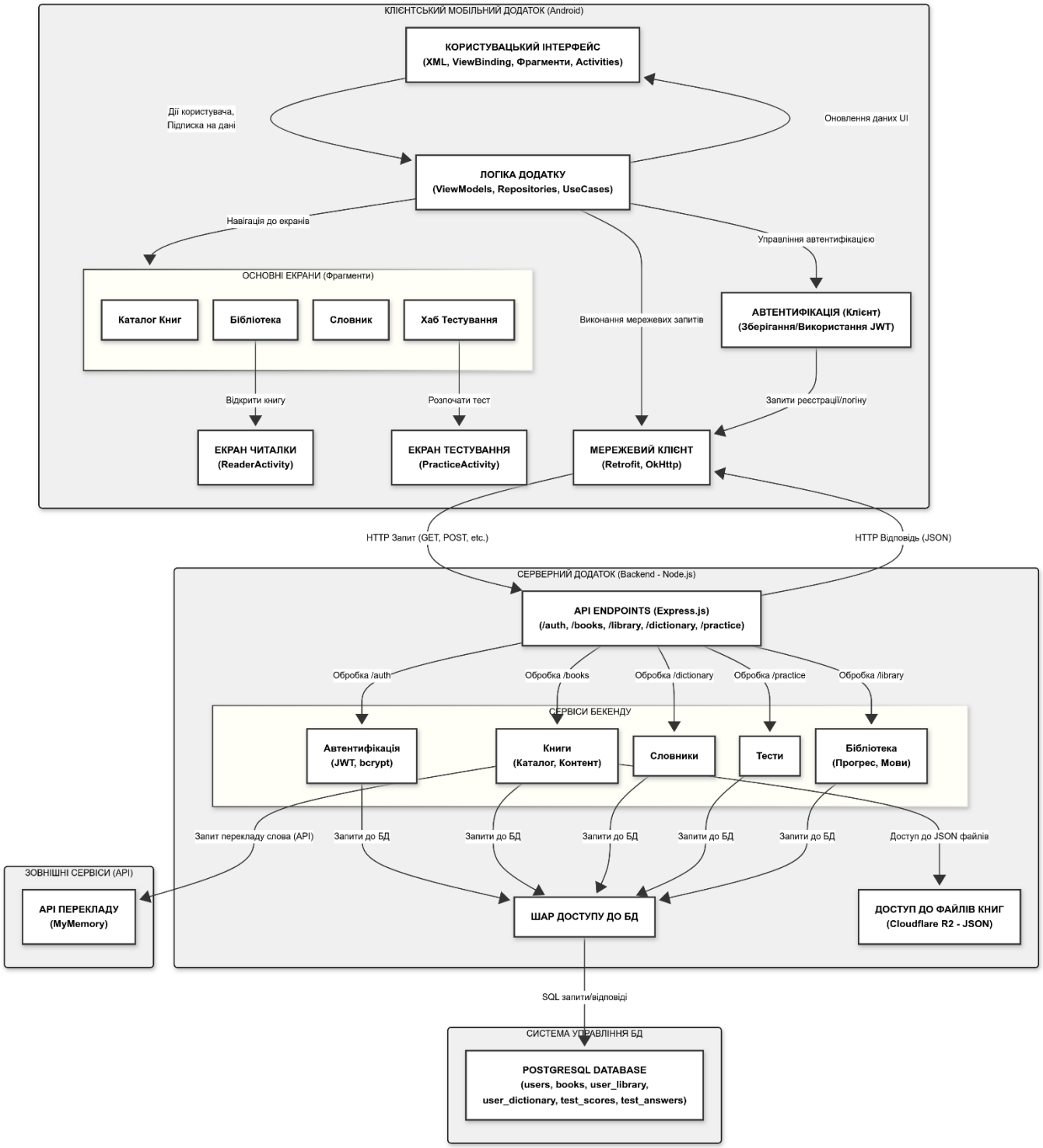
ДОДАТОК 2

Мобільний застосунок для мультилінгвального читання книжок

Структурна схема програмної компоненти системи
ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2025 р.



					ІАЛЦ.467200.005 Д2	
Зм.	Лист.	№ докум.	Підпис	Дата		
Розробив		Федосєєва О.Д.			**Мобільний застосунок для мультилінгвального читання книжок** Структурна схема програмної компоненти системи	Літ. / Арк. 1 / Акрушів 1
Перевірив		Васильєва М.Д.				КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11
Н. Контр.						
Затвердив						

ДОДАТОК 3

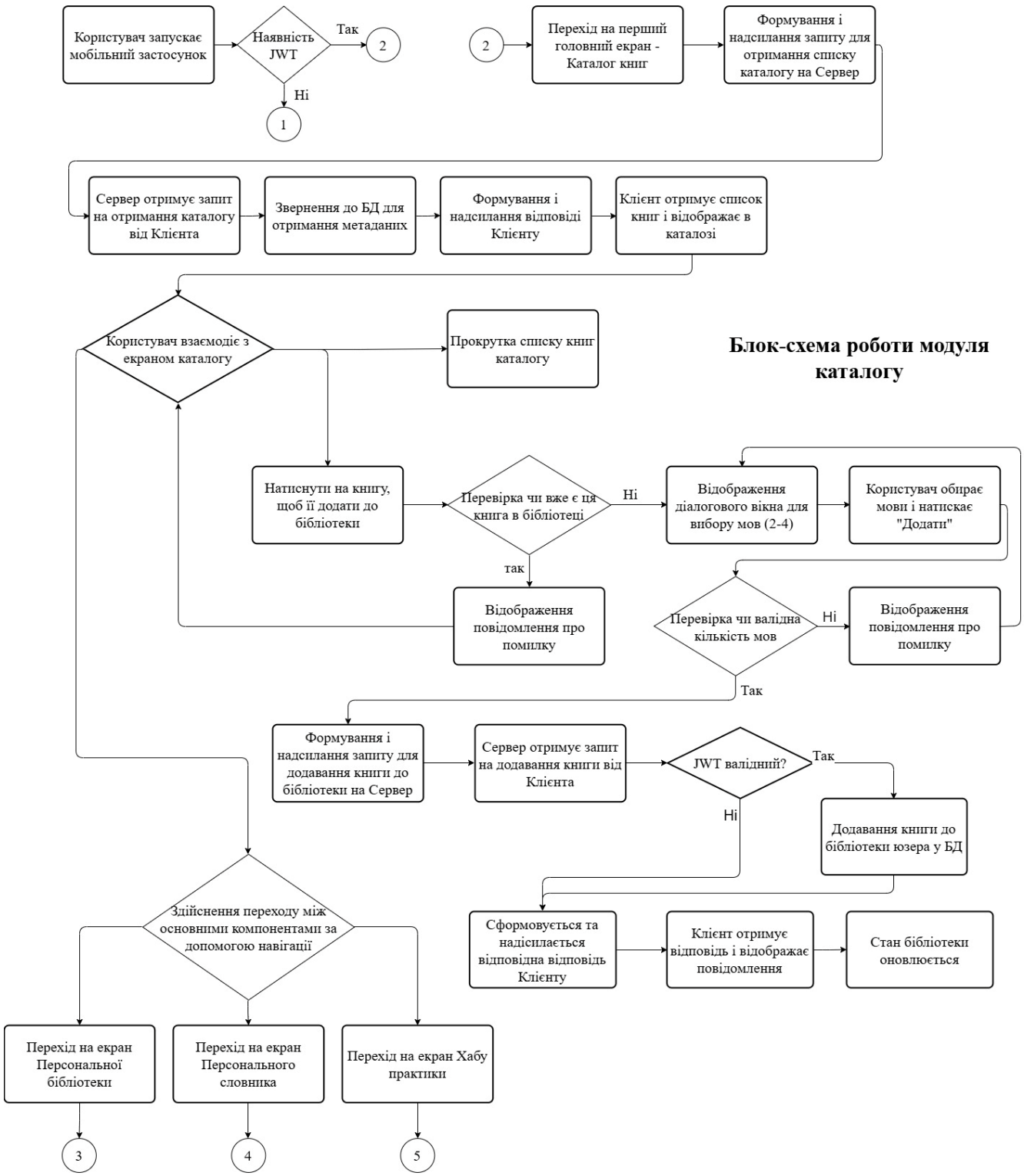
Мобільний застосунок для мультилінгвального читання книжок

Блок-схеми алгоритмів функціонування програмного продукту

ІАЛЦ.467200.004 Д3

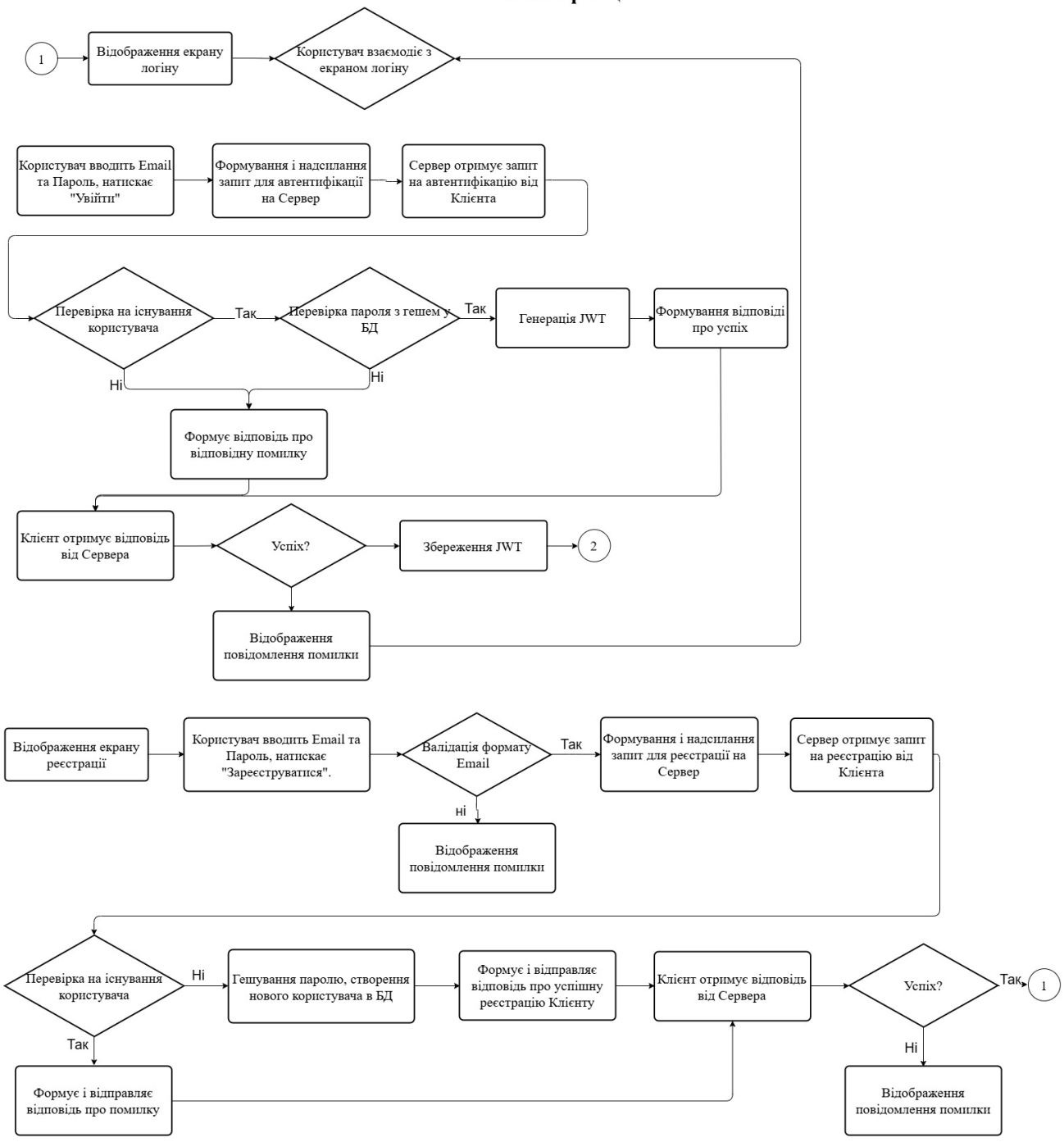
Аркушів 9

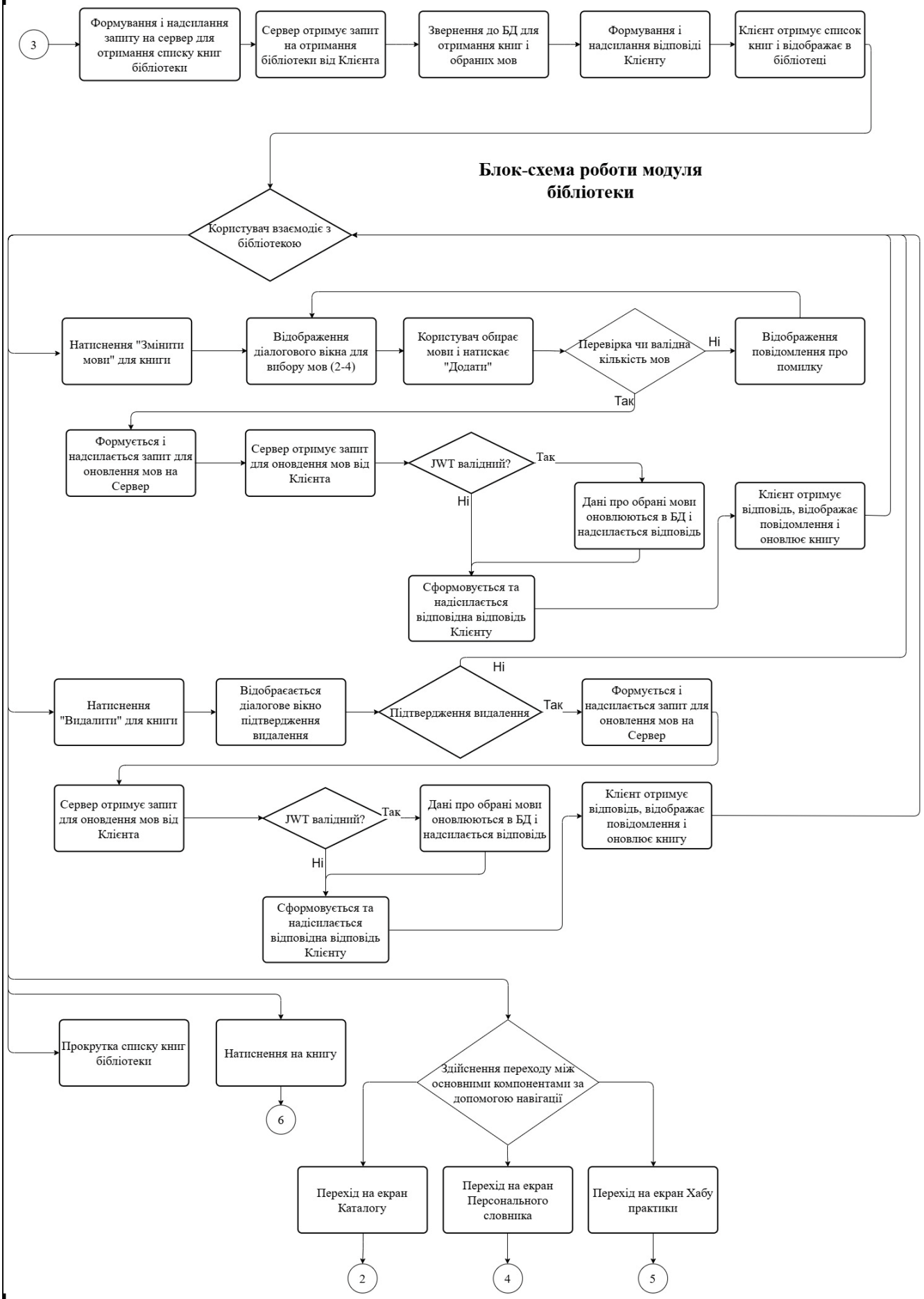
Київ 2025 р.



					ІАЛЦ.467200.006 Д3		
Зм.	Лист.	№ докум.	Підпис	Дата			
Розробив		Федосєєва О.Д.			**Мобільний застосунок для мультилінгвального читання книжок** Блок-схеми алгоритмів функціонування програмного продукту	Літ.	Арк. 1 / Акрушів 9
Перевірив		Васильєва М.Д.					
Н. Контр.						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11	
Затвердив							

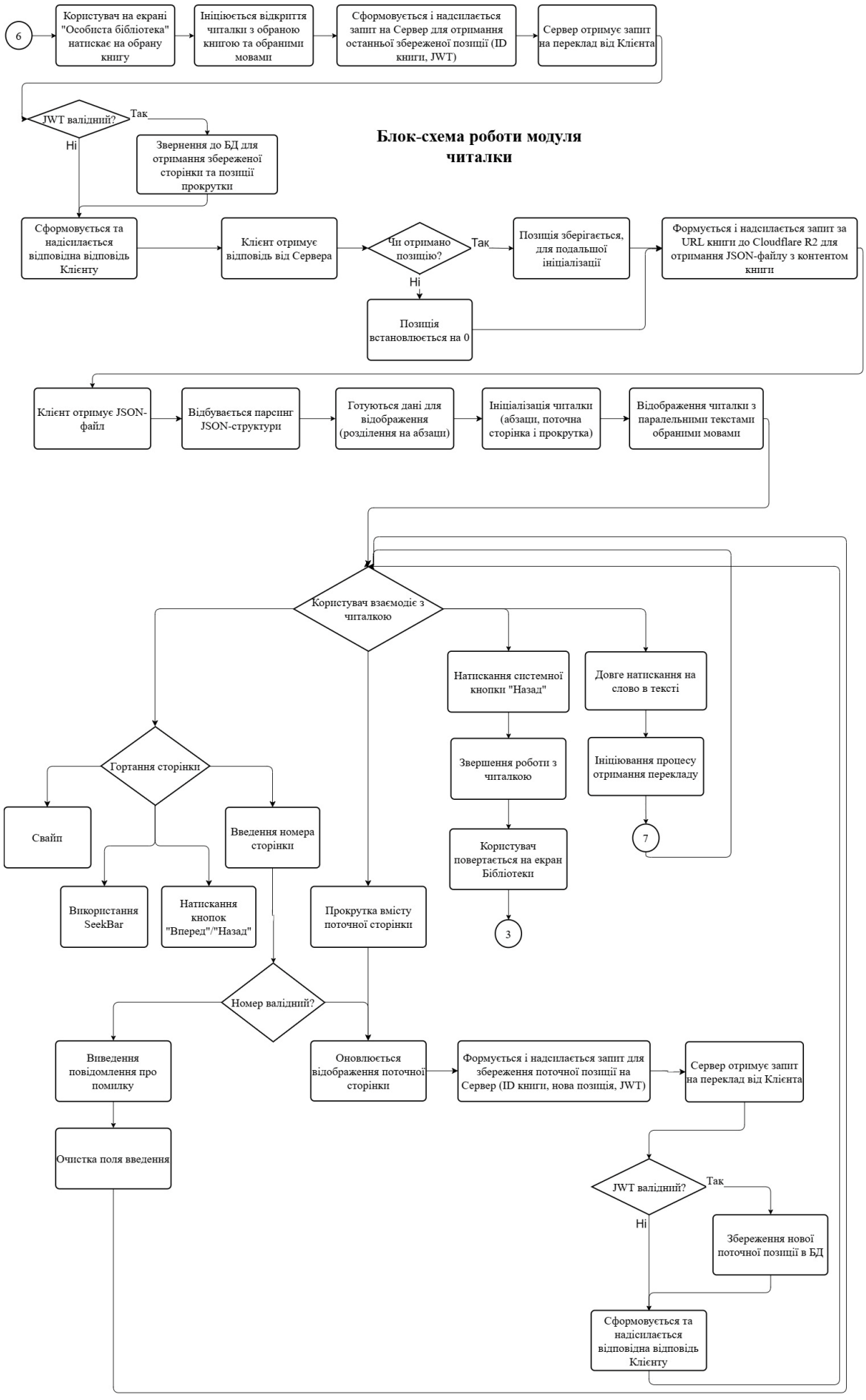
Блок-схема роботи модулів автентифікації



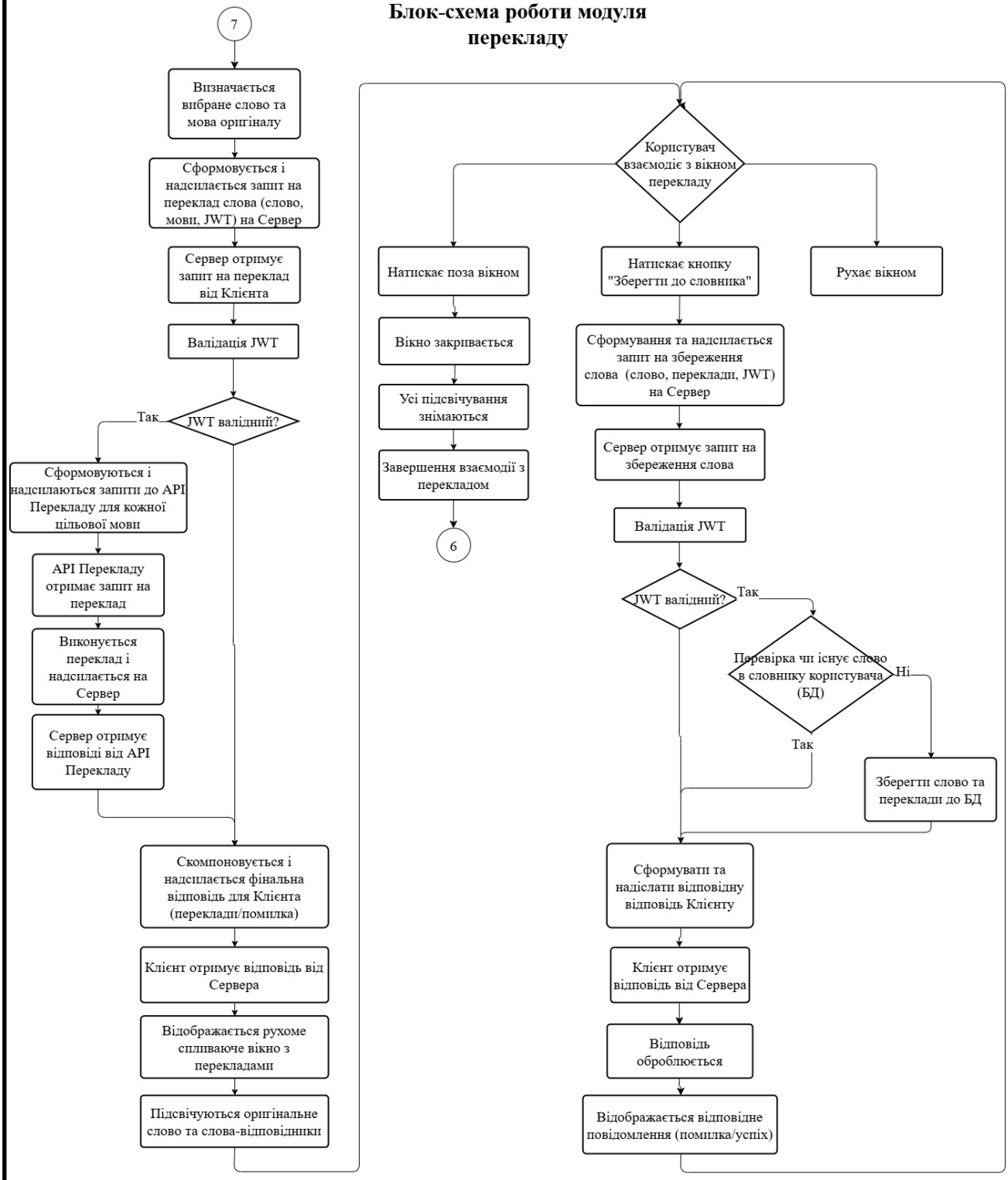


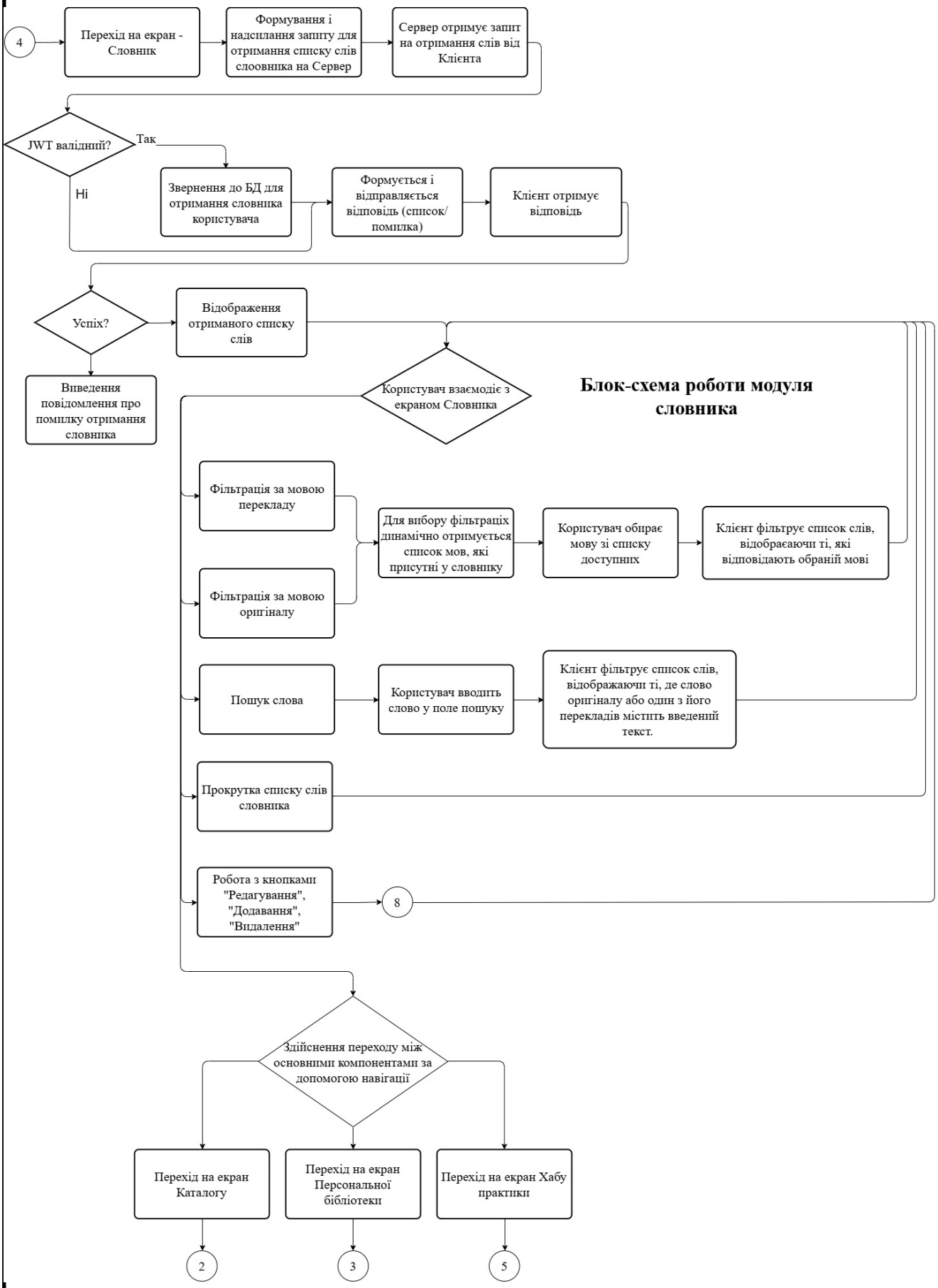
Змн.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467200.006 Д3	Арк. 3

Блок-схема роботи модуля читалки



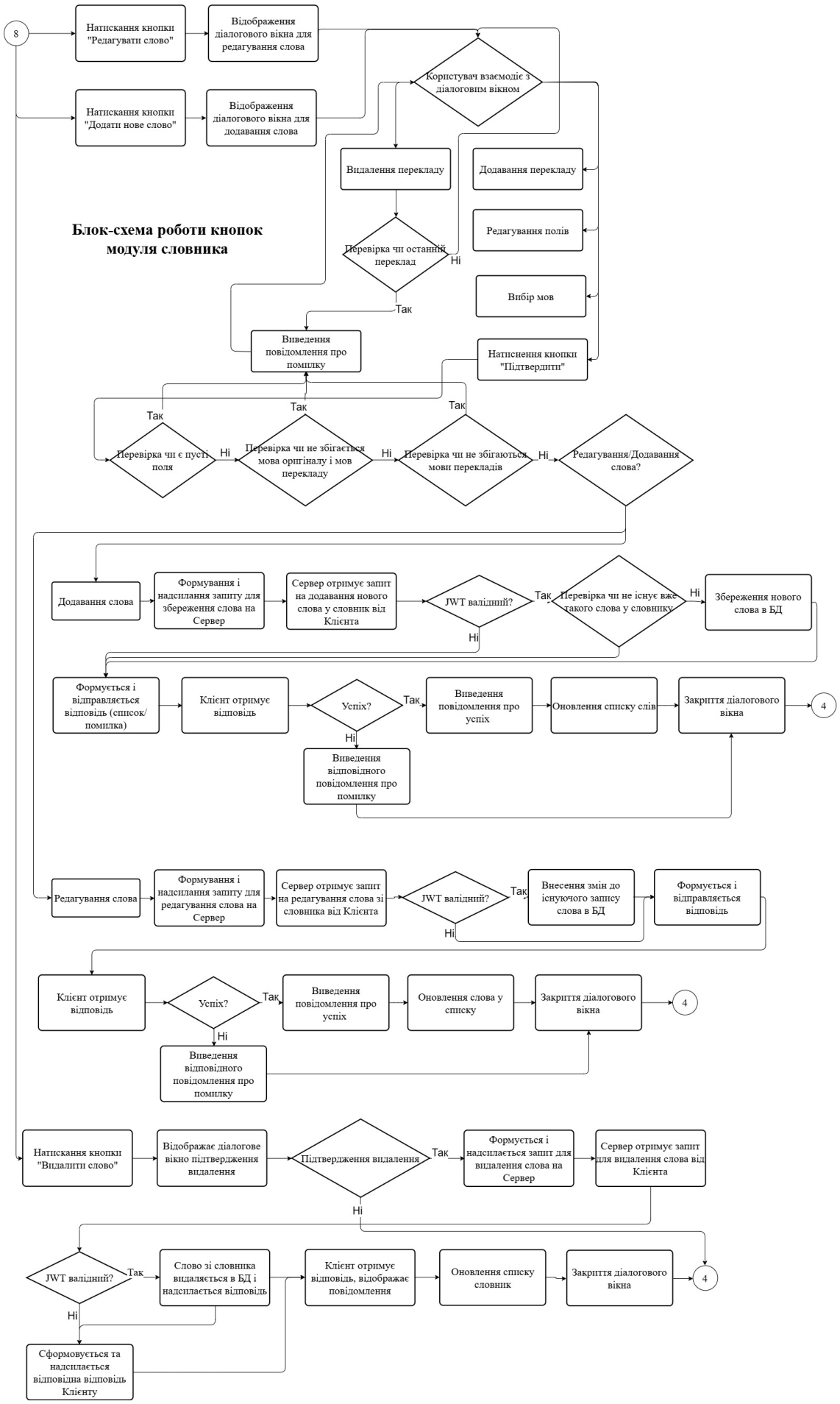
Блок-схема роботи модуля перекладу

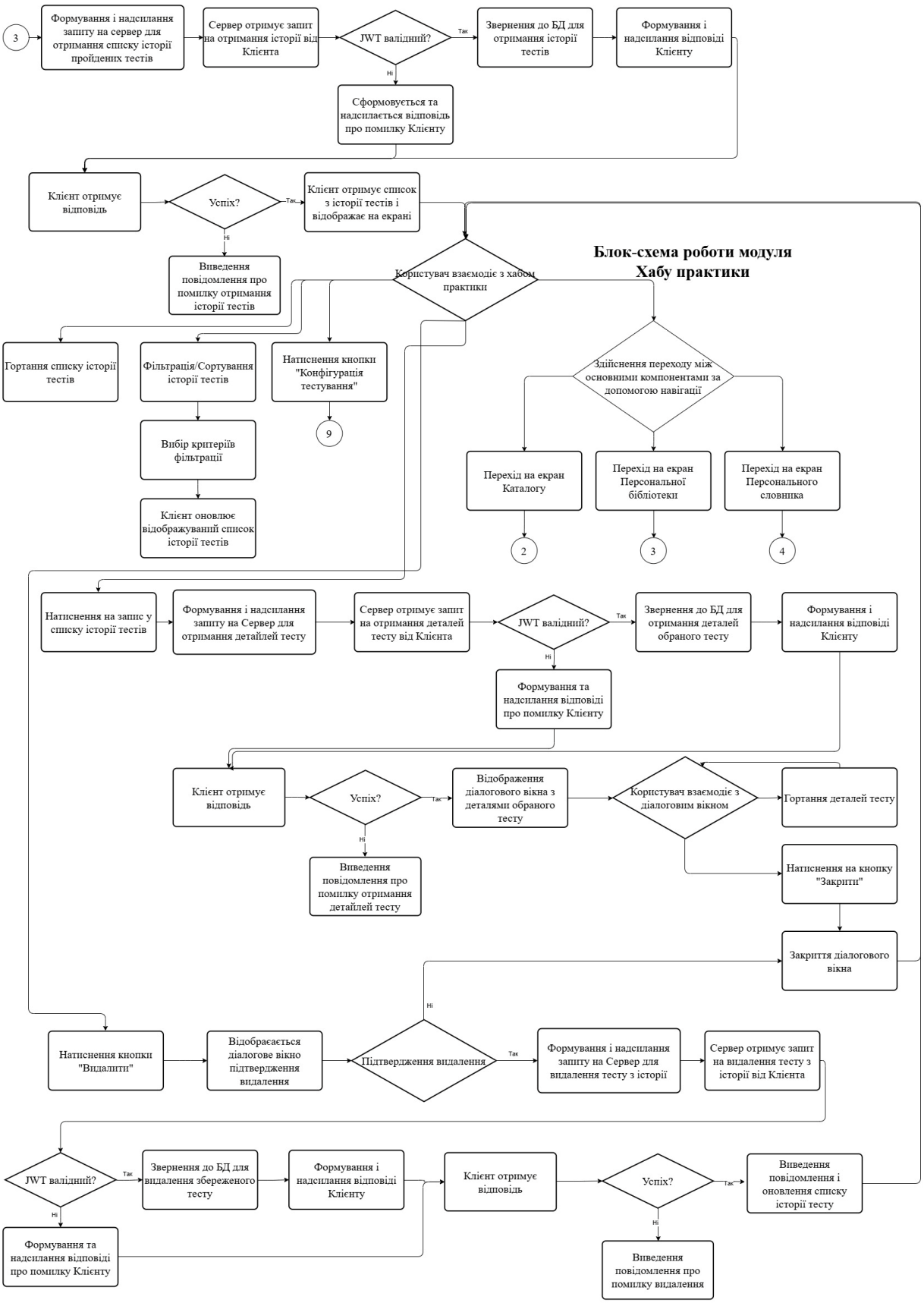




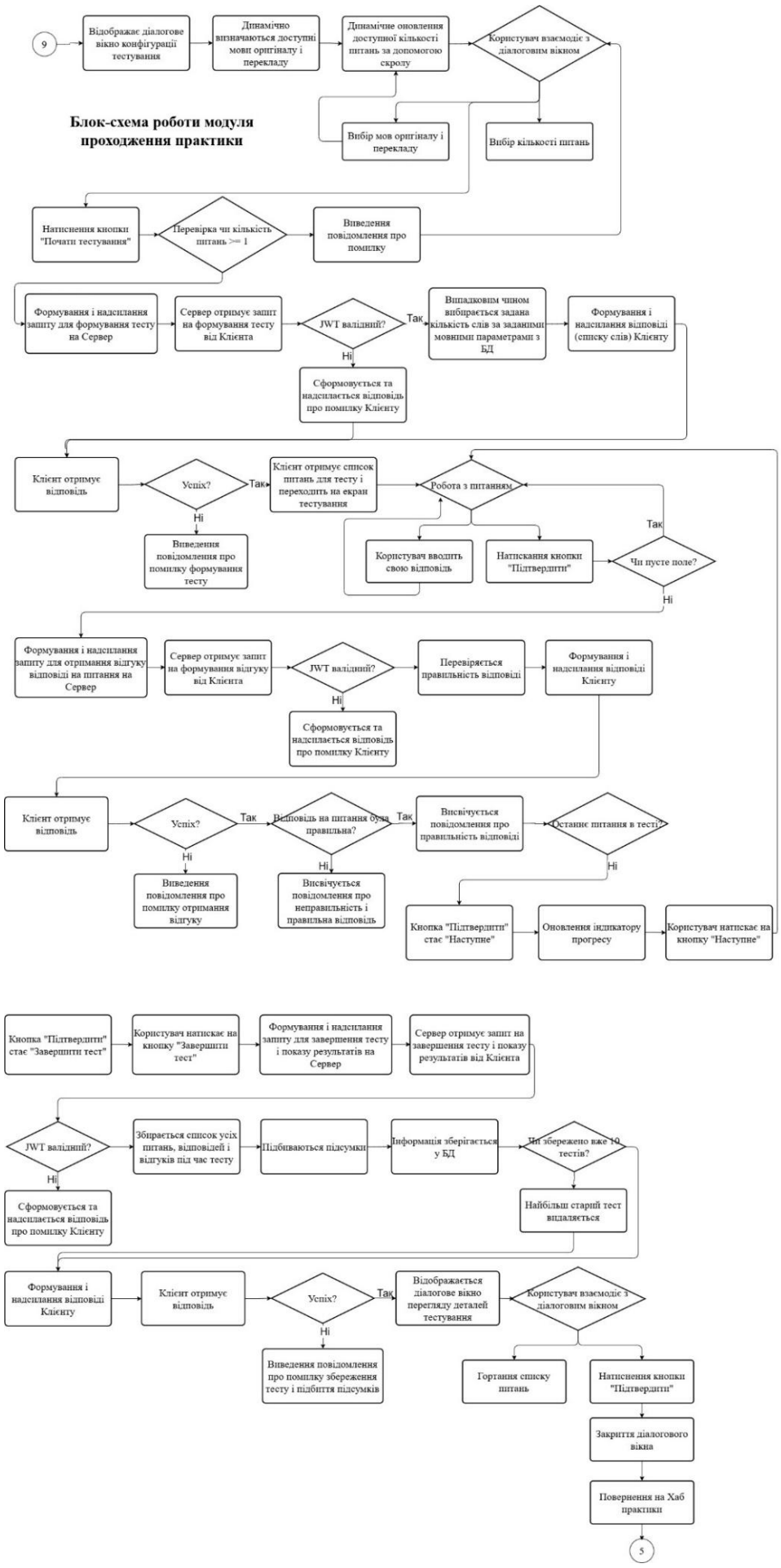
					ІАЛЦ.467200.006 Д3	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		6

Блок-схема роботи кнопок модуля словника




Блок-схема роботи модуля Хабу практики
3
Формування і надсилання запиту на сервер для отримання списку історії пройдених тестів
Сервер отримує запит на отримання історії від Клієнта
JWT валідний?
Так
Ні
Звернення до БД для отримання історії тестів
Формування і надсилання відповіді Клієнту
Сформовується та надсилається відповідь про помилку Клієнту
Клієнт отримує відповідь
Успіх?
Так
Ні
Клієнт отримує список з історії тестів і відображає на екрані
Виведення повідомлення про помилку отримання історії тестів
Користувач взаємодіє з хабом практики
Гортання списку історії тестів
Фільтрація/Сортування історії тестів
Натиснення кнопки "Конфігурація тестування"
9
Вибір критеріїв фільтрації
Клієнт оновлює відображуваний список історії тестів
Здійснення переходу між основними компонентами за допомогою навігації
Перехід на екран Каталогу
Перехід на екран Персональної бібліотеки
Перехід на екран Персонального словника
2
3
4
Натиснення на запис у списку історії тестів
Формування і надсилання запиту на Сервер для отримання детайлей тесту
Сервер отримує запит на отримання деталей тесту від Клієнта
JWT валідний?
Так
Ні
Звернення до БД для отримання деталей обраного тесту
Формування і надсилання відповіді Клієнту
Формування та надсилання відповіді про помилку Клієнту
Клієнт отримує відповідь
Успіх?
Так
Ні
Відображення діалогового вікна з деталями обраного тесту
Виведення повідомлення про помилку отримання детайлей тесту
Користувач взаємодіє з діалоговим вікном
Гортання деталей тесту
Натиснення на кнопку "Закрити"
Закриття діалогового вікна
Натиснення кнопки "Видалити"
Відобраєається діалогове вікно підтвердження видалення
Підтвердження видалення
Так
Ні
Формування і надсилання запиту на Сервер для видалення тесту з історії
Сервер отримує запит на видалення тесту з історії від Клієнта
JWT валідний?
Так
Ні
Звернення до БД для видалення збереженого тесту
Формування і надсилання відповіді Клієнту
Клієнт отримує відповідь
Успіх?
Так
Ні
Виведення повідомлення і оновлення списку історії тесту
Формування та надсилання відповіді про помилку Клієнту
Виведення повідомлення про помилку видалення

Блок-схема роботи модуля проходження практики



ДОДАТОК 4

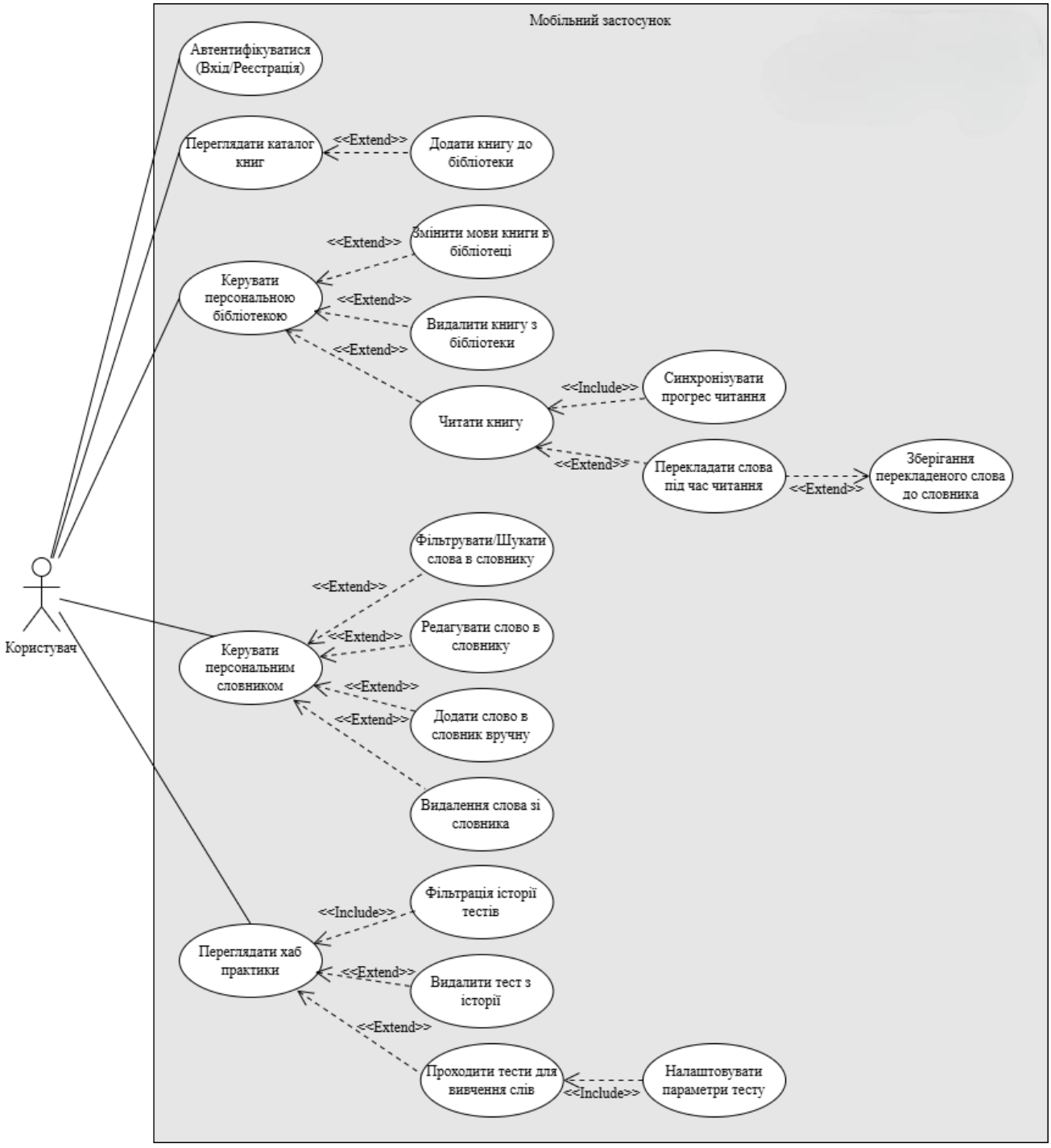
Мобільний застосунок для мультилінгвального читання книжок

Юзкейс-діаграма

ІАЛЦ.467200.004 Д4

Аркушів 1

Київ 2025 р.



					ІАЛЦ.467200.007 Д4		
Зм.	Лист.	№ докум.	Підпис	Дата			
Розробив		Федосєєва О.Д.			**Мобільний застосунок для мультилінгвального читання книжок**	Літ.	Арк. 1 / Акрушів 1
Перевірив		Васильєва М.Д.					
Н. Контр.					Юзкейс діаграма	КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11	
Затвердив							

ДОДАТОК 5

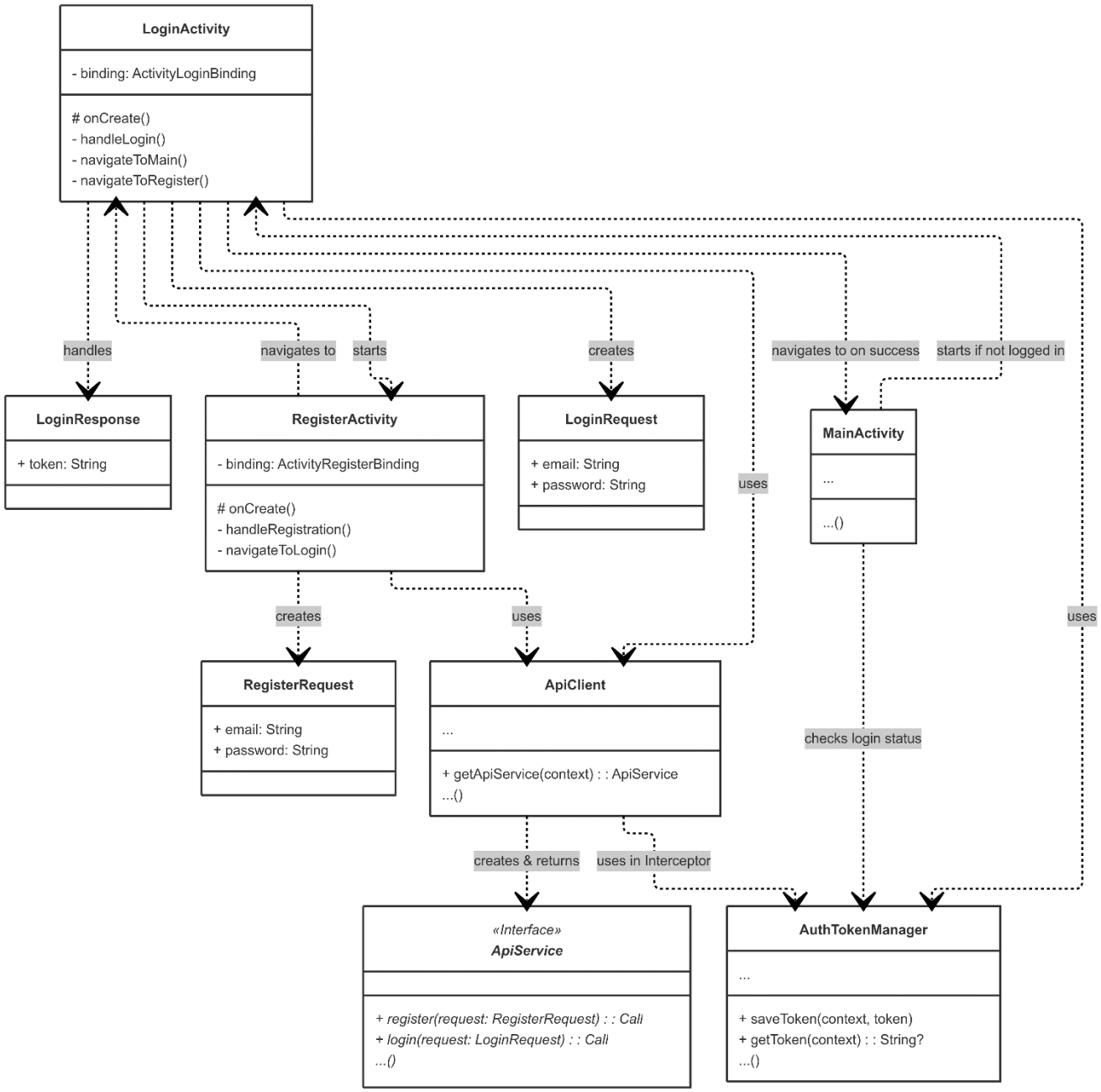
Мобільний застосунок для мультилінгвального читання книжок

Діграми класів

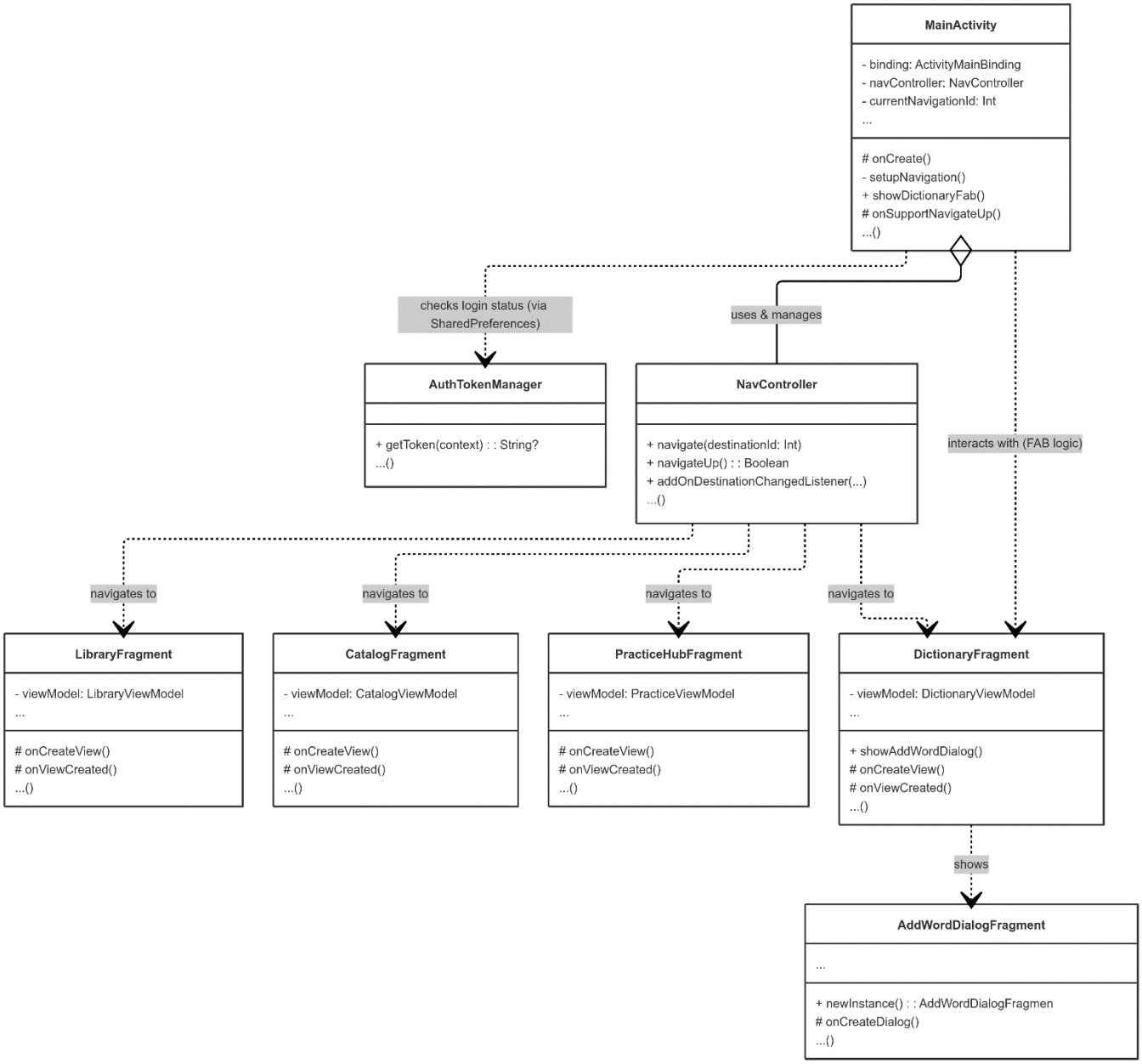
ІАЛЦ.467200.004 Д5

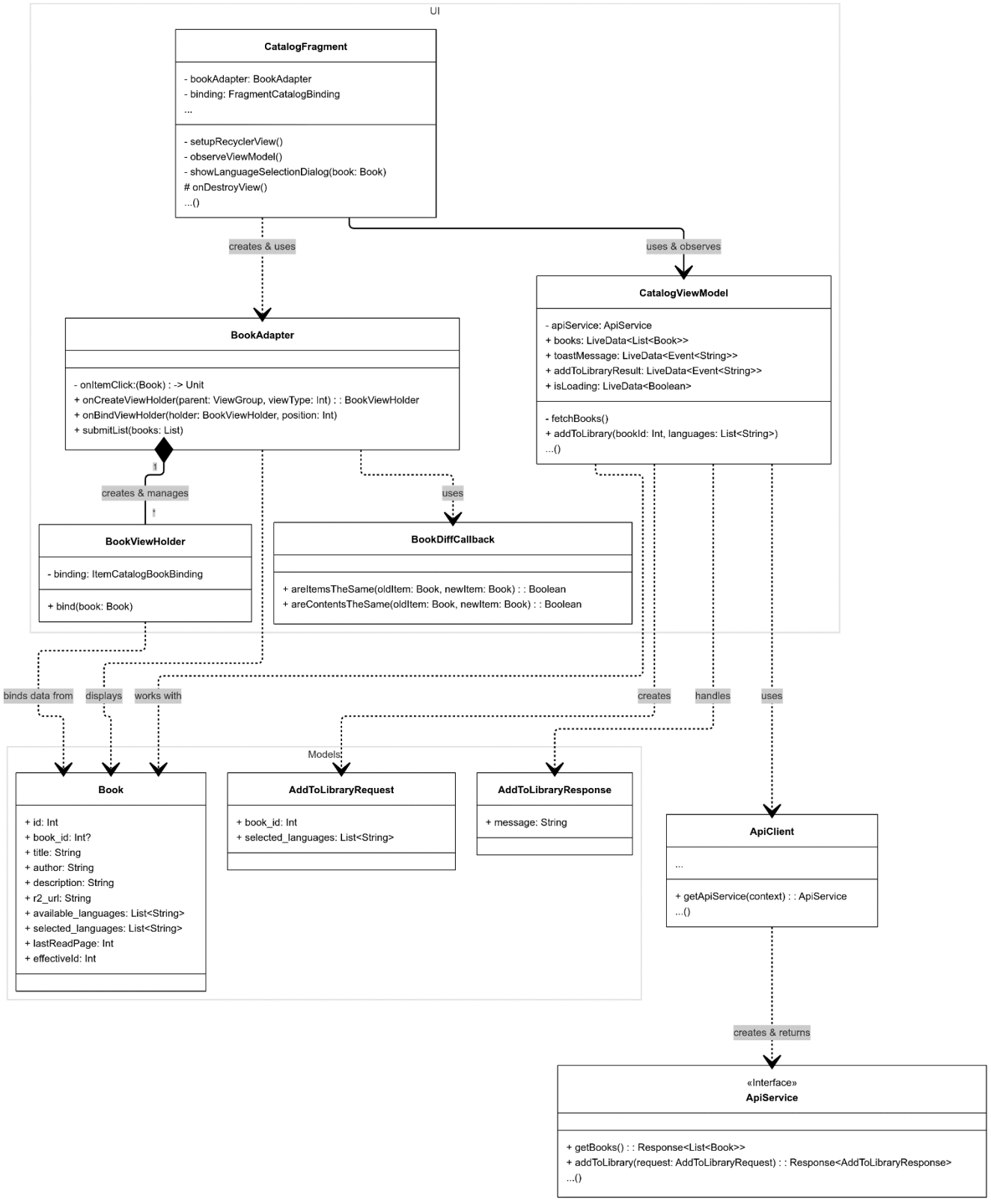
Аркушів 9

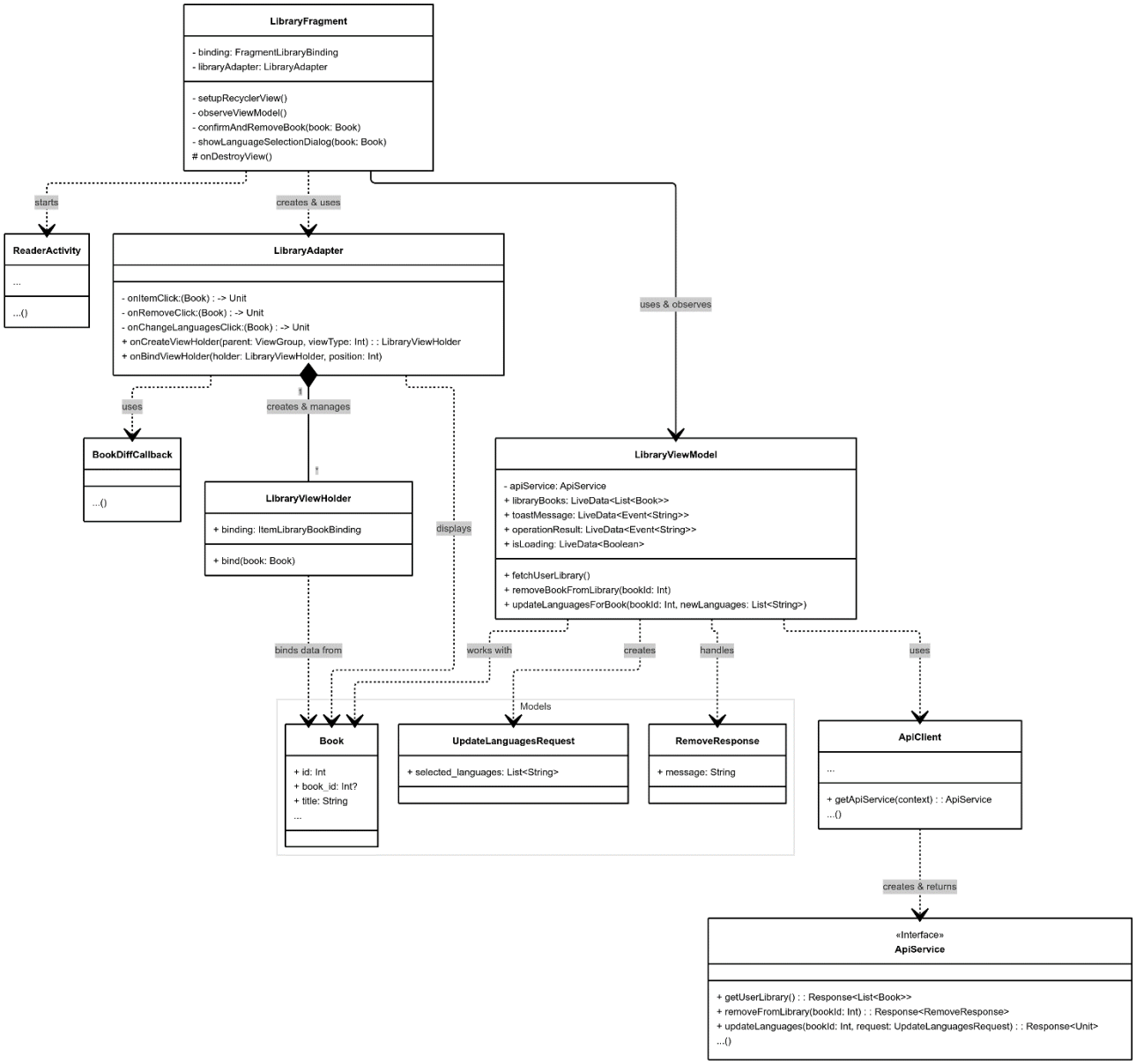
Київ 2025 р.

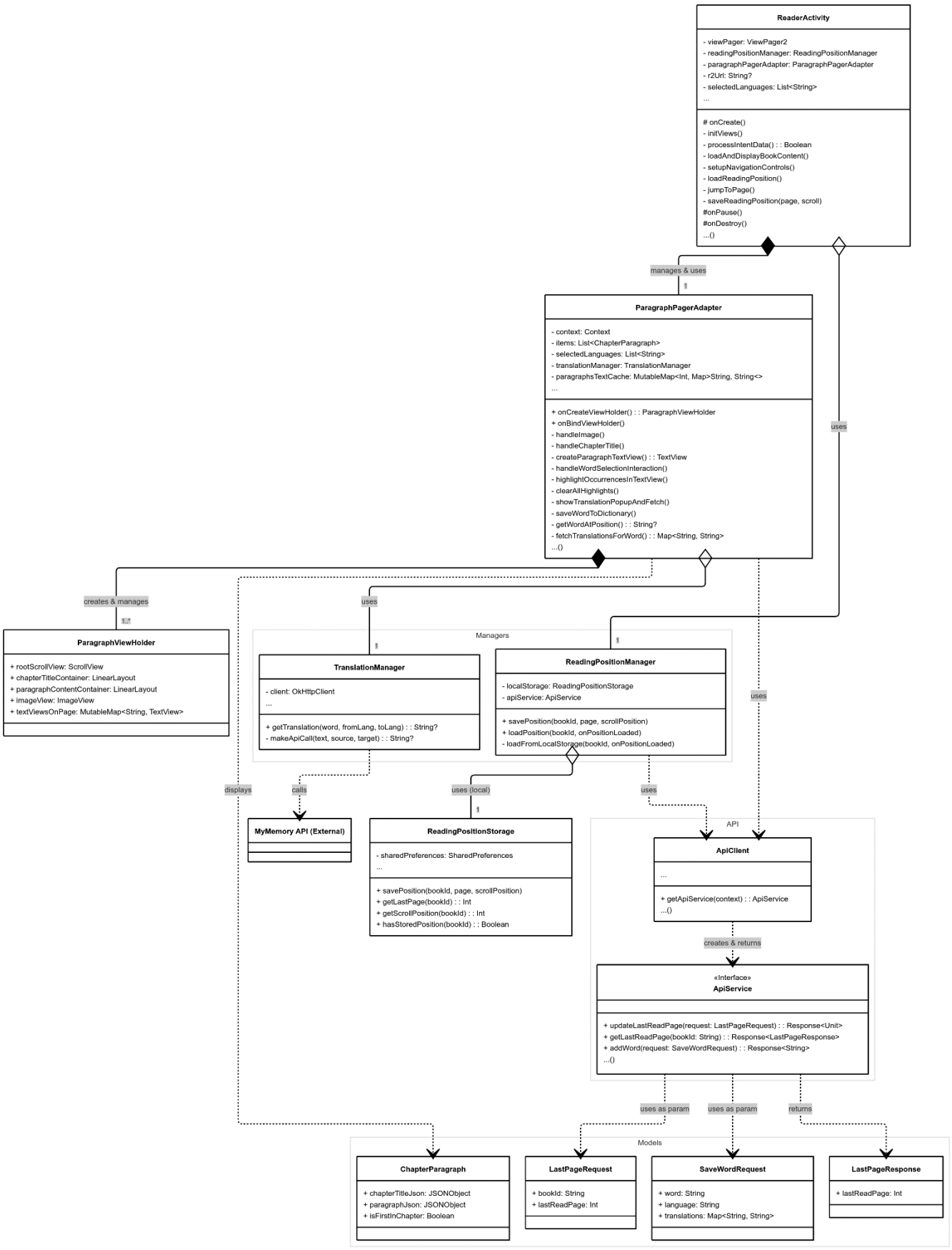


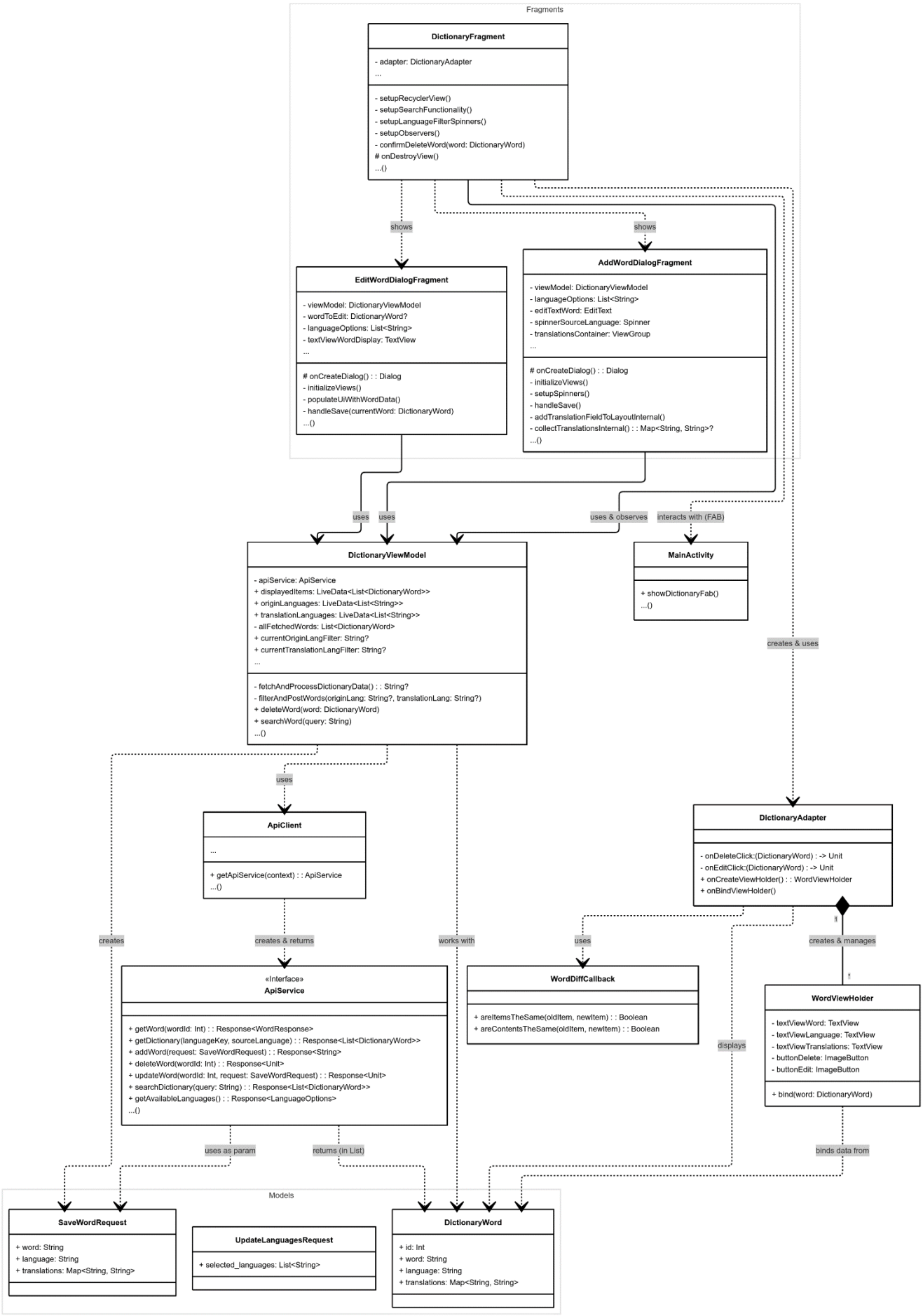
					ІАЛЦ.467200.008 Д5	
Зм.	Лист.	№ докум.	Підпис	Дата		
Розробив		Федосєєва О.Д.			**Мобільний застосунок для мультилінгвального читання книжок**	Літ. / Арк. / Акрушів: 1 / 9
Перевірив		Васильєва М.Д.				КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11
Н. Контр.						
Затвердив					Діаграми класів	

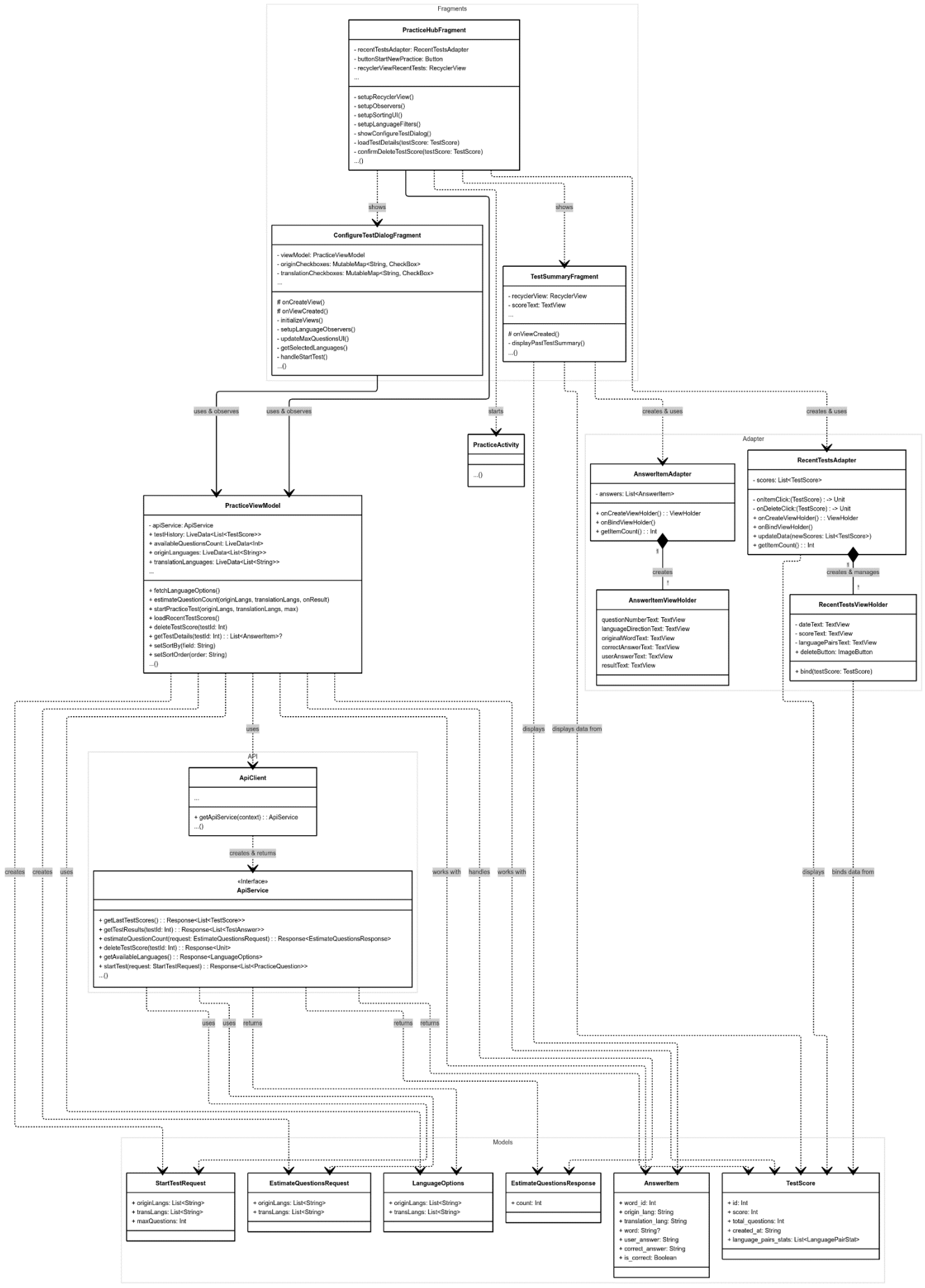

MainActivity
- binding: ActivityMainBinding
- navController: NavController
- currentNavigationId: Int
...
onCreate()
- setupNavigation()
+ showDictionaryFab()
# onSupportNavigateUp()
...()
checks login status (via SharedPreferences)
uses & manages
interacts with (FAB logic)
AuthTokenManager
+ getToken(context) : : String?
...()
NavController
+ navigate(destinationId: Int)
+ navigateUp() : : Boolean
+ addOnDestinationChangedListener(...)
...()
navigates to
navigates to
navigates to
navigates to
LibraryFragment
- viewModel: LibraryViewModel
...
# onCreateView()
# onViewCreated()
...()
CatalogFragment
- viewModel: CatalogViewModel
...
# onCreateView()
# onViewCreated()
...()
PracticeHubFragment
- viewModel: PracticeViewModel
...
# onCreateView()
# onViewCreated()
...()
DictionaryFragment
- viewModel: DictionaryViewModel
...
+ showAddWordDialog()
# onCreateView()
# onViewCreated()
...()
shows
AddWordDialogFragment
...
+ newInstance() : : AddWordDialogFragmen
# onCreateDialog()
...()

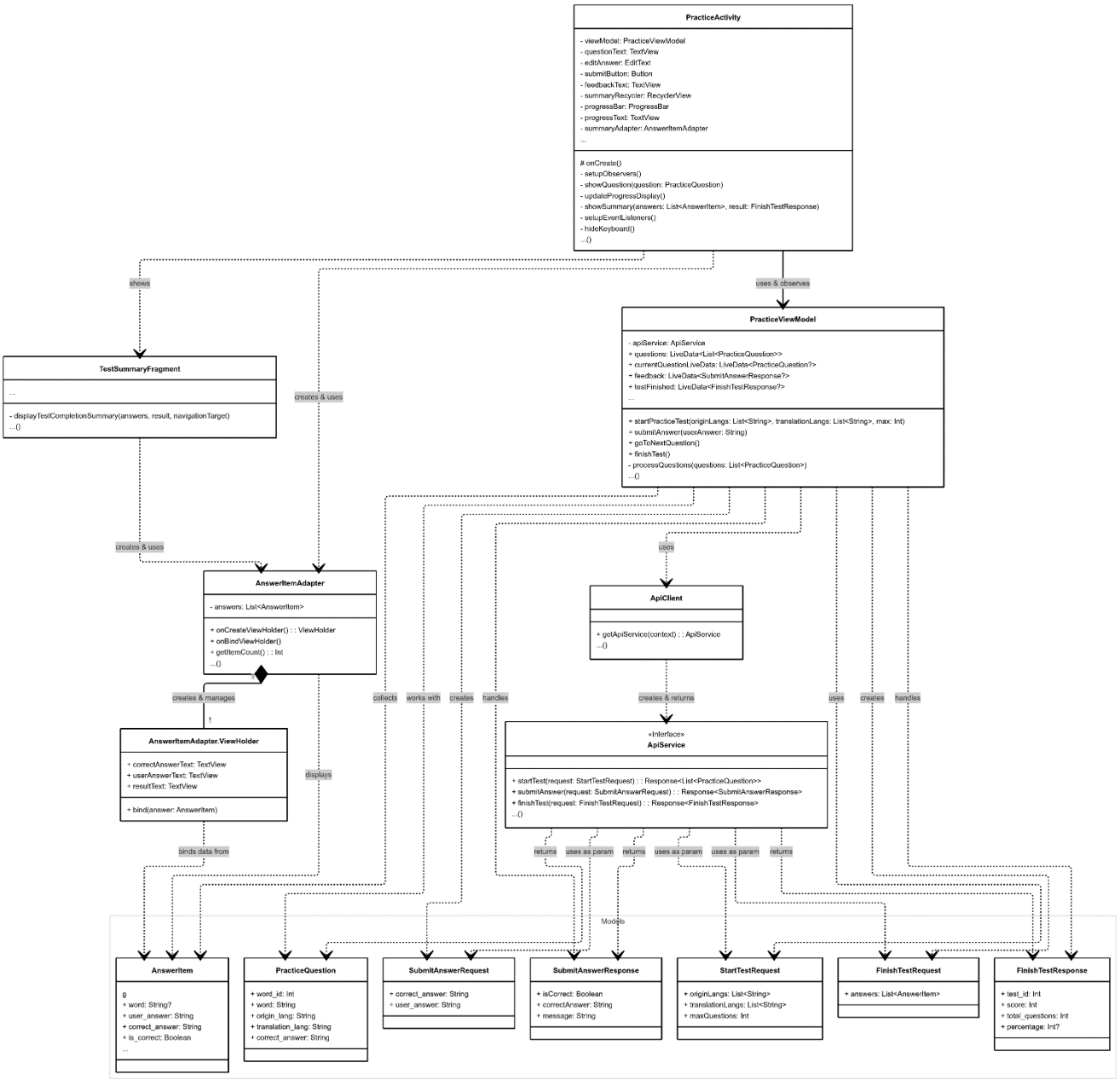

UI
CatalogFragment
- bookAdapter: BookAdapter
- binding: FragmentCatalogBinding
...
- setupRecyclerView()
- observeViewModel()
- showLanguageSelectionDialog(book: Book)
# onDestroyView()
...()
creates & uses
uses & observes
CatalogViewModel
- apiService: ApiService
+ books: LiveData<List<Book>>
+ toastMessage: LiveData<Event<String>>
+ addToLibraryResult: LiveData<Event<String>>
+ isLoading: LiveData<Boolean>
- fetchBooks()
+ addToLibrary(bookId: Int, languages: List<String>)
...()
BookAdapter
- onItemClick:(Book) : -> Unit
+ onCreateViewHolder(parent: ViewGroup, viewType: Int) : : BookViewHolder
+ onBindViewHolder(holder: BookViewHolder, position: Int)
+ submitList(books: List)
1
creates & manages
*
uses
BookViewHolder
- binding: ItemCatalogBookBinding
+ bind(book: Book)
BookDiffCallback
+ areItemsTheSame(oldItem: Book, newItem: Book) : : Boolean
+ areContentsTheSame(oldItem: Book, newItem: Book) : : Boolean
binds data from
displays
works with
creates
handles
uses
Models
Book
+ id: Int
+ book_id: Int?
+ title: String
+ author: String
+ description: String
+ r2_url: String
+ available_languages: List<String>
+ selected_languages: List<String>
+ lastReadPage: Int
+ effectiveId: Int
AddToLibraryRequest
+ book_id: Int
+ selected_languages: List<String>
AddToLibraryResponse
+ message: String
ApiClient
...
+ getApiService(context) : : ApiService
...()
creates & returns
«Interface»
ApiService
+ getBooks() : : Response<List<Book>>
+ addToLibrary(request: AddToLibraryRequest) : : Response<AddToLibraryResponse>
...()

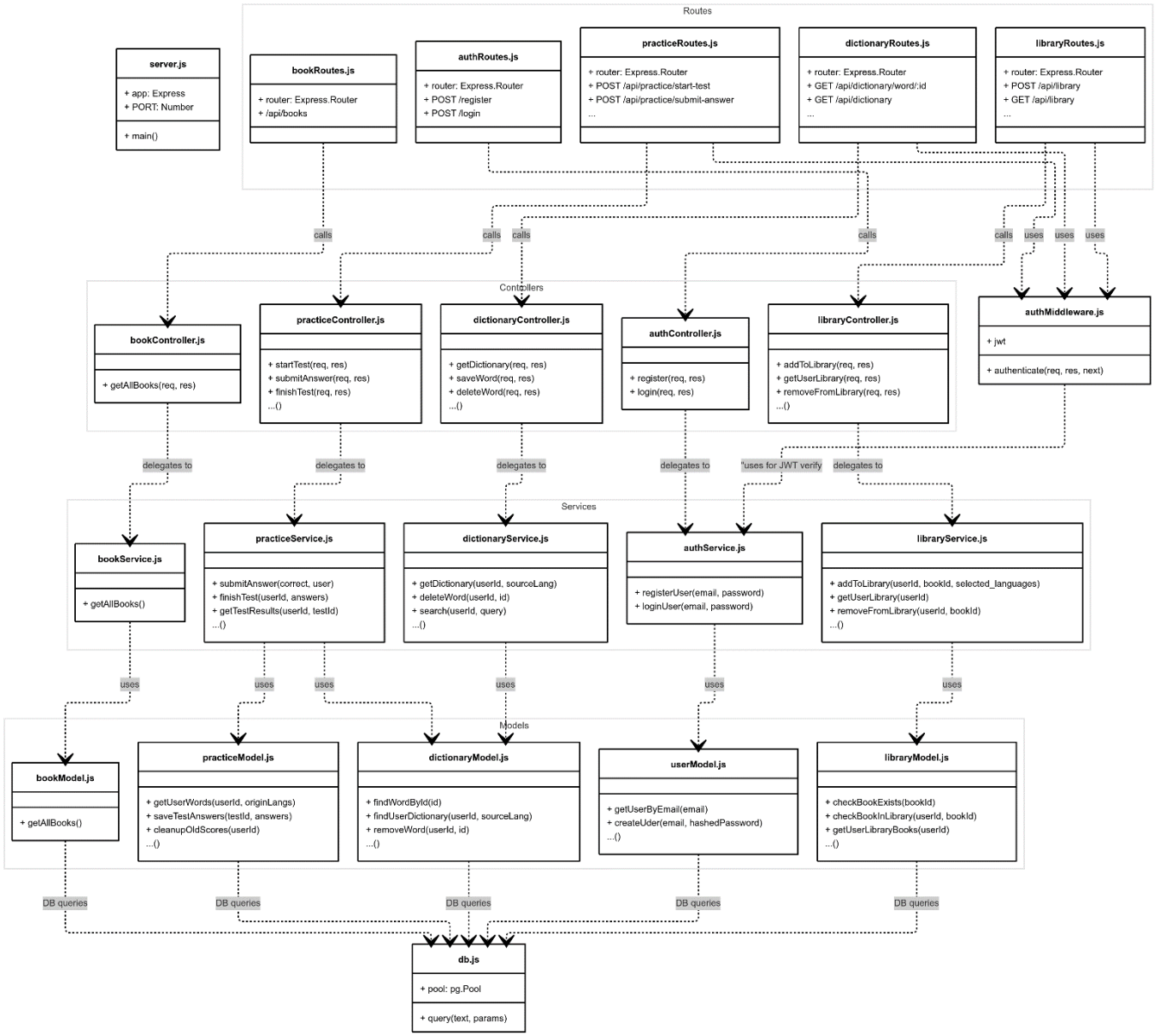

LibraryFragment
- binding: FragmentLibraryBinding
- libraryAdapter: LibraryAdapter
- setupRecyclerView()
- observeViewModel()
- confirmAndRemoveBook(book: Book)
- showLanguageSelectionDialog(book: Book)
onDestroyView()
starts
creates & uses
uses & observes
ReaderActivity
...
...()
LibraryAdapter
- onItemClick:(Book) : -> Unit
- onRemoveClick:(Book) : -> Unit
- onChangeLanguagesClick:(Book) : -> Unit
+ onCreateViewHolder(parent: ViewGroup, viewType: Int) : : LibraryViewHolder
+ onBindViewHolder(holder: LibraryViewHolder, position: Int)
uses
creates & manages
displays
BookDiffCallback
...()
LibraryVIewHolder
+ binding: ItemLibraryBookBinding
+ bind(book: Book)
LibraryViewModel
- apiService: ApiService
+ libraryBooks: LiveData<List<Book>>
+ toastMessage: LiveData<Event<String>>
+ operationResult: LiveData<Event<String>>
+ isLoading: LiveData<Boolean>
+ fetchUserLibrary()
+ removeBookFromLibrary(bookId: Int)
+ updateLanguagesForBook(bookId: Int, newLanguages: List<String>)
binds data from
works with
creates
handles
uses
Models
Book
+ id: Int
+ book_id: Int?
+ title: String
...
UpdateLanguagesRequest
+ selected_languages: List<String>
RemoveResponse
+ message: String
ApiClient
...
+ getApiService(context) : : ApiService
...()
creates & returns
«Interface»
ApiService
+ getUserLibrary() : : Response<List<Book>>
+ removeFromLibrary(bookId: Int) : : Response<RemoveResponse>
+ updateLanguages(bookId: Int, request: UpdateLanguagesRequest) : : Response<Unit>
...()


ReaderActivity
- viewPager: ViewPager2
- readingPositionManager: ReadingPositionManager
- paragraphPagerAdapter: ParagraphPagerAdapter
- r2Url: String?
- selectedLanguages: List<String>
...
onCreate()
- initViews()
- processIntentData() : : Boolean
- loadAndDisplayBookContent()
- setupNavigationControls()
- loadReadingPosition()
- jumpToPage()
- saveReadingPosition(page, scroll)
#onPause()
#onDestroy()
...()
manages & uses
1
ParagraphPagerAdapter
- context: Context
- items: List<ChapterParagraph>
- selectedLanguages: List<String>
- translationManager: TranslationManager
- paragraphsTextCache: MutableMap<Int, Map>String, String<>
...
+ onCreateViewHolder() : : ParagraphViewHolder
+ onBindViewHolder()
- handleImage()
- handleChapterTitle()
- createParagraphTextView() : : TextView
- handleWordSelectionInteraction()
- highlightOccurrencesInTextView()
- clearAllHighlights()
- showTranslationPopupAndFetch()
- saveWordToDictionary()
- getWordAtPosition() : : String?
- fetchTranslationsForWord() : : Map<String, String>
...()
uses
creates & manages
1..*
ParagraphViewHolder
+ rootScrollView: ScrollView
+ chapterTitleContainer: LinearLayout
+ paragraphContentContainer: LinearLayout
+ imageView: ImageView
+ textViewsOnPage: MutableMap<String, TextView>
uses
Managers
1
TranslationManager
- client: OkHttpClient
...
+ getTranslation(word, fromLang, toLang) : : String?
- makeApiCall(text, source, target) : : String?
1
ReadingPositionManager
- localStorage: ReadingPositionStorage
- apiService: ApiService
+ savePosition(bookId, page, scrollPosition)
+ loadPosition(bookId, onPositionLoaded)
- loadFromLocalStorage(bookId, onPositionLoaded)
uses
displays
calls
uses (local)
uses
1
MyMemory API (External)
ReadingPositionStorage
- sharedPreferences: SharedPreferences
...
+ savePosition(bookId, page, scrollPosition)
+ getLastPage(bookId) : : Int
+ getScrollPosition(bookId) : : Int
+ hasStoredPosition(bookId) : : Boolean
API
ApiClient
...
+ getApiService(context) : : ApiService
...()
creates & returns
«Interface»
ApiService
+ updateLastReadPage(request: LastPageRequest) : : Response<Unit>
+ getLastReadPage(bookId: String) : : Response<LastPageResponse>
+ addWord(request: SaveWordRequest) : : Response<String>
...()
uses as param
uses as param
returns
Models
ChapterParagraph
+ chapterTitleJson: JSONObject
+ paragraphJson: JSONObject
+ isFirstInChapter: Boolean
LastPageRequest
+ bookId: String
+ lastReadPage: Int
SaveWordRequest
+ word: String
+ language: String
+ translations: Map<String, String>
LastPageResponse
+ lastReadPage: Int


Fragments
DictionaryFragment
- adapter: DictionaryAdapter
...
- setupRecyclerView()
- setupSearchFunctionality()
- setupLanguageFilterSpinners()
- setupObservers()
- confirmDeleteWord(word: DictionaryWord)
# onDestroyView()
...()
shows
shows
EditWordDialogFragment
- viewModel: DictionaryViewModel
- wordToEdit: DictionaryWord?
- languageOptions: List<String>
- textViewWordDisplay: TextView
...
onCreateDialog() : : Dialog
- initializeViews()
- populateUiWithWordData()
- handleSave(currentWord: DictionaryWord)
...()
AddWordDialogFragment
- viewModel: DictionaryViewModel
- languageOptions: List<String>
- editTextWord: EditText
- spinnerSourceLanguage: Spinner
- translationsContainer: ViewGroup
...
onCreateDialog() : : Dialog
- initializeViews()
- setupSpinners()
- handleSave()
- addTranslationFieldToLayoutInternal()
- collectTranslationsInternal() : : Map<String, String>?
...()
uses
uses
uses & observes
interacts with (FAB)
DictionaryViewModel
- apiService: ApiService
+ displayedItems: LiveData<List<DictionaryWord>>
+ originLanguages: LiveData<List<String>>
+ translationLanguages: LiveData<List<String>>
- allFetchedWords: List<DictionaryWord>
+ currentOriginLangFilter: String?
+ currentTranslationLangFilter: String?
...
- fetchAndProcessDictionaryData() : : String?
- filterAndPostWords(originLang: String?, translationLang: String?)
+ deleteWord(word: DictionaryWord)
+ searchWord(query: String)
...()
MainActivity
+ showDictionaryFab()
...()
creates & uses
uses
ApiClient
...
+ getApiService(context) : : ApiService
...()
DictionaryAdapter
- onDeleteClick:(DictionaryWord) : -> Unit
- onEditClick:(DictionaryWord) : -> Unit
+ onCreateViewHolder() : : WordViewHolder
+ onBindViewHolder()
creates
creates & returns
works with
uses
displays
creates & manages
1
*
«Interface»
ApiService
+ getWord(wordId: Int) : : Response<WordResponse>
+ getDictionary(languageKey, sourceLanguage) : : Response<List<DictionaryWord>>
+ addWord(request: SaveWordRequest) : : Response<String>
+ deleteWord(wordId: Int) : : Response<Unit>
+ updateWord(wordId: Int, request: SaveWordRequest) : : Response<Unit>
+ searchDictionary(query: String) : : Response<List<DictionaryWord>>
+ getAvailableLanguages() : : Response<LanguageOptions>
...()
WordDiffCallback
+ areItemsTheSame(oldItem, newItem) : : Boolean
+ areContentsTheSame(oldItem, newItem) : : Boolean
WordViewHolder
- textViewWord: TextView
- textViewLanguage: TextView
- textViewTranslations: TextView
- buttonDelete: ImageButton
- buttonEdit: ImageButton
+ bind(word: DictionaryWord)
uses as param
returns (in List)
binds data from
Models
SaveWordRequest
+ word: String
+ language: String
+ translations: Map<String, String>
UpdateLanguagesRequest
+ selected_languages: List<String>
DictionaryWord
+ id: Int
+ word: String
+ language: String
+ translations: Map<String, String>


Fragments
PracticeHubFragment
- recentTestsAdapter: RecentTestsAdapter
- buttonStartNewPractice: Button
- recyclerViewRecentTests: RecyclerView
...
- setupRecyclerView()
- setupObservers()
- setupSortingUI()
- setupLanguageFilters()
- showConfigureTestDialog()
- loadTestDetails(testScore: TestScore)
- confirmDeleteTestScore(testScore: TestScore)
...()
shows
shows
ConfigureTestDialogFragment
- viewModel: PracticeViewModel
- originCheckboxes: MutableMap<String, CheckBox>
- translationCheckboxes: MutableMap<String, CheckBox>
...
onCreateView()
onViewCreated()
- initializeViews()
- setupLanguageObservers()
- updateMaxQuestionsUI()
- getSelectedLanguages()
- handleStartTest()
...()
TestSummaryFragment
- recyclerView: RecyclerView
- scoreText: TextView
...
onViewCreated()
- displayPastTestSummary()
...()
uses & observes
uses & observes
starts
creates & uses
creates & uses
PracticeActivity
...()
Adapter
AnswerItemAdapter
- answers: List<AnswerItem>
+ onCreateViewHolder() : : ViewHolder
+ onBindViewHolder()
+ getItemCount() : : Int
RecentTestsAdapter
- scores: List<TestScore>
- onItemClick:(TestScore) : -> Unit
- onDeleteClick:(TestScore) : -> Unit
+ onCreateViewHolder() : : ViewHolder
+ onBindViewHolder()
+ updateData(newScores: List<TestScore>)
+ getItemCount() : : Int
creates
creates & manages
AnswerItemViewHolder
questionNumberText: TextView
languageDirectionText: TextView
originalWordText: TextView
correctAnswerText: TextView
userAnswerText: TextView
resultText: TextView
RecentTestsViewHolder
- dateText: TextView
- scoreText: TextView
- languagePairsText: TextView
+ deleteButton: ImageButton
+ bind(testScore: TestScore)
PracticeViewModel
- apiService: ApiService
+ testHistory: LiveData<List<TestScore>>
+ availableQuestionsCount: LiveData<Int>
+ originLanguages: LiveData<List<String>>
+ translationLanguages: LiveData<List<String>>
...
+ fetchLanguageOptions()
+ estimateQuestionCount(originLangs, translationLangs, onResult)
+ startPracticeTest(originLangs, translationLangs, max)
+ loadRecentTestScores()
+ deleteTestScore(testId: Int)
+ getTestDetails(testId: Int) : : List<AnswerItem>?
+ setSortBy(field: String)
+ setSortOrder(order: String)
...()
uses
displays
displays data from
API
ApiClient
...
+ getApiService(context) : : ApiService
...()
creates & returns
creates
creates
uses
works with
handles
works with
displays
binds data from
«Interface»
ApiService
+ getLastTestScores() : : Response<List<TestScore>>
+ getTestResults(testId: Int) : : Response<List<TestAnswer>>
+ estimateQuestionCount(request: EstimateQuestionsRequest) : : Response<EstimateQuestionsResponse>
+ deleteTestScore(testId: Int) : : Response<Unit>
+ getAvailableLanguages() : : Response<LanguageOptions>
+ startTest(request: StartTestRequest) : : Response<List<PracticeQuestion>>
...()
uses
uses
returns
returns
returns
Models
StartTestRequest
+ originLangs: List<String>
+ transLangs: List<String>
+ maxQuestions: Int
EstimateQuestionsRequest
+ originLangs: List<String>
+ transLangs: List<String>
LanguageOptions
+ originLangs: List<String>
+ transLangs: List<String>
EstimateQuestionsResponse
+ count: Int
AnswerItem
+ word_id: Int
+ origin_lang: String
+ translation_lang: String
+ word: String?
+ user_answer: String
+ correct_answer: String
+ is_correct: Boolean
TestScore
+ id: Int
+ score: Int
+ total_questions: Int
+ created_at: String
+ language_pairs_stats: List<LanguagePairStat>


PracticeActivity
- viewModel: PracticeViewModel
- questionText: TextView
- editAnswer: EditText
- submitButton: Button
- feedbackText: TextView
- summaryRecycler: RecyclerView
- progressBar: ProgressBar
- progressText: TextView
- summaryAdapter: AnswerItemAdapter
...
onCreate()
- setupObservers()
- showQuestion(question: PracticeQuestion)
- updateProgressDisplay()
- showSummary(answers: List<AnswerItem>, result: FinishTestResponse)
- setupEventListeners()
- hideKeyboard()
...()
shows
uses & observes
TestSummaryFragment
...
- displayTestCompletionSummary(answers, result, navigationTarget)
...()
creates & uses
PracticeViewModel
- apiService: ApiService
+ questions: LiveData<List<PracticeQuestion>>
+ currentQuestionLiveData: LiveData<PracticeQuestion?>
+ feedback: LiveData<SubmitAnswerResponse?>
+ testFinished: LiveData<FinishTestResponse?>
...
+ startPracticeTest(originLangs: List<String>, translationLangs: List<String>, max: Int)
+ submitAnswer(userAnswer: String)
+ goToNextQuestion()
+ finishTest()
- processQuestions(questions: List<PracticeQuestion>)
...()
creates & uses
uses
AnswerItemAdapter
- answers: List<AnswerItem>
+ onCreateViewHolder() : : ViewHolder
+ onBindViewHolder()
+ getItemCount() : : Int
...()
ApiClient
+ getApiService(context) : : ApiService
...()
creates & manages
collects
works with
creates
handles
creates & returns
uses
creates
handles
1
AnswerItemAdapter.ViewHolder
+ correctAnswerText: TextView
+ userAnswerText: TextView
+ resultText: TextView
+ bind(answer: AnswerItem)
displays
«Interface»
ApiService
+ startTest(request: StartTestRequest) : : Response<List<PracticeQuestion>>
+ submitAnswer(request: SubmitAnswerRequest) : : Response<SubmitAnswerResponse>
+ finishTest(request: FinishTestRequest) : : Response<FinishTestResponse>
...()
binds data from
returns
uses as param
returns
uses as param
uses as param
returns
Models
AnswerItem
g
+ word: String?
+ user_answer: String
+ correct_answer: String
+ is_correct: Boolean
...
PracticeQuestion
+ word_id: Int
+ word: String
+ origin_lang: String
+ translation_lang: String
+ correct_answer: String
SubmitAnswerRequest
+ correct_answer: String
+ user_answer: String
SubmitAnswerResponse
+ isCorrect: Boolean
+ correctAnswer: String
+ message: String
StartTestRequest
+ originLangs: List<String>
+ translationLangs: List<String>
+ maxQuestions: Int
FinishTestRequest
+ answers: List<AnswerItem>
FinishTestResponse
+ test_id: Int
+ score: Int
+ total_questions: Int
+ percentage: Int?


Routes
server.js
+ app: Express
+ PORT: Number
+ main()
bookRoutes.js
+ router: Express.Router
+ /api/books
authRoutes.js
+ router: Express.Router
+ POST /register
+ POST /login
practiceRoutes.js
+ router: Express.Router
+ POST /api/practice/start-test
+ POST /api/practice/submit-answer
...
dictionaryRoutes.js
+ router: Express.Router
+ GET /api/dictionary/word/:id
+ GET /api/dictionary
...
libraryRoutes.js
+ router: Express.Router
+ POST /api/library
+ GET /api/library
...
calls
calls
calls
calls
calls
uses
uses
uses
Controllers
bookController.js
+ getAllBooks(req, res)
practiceController.js
+ startTest(req, res)
+ submitAnswer(req, res)
+ finishTest(req, res)
...()
dictionaryController.js
+ getDictionary(req, res)
+ saveWord(req, res)
+ deleteWord(req, res)
...()
authController.js
+ register(req, res)
+ login(req, res)
libraryController.js
+ addToLibrary(req, res)
+ getUserLibrary(req, res)
+ removeFromLibrary(req, res)
...()
authMiddleware.js
+ jwt
+ authenticate(req, res, next)
delegates to
delegates to
delegates to
delegates to
"uses for JWT verify
delegates to
Services
bookService.js
+ getAllBooks()
practiceService.js
+ submitAnswer(correct, user)
+ finishTest(userId, answers)
+ getTestResults(userId, testId)
...()
dictionaryService.js
+ getDictionary(userId, sourceLang)
+ deleteWord(userId, id)
+ search(userId, query)
...()
authService.js
+ registerUser(email, password)
+ loginUser(email, password)
libraryService.js
+ addToLibrary(userId, bookId, selected_languages)
+ getUserLibrary(userId)
+ removeFromLibrary(userId, bookId)
...()
uses
uses
uses
uses
uses
uses
Models
bookModel.js
+ getAllBooks()
practiceModel.js
+ getUserWords(userId, originLangs)
+ saveTestAnswers(testId, answers)
+ cleanupOldScores(userId)
...()
dictionaryModel.js
+ findWordById(id)
+ findUserDictionary(userId, sourceLang)
+ removeWord(userId, id)
...()
userModel.js
+ getUserByEmail(email)
+ createUder(email, hashedPassword)
...()
libraryModel.js
+ checkBookExists(bookId)
+ checkBookInLibrary(userId, bookId)
+ getUserLibraryBooks(userId)
...()
DB queries
DB queries
DB queries
DB queries
DB queries
db.js
+ pool: pg.Pool
+ query(text, params)

ДОДАТОК 6

Мобільний застосунок для мультилінгвального читання книжок

Повний перелік функціональних вимог

ІАЛЦ.467200.004 Д6

Аркушів 5

Київ 2025 р.

ФВ-БП (функціональні вимоги - бізнес-процеси):

- ФВ-БП-01: Система повинна дозволяти користувачеві реєструватися та/або автентифікуватися у застосунку (наприклад, через email/пароль).
- ФВ-БП-02: Система повинна надавати користувачеві можливість переглядати каталог доступних у застосунку книг.
- ФВ-БП-03: Користувач повинен мати змогу обрати книгу з каталогу застосунку для читання (можливо, додаючи її до своєї персональної "полиці" чи "обраного").
- ФВ-БП-04: Під час читання книги користувач повинен мати змогу обрати дві або більше мови (з тих, що доступні в книзі), які будуть відображатися одночасно.
- ФВ-БП-05: Користувач повинен мати змогу виділити слово або фразу в тексті на будь-якій з обраних мов.
- ФВ-БП-06: Після виділення слова/фрази користувач повинен мати змогу додати його до свого персонального словника, вказавши переклад(и) та, можливо, коментар чи контекст.
- ФВ-БП-07: Користувач повинен мати змогу переглядати свій персональний словник.
- ФВ-БП-08: Система повинна надавати користувачеві можливість ініціювати тестування на основі слів з його персонального словника.
- ФВ-БП-09: Система повинна проводити тестування знань слів зі словника (наприклад, у форматі карток, вибору правильного перекладу, введення перекладу тощо).
- ФВ-БП-10: Після завершення тестування система повинна відображати результати користувачеві.

ФВ-ІК (функціональні вимоги - інтерфейс користувача):

- ФВ-ІК-01: Система повинна мати інтуїтивно зрозумілий екран каталогу книг, що відображає доступні книги (наприклад, обкладинки, назви,автори, доступні мови, короткий опис). Можливо, з функціями пошуку та фільтрації.
- ФВ-ІК-02: Інтерфейс читання повинен забезпечувати чітке та синхронізоване відображення тексту обраними мовами (наприклад, паралельні колонки, розділений екран, спливаючі підказки при наведенні/тапі).
- ФВ-ІК-03: Система повинна надавати зручний механізм вибору мов для паралельного відображення.
- ФВ-ІК-04: Інтерфейс повинен підтримувати стандартні жести для навігації по книзі (гортання сторінок, зміна масштабу).
- ФВ-ІК-05: Система повинна надавати чіткий візуальний відгук при виділенні слова/фрази.
- ФВ-ІК-06: Повинен бути реалізований зручний інтерфейс (наприклад, спливаюче вікно або окремий екран) для додавання/редагування записів у словнику.
- ФВ-ІК-07: Екран словника повинен дозволяти перегляд, пошук та фільтрацію збережених слів/фраз.
- ФВ-ІК-08: Інтерфейс тестування повинен бути зрозумілим та інтерактивним, відповідаючи обраному формату тестування.
- ФВ-ІК-09: Результати тестування мають бути представлені у наочному форматі (наприклад, відсоток правильних відповідей, список помилок).

ФВ-ДН (функціональні вимоги - дані):

- ФВ-ДН-01: Система повинна зберігати інформацію про користувачів (ідентифікатори, облікові дані для входу - у безпечному вигляді).

- ФВ-ДН-02: Система повинна зберігати каталог книг, що надаються застосунком, включаючи текст книг різними мовами, метадані (назва, автор, обкладинка, опис, список доступних мов).
- ФВ-ДН-03: Система повинна зберігати інформацію про прогрес читання для кожної книги користувача (наприклад, остання позиція).
- ФВ-ДН-04: Система повинна зберігати персональний словник користувача, включаючи: оригінальне слово/фразу, його переклад(и), мову оригіналу та перекладу.
- ФВ-ДН-05: Система повинна зберігати налаштування користувача (наприклад, обрані пари мов за замовчуванням, налаштування інтерфейсу читання).
- ФВ-ДН-06: Система повинна зберігати історію та результати тестувань (дата, тип тесту, результат, список слів).
- ФВ-ДН-07: Система повинна забезпечувати цілісність даних (наприклад, зв'язок між словом у словнику та книгою, з якої воно було додано).
- ФВ-ДН-08: Система повинна керувати зберіганням контенту книг на пристрої (наприклад, кешування для офлайн-доступу обраних книг).

ФВ-ФП (функціональні вимоги - функціональна продуктивність):

- ФВ-ФП-01: Система повинна відображати каталог книг (наприклад, перші 20 книг) протягом не більше N секунд (наприклад, 2 секунди).
- ФВ-ФП-02: Система повинна завантажувати та відображати сторінку обраної книги (у двох мовах) протягом не більше M секунд (наприклад, 2-3 секунди) після її вибору.
- ФВ-ФП-03: Операція додавання слова до словника повинна виконуватися практично миттєво (менше 1 секунди) після підтвердження користувачем.
- ФВ-ФП-04: Система повинна генерувати та починати сесію тестування (наприклад, для 20 слів) протягом не більше P секунд (наприклад, 3-5 секунд).

ФВ-БЗ (функціональні вимоги - безпека):

- ФВ-БЗ-01: Система повинна забезпечувати безпечне зберігання паролів користувачів (наприклад, хешування).
- ФВ-БЗ-02: Система повинна забезпечувати автентифікацію користувача перед доступом до його персональних даних (словник, результати тестів, прогрес читання).
- ФВ-БЗ-03: (Якщо каталог завантажується з сервера) Канал зв'язку з сервером для отримання каталогу та контенту книг має бути захищеним (наприклад, HTTPS).

ФВ-ІН (функціональні вимоги - інтеграція):

- ФВ-ІН-01: Система повинна взаємодіяти з серверним API для отримання/оновлення каталогу книг та завантаження їх контенту.
- ФВ-ІН-02: Система повинна взаємодіяти з API сервісів автентифікації (наприклад, Google, Facebook), якщо такий спосіб входу реалізований.
- ФВ-ІН-03: Система може потребувати інтеграції з API сервісів перекладу для автоматичної пропозиції перекладу виділеного слова.

ФВ-ЗА (функіональні вимоги - звітность та аналітика):

- ФВ-ЗА-01: Система повинна відображати користувачеві статистику його словника (наприклад, загальна кількість слів, кількість слів за мовами).
- ФВ-ЗА-02: Система повинна відображати історію проходження тестів із зазначенням результатів.
- ФВ-ЗА-03: Система може надавати візуалізацію прогресу навчання (наприклад, графік зміни результатів тестування з часом).
- ФВ-ЗА-04: (Для адміністратора/власника застосунку) Система (або її серверна частина) може збирати знеособлену статистику використання книг (які книги популярніші, як часто читають тощо).

ФВ-НП (вункціональні вимоги - нормативно-правові):

- ФВ-НП-01: Застосунок повинен мати всі необхідні ліцензії та дозволи на розповсюдження та використання текстів книг, що надаються в каталозі.
- ФВ-НП-02: Якщо система збирає персональні дані, вона повинна відповідати вимогам щодо захисту даних (наприклад, GDPR), надаючи користувачеві контроль над своїми даними та чітку політику конфіденційності.

ДОДАТОК 7

Мобільний застосунок для мультилінгвального читання книжок

Повний перелік нефункціональних вимог

ІАЛЦ.467200.004 Д7

Аркушів 4

Київ 2025 р.

НФВ-ЗВ (нефункціональні вимоги - зручність використання):

- НФВ-ЗВ-01: Інтерфейс застосунку повинен бути інтуїтивно зрозумілим, дозволяючи новому користувачеві розпочати читання книги та додати перше слово до словника протягом не більше 1 хвилини без інструкцій.
- НФВ-ЗВ-02: Навігація між основними розділами (Каталог/Бібліотека, Читалка, Словник, Тестування, Налаштування) повинна бути простою та послідовною.
- НФВ-ЗВ-03: Процес виділення слова/фрази та додавання його до словника має бути швидким та потребувати мінімальної кількості дій (наприклад, виділення -> тап на іконку "додати").
- НФВ-ЗВ-04: Налаштування читання (розмір шрифту, вибір мов для відображення) мають бути легко доступні з екрану читалки.
- НФВ-ЗВ-05: Застосунок повинен відповідати базовим принципам доступності (Accessibility Guidelines) для мобільних платформ (iOS/Android), забезпечуючи читабельність тексту та достатній контраст елементів.

НФВ-ПД (нефункціональні вимоги - продуктивність):

- НФВ-ПД-01: Час запуску застосунку (від тапу на іконку до появи першого інтерактивного екрану) не повинен перевищувати 3 секунди на цільових пристроях.
- НФВ-ПД-02: Час відкриття книги з каталогу/бібліотеки та відображення першої сторінки у двох мовах не повинен перевищувати 2 секунди.
- НФВ-ПД-03: Гортання сторінок у читалці має бути плавним, без помітних затримок чи "завмирань".

- НФВ-ПД-04: Відгук на виділення слова/фрази та поява опції додавання до словника має бути миттєвим (менше 0.5 секунди).
- НФВ-ПД-05: Відображення списку слів у словнику (наприклад, перші 50 слів) не повинно займати більше 1 секунди.
- НФВ-ПД-06: Застосунок не повинен надмірно споживати ресурси пристрою (CPU, пам'ять, батарея) під час читання або перебування у фоновому режимі.

НФВ-НД (нефункціональні вимоги - надійність):

- НФВ-НД-01: Застосунок повинен мати високий рівень стабільності. Частота критичних збоїв (крешів), що призводять до закриття застосунку, не повинна перевищувати 0.5% від усіх сесій користувачів.
- НФВ-НД-02: Збереження даних користувача (прогрес читання, словник, результати тестів) повинно бути надійним. Втрата даних через помилки застосунку є неприпустимою.
- НФВ-НД-03: Застосунок повинен коректно обробляти переривання (наприклад, вхідний дзвінок, перехід у фоновий режим) та відновлювати свій стан при поверненні.

НФВ-ДП (нефункціональні вимоги - доступність):

- НФВ-ДП-01: Основна функціональність читання завантажених/кешованих книг та роботи з локальним словником повинна бути доступною в режимі офлайн (без підключення до Інтернету).
- НФВ-ДП-02: Застосунок повинен коректно обробляти ситуації відсутності мережевого з'єднання при спробі доступу до онлайн-функцій (наприклад, показати відповідне повідомлення, дозволити доступ до офлайн-контенту).

НФВ-БЗ (нефункціональні вимоги - безпека):

- НФВ-БЗ-01: Облікові дані користувачів (паролі) повинні зберігатися у захешованому вигляді.
- НФВ-БЗ-02: Доступ до персональних даних користувача (словник, прогрес) повинен здійснюватися лише після успішної автентифікації.
- НФВ-БЗ-03: Усі комунікації, що містять чутливі дані (облікові дані, персональні дані), повинні відбуватися через захищене з'єднання (HTTPS).
- НФВ-БЗ-04: Застосунок повинен вживати заходів для ускладнення несанкціонованого вилучення контенту книг (наскільки це можливо на клієнтській стороні).

НФВ-СМ (нефункціональні вимоги - сумісність):

- НФВ-СМ-01: Застосунок повинен підтримувати операційну систему Android версії X.X та вище (наприклад, Android 8.0+).
- НФВ-СМ-02: Інтерфейс застосунку повинен коректно адаптуватися до різних розмірів та роздільних здатностей екранів поширених моделей смартфонів та планшетів.

НФВ-СП (нефункціональні вимоги - супроводжуваність):

- НФВ-СП-01: Код застосунку повинен бути структурованим, добре документованим (де необхідно) та відповідати загальноприйнятим стандартам кодування для відповідної платформи (Android/iOS), що спрощує його модифікацію та виправлення помилок.
- НФВ-СП-02: Архітектура застосунку повинна бути модульною, що дозволяє легко додавати нові функції або змінювати існуючі з мінімальним впливом на інші частини системи.

НФВ-МШ (нефункціональні вимоги - масштабованість):

- НФВ-МШ-01: Архітектура застосунку та структура зберігання даних (словника) повинні бути розраховані на ефективну роботу при значній кількості записів у словнику користувача (наприклад, до 10 000 слів) без суттєвої деградації продуктивності операцій зі словником.
- НФВ-МШ-02: Серверна інфраструктура повинна мати можливість масштабування для підтримки зростаючої кількості користувачів та обсягу контенту.

НФВ-НВ (нефункціональні вимоги - нормативна відповідність):

- НФВ-НВ-01: Застосунок повинен використовувати лише ті тексти книг, на розповсюдження яких є відповідні юридичні права (ліцензії або суспільне надбання).
- НФВ-НВ-02: Політика конфіденційності застосунку повинна бути чіткою, легко доступною для користувача та відповідати чинному законодавству про захист персональних даних (наприклад, GDPR, якщо застосунок орієнтований на відповідний ринок).

ДОДАТОК 8

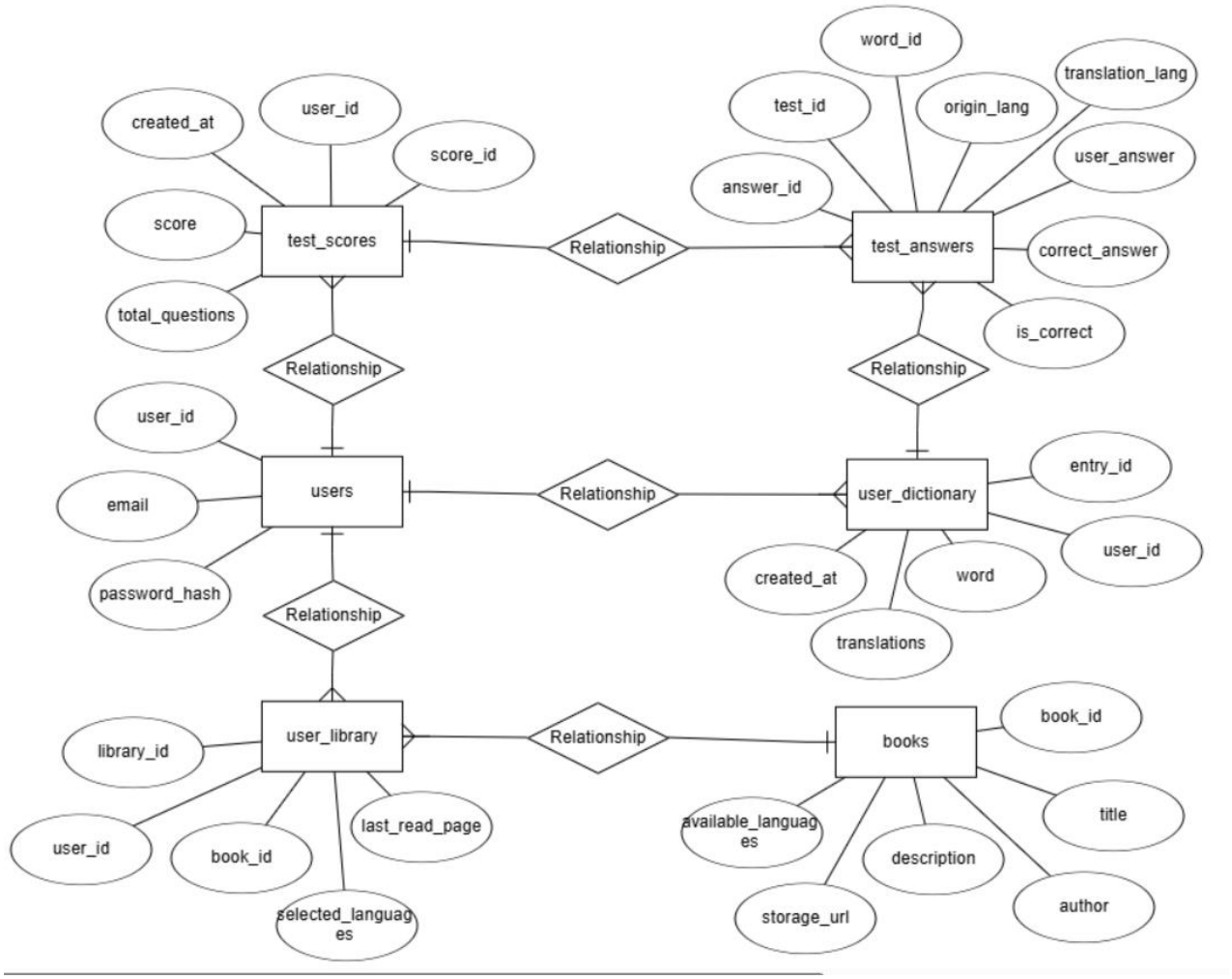
Мобільний застосунок для мультилінгвального читання книжок

Структурна схема бази даних

ІАЛЦ.467200.004 Д8

Аркушів 1

Київ 2025 р.



					ІАЛЦ.467200.011 Д8		
Зм.	Лист.	№ докум.	Підпис	Дата			
Розробив		Федосєєва О.Д.			**Мобільний застосунок для мультилінгвального читання книжок**	Літ.	Арк. 1 / Акрушів 1
Перевірив		Васильєва М.Д.					
Н. Контр.					Структурна схема бази даних	КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11	
Затвердив							

ДОДАТОК 9

Мобільний застосунок для мультилінгвального читання книжок

Лістинг програмного коду

ІАЛЦ.467200.004 Д9

Аркушів 50

Київ 2025 р.

authController.js

```
const authService =
require('../services/authService')
;

const register = async (req, res)
=> {
  const { email, password } =
req.body; // Get email and
password from request body
  try {
    const result = await
authService.registerUser(email,
password); // Call registration
service

res.status(result.status).json(res
ult.body); // Send response
  } catch (err) {
    console.error('Registration
error:', err); // Log error
    res.status(500).json({ error:
'Server error during registration'
}); // Send server error response
  }
};

const login = async (req, res) =>
{
  const { email, password } =
req.body; // Get email and
password from request body
  try {
    const result = await
authService.loginUser(email,
password); // Call login service

res.status(result.status).json(res
ult.body); // Send response
  } catch (err) {
    console.error('Login error:',
err); // Log error
    res.status(500).json({ error:
'Server error during login' }); //
Send server error response
  }
};

// Export auth controller
functions
module.exports = { register, login
};
```

bookController.js

```
const bookService =
require('../services/bookService')
;

getAllBooks = async (req, res) =>
{
  try {
    const books = await
bookService.fetchAllBooks(); //
Call service to get all books
    res.json(books); // Send books
as JSON response
  } catch (err) {
    console.error('Error fetching
books:', err); // Log error
    res.status(500).json({
message: 'Failed to retrieve
books' }); // Send error response
  }
};

// Export book controller
functions
module.exports = { getAllBooks }
```

db.js

```
const { Pool } = require('pg'); //
PostgreSQL client
require('dotenv').config(); //
Loads environment variables

// Create a PostgreSQL connection
pool using the DATABASE_URL from
.env
const pool = new Pool({
  connectionString:
process.env.DATABASE_URL,
});

module.exports = pool;
```

authMiddleware.js

```
const jwt =
require('jsonwebtoken'); // For
JWT creation and verification
require('dotenv').config();

// Middleware to authenticate
users via JWT
const authenticate = (req, res,
next) => {
  const authHeader =
req.headers.authorization; //
Expect "Bearer <token>"

  if (!authHeader ||
!authHeader.startsWith('Bearer '))
{
    return res.status(401).json({
message: 'No token or invalid
format' });
  }

  const token = authHeader.split('
')[1];

  try {
    // Verify token using the
secret from .env
    const decoded =
jwt.verify(token,
process.env.JWT_SECRET);
    req.user = decoded; // Attach
decoded user info to the request
    next(); // Proceed to the next
middleware or route handler
  } catch (err) {
    res.status(401).json({
message: 'Invalid token' });
  }
};

module.exports = authenticate;
```

bookModel.js

```
const pool = require('./db'); //
Import database connection pool

// Fetches all books from the
'books' table
const getAllBooks = async () => {
  const result = await pool.query(
    'SELECT id, title, author,
description, r2_url,
available_languages FROM books'
  );
  return result.rows; // Return
array of book objects
};

module.exports = { getAllBooks };
```

userModel.js

```
const pool = require('./db');

//  Find  a  user  by  their  email
address
const   getUserByEmail   =   async
(email) => {
  const     result     =     await
pool.query('SELECT  *  FROM  users
WHERE email = $1', [email]);
  return result.rows[0]; // Return
the first user found or undefined
};

// Create a new user with email and
hashed password
const  createUser  =  async  (email,
hashedPassword) => {
  const result = await pool.query(
    'INSERT  INTO  users  (email,
password_hash)  VALUES  ($1,  $2)
RETURNING id, email', // Return ID
and email of new user
    [email, hashedPassword]
  );
  return result.rows[0];
};

module.exports = { getUserByEmail,
createUser };
```

authRoutes.js

```
const express =
require('express');
// Import controller functions for
user registration and login
const { register, login } =
require('../controllers/authContro
ller');
```

```
const router = express.Router();
// Initialize Express router

// Define routes for user
authentication
router.post('/register',
register); // Handles new user
registration
router.post('/login', login);
// Handles user login

module.exports = router;
```

bookRoutes.js

```
const express = require('express');
// Import controller function to get
all books
const    {    getAllBooks    }    =
require('../controllers/bookContro
ller');
// Import authentication middleware
to protect routes
const        authenticate        =
require('../middleware/authMiddlew
are');

const router = express.Router();

// Define routes for book-related
operations
// GET /api/books - Fetches all
books, protected by authentication
router.get('/',       authenticate,
getAllBooks);

module.exports = router;
```

server.js

```
const http = require('http');
const express =
require('express');
const cors = require('cors'); //
Enables Cross-Origin Resource
Sharing
require('dotenv').config();

const app = express();

// Import route handlers
const authRoutes =
require('./routes/authRoutes');
const bookRoutes =
require('./routes/bookRoutes');
const libraryRoutes =
require('./routes/libraryRoutes');
const dictionaryRoutes =
require('./routes/dictionaryRoutes
');
const practiceRoutes =
require('./routes/practiceRoutes')
;

// Middleware
app.use(cors()); // Allow requests
from different origins
app.use(express.json()); // Parse
JSON request bodies

// Mount API routes
app.use('/api/auth', authRoutes);
app.use('/api/books', bookRoutes);
app.use('/api/library',
libraryRoutes);
app.use('/api/dictionary',
dictionaryRoutes);
app.use('/api/practice',
practiceRoutes);

// Server port configuration
const PORT = process.env.PORT ||
4000;

// Start the HTTP server
http.createServer(app).listen(PORT
, '0.0.0.0', () => {
    console.log(`HTTP is working
on port: ${PORT}`);
});
```

db.js

```
const { Pool } = require('pg'); //
PostgreSQL client
require('dotenv').config(); //
Loads environment variables

// Create a PostgreSQL connection
pool using the DATABASE_URL from
.env
const pool = new Pool({
  connectionString:
process.env.DATABASE_URL,
});
```

```
// Export the pool for database
interactions throughout the app
module.exports = pool;
```

LoginActivity.kt

```
package
com.example.readingapp.features.auth
import …

class LoginActivity :
AppCompatActivity() {

    companion object {
        private const val TAG =
"LoginActivity" // TAG for logging
    }

    private lateinit var binding:
ActivityLoginBinding // View binding
instance for accessing layout views

        override fun
onCreate(savedInstanceState:
Bundle?) {

super.onCreate(savedInstanceState)
        binding =
ActivityLoginBinding.inflate(layoutI
nflater)
        setContentView(binding.root)

        // Set up click listener for
the login button

binding.loginButton.setOnClickListen
er {
            val email =
binding.emailEditText.text.toString(
).trim() // Trim whitespace
            val password =
binding.passwordEditText.text.toStri
ng() // Password usually not trimmed
by default, but can be

            // Basic input
validation
            if (email.isNotEmpty()
&& password.isNotEmpty()) {
                val loginRequest =
LoginRequest(email, password)
                Log.d(TAG,
"Attempting login for email:
$email")
                // Attempt to log in
via API call

ApiClient.getApiService(this@LoginAc
tivity.applicationContext)
                    .login(loginRequ
est).enqueue(object :
Callback<LoginResponse> {
                        override fun
onResponse(call:
Call<LoginResponse>, response:
Response<LoginResponse>) {
                            if
(response.isSuccessful &&
response.body() != null) {
                                //
Login successful, process the
authentication token
                                val
token = response.body()?.token
                                if
(!token.isNullOrEmpty()) { // Check
if token is not null and not empty

Log.i(TAG, "Login successful, token
received.")

AuthTokenManager.saveToken(this@Logi
nActivity.applicationContext, token)


// Navigate to main activity on
successful login

val mainIntent =
Intent(this@LoginActivity,
MainActivity::class.java).apply {

// Clear back stack so user can't go
back to login screen

flags =
Intent.FLAG_ACTIVITY_NEW_TASK or
Intent.FLAG_ACTIVITY_CLEAR_TASK

}

startActivity(mainIntent)

finish() // Close LoginActivity
                                }
else {
```

```
// Handle case where login is
successful but token is missing or
empty

Log.e(TAG, "Login successful but
token is null or empty.")

Toast.makeText(applicationContext,
getString(R.string.login_failed_toke
n_error), Toast.LENGTH_SHORT).show()
                                }
                            } else {
                                //
Login failed, show error message
from API response
                                val
errorBody =
response.errorBody()?.string() ?:
getString(R.string.unknown_error)

Log.e(TAG, "Login Failed - Code:
${response.code()}, Message:
${response.message()}, Body:
$errorBody")

Toast.makeText(applicationContext,
getString(R.string.login_failed_serv
er_message, response.message()),
Toast.LENGTH_SHORT).show()
                            }
                        }

                        override fun
onFailure(call: Call<LoginResponse>,
t: Throwable) {
                            //
Handle network errors or other
issues during API call

Log.e(TAG, "Network Error during
login", t)

Toast.makeText(applicationContext,
getString(R.string.network_error_mes
sage, t.message),
Toast.LENGTH_SHORT).show()
                        }
                    })
            } else {
                // Prompt user if
email or password fields are empty
                Log.w(TAG, "Login
attempt with empty email or
password.")

Toast.makeText(applicationContext,
getString(R.string.enter_email_passw
ord_prompt),
Toast.LENGTH_SHORT).show()
            }
        }

        // Set up click listener for
the "Go to Register" button/link

binding.goToRegisterTextView.setOnCl
ickListener {
            Log.d(TAG, "Navigating
to RegisterActivity.")

startActivity(Intent(this,
RegisterActivity::class.java))
        }
    }
}
```

RegisterActivity.kt

```
package
com.example.readingapp.features.auth
import …

class RegisterActivity :
AppCompatActivity() {

    companion object {
        private const val TAG =
"RegisterActivity" // TAG for
logging
    }

    private lateinit var binding:
ActivityRegisterBinding // View
binding instance for accessing
layout views

    override fun
onCreate(savedInstanceState:
Bundle?) {

super.onCreate(savedInstanceState)
        binding =
ActivityRegisterBinding.inflate(layo
utInflater)
        setContentView(binding.root)
```

```
        // Set up click listener for
the registration button

binding.registerButton.setOnClickLis
tener {
            val email =
binding.emailEditTextRegister.text.t
oString().trim()
            val password =
binding.passwordEditTextRegister.tex
t.toString().trim()

            // Validate input fields
before making an API call
            if (email.isEmpty() ||
password.isEmpty()) {
                Log.w(TAG,
"Registration attempt with empty
email or password.")
                Toast.makeText(this,
getString(R.string.enter_email_passw
ord_prompt),
Toast.LENGTH_SHORT).show()

return@setOnClickListener // Exit if
validation fails
            }

            if
(!Patterns.EMAIL_ADDRESS.matcher(ema
il).matches()) {
                Log.w(TAG,
"Registration attempt with invalid
email format: $email")
                Toast.makeText(this,
getString(R.string.invalid_email_pro
mpt), Toast.LENGTH_SHORT).show()

return@setOnClickListener // Exit if
email format is invalid
            }

            val registerRequest =
RegisterRequest(email, password)
            Log.d(TAG, "Attempting
registration for email: $email")

            // Attempt to register
user via API call

ApiClient.getApiService(this@Registe
rActivity.applicationContext)
                    .register(registerRe
quest).enqueue(object :
Callback<Void> {
                        override fun
onResponse(call: Call<Void>,
response: Response<Void>) {
                            if
(response.isSuccessful) {
                                //
Registration successful

Log.i(TAG, "Registration successful
for email: $email")

Toast.makeText(applicationContext,
getString(R.string.registration_succ
essful_prompt),
Toast.LENGTH_LONG).show()

                                //
Navigate to LoginActivity, clearing
the task stack so user can't go back
to registration
                                val
loginIntent =
Intent(this@RegisterActivity,
LoginActivity::class.java).apply {

flags =
Intent.FLAG_ACTIVITY_NEW_TASK or
Intent.FLAG_ACTIVITY_CLEAR_TASK
                                }

startActivity(loginIntent)
                                //
finish()
                            } else {
                                //
Registration failed, show error
message from API response
                                val
errorBody =
response.errorBody()?.string() ?:
getString(R.string.unknown_error)

Log.e(TAG, "Registration failed -
Code: ${response.code()}, Message:
${response.message()}, Body:
$errorBody")

Toast.makeText(applicationContext,
getString(R.string.registration_fail
ed_server_message,
```

```
response.message()),
Toast.LENGTH_LONG).show()
                            }
                        }

                        override fun
onFailure(call: Call<Void>, t:
Throwable) {
                            // Handle
network errors or other issues
during API call
                            Log.e(TAG,
"Network Error during registration",
t)

Toast.makeText(applicationContext,
getString(R.string.network_error_mes
sage_registration, t.message),
Toast.LENGTH_SHORT).show()
                        }
                    })
            }
        }
}
```

CatalogFragment.kt

```
package
com.example.readingapp.core.ui.catal
og
import …

class CatalogFragment : Fragment() {

    companion object {
        private const val TAG =
"CatalogFragment"
    }

    private var _binding:
FragmentCatalogBinding? = null
    private val binding get() =
_binding!! // This property is only
valid between onCreateView and
onDestroyView.

    private lateinit var
bookAdapter: BookAdapter
    private val viewModel:
CatalogViewModel by viewModels() //
Initialize ViewModel

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        Log.d(TAG, "onCreateView
called")
        _binding =
FragmentCatalogBinding.inflate(infla
ter, container, false)

        setupRecyclerView()

        return binding.root
    }

    override fun onViewCreated(view:
View, savedInstanceState: Bundle?) {
        super.onViewCreated(view,
savedInstanceState)
        Log.d(TAG, "onViewCreated
called")
        observeViewModel()
    }

       private fun
setupRecyclerView() {
        Log.d(TAG, "Setting up
RecyclerView")
        bookAdapter = BookAdapter
{ book ->
            Log.d(TAG, "Book item
clicked: ID ${book.id}, Title:
'${book.title}'")

showLanguageSelectionDialog(book)
        }

binding.recyclerViewBooks.apply {
            layoutManager =
LinearLayoutManager(requireContext()
)
            adapter = bookAdapter
        }
    }

    private fun observeViewModel() {
        Log.d(TAG, "Observing
ViewModel LiveData")

viewModel.books.observe(viewLifecycl
eOwner) { books ->
            Log.i(TAG, "Book list
updated with ${books.size} items.")

bookAdapter.submitList(books)
```

```
        }


viewModel.toastMessage.observe(viewL
ifecycleOwner) { event ->

event.getContentIfNotHandled()?.let
{ message ->
                Log.w(TAG,
"Displaying toast: $message")

Toast.makeText(requireContext(),
message, Toast.LENGTH_LONG).show()
            }
        }


viewModel.addToLibraryResult.observe
(viewLifecycleOwner) { event ->

event.getContentIfNotHandled()?.let
{ message ->
                Log.i(TAG, "Add to
library result: $message")

Toast.makeText(requireContext(),
message, Toast.LENGTH_SHORT).show()
            }
        }


viewModel.isLoading.observe(viewLife
cycleOwner) { isLoading ->
            // Here you can
show/hide a ProgressBar

//binding.progressBar.visibility =
if (isLoading) View.VISIBLE else
View.GONE
            Log.d(TAG, "isLoading
state changed to: $isLoading")
        }
    }

    private fun
showLanguageSelectionDialog(book:
Book) {
        Log.d(TAG, "Showing language
selection dialog for book ID:
${book.id}")
        val availableLangsArray =
book.available_languages.toTypedArra
y()
        val checkedItems =
BooleanArray(availableLangsArray.siz
e) // To track selected items


AlertDialog.Builder(requireContext()
)
            .setTitle("Select 2-4
Languages") // Title for the dialog
            .setMultiChoiceItems(ava
ilableLangsArray, checkedItems) { _,
which, isChecked ->
                // Update the
checkedItems array when an item's
state changes
                checkedItems[which]
= isChecked
            }
            .setPositiveButton("Add
to Library") { _, _ ->
                val
selectedLanguages =
availableLangsArray.filterIndexed
{ index, _ -> checkedItems[index] }
                Log.d(TAG, "Selected
languages for book ID ${book.id}:
$selectedLanguages")

                if
(selectedLanguages.size < 2 ||
selectedLanguages.size > 4) {
                    Log.w(TAG,
"Invalid number of languages
selected: ${selectedLanguages.size}.
Required 2-4.")

Toast.makeText(requireContext(),
"Please select between 2 and 4
languages.",
Toast.LENGTH_SHORT).show()
                } else {

viewModel.addToLibrary(book.id,
selectedLanguages) // Call ViewModel
to add to library
                }
            }
            .setNegativeButton("Canc
el") { dialog, _ ->
                Log.d(TAG, "Language
selection cancelled for book ID:
${book.id}")
                dialog.dismiss()
```

```
            }
            .show()
    }

    override fun onDestroyView() {
        super.onDestroyView()
        Log.d(TAG, "onDestroyView
called")
        _binding = null // Important
to prevent memory leaks
    }
}
```

CatalogViewModel.kt

```
package
com.example.readingapp.core.ui.catal
og
import …

class CatalogViewModel(application:
Application) :
AndroidViewModel(application) {

    companion object {
        private const val TAG =
"CatalogViewModel"
    }

    private val apiService =
ApiClient.getApiService(application.
applicationContext)

    // LiveData to hold the list of
books from the catalog
    private val _books =
MutableLiveData<List<Book>>()
    val books: LiveData<List<Book>>
= _books

    // LiveData for general toast
messages (e.g., error fetching
books)
    private val _toastMessage =
MutableLiveData<Event<String>>()
    val toastMessage:
LiveData<Event<String>> =
_toastMessage

    // LiveData for the result of
adding a book to the library
    private val _addToLibraryResult
= MutableLiveData<Event<String>>()
    val addToLibraryResult:
LiveData<Event<String>> =
_addToLibraryResult

    private val _isLoading =
MutableLiveData<Boolean>()
    val isLoading: LiveData<Boolean>
= _isLoading

    init {
        Log.d(TAG, "ViewModel
initialized")
        fetchBooks()
    }

    private fun fetchBooks() {
        Log.i(TAG, "Attempting to
fetch books from catalog...")
        _isLoading.value = true
        viewModelScope.launch {
            try {
                val response =
apiService.getBooks()
                if
(response.isSuccessful) {
                    val fetchedBooks
= response.body() ?: emptyList()

_books.postValue(fetchedBooks)
                    Log.i(TAG,
"Successfully fetched
${fetchedBooks.size} books.")
                } else {
                    Log.e(TAG,
"Error fetching books. Code:
${response.code()}, Message:
${response.message()}")

_toastMessage.postValue(Event("Error
fetching books:
${response.code()}"))
                }
            } catch (e: Exception) {
                Log.e(TAG,
"Exception while fetching books", e)

_toastMessage.postValue(Event("Faile
d to fetch books: ${e.message}"))
            } finally {

_isLoading.postValue(false)
            }
        }
```

```
    }

    fun addToLibrary(bookId: Int,
languages: List<String>) {
        Log.i(TAG, "Attempting to
add book ID $bookId to library with
languages: $languages")
        _isLoading.value = true //
You might want a separate loading
state for this action
        val request =
AddToLibraryRequest(
            book_id = bookId,
            selected_languages =
languages
        )

        viewModelScope.launch {
            try {
                val response =
apiService.addToLibrary(request)
                if
(response.isSuccessful) {
                    val message =
response.body()?.message ?: "Book
added to library successfully"
                    Log.i(TAG, "Book
ID $bookId added to library.
Response: $message")

_addToLibraryResult.postValue(Event(
message))
                } else {
                    val errorBody =
response.errorBody()?.string()
                    Log.e(TAG,
"Error adding book ID $bookId to
library. Code: ${response.code()},
Body: $errorBody")

_addToLibraryResult.postValue(Event(
"Error adding book:
${response.code()} ${errorBody ?:
""}"))
                }
            } catch (e: Exception) {
                Log.e(TAG,
"Exception while adding book ID
$bookId to library", e)

_addToLibraryResult.postValue(Event(
"Failed to add book: ${e.message}"))
            } finally {
_isLoading.postValue(false) // Reset
loading state
            }
        }
    }
}
```

LibraryFragment.kt

```
package
com.example.readingapp.core.ui.libra
ry
import …

class LibraryFragment : Fragment() {

    companion object {
        private const val TAG =
"LibraryFragment"
    }

    private var _binding:
FragmentLibraryBinding? = null
    private val binding get() =
_binding!! // Valid between
onCreateView and onDestroyView.

    private lateinit var
libraryAdapter: LibraryAdapter
    private val viewModel:
LibraryViewModel by viewModels() //
Initialize ViewModel

    override fun onCreateView(
        inflater: LayoutInflater,
container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        Log.d(TAG, "onCreateView
called")
        _binding =
FragmentLibraryBinding.inflate(infla
ter, container, false)

        setupRecyclerView()
        // ViewModel's init block
calls fetchUserLibrary, so no need
to call it explicitly here

        return binding.root
    }
```

```
    override fun onViewCreated(view:
View, savedInstanceState: Bundle?) {
        super.onViewCreated(view,
savedInstanceState)
        Log.d(TAG, "onViewCreated
called")
        observeViewModel()
    }

    private fun setupRecyclerView()
{
        Log.d(TAG, "Setting up
RecyclerView for library books")
        libraryAdapter =
LibraryAdapter(
            onItemClick = { book ->
                Log.i(TAG, "Book
item clicked: ID
${book.effectiveId}, Title:
'${book.title}'. Launching
ReaderActivity.")
                val intent =
Intent(requireContext(),
ReaderActivity::class.java).apply {

putExtra("r2_url", book.r2_url)

putExtra("book_id",
book.effectiveId)

putStringArrayListExtra("selected_la
nguages",
ArrayList(book.selected_languages))

putExtra("book_title", book.title)
                }

startActivity(intent)
            },
            onRemoveClick = { book
->
                Log.d(TAG, "Remove
button clicked for book ID:
${book.effectiveId}")
                // Optionally, show
a confirmation dialog before
removing

confirmAndRemoveBook(book)
            },
            onChangeLanguagesClick =
{ book ->
                Log.d(TAG, "Change
languages clicked for book ID:
${book.effectiveId}")

showLanguageSelectionDialog(book)
            }
        )

binding.recyclerViewLibrary.apply {
            layoutManager =
LinearLayoutManager(requireContext()
)
            adapter = libraryAdapter
        }
    }

    private fun observeViewModel() {
        Log.d(TAG, "Observing
LibraryViewModel LiveData")

viewModel.libraryBooks.observe(viewL
ifecycleOwner) { books ->
            Log.i(TAG, "Library book
list updated with ${books.size}
items.")

libraryAdapter.submitList(books)
            // You might want to
show/hide an "empty library" message
here
            //
binding.emptyLibraryTextView.visibil
ity = if (books.isEmpty())
View.VISIBLE else View.GONE
        }


viewModel.toastMessage.observe(viewL
ifecycleOwner) { event ->

event.getContentIfNotHandled()?.let
{ message ->
                Log.w(TAG,
"Displaying general toast:
$message")

Toast.makeText(requireContext(),
message, Toast.LENGTH_LONG).show()
            }
        }


viewModel.operationResult.observe(vi
ewLifecycleOwner) { event ->
```

```
event.getContentIfNotHandled()?.let
{ message ->
                Log.i(TAG,
"Operation result: $message")

Toast.makeText(requireContext(),
message, Toast.LENGTH_SHORT).show()
            }
        }


viewModel.isLoading.observe(viewLife
cycleOwner) { isLoading ->
            // Connect to a
ProgressBar in your layout
            //
binding.libraryProgressBar.visibilit
y = if (isLoading) View.VISIBLE else
View.GONE
            Log.d(TAG, "isLoading
state changed to: $isLoading")
        }
    }

    private fun
confirmAndRemoveBook(book: Book) {

AlertDialog.Builder(requireContext()
)
            .setTitle("Remove Book")
            .setMessage("Are you
sure you want to remove
'${book.title}' from your library?")
            .setPositiveButton("Remo
ve") { _, _ ->
                Log.i(TAG,
"Confirmed removal for book ID:
${book.effectiveId}")

viewModel.removeBookFromLibrary(book
.effectiveId)
            }
            .setNegativeButton("Canc
el", null)
            .show()
    }

    private fun
showLanguageSelectionDialog(book:
Book) {
        Log.d(TAG, "Showing language
selection dialog for book ID:
${book.effectiveId} (library)")
        val availableLangsArray =
book.available_languages.toTypedArra
y()
        // Pre-select currently
selected languages
        val checkedItems =
BooleanArray(availableLangsArray.siz
e) { index ->

book.selected_languages.contains(ava
ilableLangsArray[index])
        }


AlertDialog.Builder(requireContext()
)
            .setTitle("Change
Selected Languages")
            .setMultiChoiceItems(ava
ilableLangsArray, checkedItems) { _,
which, isChecked ->
                checkedItems[which]
= isChecked // Update selection
state
            }
            .setPositiveButton("Save
Changes") { _, _ ->
                val
newSelectedLanguages =
availableLangsArray.filterIndexed
{ index, _ -> checkedItems[index] }
                Log.d(TAG, "New
selected languages for book ID
${book.effectiveId}:
$newSelectedLanguages")

                if
(newSelectedLanguages.size < 2) { //
Assuming same validation as catalog
                    Log.w(TAG,
"Invalid number of languages
selected:
${newSelectedLanguages.size}.
Required at least 2.")

Toast.makeText(requireContext(),
"Please select at least 2
languages.",
Toast.LENGTH_SHORT).show()
                } else {
                    // Call
ViewModel to update languages
```

```
viewModel.updateLanguagesForBook(boo
k.effectiveId, newSelectedLanguages)
                }
            }
            .setNegativeButton("Canc
el") { dialog, _ ->
                Log.d(TAG, "Language
change cancelled for book ID:
${book.effectiveId}")
                dialog.dismiss()
            }
            .show()
    }

    override fun onDestroyView() {
        super.onDestroyView()
        Log.d(TAG, "onDestroyView
called")
        // No need to cancel
coroutineScope here as ViewModel
uses viewModelScope which is
lifecycle-aware.
        _binding = null // Important
for preventing memory leaks
    }
}
```

LibraryViewModel.kt

```
package
com.example.readingapp.core.ui.libra
ry
import …

class LibraryViewModel(application:
Application) :
AndroidViewModel(application) {

    companion object {
        private const val TAG =
"LibraryViewModel"
    }

    private val apiService =
ApiClient.getApiService(application.
applicationContext)

    // LiveData to hold the list of
books in the user's library
    private val _libraryBooks =
MutableLiveData<List<Book>>()
    val libraryBooks:
LiveData<List<Book>> = _libraryBooks

    // LiveData for general toast
messages
    private val _toastMessage =
MutableLiveData<Event<String>>()
    val toastMessage:
LiveData<Event<String>> =
_toastMessage

    // LiveData for specific
operation results (e.g., removal,
language update)
    private val _operationResult =
MutableLiveData<Event<String>>()
    val operationResult:
LiveData<Event<String>> =
_operationResult

    private val _isLoading =
MutableLiveData<Boolean>()
    val isLoading: LiveData<Boolean>
= _isLoading

    init {
        Log.d(TAG, "ViewModel
initialized")
        fetchUserLibrary()
    }

    fun fetchUserLibrary() {
        Log.i(TAG, "Attempting to
fetch user's library books...")
        _isLoading.value = true
        viewModelScope.launch {
            try {
                val response =
apiService.getUserLibrary()
                if
(response.isSuccessful) {
                    val books =
response.body() ?: emptyList()

_libraryBooks.postValue(books)
                    Log.i(TAG,
"Successfully fetched ${books.size}
books from user library.")
                } else {
                    val errorBody =
response.errorBody()?.string()
                    Log.e(TAG,
"Error fetching user library. Code:
${response.code()}, Body:
```

```
$errorBody")

_toastMessage.postValue(Event("Error
loading library:
${response.code()}"))
                }
            } catch (e: Exception) {
                Log.e(TAG,
"Exception while fetching user
library", e)

_toastMessage.postValue(Event("Conne
ction error: ${e.message}"))
            } finally {

_isLoading.postValue(false)
            }
        }
    }

    fun
removeBookFromLibrary(bookId: Int) {
        Log.i(TAG, "Attempting to
remove book ID $bookId from
library.")
        _isLoading.value = true
        viewModelScope.launch {
            try {
                // Ensure bookId is
the correct identifier for the API
endpoint
                val response =
apiService.removeFromLibrary(bookId)
                if
(response.isSuccessful) {
                    Log.i(TAG, "Book
ID $bookId removed successfully.")

_operationResult.postValue(Event("Bo
ok removed successfully"))

fetchUserLibrary() // Refresh the
list after removal
                } else {
                    val errorBody =
response.errorBody()?.string()
                    Log.e(TAG,
"Error removing book ID $bookId.
Code: ${response.code()}, Body:
$errorBody")

_operationResult.postValue(Event("Er
ror removing book:
${response.code()}"))
                }
            } catch (e: Exception) {
                Log.e(TAG,
"Exception while removing book ID
$bookId", e)

_operationResult.postValue(Event("Co
nnection error: ${e.message}"))
            } finally {

_isLoading.postValue(false)
            }
        }
    }
        fun
updateLanguagesForBook(bookId: Int,
newLanguages: List<String>) {
        Log.i(TAG, "Attempting to
update languages for book ID $bookId
to: $newLanguages")
        _isLoading.value = true
        val request =
UpdateLanguagesRequest(selected_lang
uages = newLanguages)
        viewModelScope.launch {
            try {
                // Ensure bookId is
the correct identifier for the API
endpoint
                val response =
apiService.updateLanguages(bookId,
request)
                if
(response.isSuccessful) {
                    Log.i(TAG,
"Languages updated successfully for
book ID $bookId.")

_operationResult.postValue(Event("La
nguages updated successfully"))

fetchUserLibrary() // Refresh list
to see changes reflected
                } else {
                    val errorBody =
response.errorBody()?.string()
                    Log.e(TAG,
"Error updating languages for book
ID $bookId. Code:
${response.code()}, Body:
$errorBody")
```

```
_operationResult.postValue(Event("Er
ror updating languages:
${response.code()}"))
                }
            } catch (e: Exception) {
                Log.e(TAG,
"Exception while updating languages
for book ID $bookId", e)

_operationResult.postValue(Event("Co
nnection error: ${e.message}"))
            } finally {

_isLoading.postValue(false)
            }
        }
    }
}
```

ReaderActivity.kt

```
package
com.example.readingapp.features.read
er
import …

class ReaderActivity :
AppCompatActivity() {

    // View Properties
    private lateinit var viewPager:
ViewPager2
    private lateinit var
pageSeekBar: SeekBar
    private lateinit var
pageCountText: TextView
    private lateinit var
jumpToPageInput: EditText
    private lateinit var goButton:
Button
    private lateinit var
prevPageButton: ImageButton
    private lateinit var
nextPageButton: ImageButton

    // State & Helper Properties
    private val coroutineScope =
CoroutineScope(Dispatchers.Main +
Job()) // Scope for background tasks
tied to this activity
    private var totalPages = 0
    private lateinit var bookId:
String
    private lateinit var
readingPositionManager:
ReadingPositionManager
    private var isInitialLoad = true
// Flag to prevent saving position
before initial load completes
    private val savePositionHandler
= Handler(Looper.getMainLooper()) //
Handler for debouncing scroll save

    // Intent Data Properties
    private var r2Url: String? =
null
    private var selectedLanguages:
List<String> = emptyList()

    override fun
onCreate(savedInstanceState:
Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity_rea
der)

        initViews()

        if (!processIntentData()) {
            // If essential intent
data is missing, finish the activity
            Log.e(TAG, "Essential
intent data missing. Finishing
activity.")
            Toast.makeText(this,
"Error: Could not load book.",
Toast.LENGTH_LONG).show()
            finish()
            return
        }

        readingPositionManager =
ReadingPositionManager(this.applicat
ionContext)
        setupNavigationControls()
        loadAndDisplayBookContent()
    }

    private fun initViews() {
        viewPager =
findViewById(R.id.viewPager)
        pageSeekBar =
findViewById(R.id.pageSeekBar)
        pageCountText =
findViewById(R.id.pageCountText)
```

```
        jumpToPageInput =
findViewById(R.id.jumpToPageInput)
        goButton =
findViewById(R.id.goButton)
        prevPageButton =
findViewById(R.id.prevPageButton)
        nextPageButton =
findViewById(R.id.nextPageButton)
    }

    private fun processIntentData():
Boolean {
        bookId =
intent.getStringExtra("book_id") ?:
intent.getIntExtra("book_id",
            -1).takeIf { it != -
1 }?.toString() ?: ""
        if (bookId.isEmpty()) {
            Log.e(TAG, "Missing
'book_id' intent extra.")
            return false
        }

        val bookTitle =
intent.getStringExtra("book_title")
        supportActionBar?.title = if
(!bookTitle.isNullOrEmpty())
bookTitle else "Reader"

        r2Url =
intent.getStringExtra("r2_url")
        if (r2Url == null) {
            Log.e(TAG, "Missing
'r2_url' intent extra.")
            return false
        }

        val rawSelectedLanguages =
intent.getStringArrayListExtra("sele
cted_languages")
        if
(rawSelectedLanguages.isNullOrEmpty(
)) {
            Log.e(TAG, "Missing or
empty 'selected_languages' intent
extra.")
            return false
        }
        selectedLanguages =
rawSelectedLanguages.map
{ it.trim() }
        Log.d(TAG, "Selected
Languages for reader:
$selectedLanguages")
        return true
    }

    private fun
loadAndDisplayBookContent() {
        coroutineScope.launch {
            try {
                Log.d(TAG, "Fetching
book content from: $r2Url")
                val jsonText =
withContext(Dispatchers.IO) {

URL(r2Url).readText() // Network
call on IO dispatcher
                }

                val bookJson =
JSONObject(jsonText)
                val chaptersArray =
bookJson.getJSONArray("chapters")
                val
allChapterParagraphs =
mutableListOf<ChapterParagraph>()

                Log.d(TAG,
"Processing
${chaptersArray.length()}
chapters.")
                for (i in 0 until
chaptersArray.length()) {
                    val chapter =
chaptersArray.getJSONObject(i)
                    val
currentChapterTitleJson =
chapter.optJSONObject("title")
                    val
paragraphsArray =
chapter.getJSONArray("paragraphs")

                    if
(paragraphsArray.length() == 0) {
                        Log.w(TAG,
"Chapter index $i
('${currentChapterTitleJson?.toStrin
g()}') has no paragraphs.
Skipping.")
                        continue
                    }

                    for (j in 0
until paragraphsArray.length()) {
                        val
```

```
paragraph =
paragraphsArray.getJSONObject(j)

allChapterParagraphs.add(

ChapterParagraph(

chapterTitleJson =
currentChapterTitleJson ?:
JSONObject(),

paragraphJson = paragraph,

isFirstInChapter = (j == 0) // Mark
if it's the first paragraph of a
chapter
                            )
                        )
                    }
                }
                Log.i(TAG, "Total
paragraphs loaded:
${allChapterParagraphs.size}")

                if
(allChapterParagraphs.isNotEmpty())
{

withContext(Dispatchers.Main) { //
Switch back to Main thread for UI
updates
                        if
(viewPager.adapter == null) { //
Ensure adapter is set only once

viewPager.adapter =
ParagraphPagerAdapter(

this@ReaderActivity,

allChapterParagraphs,

selectedLanguages,

lifecycleScope // Use lifecycleScope
for adapter's coroutines
                            )

totalPages =
allChapterParagraphs.size

initializeNavigation() // Setup
seekbar and page counts
                            if
(totalPages > 0) {

loadReadingPosition() // Load last
read position
                            } else {

isInitialLoad = false // No pages,
so initial load is effectively done
                            }
                        }
                    }
                } else {
                    Log.e(TAG, "No
paragraphs to display after
parsing.")

Toast.makeText(this@ReaderActivity,
"Book content is empty.",
Toast.LENGTH_LONG).show()
                }
            } catch (e: Exception) {
                Log.e(TAG, "Error
loading or parsing book content", e)

Toast.makeText(this@ReaderActivity,
"Error loading book: ${e.message}",
Toast.LENGTH_LONG).show()
            }
        }
    }

    private fun
setupNavigationControls() {

viewPager.registerOnPageChangeCallba
ck(object :
ViewPager2.OnPageChangeCallback() {
            override fun
onPageSelected(position: Int) {

super.onPageSelected(position)

updatePageIndicators(position) //
Update UI when page changes
            }
        })


pageSeekBar.setOnSeekBarChangeListen
er(object :
SeekBar.OnSeekBarChangeListener {
```

```
            override fun
onProgressChanged(seekBar: SeekBar?,
progress: Int, fromUser: Boolean) {
                if (fromUser &&
progress < totalPages) {
                    // No immediate
jump, wait for onStopTrackingTouch
for smoother UX
                    //
updatePageIndicators(progress) //
Optionally update text immediately
                }
            }
            override fun
onStartTrackingTouch(seekBar:
SeekBar?) {}
            override fun
onStopTrackingTouch(seekBar:
SeekBar?) {
                seekBar?.let {
                    if (it.progress
< totalPages) {

viewPager.currentItem = it.progress
// Jump to page when user releases
seekbar
                    }
                }
            }
        })


prevPageButton.setOnClickListener {
            if
(viewPager.currentItem > 0) {

viewPager.currentItem -= 1
            }
        }


nextPageButton.setOnClickListener {
            if
(viewPager.currentItem < totalPages
- 1) {

viewPager.currentItem += 1
            }
        }

        goButton.setOnClickListener
{ jumpToPage() }

jumpToPageInput.setOnEditorActionLis
tener { _, actionId, _ ->
            if (actionId ==
EditorInfo.IME_ACTION_GO) {
                jumpToPage()
                true
            } else {
                false
            }
        }
    }

    private fun
initializeNavigation() {
        if (totalPages > 0) {
            pageSeekBar.max =
totalPages - 1

updatePageIndicators(viewPager.curre
ntItem) // Initialize with current
or 0
            pageSeekBar.visibility =
View.VISIBLE
        } else {
            pageSeekBar.visibility =
View.GONE
        }
    }

    @SuppressLint("SetTextI18n")
    private fun
updatePageIndicators(position: Int)
{
        if (position < 0 ||
position >= totalPages) return //
Safety check

        if (pageSeekBar.progress !=
position) {
            pageSeekBar.progress =
position
        }

        val currentPageForDisplay =
position + 1 // User-facing page
number (1-indexed)
        pageCountText.text = "Page
$currentPageForDisplay of
$totalPages"
        prevPageButton.isEnabled =
position > 0
        nextPageButton.isEnabled =
```

```
position < totalPages - 1
    }

    private fun jumpToPage() {
        try {
            val pageNumber =
jumpToPageInput.text.toString().toIn
t()
            if (pageNumber in
1..totalPages) {
                val position =
pageNumber - 1 // Convert to 0-
indexed

viewPager.currentItem = position

jumpToPageInput.text.clear()

                // Hide keyboard
after jumping
                val imm =
getSystemService(Context.INPUT_METHO
D_SERVICE) as InputMethodManager

imm.hideSoftInputFromWindow(jumpToPa
geInput.windowToken, 0)
            } else {
                Toast.makeText(this,
"Page number out of range (1-
$totalPages)",
Toast.LENGTH_SHORT).show()
            }
        } catch (e:
NumberFormatException) {
            Toast.makeText(this,
"Please enter a valid page number",
Toast.LENGTH_SHORT).show()
        }
    }

    private fun
loadReadingPosition() {
        Log.d(TAG,
"loadReadingPosition called.
isInitialLoad: $isInitialLoad")

readingPositionManager.loadPosition(
bookId) { page, scrollPosition ->
            if (page in 0 until
totalPages) {
                Log.d(TAG, "Position
loaded: page=$page,
scroll=$scrollPosition. Applying.")

viewPager.setCurrentItem(page,
false) // Set initial page without
smooth scroll

                // Post scroll to
ensure ViewPager and ViewHolder are
ready
                viewPager.post {
                    val
currentViewHolder =
(viewPager.getChildAt(0) as?
RecyclerView)
                        ?.findViewHo
lderForAdapterPosition(page) as?
ParagraphPagerAdapter.ParagraphViewH
older

currentViewHolder?.rootScrollView?.s
crollTo(0, scrollPosition)
                    Log.d(TAG,
"Scrolled to $scrollPosition on page
$page. Initial load complete.")
                    isInitialLoad =
false
                }
            } else {
                Log.w(TAG, "Loaded
page $page is out of bounds (total:
$totalPages). Defaulting to page
0.")

viewPager.setCurrentItem(0, false)
                isInitialLoad =
false
            }
        }
    }

    private fun
saveReadingPosition(page: Int,
scrollPosition: Int) {
        Log.i(TAG, "Saving reading
position: book=$bookId, page=$page,
scroll=$scrollPosition")

readingPositionManager.savePosition(
bookId, page, scrollPosition)
    }

    override fun onPause() {
        super.onPause()
        Log.d(TAG, "onPause.
```

```
isInitialLoad: $isInitialLoad")

        if (isInitialLoadComplete()
&& ::bookId.isInitialized &&
bookId.isNotEmpty()
&& ::readingPositionManager.isInitia
lized) {
            val currentPage =
viewPager.currentItem
            var scrollPosition = 0

            // Attempt to get scroll
position from the current visible
ViewHolder
            (viewPager.getChildAt(0)
as?
RecyclerView)?.layoutManager?.findVi
ewByPosition(currentPage)?.let
{ itemView ->

(itemView.findViewById(R.id.root_scr
oll_view) ?: (itemView as?
ScrollView))?.let {
                    scrollPosition =
it.scrollY
                }
            }
            // Fallback if the
direct view isn't a ScrollView or ID
doesn't match
            if (scrollPosition == 0)
{
                val
currentViewHolder =
(viewPager.getChildAt(0) as?
RecyclerView)
                    ?.findViewHolder
ForAdapterPosition(currentPage) as?
ParagraphPagerAdapter.ParagraphViewH
older
                scrollPosition =
currentViewHolder?.rootScrollView?.s
crollY ?: 0
            }


saveReadingPosition(currentPage,
scrollPosition)
        } else {
            Log.w(TAG, "Skipping
savePosition in onPause. Conditions
not met.")
        }
    }


    override fun onDestroy() {
        super.onDestroy()
        coroutineScope.cancel() //
Cancel coroutines to prevent leaks

savePositionHandler.removeCallbacksA
ndMessages(null) // Clean up handler
    }

    fun isInitialLoadComplete():
Boolean {
        return !isInitialLoad
    }

    fun
saveScrollPositionForPage(page: Int,
scrollPosition: Int) {
        // Only save if it's for the
currently viewed page to avoid
conflicts
        if (page ==
viewPager.currentItem &&
isInitialLoadComplete()) {

savePositionHandler.removeCallbacksA
ndMessages(null) // Remove previous
pending saves

savePositionHandler.postDelayed({ //
Schedule a new save

saveReadingPosition(page,
scrollPosition)
            }, 500) // Delay of
500ms
        }
    }

    companion object {
        private const val TAG =
"ReaderActivity" // TAG for logging
    }
}
```

DictionaryFragment.kt

```
package
com.example.readingapp.core.ui.dicti
onary
import …
```

```
class DictionaryFragment :
Fragment() {

    //region Companion Object and
Constants
    private companion object {
        private const val TAG =
"DictionaryFragment"
    }
    //endregion

    //region Properties
    private var _binding:
FragmentDictionaryBinding? = null
    private val binding get() =
_binding!! // Non-null assertion for
guaranteed lifecycle access

    private val viewModel:
DictionaryViewModel by viewModels()
// Lazily initialized ViewModel
    private lateinit var adapter:
DictionaryAdapter
    private var mainActivity:
MainActivity? = null // Reference to
MainActivity for FAB interaction
    //endregion

    //region Fragment Lifecycle
Methods
    override fun onAttach(context:
Context) {
        super.onAttach(context)
        if (context is MainActivity)
{
            mainActivity = context
// Store MainActivity instance
        }
    }

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding =
FragmentDictionaryBinding.inflate(in
flater, container, false)
        return binding.root
    }

    override fun onViewCreated(view:
View, savedInstanceState: Bundle?) {
        super.onViewCreated(view,
savedInstanceState)
        Log.d(TAG, "onViewCreated:
Initializing UI components.")

        setupRecyclerView()
        setupSearchFunctionality()

setupLanguageFilterSpinners()
        setupRefreshLayout()
        setupFabClickListener()
        setupObservers()
        showFab()
    }

    override fun onResume() {
        super.onResume()
        Log.d(TAG, "onResume:
Refreshing dictionary data.")
        showFab()

        // Refresh dictionary data,
respecting current filters but
forcing a network fetch.
        viewModel.loadDictionary(
            originLang =
viewModel.currentOriginLangFilter,
            translationLang =
viewModel.currentTranslationLangFilt
er,
            forceReload = true
        )
    }

    override fun onDestroyView() {
        super.onDestroyView()
        Log.d(TAG, "onDestroyView:
Clearing binding.")
        binding.recyclerView.adapter
= null // Clear adapter reference
from RecyclerView
        _binding = null // Avoid
memory leaks by clearing the binding
reference
    }

    override fun onDetach() {
        super.onDetach()
        Log.d(TAG, "onDetach:
Clearing MainActivity reference.")
        mainActivity = null // Clear
reference to MainActivity
```

```
    }
    //endregion

    //region UI Setup Methods

    private fun setupRecyclerView()
{
        adapter = DictionaryAdapter(
            onDeleteClick = { word
->
                Log.d(TAG, "Delete
clicked for word: ${word.word}")

confirmDeleteWord(word)
            },
            onEditClick = { word ->
                Log.d(TAG, "Edit
clicked for word: ${word.word}")

EditWordDialogFragment.newInstance(w
ord).show(childFragmentManager,
EditWordDialogFragment.TAG)
            }
        )

        binding.recyclerView.apply {
            layoutManager =
LinearLayoutManager(requireContext()
)

addItemDecoration(DividerItemDecorat
ion(requireContext(),
DividerItemDecoration.VERTICAL))
            this.adapter =
this@DictionaryFragment.adapter //
Set the adapter instance
        }
    }

    private fun
setupSearchFunctionality() {

binding.searchButton.setOnClickListe
ner {
            val query =
binding.searchInput.text.toString().
trim()
            Log.d(TAG, "Search
button clicked. Query: '$query'")
            if (query.isNotEmpty())
{

viewModel.searchWord(query)
            } else {

viewModel.loadDictionary(viewModel.c
urrentOriginLangFilter,
viewModel.currentTranslationLangFilt
er)
            }
        }


binding.searchInput.setOnEditorActio
nListener { _, actionId, _ ->
            if (actionId ==
EditorInfo.IME_ACTION_SEARCH) {
                val query =
binding.searchInput.text.toString().
trim()
                if
(query.isNotEmpty()) {
                    Log.d(TAG,
"Keyboard search action for query:
'$query'")

binding.spinnerOriginLanguage.setSel
ection(0, false)

binding.spinnerTranslationLanguage.s
etSelection(0, false)

viewModel.searchWord(query)
                } else {

viewModel.loadDictionary(viewModel.c
urrentOriginLangFilter,
viewModel.currentTranslationLangFilt
er)
                }
                true // Consume the
event
            } else {
                false
            }
        }
    }

    private fun
setupLanguageFilterSpinners() {
        val
onLanguageSelectedListener =
object :
AdapterView.OnItemSelectedListener {
            override fun
onItemSelected(parent:
```

```
AdapterView<*>?, view: View?,
position: Int, id: Long) {
                val originLang =
getSelectedLanguage(binding.spinnerO
riginLanguage)
                val translationLang
=
getSelectedLanguage(binding.spinnerT
ranslationLanguage)
                Log.d(TAG, "Language
filter changed. Origin: $originLang,
Translation: $translationLang.")

                if
(binding.searchInput.text.isNotBlank
()) {

binding.searchInput.setText("")
                    Log.d(TAG,
"Cleared search input due to filter
change.")
                }

viewModel.loadDictionary(originLang,
translationLang, forceReload =
false)
            }
            override fun
onNothingSelected(parent:
AdapterView<*>?) { /* No action
needed */ }
        }


binding.spinnerOriginLanguage.onItem
SelectedListener =
onLanguageSelectedListener

binding.spinnerTranslationLanguage.o
nItemSelectedListener =
onLanguageSelectedListener
    }

    private fun setupRefreshLayout()
{

binding.swipeRefresh.setOnRefreshLis
tener {
            Log.i(TAG, "Swipe to
refresh triggered.")

binding.searchInput.setText("")
            val originListener =
binding.spinnerOriginLanguage.onItem
SelectedListener
            val translationListener
=
binding.spinnerTranslationLanguage.o
nItemSelectedListener

binding.spinnerOriginLanguage.onItem
SelectedListener = null

binding.spinnerTranslationLanguage.o
nItemSelectedListener = null


binding.spinnerOriginLanguage.setSel
ection(0, false)

binding.spinnerTranslationLanguage.s
etSelection(0, false)


binding.spinnerOriginLanguage.onItem
SelectedListener = originListener

binding.spinnerTranslationLanguage.o
nItemSelectedListener =
translationListener


viewModel.loadDictionary(originLang
= null, translationLang = null,
forceReload = true)
        }
    }

    private fun
setupFabClickListener() {
        val fab =
activity?.findViewById<FloatingActio
nButton>(R.id.fab_add_word)
        fab?.setOnClickListener {
            Log.d(TAG, "FAB
(DictionaryFragment listener)
clicked, showing AddWordDialog.")

AddWordDialogFragment.newInstance().
show(childFragmentManager,
AddWordDialogFragment.TAG)
        }
    }

    private fun setupObservers() {
```

```
viewModel.displayedItems.observe(vie
wLifecycleOwner) { items ->
            Log.d(TAG,
"displayedItems observer: Updating
list with ${items.size} items.")

adapter.submitList(items)

binding.emptyView.visibility = if
(items.isEmpty()) View.VISIBLE else
View.GONE
        }


viewModel.loading.observe(viewLifecy
cleOwner) { isLoading ->
            Log.d(TAG, "Loading
state changed: $isLoading")

binding.progressBar.visibility = if
(isLoading) View.VISIBLE else
View.GONE
            if (!isLoading &&
binding.swipeRefresh.isRefreshing) {

binding.swipeRefresh.isRefreshing =
false
            }
        }


viewModel.error.observe(viewLifecycl
eOwner) { event ->

event.getContentIfNotHandled()?.let
{ errorMessage ->
                Log.e(TAG, "Error
message: $errorMessage")

Snackbar.make(binding.root,
errorMessage,
Snackbar.LENGTH_LONG).show()
            }
        }


viewModel.successMessage.observe(vie
wLifecycleOwner) { event ->

event.getContentIfNotHandled()?.let
{ successMessage ->
                Log.i(TAG, "Success
message: $successMessage")

Snackbar.make(binding.root,
successMessage,
Snackbar.LENGTH_SHORT).show()
            }
        }


viewModel.originLanguages.observe(vi
ewLifecycleOwner) { originLangs ->
            Log.d(TAG, "Origin
languages updated: $originLangs")
            val options =
listOf(getString(R.string.all_origin
_languages_filter)) + originLangs

updateSpinnerAdapter(binding.spinner
OriginLanguage, options,
viewModel.currentOriginLangFilter)
        }


viewModel.translationLanguages.obser
ve(viewLifecycleOwner)
{ translationLangs ->
            Log.d(TAG, "Translation
languages updated:
$translationLangs")
            val options =
listOf(getString(R.string.all_transl
ation_languages_filter)) +
translationLangs

updateSpinnerAdapter(binding.spinner
TranslationLanguage, options,
viewModel.currentTranslationLangFilt
er)
        }
    }
    //endregion

    //region UI Action/State Methods
    private fun showFab() {

mainActivity?.showDictionaryFab() //
Preferred way: MainActivity manages
its FAB's visibility

        // Fallback or explicit
control if MainActivity's method
doesn't cover all aspects
        // or if this fragment has
```

```
its own FAB.
        val fab =
activity?.findViewById<FloatingActio
nButton>(R.id.fab_add_word)
        fab?.visibility =
View.VISIBLE
        fab?.show()
    }

    private fun
confirmDeleteWord(word:
DictionaryWord) {
        Log.d(TAG, "Showing delete
confirmation for word:
${word.word}")

AlertDialog.Builder(requireContext()
)
            .setTitle(getString(R.st
ring.delete_word_dialog_title))
            .setMessage(getString(R.
string.delete_word_confirmation_mess
age, word.word))
            .setPositiveButton(getSt
ring(R.string.delete_action)) { _, _
->
                Log.i(TAG, "Deletion
confirmed for word: ${word.word}")

viewModel.deleteWord(word)
            }
            .setNegativeButton(getSt
ring(R.string.cancel_action), null)
            .show()
    }
    //endregion

    private fun
getSelectedLanguage(spinner:
Spinner): String? {
        val position =
spinner.selectedItemPosition
        return if (position == 0)
null else
spinner.selectedItem?.toString()
    }

    private fun
updateSpinnerAdapter(spinner:
Spinner, items: List<String>,
currentSelection: String?) {
        val oldListener =
spinner.onItemSelectedListener
spinner.onItemSelectedListener =
null

        val arrayAdapter =
ArrayAdapter(requireContext(),
android.R.layout.simple_spinner_item
, items).apply {

setDropDownViewResource(android.R.la
yout.simple_spinner_dropdown_item)
        }
        spinner.adapter =
arrayAdapter

        val selectionIndex =
items.indexOf(currentSelection).take
If { it != -1 } ?: 0

spinner.setSelection(selectionIndex,
false)


spinner.onItemSelectedListener =
oldListener
    }
}
```

DictionaryViewModel.kt

```
package
com.example.readingapp.core.ui.dicti
onary
import …

class
DictionaryViewModel(application:
Application) :
AndroidViewModel(application) {
    private companion object {
        private const val TAG =
"DictionaryViewModel"
    }

    private val apiService:
ApiService =
ApiClient.getApiService(application.
applicationContext)

    // LiveData for the list of
words to be displayed in the UI (can
be filtered list or search results)
    private val _displayedItems =
MutableLiveData<List<DictionaryWord>
>()
```

```
    val displayedItems:
LiveData<List<DictionaryWord>> =
_displayedItems

    // LiveData for loading state
    private val _loading =
MutableLiveData<Boolean>()
    val loading: LiveData<Boolean> =
_loading

    // LiveData for error messages,
wrapped in Event to ensure one-time
consumption
    private val _error =
MutableLiveData<Event<String>>()
    val error:
LiveData<Event<String>> = _error

    // LiveData for success
messages, wrapped in Event
    private val _successMessage =
MutableLiveData<Event<String>>()
    val successMessage:
LiveData<Event<String>> =
_successMessage

    // LiveData for available origin
languages for filtering
    private val _originLanguages =
MutableLiveData<List<String>>()
    val originLanguages:
LiveData<List<String>> get() =
_originLanguages

    // LiveData for available
translation languages for filtering
    private val
_translationLanguages =
MutableLiveData<List<String>>()
    val translationLanguages:
LiveData<List<String>> get() =
_translationLanguages

    // Stores all words fetched from
the API, used for client-side
filtering
    private var allFetchedWords:
List<DictionaryWord> = emptyList()
    private var
hasPerformedInitialLoad = false

    // Current filter states
    var currentOriginLangFilter:
String? = null
        private set
    var
currentTranslationLangFilter:
String? = null
        private set

    init {
        Log.i(TAG, "ViewModel
initialized, triggering initial
dictionary load.")
        loadDictionary(originLang =
null, translationLang = null,
forceReload = true) // Initial load
fetches all
    }

    fun loadDictionary(
        originLang: String? = null,
        translationLang: String? =
null,
        forceReload: Boolean = false
    ) {
        Log.d(TAG, "loadDictionary
called with originLang: $originLang,
translationLang: $translationLang,
forceReload: $forceReload. Current
filters: $currentOriginLangFilter,
$currentTranslationLangFilter")

        val sameFilters =
hasPerformedInitialLoad &&
                originLang ==
currentOriginLangFilter &&
                translationLang ==
currentTranslationLangFilter

        if (sameFilters
&& !forceReload) {
            Log.d(TAG, "Filters
unchanged and not forcing reload.
Skipping network/filter operation.")
            return
        }

        currentOriginLangFilter =
originLang
        currentTranslationLangFilter
= translationLang

        if (!hasPerformedInitialLoad
|| forceReload) {
            Log.i(TAG, "Starting to
```

```
fetch/re-fetch dictionary from
API.")
            _loading.value = true
            viewModelScope.launch {
                val errorMessage =
fetchAndProcessDictionaryData()
                if (errorMessage !=
null) {
                    _error.value =
Event(errorMessage)
                }
                // Always filter and
post, even if fetch failed (to post
emptyList) or succeeded.

filterAndPostWords(currentOriginLang
Filter,
currentTranslationLangFilter)
                _loading.value =
false
            }
        } else {
            // Data already loaded,
just apply filters locally
            Log.d(TAG, "Data already
loaded. Applying filters:
origin='$originLang',
translation='$translationLang'")
            _loading.value = true //
Show loading for filter operation if
it could be long

filterAndPostWords(originLang,
translationLang)
            _loading.value = false
        }
    }

    private suspend fun
fetchAndProcessDictionaryData():
String? {
        Log.d(TAG, "Attempting to
fetch dictionary from API.")
        return try {
            val response =
withContext(Dispatchers.IO) {

apiService.getDictionary(null)
            }

            if
(response.isSuccessful) {
                allFetchedWords =
response.body() ?: emptyList()

hasPerformedInitialLoad = true
                Log.i(TAG,
"Successfully fetched
${allFetchedWords.size} words from
API.")

                val originLangs =
allFetchedWords
                    .asSequence()
                    .mapNotNull
{ it.language?.trim()?.lowercase() }
                    .filter
{ it.isNotBlank() }
                    .distinct()
                    .sorted()
                    .toList()

                val translationLangs
= allFetchedWords
                    .asSequence()
                    .flatMap
{ it.translations.keys.asSequence()
}
                    .map
{ it.trim().lowercase() }
                    .filter
{ it.isNotBlank() }
                    .distinct()
                    .sorted()
                    .toList()


_originLanguages.postValue(originLan
gs)

_translationLanguages.postValue(tran
slationLangs)
                Log.d(TAG,
"Available origin languages:
$originLangs")
                Log.d(TAG,
"Available translation languages:
$translationLangs")
                null // No error
            } else {
                val errorMsg =
"Error loading dictionary:
${response.code()}
${response.message()}"
                Log.e(TAG, errorMsg)
                // Reset state on
```

```
error
                allFetchedWords =
emptyList()

_originLanguages.postValue(emptyList
())

_translationLanguages.postValue(empt
yList())
                errorMsg
            }
        } catch (e: Exception) {
            val errorMsg =
"Exception during dictionary fetch:
${e.localizedMessage ?: "Unknown
error"}"
            Log.e(TAG, errorMsg, e)
            allFetchedWords =
emptyList()

_originLanguages.postValue(emptyList
())

_translationLanguages.postValue(empt
yList())
            errorMsg
        }
    }

    private fun
filterAndPostWords(originLang:
String?, translationLang: String?) {
        Log.d(TAG, "Filtering words
with origin: '$originLang',
translation: '$translationLang'")
        val filteredList =
allFetchedWords.filter { word ->
            val originMatches =
originLang?.let {

word.language.equals(it, ignoreCase
= true)
            } ?: true // No origin
filter means all origin languages
match

            val translationMatches =
translationLang?.let {

word.translations.keys.any
{ transLang -> transLang.equals(it,
ignoreCase = true) }
            } ?: true // No
translation filter means all
translation languages match

            originMatches &&
translationMatches
        }
        _displayedItems.value =
filteredList
        Log.d(TAG, "Filtered words
count: ${filteredList.size}")
    }

    fun deleteWord(word:
DictionaryWord) {
        Log.i(TAG, "Attempting to
delete word: ${word.word} (ID:
${word.id})")
        _loading.value = true
        viewModelScope.launch {
            try {
                val response =
withContext(Dispatchers.IO)
{ apiService.deleteWord(word.id) }
                if
(response.isSuccessful) {
                    Log.i(TAG, "Word
deleted successfully from backend.
ID: ${word.id}")

_successMessage.value = Event("Word
'${word.word}' deleted.")
                    // Refresh the
entire dictionary to get latest
state & update language lists

loadDictionary(currentOriginLangFilt
er, currentTranslationLangFilter,
forceReload = true)
                } else {
                    val errorMsg =
"Failed to delete word:
${response.code()}
${response.message()}"
                    Log.e(TAG,
errorMsg)
                    _error.value =
Event(errorMsg)
                }
            } catch (e: Exception) {
                val errorMsg =
"Error deleting word:
${e.localizedMessage ?: "Unknown
error"}"
```

```
                Log.e(TAG, errorMsg,
e)
                _error.value =
Event(errorMsg)
            } finally {
                // Loading state
will be managed by the subsequent
loadDictionary call
                if (_loading.value
== true
&& !forceReloadNeededAfterMutation()
) {
                    _loading.value =
false
                }
            }
        }
    }

    fun updateWord(originalWord:
DictionaryWord, newWordText: String,
newTranslations: Map<String,
String>) {
        Log.i(TAG, "Attempting to
update word. ID: ${originalWord.id},
New Text: $newWordText")
        _loading.value = true
        viewModelScope.launch {
            try {
                val request =
SaveWordRequest(newWordText,
originalWord.language,
newTranslations)
                val response =
withContext(Dispatchers.IO)
{ apiService.updateWord(originalWord
.id, request) }

                if
(response.isSuccessful) {
                    Log.i(TAG, "Word
updated successfully. ID:
${originalWord.id}")

_successMessage.value = Event("Word
'${originalWord.word}' updated.")

loadDictionary(currentOriginLangFilt
er, currentTranslationLangFilter,
forceReload = true)
                } else {
                    val errorMsg =
"Failed to update word:
${response.code()}
${response.message()}"
                    Log.e(TAG,
errorMsg)
                    _error.value =
Event(errorMsg)
                }
            } catch (e: Exception) {
                val errorMsg =
"Error updating word:
${e.localizedMessage ?: "Unknown
error"}"
                Log.e(TAG, errorMsg,
e)
                _error.value =
Event(errorMsg)
            } finally {
                if (_loading.value
== true
&& !forceReloadNeededAfterMutation()
) {
                    _loading.value =
false
                }
            }
        }
    }

    fun addWord(word: String,
sourceLanguage: String,
translations: Map<String, String>) {
        Log.i(TAG, "Attempting to
add new word: $word, Language:
$sourceLanguage")
        _loading.value = true
        viewModelScope.launch {
            try {
                val request =
SaveWordRequest(word,
sourceLanguage, translations)
                val response =
withContext(Dispatchers.IO)
{ apiService.addWord(request) }

                if
(response.isSuccessful) {
                    Log.i(TAG, "Word
added successfully: $word")

_successMessage.value = Event("Word
'$word' added successfully.")

loadDictionary(currentOriginLangFilt
```

```
er, currentTranslationLangFilter,
forceReload = true)
                } else {
                    val errorMsg =
"Failed to add word:
${response.code()}
${response.message()}"
                    Log.e(TAG,
errorMsg)
                    _error.value =
Event(errorMsg)
                }
            } catch (e: Exception) {
                val errorMsg =
"Error adding word:
${e.localizedMessage ?: "Unknown
error"}"
                Log.e(TAG, errorMsg,
e)
                _error.value =
Event(errorMsg)
            } finally {
                if (_loading.value
== true
&& !forceReloadNeededAfterMutation()
) {
                    _loading.value =
false
                }
            }
        }
    }

    // Helper to check if
loadDictionary will be called, to
avoid premature _loading.value =
false
    private fun
forceReloadNeededAfterMutation():
Boolean = true // Mutations always
trigger forceReload

    fun searchWord(query: String) {
        if (query.isBlank()) {
            Log.w(TAG, "Search query
is blank. Clearing search results
and loading with current filters.")
            // Revert to showing the
filtered list based on current
filters

filterAndPostWords(currentOriginLang
Filter,
currentTranslationLangFilter)
            return
        }
        Log.i(TAG, "Searching
dictionary for query: '$query'")
        _loading.value = true
        viewModelScope.launch {
            try {
                Log.d(TAG,
"Executing API search for:
'$query'")
                val response =
withContext(Dispatchers.IO) {

apiService.searchDictionary(query)
                }

                if
(response.isSuccessful) {
                    val results =
response.body() ?: emptyList()
                    Log.i(TAG,
"Search successful. Found
${results.size} words for query:
'$query'.")

_displayedItems.value = results //
Update display with search results
                } else {
                    val errorMsg =
"Search API call failed:
${response.code()}
${response.message()}"
                    Log.e(TAG,
errorMsg)
                    _error.value =
Event(errorMsg)

_displayedItems.value = emptyList()
// Clear display on search error
                }
            } catch (e: Exception) {
                val errorMsg =
"Exception during search:
${e.localizedMessage ?: "Unknown
error"}"
                Log.e(TAG, errorMsg,
e)
                _error.value =
Event(errorMsg)

_displayedItems.value = emptyList()
// Clear display on exception
```

```
            } finally {
                _loading.value =
false
            }
        }
    }
}
```

PracticeHubFragmen.kt

```
package
com.example.readingapp.core.ui.pract
ice
import …

class PracticeHubFragment :
Fragment() {

    // --- Companion Object ---
    companion object {
        private const val TAG =
"PracticeHubFragment" // Tag for
logging
    }

    // --- ViewModel ---
    private lateinit var viewModel:
PracticeViewModel

    // --- UI Elements ---
    private lateinit var
buttonStartNewPractice: Button
    private lateinit var
recyclerViewRecentTests:
RecyclerView
    private lateinit var
textViewNoRecentTests: TextView
    // Spinners for sorting are
initialized and used within
setupSortingUI

    // --- Adapters ---
    private lateinit var
recentTestsAdapter:
RecentTestsAdapter

    // --- Fragment Lifecycle
Methods ---

    override fun onCreateView(
        inflater: LayoutInflater,
container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        Log.d(TAG, "onCreateView:
Inflating layout
R.layout.fragment_practice_hub")
        // Inflate the layout for
this fragment
        return
inflater.inflate(R.layout.fragment_p
ractice_hub, container, false)
    }

    override fun onViewCreated(view:
View, savedInstanceState: Bundle?) {
        super.onViewCreated(view,
savedInstanceState)
        Log.d(TAG, "onViewCreated:
View created, initializing ViewModel
and UI.")

        // Initialize ViewModel
        viewModel =
ViewModelProvider(requireActivity())
[PracticeViewModel::class.java]

        // Initialize UI elements
        buttonStartNewPractice =
view.findViewById(R.id.buttonStartNe
wPractice)
        recyclerViewRecentTests =
view.findViewById(R.id.recyclerViewR
ecentTests)
        textViewNoRecentTests =
view.findViewById(R.id.textViewNoRec
entTests)

        // Setup UI components and
observers
        setupRecyclerView()
        setupObservers()
        setupSortingUI(view) // For
sorting test history
        setupLanguageFilters(view)
// For filtering test history by
language pair

        // Setup button click
listener

buttonStartNewPractice.setOnClickLis
tener {
            Log.i(TAG, "Start New
Practice button clicked.")
```

```
showConfigureTestDialog()
        }

        // Load initial data
        Log.i(TAG, "Requesting
initial load of recent test
scores.")

viewModel.loadRecentTestScores()
    }

    // --- Setup Methods ---

    private fun setupRecyclerView()
{
        Log.d(TAG,
"setupRecyclerView: Initializing
RecyclerView and adapter.")
        recentTestsAdapter =
RecentTestsAdapter(
            emptyList(),
            onItemClick =
{ testScore ->
                Log.i(TAG, "Recent
test item clicked: ID
${testScore.id}")

loadTestDetails(testScore)
            },
            onDeleteClick =
{ testScore ->
                Log.i(TAG, "Delete
button clicked for test item: ID
${testScore.id}")

confirmDeleteTestScore(testScore)
            }
        )

recyclerViewRecentTests.layoutManage
r =
LinearLayoutManager(requireContext()
)

recyclerViewRecentTests.adapter =
recentTestsAdapter

        // Observe filtered test
scores (which includes sorting)

viewModel.filteredTestScores.observe
(viewLifecycleOwner) { scores ->
            Log.d(TAG,
"filteredTestScores observer in
setupRecyclerView: Updating adapter
with ${scores.size} scores.")

recentTestsAdapter.updateData(scores
)
        }
    }

    private fun setupObservers() {
        Log.d(TAG, "setupObservers:
Setting up LiveData observers.")

        // Observer for errors when
loading recent scores

viewModel.errorRecentScores.observe(
viewLifecycleOwner) { error ->
            error?.let {
                Log.e(TAG, "Error
loading recent scores: $it")

Toast.makeText(requireContext(), it,
Toast.LENGTH_LONG).show()

textViewNoRecentTests.visibility =
View.VISIBLE

recyclerViewRecentTests.visibility =
View.GONE
            }
        }

        // Observer for the list of
filtered test scores to manage UI
visibility

viewModel.filteredTestScores.observe
(viewLifecycleOwner) { scores ->
            if
(scores.isNullOrEmpty()) {
                Log.i(TAG,
"filteredTestScores observer: No
scores to display. Showing 'no
recent tests' message.")

recyclerViewRecentTests.visibility =
View.GONE

textViewNoRecentTests.visibility =
View.VISIBLE
            } else {
                Log.i(TAG,
```

```
"filteredTestScores observer:
Displaying ${scores.size} scores.
Showing RecyclerView.")

recyclerViewRecentTests.visibility =
View.VISIBLE

textViewNoRecentTests.visibility =
View.GONE
            }
        }
    }

    private fun setupSortingUI(view:
View) {
        Log.d(TAG, "setupSortingUI:
Initializing sorting spinners.")
        val sortByOptions =
listOf("Date", "Score", "Questions")
// User-friendly display names
        val sortOrderOptions =
listOf("Descending", "Ascending") //
User-friendly display names

        val spinnerSortBy: Spinner =
view.findViewById(R.id.spinnerSortBy
)
        val spinnerOrder: Spinner =
view.findViewById(R.id.spinnerSortOr
der)

        // Adapter for "Sort By"
spinner
        val sortByAdapter =
ArrayAdapter(requireContext(),
android.R.layout.simple_spinner_item
, sortByOptions)

sortByAdapter.setDropDownViewResourc
e(android.R.layout.simple_list_item_
1) // Standard dropdown item layout
        spinnerSortBy.adapter =
sortByAdapter

        // Adapter for "Sort Order"
spinner
        val sortOrderAdapter =
ArrayAdapter(requireContext(),
android.R.layout.simple_spinner_item
, sortOrderOptions)

sortOrderAdapter.setDropDownViewReso
urce(android.R.layout.simple_list_it
em_1) // Standard dropdown item
layout
        spinnerOrder.adapter =
sortOrderAdapter

        // Listener for "Sort By"
selection changes

spinnerSortBy.onItemSelectedListener
= object :
AdapterView.OnItemSelectedListener {
            override fun
onItemSelected(parent:
AdapterView<*>, view: View?,
position: Int, id: Long) {
                val value = when
(position) {
                    0 -> "date"
                    1 -> "score"
                    2 -> "questions"
                    else -> "date"
                }
                Log.d(TAG, "Sort By
selection changed to: $value
(Position: $position)")

viewModel.setSortBy(value)
            }
            override fun
onNothingSelected(parent:
AdapterView<*>) {
                // Interface method,
nothing to do here for this app
            }
        }

        // Listener for "Sort Order"
selection changes

spinnerOrder.onItemSelectedListener
= object :
AdapterView.OnItemSelectedListener {
            override fun
onItemSelected(parent:
AdapterView<*>, view: View?,
position: Int, id: Long) {
                val value = if
(position == 0) "desc" else "asc" //
0 for "Descending", 1 for
"Ascending"
                Log.d(TAG, "Sort
Order selection changed to: $value
(Position: $position)")
```

```
viewModel.setSortOrder(value)
            }
            override fun
onNothingSelected(parent:
AdapterView<*>) {
                // Interface method,
nothing to do here for this app
            }
        }
    }

    private fun
setupLanguageFilters(view: View) {
        Log.d(TAG,
"setupLanguageFilters: Initializing
language filter checkboxes.")
        val languageContainer =
view.findViewById<FlexboxLayout>(R.i
d.languageCheckboxContainer)


viewModel.allLanguagePairs.observe(v
iewLifecycleOwner) { uniquePairs ->
            Log.d(TAG,
"allLanguagePairs observer: Received
${uniquePairs.size} unique pairs.
Updating filter UI.")

            if
(uniquePairs.isEmpty() &&
languageContainer.isEmpty()) {
                Log.d(TAG, "No
unique language pairs available, and
container is already empty.")
                return@observe
            }

            val
currentlySelectedPairs =
viewModel.selectedLanguagePairs.valu
e ?: emptyList()
            val
isCurrentlyAllSelected =
currentlySelectedPairs.isEmpty()
            Log.d(TAG, "Current
selected pairs for filter:
$currentlySelectedPairs, Is 'All'
selected: $isCurrentlyAllSelected")


languageContainer.removeAllViews()

            val allCheckBox =
CheckBox(requireContext()).apply {
                text =
getString(R.string.filter_all_langua
ges)
                isChecked =
isCurrentlyAllSelected
                layoutParams =
FlexboxLayout.LayoutParams(

ViewGroup.LayoutParams.WRAP_CONTENT,

ViewGroup.LayoutParams.WRAP_CONTENT
                ).apply {
                    setMargins(8, 8,
8, 8)
                }
            }

languageContainer.addView(allCheckBo
x)

            val otherCheckBoxes =
mutableListOf<CheckBox>()

            // Central function to
update ViewModel based on current
checkbox states
            fun
updateViewModelWithSelectedPairs() {
                val
selectedFromCheckboxes =
otherCheckBoxes.filter
{ it.isChecked }.map
{ it.text.toString() }

                // If "All" is
checked OR (no specific items are
checked AND "All" isn't focused
trying to be unchecked)
                if
(allCheckBox.isChecked ||
selectedFromCheckboxes.isEmpty()) {
                    // Ensure "All"
checkbox reflects this state if it
wasn't the trigger
                    if
(!allCheckBox.isChecked) {

allCheckBox.isChecked = true
                    }
                    var
changedOthers = false
```

```
otherCheckBoxes.forEach {
                        if
(it.isChecked) {

it.isChecked = false; changedOthers
= true
                        }
                    }

if
(viewModel.selectedLanguagePairs.val
ue?.isNotEmpty() == true ||
changedOthers ||
selectedFromCheckboxes.isNotEmpty())
{

viewModel.setSelectedLanguagePairs(e
mptyList())
                    } else if
(viewModel.selectedLanguagePairs.val
ue == null) { // Initial state

viewModel.setSelectedLanguagePairs(e
mptyList())
                    }
                } else {
                    if
(allCheckBox.isChecked) {

allCheckBox.isChecked = false
                    }

viewModel.setSelectedLanguagePairs(s
electedFromCheckboxes)
                }
            }

allCheckBox.setOnCheckedChangeListen
er { _, isChecked ->
                Log.d(TAG,
"AllCheckBox listener triggered.
isChecked: $isChecked")
                // If "All" is
checked by the user, it will ensure
others are unchecked via
updateViewModelWithSelectedPairs.

updateViewModelWithSelectedPairs
will re-check "All".

updateViewModelWithSelectedPairs()
            }
```

```
            uniquePairs.forEach
{ pair ->
                val checkBox =
CheckBox(requireContext()).apply {
                    text = pair
                    isChecked =
currentlySelectedPairs.contains(pair
)
                    layoutParams =
FlexboxLayout.LayoutParams(

ViewGroup.LayoutParams.WRAP_CONTENT,

ViewGroup.LayoutParams.WRAP_CONTENT
                    ).apply {

setMargins(8, 8, 8, 8)
                    }
                }


checkBox.setOnCheckedChangeListener
{ _, isNowChecked ->
                    Log.d(TAG,
"Specific checkbox '$pair' listener.
isNowChecked: $isNowChecked")
                    var
allStateChangedProgrammatically =
false
                    if
(isNowChecked) {
                        // If a
specific checkbox is checked, "All"
must be unchecked.
                        if
(allCheckBox.isChecked) {

allCheckBox.isChecked = false //
Programmatically uncheck "All".

allStateChangedProgrammatically =
true
                        }
                    } else {
                        if
(otherCheckBoxes.none
{ it.isChecked }) {
                            if
(!allCheckBox.isChecked) {

allCheckBox.isChecked = true //
Programmatically check "All".
```

```
allStateChangedProgrammatically =
true
                                }
                            }
                        }
                        if
(!allStateChangedProgrammatically) {

updateViewModelWithSelectedPairs()
                        }
                    }

otherCheckBoxes.add(checkBox)

languageContainer.addView(checkBox)
            }
            Log.d(TAG, "Finished
adding ${otherCheckBoxes.size + 1}
checkboxes to language filter
container.")
        }
    }

    // --- UI Interaction &
Navigation ---

    private fun
showConfigureTestDialog() {
        Log.d(TAG,
"showConfigureTestDialog: Displaying
ConfigureTestDialogFragment.")

viewModel.originLanguages.removeObse
rvers(viewLifecycleOwner)

viewModel.translationLanguages.remov
eObservers(viewLifecycleOwner)

        val dialogFragment =
ConfigureTestDialogFragment.newInsta
nce()

dialogFragment.show(parentFragmentMa
nager, "configure_test_dialog")
    }

    private fun
loadTestDetails(testScore:
TestScore) {
        Log.i(TAG, "loadTestDetails:
Loading details for test ID
${testScore.id}")
        lifecycleScope.launch { //
Use lifecycleScope for coroutines
tied to the fragment's lifecycle
            try {
                val testDetails =
viewModel.getTestDetails(testScore.i
d)
                if (testDetails !=
null) {
                    Log.i(TAG,
"Successfully loaded
${testDetails.size} answer items for
test ID ${testScore.id}.")

showTestSummary(testScore,
testDetails)
                } else {
                    Log.w(TAG, "No
details available or error fetching
details for test ID
${testScore.id}.")

Toast.makeText(context,
getString(R.string.no_test_details_a
vailable),
Toast.LENGTH_SHORT).show()
                }
            } catch (e: Exception) {
                Log.e(TAG, "Error
loading test details for test ID
${testScore.id}", e)

Toast.makeText(context,
getString(R.string.error_loading_tes
t_details, e.message),
Toast.LENGTH_SHORT).show()
            }
        }
    }

    private fun
showTestSummary(testScore:
TestScore, answers:
List<AnswerItem>) {
        Log.d(TAG, "showTestSummary:
Displaying TestSummaryFragment for
past test ID ${testScore.id}.")
        val summaryFragment =
TestSummaryFragment.newInstanceForPa
stTest(
            testScore = testScore,
            answers = answers
        )
```

```
summaryFragment.show(parentFragmentM
anager, "past_test_summary")
    }

    private fun
confirmDeleteTestScore(testScore:
TestScore) {
        Log.d(TAG,
"confirmDeleteTestScore: Showing
delete confirmation dialog for test
score ID: ${testScore.id}")


AlertDialog.Builder(requireContext()
)
            .setTitle(getString(R.st
ring.delete_test_score_dialog_title)
)
            .setMessage(getString(R.
string.delete_test_score_confirmatio
n_message, testScore.score))
            .setPositiveButton(getSt
ring(R.string.delete_action)) { _, _
->
                Log.i(TAG, "Deletion
confirmed for test score ID:
${testScore.id}. Calling
ViewModel.")

viewModel.deleteTestScore(testScore.
id)
            }
            .setNegativeButton(getSt
ring(R.string.cancel_action))
{ dialog, _ ->
                Log.d(TAG, "Deletion
cancelled for test score ID:
${testScore.id}.")
                dialog.dismiss()
            }
            .show()
    }
}
```

PracticeViewModel.kt

```
package
com.example.readingapp.core.ui.pract
ice
import …

class PracticeViewModel(application:
Application) :
AndroidViewModel(application) {

    // --- Network Service ---
    private val apiService:
ApiService =
ApiClient.getApiService(application.
applicationContext)

    // --- Internal State for
Current Test Session ---
        private val
_questionsInternal =
mutableListOf<PracticeQuestion>()
    private val _results =
mutableListOf<AnswerItem>()
    private var currentIndex = 0

    // --- LiveData for Current Test
Session UI Updates ---
    private val _isLoading =
MutableLiveData<Boolean>()
    val isLoading: LiveData<Boolean>
= _isLoading
    private val _questionsLiveData =
MutableLiveData<List<PracticeQuestio
n>>()
    val questions:
LiveData<List<PracticeQuestion>> =
_questionsLiveData
    private val
_currentQuestionLiveData =
MutableLiveData<PracticeQuestion?>()
    val currentQuestionLiveData:
LiveData<PracticeQuestion?> =
_currentQuestionLiveData
        private val _feedback =
MutableLiveData<SubmitAnswerResponse
?>()
    val feedback:
LiveData<SubmitAnswerResponse?> =
_feedback
    private val _testFinished =
MutableLiveData<FinishTestResponse?>
()
    val testFinished:
LiveData<FinishTestResponse?> =
_testFinished
    private val _isLastQuestion =
MutableLiveData<Boolean>()
    val isLastQuestion:
LiveData<Boolean> get() =
_isLastQuestion
```

```
    // --- LiveData for Language
Options UI Updates ---
    val originLanguages =
MutableLiveData<List<String>>()
    val translationLanguages =
MutableLiveData<List<String>>()

    // --- Internal State for Test
History Filtering & Sorting ---
        private val _allTests =
MutableLiveData<List<TestScore>>()
    private val _sortBy =
MutableLiveData("date")
    private val _sortOrder =
MutableLiveData("desc")
    private val
_selectedLanguagePairs =
MutableLiveData<List<String>>(emptyL
ist())

    // --- LiveData for Test History
UI Updates ---
    private val _recentTestScores =
MutableLiveData<List<TestScore>>()
        val recentTestScores:
LiveData<List<TestScore>> =
_recentTestScores
    private val
_isLoadingRecentScores =
MutableLiveData<Boolean>()
        val isLoadingRecentScores:
LiveData<Boolean> =
_isLoadingRecentScores
        private val
_errorRecentScores =
MutableLiveData<String?>()
    val errorRecentScores:
LiveData<String?> =
_errorRecentScores
    private val _allLanguagePairs =
MutableLiveData<List<String>>(emptyL
ist())
    val allLanguagePairs:
LiveData<List<String>> =
_allLanguagePairs
    val selectedLanguagePairs:
LiveData<List<String>> =
_selectedLanguagePairs
    private val _filteredTests =
MutableLiveData<List<TestScore>>()
    val filteredTestScores:
LiveData<List<TestScore>> =
_filteredTests

    // --- Computed Properties
(Getters for current test state) ---
    val currentQuestion:
PracticeQuestion?
        get() =
_questionsInternal.getOrNull(current
Index)
    val currentQuestionList:
List<PracticeQuestion> get() =
_questionsInternal
    val allAnswers: List<AnswerItem>
get() = _results.toList()


    // --- Public Methods -
Initialization & Configuration ---

    fun fetchLanguageOptions() {
        Log.d("PracticeViewModel",
"Fetching language options...")
        viewModelScope.launch {
            try {
                val response =
apiService.getAvailableLanguages()
                if
(response.isSuccessful) {

response.body()?.let {

originLanguages.value =
it.originLanguages

translationLanguages.value =
it.translationLanguages

Log.i("PracticeViewModel", "Language
options fetched successfully.
Origins: ${it.originLanguages.size},
Translations:
${it.translationLanguages.size}")
                    }
                } else {
                    val errorMsg =
"Error fetching language options:
${response.errorBody()?.string()}"

Log.e("PracticeViewModel", errorMsg)
                }
            } catch (e: Exception) {

Log.e("PracticeViewModel",
```

```
"Exception fetching language
options", e)
            }
        }
    }

    fun estimateQuestionCount(
        originLangs: List<String>,
        translationLangs:
List<String>,
        onResult: (Int?) -> Unit
    ) {
        Log.d("PracticeViewModel",
"Estimating question count for
origins: $originLangs, translations:
$translationLangs")
        viewModelScope.launch {
            try {
                val response =
apiService.estimateQuestionCount(

EstimateQuestionsRequest(originLangs
, translationLangs)
                )
                if
(response.isSuccessful) {
                    val count =
response.body()?.count

Log.i("PracticeViewModel",
"Estimated question count: $count")
                    onResult(count)
                } else {

Log.e("PracticeViewModel", "Failed
to fetch estimated question count:
${response.errorBody()?.string()}")
                    onResult(null)
                }
            } catch (e: Exception) {

Log.e("PracticeViewModel",
"Exception estimating question
count: $e", e)
                onResult(null)
            }
        }
    }

    // --- Public Methods - Test
Lifecycle Management ---

    fun
startPracticeTest(originLangs:
List<String>, translationLangs:
List<String>, max: Int) {
        Log.i("PracticeViewModel",
"Starting practice test. Origins:
$originLangs, Translations:
$translationLangs, Max questions:
$max")
        _isLoading.value = true
        // Reset test state
        _questionsInternal.clear()
        currentIndex = 0

_currentQuestionLiveData.value =
null // Initially null
        _feedback.value = null
        _results.clear()
        _testFinished.value = null
        _isLastQuestion.value =
false // Default to false, will be
updated

        viewModelScope.launch {
            try {
                val request =
StartTestRequest(originLangs,
translationLangs, max)

Log.d("PracticeViewModel",
"Requesting test start from API.")
                val response =
apiService.startTest(request)

                if
(response.isSuccessful) {
                    val questions =
response.body() ?: emptyList()

Log.i("PracticeViewModel", "API
success - received ${questions.size}
questions for the test.")

processQuestions(questions)
                } else {
                    val errorBody =
response.errorBody()?.string() ?:
"Unknown API error"

processQuestions(emptyList()) //
Process empty list to reset UI
correctly
                }
            } catch (e: Exception) {
```

```
processQuestions(emptyList()) //
Process empty list on exception
            } finally {
                _isLoading.value =
false

Log.d("PracticeViewModel", "Finished
loading test start sequence.")
            }
        }
    }

    fun submitAnswer(userAnswer:
String) {
        currentQuestion?.let
{ question ->

Log.d("PracticeViewModel",
"Submitting answer: '$userAnswer'
for question ID:
${question.word_id}")
            viewModelScope.launch {
                try {
                    val response =
apiService.submitAnswer(

SubmitAnswerRequest(question.correct
_answer, userAnswer)
                    )
                    if
(response.isSuccessful) {

response.body()?.let
{ feedbackResponse ->

_feedback.value = feedbackResponse

_results.add(

AnswerItem(

word_id = question.word_id,

origin_lang = question.origin_lang,

translation_lang =
question.translation_lang,

word = question.word,

user_answer = userAnswer,
correct_answer =
question.correct_answer

// is_correct will be set by backend
or derived if needed later
                                    )
                                )

Log.i("PracticeViewModel", "Answer
submitted.")
                        }
                    } else {

Log.e("PracticeViewModel", "API
error submitting answer:
${response.errorBody()?.string()}")

_feedback.value = null // Indicate
error or no feedback
                    }
                } catch (e:
Exception) {

Log.e("PracticeViewModel",
"Exception submitting answer", e)
                    _feedback.value
= null // Indicate error or no
feedback
                }
            }
        } ?:
Log.w("PracticeViewModel",
"SubmitAnswer called but
currentQuestion is null.")
    }

    fun goToNextQuestion() {
        Log.d("PracticeViewModel",
"goToNextQuestion called. Current
index: $currentIndex, Total
questions:
${_questionsInternal.size}")
        if (currentIndex <
_questionsInternal.size - 1) {
            currentIndex++
            _feedback.value = null
// Reset feedback for the new
question

_currentQuestionLiveData.value =
_questionsInternal.getOrNull(current
Index)
            _isLastQuestion.value =
```

```
currentIndex ==
_questionsInternal.size - 1

Log.d("PracticeViewModel", "Moved to
next question. New index:
$currentIndex. Is last:
${_isLastQuestion.value}")
        } else {
            // This was the last
question or list is empty
            if
(_questionsInternal.isNotEmpty()) {

_isLastQuestion.value = true

Log.i("PracticeViewModel", "Last
question answered. Attempting to
finish test.")
                finishTest()
            } else {
                // Should not happen
if test started correctly, but
handle defensively

_currentQuestionLiveData.value =
null

_isLastQuestion.value = true

Log.w("PracticeViewModel",
"goToNextQuestion called but no
questions available. Test might not
have started or finished.")
            }
        }
    }

    fun finishTest() {
        Log.i("PracticeViewModel",
"finishTest called. Number of
results to submit:
${_results.size}")
        viewModelScope.launch {
            try {
                if
(_results.isNotEmpty()) {
                    val response =
apiService.finishTest(FinishTestRequ
est(_results))
                    if
(response.isSuccessful) {

response.body()?.let { result ->
                            val
resultWithPercentage = if
(result.percentage == null &&
result.total_questions > 0) {
                                val
calculatedPercentage =
(result.score.toDouble() * 100 /
result.total_questions).toInt()

Log.d("PracticeViewModel",
"Calculated percentage:
$calculatedPercentage for score
${result.score}/${result.total_quest
ions}")

result.copy(percentage =
calculatedPercentage)
                            } else {

result
                            }

_testFinished.value =
resultWithPercentage

Log.i("PracticeViewModel", "Test
finished successfully. Score:
${resultWithPercentage.score}/${resu
ltWithPercentage.total_questions},
Percentage:
${resultWithPercentage.percentage}")
                        }
                    } else {

Log.e("PracticeViewModel", "API
error finishing test:
${response.errorBody()?.string()}")

_testFinished.value = null //
Indicate error
                    }
                } else {

Log.w("PracticeViewModel",
"finishTest called but no results to
submit. Setting _testFinished to
null.")

_testFinished.value = null // No
results to submit, so no test to
finish.
                }
            } catch (e: Exception) {
```

```
Log.e("PracticeViewModel",
"Exception finishing test", e)
                _testFinished.value
= null // Indicate error
            }
        }
    }

    // --- Public Methods - Test
Score History Management ---

    fun loadRecentTestScores(limit:
Int = 10) { // Limit parameter is
not used in current
apiService.getLastTestScores() call
        Log.i("PracticeViewModel",
"Loading recent test scores (limit:
$limit - parameter currently unused
by API call)")
        _isLoadingRecentScores.value
= true
        _errorRecentScores.value =
null // Clear previous errors
        viewModelScope.launch {
            try {
                // Redundant
assignment, but kept to match
original structure if it had a
purpose.
                //
_isLoadingRecentScores.value = true
                //
_errorRecentScores.value = null

                val response =
apiService.getLastTestScores()
                if
(response.isSuccessful) {
                    val tests =
response.body() ?: emptyList()

_recentTestScores.value = tests //
Update the "raw" recent scores

setAllTests(tests)  // Update the
comprehensive list used for
filtering

Log.i("PracticeViewModel", "Recent
test scores loaded successfully.
Count: ${tests.size}")
                } else {
                    val errorMsg =
"Server error loading recent scores:
${response.code()}"

_errorRecentScores.value = errorMsg

_recentTestScores.value =
emptyList()

setAllTests(emptyList()) // Ensure a
consistent empty state

Log.e("PracticeViewModel", errorMsg)
                }
            } catch (e: Exception) {
                val errorMsg =
"Failed to load recent tests:
${e.message}"

_errorRecentScores.value = errorMsg

_recentTestScores.value =
emptyList()

setAllTests(emptyList())  // Ensure
a consistent empty state

Log.e("PracticeViewModel", errorMsg,
e)
            } finally {

_isLoadingRecentScores.value = false

Log.d("PracticeViewModel", "Finished
loading recent test scores
sequence.")
            }
        }
    }

    fun deleteTestScore(testId: Int)
{
        Log.i("PracticeViewModel",
"Attempting to delete test score
with ID: $testId")
        viewModelScope.launch {
            try {
                val response =
apiService.deleteTestScore(testId)
                if
(response.isSuccessful) {

Log.i("PracticeViewModel", "Test
```

```
score ID $testId deleted
successfully from server.")
                    // Update local
lists
                    val updatedList
= _allTests.value?.filter { it.id !=
testId } ?: emptyList()

setAllTests(updatedList) // This
will also refresh _filteredTests and
_recentTestScores via its own logic
or by direct update

_recentTestScores.value =
_recentTestScores.value?.filter
{ it.id != testId } ?: emptyList()
// Keep raw recent scores in sync
too

                } else {
                    val errorMsg =
"Failed to delete test: Server error
${response.code()}"

_errorRecentScores.value = errorMsg
// Use the general error LiveData
for recent scores

Log.e("PracticeViewModel",
"$errorMsg - Response:
${response.errorBody()?.string()}")
                }
            } catch (e: Exception) {
                val errorMsg =
"Failed to delete test:
${e.message}"

_errorRecentScores.value = errorMsg

Log.e("PracticeViewModel", errorMsg,
e)
            }
        }
    }

    suspend fun
getTestDetails(testId: Int):
List<AnswerItem>? = coroutineScope {
        Log.d("PracticeViewModel",
"Fetching test details for test ID:
$testId")
        try {
            val response =
apiService.getTestResults(testId)
            if
(response.isSuccessful) {
                val testResults =
response.body() ?:
return@coroutineScope null

                // For each test
result, launch an async coroutine to
fetch the word
                val deferredAnswers
= testResults.map { answerData ->
                    async {
                        val word =
getWordById(answerData.word_id)
                        AnswerItem(
                            word_id
= answerData.word_id,

origin_lang =
answerData.origin_lang,

translation_lang =
answerData.translation_lang,
                            word =
word, // Fetched word

user_answer =
answerData.user_answer,

correct_answer =
answerData.correct_answer,

is_correct = answerData.is_correct
                        )
                    }
                }
                val detailedAnswers
= deferredAnswers.awaitAll()

return@coroutineScope
detailedAnswers
            } else {

Log.e("PracticeViewModel", "Failed
to fetch test results for ID
$testId. Response code:
${response.code()}, Error:
${response.errorBody()?.string()}")

return@coroutineScope null
            }
        } catch (e: Exception) {
```

```
            return@coroutineScope
null
        }
    }

    suspend fun getWordById(wordId:
Int): String? {
        Log.d("PracticeViewModel",
"Fetching word by ID: $wordId")
        return try {
            val response =
apiService.getWord(wordId)
            if
(response.isSuccessful) {
                val word =
response.body()?.word

Log.d("PracticeViewModel", "Word
fetched for ID $wordId: $word")
                word
            } else {

Log.w("PracticeViewModel", "Failed
to fetch word for ID $wordId.
Response code: ${response.code()}")
                null
            }
        } catch (e: Exception) {

Log.e("PracticeViewModel",
"Exception fetching word for ID
$wordId", e)
            null
        }
    }


    // --- Public Methods -
Filtering and Sorting Test Scores --
-

    fun setSortBy(field: String) {
        Log.d("PracticeViewModel",
"Setting sort by: $field")
        _sortBy.value = field
        refreshFilteredTests()
    }

    fun setSortOrder(order: String)
{
        Log.d("PracticeViewModel",
"Setting sort order: $order")
        _sortOrder.value = order
        refreshFilteredTests()
    }

    fun
setSelectedLanguagePairs(pairs:
List<String>) {
        Log.d("PracticeViewModel",
"Setting selected language pairs:
$pairs")
        _selectedLanguagePairs.value
= pairs
        refreshFilteredTests()
    }

    // --- Private/Helper Methods --
-
    private fun
processQuestions(questions:
List<PracticeQuestion>) {
        Log.d("PracticeViewModel",
"Processing ${questions.size}
questions.")
        _questionsInternal.clear()

_questionsInternal.addAll(questions)
        _questionsLiveData.value =
_questionsInternal.toList() //
Notify observers with a new list

        currentIndex = 0 // Reset
index

_currentQuestionLiveData.value =
_questionsInternal.getOrNull(current
Index)
        _isLastQuestion.value =
_questionsInternal.size <= 1 &&
_questionsInternal.isNotEmpty() //
Correctly set for 0 or 1 question

        // Reset other states that
depend on new questions
        _feedback.value = null
        _results.clear() // Clear
previous answers

        if
(_questionsInternal.isEmpty()) {
            // UI should handle this
state, e.g., by showing a message.
        } else {

Log.d("PracticeViewModel", "First
```

```
question set. Total questions:
${_questionsInternal.size}. Is last:
${_isLastQuestion.value}")
        }
    }

    fun isLastQuestionAnswered():
Boolean {
        return allAnswers.size ==
currentQuestionList.size - 1
    }

    fun setAllTests(tests:
List<TestScore>) { // Changed to
public as it's called from
loadRecentTestScores and
deleteTestScore
        Log.d("PracticeViewModel",
"Setting all tests. Count:
${tests.size}")
        _allTests.value = tests

        // Extract all unique
language pairs from ALL tests
        val allPairs = tests.flatMap
{ test ->

test.language_pairs_stats.map { stat
-> stat.pair }
        }.toSet().sorted()
        _allLanguagePairs.value =
allPairs
        Log.d("PracticeViewModel",
"Extracted ${allPairs.size} unique
language pairs: $allPairs")

        // Now refresh filtered
tests based on the new 'allTests'
list and current filters
        refreshFilteredTests()
    }

        fun refreshFilteredTests()
{ // Changed to public as it might
be useful to call externally after
multiple filter changes
        val currentAllTests =
_allTests.value ?: emptyList()
        val currentSortBy =
_sortBy.value ?: "date"
        val currentSortOrder =
_sortOrder.value ?: "desc"
        val currentSelectedPairs =
_selectedLanguagePairs.value ?:
emptyList()

        Log.d("PracticeViewModel",
"Refreshing filtered tests. " +
                "Total available:
${currentAllTests.size}, " +
                "SortBy:
$currentSortBy, SortOrder:
$currentSortOrder, " +
                "SelectedPairs:
$currentSelectedPairs")

        val sortedAndFiltered =
filterAndSortTests(
            currentAllTests,
            currentSortBy,
            currentSortOrder,
            currentSelectedPairs
        )
        _filteredTests.value =
sortedAndFiltered
        Log.i("PracticeViewModel",
"Filtered tests updated. New count:
${sortedAndFiltered.size}")
    }

    fun filterAndSortTests( //
Changed to public as it's a pure
function and could be useful
        tests: List<TestScore>,
        sortBy: String = "date",
        sortOrder: String = "desc",
        selectedLanguagePairs:
List<String> = emptyList()
    ): List<TestScore> {
        Log.d("PracticeViewModel",
"filterAndSortTests called. Input
tests: ${tests.size}, sortBy:
$sortBy, sortOrder: $sortOrder,
selectedPairs:
$selectedLanguagePairs")

        // Step 1: Filter by
selected language pairs
        val filtered = if
(selectedLanguagePairs.isEmpty()) {

Log.d("PracticeViewModel", "No
language pairs selected, using all
${tests.size} tests for sorting.")
            tests // No language
filter applied
```

```
        } else {
            tests.filter { test ->

test.language_pairs_stats.any
{ langPairStat ->

selectedLanguagePairs.contains(langP
airStat.pair)
                }
            }.also {

Log.d("PracticeViewModel", "Filtered
by language pairs. ${it.size} tests
matched.")
            }
        }

        // Step 2: Create comparator
based on sortBy field
        val comparator = when
(sortBy) {
            "score" ->
compareBy<TestScore> {
                // Calculate
percentage score for comparison
                if
(it.total_questions == 0) 0.0 else
it.score.toDouble() /
it.total_questions
            }
            "questions" ->
compareBy<TestScore>
{ it.total_questions }
            "date" ->
compareBy<TestScore>
{ it.created_at }
            else -> {
                compareBy<TestScore>
{ it.created_at }
            }
        }

        // Step 3: Apply sort order
(ascending or descending)
        val finalComparator = if
(sortOrder.equals("asc", ignoreCase
= true)) {
            comparator
        } else {
            comparator.reversed() //
Default to descending if not "asc"
        }

        // Step 4: Sort the filtered
list
        return
filtered.sortedWith(finalComparator)
.also {

Log.d("PracticeViewModel", "Sorting
complete. Final list size:
${it.size}")
        }
    }
}
```

PracticeActivity.kt

```
package
com.example.readingapp.features.prac
tice
import …

class PracticeActivity :
AppCompatActivity() {

    // ViewModel
    private lateinit var viewModel:
PracticeViewModel

    // UI Elements
    private lateinit var
questionText: TextView
    private lateinit var editAnswer:
EditText
    private lateinit var
submitButton: Button
    private lateinit var
feedbackText: TextView
    private lateinit var
summaryRecycler: RecyclerView
    private lateinit var
progressBar: ProgressBar
    private lateinit var
progressText: TextView

    // Adapters and State
    private lateinit var
summaryAdapter: AnswerItemAdapter
    private var awaitingSubmission =
true // Tracks if we are waiting for
an answer submission or for user to
proceed to next question

    // --- Lifecycle Methods ---
```

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

super.onCreate(savedInstanceState)

setContentView(R.layout.activity_pra
ctise) // Reuse the layout for
practice
        Log.d("PracticeActivity",
"onCreate: Activity starting.")

        // Initialize ViewModel
        viewModel =
ViewModelProvider(this)[PracticeView
Model::class.java]

        // Get parameters from
intent
        val originLangs =
intent.getStringArrayListExtra("orig
inLangs") ?: arrayListOf()
        val translationLangs =
intent.getStringArrayListExtra("tran
slationLangs") ?: arrayListOf()
        val maxQuestions =
intent.getIntExtra("maxQuestions",
10)
        Log.d("PracticeActivity",
"onCreate: Intent extras -
originLangs: $originLangs,
translationLangs: $translationLangs,
maxQuestions: $maxQuestions")

        // Initialize UI Views
        questionText =
findViewById(R.id.textQuestion)
        editAnswer =
findViewById(R.id.editAnswer)
        submitButton =
findViewById(R.id.buttonSubmit)
        feedbackText =
findViewById(R.id.feedbackText)
        summaryRecycler =
findViewById(R.id.recyclerSummary)
// RecyclerView for summary items,
if displayed directly in this
activity.
        progressBar =
findViewById(R.id.progressBar)
        progressText =
findViewById(R.id.progressText)

        // Initialize RecyclerView
for summary
        summaryAdapter =
AnswerItemAdapter(emptyList())

summaryRecycler.layoutManager =
LinearLayoutManager(this)
        summaryRecycler.adapter =
summaryAdapter

        // Set initial UI state:
Show loading state
        questionText.visibility =
View.GONE
        editAnswer.visibility =
View.GONE
        submitButton.visibility =
View.GONE
        Log.d("PracticeActivity",
"onCreate: Initial UI set to loading
state.")

        // Observe ViewModel
LiveData
        setupObservers()

        // Start loading questions
for the practice test
        Log.d("PracticeActivity",
"onCreate: Starting practice test.")

viewModel.startPracticeTest(originLa
ngs, translationLangs, maxQuestions)

        // Initial progress display
update
        updateProgressDisplay()

        // Setup general event
listeners
        setupEventListeners()
    }

    override fun onDestroy() {
        super.onDestroy()
        Log.d("PracticeActivity",
"onDestroy: Activity is being
destroyed.")
    }

    override fun finish() {
        Log.d("PracticeActivity",
"finish: Activity is being
```

```
finished.")
        super.finish()
    }

    // --- ViewModel Observer Setup
---
    private fun setupObservers() {
        // Observe loading state to
show/hide question input fields

viewModel.isLoading.observe(this)
{ isLoading ->

Log.d("PracticeActivity", "isLoading
observer: $isLoading")
            if (isLoading) {

questionText.visibility = View.GONE

editAnswer.visibility = View.GONE

submitButton.visibility = View.GONE
            } else {

questionText.visibility =
View.VISIBLE

editAnswer.visibility = View.VISIBLE

submitButton.visibility =
View.VISIBLE
            }
        }

        // Observe the list of
questions

viewModel.questions.observe(this)
{ questions ->
            if (questions.isEmpty()
&& viewModel.isLoading.value ==
false) {

Log.w("PracticeActivity", "questions
observer: No practice questions
available.")
                Toast.makeText(
                    this,

getString(R.string.message_no_practi
ce_questions),

Toast.LENGTH_LONG
                ).show()
                finish() // Close
the activity if no questions
                return@observe
            }

Log.d("PracticeActivity", "questions
observer: Received ${questions.size}
questions.")
        }

        // Observe the current
question to display it

viewModel.currentQuestionLiveData.ob
serve(this) { question ->
            if (question != null) {

Log.d("PracticeActivity",
"currentQuestionLiveData observer:
New question displayed.")

showQuestion(question)

updateProgressDisplay()


submitButton.setText(R.string.action
_submit)

submitButton.setOnClickListener {
                    if
(!awaitingSubmission) {

Log.d("PracticeActivity", "Submit
button: Click ignored, not awaiting
submission.")

return@setOnClickListener
                    }
                    val answer =
editAnswer.text.toString().trim()
                    if
(answer.isNotEmpty()) {

Log.d("PracticeActivity", "Submit
button: Submitting answer.")

viewModel.submitAnswer(answer)

awaitingSubmission = false

hideKeyboard()
```

```
                    } else {

Log.d("PracticeActivity", "Submit
button: Answer is empty, submission
skipped.")
                    }
                }
            }
        }

        // Observe feedback for the
submitted answer and update UI

viewModel.feedback.observe(this)
{ response ->
            response?.let {

Log.d("PracticeActivity", "feedback
observer: Received feedback -
Correct: ${it.isCorrect}")
                val feedbackMsg = if
(it.isCorrect) {
                    val correctText
=
getString(R.string.feedback_correct_
text)
                    "<font
color='#2e7d32'>$correctText
☑</font>"
                } else {
                    val
incorrectText =
getString(R.string.feedback_incorrec
t_text)
                    val
correctAnswerPrefix =
getString(R.string.feedback_correct_
answer_prefix)
                    "<font
color='#c62828'>$incorrectText
❌</font><br/>$correctAnswerPrefix
${it.correctAnswer}"
                }
                feedbackText.text =
HtmlCompat.fromHtml(feedbackMsg,
HtmlCompat.FROM_HTML_MODE_LEGACY)


updateProgressDisplay()

                val
currentQuestionsCompleted =
viewModel.allAnswers.size
                val totalQuestions =
viewModel.currentQuestionList.size
                val
questionsRemaining = totalQuestions
- currentQuestionsCompleted


Log.d("PracticeActivity", "Questions
completed:
$currentQuestionsCompleted, Total:
$totalQuestions, Remaining:
$questionsRemaining")

                awaitingSubmission =
false

                // Check if this is
the LAST question (not if we've
finished all questions)
                // This means we
need to check if we're on the last
question, not if we've answered all
questions
                if
(questionsRemaining <= 0 ||
viewModel.isLastQuestionAnswered())
{

Log.d("PracticeActivity", "feedback
observer: Last question answered.
Setting button to 'Finish Test'.")

submitButton.setText(R.string.action
_finish_test)

submitButton.setOnClickListener {

Log.d("PracticeActivity", "Finish
Test button: Clicked.")

viewModel.finishTest()
                    }
                } else {

Log.d("PracticeActivity", "feedback
observer: More questions remain.
Setting button to 'Next'.")

submitButton.setText(R.string.action
_next)

submitButton.setOnClickListener {
```

```
Log.d("PracticeActivity", "Next
button: Clicked, proceeding to next
question.")

feedbackText.text = ""

editAnswer.text.clear()

awaitingSubmission = true

viewModel.goToNextQuestion()
                    }
                }
            }
        }

viewModel.testFinished.observe(this)
{ result ->
            result?.let {

Log.d("PracticeActivity",
"testFinished observer: Test
finished, showing summary.")

showSummary(viewModel.allAnswers,
it)
            }
        }
    }

    // --- UI Update Functions ---

    private fun
showQuestion(question:
PracticeQuestion) {
        questionText.text =
getString(R.string.question_format_t
ranslate, question.origin_lang,
question.translation_lang,
question.word)
        editAnswer.text.clear()
        feedbackText.text = ""
        Log.i("PracticeActivity",
"showQuestion: Displaying question
for word '${question.word}'")
    }

    private fun
updateProgressDisplay() {
        val totalQuestions =
viewModel.currentQuestionList.size
        val answeredQuestions =
viewModel.allAnswers.size

        progressBar.max = if
(totalQuestions > 0) totalQuestions
else 1
        progressBar.progress =
answeredQuestions
        progressText.text =
getString(R.string.progress_format,
answeredQuestions, totalQuestions)
        Log.d("PracticeActivity",
"updateProgressDisplay:
$answeredQuestions/$totalQuestions")
    }

    private fun showSummary(answers:
List<AnswerItem>, result:
FinishTestResponse) {
        Log.i("PracticeActivity",
"showSummary: Displaying test
summary.")
        val summaryFragment =
TestSummaryFragment.newInstanceForTe
stCompletion(
            answers = answers,
            result = result,
            navigationTarget =
R.id.navigation_practice
        )

summaryFragment.show(supportFragment
Manager, "test_summary")
    }

    // --- Event Listener Setup ---

    private fun
setupEventListeners() {
        // Fallback submit listener
(often overridden by observers)

submitButton.setOnClickListener {
            val answer =
editAnswer.text.toString().trim()
            if (answer.isNotEmpty())
{

viewModel.submitAnswer(answer)
                hideKeyboard()
            }
        }
```