МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)
(рівень вищої освіти)

на тему **Мобільний застосунок для вивчення іноземних слів**

Виконав: студент 4 курсу, групи 6.1211-ПЗС
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
Д.П. Опрятов
(ініціали та прізвище)
Керівник доцент, к.т.н, доцент А.І. Безверхий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент Директор ТОВ «Дискус»
Р.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення

Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 Інженерія програмного забезпечення
(код та назва)

Освітня програма Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри Тетяна КРИТСЬКА

“ 01 ” березня 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Опрятову Данилу Павловичу
(прізвище, ім’я, по батькові)

1. Тема роботи Мобільний застосунок для вивчення іноземних слів
керівник роботи Безверхий Анатолій Ігорович, доцент,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від 27.12.2024 р. № 2029-с
2. Строк подання студентом кваліфікаційної роботи 02.06.2025
3. Вихідні дані бакалаврської роботи
   • комплект нормативних документів ;
   • технічне завдання до роботи.

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   • огляд та збір літератури стосовно теми кваліфікаційної роботи;
   • огляд та аналіз існуючих рішень та аналогів;
   • дослідження методів інтервального повторення та їх застосування в мобільних додатках;
   • створення програмного продукту та його опис;
   • перелік вимог для роботи програми;
   • розробка висновків та пропозицій.
4. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) слайдів презентації

Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

Дата видачі завдання 01.03.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області	01.03 – 10.03.25	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	11.03 – 12.03. 25	виконано
3	Аналіз існуючих аналогів	13.03 – 14.03. 25	виконано
4	Детальний аналіз API для перекладу (DeepL API)	15.03 – 20.03. 25	виконано
5	Розробка структури бази даних для зберігання слів, перекладів, налаштувань користувача	21.03 – 26.03. 25	виконано
6	Налаштування стартових CRUD операцій для роботи з базою даних	27.03 – 28.03. 25	виконано
7	Розробка 3 алгоритмів для режиму навчання	29.03 – 13.04. 25	виконано
8	Розробка API для роботи з сервісом перекладу	14.04 – 16.04. 25	виконано
9	Аналіз клієнтської частини та узгодження подальшої розробки з науковим керівником	17.04 – 19.04.25	виконано
10	Створення дизайну інтерфейсу користувача	20.04 – 01.0525	виконано
11	Розробка стартового проекту та налагодження взаємодії з даними через API	02.05 – 13.05. 25	виконано
12	Реалізація системи локальних сповіщень для нагадування про навчання	13.05 – 20.05. 25	виконано
13	Остаточна перевірка працездатності всіх складових	21.05 – 25.05.25	виконано
14	Створення презентації	26.05 – 28.05.25	виконано
15	Оформлення звіту	29.05 – 02.05.25	виконано

Студент _____ Д.П. Опрятов
(підпис) (прізвище та ініціали)

Керівник роботи _____ А.І. Безверхий
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок — 63,

Рисунків —10,

Джерел — 15.

Опрятов Д. П. Мобільний застосунок для вивчення іноземних слів: кваліфікаційна робота бакалавра спеціальності 121 «Програмне забезпечення систем» / наук. керівник А.І. Безверхий. Запоріжжя : ЗНУ, 2025. 63 с.

У кваліфікаційній роботі досліджено проблему ефективного вивчення іноземних слів з використанням сучасних мобільних технологій. Проаналізовано існуючі додатки для вивчення іноземних слів та виявлено їх недоліки, такі як відсутність гнучкого механізму повторення й обмежена інтеграція з зовнішніми сервісами перекладу.

Метою роботи є розробка мобільного застосунку для Android, що дозволяє користувачам ефективно вивчати іноземні слова, використовуючи інтеграцію з API перекладу. Основними завданнями є: реалізація механізму додавання нових слів користувачем; отримання перекладів через API; розробка системи інтерактивних вправ для запам'ятовування (зіставлення слів та перекладів); реалізація системи локальних сповіщень для нагадування про необхідність повторення слів.

У процесі роботи було реалізовано основний функціонал застосунку на мові програмування Kotlin з використанням Android Studio. Проведено тестування розробленого застосунку на емуляторі та реальних пристроях.

Розроблений мобільний застосунок забезпечує користувачам гнучкий механізм нагадування для повторення матеріалу та інтегрується з сучасними сервісами перекладу, що сприяє підвищенню ефективності навчання.

Ключові слова: *мобільний застосунок, Android, вивчення іноземних слів, DeepL API, локальні сповіщення, інтерактивні вправи, Kotlin*

ABSTRACT

Pages — 63,

Drawings — 10,

Sources — 15.

Opriatov, D. P. Mobile Application for Learning Foreign Words: Bachelor's Qualification Work, Speciality 121 "Software Engineering" / Scientific Supervisor A.I. Bezverkhyi. Zaporizhzhia: ZNU, 2025. X p.

This qualification work investigates the problem of effectively learning foreign words using modern mobile technologies. Existing applications for learning foreign words were analyzed, and their shortcomings were identified, such as the lack of a flexible repetition mechanism and limited integration with external translation services.

The aim of this work is to develop an Android mobile application that allows users to effectively learn foreign words, utilizing integration with a translation API. The main tasks are: implementing a mechanism for users to add new words; obtaining translations via the API; developing a system of interactive exercises for memorization (matching words and translations); implementing a system of local notifications to remind users of the need to review words.

In the process of this work, the core functionality of the application was implemented using the Kotlin programming language and Android Studio. The developed application was tested on an emulator and real devices.

The developed mobile application provides users with a flexible reminder mechanism for repeating material and integrates with modern translation services, which contributes to increased learning efficiency.

Keywords: *mobile application, Android, foreign language learning, DeepL API, local notifications, interactive exercises, Kotlin*

ЗМІСТ

ВСТУП 8

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ 12

1.1 Огляд існуючих мобільних застосунків 12

1.1.1 Аналіз функціональних можливостей популярних аналогів 12

1.1.2 Виявлення переваг та недоліків існуючих застосунків 13

1.1.3 Аргументація доцільності розробки нового мобільного застосунку 14

1.2 Огляд сучасних підходів до вивчення слів 15

1.2.1 Огляд API для перекладу 15

1.2.2 Методики запам'ятовування 16

1.3 Постановка задачі кваліфікаційної роботи 17

1.4 Вибір та аргументація технологічного стеку 18

1.5 Висновки до розділу 1 20

РОЗДІЛ 2 ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ 22

2.1 Аналіз вимог до застосунку 22

2.1.1 Функціональні вимоги 22

2.1.2 Нефункціональні вимоги 24

2.2 Архітектура системи 25

2.2.1 Вибір архітектурного патерну та його аргументація 25

2.2.2 Структура даних та база даних 27

2.2.3 Технології для локальних сповіщень 27

2.2.4 Модулі застосунку 28
2.2.5 Користувацький інтерфейс 28
2.3 Висновки до розділу 2 29
РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ 30
3.1 Середовище розробки та інструменти 30
3.2 Реалізація основного функціоналу 31
3.2.1 Структура проєкту та архітектура MVVM 31
3.2.2 Реалізація шару даних 33
3.2.3 Управління словником користувача 44
3.2.4 Реалізація режимів навчання 47
3.2.5 Система локальних сповіщень та інтервального повторення 50
3.2.6 Реалізація користувацького інтерфейсу та навігації 53
3.3 Реалізація і тестування мобільного застосунку 59
3.3.1 Реалізація 59
3.3.2 Апробація 59
3.4 Висновки до розділу 3 60
ВИСНОВКИ 62
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 63

ВСТУП

Вивчення іноземних мов є невід'ємною частиною сучасного життя, відкриваючи можливості для особистісного та професійного зростання. Ефективне запам'ятовування нових слів — один з головних аспектів успішного оволодіння мовою. Мобільні застосунки стають дедалі популярнішими інструментами для навчання, пропонуючи зручність та доступність [9]. Розробка мобільного застосунку, що спрощує та оптимізує процес вивчення іноземних слів, є актуальним завданням, що відповідає потребам сучасних користувачів. Застосунок, який використовує інтерактивні методи запам'ятовування, може значно підвищити ефективність навчання.

Технологічний прогрес, зокрема розвиток мобільних технологій та доступ до API перекладачів, дозволяє створювати інноваційні інструменти для вивчення мов.

Мобільні застосунки для вивчення мов також можуть бути корисними для освітніх установ, мовних шкіл та репетиторів, надаючи їм додаткові інструменти для навчання та контролю прогресу учнів.

Застосування інформаційних технологій дозволяє автоматизувати процес навчання та зробити вивчення мови більш захопливим та ефективним.

В сучасному світі вивчення іноземних мов є важливим для особистого та професійного розвитку. Для полегшення цього процесу створюються інформаційні системи, які об'єднують бази даних, інформаційні технології та технічні засоби. Їх функції включають надійне зберігання інформації, виконання специфічних перетворень та обчислень, а також надання зручного інтерфейсу користувачам.

Інформаційні системи використовуються в різних галузях, таких як освіта, перекладацька діяльність, міжнародні комунікації та багато інших.

Мета дослідження

Розробка мобільного застосунку для ефективного вивчення іноземних слів з використанням сучасних технологій (локальні сповіщення, API перекладу).

Завдання дослідження

Проаналізувати існуючі мобільні застосунки для вивчення іноземних мов, виявити їх переваги та недоліки.

1. Реалізувати інтерактивні режими навчання та систему інтервальних нагадувань для ефективного запам'ятовування слів.
2. Вибрати оптимальні технології для розробки мобільного застосунку (платформа, мова програмування, API).
3. Спроектувати архітектуру та інтерфейс застосунку.
4. Реалізувати основний функціонал застосунку: додавання, редагування та видалення слів користувачем, інтеграція з API перекладу, система інтерактивних вправ, механізм інтервального нагадування про навчання та система локальних сповіщень.

Об’єкт дослідження

Процес розробки мобільного застосунку для вивчення іноземних слів на платформі Android.

Предмет дослідження

Методи та технології створення мобільного застосунку для вивчення іноземних слів, що включають використання локальних сповіщень, API перекладу та інтерактивних методів запам'ятовування.

Методи дослідження

Теоретичні: аналіз наукової літератури та існуючих рішень у сфері мобільних застосунків для вивчення мов; системний аналіз для проєктування

архітектури застосунку.

Емпіричні: об'єктно-орієнтоване програмування; проєктування баз даних; розробка програмного забезпечення з використанням Android SDK та Android Studio; інтеграція з API сторонніх сервісів; тестування програмного продукту.

Практичне значення одержаних результатів

Розроблений мобільний застосунок надає користувачам зручний та ефективний інструмент для поповнення словникового запасу іноземної мови. Практичне значення полягає у:

1. Підвищенні ефективності вивчення іноземних слів завдяки застосуванню інтерактивних вправ.
2. Наданні користувачам доступу до якісних перекладів через інтеграцію з API DeepL.
3. Підвищенні регулярності занять за допомогою системи локальних сповіщень.

Глосарій

Мобільний застосунок — Програмне забезпечення, розроблене для роботи на мобільних пристроях (смартфонах, планшетах).

Android — Операційна система для мобільних пристроїв, розроблена Google.

API (Application Programming Interface / Прикладний програмний інтерфейс) — Набір визначень, протоколів та інструментів для створення програмного забезпечення та взаємодії різних програмних компонентів.

DeepL API — Сервіс машинного перекладу, що надає API для інтеграції функцій перекладу в інші застосунки.

Локальні сповіщення — Сповіщення, що генеруються та плануються безпосередньо застосунком для відображення користувачеві у визначений час, навіть якщо застосунок не активний або знаходиться у фоновому режимі. Реалізовані за допомогою WorkManager.

Інтервальне повторення (Spaced Repetition) — Техніка навчання, що базується на повторенні навчального матеріалу через зростаючі проміжки часу для покращення довготривалого запам'ятовування.

Kotlin — Сучасна статично типізована мова програмування, офіційно підтримувана Google для розробки під Android.

Room Persistence Library — Офіційна бібліотека Android Jetpack, що надає абстракцію над SQLite для спрощеної роботи з базами даних у застосунках.

MVVM (Model-View-ViewModel) — Архітектурний патерн проєктування, що сприяє розділенню логіки представлення даних (View) від бізнес-логіки (ViewModel) та даних (Model).

UI (User Interface / Користувацький інтерфейс) — Сукупність засобів, за допомогою яких користувач взаємодіє з програмою (елементи керування, вікна, кнопки тощо).

UX (User Experience / Користувацький досвід) — Загальне враження та рівень задоволеності користувача від взаємодії з продуктом, системою чи послугою.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Огляд існуючих мобільних застосунків

Ринок мобільних застосунків для вивчення іноземних мов є досить насиченим та пропонує користувачам широкий вибір інструментів з різним функціоналом та підходами до навчання. Задля аргументації доцільності розробки нового продукту необхідно провести аналіз популярних аналогів, виявити їх сильні та слабкі сторони.

1.1.1 Аналіз функціональних можливостей популярних аналогів

Розглянемо декілька провідних застосунків у цій сфері:

1. Duolingo [1]: Один з найпопулярніших застосунків, що пропонує вивчення багатьох мов через ігрові уроки. Основний акцент робиться на граматиці, лексиці, читанні та аудіюванні. Використовує систему "дерева навичок", бали досвіду, "життя" для мотивації. Має вбудований словник вивчених слів. Пропонує короткі уроки, що зручно для навчання "на ходу". На **Помилка! Джерело посилання не знайдено.** з ображено графічний інтерфейс додатка.
2. Memrise [2]: Застосунок, що фокусується на запам'ятовуванні слів та фраз за допомогою мнемотехнік, відео з носіями мови та методу інтервальних повторень. Пропонує офіційні курси та курси, створені спільнотою. Має режими "класичного повторення" та "швидкого повторення".
3. Quizlet [3]: Платформа для створення навчальних наборів (карток) на будь-яку тему, включаючи іноземні мови. Пропонує різні режими навчання: картки, заучування, тести, підбір. Можливість додавати зображення та аудіо. Має функції для спільної роботи та для використання наборів, створених іншими користувачами.

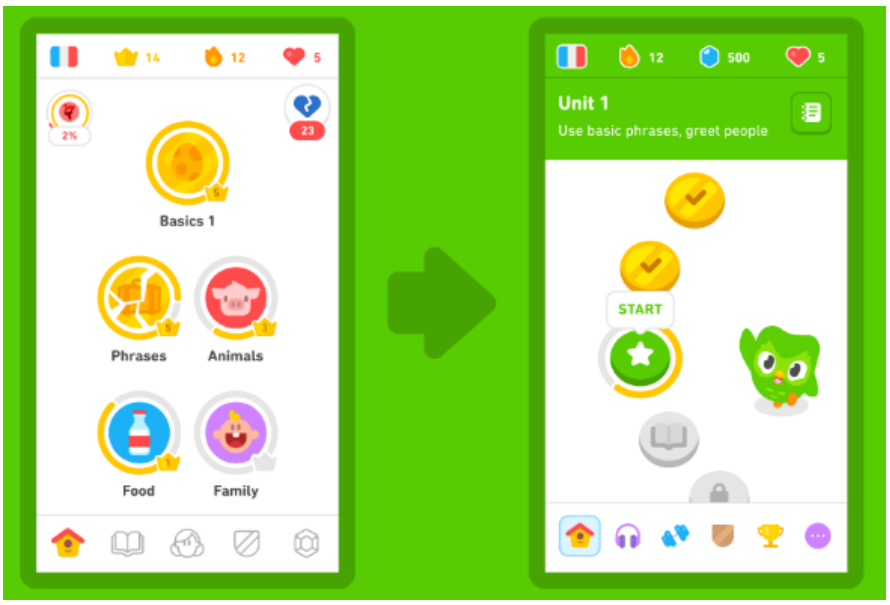


Рисунок 1 – *Інтерфейс додатка Duolingo*

1.1.2 Виявлення переваг та недоліків існуючих застосунків

На основі аналізу можна виділити наступні загальні переваги та недоліки [10]:

1. Переваги:
   1) Інтерактивність та гейміфікація (Duolingo, Memrise): Роблять процес навчання більш захопливим.
   2) Використання науково аргументованих методик (Memrise): Наприклад, інтервальні повторення.
   3) Велика база контенту (Duolingo, Memrise, Quizlet): Готові курси та набори карток.
   4) Кросплатформність та синхронізація: Можливість навчатися на різних пристроях.
   5) Можливість створення власного контенту (Quizlet, Memrise): Гнучкість для користувачів.
2. Недоліки:
   1) Обмежена гнучкість у створенні та управлінні власними списками слів: Це ускладнює роботу користувачів, які бажають додавати слова з власних джерел.

2) Недостатня інтеграція з якісними API перекладу: Часто переклад або обмежений вбудованим словником, або вимагає ручного введення. Можливість швидко отримати якісний переклад для нового слова через сучасні API (як DeepL) не завжди реалізована.
3) Монетизація: Наявність реклами або платних підписок може обмежувати доступ до повного функціоналу.
4) Фокус на загальному вивченні мови, а не на цілеспрямованому поповненні словникового запасу: Деякі застосунки (наприклад, Duolingo) інтегрують вивчення слів у ширший контекст граматики та навичок, що може бути неоптимальним для користувачів, які хочуть зосередитися саме на лексиці.

1.1.3 Аргументація доцільності розробки нового мобільного застосунку

Аналіз існуючих рішень показує, що попри велику кількість застосунків, існує ніша для продукту, який би поєднував зручність управління власними списками слів, глибоку інтеграцію з якісними сервісами перекладу, ефективні методики запам'ятовування та проактивну систему нагадувань.

Доцільність розробки нового мобільного застосунку аргументується наступними факторами:

1. Потреба в зручному інструменті для роботи з власними словами: Багато користувачів стикаються з новими словами під час читання, перегляду фільмів або спілкування і потребують простого способу їх збереження, отримання якісного перекладу та подальшого вивчення.
2. Інтеграція з передовими API перекладу: Використання API, таких як DeepL, може забезпечити високу точність перекладу.
3. Реалізація ефективних та гнучких методів запам'ятовування: Поєднання інтерактивних вправ (наприклад, зіставлення слів та перекладів, введення перекладу) з адаптивним алгоритмом інтервального нагадування може значно підвищити ефективність навчання.

4. Фокус на простоті та користувацькому досвіді: Створення інтуїтивно зрозумілого інтерфейсу, що не перевантажений зайвими функціями, дозволить користувачам зосередитися на головному — вивченні слів.

1.2 Огляд сучасних підходів до вивчення слів

Ефективне вивчення іноземних слів базується на поєднанні перевірених часом методик та сучасних технологічних можливостей. Розглянемо ключові підходи, які є актуальними для розробки сучасного мобільного застосунку для вивчення слів.

1.2.1 Огляд API для перекладу

Фундаментальною потребою при вивченні нових слів є отримання точного перекладу. Сучасні сервіси машинного перекладу, такі як DeepL API, Google Translate API, Microsoft Translator API, пропонують такий функціонал. Серед них DeepL API [6] виділяється своєю здатністю надавати високоякісні та природні переклади, що робить його оптимальним вибором для інтеграції в застосунок.

1. Переваги інтеграції з API перекладу:
    1) Швидкість та зручність: Користувач може миттєво отримати переклад невідомого слова без необхідності перемикатися між застосунками чи шукати в паперовому словнику.
    2) Якість перекладу: Провідні API, як DeepL, відомі своєю здатністю надавати високоякісні переклади.
    3) Підтримка багатьох мов: Можливість легко додавати підтримку нових мовних пар.
2. Виклики інтеграції:
    1) Вартість: Більшість якісних API є платними.
    2) Обмеження запитів: Можуть існувати ліміти на кількість запитів за певний період.

3) Залежність від інтернет-з'єднання: Для отримання перекладу потрібен доступ до мережі.

1.2.2 Методики запам'ятовування

Однією з найбільш ефективних методик для довготривалого запам'ятовування є інтервальне повторення (Spaced Repetition) [4]. Цей підхід базується на принципі повторення навчального матеріалу через зростаючі проміжки часу. Ідея полягає в тому, щоб повторювати інформацію саме тоді, коли вона починає забуватися, що дозволяє перемістити її з короткотривалої пам'яті до довготривалої. Концепція інтервального повторення тісно пов'язана з "кривою забування" Германа Еббінгауза, яка демонструє, як швидко інформація втрачається з пам'яті без повторення [5]. Застосування адаптивних алгоритмів інтервального повторення дозволяє оптимізувати час, витрачений на навчання, та значно підвищити ефективність запам'ятовування.

Пасивне заучування слів, як правило, менш ефективне, ніж активна взаємодія з матеріалом через інтерактивні вправи. Такі вправи стимулюють різні когнітивні процеси, що сприяє кращому запам'ятовуванню та розумінню.

1. Типи популярних інтерактивних вправ:
   1) Флеш-картки (Flashcards): Класичний метод, де на одній стороні картки слово, на іншій — переклад.
   2) Введення перекладу (Typing): Користувач бачить слово і має ввести його переклад.
   3) Множинний вибір (Multiple Choice): Користувачеві пропонується слово та кілька варіантів перекладу, з яких потрібно вибрати правильний.
2. Переваги інтерактивних вправ:
   1) Активне залучення: Користувач не просто споживає інформацію, а активно з нею працює.
   2) Різноманітність: Можливість змінювати типи вправ запобігає монотонності та підтримує інтерес.

3) Миттєвий зворотний зв'язок: Користувач одразу бачить, чи правильно він відповів, що допомагає коригувати помилки.

1.3 Постановка задачі кваліфікаційної роботи

На основі проведеного аналізу предметної області, існуючих рішень та сучасних підходів до вивчення іноземних слів, формулюється наступна задача кваліфікаційної роботи:

Розробити мобільний застосунок для платформи Android, призначений для ефективного вивчення іноземних слів, який реалізує наступний функціонал:

1. Управління словником користувача:
   1) Можливість додавання нових слів та фраз користувачем.
   2) Інтеграція з API сервісу перекладу (DeepL API) для автоматичного отримання перекладу.
   3) Редагування та видалення слів зі словника.
2. Механізми вивчення та повторення:
   1) Флеш-картки.
   2) Введення перекладу.
   3) Множинний вибір.
   4) Впровадження алгоритму інтервального нагадування про навчання для оптимізації процесу запам'ятовування, реалізованого за допомогою системи локальних сповіщень.
3. Інтерфейс користувача:
   1) Розробка інтуїтивно зрозумілого та зручного користувацького інтерфейсу.
   2) Забезпечення чіткого відображення прогресу користувача у вправах.

1.4 Вибір та аргументація технологічного стеку

Для реалізації мобільного застосунку для вивчення іноземних слів було обрано наступний стек технологій, який відповідає сучасним стандартам Android-розробки та забезпечує виконання всіх функціональних та нефункціональних вимог.

1. Платформа: Android
    1) Android є найпопулярнішою мобільною операційною системою у світі, що забезпечує максимальне охоплення потенційних користувачів [7]. Розробка під Android дозволяє використовувати багато створених інструментів та бібліотек. Це відповідає меті створення мобільного застосунку.
2. Мова програмування: Kotlin
    1) Kotlin є офіційно рекомендованою мовою для розробки Android-застосунків від Google [11]. Вона пропонує низку переваг порівняно з Java, включаючи:
        a. Концепція безпеки від null (Null Safety): Зменшує кількість помилок NullPointerException, підвищуючи стабільність застосунку.
        b. Лаконічність коду: Дозволяє писати менше коду для досягнення тієї ж функціональності, що прискорює розробку та покращує читабельність.
        c. Повна сумісність з Java: Дозволяє використовувати існуючі Java-бібліотеки та фреймворки.
3. Середовище розробки: Android Studio
    1) Android Studio є офіційним інтегрованим середовищем розробки (IDE) від Google, спеціально розробленим для Android-розробки. На **Помилка! Джерело посилання не знайдено.** зображено графічний і нтерфейс IDE Android Studio. Воно надає повний набір інструментів, необхідних для створення, тестування та налагодження застосунків,

включаючи:

a. Вбудований емулятор.
b. Інструменти для профілювання продуктивності.
c. Інтеграцію з системою збірки Gradle.
d. Розширені можливості для автодоповнення коду, рефакторингу та аналізу.

4. База даних: Room Persistence Library (на базі SQLite)
   1) Room є частиною Android Jetpack і надає абстракцію над SQLite, значно спрощуючи роботу з локальними базами даних [12]. Його переваги включають:
      a. Об'єктно-реляційне відображення (ORM): Дозволяє працювати з даними як з об'єктами Kotlin, а не з SQL-запитами.
      b. Перевірка SQL-запитів під час компіляції: Запобігає багатьом помилкам, які могли б виникнути під час виконання.
      c. Інтеграція з LiveData: Забезпечує асинхронну роботу з даними, що ідеально поєднується з обраним архітектурним патерном MVVM.
5. API перекладу: DeepL API
   1) Як зазначено в розділі 1.2.1, DeepL API відомий своєю високою якістю машинного перекладу. Інтеграція з DeepL API дозволяє автоматично отримувати якісні переклади для нових слів, що є ключовою функціональною вимогою застосунку.
6. Локальні сповіщення: WorkManager
   1) WorkManager є рекомендованою бібліотекою Android Jetpack для виконання відкладених, але гарантованих фонових завдань [14]. Це ідеальне рішення для реалізації системи інтервальних нагадувань, оскільки WorkManager:
      a. Гарантує виконання завдань: Навіть якщо застосунок закритий або пристрій перезавантажено.
      b. Оптимізований для роботи з обмеженнями системи: Враховує режими енергозбереження (Doze Mode, App Standby),

забезпечуючи своєчасну доставку сповіщень.

c. Надає гнучкі можливості планування: Дозволяє налаштовувати затримки та умови для запуску нагадувань.

7. Архітектурний патерн: MVVM (Model-View-ViewModel)

1) Детальна аргументація вибору MVVM наведена в розділі 2.2.1. Цей патерн забезпечує чітке розділення відповідальності між компонентами застосунку, що значно покращує тестованість, підтримуваність та масштабованість коду. Його інтеграція з LiveData та ViewModel з Android Jetpack робить його гарним вибором для сучасної Android-розробки.

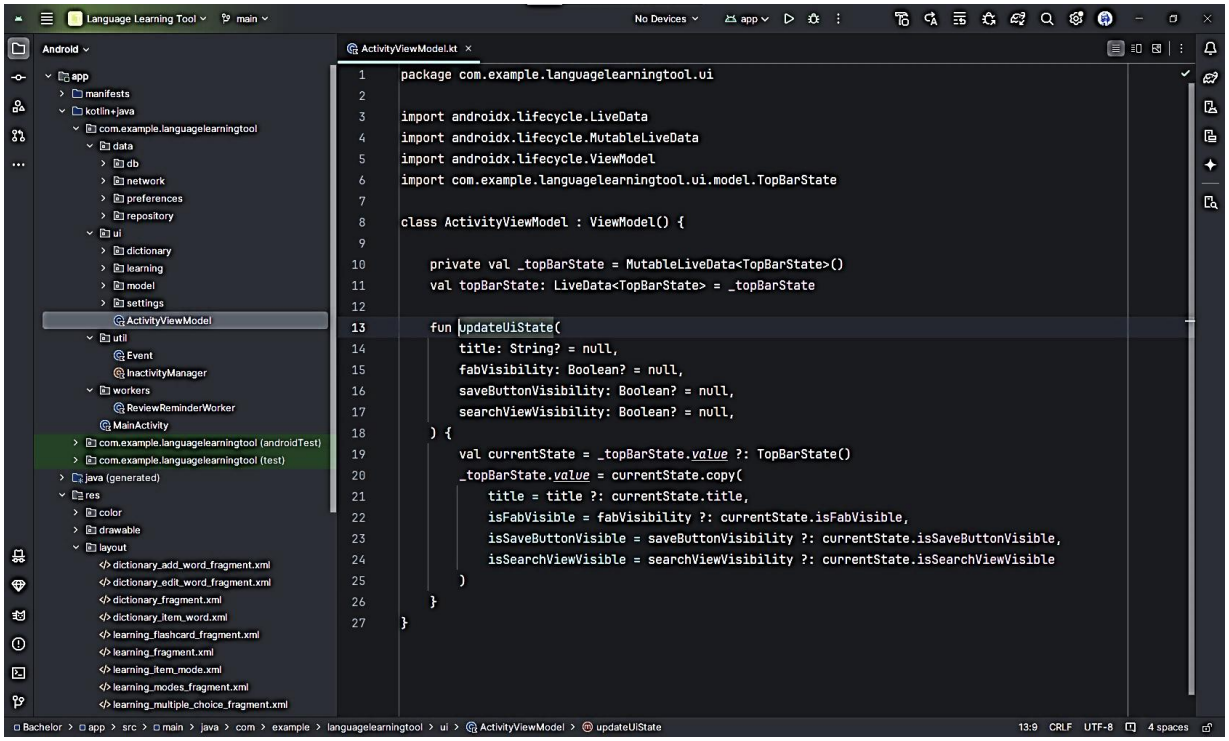


Рисунок 2 – *Графічний інтерфейс IDE Android Studio*

1.5 Висновки до розділу 1

1. Визначено актуальність проблеми ефективного вивчення іноземних слів та ключову роль мобільних технологій у її вирішенні.
2. Проаналізовано існуючі мобільні застосунки для вивчення слів

(Duolingo, Memrise, Quizlet), виявлено їх переваги та недоліки.

3. Розглянуто сучасні підходи до вивчення слів: інтеграція з API перекладу та інтерактивні вправи, що ляжуть в основу розробки.
4. Сформульовано постановку задачі кваліфікаційної роботи.

РОЗДІЛ 2 ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ

2.1 Аналіз вимог до застосунку

2.1.1 Функціональні вимоги

Основні функціональні вимоги до мобільного застосунку для вивчення іноземних слів включають:

1. Управління словником користувача:
   1) Користувач повинен мати можливість додавати нові слова до свого особистого словника.
   2) При додаванні слова користувач повинен мати можливість ввести оригінальне слово та вибрати мову оригіналу і мову перекладу.
   3) Система повинна автоматично отримувати переклад введеного слова через інтеграцію з API сервісу перекладу (DeepL API).
   4) Користувач повинен мати можливість переглядати свій словник.
   5) Користувач повинен мати можливість редагувати раніше додані слова та їх переклади.
   6) Користувач повинен мати можливість видаляти слова зі свого словника.
2. Механізми вивчення та повторення слів:
   1) Система повинна надавати режим навчання за допомогою флеш-карток (відображення слова на одній мові та його перекладу на іншій).
   2) Система повинна надавати інтерактивну вправу "Введення перекладу", де користувач має ввести переклад запропонованого слова.
   3) Система повинна надавати інтерактивну вправу "Множинний вибір", де користувач має обрати правильний переклад з кількох запропонованих варіантів.
   4) Система повинна реалізовувати алгоритм інтервальних нагадувань

для визначення оптимального часу наступного повторення слів.

3. Інтерфейс користувача та навігація:
   1) Застосунок повинен мати інтуїтивно зрозумілий та легкий у використанні інтерфейс.
   2) Навігація між основними розділами застосунку (словник, режими навчання, налаштування) повинна бути простою та логічною.

На Рис. 3 представлена діаграма варіантів використання, що ілюструє основні функціональні можливості застосунку та взаємодії користувача з ними.



Рисунок 3 – *Діаграма використання між користувачем та застосунком*

2.1.2 Нефункціональні вимоги

Основні нефункціональні вимоги:

1) Продуктивність:
    1) Час відповіді застосунку на дії користувача (наприклад, відкриття екранів) не повинен перевищувати 0.5-1 секунди.
    2) Запит на переклад слова через API та отримання відповіді повинен виконуватися протягом 1-3 секунд за умови стабільного інтернет-з'єднання.
    3) Застосунок повинен ефективно використовувати ресурси мобільного пристрою (пам'ять, процесор), щоб уникнути надмірного споживання заряду батареї.
2) Зручність використання:
    1) Інтерфейс застосунку повинен бути інтуїтивно зрозумілим для користувачів.
    2) Основні функції повинні бути легко доступні.
    3) Застосунок повинен надавати зворотний зв'язок на дії користувача.
3) Надійність:
    1) Застосунок повинен стабільно працювати без непередбачених збоїв та помилок.
    2) Дані користувача (словник) повинні надійно зберігатися локально на пристрої.
    3) Система повинна коректно обробляти ситуації відсутності інтернет-з'єднання при спробі отримати переклад через API (наприклад, виводити відповідне повідомлення).
4) Сумісність:
    1) Застосунок повинен підтримуватися на пристроях з операційною системою Android версії 8.0 (API level 26) та вище.
5) Безпека:
    1) API-ключ для сервісу перекладу повинен зберігатися безпечно та не бути доступним у відкритому вигляді в коді застосунку.

2.2 Архітектура системи

2.2.1 Вибір архітектурного патерну та його аргументація

MVVM (Model-View-ViewModel) – це архітектурний патерн, який сприяє чіткому розділенню логіки представлення даних (View) від бізнес-логіки (ViewModel) та даних (Model). Цей патерн забезпечує високу гнучкість, тестування та підтримуваність коду, що є особливо важливим для складних мобільних застосунків [13].

Компоненти MVVM у контексті Android:

1. View (Представлення): Відповідає за відображення інтерфейсу користувача та обробку подій від користувача. В Android це Activity або Fragment. View спостерігає за змінами даних у ViewModel та оновлює UI відповідно.
2. ViewModel (Модель представлення): Містить логіку представлення даних для View. ViewModel не має прямого доступу до View, але надає дані через LiveData, які View може спостерігати. ViewModel також обробляє запити від View (наприклад, натискання кнопок) та взаємодіє з Model для отримання або збереження даних. Ключовою перевагою ViewModel є її життєвий цикл, який переживає зміни конфігурації (наприклад, поворот екрану), запобігаючи втраті даних.
3. Model (Модель): Представляє бізнес-логіку та дані застосунку. Включає репозиторії, джерела даних (база даних, мережеві API). Model не залежить від View або ViewModel.

Аргументація вибору MVVM:

1. Розділення відповідальності: Чітке розділення між UI, логікою UI та бізнес-логікою робить код більш організованим та легким для розуміння.
2. Тестованість: ViewModel не залежить від Android SDK, що дозволяє легко тестувати бізнес-логіку без використання емулятора або пристрою.

3. Життєвий цикл: ViewModel та LiveData інтегровані з життєвим циклом Android-компонентів, що допомагає уникнути витоків пам'яті та спрощує управління станом UI при змінах конфігурації.
4. Підтримуваність та масштабованість: Зміни в UI не впливають на бізнес-логіку, і навпаки, що спрощує внесення змін та додавання нового функціоналу.

На **Помилка! Джерело посилання не знайдено.** представлена у загальнена діаграма архітектури MVVM, що ілюструє взаємодію між її основними компонентами в контексті Android-застосунку.

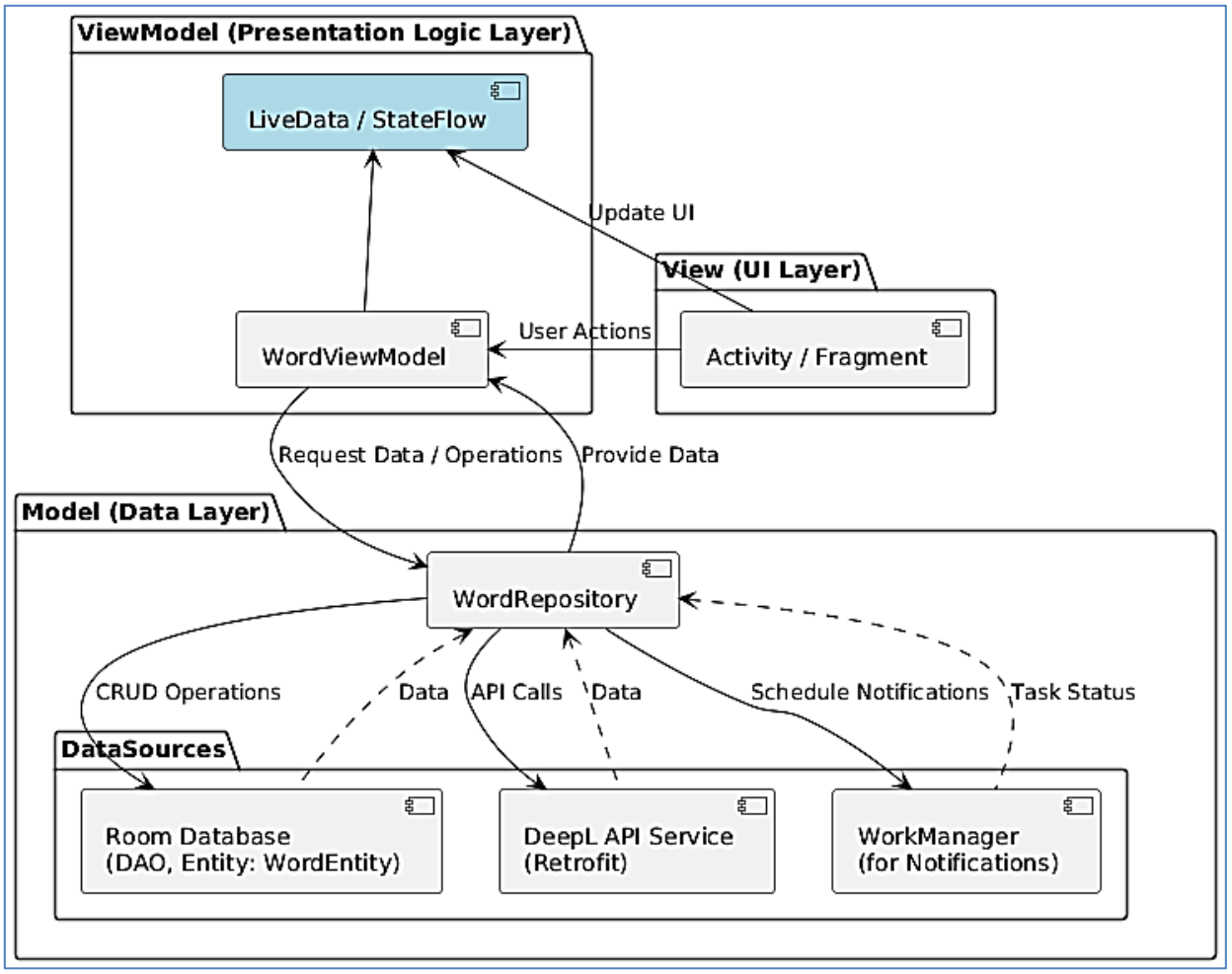


Рисунок 4 – Модель архітектури MVVM для Android застосунку

2.2.2 Структура даних та база даних

Для локального зберігання даних застосунку, зокрема словника користувача, було обрано Room Persistence Library. Room є частиною Android Jetpack і надає абстракцію над SQLite, спрощуючи роботу з базами даних та забезпечуючи компіляційну перевірку SQL-запитів.

Основні компоненти Room:

1. Entity (Сутність): Представляє таблицю в базі даних. Кожен екземпляр сутності відповідає рядку в таблиці.
2. DAO (Data Access Object): Інтерфейс, що містить методи для взаємодії з базою даних (вставка, оновлення, видалення, запити).
3. Database (База даних): Абстрактний клас, що є точкою доступу до бази даних та DAO.

2.2.3 Технології для локальних сповіщень

На платформі Android для планування та виконання відкладених, але гарантованих фонових завдань, таких як надсилання сповіщень, оптимальним рішенням є використання бібліотеки WorkManager з Android Jetpack. WorkManager надає гнучкий та надійний API для планування фонової роботи.

Переваги використання WorkManager для локальних сповіщень:

1. Гарантоване виконання: WorkManager забезпечує виконання запланованих завдань, навіть якщо пристрій перезавантажується або застосунок примусово закривається.
2. Стійкість до обмежень системи: Він враховує системні обмеження, такі як Doze Mode та App Standby, оптимізуючи використання батареї та забезпечуючи доставку сповіщень.
3. Гнучкість у плануванні: Дозволяє планувати одноразові або періодичні завдання з різними умовами (наприклад, наявність мережі, заряд батареї).
4. Простота інтеграції: Спрощує роботу з фоновими завданнями порівняно з нижньорівневими API, такими як AlarmManager або JobScheduler.

2.2.4 Модулі застосунку

Основні модулі застосунку:

1. Модуль даних (Model Layer): Модуль є основою для роботи з усіма джерелами даних застосунку. Він повністю абстрагує логіку отримання, зберігання та обробки даних від UI та бізнес-логіки.
2. Модуль бізнес-логіки та представлення (ViewModel Layer): Модуль містить ViewModel, які є посередниками між View та Model. Вони готують дані для відображення, обробляють дії користувача та взаємодіють з репозиторієм. ViewModel не мають прямого доступу до View, що забезпечує їхню незалежність від життєвого циклу UI та легкість тестування.
3. Модуль користувацького інтерфейсу (View Layer): Модуль складається з Activity та Fragment, які відповідають за відображення UI та обробку взаємодії з користувачем. Вони спостерігають за LiveData з відповідних ViewModel та оновлюють UI.
4. Модуль утиліт (util): Містить загальні допоміжні класи, які можуть бути використані в різних частинах застосунку.
5. Модуль фонових завдань (workers): Містить класи, що виконують фонові операції.

2.2.5 Користувацький інтерфейс

Користувацький інтерфейс (UI) мобільного застосунку розроблено з акцентом на інтуїтивність, простоту використання та візуальну привабливість, дотримуючись принципів Material Design [8]. Метою є забезпечення максимальної зручності для користувача та мінімізація кількості кроків для виконання основних операцій.

Застосунок побудований навколо MainActivity, яка слугує основним контейнером для всіх фрагментів.

2.3 Висновки до розділу 2

1. Детально проаналізовано та сформульовано функціональні та нефункціональні вимоги до мобільного застосунку, що забезпечує чітке розуміння очікуваного функціоналу та характеристик системи.
2. Обґрунтовано вибір архітектурного патерну MVVM, що сприятиме розробці гнучкого, тестованого та підтримуваного коду, а також описано його ключові компоненти в контексті Android-розробки.
3. Спроектовано структуру бази даних з використанням Room Persistence Library для зберігання словника користувача та обрано WorkManager для реалізації надійної системи локальних сповіщень.
4. Визначено основні модулі застосунку та закладено принципи побудови користувацького інтерфейсу на основі Material Design, що забезпечить логічну організацію коду та зручну взаємодію з користувачем.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ

У цьому розділі детально описано процес реалізації основного функціоналу мобільного застосунку для вивчення іноземних слів, включаючи опис структури проєкту, архітектури MVVM, реалізацію шару даних, управління словником, режими навчання, систему сповіщень та користувацький інтерфейс.

3.1 Середовище розробки та інструменти

Розробка мобільного застосунку здійснювалася з використанням сучасних та рекомендованих інструментів для платформи Android. Основним середовищем розробки слугувало Android Studio, офіційне IDE від Google. Мовою програмування було обрано Kotlin, завдяки її лаконічності, безпеці та повній сумісності з Java. Система збірки проєкту базується на Gradle.
Ключові компоненти та бібліотеки, використані в проєкті, включають:

1. Android SDK: Набір інструментів для розробки Android-застосунків.
2. Android Jetpack: Колекція бібліотек для спрощення розробки, зокрема:
    1) ViewModel: Для реалізації ViewModel у патерні MVVM.
    2) LiveData: Для створення спостережуваних об'єктів даних.
    3) Room Persistence Library: Для роботи з локальною базою даних SQLite.
    4) Navigation Component: Для реалізації навігації між екранами.
    5) WorkManager: Для планування фонових завдань (локальні сповіщення).
3. Retrofit: Для мережевої взаємодії з DeepL API.
4. Gson: Для серіалізації/десеріалізації JSON-даних.
5. Kotlin Coroutines: Для асинхронного програмування.

3.2 Реалізація основного функціоналу

3.2.1 Структура проєкту та архітектура MVVM

Архітектура застосунку базується на патерні MVVM (Model-View-ViewModel), що забезпечує чітке розділення відповідальності між компонентами, покращує тестованість та підтримуваність коду.

Файлова структура проєкту, організована за функціональними модулями та шарами архітектури, представлена на Рисунок 5.

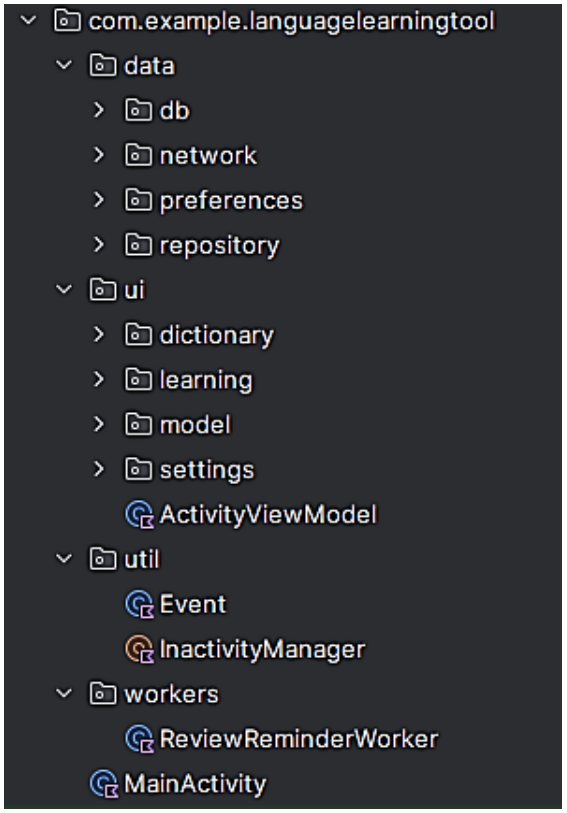


Рисунок 5 – Файлова структура проекту

Навігація в застосунку реалізована за допомогою Android Navigation Component. MainActivity містить NavHostFragment, який управляє переходами між різними фрагментами, визначеними у графі навігації (res/navigation/nav_graph.xml). NavController використовується для програмної навігації між екранами.

Зв'язок між View (фрагментами) та ViewModel здійснюється за

допомогою LiveData. ViewModel надає LiveData об'єкти, на які підписується View. При зміні даних у LiveData, View автоматично оновлює відповідні елементи UI. Це забезпечує реактивність інтерфейсу та дотримання життєвого циклу компонентів Android. На Лістингу 1 показано приклад спостереження за wordList у DictionaryViewModel з боку DictionaryFragment.

Лістинг 1 – Приклад спостереження LiveData у DictionaryFragment.kt

```
dictionaryViewModel.wordList.observe(viewLifecycleOwner) {
words ->
    words?.let {
        dictionaryAdapter.submitList(it)
        binding.tvEmptyList.isVisible = it.isEmpty()
        binding.rvWords.isVisible = it.isNotEmpty()
    }
}
```

У цьому фрагменті коду DictionaryFragment підписується на wordList (об'єкт LiveData<List<Word>>) з DictionaryViewModel. Коли список слів оновлюється у ViewModel, лямбда-вираз виконується, і адаптер dictionaryAdapter отримує новий список для відображення. Також оновлюється видимість текстового поля tvEmptyList та списку rvWords залежно від того, чи порожній список слів.

У MainActivity.kt використовується ActivityViewModel для управління станом верхньої панелі (Toolbar), включаючи заголовок, видимість кнопки додавання (FAB), кнопки збереження та поля пошуку. Це дозволяє різним фрагментам динамічно змінювати вигляд та функціонал Toolbar відповідно до поточного контексту (див. Лістинг 2).

Лістинг 2 – Фрагмент MainActivity.kt, що демонструє оновлення UI через ActivityViewModel

```
activityViewModel.topBarState.observe(this) { state ->
    if (state.title.isNotEmpty()) {
        binding.navTitle.text = state.title
    } else {
        navController.currentDestination?.label?.let {
```

```
            binding.navTitle.text = it.toString()
        } ?: run {
            binding.navTitle.text =
getString(R.string.app_name)
        }
    }

    binding.fabAdd.visibility = if (state.isFabVisible)
View.VISIBLE else View.GONE
    binding.saveWordBtn.visibility =
        if (state.isSaveButtonVisible) View.VISIBLE else
View.GONE

    val searchViewVisible = state.isSearchViewVisible
    binding.searchView.visibility = if (searchViewVisible)
View.VISIBLE else View.GONE
}
```

Цей код показує, як MainActivity спостерігає за topBarState з ActivityViewModel. При зміні стану (наприклад, при переході на новий фрагмент, який оновлює цей стан), відповідні елементи UI, такі як заголовок (binding.navTitle), FAB (binding.fabAdd), кнопка збереження (binding.saveWordBtn) та поле пошуку (binding.searchView), оновлюють свій вигляд та видимість.

3.2.2 Реалізація шару даних

Шар даних (Model) відповідає за надання та управління даними застосунку. Він інкапсулює логіку роботи з різними джерелами даних: локальною базою даних SQLite (через Room), мережевим API (DeepL) та SharedPreferences. Основним компонентом цього шару є WordRepository.

Локальна база даних (Room)

Для зберігання словника користувача (слів, їх перекладів та мов) використовується бібліотека Room.

1. **Entity (Word.kt)**: Клас даних, що представляє таблицю words_table у базі даних. Кожен екземпляр Word відповідає одному запису (слову) у словнику (див. Табл. 1).

Таблиця 1 – Структура даних таблиці Word

Поле	Тип даних	Опис
id	Int	Унікальний ідентифікатор слова
originalWord	String	Оригінальне слово або фраза
translation	String	Переклад слова або фрази
sourceLanguage	String	Код мови оригіналу
targetLanguage	String	Код мови перекладу

2. **DAO (WordDao.kt)**: Інтерфейс Data Access Object, що визначає методи для взаємодії з таблицею слів. Включає операції CRUD (Create, Read, Update, Delete), пошук, отримання випадкових слів та перевірку на існування дублікатів. Запити до бази даних визначені за допомогою анотацій Room (див. Лістинг 3).

Лістинг 3 – Фрагмент анотацій у WordDao

```
@Dao
interface WordDao {
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun insert(word: Word)

    @Query("SELECT * FROM words_table ORDER BY originalWord
ASC")
    fun getAllWords(): LiveData<List<Word>>

    @Query("SELECT * FROM words_table WHERE originalWord
LIKE :searchQuery OR translation LIKE :searchQuery")
    fun searchWords(searchQuery: String):
LiveData<List<Word>>

    @Query("SELECT * FROM words_table ORDER BY RANDOM()
LIMIT :count")
    suspend fun getRandomWords(count: Int): List<Word>

    @Delete
```

```
    suspend fun deleteWords(words: List<Word>)

    @Update
    suspend fun update(word: Word)

    @Query("""
        SELECT COUNT(*) FROM words_table
        WHERE (LOWER(originalWord) = LOWER(:originalWord)
AND sourceLanguage = :sourceLanguage AND targetLanguage =
:targetLanguage AND LOWER(translation) =
LOWER(:translation))
           OR (LOWER(originalWord) = LOWER(:translation)
AND sourceLanguage = :targetLanguage AND targetLanguage =
:sourceLanguage AND LOWER(translation) =
LOWER(:originalWord))
    """)
    suspend fun getExistingWordCount(
        originalWord: String,
        translation: String,
        sourceLanguage: String,
        targetLanguage: String
    ): Int
}
```

1) insert(word: Word): Асинхронний метод для вставки нового слова. OnConflictStrategy.IGNORE означає, що якщо слово з таким id вже існує, вставка буде проігнорована.
2) getAllWords(): LiveData<List<Word>>: Повертає LiveData список всіх слів, відсортованих за оригінальним словом. LiveData дозволяє UI автоматично оновлюватися при зміні даних.
3) searchWords(searchQuery: String): LiveData<List<Word>>: Повертає LiveData список слів, що відповідають пошуковому запиту (пошук по оригінальному слову або перекладу).
4) getRandomWords(count: Int): List<Word>: Асинхронний метод для отримання вказаної кількості випадкових слів, використовується для режимів навчання.
5) deleteWords(words: List<Word>): Асинхронний метод для видалення списку слів.
6) update(word: Word): Асинхронний метод для оновлення існуючого

слова.

7) getExistingWordCount(...): Асинхронний метод для перевірки, чи існує вже слово з такою ж парою "оригінал-переклад" (або навпаки) та мовами. Використовується для уникнення дублікатів.

3. **Database (AppDatabase.kt)**: Абстрактний клас, що успадковується від RoomDatabase. Він визначає список Entities (Word::class) та надає доступ до DAO (wordDao()). Реалізований як Singleton для забезпечення єдиного екземпляру бази даних у застосунку (див. Лістинг 4).

Лістинг 4 – Фрагмент AppDatabase

```
@Database(entities = [Word::class], version = 1,
exportSchema = false)
abstract class AppDatabase : RoomDatabase() {

    abstract fun wordDao(): WordDao

    companion object {
        @Volatile
        private var INSTANCE: AppDatabase? = null

        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java,
                    "word_database"
                )
                    .build()
                INSTANCE = instance
                instance
            }
        }
    }
}
```

1) @Database(...): Анотація, що позначає клас як базу даних Room. entities вказує на класи сутностей, version — версія схеми бази даних, exportSchema = false вимикає експорт схеми.
2) wordDao(): Абстрактний метод, що повертає екземпляр WordDao. Room

автоматично генерує його реалізацію.

3) companion object: Містить логіку для створення Singleton екземпляру AppDatabase.
4) @Volatile private var INSTANCE: Забезпечує, що значення INSTANCE завжди актуальне для всіх потоків.
5) getDatabase(context: Context): Метод для отримання екземпляру бази даних. Використовує synchronized блок для потокобезпечного створення екземпляру. Room.databaseBuilder використовується для конфігурації та створення бази даних з ім'ям "word_database".

Мережева взаємодія (Retrofit та DeepL API)

Для автоматичного перекладу слів використовується DeepL API. Взаємодія з API реалізована за допомогою бібліотеки Retrofit.

4. **API Service (DeepLApiService.kt)**: Інтерфейс, що описує HTTP-запити до DeepL API. Використовує анотації Retrofit для визначення ендпоінтів, методу запиту (POST), заголовків та тіла запиту (див. Лістинг 5).

Лістинг 5 – Інтерфейс DeepLApiService

```
interface DeepLApiService {
    companion object {
        const val BASE_URL_FREE = https://api-
free.deepl.com/v2/

    }
    @FormUrlEncoded
    @POST("translate")
    suspend fun translateText(
        @Header("Authorization") authHeader: String,
        @Field("text") textToTranslate: String,
        @Field("target_lang") targetLanguage: String,
        @Field("source_lang") sourceLanguage: String? =
null
    ): Response<DeepLResponse>
}
```

1. BASE_URL_FREE: Базова URL для безкоштовної версії DeepL API.
2. @FormUrlEncoded: Вказує, що тіло запиту буде URL-кодованим.

3. @POST("translate"): Визначає POST-запит до ендпоінту "translate".
4. suspend fun translateText(...): Асинхронна функція для виконання запиту на переклад.
    1) @Header("Authorization") authHeader: String: Додає заголовок "Authorization" зі значенням API-ключа.
    2) @Field("text") textToTranslate: String: Поле "text" у тілі запиту, що містить текст для перекладу.
    3) @Field("target_lang") targetLanguage: String: Поле "target_lang" для цільової мови.
    4) @Field("source_lang") sourceLanguage: String? = null: Необов'язкове поле "source_lang" для мови оригіналу.
5. Response<DeepLResponse>: Тип відповіді, обгорнутий у retrofit2.Response для доступу до метаданих відповіді (код стану, заголовки тощо). DeepLResponse — це клас даних для розпарсингу JSON-відповіді.

5. **Data Classes (DeepLResponse.kt)**: Класи даних для десеріалізації JSON-відповіді від DeepL API за допомогою Gson (див. Лістинг 6).

Лістинг 6 – Класи даних DeepLResponse.kt

```
data class DeepLResponse(
    val translations: List<Translation>
)

data class Translation(
    @SerializedName("detected_source_language")
    val detectedSourceLanguage: String,
    val text: String
)
```

1. DeepLResponse: Основний клас відповіді, містить список об'єктів Translation.
2. Translation: Містить визначену мову оригіналу (detectedSourceLanguage) та перекладений текст (text).

3. @SerializedName("detected_source_language"): Анотація Gson, що вказує на відповідність поля JSON detected_source_language полю Kotlin detectedSourceLanguage.

6. **Retrofit Instance (RetrofitInstance.kt)**: Об'єкт-singleton, що створює та налаштовує екземпляр Retrofit, включаючи базову URL та конвертер Gson (див. Лістинг 7).

Лістинг 7 – Об'єкт RetrofitInstance.kt

```
object RetrofitInstance {

    private val loggingInterceptor =
HttpLoggingInterceptor().apply {
        level = HttpLoggingInterceptor.Level.BODY
    }

    private val client = OkHttpClient.Builder()
        .addInterceptor(loggingInterceptor
        .build()

    val api: DeepLApiService by lazy {
        Retrofit.Builder()
            .baseUrl(DeepLApiService.BASE_URL_FREE)
            .client(client)

.addConverterFactory(GsonConverterFactory.create())
            .build()
            .create(DeepLApiService::class.java)
    }
}
```

Цей об'єкт надає готовий до використання екземпляр DeepLApiService через властивість api. Логування HTTP запитів та відповідей корисне для налагодження.

Налаштування застосунку (SharedPreferences)

Для зберігання налаштувань користувача, зокрема параметрів сповіщень (чи ввімкнені інтервали повторень), використовується AppSettingsManager.kt. Цей клас інкапсулює роботу з SharedPreferences, надаючи зручні методи для

збереження та завантаження налаштувань. (див. Лістинг 8).

Лістинг 8 – Фрагмент AppSettingsManager.kt

```
class AppSettingsManager(context: Context) {

    companion object {
        const val PREFS_APP_SETTINGS = "app_settings_prefs"
        const val KEY_NOTIFICATIONS_ENABLED =
"notifications_enabled"
        const val KEY_CURRENT_INTERVAL_INDEX =
"current_notification_interval_index"
    }

    private val sharedPreferences: SharedPreferences =

context.applicationContext.getSharedPreferences(PREFS_APP_S
ETTINGS, Context.MODE_PRIVATE)

    fun setNotificationsEnabled(enabled: Boolean) {
sharedPreferences.edit().putBoolean(KEY_NOTIFICATIONS_ENABL
ED, enabled).apply()
    }

    fun areNotificationsEnabled(): Boolean {
        return
sharedPreferences.getBoolean(KEY_NOTIFICATIONS_ENABLED,
false)
    }

    fun setCurrentIntervalIndex(index: Int) {
        sharedPreferences.edit()
            .putInt(KEY_CURRENT_INTERVAL_INDEX,
index.coerceIn(0, NUM_INTERVALS - 1)).apply()
    }

    fun getCurrentIntervalIndex(): Int {
        return
sharedPreferences.getInt(KEY_CURRENT_INTERVAL_INDEX, 0)
    }

    fun saveAllIntervalSettings(settings:
List<IntervalSetting>)

    fun getIntervalSetting(intervalIndex: Int):
IntervalSetting
```

1. PREFS_APP_SETTINGS: Ім'я файлу SharedPreferences.
2. KEY_NOTIFICATIONS_ENABLED, KEY_CURRENT_INTERVAL_INDEX: Приклади ключів для зберігання налаштувань.
3. Методи setNotificationsEnabled та areNotificationsEnabled керують станом увімкнення сповіщень.
4. Методи setCurrentIntervalIndex та getCurrentIntervalIndex керують поточним індексом в циклі інтервалів сповіщень.
5. Методи saveAllIntervalSettings та getIntervalSetting дозволяють зберігати та отримувати конфігурації для кожного з п'яти доступних інтервалів сповіщень, включаючи значення та одиницю виміру часу (хвилини, години, дні).

Репозиторій (WordRepository.kt)

WordRepository є ключовим елементом шару даних. Він слугує єдиною точкою доступу до даних для ViewModel, абстрагуючи деталі їх отримання та зберігання (див. Лістинг 9).

Основні функції WordRepository:

1. Інкапсулює логіку взаємодії з WordDao (для локальних даних) та DeepLApiService (для перекладу через мережу).
2. Надає методи для додавання слів (з автоматичним перекладом), оновлення, видалення, пошуку та отримання слів.
3. Використовує Kotlin Coroutines (зокрема withContext(Dispatchers.IO)) для виконання операцій з базою даних та мережевих запитів у фоновому потоці, не блокуючи основний потік UI.
4. Обробляє можливі помилки, наприклад, WordAlreadyExistsException при спробі додати слово, яке вже існує.
5. API-ключ для DeepL отримується з BuildConfig.DEEPL_API_KEY, що забезпечує його безпечне зберігання поза кодом.
6. Реалізує логіку очищення тексту (cleanTextForStorage) перед

збереженням слів (видалення зайвих пробілів, розділових знаків наприкінці, переведення в нижній регістр).

Лістинг 9 – Фрагмент методу insertWordWithTranslation з WordRepository.kt

```
class WordRepository(application: Application) {
    // ...
    suspend fun insertWordWithTranslation(
        userInputOriginalWord: String,
        sourceLang: String,
        targetLang: String
    ) {
        withContext(Dispatchers.IO) {
            val cleanedOriginal =
cleanTextForStorage(userInputOriginalWord)
            // ... перевірка на порожній рядок
            try {
                val response =
deepLApiService.translateText(
                    authHeader = deepLApiKey,
                    textToTranslate = cleanedOriginal,
                    sourceLanguage = sourceLang,
                    targetLanguage = targetLang
                )
                if (response.isSuccessful &&
response.body() != null) {
                    val rawTranslation =
response.body()?.translations?.firstOrNull()?.text
                    // ... обробка перекладу, перевірка на
існування, збереження
                    val cleanedTranslation =
cleanTextForStorage(rawTranslation!!)
                    val existingCount =
wordDao.getExistingWordCount(
                        originalWord = cleanedOriginal,
                        translation = cleanedTranslation,
                        sourceLanguage = sourceLang,
                        targetLanguage = targetLang
                    )
                    if (existingCount > 0) {
                        throw
WordAlreadyExistsException("The word pair ... already
exists.")
                    }
                    val newWord = Word(/* ... */)
                    wordDao.insert(newWord)
```

```
                } else {
                    // ... обробка помилки API
                }
            } catch (e: WordAlreadyExistsException) {
                throw e
            } catch (e: Exception) {
                // ... обробка інших помилок
            }
        }
    }
    // ... інші методи
}
```

У цьому методі відбувається очищення введеного слова, запит на переклад до DeepL API, очищення отриманого перекладу, перевірка на існування дубліката в базі даних, і, якщо все гаразд, збереження нового слова. Також обробляються різні можливі помилки.

На Рисунок 6 представлена діаграма, що ілюструє взаємодію компонентів у шарі даних.

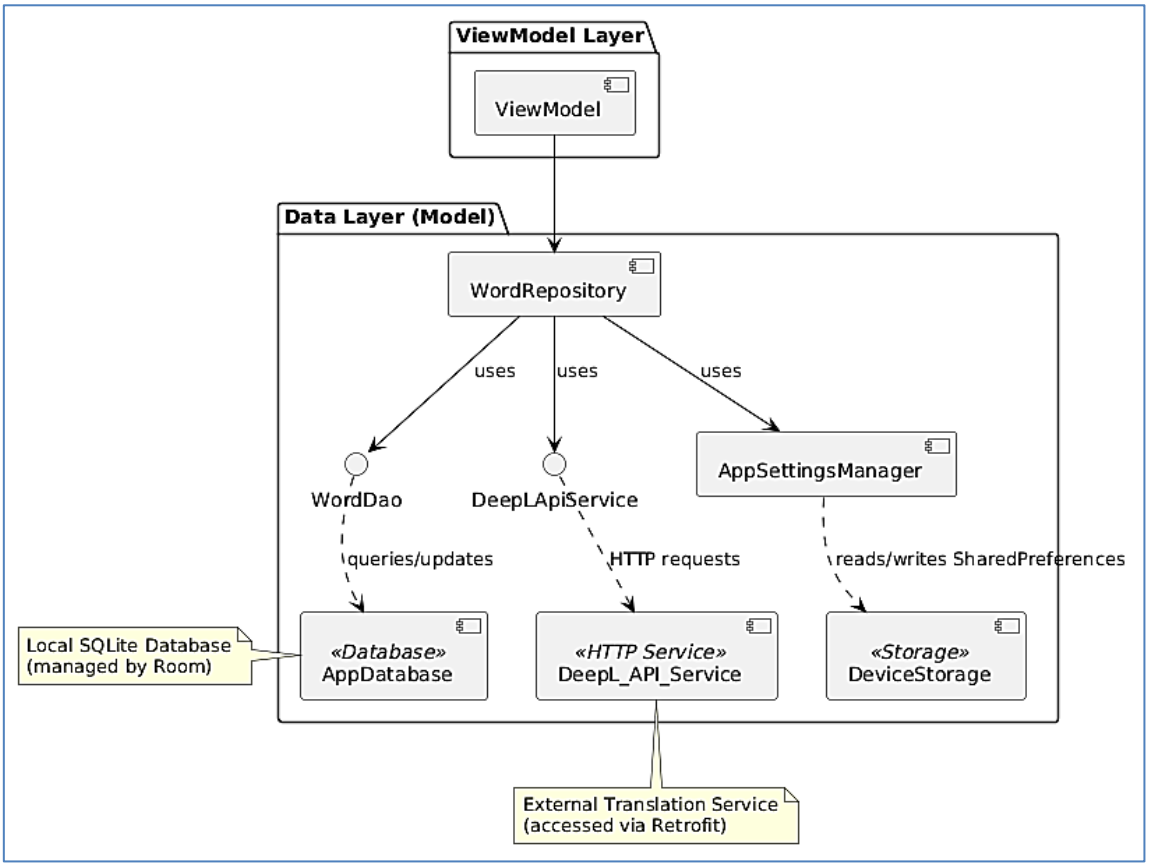


Рисунок 6 – Діаграма взаємодії компонентів у шарі даних

3.2.3 Управління словником користувача

Функціонал управління словником користувача реалізований через взаємодію кількох компонентів: фрагментів (DictionaryFragment, AddWordFragment, EditWordFragment), DictionaryViewModel та WordRepository.

Додавання слів:

Користувач додає нові слова через AddWordFragment.kt.

1. UI та введення даних: AddWordFragment містить поля для введення слова, випадаючі списки (Spinner) для вибору мови оригіналу та мови перекладу (availableLanguages, selectedSourceLanguage, selectedTargetLanguage). Кнопка "Swap Languages" дозволяє швидко поміняти місцями вибрані мови. Логіка блокування слухачів спінерів (blockSpinnerListeners) запобігає небажаним рекурсивним оновленням при програмній зміні вибору в спінерах.
2. Обробка збереження: При натисканні кнопки "Зберегти" (яка управляється з MainActivity через ActivityViewModel), викликається метод handleSaveWord() у AddWordFragment.
3. Валідація: handleSaveWord() перевіряє, чи введено слово та чи вибрані різні мови.
4. Виклик ViewModel: Якщо валідація успішна, викликається dictionaryViewModel.addWord(word, sourceLang.code, targetLang.code).
5. Взаємодія ViewModel з репозиторієм: DictionaryViewModel.addWord() (див. Лістинг 10) встановлює стан завантаження (_isLoading.value = true), викликає repository.insertWordWithTranslation(), обробляє можливі винятки (WordAlreadyExistsException, Exception) та оновлює _errorMessage або _wordAddedSuccessfullyEvent.

Лістинг 10 – Метод addWord у DictionaryViewModel.kt

```
fun addWord(originalWord: String, sourceLang: String,
targetLang: String) {
    viewModelScope.launch {
```

```
            _isLoading.value = true
            try {

repository.insertWordWithTranslation(originalWord,
sourceLang, targetLang)
                _wordAddedSuccessfullyEvent.value =
Event(originalWord.trim())
            } catch (e: WordAlreadyExistsException) {
                // обробка помилок
            } catch (e: Exception) {
                // обробка помилок
            } finally {
                _isLoading.value = false
            }
        }
}
```

Після успішного додавання слова поле введення в AddWordFragment очищується, і відображається повідомлення Toast.

Перегляд слів:

Слова відображаються у DictionaryFragment.kt за допомогою RecyclerView та DictionaryAdapter.kt.

1. Отримання списку слів: DictionaryFragment спостерігає за dictionaryViewModel.wordList. wordList у DictionaryViewModel є MediatorLiveData, який комбінує дані з baseWordList (результат repository.allWords або repository.searchWords) та фільтри за мовами (_selectedSourceFilterLanguage, _selectedTargetFilterLanguage).
2. Фільтрація та трансформація: Метод filterAndTransformWords() у DictionaryViewModel відповідає за фільтрацію слів за вибраними мовами оригіналу та перекладу. Особливістю є те, що якщо слово не відповідає фільтру в прямому вигляді (originalWord як source, translation як target), перевіряється його "зворотна" форма (translation як source, originalWord як target), і якщо вона відповідає, то така трансформована форма додається до результату. Це дозволяє користувачеві бачити слово, навіть якщо він встановив фільтр, наприклад, "EN -> UK", а слово

в базі збережено як "UK -> EN". Результат сортується за оригінальним словом.

Редагування слів:

Редагування відбувається у EditWordFragment.kt.

1. Навігація та завантаження слова: DictionaryFragment при виборі опції "Редагувати" (доступно, коли вибрано лише одне слово) навігує до EditWordFragment, передаючи wordId як аргумент. EditWordFragment викликає dictionaryViewModel.loadWordForEditing(wordId), який завантажує дані слова з репозиторію та зберігає їх у _wordToEdit LiveData.
2. Заповнення форми: EditWordFragment спостерігає за _wordToEdit та заповнює поля форми (оригінальне слово, переклад, мови) даними завантаженого слова.
3. Збереження змін: При натисканні кнопки "Зберегти", EditWordFragment.handleSaveChanges() збирає дані з форми, проводить валідацію та викликає dictionaryViewModel.updateWord(...).
4. Оновлення у ViewModel: DictionaryViewModel.updateWord() викликає repository.updateWord(), обробляє помилки (включаючи перевірку на існування дубліката, виключаючи поточне слово, через getExistingWordCountExcludingId у DAO) та сповіщає про успішне оновлення через _wordUpdatedSuccessfullyEvent. Після успішного оновлення відбувається повернення на попередній екран.

Видалення слів:

1. Режим вибору: Довгий клік на слові у DictionaryFragment активує режим вибору. DictionaryAdapter повідомляє DictionaryViewModel про зміну стану вибору слова (toggleWordSelection). DictionaryViewModel зберігає множину ID вибраних слів у _selectedWords LiveData.
2. Оновлення UI: DictionaryFragment спостерігає за isSelectionModeActive

та selectedWords для оновлення UI (зміна заголовка Toolbar, показ/приховування кнопок дій "Видалити", "Редагувати"). DictionaryAdapter також оновлює вигляд вибраних елементів.

3. Підтвердження видалення: При натисканні кнопки "Видалити", dictionaryViewModel.onDeleteSelectedClicked() готує список об'єктів Word для видалення (на основі _selectedWords та поточного baseWordList.value) та ініціює показ діалогу підтвердження через _showDeleteConfirmationEvent.
4. Видалення: Якщо користувач підтверджує видалення у діалозі (showDeleteConfirmationDialog у DictionaryFragment), викликається dictionaryViewModel.confirmDeleteSelectedWords(wordsToDelete), який, у свою чергу, викликає repository.deleteWords(). Після видалення вибір очищується.

3.2.4 Реалізація режимів навчання

Застосунок пропонує три режими навчання, кожен з яких реалізований у власному фрагменті та має відповідний ViewModel. Перехід до режимів навчання здійснюється з LearningModesFragment.kt, який відображає список доступних режимів за допомогою LearningModesAdapter.kt.

Флеш-картки (FlashcardFragment.kt, FlashcardViewModel.kt)

Цей режим дозволяє вивчати слова, переглядаючи картки з оригіналом та перекладом.

1. Завантаження слів: FlashcardViewModel.loadRandomWords(count: Int) асинхронно завантажує вказану кількість випадкових слів з WordRepository. Результат зберігається у _wordsForFlashcards LiveData.
2. Відображення картки: FlashcardFragment спостерігає за _wordsForFlashcards. updateCurrentCardDisplay() відповідає за відображення поточного слова. Спочатку показується оригінальне слово (isShowingOriginal = true).

3. Перегортання картки: Клік на картці (binding.cardView) інвертує isShowingOriginal та викликає updateCurrentCardDisplay() для показу перекладу (або навпаки).
4. Наступне слово: Кнопка "Next" (binding.btnNext) інкрементує currentWordIndex. Якщо досягнуто кінця списку, слова перемішуються (currentWords.shuffled()), індекс скидається на 0, і відображається Toast-повідомлення (див. Лістинг 11).
5. Обробка стану: ViewModel також надає isLoading та errorMessage для відображення стану завантаження та можливих помилок. Якщо слів немає, відображається відповідне повідомлення (binding.tvNoWords). Також лічильник слів (binding.tvFlashcardWordCount) показує прогрес.

Лістинг 11 – Фрагмент FlashcardFragment.kt: логіка перегортання та наступного слова

```
private fun flipCard() {
    if (currentWords.isEmpty() ||
(flashcardViewModel.isLoading.value == true)) return
    isShowingOriginal = !isShowingOriginal
    updateCurrentCardDisplay()
}

private fun showNextWord() {
    if (currentWords.isEmpty() ||
(flashcardViewModel.isLoading.value == true)) {
        updateCurrentCardDisplay()
        return
    }

    currentWordIndex++
    if (currentWordIndex >= currentWords.size) {
        currentWordIndex = 0
        currentWords = currentWords.shuffled()
        Toast.makeText(requireContext(),
getString(R.string.words_finished_shuffling),
Toast.LENGTH_SHORT).show()
        if (currentWords.isEmpty()) {
            updateCurrentCardDisplay()
            return
        }
    }
    isShowingOriginal = true
```

```
    updateCurrentCardDisplay()
}
```

Введення перекладу (TypeTranslationFragment.kt, TypeTranslationViewModel.kt)

У цьому режимі користувачеві показується слово однією мовою, і він має ввести його переклад.

1. Завантаження слів: TypeTranslationViewModel.loadWordsForTyping (count: Int) завантажує випадкові слова та зберігає їх у wordsForTyping. Перше слово для вгадування встановлюється у _currentWordToGuess.
2. Відображення завдання: TypeTranslationFragment спостерігає за _currentWordToGuess та відображає слово, яке потрібно перекласти. Підказка в полі введення (binding.inputLayoutAnswer.hint) динамічно змінюється, вказуючи мову, якою потрібно ввести переклад.
3. Перевірка відповіді: Користувач вводить відповідь у EditText (binding.etEnglishAnswer). Натискання кнопки "Check Answer" (binding.btnCheckAnswer) або клавіші "Done" на клавіатурі викликає handleButtonClick(). Цей метод викликає typeTranslationViewModel. checkTypedAnswer(userAnswer).
4. Логіка ViewModel: TypeTranslationViewModel.checkTypedAnswer() порівнює відповідь користувача (після trim() та ігноруючи регістр) з правильним перекладом (currentWord.originalWord). Результат встановлюється у _checkResultState LiveData (типу CheckResult, який є sealed class з варіантами Idle, Correct, Incorrect).
5. Зворотний зв'язок: TypeTranslationFragment спостерігає за _checkResultState.
    1) Idle: Очищує поле введення, вмикає його, встановлює фокус, показує клавіатуру. Кнопка має текст "Check Answer".
    2) Correct: Показує повідомлення "Correct", вимикає поле введення. Кнопка змінює текст на "Next Word".
    3) Incorrect: Показує повідомлення з правильною відповіддю. Кнопка

змінює текст на "Next Word".

6. Наступне слово: Якщо стан Correct або Incorrect, натискання кнопки викликає typeTranslationViewModel.prepareNextWordForTyping(), який завантажує наступне слово або перемішує список, якщо досягнуто кінця.
7. Стани: Також обробляються isLoading, errorMessage, noWordsForTypingAvailable та currentWordDisplayInfo (лічильник слів).

Множинний вибір (MultipleChoiceFragment.kt, MultipleChoiceViewModel.kt)

Користувачеві пропонується слово та кілька варіантів перекладу, з яких потрібно вибрати правильний.

Завантаження слів та генерація запитання: MultipleChoiceViewModel.loadWordsForQuiz(minWordsRequired: Int) завантажує слова. prepareNextQuestion() вибирає поточне слово (currentWord), визначає правильну відповідь (currentWord.translation) та генерує неправильні варіанти з інших слів. Варіанти перемішуються. Створюється об'єкт QuizQuestion, і стан встановлюється в _quizState.value = QuizState.QuestionDisplayed(question). QuizState є sealed class, який містить Idle, QuestionDisplayed, AnsweredCorrectly, AnsweredIncorrectly.

3.2.5 Система локальних сповіщень та інтервального повторення

Система локальних сповіщень реалізована для нагадування користувачеві про необхідність повторення слів, базуючись на принципі інтервального повторення. Ключовими компонентами є AppSettingsManager, SettingsFragment та SettingsViewModel для налаштувань, InactivityManager для логіки планування та ReviewReminderWorker для виконання фонового завдання та показу сповіщення.

Налаштування сповіщень (AppSettingsManager.kt, SettingsFragment.kt, SettingsViewModel.kt)

1. AppSettingsManager зберігає:

   1) Чи ввімкнені сповіщення (KEY_NOTIFICATIONS_ENABLED).
   2) Поточний індекс в циклі інтервалів (KEY_CURRENT_INTERVAL_INDEX).
   3) Налаштування для кожного з NUM_INTERVALS (за замовчуванням 5) інтервалів: значення (KEY_INTERVAL_VALUE_PREFIX) та одиницю виміру (KEY_INTERVAL_UNIT_ORDINAL_PREFIX — хвилини, години, дні, представлені в TimeUnitOption enum).
2. SettingsFragment надає UI для зміни цих налаштувань: перемикач для ввімкнення/вимкнення сповіщень та поля для налаштування кожного інтервалу.
3. SettingsViewModel завантажує та зберігає ці налаштування через AppSettingsManager. При ввімкненні сповіщень викликає InactivityManager.resetIntervalCycle() та InactivityManager.scheduleInactivityReminder(). При вимкненні — InactivityManager.cancelReminders(). Збереження налаштувань інтервалів також викликає перепланування сповіщення.

Планувальник сповіщень (InactivityManager.kt)

1. scheduleInactivityReminder(context: Context) (див. Лістинг 15):
   1) Перевіряє, чи ввімкнені сповіщення в AppSettingsManager. Якщо ні, скасовує всі заплановані роботи.
   2) Отримує поточний індекс циклу інтервалів та відповідне налаштування інтервалу (значення та одиницю).
   3) Конвертує налаштування інтервалу в мілісекунди (convertToMillis).
   4) Якщо затримка валідна, створює OneTimeWorkRequestBuilder<ReviewReminderWorker> з цією затримкою та передає поточний індекс циклу як вхідні дані (Data).
   5) Планує унікальну роботу (REMINDER_WORK_NAME) за допомогою WorkManager.getInstance(context).enqueueUniqueWork() з

фунцією ExistingWorkPolicy.REPLACE (якщо робота вже запланована, то вона буде замінена новою створеною).

2. cycleToNextInterval(context: Context): Збільшує поточний індекс циклу інтервалів (з переходом на 0, якщо досягнуто кінця) та зберігає його.
3. resetIntervalCycle(context: Context): Скидає індекс циклу на 0.
4. cancelReminders(context: Context): Скасовує роботу з ім'ям REMINDER_WORK_NAME.
5. onAppOpened(context: Context): Викликається при відкритті застосунку, скасовує поточне сповіщення (користувач активний).
6. onAppClosed(context: Context): Викликається при закритті застосунку (наприклад, в MainActivity.onStop()), планує наступне сповіщення.

Лістинг 12 – Метод scheduleInactivityReminder у InactivityManager.kt

```
fun scheduleInactivityReminder(context: Context) {
    val appSettingsManager =
AppSettingsManager(context.applicationContext)

    if (!appSettingsManager.areNotificationsEnabled()) {
        cancelReminders(context)
        return
    }

    val currentCycleIndex =
appSettingsManager.getCurrentIntervalIndex()
    val currentIntervalSetting =
appSettingsManager.getIntervalSetting(currentCycleIndex)

    val delayMillis = convertToMillis(
        currentIntervalSetting.value.toLong(),
        currentIntervalSetting.unit
    )

    if (delayMillis <= 0) {
        return
    }
    val workData = Data.Builder()
        .putInt(KEY_CURRENT_CYCLE_INDEX_FOR_WORKER,
currentCycleIndex)
        .build()

    val reminderWorkRequest =
```

```
OneTimeWorkRequestBuilder<ReviewReminderWorker>()
        .setInitialDelay(delayMillis,
TimeUnit.MILLISECONDS)
        .setInputData(workData)
        .build()

WorkManager.getInstance(context.applicationContext).enqueue
UniqueWork(
        REMINDER_WORK_NAME,
        ExistingWorkPolicy.REPLACE,
        reminderWorkRequest
    )
}
```

Робітник для сповіщень (ReviewReminderWorker.kt)

1. Клас, що успадковує CoroutineWorker, для виконання роботи у фоновому потоці.
2. doWork():
    1) Отримує currentCycleIndexFromWorker з вхідних даних.
    2) Перевіряє кількість слів у словнику через WordRepository.getWordCount(). Якщо слів немає, сповіщення не надсилається.
    3) Якщо слова є, викликає sendInactivityNotification() для створення та показу сповіщення.
    4) Викликає InactivityManager.cycleToNextInterval() для переходу до наступного інтервалу в циклі.
    5) Повертає Result.success() або Result.failure().
3. sendInactivityNotification(context: Context, title: String, messageBody: String): Створює канал сповіщень (якщо ще не створений), налаштовує NotificationCompat.Builder (іконка, заголовок, текст, звук) та показує сповіщення через NotificationManager.

3.2.6 Реалізація користувацького інтерфейсу та навігації

Користувацький інтерфейс (UI) застосунку спроєктовано з використанням XML-розмітки та компонентів Material Design для

забезпечення зручності та сучасного вигляду. Навігація між екранами реалізована за допомогою Android Navigation Component, що дозволяє централізовано керувати переходами та передачею даних між фрагментами.

Основною точкою входу в UI є MainActivity, розмітка якої визначена у файлі main_activity.xml. Вона містить Toolbar для відображення заголовка та контекстних дій (пошук, збереження), NavHostFragment для розміщення всіх фрагментів застосунку, та BottomNavigationView для основної навігації між трьома головними розділами: «Словник», «Навчання» та «Налаштування».

Структура навігації визначена у файлі res/navigation/nav_graph.xml. Вона описує всі екрани (фрагменти) та можливі переходи між ними. Стартовим екраном (app:startDestination) є dictionaryFragment. На Рис. 7 представлена візуальна схема графу навігації.

Згідно з графом навігації, BottomNavigationView забезпечує перемикання між трьома основними фрагментами: dictionaryFragment, learningModesFragment та settingsFragment. З цих головних екранів користувач може переходити до більш специфічних, наприклад:

1. З екрана словника (dictionaryFragment) можна перейти до додавання слова (action_dictionaryFragment_to_addWordFragment) або редагування (action_dictionaryFragment_to_editWordFragment), передаючи у другому випадку ідентифікатор слова wordId.
2. З екрана вибору режимів навчання (learningModesFragment) можна перейти до одного з трьох режимів: флеш-карток (flashcardFragment), введення перекладу (typeTranslationFragment) або множинного вибору (multipleChoiceFragment).

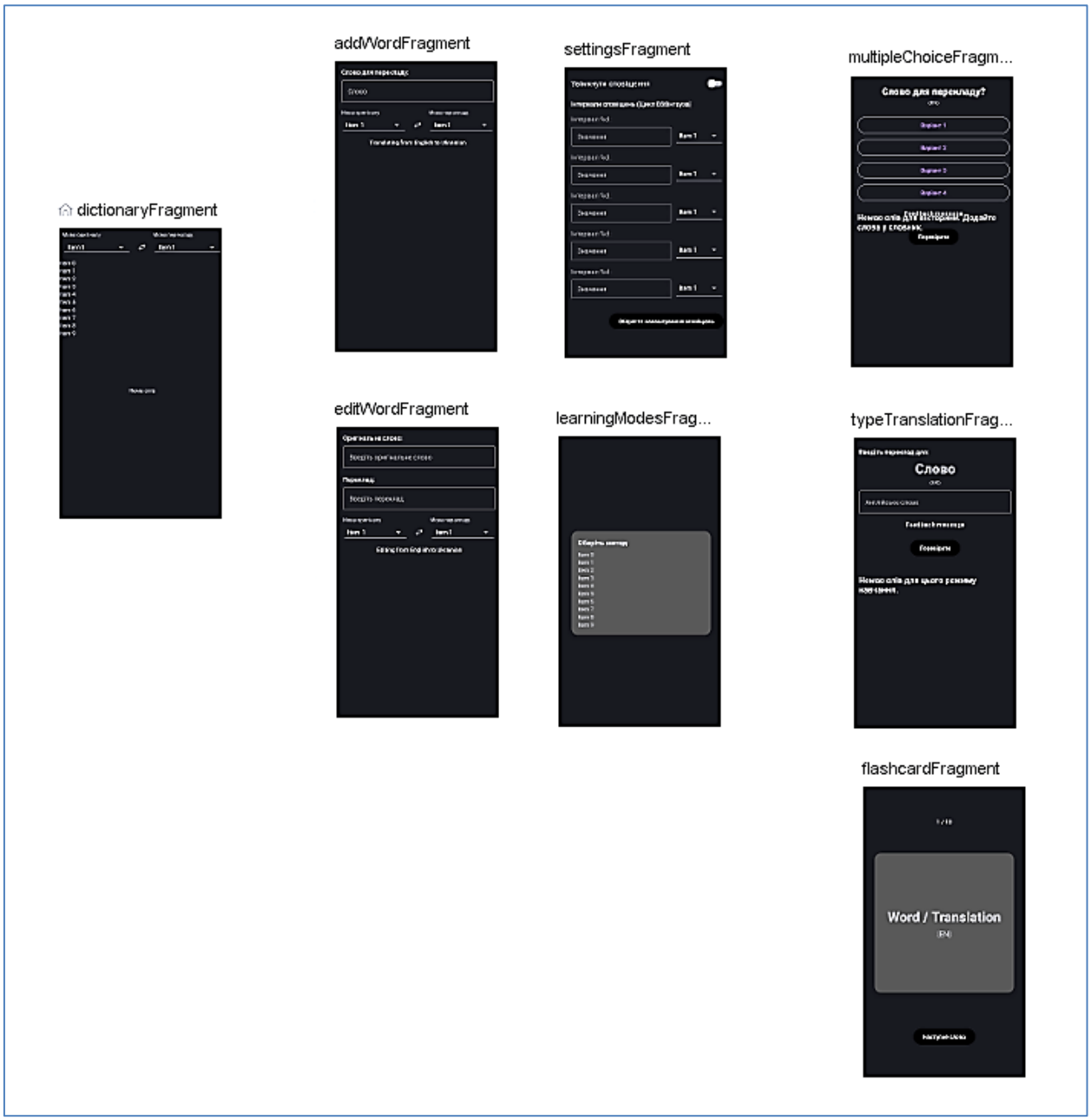


Рисунок 7 – Схема графу навігації застосунку

Основні екрани застосунку

Екран «Словник» (dictionary_fragment.xml): Це головний екран застосунку (див. Рис. 8). Він містить фільтри для вибору мови оригіналу та перекладу (Spinner), кнопку для їх швидкої заміни та RecyclerView для відображення списку слів. Якщо словник порожній, відображається відповідне повідомлення. З цього екрана здійснюється навігація до додавання, редагування та видалення слів.

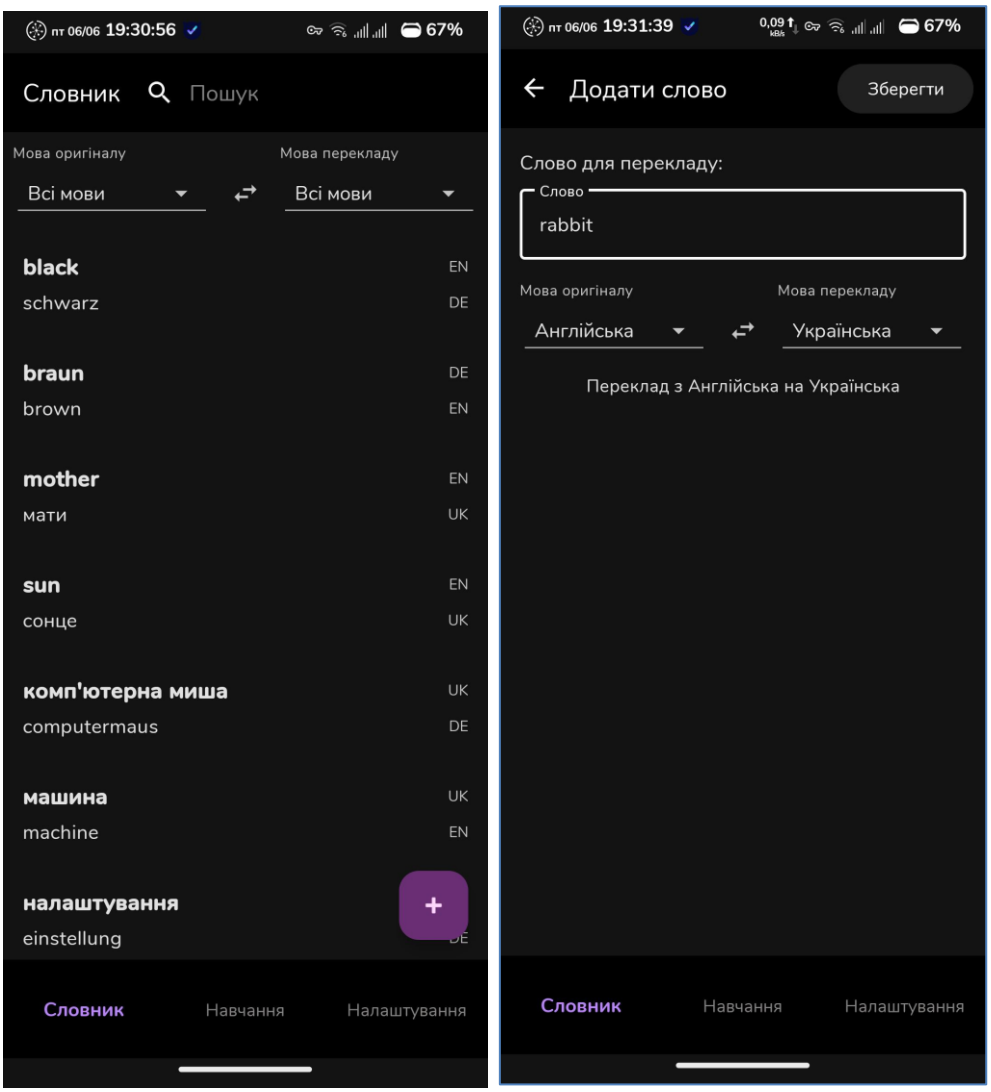


Рисунок 8 – Екрани "Словник" та "Додавання слова"

Екран «Додавання слова» (dictionary_add_word_fragment.xml): Фрагмент, що дозволяє користувачеві ввести нове слово (TextInputEditText), вибрати мову оригіналу та перекладу за допомогою Spinner (див. Рис. 8).

Екран «Редагування слова» (dictionary_edit_word_fragment.xml): Схожий на екран додавання, але містить два поля для введення — для оригінального слова та його перекладу, що дозволяє користувачеві редагувати обидві частини перекладу.

Екран «Режими навчання» (learning_modes_fragment.xml): Виконує роль меню. Він містить RecyclerView (learningModesRecyclerView) всередині CardView, де відображається список доступних режимів навчання для вибору користувачем.

Екрани режимів навчання:

1. Флеш-картки (learning_flashcard_fragment.xml): Центральним елементом є CardView, що імітує картку, яка "перегортається" при натисканні, показуючи то слово, то його переклад (див. Рис. 9).
2. Введення перекладу (learning_type_translation_fragment.xml): Показує слово для перекладу та TextInputLayout з полем для введення відповіді користувачем (див. Рис. 9).
3. Множинний вибір (learning_multiple_choice_fragment.xml): Містить текстове поле для слова-запитання (tvQuizWordPrompt) та чотири кнопки з варіантами відповідей (див. Рис. 9).

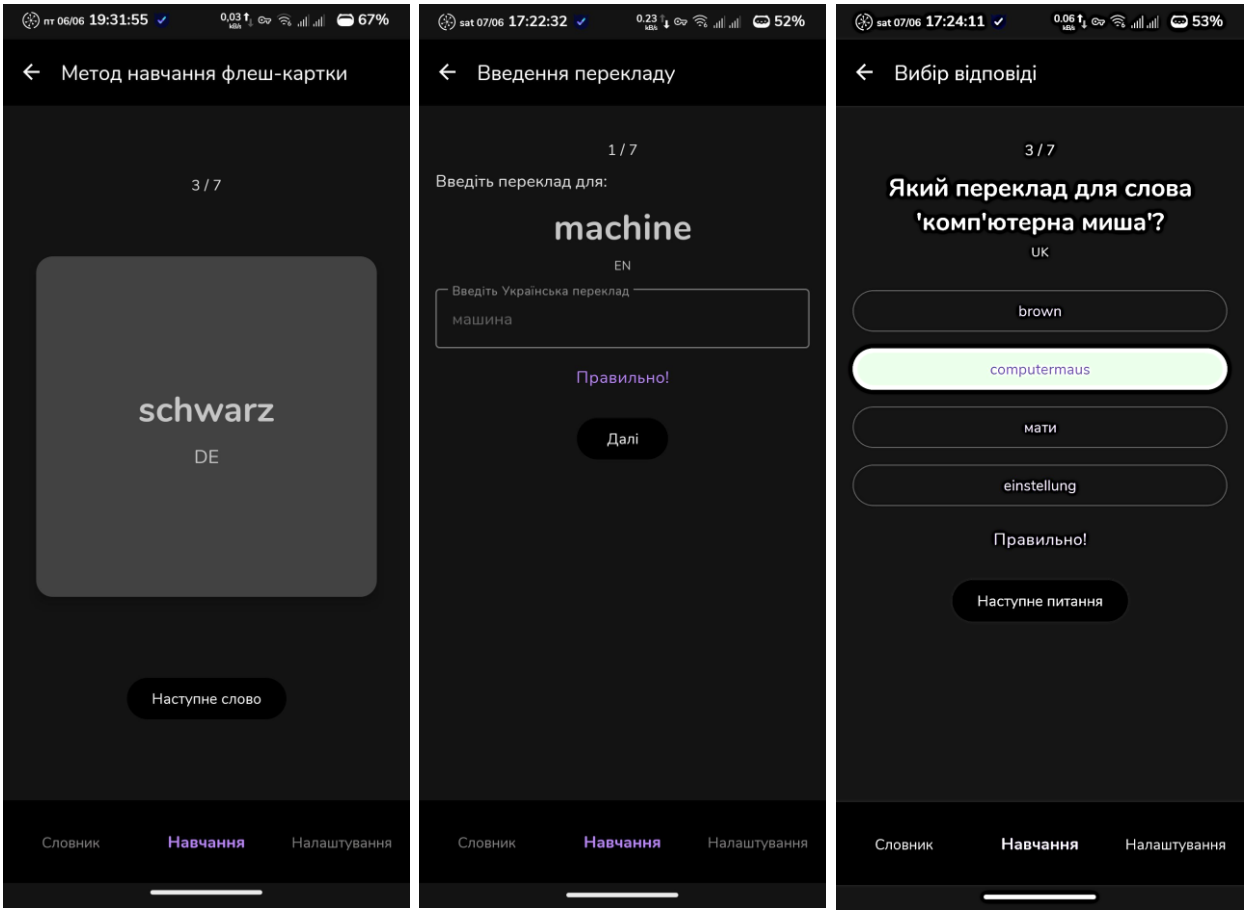


Рисунок 9 – Екрани "Флеш-картки", "Введення перекладу" та "Множинний вибір"

Екран «Налаштування» (settings_fragment.xml): Надає користувачеві можливість керувати налаштуваннями застосунку. Головним елементом є SwitchMaterial для ввімкнення/вимкнення сповіщень. При його активації з'являється блок з детальними налаштуваннями п'яти інтервалів повторення, кожен з яких реалізований за допомогою включеної (include) розмітки settings_partial_interval.xml (див. Рис. 10).

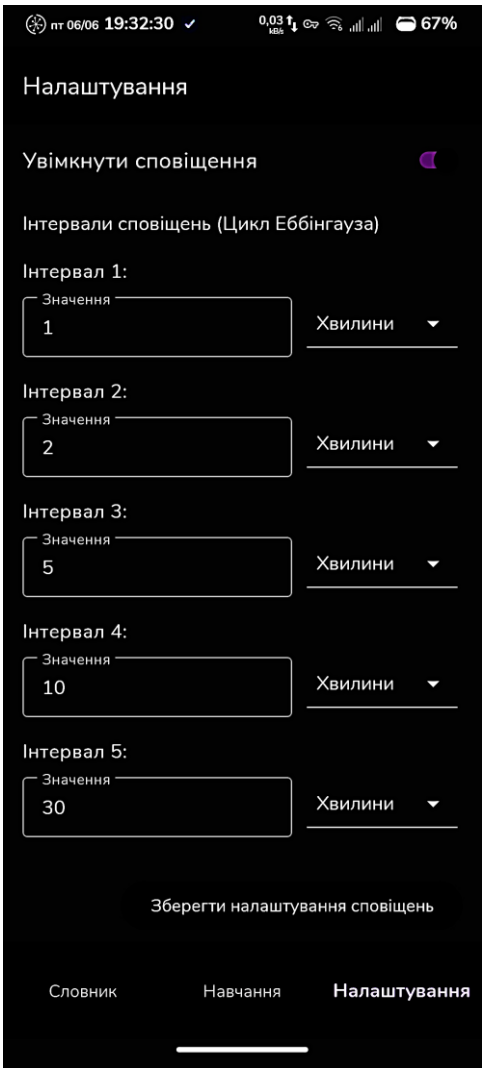


Рисунок 10 – Екран "Налаштування"

3.3 Реалізація і тестування мобільного застосунку

3.3.1 Реалізація

Об’єм коду застосунка:

1. Kotlin — 143 кілобайт, ~ 3800 рядків.
2. XML — 54 кілобайт, ~ 1200 рядків.

Об’єм вихідного застосунка:

1. Додаток в форматі “.apk” — 13500 кілобайт.

Тестування додатку було проведено на смартфоні Poco F5. На Таблиці 2 зображено стан швидкодії додатку.

Таблиця 2 – Швидкодія додатку

Стан додатку	**CPU, %**	**RAM, MB**
У фоні	~ 0-5	~ 4
Активний	~ 1-10	~ 55

3.3.2 Апробація

Апробація розробленого мобільного застосунку проводилася з метою перевірки його відповідності функціональним та нефункціональним вимогам, визначеним у розділі 2. Тестування було спрямоване на виявлення та усунення можливих дефектів, а також на перевірку стабільності роботи та зручності користувацького інтерфейсу.

Основним методом тестування було ручне функціональне тестування, яке базувалося на розроблених тест-кейсах. Тест-кейси охоплювали всі основні сценарії використання застосунку, від управління словником до режимів навчання та системи сповіщень:

1. Додавання нового слова: Слово автоматично перекладається через DeepL API, зберігається в базі даних та з'являється у списку.

Відображається повідомлення про успішне додавання.

2. Спроба додати дублікат: Слово не додається. Користувач отримує повідомлення про помилку.
3. Редагування слова: Зміни зберігаються в базі даних. Список слів оновлюється. Користувач повертається на екран словника.
4. Видалення слів: Вибрані слова видаляються з бази даних та зникають зі списку. Режим вибору деактивується.
5. Фільтрація словника: У списку відображаються лише слова, що відповідають парі EN → UK (або UK → EN, коли вони збережені у зворотному порядку).
6. Режим навчання «Флеш-картки»: Картка перегортається, показуючи переклад. Кнопка «Наступне» завантажує нове слово. Лічильник слів оновлюється.
7. Режим навчання «Введення перекладу»: При правильній відповіді з’являється позитивний відгук. При неправильній — негативний, з показом правильної відповіді.
8. Система сповіщень: Через 1 хвилину система надсилає локальне сповіщення з нагадуванням про навчання.

3.4 Висновки до розділу 3

1. Шар даних: Створено механізм управління даними на основі WordRepository, який інкапсулює роботу з локальною базою даних Room для зберігання словника користувача та інтеграцію з DeepL API за допомогою Retrofit.
2. Управління словником: Реалізовано CRUD-операції (додавання, перегляд, редагування, видалення) для слів, включаючи логіку фільтрації та пошуку, що забезпечує користувачеві гнучкі інструменти для роботи зі своїм словником.
3. Режими навчання: Розроблено три інтерактивні режими навчання

(«Флеш-картки», «Введення перекладу», «Множинний вибір»), які сприяють активному залученню користувача та підвищенню ефективності запам'ятовування.

4. Система сповіщень: Впроваджено систему інтервальних нагадувань за допомогою WorkManager, яка дозволяє планувати локальні сповіщення для стимулювання регулярного повторення матеріалу.
5. Користувацький інтерфейс: Створено інтуїтивно зрозумілий та візуально привабливий інтерфейс на основі XML-розмітки та принципів Material Design. Реалізовано через Navigation Component.

ВИСНОВКИ

1. У ході виконання кваліфікаційної роботи було досліджено проблему ефективного вивчення іноземних слів з використанням сучасних мобільних технологій. Аналіз предметної області підтвердив актуальність розробки інструментів, що поєднують зручність, ефективні методики запам'ятовування та інтеграцію з якісними сервісами перекладу.
2. Проведено аналіз існуючих мобільних застосунків, таких як Duolingo, Memrise та Quizlet. Виявлено, що попри велику кількість рішень, існує потреба в продукті, який би надавав користувачам більшу гнучкість в управлінні власними словниками та використовував передові API для отримання точних перекладів. Також було досліджено методику інтервального повторення, яка лягла в основу системи нагадувань.
3. На основі аналізу вимог було спроєктовано архітектуру мобільного застосунку. Обрано архітектурний патерн MVVM (Model-View-ViewModel). Розроблено структуру локальної бази даних за допомогою Room Persistence Library, спроєктовано систему локальних сповіщень на базі WorkManager та визначено основні модулі системи та принципи побудови користувацького інтерфейсу.
4. В результаті виконання кваліфікаційної роботи було створено повноцінний мобільний застосунок для платформи Android, який дозволяє користувачам ефективно вивчати іноземні слова. Застосунок реалізовано мовою програмування Kotlin з використанням Android Studio та ключових компонентів Android Jetpack (ViewModel, LiveData, Room, Navigation Component, WorkManager). Реалізовано інтеграцію з DeepL API для автоматичного перекладу, три інтерактивні режими навчання та гнучку систему інтервальних нагадувань.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Duolingo Blog. URL: 1 (date of access: 29.05.2025).
2. Memrise Blog. URL: https://www.memrise.com/blog (date of access: 29.05.2025).
3. Quizlet Blog. URL: https://quizlet.com/blog (date of access: 29.05.2025).
4. Kang S. H. K. Spaced Repetition Promotes Efficient and Effective Learning. *Policy Insights from the Behavioral and Brain Sciences*. 2016. Vol. 3, no. 1. P. 12–19. URL: https://doi.org/10.1177/2372732215624708 (date of access: 22.03.2025).
5. Murre J. M. J., Dros J. Replication and Analysis of Ebbinghaus' Forgetting Curve. *PLOS ONE*. 2015. Vol. 10, no. 7. P. e0120644. URL: https://doi.org/10.1371/journal.pone.0120644 (date of access: 22.03.2025).
6. DeepL API Documentation.URL: https://developers.deepl.com/docs (date of access: 22.03.2025).
7. Write your app. *Android Developers*. URL: https://developer.android.com/studio/write (date of access: 22.03.2025).
8. Material Design. URL: https://m3.material.io (date of access: 30.05.2025).
9. Kang J. Benefits and Challenges of Mobile-Learning Brought to Student Learning Outcomes in Higher Education: A Systematic Review from 2014-2023. International Journal of Academic Research in Progressive Education and Development. 2024. Vol. 13, no. 1. URL: https://doi.org/10.6007/ijarped/v13-i1/20698 (date of access: 30.05.2025).
10. Essafi M., Belfakir L., Moubtassime M. Investigating Mobile-Assisted Language Learning Apps: Babbel, Memrise, and Duolingo as a Case Study. Journal of Curriculum and Teaching. 2024. Vol. 13, no. 2. P. 197. URL: https://doi.org/10.5430/jct.v13n2p197 (date of access: 30.05.2025).
11. Kotlin and Android | Android Developers. URL: https://developer.android.com/kotlin (date of access: 30.05.2025).

12.Room | Jetpack | Android Developers. URL: https://developer.android.com/jetpack/androidx/releases/room (date of access: 30.05.2025).

13. ViewModel overview | App architecture | Android Developers. URL: https://developer.android.com/topic/libraries/architecture/viewmodel (date of access: 30.05.2025).

14. App Architecture: Data Layer - Schedule Task with WorkManager - Android Developers | App architecture. URL: https://developer.android.com/topic/libraries/architecture/workmanager (date of access: 30.05.2025).

15. Опрятов Д.П., магістрант, Полякова Н.П., доцент — науковий керівник. Мобільний застосунок для вивчення іноземних слів. *Молода наука 2025 :* зб. науков. праць студентів, аспірантів, докторантів і молодих вчених. Запоріжжя : ЗНУ, 2025. Т.5. 404 – 406 с.