МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ГОНЧАР АЛІНА АНАТОЛІЇВНА

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
д-р. техн. наук, професор
_____ Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ МОБІЛЬНОГО ДОДАТКУ ДЛЯ ЧИТАННЯ ІЗ ВБУДОВАНИМ ІНТЕРАКТИВНИМ СЛОВНИКОМ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Тетяна БАЦЕНКО, доцент кафедри
інформаційних технологій,
к. т. н., доцент
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2026

АНОТАЦІЯ

Гончар А. А. Розробка клієнтської частини мобільного додатку для читання із вбудованим інтерактивним словником. Спеціальність 122 «Комп'ютерні науки» освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса. Вінниця, 2026.

У кваліфікаційній роботі розроблено клієнтську частину мобільного додатку LexiRead для читання електронних книг із вбудованим інтерактивним словником. Реалізовано авторизацію користувача, бібліотеку книг, імпорт файлів TXT, PDF, EPUB і DOCX, екран читання, переклад слова через зовнішній REST API, збереження слів, флеш-картки, тести, закладки, налаштування читання, світлу й темну тему. Проведено unit- та widget-тестування клієнтської частини.

Ключові слова: мобільний додаток, Flutter, Dart, інтерактивний словник, електронні книги, REST API, клієнтська частина.

61 с. 28 рис., 11 табл., 5 дод., 44 джерела.

ABSTRACT

Honchar A. A. Development of the Client Side of a Mobile Reading Application with an Integrated Interactive Dictionary. Specialty 122 "Computer Science". Educational Programme "Computer Science". Vasyl' Stus Donetsk National University. Vinnytsia, 2026.

The qualification paper presents the development of the client side of the LexiRead mobile application for reading electronic books with an integrated interactive dictionary. The implemented functionality includes user authentication, a book library, TXT, PDF, EPUB, and DOCX import, a reading screen, word translation through an external REST API, word saving, flashcards, quizzes, bookmarks, reading settings, and light and dark themes. Unit and widget testing of the client side was performed.

Keywords: mobile application, Flutter, Dart, interactive dictionary, electronic books, REST API, client side.

61 pages, 28 figures, 11 tables, 6 appendices, 44 references.

ЗМІСТ

ВСТУП

Читання електронних текстів іноземною мовою є одним із практичних способів розширення словникового запасу, розвитку навичок розуміння контексту та самостійної роботи з іншомовними матеріалами. У мобільному середовищі такий підхід є особливо зручним, оскільки користувач може працювати з текстами незалежно від місця перебування. Водночас під час читання часто виникає потреба швидко з'ясувати значення окремого слова, не перериваючи процес роботи з книгою [1].

Використання зовнішніх словників або онлайн-перекладачів у такій ситуації потребує додаткових дій: копіювання слова, переходу до іншого застосунку, отримання перекладу та повернення до тексту. Це порушує безперервність читання й знижує зручність роботи з матеріалом. Тому актуальним є створення мобільного застосунку, у якому читання електронних книг поєднується з інтерактивною словниковою підтримкою безпосередньо в інтерфейсі читача.

Сучасні мобільні додатки для читання зазвичай забезпечують перегляд тексту, налаштування параметрів відображення, створення закладок і збереження позиції читання. Окремі рішення також містять словникові функції або інструменти для вивчення лексики. Проте не всі застосунки забезпечують цілісний сценарій, у якому читання тексту, отримання перекладу, збереження слова та подальше повторення лексики поєднані в межах одного мобільного інтерфейсу. Це зумовлює доцільність розробки клієнтської частини мобільного додатку LexiRead [2].

У межах бакалаврської роботи розроблено клієнтську частину мобільного додатку LexiRead для читання електронних книг із вбудованим інтерактивним словником. Додаток реалізовано з використанням Flutter і мови програмування Dart для платформи Android. Клієнтська частина забезпечує користувацький інтерфейс, навігацію між екранами, локальне керування станом, імпорт файлів книг, взаємодію із зовнішнім REST API, обробку JSON-даних, збереження частини налаштувань і відображення результатів у мобільному інтерфейсі.

Бакалаврська робота є складовою комплексного програмного проєкту. У межах цієї роботи розглядається саме клієнтська частина мобільного застосунку. Серверна частина, база даних і внутрішня backend-логіка не є предметом розробки та дослідження. Вони описуються лише як зовнішній REST API, з яким взаємодіє Flutter-клієнт.

Метою бакалаврської роботи є розробка клієнтської частини мобільного додатку для читання електронних книг із вбудованим інтерактивним словником, що забезпечує зручну взаємодію користувача з текстом, отримання перекладу слів, збереження нової лексики та використання навчальних інструментів для її повторення.

Для досягнення поставленої мети необхідно виконати такі завдання:

1. Проаналізувати предметну область мобільних додатків для читання та використання інтерактивних словників під час роботи з іншомовними текстами.
2. Дослідити існуючі програмні рішення для читання й вивчення лексики, визначити їхні переваги та обмеження.
3. Охарактеризувати функції інтерактивного словника та визначити його роль у мобільному додатку для читання.
4. Сформувати функціональні та нефункціональні вимоги до клієнтської частини мобільного застосунку.
5. Обґрунтувати вибір технологій і засобів реалізації клієнтської частини.
6. Спроєктувати архітектуру клієнтської частини, структуру інтерфейсу користувача та навігацію між екранами.
7. Спроєктувати механізм взаємодії клієнтської частини з інтерактивним словником через зовнішній REST API.
8. Реалізувати основні функціональні модулі клієнтської частини мобільного додатку.
9. Провести тестування клієнтської частини та проаналізувати результати роботи.

Об’єктом дослідження є процес читання електронних текстів іноземною мовою у мобільному середовищі.

Предметом дослідження є методи, засоби та програмні рішення для реалізації клієнтської частини мобільного застосунку для читання з інтерактивним словником.

Методи дослідження включають аналіз предметної області, порівняльний аналіз існуючих програмних рішень, формування функціональних і нефункціональних вимог, моделювання користувацьких сценаріїв, проєктування структури клієнтської частини, розробку інтерфейсу засобами Flutter, інтеграцію з REST API та тестування реалізованого програмного продукту.

Практичне значення одержаних результатів полягає у створенні клієнтської частини мобільного додатку LexiRead, яка може бути використана для читання електронних книг іноземною мовою, швидкого отримання перекладу слів без виходу з екрана читання, збереження нової лексики та її повторення за допомогою флеш-карток і тестів. Розроблений Flutter-клієнт може бути основою для подальшого розвитку комплексної системи читання й підтримки вивчення іноземної лексики.

Апробація результатів дослідження. Основні результати бакалаврської роботи були представлені на VII Всеукраїнській науково-практичній конференції здобувачів вищої освіти та молодих вчених «Прикладні інформаційні технології», що відбулася 15 травня 2026 року в Донецькому національному університеті імені Василя Стуса. За темою роботи підготовлено тези доповіді: Гончар А. А., Баценко Т. В. «Проєктування клієнтської частини мобільного додатку для читання з функцією інтерактивного словника»

Структура кваліфікаційної (бакалаврської) роботи. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел і додатків. Основна частина містить 12 підрозділів, 6 таблиць, 15 рисунків у тексті роботи та 44 джерела. Загальний обсяг роботи становить сторінок.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області мобільних додатків для читання

Поширення смартфонів і планшетів змінило способи споживання текстового контенту. За даними Pew Research Center (2025), близько 31% дорослих американців прочитали електронну книгу за попередній рік порівняно з 17% у 2011 р. [1]. Це свідчить про зростання ролі електронного читання та розвиток спеціалізованих мобільних рішень [1].

Мобільні застосунки для читання забезпечують відкриття текстів у форматах TXT, PDF, EPUB, DOCX, налаштування відображення, закладки та збереження позиції читання. У багатьох рідерах також передбачено пошук за ключовими словами в тексті [2; 4].

Окремим напрямом є інтеграція словникових інструментів. Натискання або утримання слова в тексті може відкривати контекстне вікно з визначенням або перекладом, що зменшує потребу переходити до іншого застосунку. Дослідження мобільного навчання також пов’язують роботу з незнайомими словами в контексті з ефективнішим засвоєнням лексики порівняно з ізольованим вивченням слів [5; 6].

Актуальною тенденцією є поєднання рідера з інструментами вивчення іноземних мов. Навчальні застосунки доповнюють базовий словниковий пошук перекладом фраз, мультимовними довідниками та механізмами формування словникового запасу. Технологія інтервального повторення, застосована, зокрема, у Readlang, використовується для довготривалого запам’ятовування лексики [7; 8].

Ринок мобільних застосунків для читання охоплює рішення різного рівня складності: від базових переглядачів текстів до платформ для мовного навчання. Частина з них зосереджена на комфорті читання без навчального сценарію, інша

на вивченні мови без повноцінного інтерфейсу рідера. Це створює потребу в застосунку, який поєднує обидва підходи в одному мобільному середовищі.

1.2 Аналіз існуючих аналогів

Для обґрунтування доцільності розроблення LexiRead проаналізовано чотири поширені рішення у відповідній предметній галузі: Amazon Kindle, Google Play Books, LingQ та Readlang. Критеріями порівняння обрано наявність вбудованого словника, можливість збереження слів, інструменти повторення лексики та зручність інтерфейсу.

Amazon Kindle підтримує формати AZW, MOBI, PDF та EPUB, надає налаштування параметрів відображення тексту і містить вбудовані словники кількох мов. Функція Vocabulary Builder автоматично зберігає переглянуті слова та формує картки для повторення. Обмеженням є відсутність тестів для перевірки лексики та неповна підтримка сторонніх форматів на Android [2; 4].

Google Play Books призначений для читання електронних книг і підтримує синхронізацію закладок та прогресу між пристроями. Під час читання користувач може перекладати виділене слово або фрагмент через інтегровані сервіси. Водночас у цьому рішенні не передбачено окремого сценарію збереження слів і повторення лексики [5].

LingQ орієнтований на вивчення мов через читання. Платформа дає змогу імпортувати тексти різних типів, зберігати слова у персональному словнику та відстежувати навчальний прогрес. Повторення лексики реалізовано через картки. Основним обмеженням є складніший інтерфейс, спрямований насамперед на мовне навчання, а не на комфортне читання [7].

Readlang є веборієнтованим інструментом для читання з миттєвим перекладом слів і збереженням лексики для подальшого повторення за принципом інтервального навчання. Обмеженнями є відсутність нативного мобільного застосунку та вужчі можливості рідера порівняно з повноцінними мобільними рішеннями [6; 8]. Для систематизації результатів порівняльний аналіз подано в таблиці 1.1.

Таблиця 1.1 — Порівняльна характеристика мобільних додатків для читання електронних книг

Функція / Додаток	**Kindle**	**Google Play Books**	**LingQ**	**Readlang**
Вбудований словник	наявний	наявний	наявний	наявний
Переклад тексту	частковий	наявний	наявний	наявний
Збереження слів	обмежене	відсутнє	наявне	наявне
Флеш-картки	наявні	відсутні	наявні	наявні
Синхронізація	наявна	наявна	наявна	наявна
Платформи	мобільні та ПК	мобільні та веб	мобільні та веб	веб
Зручність використання	висока	висока	середня	середня

Порівняння виявило спільні обмеження розглянутих рішень. У більшості застосунків інтерактивний словник виконує довідкову функцію і не пов’язаний із систематичним закріпленням лексики. Збереження слів або відсутнє, або реалізоване частково. Флеш-картки та тести наявні лише в окремих рішеннях і не завжди інтегровані в процес читання. Додатковим обмеженням є орієнтація частини платформ на веб або фірмові формати, що ускладнює використання довільних текстових файлів на Android [2; 5].

Жодне з розглянутих рішень не забезпечує повного поєднання читання, інтерактивного перекладу та системного повторення лексики в межах одного нативного мобільного застосунку. Це підтверджує доцільність розроблення LexiRead як Flutter-клієнта з відповідним функціональним комплексом.

1.3 Огляд інтерактивних словників та підходів до їх реалізації

Інтерактивний словник — це програмний компонент, інтегрований безпосередньо в середовище читання тексту, який забезпечує оперативний доступ до перекладу або тлумачення слова без переходу до зовнішнього застосунку. На відміну від автономних словникових інструментів, де пошук

відбувається поза контекстом тексту, інтерактивний словник зберігає безперервність процесу читання та знижує когнітивне навантаження на користувача [5; 7].

Ключовим механізмом взаємодії з інтерактивним словником є натискання або довге утримання слова в тексті. У відповідь інтерфейс відображає контекстне вікно або панель із перекладом і супровідною інформацією. У LexiRead цей сценарій реалізується через DictionaryBottomSheet — нижню панель, яка відкривається після натискання на будь-яке слово в ReaderScreen.

Серед функцій, характерних для сучасних інтерактивних словників, виділяються:

– миттєвий переклад або тлумачення слова з урахуванням мовного контексту;
– можливість збереження слова до персонального словника користувача;
– ведення списку збережених слів для подальшого перегляду;
– реалізація механізмів повторення лексики — флеш-карток або тестових вправ;
– кешування результатів попередніх запитів для підвищення швидкодії.

З точки зору технічної реалізації, існує кілька підходів до організації роботи інтерактивного словника в мобільному застосунку:

– локальне зберігання словникової бази — забезпечує автономність, але обмежена обсягом пам'яті та складністю оновлення;
– серверна обробка запитів через REST API — дозволяє використовувати актуальні дані та масштабувати систему, але вимагає наявності з'єднання з мережею [9; 10];
– використання зовнішніх сервісів перекладу — спрощує реалізацію, проте супроводжується залежністю від стороннього API та можливими обмеженнями на кількість запитів;
– комбінований підхід — поєднує локальне кешування відповідей із серверними запитами, що дозволяє досягти балансу між швидкодією, актуальністю даних і зниженням мережевого навантаження.

У клієнтській частині LexiRead реалізовано комбінований підхід: запити на переклад надсилаються до зовнішнього REST API через endpoint POST /translate, а локально через shared_preferences зберігаються токен авторизації, налаштування відображення, мовні параметри та кешовані дані. Сервіс DictionaryService також підтримує сесійний in-memory кеш, що прискорює повторні запити протягом поточного сеансу роботи. Таке рішення відповідає архітектурі Flutter-клієнта та не потребує вбудованої бази даних на пристрої.

Дослідження підтверджують, що використання словникового компонента безпосередньо в контексті читання позитивно впливає на засвоєння лексики порівняно з ізольованим вивченням слів. Це обґрунтовує інтеграцію інтерактивного словника в екран читання як ключову функціональну вимогу до клієнтської частини LexiRead [11; 12].

1.4 Постановка задачі та вимоги до клієнтської частини мобільного додатку

На основі аналізу предметної галузі, розглянутих аналогів і функціональних можливостей інтерактивних словників сформовано вимоги до клієнтської частини мобільного додатку LexiRead. Визначення таких вимог є необхідним етапом перед проєктуванням архітектури, навігації та механізмів взаємодії з інтерактивним словником [13; 14].

У межах бакалаврської роботи розробляється виключно клієнтська частина застосунку. Вона відповідає за інтерфейс користувача, навігацію між екранами, локальне управління станом, обробку дій користувача, імпорт файлів книг, надсилання запитів до зовнішнього REST API, парсинг JSON-відповідей і відображення результатів у мобільному інтерфейсі. Серверна частина, база даних і backend-логіка не є предметом цієї роботи — вони розглядаються виключно як зовнішній REST API, з яким взаємодіє Flutter-клієнт.

До функціональних вимог клієнтської частини належать:

– реєстрація та вхід користувача з JWT-авторизацією;
– перегляд бібліотеки книг та управління нею;

- додавання книги вручну або через імпорт файлу у форматах TXT, PDF, DOCX, EPUB;
- відкриття книги для посторінкового читання;
- натискання на слово в тексті та отримання перекладу через REST API без переходу на інший екран;
- відображення перекладу у нижній словниковій панелі (DictionaryBottomSheet);
- збереження слова до персонального словника через REST API;
- перегляд збережених слів на DictionaryScreen;
- повторення лексики через флеш-картки (FlashcardsScreen, FlashcardLearningScreen);
- тестування знань слів на QuizScreen;
- робота із закладками (BookmarksScreen);
- налаштування теми оформлення та параметрів читання (SettingsScreen).

Вимоги до інтерактивного словника виокремлені як ключові. Взаємодія з ним має відбуватися без виходу з екрана читання: клієнтська частина обробляє натискання на слово, формує запит до зовнішнього REST API (POST /translate), отримує відповідь, перетворює її у Dart-модель і відображає результат у нижній панелі. Після отримання перекладу користувач може зберегти слово до персонального словника через окремий запит (POST /dictionary). Такий підхід забезпечує безперервність читання та поєднує перегляд тексту з поступовим формуванням лексичного запасу [5; 9].

Нефункціональні вимоги визначають якісні характеристики клієнтської частини. Інтерфейс має бути послідовним, зрозумілим і адаптованим до мобільного використання. Основні сценарії — читання, переклад слова, збереження лексики та її повторення — повинні виконуватися без зайвих переходів між екранами. Застосунок не повинен блокувати UI під час асинхронних API-запитів і має коректно відображати стани завантаження, помилки та відсутності даних [14].

Важливою нефункціональною вимогою є підтримуваність коду. Структура Flutter-проєкту поділяється на папки screens, widgets, services, models, theme та data, що відповідає принципу розмежування відповідальності між рівнями інтерфейсу, логіки та даних. Такий поділ дозволяє незалежно розвивати окремі модулі: бібліотеку, читач, словник, картки, тести та налаштування. При цьому повноцінний офлайн-режим у поточній версії не реалізовано: словникові запити потребують з'єднання з мережею, а локально зберігаються токен, налаштування та окремі кешовані дані [15; 16].

Сформовані функціональні та нефункціональні вимоги визначають межі відповідальності клієнтської частини LexiRead і є основою для проєктування архітектури, навігації й механізму словникової взаємодії у наступному розділі.

РОЗДІЛ 2
ПРОЄКТУВАННЯ КЛІЄНТСЬКОЇ ЧАСТИНИ МОБІЛЬНОГО ДОДАТКУ

2.1 Вибір технологій та засобів розробки

Вибір технологічного стеку для клієнтської частини визначає структуру Flutter-проєкту, спосіб побудови інтерфейсу, навігацію, механізм API-взаємодії та можливості подальшої підтримки. Для LexiRead основними критеріями вибору стали продуктивність UI-рендерингу, підтримка асинхронного програмування, робота з файлами різних форматів і наявність відповідних pub-пакетів [17; 18].

Основним фреймворком для реалізації клієнтської частини обрано Flutter. Його декларативна модель на основі віджетів дає змогу поділяти екрани на логічні компоненти, повторно використовувати їх у різних частинах застосунку та керувати станом відображення. Для LexiRead це важливо, оскільки картки книг, словникові записи та інші UI-компоненти використовуються в кількох сценаріях. Flutter також забезпечує нативну продуктивність на Android завдяки власному механізму рендерингу, який не залежить від WebView або нативних платформних компонентів [17; 19].

Мовою програмування клієнтської частини є Dart. Вона підтримує строгу типізацію, об’єктно-орієнтовану модель, класи, інтерфейси та асинхронне програмування через Future і async/await. У LexiRead ці можливості потрібні для отримання списку книг, авторизації, запитів на переклад, збереження слів, завантаження карток і оновлення прогресу читання. Dart дає змогу описувати такі операції послідовно, без ускладнених callback-конструкцій [20].

Цільовою платформою реалізації та тестування визначено Android. Хоча Flutter підтримує кілька платформ, у роботі розглянуто саме Android-клієнт, що дає змогу конкретизувати вимоги до інтерфейсу, навігації та роботи з файлами пристрою [19].

Для взаємодії із зовнішнім REST API використано пакет http. У LexiRead він застосовується в ApiService для мережевих запитів, пов’язаних з

авторизацією, бібліотекою, перекладом (POST /translate), збереженням слів (POST /dictionary), прогресом читання та закладками. До авторизованих запитів додається заголовок Authorization: Bearer <token>. Пакет обрано як стабільне й легковаге рішення без надлишкових залежностей [9; 18].

Для локального збереження параметрів використано shared_preferences. У LexiRead пакет зберігає дані у форматі ключ-значення: токен авторизації (TokenStorage), тему оформлення, параметри читача, мови оригіналу й перекладу, кешовані обкладинки та фрагменти великих текстів. shared_preferences не замінює серверну базу даних: збережені слова, закладки та прогрес читання зберігаються на сервері й отримуються через REST API [21; 22].

Для вибору файлів із пам’яті пристрою використано file_picker. Пакет задіяний в AddBookScreen і відкриває нативний діалог вибору файлу з підтримкою TXT, PDF, DOCX та EPUB. Обраний файл передається до FileTextExtractor для витягування текстового вмісту [4; 23].

Для обробки PDF використано syncfusion_flutter_pdf. Пакет витягує текст зі сторінок PDF-документа на клієнтському рівні, без серверної обробки. Результат передається до AddBookScreen як рядок, придатний для збереження вмісту книги [24].

Керування станом реалізовано без сторонніх state management-бібліотек. Локальний стан екранів оновлюється через setState у StatefulWidget, а глобальні налаштування застосунку — через ValueNotifier<AppSettings> у SettingsService. Provider, Riverpod і Bloc у проєкті не використовуються. Такий підхід відповідає поточному обсягу функцій і не ускладнює структуру зайвими архітектурними шарами [15; 25].

UI-компоненти побудовано на Material Design, доступному у Flutter SDK. Це забезпечує використання стандартних Android-елементів: кнопок, полів введення, карток, панелей, діалогів, BottomSheet та індикаторів завантаження без підключення додаткових пакетів [17; 26].

Для тестування клієнтської частини використано flutter_test, вбудований у Flutter SDK. Він застосовується для unit-тестів моделей і валідаторів, а також для

widget-тестів окремих компонентів [27; 28]. Детальний опис тестування наведено у підрозділі 3.3. Загальну характеристику технологічного стеку клієнтської частини зведено в таблиці 2.1.

Таблиця 2.1 — Технологічний стек клієнтської частини мобільного додатку LexiRead

Технологія / засіб	**Призначення у LexiRead**	**Обґрунтування вибору**
Flutter 3.x	Побудова мобільного UI	Одна кодова база для Android, декларативна модель віджетів, висока продуктивність рендерингу
Dart 3.x	Мова клієнтської логіки	Єдина мова екосистеми Flutter, підтримка async/await, строга типізація
Material Design (Flutter SDK)	UI-компоненти	Стандартизовані Android-компоненти без додаткових залежностей
http ^1.5.0	HTTP-запити до REST API	Легковагий пакет, підтримка GET/POST/PUT/DELETE, заголовки авторизації
shared_preferences ^2.5.3	Локальне збереження параметрів	Збереження токена, теми, налаштувань читання, мовних параметрів
file_picker ^8.3.7	Вибір файлів книг	Підтримка TXT, PDF, DOCX, EPUB, нативний діалог вибору файлів
syncfusion_flutter_pdf ^28.2.12	Витягування тексту з PDF	Надійна обробка PDF без серверної частини, підтримка багатосторінкових документів
flutter_test	Unit та widget тести	Вбудований у Flutter SDK, підтримка тестування без додаткових залежностей
ValueNotifier<AppSettings>	Глобальне управління налаштуваннями	Реактивне оновлення теми та параметрів читання без сторонніх бібліотек

Обраний технологічний стек відповідає функціональним вимогам, сформованим у розділі 1. Flutter і Dart забезпечують побудову мобільного

інтерфейсу та реалізацію клієнтської логіки, пакет http використовується для взаємодії з REST API, shared_preferences забезпечує локальне збереження параметрів користувача, а file_picker і syncfusion_flutter_pdf реалізують імпорт та обробку файлів. Для перевірки окремих компонентів застосунку використовується пакет flutter_test. Сукупність обраних технологій формує технічну основу для подальшого проєктування архітектури клієнтської частини, розглянутої у наступному підрозділі. Допоміжну схему технологічного стеку клієнтської частини мобільного додатку подано на рисунку Б.1.

2.2 Проєктування архітектури клієнтської частини мобільного додатку

Архітектура клієнтської частини LexiRead визначає організацію коду Flutter-проєкту, взаємодію екранів, сервісів і моделей даних, а також спосіб звернення клієнта до зовнішнього REST API. Обрана структура має забезпечувати підтримуваність і тестованість застосунку без надмірного ускладнення, недоцільного для бакалаврського проєкту [15; 29].

Клієнтська частина побудована за комбінованим підходом feature-based + layered: структура проєкту орієнтована на функціональні частини застосунку, а всередині кожної частини збережено поділ між рівнями. Це не є строгою Clean Architecture, MVVM або MVC, оскільки окремий Repository layer і ViewModel-класи не реалізовано. Архітектуру коректно описувати як модульну клієнтську структуру з розмежуванням відповідальності між чотирма рівнями [16; 30]. Загальну архітектуру клієнтської частини мобільного додатку LexiRead наведено на рисунку 2.1. Окремі результати проєктування клієнтської частини були попередньо апробовані у тезах, присвячених архітектурі мобільного додатку для читання з інтерактивним словником [31].

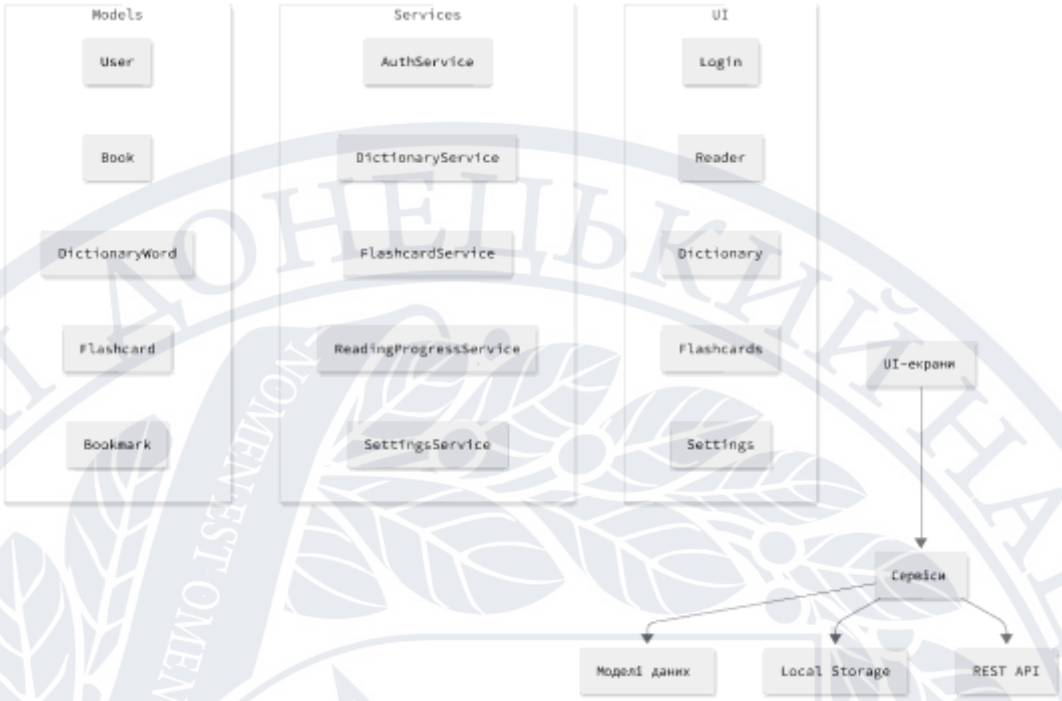

Рисунок 2.1 – Архітектура клієнтської частини мобільного додатку LexiRead

Перший рівень становить інтерфейсний шар (Presentation Layer). До нього належать каталоги screens і widgets. Екрани забезпечують відображення даних, обробку натискань і введення користувача, а також показ станів завантаження, помилки та відсутності даних. Повторно використовувані компоненти з каталогу widgets, зокрема картка книги або словниковий рядок, дозволяють відокремити окремі фрагменти інтерфейсу від конкретних екранів. Компонент DictionaryBottomSheet відкривається поверх ReaderScreen через showModalBottomSheet та виконує роль точки входу до словникового сценарію без переходу на окремий повноекранний інтерфейс [17; 26].

Другий рівень становить сервісний шар (Service / Logic Layer). Каталог services містить класи, які відокремлюють API-взаємодію, роботу з локальним сховищем та обробку файлів від компонентів інтерфейсу. Центральним елементом цього шару є ApiService, який виступає єдиною точкою доступу до зовнішнього REST API. Даний сервіс формує HTTP-запити, додає заголовки авторизації з Bearer token та забезпечує обробку мережевих помилок. Інші сервіси взаємодіють із ApiService відповідно до власного функціонального призначення. Зокрема, AuthService реалізує процеси авторизації та реєстрації

користувачів, BookService забезпечує роботу з бібліотекою книг, DictionaryService відповідає за переклад і збереження слів, FlashcardService використовується для роботи з навчальними картками, а QuizService реалізує функціонал тестування. Крім цього, BookmarkService забезпечує роботу із закладками, ReadingProgressService відповідає за збереження позиції читання, SettingsService керує параметрами інтерфейсу, TokenStorage виконує локальне збереження токена авторизації, а FileTextExtractor реалізує витягування тексту з файлів різних форматів [9; 32].

Третій рівень становить модельний шар (Data / Model Layer). Каталог models містить Dart-класи, що описують структуру даних застосунку. Кожна модель реалізує метод fromJson для десеріалізації JSON-відповідей від REST API. До основних моделей належать Book, User, DictionaryWord, DictionaryEntry, SavedWord, Flashcard, QuizQuestion, QuizAnswerResult, Bookmark і ReadingProgress. Використання типізованих моделей зменшує ризик помилок під час роботи з даними та спрощує їх використання на рівні інтерфейсу [20; 33].

Четвертий рівень становить шар оформлення (Theme Layer). Каталог theme містить кольори, стилі тексту, конфігурації світлої та темної теми. Централізоване зберігання цих параметрів дозволяє змінювати тему з SettingsScreen та застосовувати її до екранів через ValueNotifier<AppSettings> [17; 26].

Схему взаємодії між рівнями можна описати таким чином: екран отримує дію користувача, викликає метод відповідного сервісу, після чого сервіс звертається до ApiService. Далі ApiService надсилає HTTP-запит, а отримана JSON-відповідь перетворюється через Model.fromJson [9; 25]. Після цього дані повертаються до екрана, а інтерфейс оновлюється через setState. Структуру Flutter-проєкту LexiRead наведено на рисунку Б.2.

Кореневий каталог lib містить файли main.dart і app.dart. Файл main.dart є точкою входу до застосунку, оскільки виконує ініціалізацію SettingsService та запускає MaterialApp [17; 34]. Файл app.dart відповідає за конфігурацію

застосунку, встановлення теми, налаштування маршрутів через onGenerateRoute та визначення початкового маршруту /login.

Каталог screens організовано за функціональними підкаталогами auth, library, reader, dictionary, flashcards, quiz, settings і profile. Каталог widgets містить DictionaryBottomSheet та інші повторно використовувані компоненти. Каталог data використовується для допоміжних локальних структур, зокрема переліку підтримуваних мов. Каталог assets/fonts містить шрифт Nexa, який використовується в оформленні інтерфейсу. Каталог test призначений для unit- та widget-тестів.

Локальне збереження даних реалізовано через shared_preferences без використання SQLite, Hive або Firebase. На пристрої зберігаються access token через TokenStorage, тема та параметри читача через SettingsService, мовні налаштування, кешовані обкладинки книг у форматі base64, а також, за потреби, фрагменти великих текстів через BookService. Збережені слова, закладки та прогрес читання розміщуються на сервері та отримуються через REST API. In-memory кеш у DictionaryService прискорює повторні запити протягом одного сеансу роботи застосунку, проте очищується після його перезапуску [21; 22].

Обрана архітектура забезпечує розмежування відповідальності між окремими частинами клієнтського застосунку та формує основу для подальшого проєктування навігації й інтерфейсу, розглянутих у наступному підрозділі.

2.3 Проєктування навігації та структури користувацького інтерфейсу

Навігацію у LexiRead реалізовано через стандартний Navigator і централізоване налаштування маршрутів за допомогою onGenerateRoute у файлі app.dart. Бібліотеки go_router і auto_route у проєкті не використовуються. Такий підхід зберігає контроль над переходами без додаткових залежностей і відповідає кількості екранів поточної версії [34].

Стартовим маршрутом є /login. LoginScreen перевіряє наявність збереженого токена і, якщо токен існує, перенаправляє користувача до /library.

Так реалізовано базову перевірку авторизації без окремого механізму protected routes. У проєкті визначено маршрути /login, /register, /library, /library/add, /library/edit, /reader, /bookmarks, /dictionary, /word-detail, /flashcards, /flashcards/learning, /quiz, /profile і /settings. Навігаційну структуру застосунку наведено на рисунку 2.2.

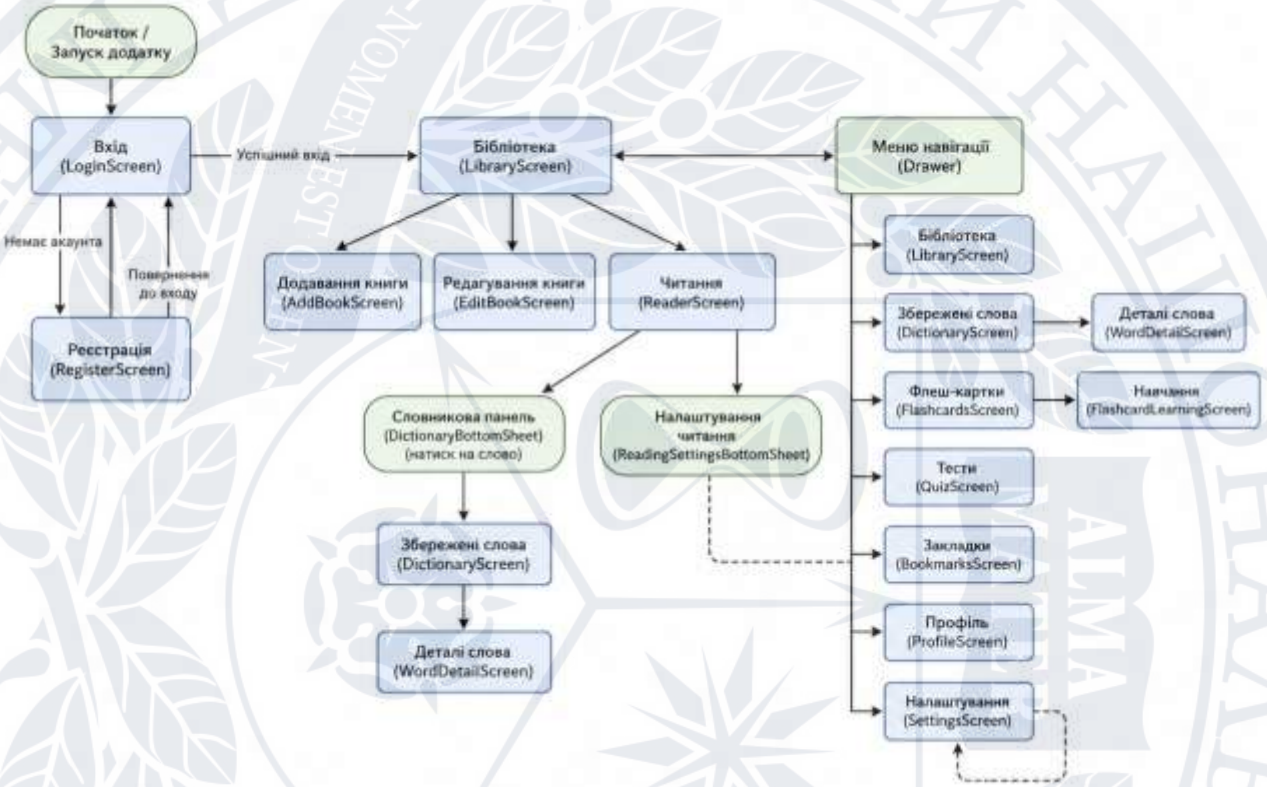

Рисунок 2.2 – Схема навігації між екранами мобільного додатку LexiRead

Ієрархія переходів має таку логіку. LoginScreen є точкою входу; з нього можливий перехід до RegisterScreen або, після авторизації, до LibraryScreen. LibraryScreen виконує роль головного екрана і забезпечує перехід до AddBookScreen, EditBookScreen та ReaderScreen. З ReaderScreen доступний DictionaryBottomSheet через showModalBottomSheet — без навігаційного переходу, поверх поточного екрана. Додаткові розділи — DictionaryScreen, FlashcardsScreen, QuizScreen, BookmarksScreen, ProfileScreen і SettingsScreen — доступні через навігаційне меню (Drawer). Перелік екранів та їх призначення наведено в таблиці 2.2.

Таблиця 2.2 — Екрани клієнтської частини мобільного додатку LexiRead

Екран / компонент	**Файл**	**Призначення**
LoginScreen	screens/auth/login_screen.dart	Вхід користувача за email і паролем
RegisterScreen	screens/auth/register_screen.dart	Реєстрація нового облікового запису
LibraryScreen	screens/library/library_screen.dart	Перегляд бібліотеки книг, навігаційний центр
AddBookScreen	screens/library/add_book_screen.dart	Додавання книги вручну або з файлу
EditBookScreen	screens/library/edit_book_screen.dart	Редагування метаданих книги
ReaderScreen	screens/reader/reader_screen.dart	Посторінкове читання книги
DictionaryBottomSheet	widgets/dictionary_bottom_sheet.dart	Словникова панель поверх ReaderScreen
DictionaryScreen	screens/dictionary/dictionary_screen.dart	Список збережених слів
WordDetailScreen	screens/dictionary/word_detail_screen.dart	Деталі окремого збереженого слова
FlashcardsScreen	screens/flashcards/flashcards_screen.dart	Список карток зі збереженої лексики
FlashcardLearningScreen	screens/flashcards/flashcard_learning_screen.dart	Навчальна сесія з флеш-картками
QuizScreen	screens/quiz/quiz_screen.dart	Тестування знань збереженої лексики
BookmarksScreen	screens/library/bookmarks_screen.dart	Перегляд і використання закладок
SettingsScreen	screens/settings/settings_screen.dart	Налаштування теми, мов, параметрів читання
ProfileScreen	screens/profile/profile_screen.dart	Профіль поточного користувача

Структура користувацького інтерфейсу побудована навколо трьох функціональних зон. Авторизаційна зона охоплює LoginScreen і RegisterScreen:

вони містять поля введення з валідацією, стан завантаження під час запиту та повідомлення про помилку в разі невдалої авторизації. Читацька зона включає LibraryScreen, AddBookScreen, EditBookScreen і ReaderScreen. Навчально-словникова зона об’єднує DictionaryScreen, FlashcardsScreen, FlashcardLearningScreen, QuizScreen, BookmarksScreen та SettingsScreen. Під час проєктування інтерфейсу мобільного застосунку враховано зрозумілість переходів, передбачуваність реакцій системи та зручність взаємодії користувача з основними екранами [35].

LibraryScreen є центральним екраном після входу. Він відображає список книг користувача, отриманий через GET /books, і забезпечує перехід до додавання нової книги або читання. Для бібліотеки передбачено стани завантаження, порожнього списку та помилки API.

ReaderScreen є ключовим екраном основного сценарію. Текст книги відображається посторінково через PageView. Кожна сторінка рендериться через RichText: перед показом текст розбивається регулярним виразом на слова та пробільні символи. Для кожного слова створюється TextSpan з TapGestureRecognizer, що дає змогу натискати на окреме слово без SelectableText або SelectionArea. ReaderScreen також відкриває ReadingSettingsBottomSheet і підтримує додавання закладок [36; 37].

DictionaryBottomSheet відкривається поверх ReaderScreen і відображає переклад натиснутого слова. Компонент не є окремим маршрутом, а викликається через showModalBottomSheet. Панель містить вибране слово, переклад, транскрипцію та частину мови за наявності у відповіді API, контекст із книги, вибір мов оригіналу й перекладу, а також кнопку «У словник» [9; 26]. Взаємодію з API детально описано в підрозділі 2.4.

FlashcardsScreen відображає навчальні картки, сформовані зі збережених слів (GET /flashcards). FlashcardLearningScreen реалізує навчальну сесію: картки показуються послідовно, а результат відповіді надсилається через PUT /flashcards/{id}. QuizScreen забезпечує тестування; питання формуються локально у QuizService зі словникових даних, без виклику GET /quiz/question. Це

відображає поточну реалізацію та коректно описується як локальна генерація питань.

SettingsScreen забезпечує зміну теми, мов оригіналу й перекладу, а також параметрів читача: розміру шрифту, гарнітури, теми сторінки, кольору тексту, міжрядкового інтервалу, вирівнювання та комфортного режиму. Параметри зберігаються через SettingsService у shared_preferences і застосовуються до ReaderScreen через ValueNotifier<AppSettings> [21; 25].

Для екранів, що залежать від API-запитів, передбачено чотири стани: завантаження (CircularProgressIndicator), успішне отримання даних, помилка мережі або сервера та відсутність даних. Ці стани використовуються в LibraryScreen, DictionaryScreen, FlashcardsScreen, QuizScreen і BookmarksScreen. Явне відображення станів робить поведінку інтерфейсу зрозумілою в разі порожнього списку або мережевого збою [39; 40].

Проєктування навігації та інтерфейсу LexiRead забезпечує послідовний перехід від авторизації до читання, словникової взаємодії та повторення лексики. Наступний підрозділ присвячено клієнтській логіці взаємодії з інтерактивним словником.

2.4 Проєктування взаємодії з інтерактивним словником

Взаємодія з інтерактивним словником є центральним функціональним сценарієм LexiRead, який поєднує читання тексту з опрацюванням нової лексики. Основна вимога до цього сценарію полягає в отриманні перекладу без виходу з ReaderScreen і без копіювання слова до стороннього перекладача. У проєкті її реалізовано через TapGestureRecognizer на словах тексту, модальний BottomSheet і REST API-запит на переклад [5; 36]. Серверна обробка запитів через REST API дозволяє відокремити клієнтську логіку від backend-рівня та організувати обмін даними між мобільним інтерфейсом і сервером через стандартизовані HTTP-запити [38].

На рівні клієнтської частини словниковий сценарій складається з трьох етапів: розпізнавання слова в тексті та відкриття словникової панелі; отримання перекладу через зовнішній REST API; збереження слова до персонального словника. Кожен етап виконується відповідними компонентами клієнтської архітектури, описаної у підрозділі 2.2. Послідовність взаємодії між компонентами наведено на рисунку 2.3.

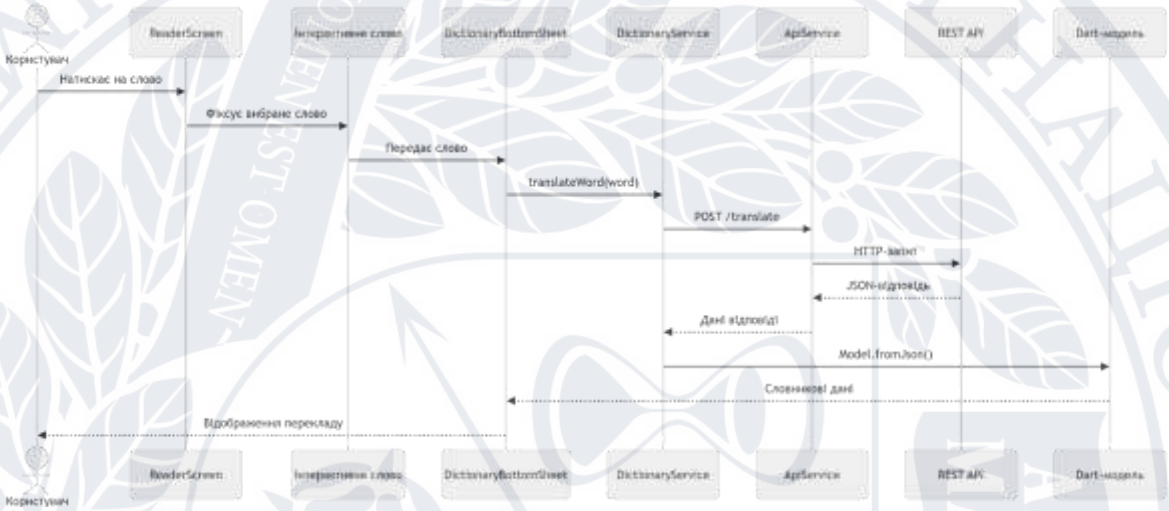

Рисунок 2.3 – Sequence diagram сценарію отримання перекладу слова

На першому етапі текст книги в ReaderScreen рендериться через RichText. Перед відображенням текст сторінки попередньо обробляється: за допомогою регулярного виразу RegExp(r'\S+|\s+') він розбивається на окремі токени, зокрема слова та пробільні символи. Для кожного слова створюється TextSpan із TapGestureRecognizer, який реагує на натискання. Після натискання DictionaryBottomSheet відкривається через showModalBottomSheet. Механізм не потребує SelectableText, SelectionArea або контекстного меню; єдиним тригером словникової взаємодії є натискання на слово [36; 37].

На другому етапі DictionaryBottomSheet викликає DictionaryService.translateWord і передає слово, мову оригіналу та мову перекладу. DictionaryService через ApiService формує POST-запит до /translate з полями word, source_language і target_language. Успішна відповідь повертає JSON з перекладом; сервіс десеріалізує її в модель DictionaryWord через fromJson і передає результат до BottomSheet. Інтерфейс оновлює стан через setState та

відображає слово, переклад, транскрипцію, частину мови за наявності, контекст із книги, вибір мов і кнопку «У словник» [9; 20].

Збереження слова реалізовано окремим кроком. Після отримання перекладу користувач натискає «У словник». DictionaryBottomSheet викликає DictionaryService.saveWord, який надсилає POST-запит до /dictionary з полями word, translation, source_language і target_language. Після успішного збереження панель показує підтвердження, а слово стає доступним на DictionaryScreen. Діаграму активності для сценарію перекладу та збереження наведено на рисунку Б.3.

У словниковому сценарії передбачено чотири стани DictionaryBottomSheet: завантаження під час очікування відповіді API, успішне отримання перекладу, помилка та стан після збереження слова. Явне відображення цих станів забезпечує передбачуваність інтерфейсу й надає користувачу зворотний зв’язок у разі мережевих збоїв або відсутності перекладу [39].

Поточна реалізація має визначені обмеження. Переклад і збереження слів залежать від зовнішнього REST API, тому без мережевого з’єднання DictionaryBottomSheet відображає стан помилки. Постійне офлайн-сховище словника на клієнті не реалізовано. DictionaryService містить in-memory кеш для повторних запитів у межах сеансу, але цей кеш очищується після перезапуску застосунку. Локально через shared_preferences зберігаються мовні налаштування, а не словникові записи.

Збережена лексика є вхідними даними для навчальних модулів. FlashcardsScreen отримує картки через GET /flashcards, де кожна картка пов’язана зі збереженим словом. QuizScreen формує тестові питання локально в QuizService зі словникових даних, без виклику серверного ендпоінта. Тому словниковий сценарій не завершується збереженням слова, а переходить у навчальний ланцюжок: читання → переклад → збереження → повторення через картки → тестування [11; 12].

Проєктування взаємодії з інтерактивним словником завершує розгляд клієнтської архітектури, навігації та UI. У наступному розділі описано фактичну реалізацію зазначених компонентів і результати тестування клієнтської частини.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКУ

3.1 Реалізація основного функціоналу клієнтської частини

Після завершення етапу проєктування було виконано реалізацію клієнтської частини мобільного додатку LexiRead. Основним завданням цього етапу стало створення працездатного Flutter-застосунку, який забезпечує взаємодію користувача з електронними книгами, інтерактивним словником, збереженими словами, навчальними картками, тестами, закладками та налаштуваннями читання. Реалізація клієнтської частини виконувалася з урахуванням архітектурних рішень, описаних у другому розділі, а також функціональних і нефункціональних вимог, сформованих у першому розділі.

Клієнтську частину мобільного додатку реалізовано з використанням Flutter і мови програмування Dart. Структура проєкту побудована таким чином, щоб розділити основні частини застосунку за їх призначенням. Екрани користувацького інтерфейсу розміщено у каталозі screens, повторно використовувані компоненти у widgets, сервіси взаємодії з API та локальними даними у services, моделі даних у models, а параметри теми оформлення у theme. Така організація дозволяє відокремити інтерфейс від сервісної логіки та спрощує подальшу підтримку програмного продукту.

Архітектуру та структуру Flutter-проєкту розглянуто у розділі 2 і докладно наведено на рисунку Б.2 у додатку Б. У цьому підрозділі основну увагу зосереджено на реалізації функціональних модулів клієнтської частини: авторизації, бібліотеки книг, імпорту файлів, читання, словника, флеш-карток, тестування, закладок і налаштувань. Фрагменти програмного коду ключових компонентів, зокрема ApiService та маршрутизації у app.dart, наведено у додатку Д.

У реалізованому застосунку файл main.dart виконує роль точки входу, а файл app.dart відповідає за налаштування основного застосунку, маршрутизацію

та підключення теми. Навігація між екранами реалізована за допомогою стандартного механізму Navigator та централізованого опису маршрутів через onGenerateRoute. Такий підхід дозволяє організувати переходи між екранами авторизації, бібліотеки, читання, словника, флеш-карток, тестування, закладок, профілю та налаштувань.

У клієнтській частині реалізовано модуль авторизації користувача. Він включає екрани входу та реєстрації, а також сервіс AuthService, який забезпечує взаємодію з відповідними API-запитами. На екрані входу користувач вводить електронну пошту та пароль, після чого застосунок надсилає запит до серверної частини. У разі успішної авторизації клієнт отримує access token, який зберігається локально за допомогою TokenStorage і shared_preferences. Надалі цей токен використовується під час виконання авторизованих запитів до API.

Приклад екрана авторизації наведено на рисунку 3.1, а екран реєстрації користувача — на рисунку А.1.

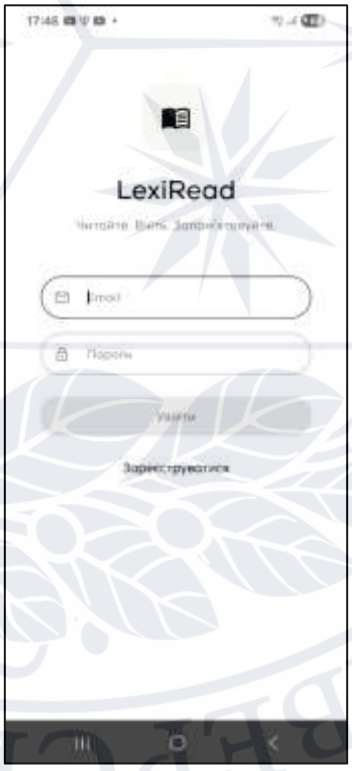

Рисунок 3.1 – Екран авторизації користувача

Після успішної авторизації відкривається екран бібліотеки книг. Модуль бібліотеки реалізовано на основі LibraryScreen і BookService. LibraryScreen

відображає список книг користувача у вигляді карток з основною інформацією про книгу. Через цей екран користувач відкриває книгу для читання, переходить до додавання нового матеріалу або виконує дії з керування бібліотекою. За відсутності книг інтерфейс показує порожній стан із пропозицією додати першу книгу. Приклад екрана бібліотеки наведено на рисунку 3.2. Навігаційне меню, через яке відкриваються словник, флеш-картки, тести, закладки, профіль і налаштування, наведено на рисунку А.2.

Рисунок 3.2 – Екран бібліотеки книг

Додавання книги реалізовано через AddBookScreen. Екран дає змогу ввести основні відомості про книгу та вибрати файл із пам’яті пристрою. Для вибору файлів використано file_picker, а для обробки текстового вмісту — FileTextExtractor. Застосунок підтримує текстові матеріали у форматах TXT, PDF, DOCX та EPUB. Для PDF-файлів використано syncfusion_flutter_pdf, що витягує текстовий вміст документа для подальшого відображення в ReaderScreen. Стан після імпорту файлу наведено на рисунку А.3, фрагмент витягування тексту з PDF — у додатку Д.

Приклад екрана додавання книги наведено на рисунку 3.3.

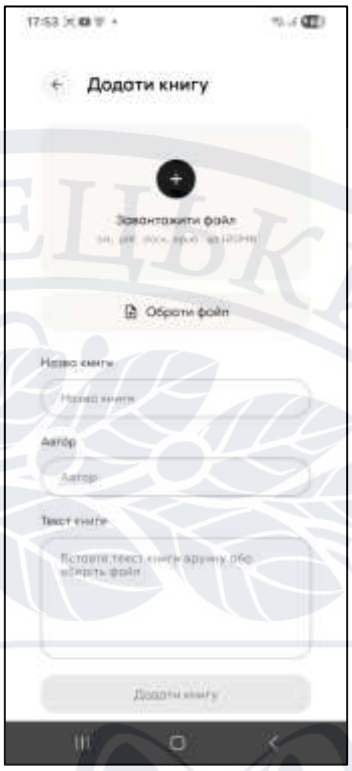

Рисунок 3.3 – Екран додавання книги

Модуль читання є центральною частиною клієнтського застосунку. ReaderScreen відображає текст книги, підтримує посторінкову навігацію, роботу з поточною позицією читання та доступ до додаткових функцій. На цьому екрані користувач переходить між сторінками, переглядає прогрес, відкриває налаштування читання та додає закладки. ReaderScreen також є основою для інтерактивного словника: слова в тексті реагують на натискання і відкривають словникову панель. Панель налаштування параметрів читання представлено на рисунку А.4. Приклад екрана читання наведено на рисунку 3.4.

Рисунок 3.4 – Екран читання книги ReaderScreen

Збереження позиції читання виконується через окремий сервіс прогресу. Клієнт отримує поточний прогрес книги та передає оновлену позицію після зміни сторінки або завершення читання фрагмента. Такий механізм дає змогу повернутися до попереднього місця після повторного відкриття книги. У клієнтській частині реалізовано відображення й оновлення прогресу через API; внутрішня серверна логіка збереження цих даних не розглядається.

ReaderScreen також підтримує роботу із закладками. Користувач може зберегти важливу позицію в книзі та пізніше повернутися до неї через BookmarksScreen. Модуль закладок реалізовано за допомогою BookmarksScreen і BookmarkService. На екрані закладок відображається список збережених позицій, пов’язаних із книгами користувача. Цей інструмент спрощує повторне відкриття потрібних фрагментів тексту. Екран закладок BookmarksScreen наведено на рисунку А.5.

Модуль збережених слів реалізовано через DictionaryScreen і DictionaryService. DictionaryScreen відображає персональний словник користувача, сформований під час читання та взаємодії зі словниковою панеллю.

Користувач може переглядати збережені слова, шукати записи, відкривати детальну інформацію про слово або видаляти непотрібні записи. Модуль пов’язує процес читання з подальшим повторенням лексики. Приклад екрана збережених слів наведено на рисунку 3.5.

Рисунок 3.5 – Екран збережених слів користувача

Для повторення збережених слів реалізовано модуль флеш-карток, що складається з FlashcardsScreen, FlashcardLearningScreen і FlashcardService. FlashcardsScreen відображає доступні картки та відкриває навчальну сесію. Під час навчання користувач переглядає картку, перевіряє себе та переходить до наступних слів. Модуль доповнює словникову функціональність і перетворює перекладені слова з довідкових записів на матеріал для повторення. Екрани навчальної сесії флеш-карток наведено на рисунках А.6 і А.7.

Приклад екрана флеш-карток наведено на рисунку 3.6.

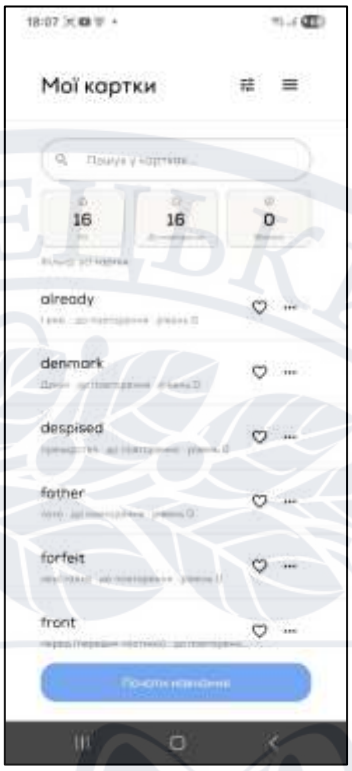

Рисунок 3.6 – Екран флеш-карток

Окремо реалізовано модуль тестування слів, що включає QuizScreen і QuizService. QuizScreen відображає питання, варіанти відповідей і результат вибору користувача. У поточній клієнтській реалізації питання формуються на основі словникових даних, що дає змогу перевіряти засвоєння лексики, збереженої під час читання. Модуль є логічним продовженням словникового сценарію, оскільки забезпечує активну перевірку знань. Приклад екрана тестування та результату його проходження наведено на рисунках А.8 і А.9. Приклад екрана тестування наведено на рисунку 3.7.

Рисунок 3.7 – Екран тестування слів

Модуль налаштувань реалізовано через SettingsScreen, SettingsService і модель AppSettings. Він відповідає за зміну параметрів інтерфейсу та читання: світлої й темної теми, мови оригіналу, мови перекладу, розміру тексту, шрифту, теми сторінки, кольору тексту, міжрядкового інтервалу та режиму вирівнювання. Для глобального оновлення параметрів використано ValueNotifier<AppSettings>, що дає змогу застосовувати зміни без зовнішніх state management-бібліотек. Приклад екрана налаштувань наведено на рисунку 3.8, фрагмент збереження параметрів через shared_preferences у SettingsService — у додатку Д.

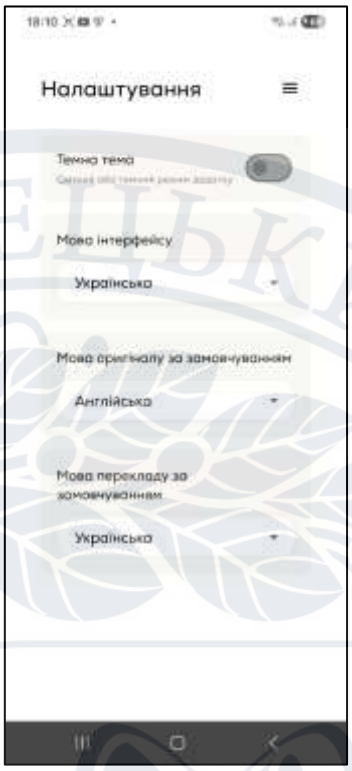

Рисунок 3.8 – Екран налаштувань користувача

Стан у реалізованому застосунку керується переважно на рівні окремих екранів через setState. Цей механізм оновлює інтерфейс після завантаження даних, завершення API-запиту, зміни стану завантаження, виникнення помилки або виконання дії користувача. Глобальні параметри налаштувань винесено в SettingsService, де використано ValueNotifier. Завдяки цьому зміна теми або параметрів читання застосовується до різних частин інтерфейсу.

У клієнтській частині передбачено обробку основних станів інтерфейсу: завантаження, успішного отримання даних, помилки та відсутності даних. Завантаження відображається під час авторизації, отримання бібліотеки, відкриття словника, завантаження збережених слів, флеш-карток, тестів і закладок. Помилка показується в разі мережевих проблем, некоректної відповіді API або помилок обробки даних. Порожній стан використовується, коли список книг, збережених слів, флеш-карток або закладок не містить записів. Загальну характеристику реалізованих функціональних модулів клієнтської частини наведено в таблиці 3.1.

Таблиця 3.1 — Основні функціональні модулі клієнтської частини LexiRead

Модуль	**Основні компоненти**	**Реалізований функціонал**
Авторизація	`LoginScreen`, `RegisterScreen`, `AuthService`, `TokenStorage`	Вхід, реєстрація, збереження access token
Бібліотека	`LibraryScreen`, `BookService`	Отримання та відображення списку книг
Додавання книг	`AddBookScreen`, `FileTextExtractor`, `file_picker`	Додавання книги вручну та імпорт файлу
Читання	`ReaderScreen`, `ReadingProgressService`	Посторінкове читання, відображення тексту, збереження прогресу
Закладки	`BookmarksScreen`, `BookmarkService`	Додавання, перегляд і видалення закладок
Словник	`DictionaryBottomSheet`, `DictionaryScreen`, `DictionaryService`	Переклад слова, збереження слова, перегляд словника
Флеш-картки	`FlashcardsScreen`, `FlashcardLearningScreen`, `FlashcardService`	Повторення збережених слів у форматі карток
Тестування	`QuizScreen`, `QuizService`	Перевірка знань слів у форматі тесту
Налаштування	`SettingsScreen`, `SettingsService`, `AppSettings`	Зміна теми, мов і параметрів читання
Тема інтерфейсу	`app_theme.dart`, `app_colors.dart`	Підтримка світлої та темної теми

Реалізовані модулі пов’язані навігацією та спільною логікою роботи з даними. Користувач додає книгу в бібліотеку, відкриває її в ReaderScreen, натискає на незнайоме слово, зберігає його у словник, а потім повторює за допомогою флеш-карток або тестів. Цей сценарій показує, що функції застосунку формують єдиний процес роботи з іншомовним текстом, а не існують як ізольовані розділи.

У реалізованому застосунку не використано окремий Repository layer або ViewModel-рівень. Взаємодію з даними організовано через сервісні класи, які

виконують API-запити, обробляють відповіді та повертають результат екранам. Такий підхід відповідає фактичній структурі проєкту та поточній складності бакалаврського програмного продукту. У подальшому застосунок можна розширити складнішим підходом до керування станом, однак для цієї реалізації це не є обов'язковою вимогою.

У межах реалізації клієнтської частини LexiRead створено основні функціональні модулі для роботи з електронними книгами, інтерактивним словником і навчальними інструментами: авторизацію, бібліотеку книг, додавання книг, екран читання, прогрес читання, закладки, збережені слова, флеш-картки, тестування та налаштування інтерфейсу. Клієнтська частина забезпечує взаємодію користувача з основними функціями застосунку та створює основу для детальнішого розгляду інтеграції інтерактивного словника у наступному підрозділі.

3.2 Інтеграція інтерактивного словника

Інтеграція інтерактивного словника є одним із центральних функціональних елементів клієнтської частини мобільного додатку LexiRead. Саме цей модуль відрізняє застосунок від звичайного мобільного рідера, оскільки забезпечує користувачу можливість отримувати переклад незнайомого слова безпосередньо під час читання книги. У межах реалізації клієнтської частини основну увагу приділено тому, щоб словникова взаємодія не порушувала процес читання, не вимагала переходу до стороннього сервісу та була доступною через мінімальну кількість дій.

Функціональність інтерактивного словника реалізовано у зв'язці з екраном читання ReaderScreen. На цьому екрані текст книги відображається не як суцільний статичний блок, а як набір інтерактивних текстових фрагментів. Для цього використовується віджет RichText, у межах якого окремі частини тексту формуються за допомогою TextSpan. Слова в тексті обробляються таким чином,

щоб користувач міг натиснути на конкретне слово та ініціювати словниковий запит.

У процесі реалізації текст сторінки розбивається на окремі фрагменти, серед яких визначаються слова та пробіли. Для кожного слова створюється текстовий елемент із прив’язаним обробником натискання. Обробка дії користувача здійснюється за допомогою TapGestureRecognizer. Після натискання на слово застосунок відкриває нижню словникову панель, у якій виконується запит на переклад і відображається результат.

Такий підхід є важливим для збереження цілісності користувацького сценарію. Користувач не копіює слово, не переходить до окремого перекладача й не залишає екран читання. Уся взаємодія відбувається в межах поточного контексту книги. Це забезпечує безперервність читання та робить словниковий модуль частиною основного сценарію роботи з іншомовним текстом.

Після натискання на слово відкривається компонент DictionaryBottomSheet. Він реалізований як нижня модальна панель, що з’являється поверх екрана читання. Використання саме такого формату подання словникової інформації є доцільним для мобільного застосунку, оскільки панель не потребує повного переходу на інший екран і дозволяє швидко повернутися до читання. Крім того, нижня панель є зручною для взаємодії на смартфоні, оскільки розташовується в зоні, доступній для натискання однією рукою.

DictionaryBottomSheet виконує роль інтерфейсного компонента для відображення результатів словникового запиту. У ньому передбачено виведення вибраного слова, перекладу, частини мови за наявності, контексту з книги, а також вибору мови оригіналу та мови перекладу. Окремим елементом є кнопка «У словник», за допомогою якої користувач може зберегти перекладене слово у персональний словник. Приклад нижньої словникової панелі наведено на рисунку 3.9.

Рисунок 3.9 – Нижня словникова панель DictionaryBottomSheet

Технічно словникова взаємодія реалізована через сервісний клас DictionaryService. Цей сервіс відповідає за операції перекладу слова, отримання списку збережених слів, додавання слова до словника та видалення словникового запису. Для виконання HTTP-запитів DictionaryService використовує ApiService, який централізує роботу із зовнішнім REST API. Фрагменти реалізації класів DictionaryService і ApiService наведено у додатку Д.

Під час отримання перекладу вибране слово передається до методу translateWord. Далі формується HTTP-запит до endpoint /translate. Клієнтська частина передає до API слово, мову оригіналу та мову перекладу. Після отримання відповіді JSON-дані перетворюються у відповідну Dart-модель, після чого результат відображається у DictionaryBottomSheet. Такий підхід відповідає загальній архітектурі клієнтської частини: інтерфейс ініціює дію, сервіс виконує логіку взаємодії з API, модель структурує отримані дані, а UI відображає результат. Логіку отримання перекладу слова доцільно подати у вигляді схеми на рисунку 3.10.

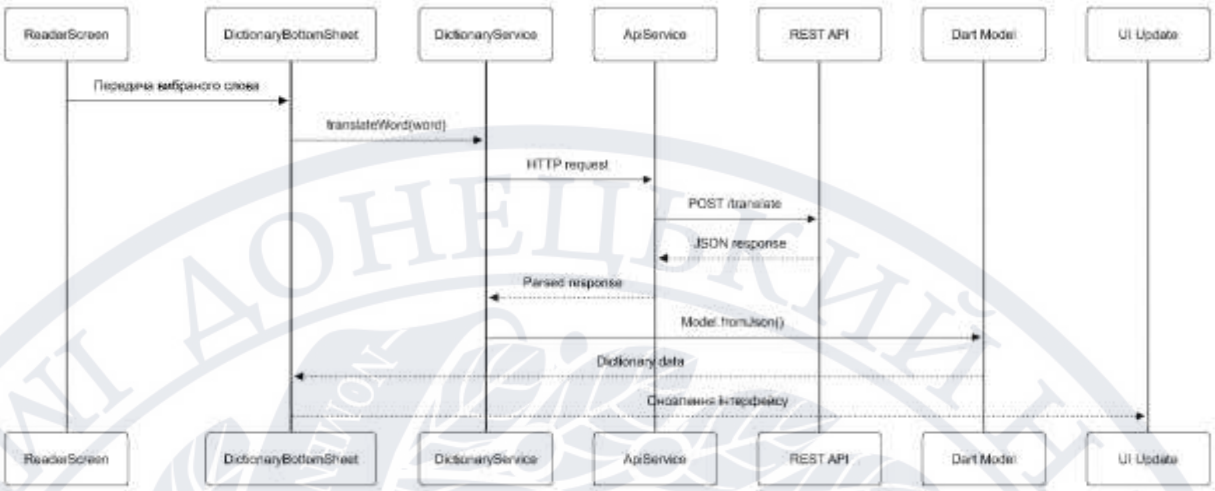

Рисунок 3.10 – Послідовність отримання перекладу слова у клієнтській частині

Окрему роль у словниковій інтеграції відіграє обробка станів інтерфейсу. Оскільки переклад слова виконується через мережевий запит, застосунок коректно реагує на різні результати цієї операції. Під час очікування відповіді від API у словниковій панелі відображається індикатор завантаження. Після успішного отримання даних користувач бачить переклад та додаткову інформацію. Якщо запит завершується помилкою, інтерфейс повідомляє про неможливість отримати переклад (рисунок А.10). Якщо відповідь не містить необхідних даних, відображається порожній стан або повідомлення про відсутність результату.

Наявність таких станів є важливою для зручності використання застосунку. Якщо користувач натискає на слово, але не бачить жодної реакції інтерфейсу, це може створити враження помилки або зависання застосунку. Відображення індикатора завантаження, результату або повідомлення про помилку робить поведінку системи передбачуваною та зрозумілою.

Сценарій збереження слова є важливою частиною навчальної логіки застосунку. Якщо переклад використовується лише один раз і не зберігається, словник виконує переважно довідкову функцію. У LexiRead збережене слово переходить до персонального словника користувача й надалі може використовуватися у флеш-картках і тестах. Таким чином, словниковий модуль забезпечує зв’язок між читанням і подальшим повторенням лексики.

Екран збережених слів реалізований як окремий функціональний розділ застосунку. Він дозволяє переглядати слова, які користувач додав під час читання, шукати необхідні записи, відкривати детальну інформацію про слово та видаляти записи. Збережені слова отримуються через DictionaryService, який виконує запит до endpoint /dictionary. Видалення слова здійснюється через відповідний DELETE-запит. У клієнтській частині реалізовано не серверне збереження словника, а інтерфейсну та сервісну логіку доступу до цих даних. Загальну характеристику реалізованих словникових операцій наведено в таблиці 3.2. Повний перелік REST API-запитів із параметрами та очікуваними відповідями наведено у таблицях В.1 і В.2.

Таблиця 3.2 — Загальна характеристику реалізованих словникових операцій

Операція	**Компонент клієнтської частини**	**API-запит**	**Результат для користувача**
Вибір слова в тексті	ReaderScreen, TextSpan, TapGestureRecognizer	API-запит ще не виконується	Відкривається словникова панель
Отримання перекладу	DictionaryBottomSheet, DictionaryService	POST /translate	Відображається переклад слова
Збереження слова	DictionaryBottomSheet, DictionaryService	POST /dictionary	Слово додається до персонального словника
Перегляд збережених слів	DictionaryScreen, DictionaryService	GET /dictionary	Відображається список збережених слів
Видалення слова	DictionaryScreen, DictionaryService	DELETE /dictionary/{id}	Слово видаляється зі списку

Під час реалізації словникового модуля розмежовано клієнтську та серверну відповідальність. Клієнтська частина обробляє натискання на слово, відкриває словникову панель, надсилає запит, обробляє відповідь, відображає переклад і зберігає результат за дією користувача. Серверна частина обробляє запит, формує переклад і зберігає дані. У цій бакалаврській роботі серверна реалізація не деталізується, оскільки не є предметом дослідження.

Для коректної роботи словникового модуля враховано мовні налаштування. У SettingsScreen користувач визначає мову оригіналу та мову перекладу за замовчуванням. Ці параметри використовуються під час формування запиту на переклад, що дає змогу адаптувати словник до текстів різними мовами.

Інтерактивний словник не винесено в ізольований сценарій, а пов'язано з іншими модулями. Слово, перекладене під час читання, може бути збережене та відображене у списку збережених слів. Далі ці дані використовуються у флеш-картках і тестах. Така логіка формує послідовний користувацький шлях: читання тексту, виявлення незнайомого слова, отримання перекладу, збереження слова та подальше повторення.

У реалізованій версії словникова взаємодія не базується на стандартному механізмі виділення тексту. У застосунку не використано SelectableText, SelectionArea або контекстне меню Android для перекладу. Основним механізмом є натискання на слово, сформоване через TextSpan і TapGestureRecognizer. Це формулювання важливе для коректного опису фактичної реалізації LexiRead і не створює хибного уявлення про використання інших механізмів роботи з текстом.

Приклади JSON-відповідей для перекладу слова та збереженого словникового запису подано у прикладах В.3–В.4. Повні фрагменти програмної реалізації інтерактивних слів, словникової панелі, методів перекладу та збереження слова наведено у додатку Д.

Отже, у межах клієнтської частини LexiRead інтегровано інтерактивний словник, який забезпечує переклад слова без виходу з процесу читання. Реалізація базується на взаємодії ReaderScreen, DictionaryBottomSheet, DictionaryService, ApiService і зовнішнього REST API. Користувач може натиснути на слово, отримати переклад, переглянути додаткову інформацію та зберегти слово у персональний словник. Такий підхід забезпечує цілісну взаємодію між читанням, перекладом і подальшим навчальним опрацюванням лексики.

3.3 Тестування додатку

Тестування є необхідним етапом перевірки працездатності клієнтської частини мобільного додатку LexiRead. Його проведення дає змогу оцінити коректність реалізації основних функціональних модулів, перевірити стабільність роботи інтерфейсу, виявити помилки під час взаємодії з даними та підтвердити відповідність реалізованого функціоналу вимогам, сформованим у першому розділі роботи. Оскільки в межах бакалаврської роботи розробляється саме клієнтська частина, основна увага під час тестування приділяється екранам застосунку, роботі з локальним станом, обробці API-відповідей, імпорту файлів, валідації форм і відображенню станів інтерфейсу [28; 42].

Метою тестування є перевірка того, що клієнтська частина коректно реагує на дії користувача, не блокує інтерфейс під час асинхронних операцій, правильно обробляє дані, отримані з API, та забезпечує передбачувану поведінку у типових і помилкових ситуаціях. У процесі тестування враховано основні сценарії використання додатку: авторизацію, перегляд бібліотеки, додавання книги, читання тексту, отримання перекладу слова, збереження слова, перегляд словника, роботу з флеш-картками, проходження тестів, роботу із закладками та зміну налаштувань. Перевірка API-взаємодії є важливою для клієнт-серверних систем, оскільки помилки у форматі запитів, відповідей або кодах статусу можуть призводити до некоректної роботи інтерфейсу користувача [43].

Тестування клієнтської частини виконувалося на декількох рівнях. Перший рівень охоплює статичний аналіз коду, що дозволяє виявити синтаксичні помилки, некоректне використання типів, невідповідності стилю та потенційні проблеми у Dart-коді. Другий рівень становить автоматизоване тестування окремих компонентів, зокрема валідаторів, моделей, обробки файлів і базових віджетів. Третій рівень передбачає ручне функціональне тестування основних користувацьких сценаріїв, які пов’язані з роботою інтерфейсу та переходами між екранами [41; 42].

Для перевірки якості коду було використано команду dart analyze. Її виконання дозволяє перевірити проєкт на наявність помилок аналізатора, проблем типізації та попереджень, які можуть впливати на стабільність роботи застосунку. За результатами перевірки не було виявлено помилок аналізу коду, що свідчить про коректність структури Dart-файлів і відповідність базовим вимогам статичного аналізу.

Результат виконання статичного аналізу доцільно подати на рисунку 3.11.

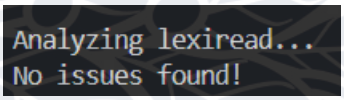

Рисунок 3.11 – Результат виконання команди dart analyze

Автоматизоване тестування виконувалося засобами Flutter Test. У проєкті передбачено тести для перевірки окремих частин клієнтської логіки. Зокрема, перевірялися валідатори форм, обробка текстових файлів, витягування тексту з PDF, обробка багатосторінкових PDF-документів, реакція на некоректний PDF-файл, збереження структури параграфів і розділів EPUB, перетворення JSON-даних у моделі флеш-карток, перетворення даних у модель питання тесту та базове відкриття екрана авторизації [27; 28]. Приклади unit-тестів для FileTextExtractor та widget-тесту початкового екрана подано у додатку Д.

Під час запуску автоматизованих тестів використовувалася команда flutter test --no-pub. За результатами виконання тестів підтверджено проходження 12 тестових перевірок. Це свідчить про коректність реалізації окремих компонентів клієнтської частини, однак не означає повного автоматизованого покриття всіх користувацьких сценаріїв. Окремі integration tests у поточній версії проєкту не реалізовувалися, тому комплексна перевірка наскрізних сценаріїв виконувалася вручну. Результат виконання автоматизованих тестів наведено на рисунку 3.12.

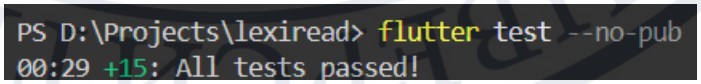

Рисунок 3.12 – Результат виконання команди flutter test --no-pub

Загальну характеристику автоматизованих перевірок наведено в таблиці Е.2.

Окрім автоматизованих перевірок, було проведено ручне функціональне тестування основних сценаріїв взаємодії користувача із застосунком. Такий підхід є доцільним для клієнтської частини мобільного додатку, оскільки значна частина функціоналу пов’язана з інтерфейсом, навігацією, візуальними станами та реакцією застосунку на дії користувача.

Під час ручного тестування було перевірено авторизацію та реєстрацію користувача, завантаження бібліотеки книг, додавання книги з файлу, відкриття книги на екрані читання, перехід між сторінками, збереження прогресу, створення закладок, натискання на слово в тексті, відкриття словникової панелі, отримання перекладу, збереження слова до персонального словника, перегляд збережених слів, роботу флеш-карток, проходження тестів і зміну налаштувань читання.

Результати ручної перевірки показали, що основні користувацькі сценарії виконуються коректно. Застосунок відображає стани завантаження під час виконання запитів, повідомляє користувача про помилки, коректно обробляє порожні списки та не блокує інтерфейс під час асинхронних операцій. Особливу увагу було приділено перевірці інтерактивного словника, оскільки цей модуль є ключовим для теми роботи. Було підтверджено, що після натискання на слово в ReaderScreen відкривається словникова панель, виконується запит на переклад, результат відображається користувачу, а слово може бути збережене до персонального словника. Розширений перелік ручних тестових сценаріїв клієнтської частини наведено в таблиці Е.1.

Під час тестування особливу увагу приділено перевірці станів інтерфейсу. Для мобільного застосунку, що взаємодіє із зовнішнім API, важливо коректно відображати не лише успішні результати, а й проміжні та помилкові стани. З цією метою перевірялися індикатори завантаження, повідомлення про помилки, порожні списки та реакція інтерфейсу на відсутність даних. Такі перевірки дозволяють переконатися, що користувач отримує зрозумілий зворотний зв’язок під час роботи із застосунком.

Окремим результатом тестування є підтвердження того, що клієнтська частина не блокує інтерфейс під час виконання асинхронних операцій. API-запити, імпорт файлів і отримання словникових даних виконуються без необхідності перезапуску екрана або застосунку. У випадку помилки користувач залишається в поточному сценарії та може повторити дію або повернутися до попереднього екрана.

Водночас слід зазначити, що тестування має певні обмеження. У поточній версії проєкту не реалізовано окремі integration tests, які автоматично перевіряли б повний шлях користувача від авторизації до читання, перекладу, збереження слова та проходження тесту. Також автоматизоване тестування не охоплює всі екрани інтерфейсу. Ці обмеження не впливають на працездатність базового функціоналу, однак визначають напрями подальшого вдосконалення тестової бази.

Отже, у межах тестування клієнтської частини мобільного додатку LexiRead було виконано статичний аналіз коду, автоматизовані перевірки окремих компонентів і ручне функціональне тестування основних користувацьких сценаріїв. Результати перевірки підтвердили коректність роботи основних модулів: авторизації, бібліотеки, імпорту книг, читання, інтерактивного словника, збережених слів, флеш-карток, тестів, закладок і налаштувань. Виявлені обмеження пов’язані переважно з неповним автоматизованим покриттям і можуть бути усунуті під час подальшого розвитку застосунку.

3.4 Оцінювання продуктивності та аналіз результатів роботи клієнтської частини

Після реалізації та функціонального тестування клієнтської частини мобільного додатку було проведено оцінювання результатів її роботи. На відміну від підрозділу 3.3, де основну увагу приділено перевірці коректності окремих функцій, у цьому підрозділі розглянуто швидкодію, стабільність, поведінку

інтерфейсу та придатність застосунку до практичного використання в межах основних користувацьких сценаріїв.

Метою оцінювання було визначення того, наскільки реалізована клієнтська частина відповідає нефункціональним вимогам, пов’язаним із відкриттям електронних книг, роботою ReaderScreen, швидкістю відкриття DictionaryBottomSheet, виконанням запитів до REST API, відображенням словникових записів, навігацією між екранами та стабільністю інтерфейсу під час тривалого використання.

Оцінювання проводилося шляхом ручного тестування застосунку на Android-пристрої. Під час перевірки виконувалися типові дії користувача: запуск застосунку, авторизація, відкриття бібліотеки, імпорт електронної книги, перехід до екрана читання, натискання на слова в тексті, отримання перекладу через REST API, збереження слів до словника, перегляд словникових записів, відкриття флеш-карток, проходження тесту та зміна параметрів читання. Для точнішого кількісного профілювання в подальшому доцільно використовувати release-збірку застосунку разом з Android Studio Profiler.

Під час перевірки використовувалися EPUB-файли малого, середнього та великого обсягу. Для цього формату некоректно фіксувати сталу кількість сторінок, оскільки кількість сторінок залежить від розміру екрана, шрифту, міжрядкового інтервалу та параметрів пагінації. Тому в межах оцінювання враховувався саме умовний обсяг текстового вмісту та поведінка ReaderScreen після імпорту книги.

Результати таблиці 3.3 свідчать, що клієнтська частина LexiRead коректно виконує основні сценарії використання. Найстабільнішими є переходи між екранами, відкриття словникової панелі, зміна налаштувань і робота з невеликими та середніми текстовими матеріалами. Найбільше навантаження виникає під час відкриття великих EPUB-файлів, оскільки застосунок має витягнути текст, підготувати його до відображення та сформувати структуру для посторінкового читання.

Таблиця 3.3 — Результати ручного оцінювання основних сценаріїв роботи мобільного додатку

Сценарій перевірки	**Умова виконання**	**Результат ручного тестування**	**Інтерпретація результату**
Відкриття EPUB-книги малого обсягу	Імпорт і відкриття короткого текстового матеріалу	Книга відкривається без помітної затримки	Застосунок коректно обробляє невеликі файли та швидко переходить до ReaderScreen
Відкриття EPUB-книги середнього обсягу	Імпорт і відкриття книги з більшим текстовим вмістом	Спостерігається коротка затримка під час підготовки тексту	Затримка є прийнятною для сценарію читання та не призводить до зависання інтерфейсу
Відкриття EPUB-книги великого обсягу	Імпорт і відкриття великого текстового файлу	Час попередньої обробки збільшується	Великі книги створюють більше навантаження на клієнтську частину та потребують подальшої оптимізації
Відкриття DictionaryBottomSheet	Натискання на слово в ReaderScreen	Нижня словникова панель відкривається одразу після дії користувача	Інтерфейс швидко реагує на натискання та не блокує процес читання
Отримання перекладу через REST API	Надсилання запиту до /translate	Під час очікування відповіді відображається стан завантаження	Затримка залежить не лише від Flutter-клієнта, а й від мережі та швидкості відповіді API
Збереження слова до словника	Натискання кнопки «У словник»	Після успішного запиту користувач отримує підтвердження	Сценарій збереження слова працює як продовження словникової взаємодії
Перехід між екранами	LibraryScreen, ReaderScreen, DictionaryScreen, SettingsScreen	Переходи виконуються без помітного зависання	Навігаційна структура є достатньо стабільною для базового використання
Зміна параметрів читання	Зміна теми, розміру шрифту, мовних параметрів	Налаштування застосовуються без перезапуску застосунку	Використання SettingsService і ValueNotifier забезпечує коректне оновлення інтерфейсу

Оцінювання проводилося шляхом ручного тестування застосунку на Android-пристрої. Під час перевірки виконувалися типові дії користувача: запуск

застосунку, авторизація, відкриття бібліотеки, імпорт електронної книги, перехід до екрана читання, натискання на слова в тексті, отримання перекладу через REST API, збереження слів до словника, перегляд словникових записів, відкриття флеш-карток, проходження тесту та зміна параметрів читання. Для точнішого кількісного профілювання в подальшому доцільно використовувати release-збірку застосунку разом з Android Studio Profiler [44].

Для оцінювання масштабованості словникового модуля було розглянуто змодельований сценарій роботи з різною кількістю збережених слів. Такий сценарій дає змогу визначити потенційні обмеження екрана DictionaryScreen у разі активного використання застосунку протягом тривалого часу. Результати подано в таблиці 3.4.

Таблиця 3.4 — Оцінювання впливу кількості збережених слів на роботу DictionaryScreen

Кількість збережених слів	**Характер сценарію**	**Очікувана поведінка інтерфейсу**	**Потенційна проблема**
До 1000 записів	Типовий користувацький словник	Список відкривається без помітної затримки	Критичних обмежень не виявлено
Близько 3000 записів	Активне використання словника	Можлива коротка затримка під час першого завантаження	Зростає навантаження на побудову списку
Близько 5000 записів	Змодельований навантажувальний сценарій	Можлива помітніша затримка під час відкриття екрана	Великий список може впливати на плавність першого рендерингу
Понад 5000 записів	Розширений навантажувальний сценарій	Імовірне зростання часу відкриття та використання пам’яті	Поточна реалізація може бути недостатньою для дуже великих словників

Таблиця 3.4 показує, що поточна реалізація DictionaryScreen є придатною для типового сценарію, коли користувач зберігає обмежену кількість слів. Водночас у разі накопичення кількох тисяч записів може виникнути потреба в додатковій оптимізації. Основним напрямом такої оптимізації є посторінкове завантаження словникових записів або ліниве підвантаження елементів під час прокручування. Це дозволить зменшити навантаження на інтерфейс і зробити час відкриття словника менш залежним від загальної кількості збережених слів.

Стабільність застосунку оцінювалася під час тривалішого ручного використання. У межах перевірки виконувалися повторні переходи між LibraryScreen, ReaderScreen, DictionaryScreen, FlashcardsScreen, QuizScreen і SettingsScreen, відкривалися словникові панелі, змінювалися параметри читання та виконувалися запити до API. Під час такого тестування не було виявлено аварійного завершення застосунку в базових сценаріях. Інтерфейс залишався керованим, а стани завантаження, помилки та відсутності даних відображалися коректно.

Плавність роботи інтерфейсу оцінювалася за поведінкою екранів під час навігації, прокручування списків, відкриття словникової панелі та посторінкового читання. У звичайних сценаріях взаємодії інтерфейс працював стабільно. Найбільше навантаження спостерігалося під час попередньої обробки великих книг і потенційно може виникати під час відкриття дуже великих словникових списків. Такі обмеження не блокують базове використання застосунку, але визначають напрями подальшого розвитку.

Проведене оцінювання також підтвердило важливість коректної обробки станів інтерфейсу. У застосунку передбачено стани завантаження, успішного отримання даних, помилки та порожнього списку. Це особливо важливо для сценаріїв, які залежать від REST API: авторизації, отримання книг, перекладу слова, збереження лексики, відкриття флеш-карток і закладок. Завдяки цьому користувач отримує зрозумілий зворотний зв’язок і не сприймає затримку відповіді як зависання застосунку.

Результати ручного тестування показали, що клієнтська частина LexiRead загалом відповідає основним нефункціональним вимогам у межах базових сценаріїв використання. Застосунок забезпечує відкриття електронних книг, посторінкове читання, швидке відкриття DictionaryBottomSheet, отримання перекладу через REST API, збереження слів, перегляд словника, повторення лексики за допомогою флеш-карток і проходження тестів. При цьому серверна частина розглядається лише як зовнішній REST API, а оцінювання стосується саме поведінки Flutter-клієнта.

Основними обмеженнями поточної реалізації є залежність словникового модуля від доступності мережі та REST API, відсутність повноцінного офлайн-режиму, збільшення часу попередньої обробки великих EPUB-файлів і потенційне зниження швидкодії DictionaryScreen при дуже великій кількості словникових записів. Ці обмеження не перешкоджають практичному використанню застосунку в базовому сценарії, однак мають бути враховані під час подальшого розвитку.

Отже, аналіз результатів роботи показав, що розроблена клієнтська частина мобільного додатку LexiRead є працездатною, стабільною в основних сценаріях і придатною до практичного використання як мобільний Flutter-клієнт для читання іншомовних текстів із вбудованою словниковою підтримкою. Подальше вдосконалення доцільно спрямувати на оптимізацію роботи з великими EPUB-файлами, реалізацію пагінації для словникових записів, розширення локального кешування та проведення точного профілювання продуктивності за допомогою спеціалізованих інструментів.

ВИСНОВКИ

У межах бакалаврської роботи вирішено задачу розробки клієнтської частини мобільного додатку LexiRead для читання електронних книг із вбудованим інтерактивним словником. Актуальність роботи зумовлена відсутністю нативного Android-рішення, що поєднує зручне читання іншомовних текстів, контекстний переклад слова та систематичне повторення нової лексики в межах єдиного застосунку.

У першому розділі проаналізовано предметну область мобільних додатків для читання, розглянуто роль інтерактивного словника у процесі роботи з іншомовними текстами та виконано порівняльний аналіз існуючих рішень (Amazon Kindle, Google Play Books, LingQ, Readlang). Аналіз виявив, що жоден із розглянутих застосунків не забезпечує повного поєднання читання, контекстного перекладу і систематичного повторення лексики в нативному Android-середовищі. На основі цього сформовано функціональні та нефункціональні вимоги до клієнтської частини LexiRead.

У другому розділі було обґрунтовано вибір технологій і засобів розробки клієнтської частини. Для реалізації мобільного додатку обрано Flutter і мову програмування Dart, що дозволило створити Android-клієнт із використанням віджетної моделі побудови інтерфейсу. Оформлення інтерфейсу виконано на основі Material Design, HTTP-запити реалізовано за допомогою пакета http, локальне збереження токена й налаштувань організовано через shared_preferences, вибір файлів книг забезпечено засобами file_picker, а витягування тексту з PDF-документів виконано з використанням syncfusion_flutter_pdf. Також було спроєктовано структуру Flutter-проєкту, навігацію між екранами, модель взаємодії екранів із сервісами та механізм обробки JSON-відповідей, отриманих від зовнішнього REST API.

Особливу увагу в роботі приділено проєктуванню взаємодії з інтерактивним словником. Було визначено, що словниковий сценарій має виконуватися без виходу з екрана читання. Для цього у клієнтській частині передбачено взаємодію між ReaderScreen, нижньою словниковою панеллю,

сервісом словника та зовнішнім REST API. Користувач натискає на слово у тексті книги, після чого відкривається словникова панель, виконується запит на переклад, результат відображається в інтерфейсі, а слово за потреби може бути збережене до персонального словника. Такий сценарій поєднує читання, переклад і подальше навчальне опрацювання лексики в межах одного мобільного застосунку.

У третьому розділі описано реалізацію функціональних модулів клієнтської частини: авторизації, бібліотеки книг, імпорту файлів (TXT, PDF, EPUB, DOCX), читання, інтерактивного словника, збережених слів, флеш-карток, тестування, закладок і налаштувань. Реалізовано взаємодію з 18 endpoints зовнішнього REST API. Словниковий модуль інтегровано безпосередньо в ReaderScreen через механізм TextSpan і TapGestureRecognizer, що забезпечує переклад слова без виходу з екрана читання. Проведено статичний аналіз коду (dart analyze — No issues found), автоматизоване тестування 12 компонентів (flutter test — 12 passed) та ручне функціональне тестування 15 основних сценаріїв.

У клієнтській частині реалізовано взаємодію із зовнішнім REST API. Через API виконуються запити, пов’язані з авторизацією, отриманням бібліотеки книг, додаванням і редагуванням книг, перекладом слова, збереженням словникових записів, роботою із закладками, прогресом читання та флеш-картками. Отримані JSON-відповіді перетворюються у відповідні Dart-моделі, після чого дані відображаються на екранах застосунку. У межах роботи не розроблялися серверна частина, база даних або внутрішня backend-логіка; вони розглядалися лише як зовнішній REST API, з яким взаємодіє Flutter-клієнт.

Реалізований інтерактивний словник забезпечує переклад слова без переходу до іншого екрана або стороннього застосунку. Після отримання перекладу користувач може зберегти слово до персонального словника. Надалі збережена лексика використовується в навчальних модулях: флеш-картках і тестах. Таким чином, у додатку реалізовано послідовний сценарій роботи з іншомовним текстом: додавання книги, читання, вибір незнайомого слова,

отримання перекладу, збереження слова та його повторення. Це відповідає поставленим функціональним вимогам і демонструє зв’язок між читацьким і навчальним компонентами застосунку.

У роботі також реалізовано налаштування інтерфейсу та читання. Користувач може змінювати тему оформлення, обирати світлий або темний режим, налаштовувати параметри відображення тексту, мови оригіналу та перекладу. Для керування локальним станом екранів використано setState, а для глобальних параметрів налаштувань — ValueNotifier. Локально зберігаються токен авторизації, параметри читання, мовні налаштування та допоміжні дані. Повноцінний офлайн-режим у поточній версії не реалізовано, оскільки переклад слів і частина функцій залежать від доступності зовнішнього REST API.

Для перевірки працездатності клієнтської частини було проведено статичний аналіз коду, автоматизоване unit- та widget-тестування, а також ручну перевірку основних користувацьких сценаріїв. Під час тестування перевірялися валідація форм, імпорт текстових файлів, обробка PDF і EPUB, перетворення JSON-даних у Dart-моделі, відкриття початкового екрана, робота основних модулів, стани завантаження, помилки та відсутності даних. Результати тестування підтвердили коректність роботи основного функціоналу клієнтської частини. Водночас integration tests у поточній версії не реалізовано, що визначає один із напрямів подальшого вдосконалення тестової бази.

Практичне значення отриманих результатів полягає у створенні клієнтської частини мобільного додатку, яка може використовуватися для читання електронних книг іноземною мовою, швидкого отримання перекладу слів, формування персонального словника та повторення лексики за допомогою флеш-карток і тестів. Розроблений Flutter-клієнт може бути використаний як основа для подальшого розвитку комплексної системи читання та підтримки вивчення іноземної лексики. Перспективами розвитку є розширення автоматизованого тестування, оптимізація роботи з великими файлами, удосконалення обробки PDF-документів, покращення збереження даних на клієнтському рівні та розширення навчальних сценаріїв.

Поставлену мету бакалаврської роботи досягнуто: розроблено клієнтську частину мобільного додатку LexiRead засобами Flutter і Dart, що забезпечує читання електронних книг, інтерактивний переклад слова безпосередньо у процесі читання, збереження нової лексики, роботу з флеш-картками, тестування, закладки та налаштування інтерфейсу. Реалізований програмний продукт відповідає сформованим вимогам і демонструє практичне застосування Flutter-стеку для побудови клієнтської частини мобільного застосунку з інтеграцією зовнішнього REST API.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Clinton V. Reading from paper compared to screens: A systematic review and meta-analysis. Journal of Research in Reading. 2019. Vol. 42, No. 2. P. 288–325. DOI: 10.1111/1467-9817.12269.
2. Baron N. S. Words Onscreen: The Fate of Reading in a Digital World. Oxford: Oxford University Press, 2015. 320 p.
3. Perrin A. Three-in-ten Americans now read e-books. Pew Research Center. 2022. URL: https://www.pewresearch.org/internet/2022/01/06/three-in-ten-americans-now-read-e-books/ (дата звернення: 18.05.2026).
4. Garrish M., Gylling M. EPUB 3 Best Practices. Sebastopol : O'Reilly Media, 2013. 372 p.
5. Kukulska-Hulme A. Mobile-assisted language learning revisited. The Cambridge Handbook of Technology and Language Learning / ed. M. J. Thomas, H. Reinders, M. Warschauer. Cambridge : Cambridge University Press, 2020. P. 217–238.
6. Webb S., Nation I. S. P. How Vocabulary Is Learned. Oxford : Oxford University Press, 2017. 336 p.
7. Lin J.-J., Lin H. Mobile-assisted ESL/EFL vocabulary learning: a systematic review and meta-analysis. Computer Assisted Language Learning. 2019. Vol. 32, No. 8. P. 878–919. DOI: 10.1080/09588221.2018.1541359.
8. Mahdi H. S. Effectiveness of mobile devices on vocabulary learning: A meta-analysis. Journal of Educational Computing Research. 2018. Vol. 56, No. 1. P. 134–154. DOI: 10.1177/0735633117698826.
9. Bogner J., Kotstein S., Pfaff T. Do RESTful API Design Rules Have an Impact on the Understandability of Web APIs? A Web-Based Experiment with API Descriptions. arXiv preprint. 2023. URL: https://arxiv.org/abs/2305.07346 (дата звернення: 19.05.2026).
10. Fielding R., Nottingham M., Reschke J. HTTP Semantics. RFC 9110. Internet Engineering Task Force, 2022. URL: https://www.rfc-editor.org/rfc/rfc9110 (дата звернення: 20.05.2026).

11. Nation I. S. P. Learning Vocabulary in Another Language. 2nd ed. Cambridge: Cambridge University Press, 2013. 624 p.
12. Kim Y., Webb S. The effects of spaced practice on second language learning: A meta-analysis. Language Learning. 2022. Vol. 72, No. 1. P. 269–319. DOI: 10.1111/lang.12479.
13. ISO/IEC/IEEE 29148:2018. Systems and software engineering — Life cycle processes — Requirements engineering. International Organization for Standardization, 2018. URL: https://www.iso.org/standard/72089.html (дата звернення: 20.05.2026).
14. ISO 9241-11:2018. Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts. International Organization for Standardization, 2018. URL: https://www.iso.org/standard/63500.html (дата звернення: 20.05.2026).
15. Sommerville I. Engineering Software Products: An Introduction to Modern Software Engineering. Boston : Pearson, 2019. 352 p.
16. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. New York : McGraw-Hill Education, 2020. 705 p.
17. Flutter documentation. URL: https://docs.flutter.dev/ (дата звернення: 21.05.2026).
18. Guide to app architecture. Android Developers. URL: https://developer.android.com/topic/architecture (дата звернення: 21.05.2026).
19. Architectural overview. Flutter documentation. URL: https://docs.flutter.dev/resources/architectural-overview (дата звернення: 21.05.2026).
20. Dart documentation. URL: https://dart.dev/docs (дата звернення: 24.05.2026).
21. shared_preferences package. Dart package repository. URL: https://pub.dev/packages/shared_preferences (дата звернення: 22.05.2026).

22. App data and files. Android Developers. URL: https://developer.android.com/training/data-storage (дата звернення: 22.05.2026).
23. http package. Dart package repository. URL: https://pub.dev/packages/http (дата звернення: 22.05.2026).
24. file_picker package. Dart package repository. URL: https://pub.dev/packages/file_picker (дата звернення: 22.05.2026).
25. syncfusion_flutter_pdf package. Dart package repository. URL: https://pub.dev/packages/syncfusion_flutter_pdf (дата звернення: 22.05.2026).
26. Flutter state management. Flutter documentation. URL: https://docs.flutter.dev/data-and-backend/state-mgmt/intro (дата звернення: 23.05.2026).
27. Material Design. URL: https://m3.material.io/ (дата звернення: 23.05.2026).
28. Testing Flutter apps. Flutter documentation. URL: https://docs.flutter.dev/testing/overview (дата звернення: 23.05.2026).
29. An introduction to unit testing. Flutter documentation. URL: https://docs.flutter.dev/cookbook/testing/unit/introduction (дата звернення: 23.05.2026).
30. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.
31. Гончар А. А., Бацєнко Т. В. Проєктування клієнтської частини мобільного додатку для читання з функцією інтерактивного словника. Донецький національний університет імені Василя Стуса. 2026.
32. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. Internet Engineering Task Force, 2015. URL: https://www.rfc-editor.org/rfc/rfc7519 (дата звернення: 23.05.2026).

33. Bray T. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 8259. Internet Engineering Task Force, 2017. URL: https://www.rfc-editor.org/rfc/rfc8259 (дата звернення: 23.05.2026).
34. Flutter navigation and routing. Flutter documentation. URL: https://docs.flutter.dev/ui/navigation (дата звернення: 23.05.2026).
35. Гуменюк К. В., Січко Т. В. Передові методи анімації вебінтерфейсів: вплив на користувацький досвід. Прикладні інформаційні технології. 2025. С. 13–15.
36. RichText class. Flutter API documentation. URL: https://api.flutter.dev/flutter/widgets/RichText-class.html (дата звернення: 23.05.2026).
37. TapGestureRecognizer class. Flutter API documentation. URL: https://api.flutter.dev/flutter/gestures/TapGestureRecognizer-class.html (дата звернення: 23.05.2026).
38. Павлов Д. Л., Січко Т. В. Принцип роботи Web API та його застосування. Прикладні інформаційні технології. 2023. С. 166–168.
39. ISO/IEC 25010:2023. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model. International Organization for Standardization, 2023. URL: https://www.iso.org/standard/78176.html (дата звернення: 20.05.2026).
40. ISO 9241-210:2019. Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems. International Organization for Standardization, 2019. URL: https://www.iso.org/standard/77520.html (дата звернення: 20.05.2026).
41. dart analyze. Dart documentation. URL: https://dart.dev/tools/dart-analyze (дата звернення: 24.05.2026).
42. ISO/IEC/IEEE 29119-1:2022. Software and systems engineering — Software testing — Part 1: Concepts and definitions. International Organization for Standardization, 2022. URL: https://www.iso.org/standard/81291.html (дата звернення: 20.05.2026).

43. Сіклічук А. С., Січко Т. В. Тестування API як інструмент виявлення помилок у роботі клієнт-серверних систем. Комп'ютерні технології обробки даних. 2024. С. 275–278.
44. Нескородєва Т.В., Федоров Є.Є., Січко Т.В., Нескородєва А.Р. Експертні та рекомендаційні системи: навч. посібник. Вінниця: ДонНУ імені Василя Стуса, 2022, 208 с.

ДОДАТКИ

ДОДАТОК А

Основні екрани мобільного додатку LexiRead

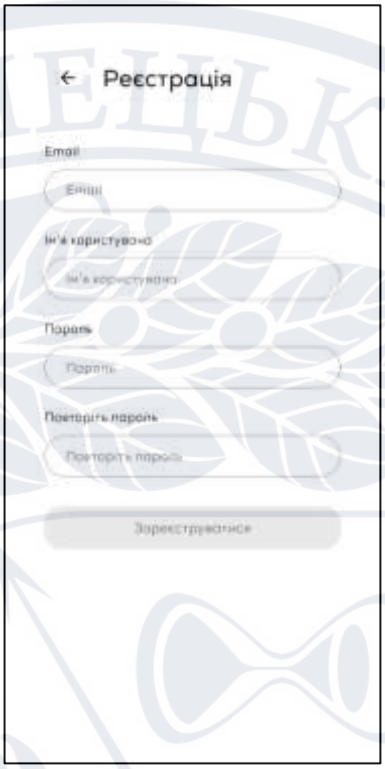

Рисунок А.1 — Екран реєстрації користувача

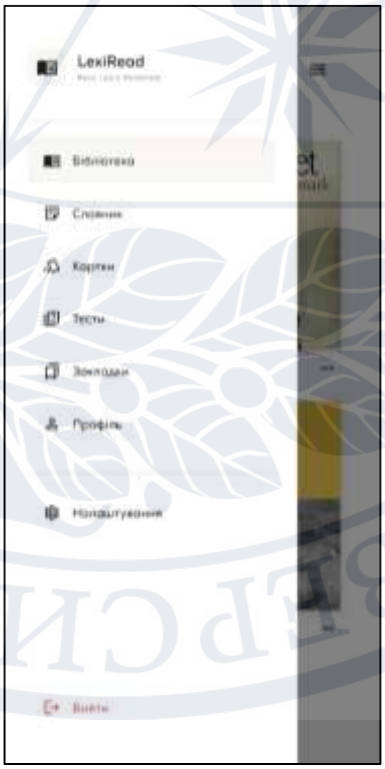

Рисунок А.2 — Навігаційне меню мобільного додатку

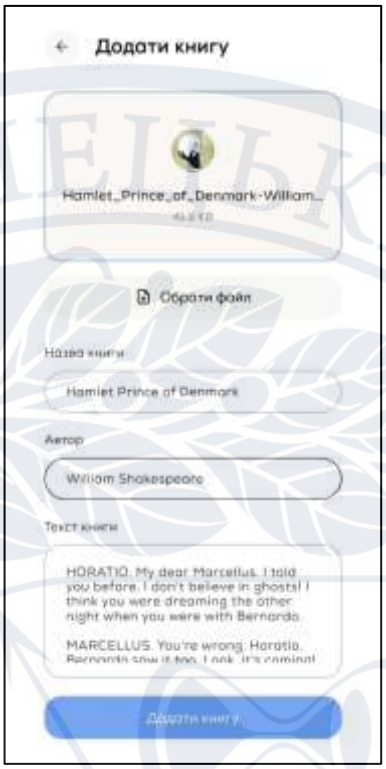

Рисунок А.3 — Екран додавання книги після імпорту файлу

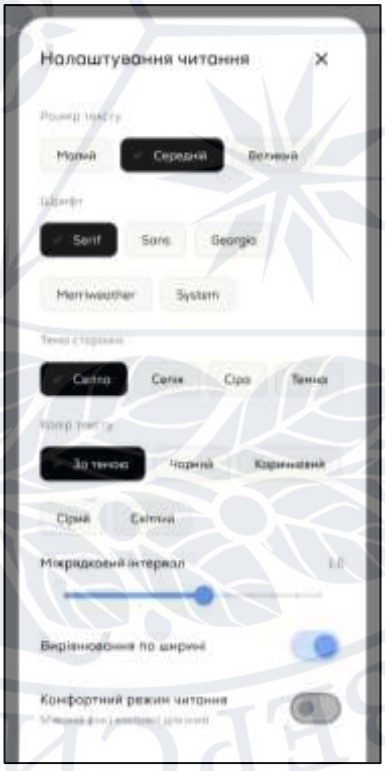

Рисунок А.4 — Панель налаштування параметрів читання

Рисунок А.5 — Екран закладок

Рисунок А.6 — Екран навчальної сесії з флеш-картками

Рисунок А.7 — Екран навчальної сесії з флеш-картками

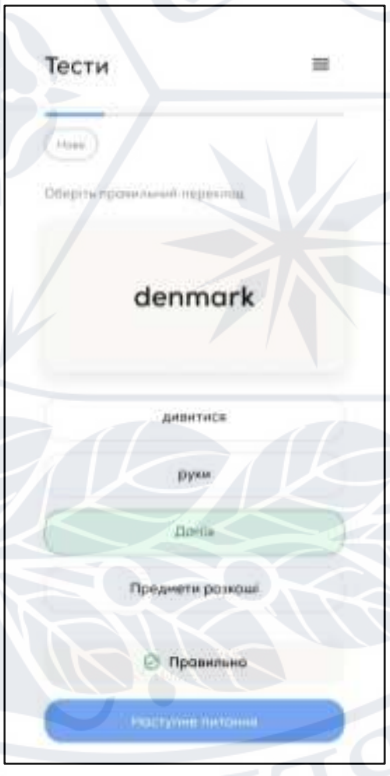

Рисунок А.8 — Екран тестування слів

Рисунок А.9 — Результат проходження тесту

Рисунок А.10 — Повідомлення про помилку під час отримання перекладу

ДОДАТОК Б

Допоміжні схеми проєктування клієнтської частини мобільного додатку

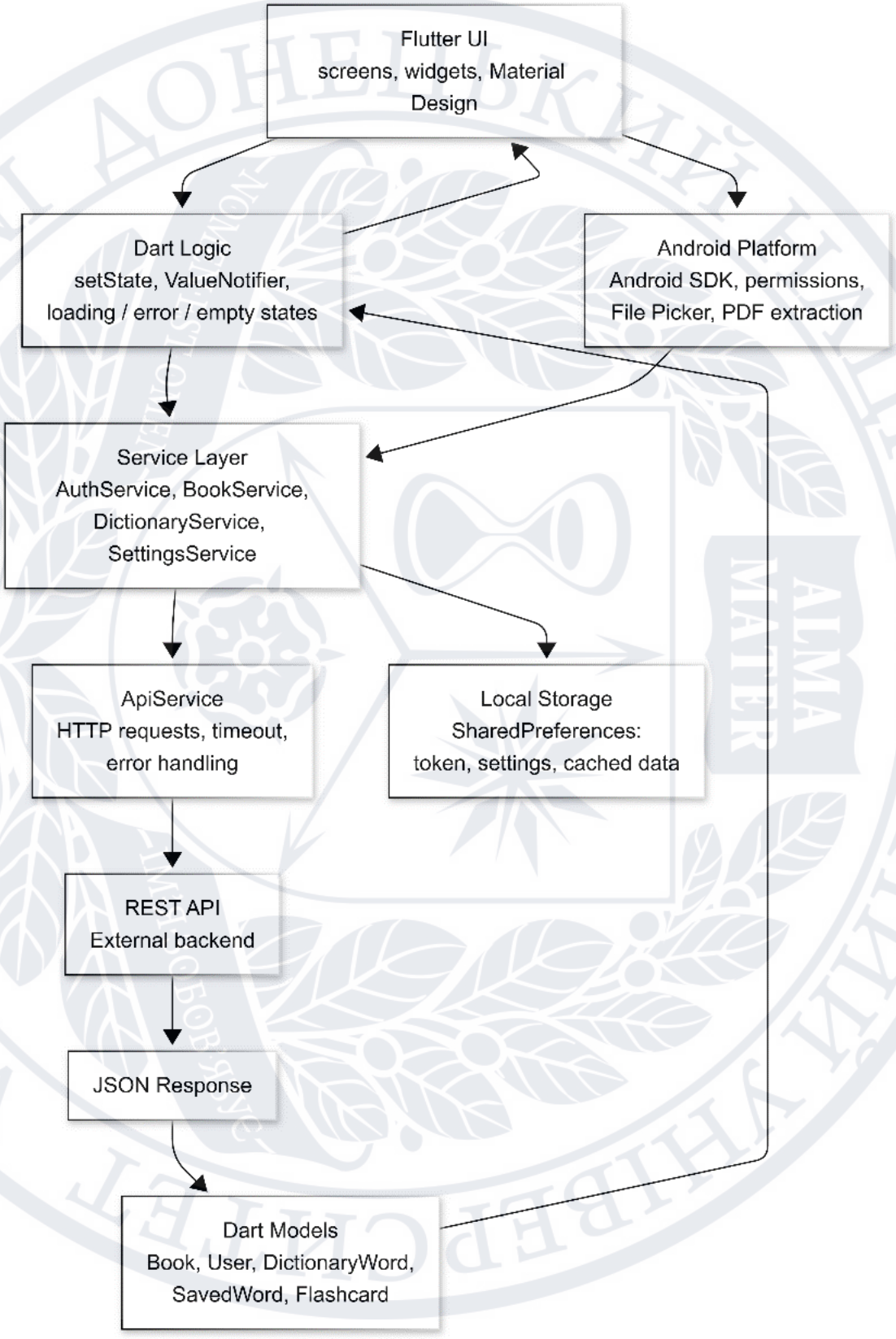

Рисунок Б.1 – Технологічний стек клієнтської частини мобільного додатку

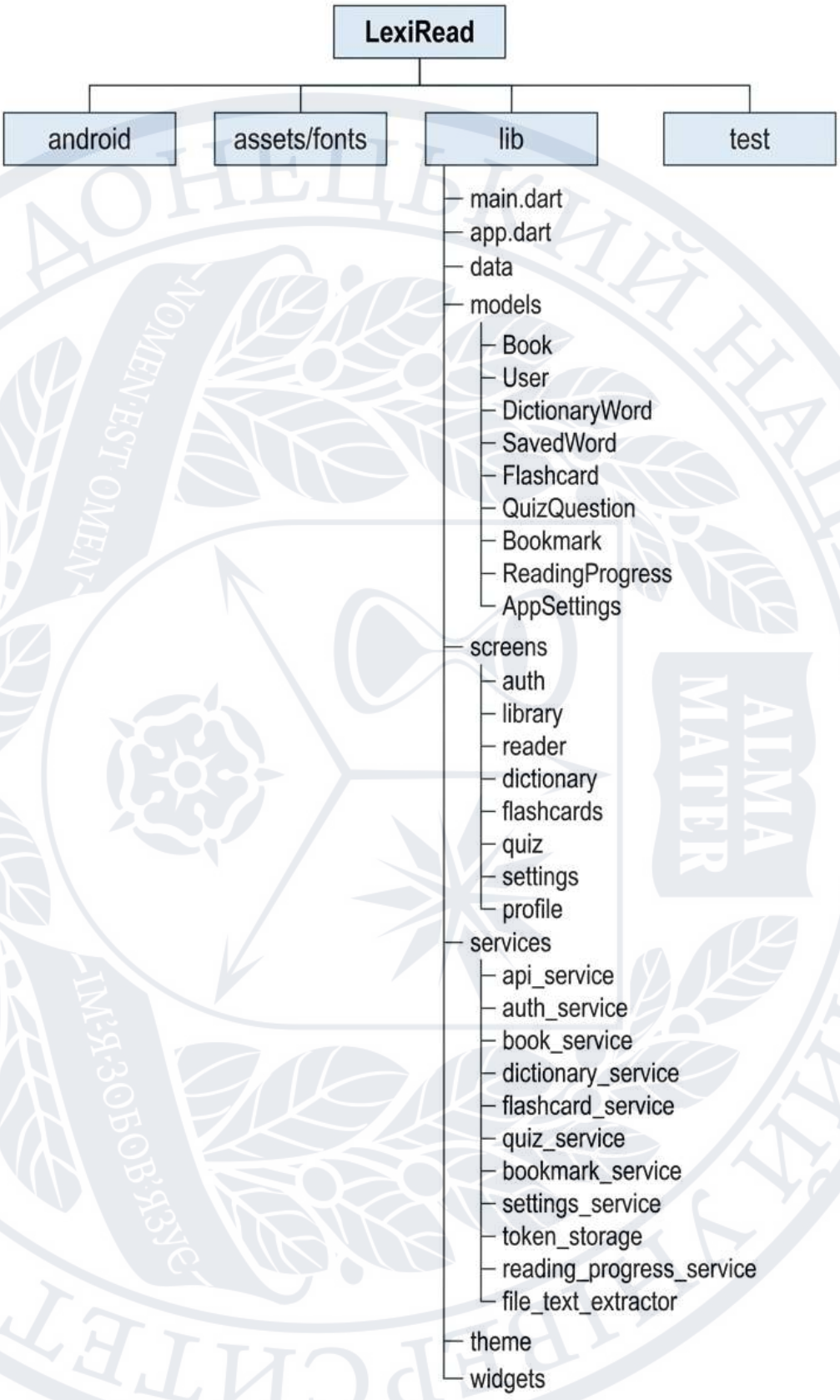

Рисунок Б.2 – Структура Flutter-проєкту LexiRead

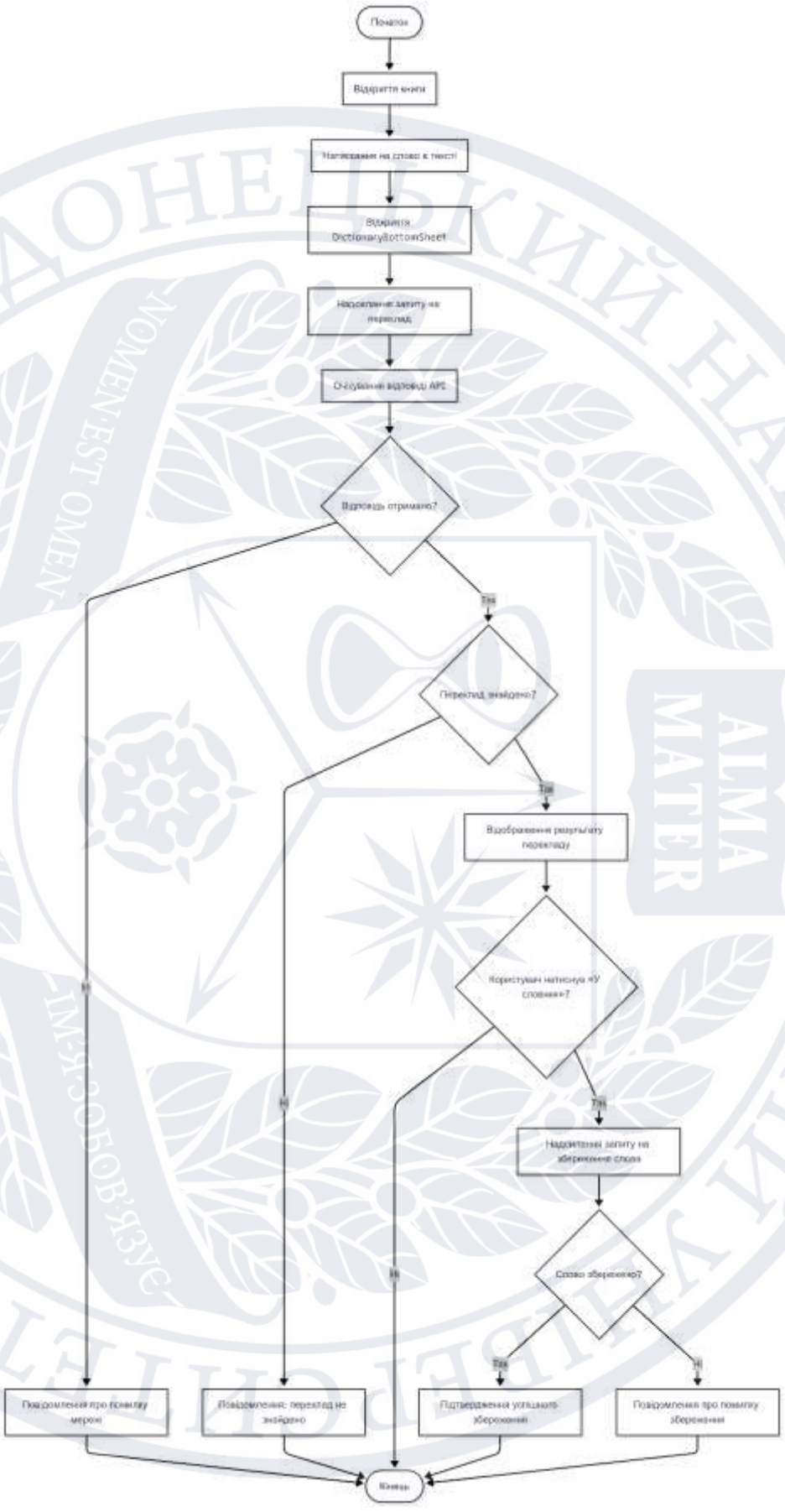

Рисунок Б.3 – Activity diagram сценарію перекладу та збереження слова

ДОДАТОК В

API-взаємодія клієнтської частини із зовнішнім REST API

Таблиця В.1 — Основні REST API-запити клієнтської частини LexiRead

HTTP-метод	**Endpoint**	**Призначення**	**Вхідні параметри**	**Результат**
POST	`/api/auth/login`	Авторизація користувача	email, password	JWT-токен та дані користувача
POST	`/api/auth/register`	Реєстрація нового користувача	username, email, password	Створення облікового запису
GET	`/api/books`	Отримання списку книг	category, search	Масив книг
GET	`/api/books/{id}`	Отримання інформації про книгу	id книги	Дані книги
POST	`/api/books/upload`	Завантаження книги користувачем	файл EPUB/PDF	Інформація про додану книгу
GET	`/api/user/library`	Отримання бібліотеки користувача	token авторизації	Список збережених книг
POST	`/api/bookmarks`	Додавання закладки	bookId, page	Збереження позиції читання
GET	`/api/bookmarks/{bookId}`	Отримання закладок книги	bookId	Масив закладок
PUT	`/api/user/settings`	Оновлення налаштувань користувача	theme, fontSize, language	Оновлені параметри
GET	`/api/statistics`	Отримання статистики читання	token авторизації	Дані про прогрес читання

Таблиця В.2 – API-запити словникового модуля

HTTP-метод	**Endpoint**	**Призначення**	**Вхідні параметри**	**Результат**
POST	`/api/dictionary/translate`	Переклад вибраного слова	word, sourceLanguage, targetLanguage	Переклад слова
GET	`/api/dictionary/definition/{word}`	Отримання визначення слова	word	Тлумачення слова
GET	`/api/dictionary/examples/{word}`	Отримання прикладів використання	word	Список прикладів
POST	`/api/dictionary/save`	Збереження слова до словника	userId, word, translation	Додавання слова
GET	`/api/dictionary/saved`	Отримання списку збережених слів	userId	Масив слів
DELETE	`/api/dictionary/delete/{id}`	Видалення слова зі словника	id слова	Підтвердження видалення
POST	`/api/flashcards/generate`	Генерація флеш-карток	savedWords	Набір карток
GET	`/api/flashcards`	Отримання флеш-карток	userId	Масив флеш-карток

		користувача		
POST	/api/quiz/start	Початок тестування слів	dictionaryId	Дані тесту
POST	/api/quiz/result	Збереження результатів тесту	score, answers	Результати проходження

Приклад В.3 — JSON-відповідь для перекладу слова

```
{
  "word": "flying",
  "translation": "летіти",
  "transcription": "[ˈflaɪɪŋ]",
  "part_of_speech": "verb",
  "example": "Scarves were flying in the icy wind."
}
```

Приклад В.4 — JSON-відповідь для збереженого словникового запису

```
{
  "id": 12,
  "difficulty": "medium",
  "saved_at": "2026-05-19T12:30:00",
  "dictionary_entry": {
    "id": 5,
    "word": "flying",
    "translation": "летіти",
    "source_language": "en",
    "target_language": "uk",
    "part_of_speech": "verb",
    "example": "Scarves were flying in the icy wind."
  }
}
```

ДОДАТОК Д

Фрагменти програмного коду ключових компонентів

Налаштування маршрутів у app.dart:

```
class LexiReadApp extends StatefulWidget {
  const LexiReadApp({super.key});

  @override
  State<LexiReadApp> createState() => _LexiReadAppState();
}

class _LexiReadAppState extends State<LexiReadApp> {
  final _settingsService = SettingsService();

  @override
  void initState() {
    super.initState();
    _settingsService.getSettings();
  }

  @override
  Widget build(BuildContext context) {
    return ValueListenableBuilder<AppSettings>(
      valueListenable: SettingsService.settingsNotifier,
      builder: (context, settings, _) {
        return MaterialApp(
          title: 'LexiRead',
          debugShowCheckedModeBanner: false,
          theme: AppTheme.light,
          darkTheme: AppTheme.dark,
          themeMode:   settings.darkTheme   ?   ThemeMode.dark   :
ThemeMode.light,
          initialRoute: LoginScreen.routeName,
          onGenerateRoute: _onGenerateRoute,
        );
      },
    );
  }

  Route<void> _onGenerateRoute(RouteSettings settings) {
    switch (settings.name) {
      case LoginScreen.routeName:
        return _route(LoginScreen());
      case RegisterScreen.routeName:
        return _route(RegisterScreen());
      case LibraryScreen.routeName:
        return _route(LibraryScreen());
      case AddBookScreen.routeName:
        return _route(AddBookScreen());
```

```
      case EditBookScreen.routeName:
        return  _route(EditBookScreen(book:  settings.arguments  as
Book));
      case BookmarksScreen.routeName:
        return _route(BookmarksScreen());
      case DictionaryScreen.routeName:
        return _route(DictionaryScreen());
      case FlashcardsScreen.routeName:
        return _route(FlashcardsScreen());
      case QuizScreen.routeName:
        return _route(QuizScreen());
      case FlashcardLearningScreen.routeName:
        return _route(FlashcardLearningScreen());
      case ProfileScreen.routeName:
        return _route(ProfileScreen());
      case SettingsScreen.routeName:
        return _route(SettingsScreen());
      case ReaderScreen.routeName:
        final args = settings.arguments;
        if (args is ReaderScreenArgs) {
          return _route(
              ReaderScreen(book:      args.book,      initialPage:
args.initialPage));
        }
        return _route(ReaderScreen(book: args as Book));
      case WordDetailScreen.routeName:
        final word = settings.arguments as DictionaryWord;
        return _route(WordDetailScreen(word: word));
      default:
        return _route(LoginScreen());
    }
  }

  MaterialPageRoute<void> _route(Widget screen) {
    return MaterialPageRoute<void>(builder: (_) => screen);
  }
}
```

Централізована робота з REST API в ApiService:

```
class ApiConfig {
  static const String baseUrl =
      'https://pregnancy-throng-matrix.ngrok-free.dev';

  static Uri uri(String path) => Uri.parse('$baseUrl$path');
}

class ApiException implements Exception {
  const ApiException(this.message, {this.statusCode});

  final String message;
  final int? statusCode;
```

```
  @override
  String toString() => message;
}

class ApiService {
  ApiService({http.Client? client, TokenStorage? tokenStorage})
      : _client = client ?? http.Client(),
        _tokenStorage = tokenStorage ?? TokenStorage();

  final http.Client _client;
  final TokenStorage _tokenStorage;
  static const _requestTimeout = Duration(seconds: 45);
  static const _longRequestTimeout = Duration(seconds: 120);
  static const _retryDelay = Duration(seconds: 2);

  Future<bool> checkBackendConnection() async {
    final uri = ApiConfig.uri('/docs');
    print('BASE URL: ${ApiConfig.baseUrl}');
    print('REQUEST URL: ${uri.toString()}');
    try {
      final response = await _client.get(
        uri,
        headers: const {
          'Accept': 'text/html,application/json',
        },
      ).timeout(_requestTimeout);
      print('URL: ${uri.toString()}');
      print('STATUS: ${response.statusCode}');
      print('BODY: ${_safeLogText(response.body)}');
      return response.statusCode >= 200 && response.statusCode <
500;
    } on SocketException {
      print('REQUEST URL: ${uri.toString()}');
      print('URL: ${uri.toString()}');
      print('STATUS: NETWORK_ERROR');
      print('BODY: Не вдалося підключитися до backend');
      throw const ApiException(
        'Backend недоступний або немає підключення до мережі',
      );
    } on TimeoutException {
      print('REQUEST URL: ${uri.toString()}');
      print('URL: ${uri.toString()}');
      print('STATUS: NETWORK_TIMEOUT');
      print('BODY: Backend відповідає довше, ніж очікувалося');
      throw const ApiException(
        'Backend недоступний або немає підключення до мережі',
      );
    } on FormatException {
      print('REQUEST URL: ${uri.toString()}');
      print('URL: ${uri.toString()}');
      print('STATUS: FORMAT_ERROR');
```

```
      print('BODY: Некоректна адреса backend');
      throw const ApiException('Некоректна адреса backend.');
    } on http.ClientException catch (error) {
      print('REQUEST URL: ${uri.toString()}');
      print('URL: ${uri.toString()}');
      print('STATUS: NETWORK_ERROR');
      print('BODY: ${error.message}');
      throw const ApiException(
        'Backend недоступний або немає підключення до мережі',
      );
    }
  }

  Future<dynamic> get(String path, {bool auth = true, String?
debugLabel}) {
    return _send('GET', path, auth: auth, debugLabel: debugLabel);
  }

  Future<dynamic> post(
    String path, {
    Map<String, dynamic>? body,
    bool auth = true,
    String? debugLabel,
  }) {
    return _send('POST', path, body: body, auth: auth, debugLabel:
debugLabel);
  }

  Future<dynamic> postForm(
    String path, {
    required Map<String, String> body,
    bool auth = true,
    String? debugLabel,
  }) {
    return _send(
      'POST_FORM',
      path,
      body: body,
      auth: auth,
      debugLabel: debugLabel,
    );
  }

  Future<dynamic> put(
    String path, {
    Map<String, dynamic>? body,
    bool auth = true,
    String? debugLabel,
  }) {
    return _send('PUT', path, body: body, auth: auth, debugLabel:
debugLabel);
  }
```

```
  Future<dynamic> delete(String path, {bool auth = true, String?
debugLabel}) {
    return _send('DELETE', path, auth: auth, debugLabel:
debugLabel);
  }

  Future<dynamic> _send(
    String method,
    String path, {
    Map<String, dynamic>? body,
    bool auth = true,
    String? debugLabel,
  }) async {
    final uri = ApiConfig.uri(path);
    final token = await _tokenStorage.getToken();

    print('BASE URL: ${ApiConfig.baseUrl}');
    print('REQUEST URL: ${uri.toString()}');
    print('TOKEN: ${_maskToken(token)}');

    final headers = <String, String>{
      'Content-Type': method == 'POST_FORM'
          ? 'application/x-www-form-urlencoded'
          : 'application/json',
      'Accept': 'application/json',
    };
    if (ApiConfig.baseUrl.contains('ngrok')) {
      headers['ngrok-skip-browser-warning'] = 'true';
    }
    if (auth) {
      if (token == null || token.isEmpty) {
        print('REQUEST URL: ${uri.toString()}');
        print('URL: ${uri.toString()}');
        print('STATUS: 401');
        print('BODY: Missing access token');
        throw const ApiException(
          'Потрібно увійти в акаунт',
          statusCode: 401,
        );
      }
      headers['Authorization'] = 'Bearer $token';
    }

    try {
      final response = await _sendWithRetry(
        method: method,
        uri: uri,
        path: path,
        headers: headers,
        body: body,
      );
```

```
      _logResponse(uri, response);
      if (debugLabel != null && debugLabel.trim().isNotEmpty) {
        final label = debugLabel.trim();
        print('$label URL: ${uri.toString()}');
        if (body != null && !_isSensitiveDebugLabel(label)) {
          print('$label                    REQUEST                    BODY:
${_safeLogText(jsonEncode(body))}');
        }
        print('$label STATUS: ${response.statusCode}');
        print('$label BODY: ${_safeLogText(response.body)}');
      }
      if (method == 'POST' && path == '/books') {
        print('CREATE BOOK STATUS: ${response.statusCode}');
        print('CREATE BOOK BODY: ${_safeLogText(response.body)}');
      }

      if (response.statusCode == 204) return null;

      final responseText = utf8.decode(response.bodyBytes);
      final decoded = _decodeJsonOrNull(responseText);

      if (response.statusCode < 200 || response.statusCode >= 300)
{
        print('API ERROR: ${_safeLogText(responseText)}');
        throw ApiException(
          _isTunnelOffline(responseText)
              ? 'Backend недоступний або тунель offline'
              : responseText.isNotEmpty
                  ? responseText
                  :          _messageForStatus(response.statusCode,
decoded),
          statusCode: response.statusCode,
        );
      }

      if (decoded == null && responseText.trim().isNotEmpty) {
        print('REQUEST URL: ${uri.toString()}');
        print('STATUS: FORMAT_ERROR');
        print('BODY: Backend повернув некоректну відповідь');
        throw  const  ApiException('Backend  повернув  некоректну
відповідь.');
      }

      return decoded;
    } on SocketException {
      print('REQUEST URL: ${uri.toString()}');
      print('URL: ${uri.toString()}');
      print('STATUS: NETWORK_ERROR');
      print('BODY: Не вдалося підключитися до backend');
      throw const ApiException(
        'Backend недоступний або немає підключення до мережі',
```

```
      );
    } on TimeoutException {
      print('REQUEST URL: ${uri.toString()}');
      print('URL: ${uri.toString()}');
      print('STATUS: NETWORK_TIMEOUT');
      print('BODY: Backend відповідає довше, ніж очікувалося');
      throw const ApiException(
        'Backend відповідає занадто довго. Спробуйте ще раз.',
      );
    } on FormatException {
      print('REQUEST URL: ${uri.toString()}');
      print('URL: ${uri.toString()}');
      print('STATUS: FORMAT_ERROR');
      print('BODY: Backend повернув некоректну відповідь');
      throw const ApiException(
        'Backend повернув некоректну відповідь.',
      );
    } on http.ClientException catch (error) {
      print('REQUEST URL: ${uri.toString()}');
      print('URL: ${uri.toString()}');
      print('STATUS: NETWORK_ERROR');
      print('BODY: ${error.message}');
      throw const ApiException(
        'Backend недоступний або немає підключення до мережі',
      );
    }
  }

  void _logResponse(Uri uri, http.Response response) {
    print('REQUEST URL: ${uri.toString()}');
    print('URL: ${uri.toString()}');
    print('STATUS: ${response.statusCode}');
    print('BODY: ${_safeLogText(response.body)}');
  }

  String _safeLogText(String value) {
    const maxLogCharacters = 4000;
    if (value.length <= maxLogCharacters) return value;
    return '${value.substring(0, maxLogCharacters)}... '
        '[truncated log, total ${value.length} chars]';
  }

  String _maskToken(String? token) {
    if (token == null || token.isEmpty) return '';
    if (token.length <= 12) return '***';
    return                                    '${token.substring(0,
6)}...${token.substring(token.length - 4)}';
  }

  Future<http.Response> _sendWithRetry({
    required String method,
    required Uri uri,
```

```
    required String path,
    required Map<String, String> headers,
    required Map<String, dynamic>? body,
  }) async {
    final maxAttempts = _canRetry(method, path) ? 2 : 1;
    for (var attempt = 1; attempt <= maxAttempts; attempt++) {
      try {
        return await _performRequest(
          method: method,
          uri: uri,
          headers: headers,
          body: body,
        ).timeout(_timeoutFor(method, path));
      } on TimeoutException {
        if (attempt >= maxAttempts) rethrow;
        print('REQUEST RETRY: ${uri.toString()} attempt ${attempt +
1}');
        await Future<void>.delayed(_retryDelay);
      } on SocketException {
        if (attempt >= maxAttempts) rethrow;
        print('REQUEST RETRY: ${uri.toString()} attempt ${attempt +
1}');
        await Future<void>.delayed(_retryDelay);
      } on http.ClientException {
        if (attempt >= maxAttempts) rethrow;
        print('REQUEST RETRY: ${uri.toString()} attempt ${attempt +
1}');
        await Future<void>.delayed(_retryDelay);
      }
    }
    throw TimeoutException('Request retry failed');
  }

  Future<http.Response> _performRequest({
    required String method,
    required Uri uri,
    required Map<String, String> headers,
    required Map<String, dynamic>? body,
  }) {
    return switch (method) {
      'GET' => _client.get(uri, headers: headers),
      'POST' => _client.post(
          uri,
          headers: headers,
          body: jsonEncode(body ?? {}),
        ),
      'POST_FORM' => _client.post(
          uri,
          headers: headers,
          body: (body ?? {}).map(
            (key, value) => MapEntry(key, value.toString()),
          ),
```

```
        ),
      'PUT' => _client.put(
          uri,
          headers: headers,
          body: jsonEncode(body ?? {}),
        ),
      'DELETE' => _client.delete(uri, headers: headers),
      _ => throw ApiException('Непідтримуваний метод запиту: $method'),
    };
  }

  bool _isSensitiveDebugLabel(String label) {
    final normalized = label.toUpperCase();
    return normalized.contains('LOGIN') || normalized.contains('REGISTER');
  }

  bool _canRetry(String method, String path) {
    if (method == 'GET') return true;
    return method == 'POST' && path == '/translate';
  }

  Duration _timeoutFor(String method, String path) {
    if (path == '/books' && method == 'POST') return _longRequestTimeout;
    if (path.startsWith('/books/') && method == 'PUT') {
      return _longRequestTimeout;
    }
    if (path == '/translate') return _longRequestTimeout;
    return _requestTimeout;
  }

  String _messageForStatus(int statusCode, dynamic decoded) {
    final detail = _extractDetail(decoded);
    if (detail != null && detail.isNotEmpty) return detail;

    return switch (statusCode) {
      401 => 'Сесія закінчилась. Увійди ще раз.',
      404 => 'Дані не знайдено.',
      409 => 'Такий запис уже існує.',
      422 => 'Перевір введені дані.',
      502 => 'Сервіс перекладу тимчасово недоступний.',
      _ => 'Помилка запиту ($statusCode).',
    };
  }

  String? _extractDetail(dynamic decoded) {
    if (decoded is Map<String, dynamic>) {
      final detail = decoded['detail'];
      if (detail is String) return detail;
      if (detail is List) {
```

```
        return detail.map((item) {
          if (item is Map && item['msg'] != null) return
item['msg'].toString();
          return item.toString();
        }).join('\n');
      }
    }
    return null;
  }

  dynamic _decodeJsonOrNull(String responseText) {
    if (responseText.trim().isEmpty) return null;
    try {
      return jsonDecode(responseText);
    } on FormatException {
      return null;
    }
  }

  bool _isTunnelOffline(String responseText) {
    final lower = responseText.toLowerCase();
    return lower.contains('endpoint is offline') ||
        lower.contains('tunnel') && lower.contains('offline') ||
        lower.contains('cloudflare') && lower.contains('offline');
  }
}
```

Обробка натискання на слово в ReaderScreen:

```
@override
  Widget build(BuildContext context) {
    final spans = <InlineSpan>[];
    final matches = RegExp(r'\S+|\s+').allMatches(text);
    for (final match in matches) {
      final token = match.group(0) ?? '';
      if (token.trim().isEmpty) {
        spans.add(TextSpan(text: token));
      } else {
        spans.add(
          TextSpan(
            text: token,
            recognizer: TapGestureRecognizer()
              ..onTap = () => showModalBottomSheet<bool>(
                    context: context,
                    isScrollControlled: true,
                    backgroundColor: Colors.transparent,
                    builder: (_) => DictionaryBottomSheet(
                      selectedWord: token,
                      book: book,
                      paragraphIndex: pageIndex,
                      characterIndex: text.indexOf(token),
                      example: _exampleFor(text, token),
```

```
                    ),
                  ),
            ),
          );
        }
    }
```

Реалізація нижньої словникової панелі DictionaryBottomSheet:

```
class DictionaryBottomSheet extends StatefulWidget {
  const DictionaryBottomSheet({
    required this.selectedWord,
    required this.book,
    required this.example,
    required this.characterIndex,
    this.paragraphIndex,
    super.key,
  });

  final String selectedWord;
  final Book book;
  final String example;
  final int? paragraphIndex;
  final int characterIndex;

  @override
  State<DictionaryBottomSheet>          createState()          =>
_DictionaryBottomSheetState();
}
```

```
Future<void> _translate() async {
    try {
      final word = await _dictionaryService.translateWord(
        rawWord: widget.selectedWord,
        sourceBookId: widget.book.id,
        sourceParagraphIndex: widget.paragraphIndex,
        sourceCharacterIndex: widget.characterIndex,
        example: widget.example,
        sourceLanguage: _sourceLanguage,
        targetLanguage: _targetLanguage,
      );
      if (!mounted) return;
      final translation = word.translation.trim();
      setState(() {
        _word = word.copyWith(
          translation:
              translation.isEmpty || translation == 'translation not
available'
                  ? 'Переклад недоступний'
                  : translation,
          originalLanguage: _sourceLanguage,
```

```
          targetLanguage: _targetLanguage,
        );
        _isLoading = false;
      });
    } on ApiException catch (error) {
      if (!mounted) return;
      setState(() {
        _error = error.message;
        _isLoading = false;
      });
    } catch (_) {
      if (!mounted) return;
      setState(() {
        _error = 'Переклад недоступний';
        _isLoading = false;
      });
    }
  }

  Future<void> _saveWord() async {
    final word = _word;
    if (word == null) return;
    if (!_canPersistWord) {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Спочатку потрібен коректний
переклад')),
      );
      return;
    }
    setState(() => _isSaving = true);
    try {
      final prepared = word.copyWith(
          originalLanguage:    _sourceLanguage,    targetLanguage:
_targetLanguage);
      await _dictionaryService.saveWord(prepared);
      if (!mounted) return;
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Слово додано до словника')),
      );
      Navigator.pop(context, true);
    } on ApiException catch (error) {
      if (!mounted) return;
      ScaffoldMessenger.of(context)
          .showSnackBar(SnackBar(content: Text(error.message)));
    } finally {
      if (mounted) setState(() => _isSaving = false);
    }
  }
}
```

Метод перекладу слова в DictionaryService:

```
Future<DictionaryWord> translateWord({
    required String rawWord,
    int? sourceBookId,
    int? sourceParagraphIndex,
    int? sourceCharacterIndex,
    String? example,
    String sourceLanguage = 'en',
    String targetLanguage = 'uk',
  }) async {
    final normalized = _normalize(rawWord);
    final response = await _api.post(
      '/translate',
      body: {
        'word': normalized.isEmpty ? rawWord : normalized,
        'source_language': sourceLanguage,
        'target_language': targetLanguage,
      },
    );
    final data = _asStringMap(response) ?? {};
    final payload = _translationPayload(data);
    final maps = payload == data ? [data] : [data, payload];
    final fallbackWord = normalized.isEmpty ? rawWord : normalized;
```

Метод збереження слова в DictionaryService:

```
 Future<DictionaryWord> saveWord(
    DictionaryWord word, {
    String? debugLabel,
  }) async {
    final body = word.toSaveJson();
    final translation = (body['translation'] ??
'').toString().trim();
    if (!_isUsefulTranslation(translation)) {
      throw const ApiException('Спочатку потрібен коректний
переклад.');
    }

    print('SAVE WORD BODY: ${jsonEncode(body)}');
    try {
      final response = await _api.post(
        '/dictionary',
        body: body,
        debugLabel: debugLabel ?? 'SAVE WORD',
      );
      final data = _asStringMap(response);
      print('SAVE WORD RESPONSE: ${jsonEncode(data ?? response)}');
      if (data == null) {
        final fallback = word.copyWith(id: _tempId--);
        await _saveLocalExample(fallback);
        return _cacheWord(fallback);
      }
```

```
      final savedJson = _asStringMap(data['saved_word']) ??
          _asStringMap(data['data']) ??
          data;
      final saved = SavedWord.fromJson(savedJson).toDictionaryWord(
        isFavorite:
_favoriteIds.contains(_asSavedWordId(savedJson)),
      );
      final merged = _mergeKnownClientFields(saved, word);
      await _saveLocalExample(merged);
      return _cacheWord(merged);
    } on ApiException catch (error) {
      if (error.statusCode != 409) rethrow;
      final existing = await _findExistingWord(word);
      if (existing != null) {
        final merged = _mergeKnownClientFields(existing, word);
        await _saveLocalExample(merged);
        return _cacheWord(merged);
      }
      rethrow;
    }
  }
```

Приклад перетворення JSON у Dart-модель через fromJson:

```
factory DictionaryWord.fromJson(Map<String, dynamic> json) {
    final container =
        _nestedMap(json, 'data') ?? _nestedMap(json, 'result') ??
json;
    final source = _nestedMap(json, 'dictionary_entry') ??
        _nestedMap(container, 'dictionary_entry') ??
        _nestedMap(container, 'entry') ??
        _nestedMap(container, 'word_data') ??
        _nestedMap(container, 'dictionary_word') ??
        _nestedMap(container, 'saved_word') ??
        container;
    return DictionaryWord(
      id:    _asInt(json['id']    ??    json['saved_word_id']    ??
source['id']),
      userId:    _asInt(json['user_id']    ??    json['userId']    ??
source['user_id']),
      word: _firstText(
        json,
        source,
        const ['word', 'source_word', 'original_word', 'text'],
      ),
      translation: _firstText(
        json,
        source,
        const [
          'translation',
          'translated_text',
          'translatedText',
```

```
          'target_text',
          'targetText',
          'meaning',
          'definition',
          'text',
        ],
      ),
      transcription: _firstText(
        json,
        source,
        const   ['transcription',   'phonetic',   'pronunciation',
'ipa'],
      ),
      partOfSpeech: _firstText(
        json,
        source,
        const [
          'part_of_speech',
          'partOfSpeech',
          'part_of_speech_label',
          'pos',
          'lexical_category',
          'lexicalCategory',
          'word_type',
          'wordType',
          'speech_part',
        ],
      ),
      originalLanguage: (json['source_language'] ??
              source['source_language'] ??
              json['originalLanguage'] ??
              'en')
          .toString(),
      targetLanguage: (json['target_language'] ??
              source['target_language'] ??
              json['targetLanguage'] ??
              'uk')
          .toString(),
      sourceBookId: _nullableInt(json['source_book_id'] ??
          source['source_book_id'] ??
          json['sourceBookId']),
      sourceParagraphIndex: _nullableInt(
          json['source_paragraph_index']                          ??
source['source_paragraph_index']),
      sourceCharacterIndex: _nullableInt(
          json['source_character_index']                          ??
source['source_character_index']),
      example: _firstText(
        json,
        source,
        const [
          'example',
```

```
          'example_sentence',
          'exampleSentence',
          'usage_example',
          'usageExample',
          'sentence',
          'context',
          'context_sentence',
          'contextSentence',
        ],
      ),
      savedAt: _asDate(json['saved_at'] ?? json['created_at']),
      clickCount: _asInt(json['click_count'] ?? 0),
      isFavorite:     json['is_favorite']     ==     true     ||
source['is_favorite'] == true,
      difficulty:
          (json['difficulty']    ??    source['difficulty']    ??
'medium').toString(),
    );
  }

  Map<String, dynamic> toSaveJson() {
    return {
      'word': word,
      'translation': translation,
      'source_language': originalLanguage,
      'target_language': targetLanguage,
      if (sourceBookId != null) 'source_book_id': sourceBookId,
      'difficulty': difficulty,
    };
  }
```

Збереження параметрів через shared_preferences:

```
class SettingsService {
  static final ValueNotifier<AppSettings> settingsNotifier =
      ValueNotifier<AppSettings>(AppSettings.defaults());
  static bool _loaded = false;

  static AppSettings get current => settingsNotifier.value;

  Future<AppSettings> getSettings() async {
    await _ensureLoaded();
    return settingsNotifier.value;
  }

  Future<AppSettings> updateSettings(AppSettings settings) async {
    await _ensureLoaded();
    settingsNotifier.value = settings;
    await _persist(settings);
    return settings;
  }
```

```
  Future<void> _ensureLoaded() async {
    if (_loaded) return;
    try {
      final prefs = await SharedPreferences.getInstance();
      final defaults = AppSettings.defaults();
      settingsNotifier.value = AppSettings(
        darkTheme:           prefs.getBool('darkTheme')          ??
defaults.darkTheme,
        readerFontSize: _enumByName(
          ReaderFontSize.values,
          prefs.getString('readerFontSize'),
          defaults.readerFontSize,
        ),
        readerFontFamily: _enumByName(
          ReaderFontFamily.values,
          prefs.getString('readerFontFamily'),
          defaults.readerFontFamily,
        ),
        readerTheme: _enumByName(
          ReaderTheme.values,
          prefs.getString('readerTheme'),
          defaults.readerTheme,
        ),
        readerTextColor: _enumByName(
          ReaderTextColor.values,
          prefs.getString('readerTextColor'),
          defaults.readerTextColor,
        ),
        readerLineHeight:
            prefs.getDouble('readerLineHeight')               ??
defaults.readerLineHeight,
        readerJustify:      prefs.getBool('readerJustify')     ??
defaults.readerJustify,
        readerComfortMode:
            prefs.getBool('readerComfortMode')                ??
defaults.readerComfortMode,
        interfaceLanguage:
            prefs.getString('interfaceLanguage')              ??
defaults.interfaceLanguage,
        sourceLanguage:
            prefs.getString('sourceLanguage')                 ??
defaults.sourceLanguage,
        targetLanguage:
            prefs.getString('targetLanguage')                 ??
defaults.targetLanguage,
      );
    } on MissingPluginException {
      settingsNotifier.value = AppSettings.defaults();
    }
    _loaded = true;
  }
```

```
  Future<void> _persist(AppSettings settings) async {
    try {
      final prefs = await SharedPreferences.getInstance();
      await prefs.setBool('darkTheme', settings.darkTheme);
      await                          prefs.setString('readerFontSize',
settings.readerFontSize.name);
      await                        prefs.setString('readerFontFamily',
settings.readerFontFamily.name);
      await                             prefs.setString('readerTheme',
settings.readerTheme.name);
      await                         prefs.setString('readerTextColor',
settings.readerTextColor.name);
      await                        prefs.setDouble('readerLineHeight',
settings.readerLineHeight);
      await prefs.setBool('readerJustify', settings.readerJustify);
      await                          prefs.setBool('readerComfortMode',
settings.readerComfortMode);
      await                       prefs.setString('interfaceLanguage',
settings.interfaceLanguage);
      await                           prefs.setString('sourceLanguage',
settings.sourceLanguage);
      await                           prefs.setString('targetLanguage',
settings.targetLanguage);
    } on MissingPluginException {
      // Allows widget tests to run without platform plugins.
    }
  }

  T _enumByName<T extends Enum>(List<T> values, String? name, T
fallback) {
    if (name == null) return fallback;
    for (final value in values) {
      if (value.name == name) return value;
    }
    return fallback;
  }
}
```

Витягування тексту з PDF у FileTextExtractor:

```
static FileTextExtractionResult _extractPdf(List<int> bytes) {
    final coverBytes = _firstEmbeddedImageBytes(bytes);
    if (!_looksLikePdf(bytes)) {
      return FileTextExtractionResult.unsupported(
        'Не вдалося прочитати PDF',
        coverBytes: coverBytes,
      );
    }

    PdfDocument? document;
    try {
      document = PdfDocument(inputBytes: bytes);
```

```
      final extractor = PdfTextExtractor(document);
      final buffer = StringBuffer();
      final totalPages = document.pages.count;
      if (totalPages <= 0) {
        return FileTextExtractionResult.unsupported(
          'Не вдалося прочитати PDF',
          coverBytes: coverBytes,
        );
      }

      for (var pageIndex = 0; pageIndex < totalPages; pageIndex++)
{
        final pageText = _extractPdfPage(
          extractor: extractor,
          pageIndex: pageIndex,
        );
        final cleanedPage = _cleanPdfText(pageText);
        if (cleanedPage.isEmpty) continue;
        if (buffer.isNotEmpty) buffer.writeln('\n');
        buffer.writeln(cleanedPage);
      }

      final text = _cleanPdfText(buffer.toString());
      if (text.isEmpty) {
        return FileTextExtractionResult.unsupported(
          'Не вдалося прочитати PDF',
          coverBytes: coverBytes,
        );
      }
      return FileTextExtractionResult.success(
        text,
        coverBytes: coverBytes,
        processedPages: totalPages,
        totalPages: totalPages,
      );
    } catch (_) {
      return FileTextExtractionResult.unsupported(
        'Не вдалося прочитати PDF',
        coverBytes: coverBytes,
      );
    } finally {
      document?.dispose();
    }
  }

  static String _extractPdfPage({
    required PdfTextExtractor extractor,
    required int pageIndex,
  }) {
    try {
      return extractor.extractText(
        startPageIndex: pageIndex,
```

```
        endPageIndex: pageIndex,
      );
    } catch (_) {
      throw const FormatException('PDF page text extraction
failed');
    }
  }

  static bool _looksLikePdf(List<int> bytes) {
    if (bytes.isEmpty) return false;
    final headerLength = bytes.length < 1024 ? bytes.length : 1024;
    final header =
latin1.decode(bytes.take(headerLength).toList());
    return header.contains('%PDF');
  }
```

Приклад unit тесту:

```
void main() {
  test('extracts UTF-8 text from txt files', () {
    final result = FileTextExtractor.extract(
      bytes: utf8.encode('Привіт LexiRead'),
      extension: 'txt',
    );

    expect(result.hasText, isTrue);
    expect(result.text, contains('LexiRead'));
  });

  test('extracts readable text from pdf files', () {
    final document = PdfDocument();
    final page = document.pages.add();
    page.graphics.drawString(
      'Readable PDF text for LexiRead',
      PdfStandardFont(PdfFontFamily.helvetica, 14),
    );
    final bytes = document.saveSync();
    document.dispose();

    final result = FileTextExtractor.extract(bytes: bytes,
extension: 'pdf');

    expect(result.hasText, isTrue);
    expect(result.text, contains('Readable PDF text'));
  });

  test('normalizes extra spacing in pdf text', () {
    final document = PdfDocument();
    final page = document.pages.add();
    page.graphics.drawString(
      'Soft-\nware text , with extra spaces .',
      PdfStandardFont(PdfFontFamily.helvetica, 14),
```

```
    );
    final bytes = document.saveSync();
    document.dispose();

    final result = FileTextExtractor.extract(bytes: bytes,
extension: 'pdf');

    expect(result.hasText, isTrue);
    expect(result.text, isNot(contains('  ')));
    expect(result.text, isNot(contains(' ,')));
    expect(result.text, isNot(contains(' .')));
  });

  test('extracts all pages from multi-page pdf files', () {
    final document = PdfDocument();
    for (var index = 1; index <= 3; index++) {
      final page = document.pages.add();
      page.graphics.drawString(
        'PDF page $index paragraph for LexiRead',
        PdfStandardFont(PdfFontFamily.helvetica, 14),
      );
    }
    final bytes = document.saveSync();
    document.dispose();

    final result = FileTextExtractor.extract(bytes: bytes,
extension: 'pdf');

    expect(result.hasText, isTrue);
    expect(result.processedPages, 3);
    expect(result.totalPages, 3);
    expect(result.text, contains('PDF page 1 paragraph'));
    expect(result.text, contains('PDF page 2 paragraph'));
    expect(result.text, contains('PDF page 3 paragraph'));
  });

  test('does not treat invalid bytes as a readable pdf', () {
    final result = FileTextExtractor.extract(
      bytes: utf8.encode('not a pdf'),
      extension: 'pdf',
    );

    expect(result.hasText, isFalse);
    expect(result.message, contains('PDF'));
  });

  test('preserves EPUB paragraph and chapter structure', () {
    final bytes = _storedZip({
      'OPS/chapter1.xhtml': '''
        <html><body>
          <h1>Chapter One</h1>
          <p>First paragraph.</p>
```

```
          <p>Second paragraph.</p>
          <section>
            <h2>Scene</h2>
            <p>Third<br/>line.</p>
          </section>
        </body></html>
      ''',
      'OPS/chapter2.xhtml': '<html><body><p>Next
chapter.</p></body></html>',
    });

    final result = FileTextExtractor.extract(bytes: bytes,
extension: 'epub');

    expect(result.hasText, isTrue);
    expect(result.text, contains('Chapter One\n\nFirst
paragraph.'));
    expect(result.text, contains('First paragraph.\n\nSecond
paragraph.'));
    expect(result.text, contains('Scene\n\nThird\nline.'));
    expect(result.text, contains('line.\n\nNext chapter.'));
  });

  test('extracts cover bytes from epub and docx packages when
available', () {
    final image = List<int>.filled(1400, 0x7A);
    final epub = FileTextExtractor.extract(
      bytes: _storedZip({
        'OPS/cover.jpg': String.fromCharCodes(image),
        'OPS/chapter.xhtml':
'<html><body><p>Text.</p></body></html>',
      }),
      extension: 'epub',
    );
    final docx = FileTextExtractor.extract(
      bytes: _storedZip({
        'word/media/cover.jpg': String.fromCharCodes(image),
        'word/document.xml':

'<w:document><w:p><w:t>Text.</w:t></w:p></w:document>',
      }),
      extension: 'docx',
    );

    expect(epub.coverBytes, isNotNull);
    expect(docx.coverBytes, isNotNull);
  });
}

List<int> _storedZip(Map<String, String> files) {
  final output = <int>[];
  final centralDirectory = <int>[];
```

```
for (final entry in files.entries) {
  final nameBytes = utf8.encode(entry.key);
  final dataBytes = utf8.encode(entry.value);
  final localOffset = output.length;

  _u32(output, 0x04034b50);
  _u16(output, 20);
  _u16(output, 0);
  _u16(output, 0);
  _u16(output, 0);
  _u16(output, 0);
  _u32(output, 0);
  _u32(output, dataBytes.length);
  _u32(output, dataBytes.length);
  _u16(output, nameBytes.length);
  _u16(output, 0);
  output
    ..addAll(nameBytes)
    ..addAll(dataBytes);

  _u32(centralDirectory, 0x02014b50);
  _u16(centralDirectory, 20);
  _u16(centralDirectory, 20);
  _u16(centralDirectory, 0);
  _u16(centralDirectory, 0);
  _u16(centralDirectory, 0);
  _u16(centralDirectory, 0);
  _u32(centralDirectory, 0);
  _u32(centralDirectory, dataBytes.length);
  _u32(centralDirectory, dataBytes.length);
  _u16(centralDirectory, nameBytes.length);
  _u16(centralDirectory, 0);
  _u16(centralDirectory, 0);
  _u16(centralDirectory, 0);
  _u16(centralDirectory, 0);
  _u32(centralDirectory, 0);
  _u32(centralDirectory, localOffset);
  centralDirectory.addAll(nameBytes);
}

final centralOffset = output.length;
output.addAll(centralDirectory);

_u32(output, 0x06054b50);
_u16(output, 0);
_u16(output, 0);
_u16(output, files.length);
_u16(output, files.length);
_u32(output, centralDirectory.length);
_u32(output, centralOffset);
_u16(output, 0);
```

```
  return Uint8List.fromList(output);
}

void _u16(List<int> output, int value) {
  output
    ..add(value & 0xff)
    ..add((value >> 8) & 0xff);
}

void _u32(List<int> output, int value) {
  output
    ..add(value & 0xff)
    ..add((value >> 8) & 0xff)
    ..add((value >> 16) & 0xff)
    ..add((value >> 24) & 0xff);
}
```

Приклад widget тесту:

```
void main() {
  testWidgets('LexiRead opens login screen', (tester) async {
    await tester.pumpWidget(const LexiReadApp());

    expect(find.text('LexiRead'), findsOneWidget);
    expect(find.text('Увійти'), findsOneWidget);
    expect(find.text('Зареєструватися'), findsOneWidget);
  });
}
```

ДОДАТОК Е

Тестові сценарії та результати тестування клієнтської частини

Таблиця Е.1 — Ручні тестові сценарії клієнтської частини

ID	Сценарій	Дія користувача	Очікуваний результат	Статус
TC-01	Авторизація	Ввести email і пароль	Відкривається LibraryScreen	Успішно
TC-02	Реєстрація	Заповнити форму реєстрації	Створюється обліковий запис	Успішно
TC-03	Відкриття бібліотеки	Перейти до LibraryScreen	Відображається список книг	Успішно
TC-04	Додавання книги	Вибрати TXT/PDF/EPUB/DOCX	Книга додається в бібліотеку	Успішно
TC-05	Відкриття книги	Натиснути на книгу	Відкривається ReaderScreen	Успішно
TC-06	Перегортання сторінок	Перейти між сторінками	Текст оновлюється коректно	Успішно
TC-07	Виклик словника	Натиснути на слово	Відкривається DictionaryBottomSheet	Успішно
TC-08	Отримання перекладу	Дочекатися відповіді API	Відображається переклад	Успішно
TC-09	Збереження слова	Натиснути «У словник»	Слово додається у словник	Успішно
TC-10	Перегляд словника	Відкрити DictionaryScreen	Відображається список слів	Успішно
TC-11	Робота з картками	Відкрити FlashcardsScreen	Відображаються картки	Успішно
TC-12	Тестування	Відкрити QuizScreen	Відображаються питання тесту	Успішно
TC-13	Закладки	Додати закладку	Закладка зберігається	Успішно
TC-14	Зміна теми	Перемкнути тему	Тема застосунку змінюється	Успішно
TC-15	Error state	Виконати запит без API	Відображається повідомлення про помилку	Успішно

Таблиця Е.2 – Автоматизовані unit/widget тести

ID	Файл тесту	Об’єкт перевірки	Тип тесту	Статус
AT-01	form_validators_test.dart	Валідація email	Unit	Успішно
AT-02	file_text_extractor_test.dart	Імпорт TXT	Unit	Успішно
AT-03	file_text_extractor_test.dart	Витягування PDF	Unit	Успішно
AT-04	file_text_extractor_test.dart	Багатосторінковий PDF	Unit	Успішно
AT-05	file_text_extractor_test.dart	Некоректний PDF	Unit	Успішно
AT-06	file_text_extractor_test.dart	Обробка EPUB	Unit	Успішно
AT-07	flashcard_model_test.dart	Модель Flashcard	Unit	Успішно
AT-08	quiz_question_test.dart	Модель QuizQuestion	Unit	Успішно
AT-09	quiz_question_test.dart	Fallback quiz logic	Unit	Успішно
AT-10	widget_test.dart	Відкриття LoginScreen	Widget	Успішно
AT-11	flutter test --no-pub	Запуск тестового набору	Unit/Widget	Успішно