Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

# КВАЛІФІКАЦІЙНА РОБОТА

«Інформаційна система для вивчення іноземних мов з використанням зовнішніх API-сервісів»
Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.2201111.01.26.ПЗ

Виконав студент IV курсу, група ІПЗ-22-1 _____ Тимофій СУМКА
Підпис Ім'я, ПРІЗВИЩЕ

Керівник канд. техн. наук, доцент _____ Оксана ЯШИНА
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. техн. наук, доцент _____ Юрій ФОРКУН
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

**До захисту допускаю:**
Завідувач кафедри інженерії
програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

1 червня 2026 р.

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
02 01 2026 р.

# ЗАВДАННЯ
# НА КВАЛІФІКАЦІЙНУ РОБОТУ

Сумці Тимофію Сергійовичу
Прізвище, ім'я, по батькові студента

1. Тема роботи Інформаційна система для вивчення іноземних мов з використанням зовнішніх API-сервісів

Керівник роботи Яшина Оксана Миколаївна, канд. техн. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 № 7
2. Строк подання студентом роботи на кафедру 01.06.2026 р.
3. Вихідні дані до роботи Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задачі, проєктування програмного забезпечення, програмна реалізація та тестування програмного забезпечення
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Презентаційні матеріали (слайди, 22 шт), діаграма варіантів використання системи (Use Case), діаграма міжмодульних залежностей за принципами Clean Architecture, діаграма станів процесу запиту інформації про слово, діаграма алгоритму розрахунку інтервалів навчання (SRS), діаграма сутностей системи (ERD), діаграма послідовності процесу тестування, таблиця аналізу конкурентів.

6. Консультанти розділів кваліфікаційної роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Нормоконтроль | Форкун Ю. В., канд. техн. наук, доцент | 05.05.26 | 25.05.26 |
| Антиплагіат | Форкун Ю. В., канд. техн. наук, доцент | 05.05.26 | 25.05.26 |

7. Дата видачі завдання « 02 » січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

| Назва етапів (розділів) кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|
| 1 Ознайомлення з тематикою дипломного проєктування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР) | 01.12- 31.12.2025 | |
| 2 Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання | 01.01 - 20.02.2026 | |
| 3 Проєктування програмного забезпечення | 21.02 - 20.03.2026 | |
| 4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ | 21.03 - 30.04.2026 | |
| 5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог | 01.05 - 25.05.2026 | |
| 6 Попередній захист КвР | травень 2026 | Згідно графіка |
| 7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки. | 26.05 - 30.06.2026 | |
| 8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР | з 01.06.2026 | |

Студент _____ Тимофій СУМКА
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Оксана ЯШИНА
Підпис Ім'я, ПРІЗВИЩЕ

# АНОТАЦІЯ

Тема кваліфікаційної роботи: «Інформаційна система для вивчення іноземних мов з використанням зовнішніх API-сервісів»

Автор роботи: Сумка Тимофій Сергійович

Керівник роботи: Яшина Оксана Миколаївна

Пояснювальна записка: 80 с., 24 рис., 14 табл., 3 дод., 40 джерел.

Графічна частина: 3 креслення

ANDROID, FIREBASE, KOTLIN, SRS-АЛГОРИТМ, МОБІЛЬНИЙ ЗАСТОСУНОК.

Метою кваліфікаційної роботи є створення інформаційної системи для інтерактивного вивчення іноземних мов із використанням зовнішніх сервісів, що забезпечує персоналізований та інтерактивний навчальний процес, збереження та обробку навчальних даних у хмарному середовищі, а також інтеграцію з зовнішніми API.

У кваліфікаційній роботі проведено аналіз предметної області та існуючих рішень у сфері цифрових платформ для вивчення іноземних мов, визначено функціональні та нефункціональні вимоги до інформаційної системи, розроблено загальну архітектуру програмного забезпечення, структуру клієнтської та серверної частин, а також модель зберігання даних.

Для реалізації клієнтської частини застосунку використано мову програмування Kotlin та платформу Android, для збереження та синхронізації даних - хмарний сервіс Firebase, а для реалізації серверної логіки та взаємодії з зовнішніми сервісами - онлайн-сервер на базі фреймворку Ktor.

У результаті проєктування здійснена програмна реалізація інформаційної системи для інтерактивного вивчення іноземних мов із використанням зовнішніх сервісів, що забезпечує персоналізований та інтерактивний навчальний процес.

27.05.26
Дата

Підпис

# ВІДОМІСТЬ ДОКУМЕНТІВ

| № рядка | Формат | Позначення документа | Найменування документа | К-сть аркушів | № екз. | Примітка |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | Текстові документи | | | |
| | | | | | | |
| 1 | А4 | КвРІПЗ. 2201111.01.26.ПЗ | Пояснювальна записка | 80 | | |
| 2 | А4 | | Завдання на кваліфіка- | 1 | | |
| | | | ційну роботу | | | |
| 3 | А4 | | Анотація | 1 | | |
| | | | | | | |
| | | | Графічні документи | | | |
| | | | | | | |
| 4 | А3 | КвРІПЗ. 2201111.01.26.Е8 | Діаграма варіантів | 1 | | |
| | | | використання | | | |
| 5 | А3 | КвРІПЗ. 2201111.01.26.Е8 | Діаграма зв'язків | 1 | | |
| | | | модулів | | | |
| 6 | А3 | КвРІПЗ. 2201111.01.26.Е8 | Діаграма класів | 1 | | |
| 7 | А4 | КвРІПЗ. 2201111.01.26.Е8 | Презентаційні матеріали | 22 | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

| Змн. | Арк. | № докум. | Підпис | Дата | КвРІПЗ.2201111.01.26.ВД | | |
|---|---|---|---|---|---|---|---|
| Виконав | | Сумка Т. С. | | 27.05.26 | Інформаційна система для вивчення іноземних мов з використанням зовнішніх API-сервісів | Літ. | Арк. 1 / Аркушів 1 |
| Керівник | | Яшина О. М. | | 27.05.26 | | | |
| Н. контр. | | Форкун Ю. В. | | 27.05.26 | Відомість документів | ХНУ, ІПЗ-22-1 | |
| Зав. каф. | | Бедратюк Л. П. | | 01.06.26 | | | |

# ЗМІСТ

Перелік скорочень ..... 7
Вступ ..... 8
1 Дослідження предметної області та постановка задачі ..... 10
1.1 Змістовий аналіз предметної області, її структурних та функціональних особливостей ..... 10
1.2 Аналіз наявного програмно-технічного забезпечення предметної області ..... 16
1.3 Визначення функціональних та нефункціональних вимог до програмного забезпечення ..... 23
1.4 Висновки до першого розділу. Постановка задачі ..... 27
2 Проєктування програмного забезпечення ..... 29
2.1 Вибір типу архітектури та шаблонів проєктування ..... 29
2.2 Опис декомпозиції ..... 33
2.2.1 Загальна декомпозиція ..... 33
2.2.2 Модульна декомпозиція ..... 34
2.2.3 Декомпозиція моделі переходів станів ..... 36
2.3 Опис залежностей ..... 38
2.3.1 Міжмодульні залежності ..... 39
2.3.2 Залежності всередині даних ..... 39
2.3.3 Залежності між станами ..... 40
2.4 Опис інтерфейсів ..... 41
2.5 Детальне проєктування модулів ..... 43
2.6 Детальне проєктування даних ..... 47
2.7 Аналіз та вибір технологій і методів реалізації застосунку ..... 50
2.8 Висновки до другого розділу ..... 53

3 Програмна реалізація та тестування програмного забезпечення ..... 55
3.1 Програмна реалізація модулів ..... 55
3.2 Розроблення бази даних ..... 59
3.3 Вимоги до технічних та програмних засобів ..... 62
3.4 Тестування програмного забезпечення ..... 63
3.4.1 Вибір та обґрунтування методів тестування застосунку ..... 63
3.4.2 Валідація та верифікація програмного забезпечення ..... 65
3.4.3 Аналіз результатів тестування ..... 69
3.5 Висновки до третього розділу ..... 72
Висновки ..... 74
Перелік джерел посилання ..... 77
Додаток А Графічні матеріали ..... 81
Додаток Б Програмний код основних модулів ..... 84
Додаток В Презентаційні матеріали ..... 93

# ПЕРЕЛІК СКОРОЧЕНЬ

| | | |
|---|---|---|
| API | - | Application Programming Interface |
| CRUD | - | Create, Read, Update, Delete |
| DAO | - | Data Access Object |
| EF | - | Ease Factor |
| ERD | - | Entity-Relationship Diagram |
| FSRS | - | Free Spaced Repetition Scheduler |
| HTML | - | HyperText Markup Language |
| TTS | - | Text To Speech |
| SRS | - | Spaced Repetition System |
| БД | - | База даних |
| ОС | - | Операційна система |
| ПЗ | - | Програмне забезпечення |
| СКБД | - | Система керування базами даних |

# ВСТУП

Популярність мобільних платформ як основного засобу взаємодії з цифровим контентом стабільно зростає з кожним роком, що призводить до активного поширення мобільних застосунків у різних сферах діяльності. Завдяки своїй доступності, мобільності та зручності повсякденного використання, мобільні застосунки успішно конкурують з іншими платформами.

Мобільні застосунки, зокрема для навчання та самоосвіти, займають провідні позиції серед найпопулярніших категорій застосунків як на платформах iOS, так і Android. Особливу популярність, мають застосунки для вивчення іноземних мов, через останні тенденції до вивчення додаткових мов, та цифровізації що полегшує процес, поєднуючи зручність використання, інтерактивність та можливість самостійного навчання у будь-який час.

На сьогоднішній день існує безліч різних готових програмних рішень для цього, які реалізують як базові функції на кшталт вправ для засвоєння лексики та граматики, електронних словників, карток для запам'ятовування та тестів, так і повноцінні готові освітні програми для повного вивчення мови від базового рівня з використанням вправ та ігрових механік. Незважаючи на таке широке різноманіття, значна частина таких рішень має обмежені можливості персоналізації та гнучкості, залишаючи користувача взаємодіяти тільки з заздалегідь підготовленим контентом.

Особливою проблемою є ефективне вивчення та запам’ятовування нової лексики, що вимагає систематичного повторення та обширної бази знань, здатної задовольнити будь-які потреби. У цьому випадку, інтеграція інтервального повторення та зовнішніх сервісів (онлайн-словників, перекладачів, сервісів прикладів уживання та озвучення), в межах одного застосунку, дозволяє вирішити ці проблеми, та підвищити якість подання навчального матеріалу, підлаштовуючись під кожного користувача індивідуально.

Актуальність теми кваліфікаційної роботи зумовлена високою

популярністю мобільних застосунків для вивчення іноземних мов, об’єктивною потребою користувачів у зручних та інтерактивних інструментах навчання, а також необхідністю створення програмного забезпечення, що поєднує традиційні методи вивчення мови з можливостями сучасних зовнішніх сервісів. Розроблення такої інформаційної системи є доцільним з огляду на практичну значущість результатів та відповідність сучасним тенденціям ринку мобільних застосунків.

У зв’язку з цим, актуальним є створення інформаційної системи у вигляді мобільного застосунку з хмарною архітектурою. Реалізації серверної частини на базі Ktor, дозволяє залучити зовнішні сервіси та готові бази знань, для забезпечення додаткової гнучкості процесу та функціонального наповнення.

Метою кваліфікаційної роботи є створення інформаційної системи для інтерактивного вивчення іноземних мов із використанням зовнішніх сервісів, що забезпечує персоналізований та інтерактивний навчальний процес, збереження та обробку навчальних даних у хмарному середовищі, а також інтеграцію з зовнішніми API.

Для досягнення поставленої мети та успішної реалізації інформаційної системи необхідно розв'язати такі завдання проєктування:

– виконати аналіз предметної області мобільних застосунків для вивчення іноземних мов;

– проаналізувати існуючі програмні рішення та визначити їх недоліки;

– встановити функціональні та нефункціональні вимоги до системи;

– виконати детальне проєктування архітектури;

– обґрунтувати вибір технологій;

– виконати програмну реалізацію;

– реалізувати інтеграцію із зовнішніми сервісами через API;

– провести тестування розробленої системи;

– проаналізувати отримані результати за вимогами.

# 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

## 1.1 Змістовий аналіз предметної області, її структурних та функціональних особливостей

Популярність мобільних платформ стабільно зростає, стаючи головним інструментом взаємодії з інформацією у повсякденному житті. За останнє десятиліття, спостерігається чітка зміна - частка ринку мобільних пристроїв у світі перейняла лідерську позицію над комп'ютерними пристроями, та демонструє подальше зростання (рисунок 1.1) [1].

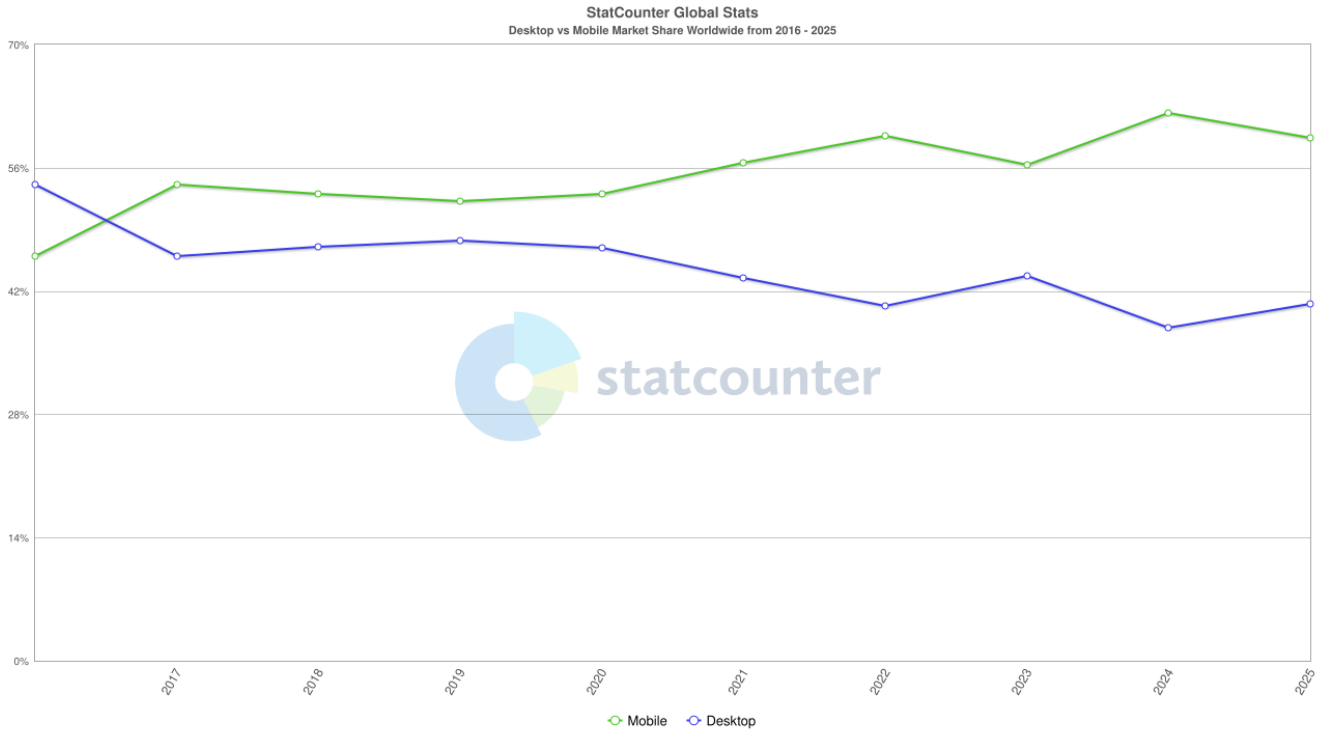



Рисунок 1.1 - Частка ринку настільних комп'ютерів та мобільних пристроїв у світі

Такий тренд мобільних пристроїв зумовлений їхньою портативністю та постійним розширенням можливостей. Сучасні смартфони поєднують в собі новітні процесори та технології, що дозволяє реалізовувати складні програми, які раніше були доступні лише на ПК. Це призвело до того, що мобільна платформа

стала не просто засобом зв'язку, а ключовим середовищем для навчання, роботи та саморозвитку, формуючи стратегію «Mobile-First» для більшості сучасних продуктів. Це поняття ввів Люк Вроблевські у своїй книзі «Mobile First», воно означає, що розробку будь-якого сучасного продукту варто починати саме з мобільної версії [2]. Він виділяє три причини:

– колосальне зростання кількості мобільних пристроїв;

– обмеженість ресурсів (що змушує фокусуватися на головному);

– нові можливості (геолокація, тач-інтерфейс).

Головними конкурентами в цій області є операційні системи Android та iOS. Вони розповсюджують застосунки через свої офіційні маркетплейси Google Play та App Store відповідно. На сьогодні, платформа Android є більш розповсюдженою, та домінує серед звичайних користувачів завдяки відкритості системи та широкому спектру пристроїв. Аналізуючи динаміку останніх десяти років, можна встановити, що частка завантажень у Google Play, стабільно перевищує показники App Store майже у 4 рази (рисунок 1.2) [3].

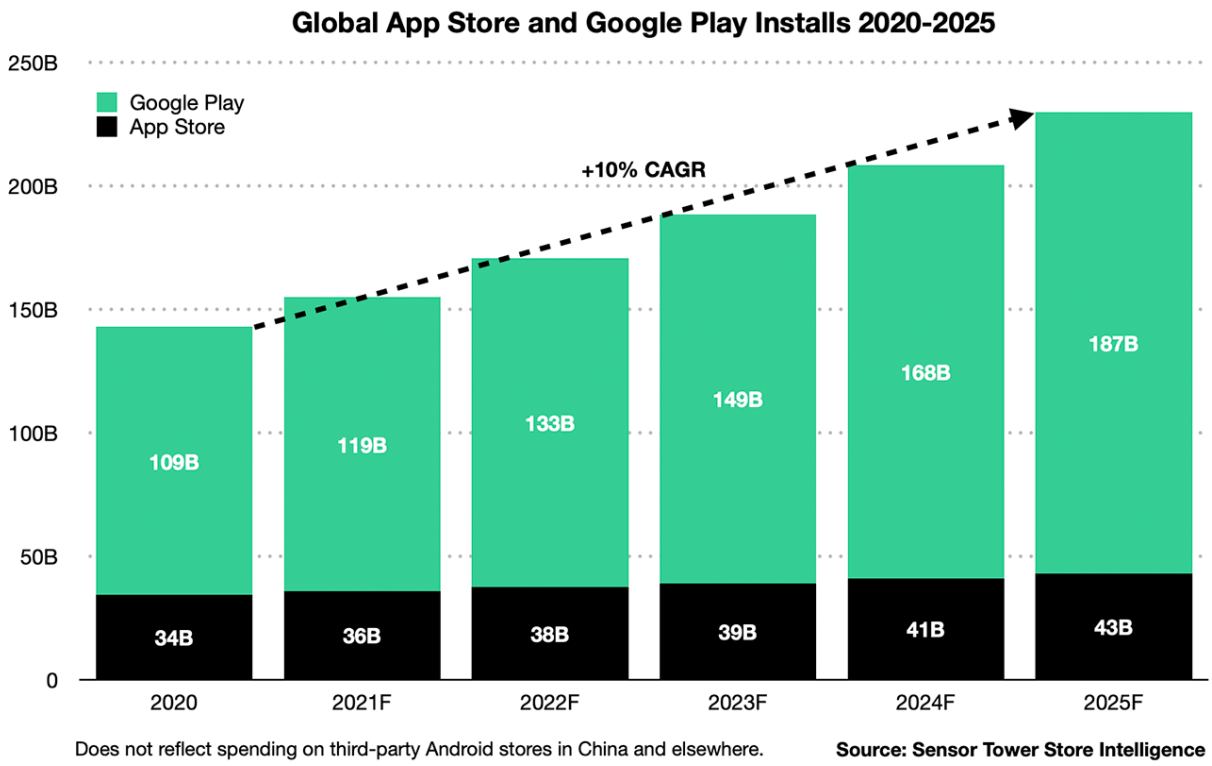



Рисунок 1.2 - Глобальна статистика завантажень мобільних застосунків

Серед категорій маркетплейсу, у Топ-3 за кількістю застосунків у Google Play, знаходяться «Education», «Business» та «Tools» (рисунок 1.3), з повним домінуванням безкоштовного поширення над умовно-безкоштовним [4]. З чого виходить, що для успіху освітнього застосунку, необхідно пропонувати користувачам достатній базовий функціонал безкоштовно, залучаючи унікальними та захопливими механіками утримання.

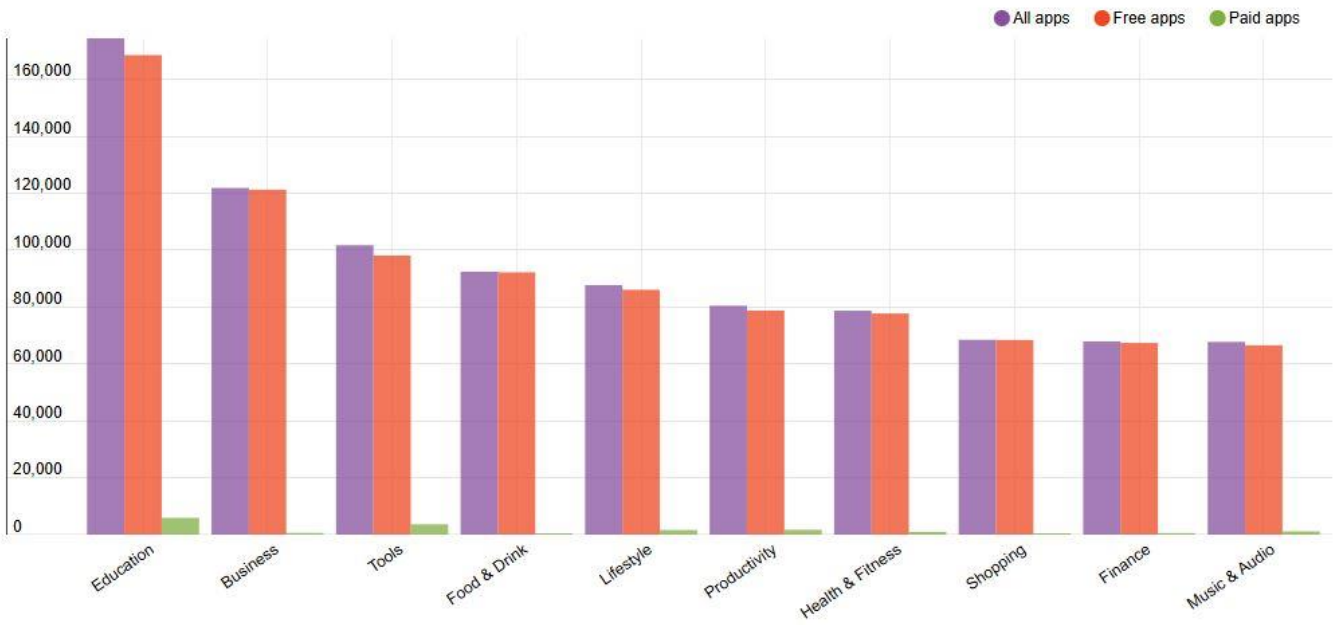



Рисунок 1.3 - Найпопулярніші категорії Google Play (лютий 2026)

Як зазначено раніше, найбільш поширеною категорією на платформі є «Education». Вона охоплює категорії від цифровізації класичного освітнього процесу в закладах освіти, до спеціалізованих курсів або інструментів самоосвіти. Висока популярність цієї категорії, зумовлена такими факторами:

– тенденція нових поколінь витрачати більше часу в мобільних пристроях;
– використання вільного часу з користю;
– цілодобова доступність;
– широкий спектр інтерактивності для учнів та дітей;
– відсутність страху зробити помилку;
– портативність.

Аналітика за 2024 рік свідчить, що найбільш прибутковими та популярними підсекторами в межах цієї категорії, є платформи для онлайн-курсів, системи безпосереднього навчання, та застосунки для вивчення іноземних мов (рисунок 1.4) [5]. Така популярність підтверджує актуальність самоосвіти, та потребу в додаткових інструментах, для полегшення цього процесу. Одним з таких процесів є вивчення іноземних мов, що користується великим попитом у широко спектра користувачів - як для школярів та студентів, які вчать другу мову у закладі освіти, так і для інтеграції у новій країні або для професійних потреб.

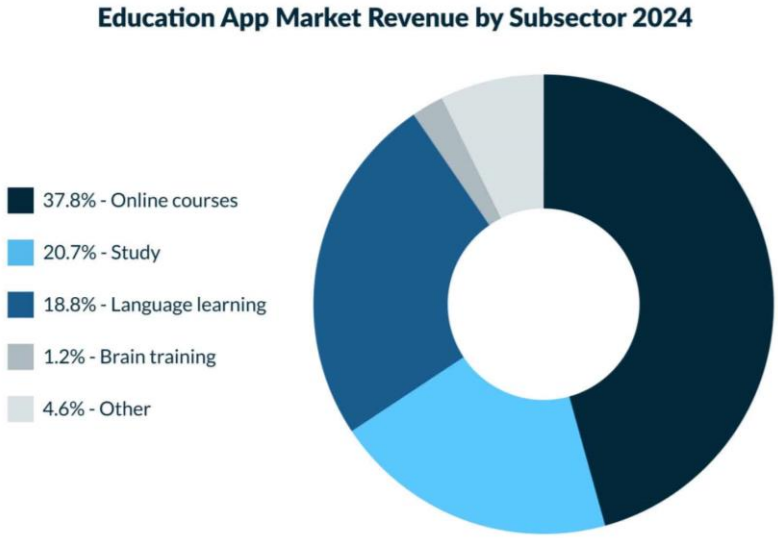



Рисунок 1.4 - Дохід від освітніх додатків за підсекторами

Особливо актуальним це питання є для України, в контексті євроінтеграційних процесів, що потребує від фахівців різних галузей високого рівня володіння англійською та іншими європейськими мовами.

Згідно з профільним дослідженням причин низької ефективності навчання [6], головними бар'єрами у цьому процесі є:

– відсутність системи повторення неефективність механічного завчання;
– відсутність, або швидка втрата мотивації;
– дефіцит уваги;
– нездатність вийти із зони комфорту та страх помилки;
– навички вчителя/матеріалу.

Більшість наявних застосунків зосереджуються на тому, щоб надати структурований матеріал, або набір інструментів для роботи, проте не забезпечують адаптивності процесу навчання під індивідуальні особливості. Для вирішення проблеми індивідуальності процесу запам’ятовування, практикується методика Spaced Repetition System (SRS).

Інтервальне повторення це техніка утримання в пам’яті, яка полягає в повторенні вивченого навчального матеріалу за певними, постійно зростаючими інтервалами. Даний метод, як спосіб швидкого і ефективного заучування будь-якої інформації, відомий у експериментальній психології [7].

Ця техніка базується на дослідженнях кривої забування (рисунок 1.5), фундаментальної закономірності яка описує процес експоненціального зниження обсягу збереженої інформації у пам’яті за відсутності активного повторення [8].

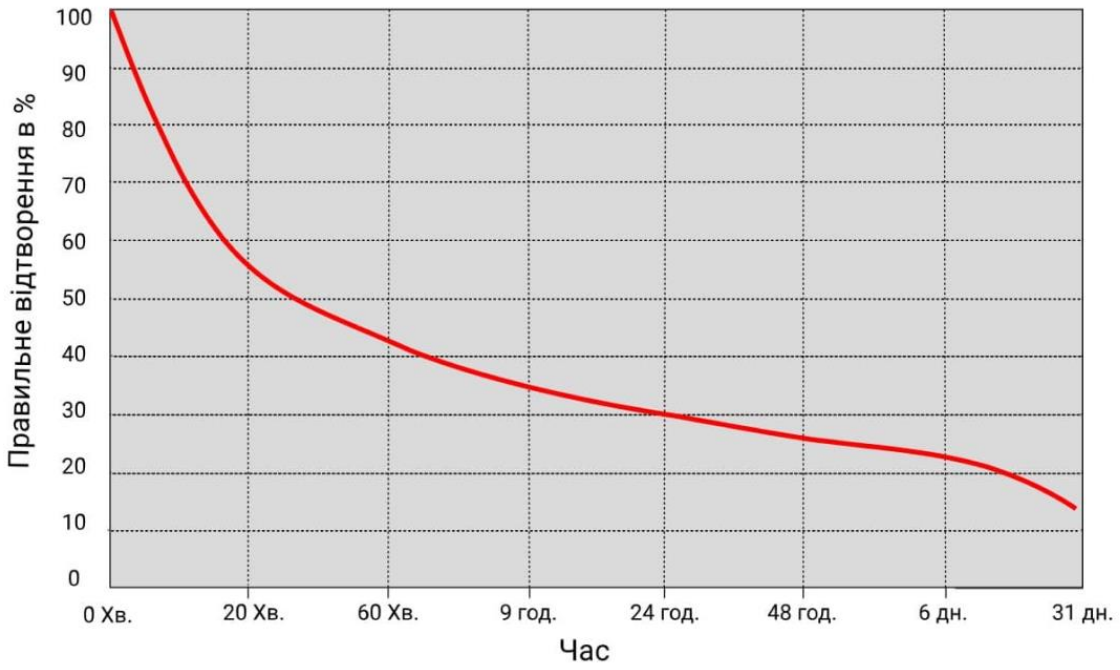



Рисунок 1.5 - Крива забування Еббінгауза

Для оптимізації запам’ятовування, сутність алгоритму полягає в оптимізації інтервалів між повторенням знань: система пропонує перевірку знань саме в той момент, коли ймовірність забування слова є максимальною, підлаштовуючись під користувача, та поступово збільшуючи інтервали. Це

дозволяє ефективно переводити інформацію з короткочасної у довготривалу пам'ять, не перевантажуючи людину інтенсивним зазубрюванням (рисунок 1.6).

Впровадження SRS-алгоритму дозволяє автоматизувати процес відслідковування і вирахування інтервалів, та опрацювання персональних даних про успішність користувача, використовуючи індивідуальний список слів, щоб використовувати його для навчання. Це вирішує ключову проблему з запам’ятовуванням та індивідуальним підходом до навчання.

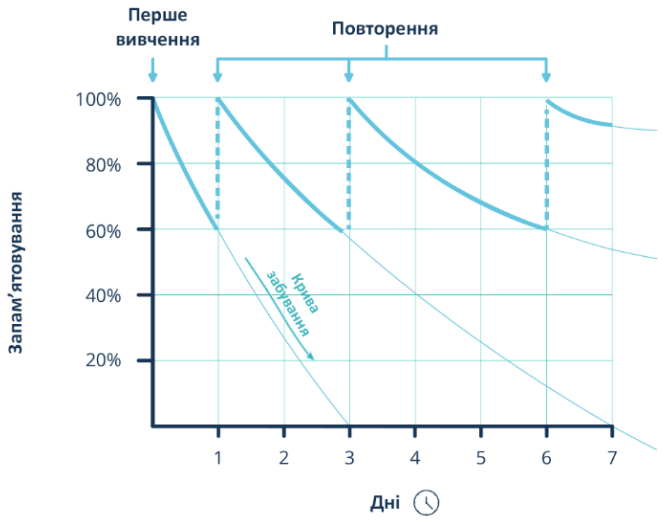



Рисунок 1.6 - Модельна схема алгоритму інтервальних повторень

Окрім вирішення проблеми з вивченням, критичним залишається втрата мотивації та розсіювання уваги, що може призвести до припинення навчального процесу. Емпіричні дослідження доводять, що гейміфікація, є одним із найефективніших інструментів для утримання уваги користувачів та підтримки довгострокової залученості [9]. За аналогією з ігровими застосунками, вона передбачає впровадження модулів взаємодії та тригери, які забезпечують зворотний зв'язок на дії користувача, та заохочують його через систему винагород, рівнів та ігрових досягнень.

Однак, для ефективної реалізації таких механік, необхідно розуміти образ типового користувача, для визначення його звичок та поведінки. Спираючись на дослідження, щодо використання персоналізованих інструментів у навчанні [10],

було встановлено, що цим типовим образом є це молода людина (зазвичай студент або фахівець), із рівнем володіння мовою A1 або B1, обмеженим вільним часом, ключовими характеристиками якої є:

– потреба у миттєвій валідації дій, що передбачає негайне підтвердження правильності виконання завдань;

– схильність до мікронавчання (Micro-learning), яка полягає у тому, що користувачі віддають перевагу коротким (від 2 до 5 хвилин) навчальним сесіям, що легко вписуються в повсякденний графік;

– важливість візуалізації прогресу, оскільки необхідно бачити кількісне та якісне відображення своїх успіхів;

– чутливість до психологічного комфорту, тому що цифровий простір дозволяє навчатися без страху соціального засудження за помилки.

Таким чином, проведений аналіз показав, що незважаючи на велику кількість готових рішень, реалізація застосунку, який би надавав користувачу повну свободу дій, підлаштовуючись під неї, та допомагаючи за рахунок гейміфікації для утримання уваги, все ще є актуальним. Це дозволить створити персоналізоване середовище навчання, адаптивне під потреби сучасної аудиторії.

## 1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Після проведення аналізу предметної області, та визначення ключових потреб користувачів, наступним етапом проєктування є дослідження вже існуючих рішень на ринку. Його метою є визначення сучасних стандартів щодо програмних рішень, їх недоліків та визначення можливостей для покращень.

Для проведення комплексного аналізу, було відібрано застосунки з Google Play, які посідають лідерські позиції в категорії «Education». Оцінка проводилася за 28 функціональними та технічними критеріями, що охоплюють базовий функціонал, ігрові механіки та варіанти роботи з даними. Результати цього

аналізу заведено в таблицю, і наведені у (Додатку А.1).

Проведений аналіз дозволив виділити ключові вимоги ринку за методикою PoP/PoD (таблиця 1.1). Точки відповідності показують функції, що є майже у всіх готових рішень ринку, та є фактичним стандартом для сучасного освітнього ПЗ. Точки унікальності ж показують особливі випадки, що взагалі не зустрічаються, або зустрічаються рідко, через що можуть виділити продукт серед інших.

Таблиця 1.1 - Результат комплексного аналізу конкурентів

| **Точки відповідності** | **Точки унікальності** |
|---|---|
| – технологія інтервальних повторень (SRS) ;<br>– підтримка флеш-карток;<br>– система сповіщень;<br>– хмарна синхронізацію прогресу. | – гнучкість алгоритмів;<br>– гейміфікація;<br>– імпортування власних даних;<br>– тестування рівня знань;<br>– персонаж-маскот;<br>– вбудований перекладач. |

Окрім загального огляду ринку мобільних освітніх застосунків, було проведено детальне дослідження та порівняльний аналіз програмних аналогів, які представляють найкраще наявне технічне рішення у конкретній проблематиці

1. Duolingo (Duolingo Inc.) [11] - абсолютний лідер за рівнем гейміфікації та утримання уваги користувача. Проєкт є стабільним лідером в категорії освіти, та демонструє ефективне поєднання освітнього процесу з ігровими механіками. Пропонує заготовлені курси зі словами, аудіо-вимовою та візуалізацією та призначений для ігрового навчання іноземним мовам з фокусом на базові рівні.

Аналізуючи основні вікна інтерфейсу Duolingo (рисунок 1.7), можна визначити такі особливості:

– використання лінійної карти прогресу;

– мінімалістичний та яскравий дизайн;

– яскраві персонажі та високий акцент на маскоті;

– фірмовий колір - зелений, використовується в деталях інтерфейсу;

– гейміфікація інтерфейсу;

– миттєвий візуальний та звуковий фідбек.


Рисунок 1.7 - Інтерфейс застосунку Duolingo

Окрім приємного інтерфейсу та візуального фідбеку, застосунок має великий функціонал, де окремої уваги заслуговують методи гейміфікація. Вони полягають у щоденних завданнях, інтерактивних вправах, реальних життєвих сценаріях, соціальних лігах, ударного режиму та досягнень, приклад яких наведено нижче (рисунок 1.8). Це призводить до довгого утримання користувачів у системі, пропонуючи одразу декілька стимулів.

Застосунок використовує власний допрацьований алгоритм реалізації SRS - Half-Life Regression (HLR), який застосовує машинне навчання та інтегрується з підсистемою гейміфікації. На відміну від класичних підходів, він не просто розраховує момент забування, а передбачає його на основі результатів проходження користувачем вправ, історії проходження минулих тестів, його персональних налаштувань та отриманих досягнень.

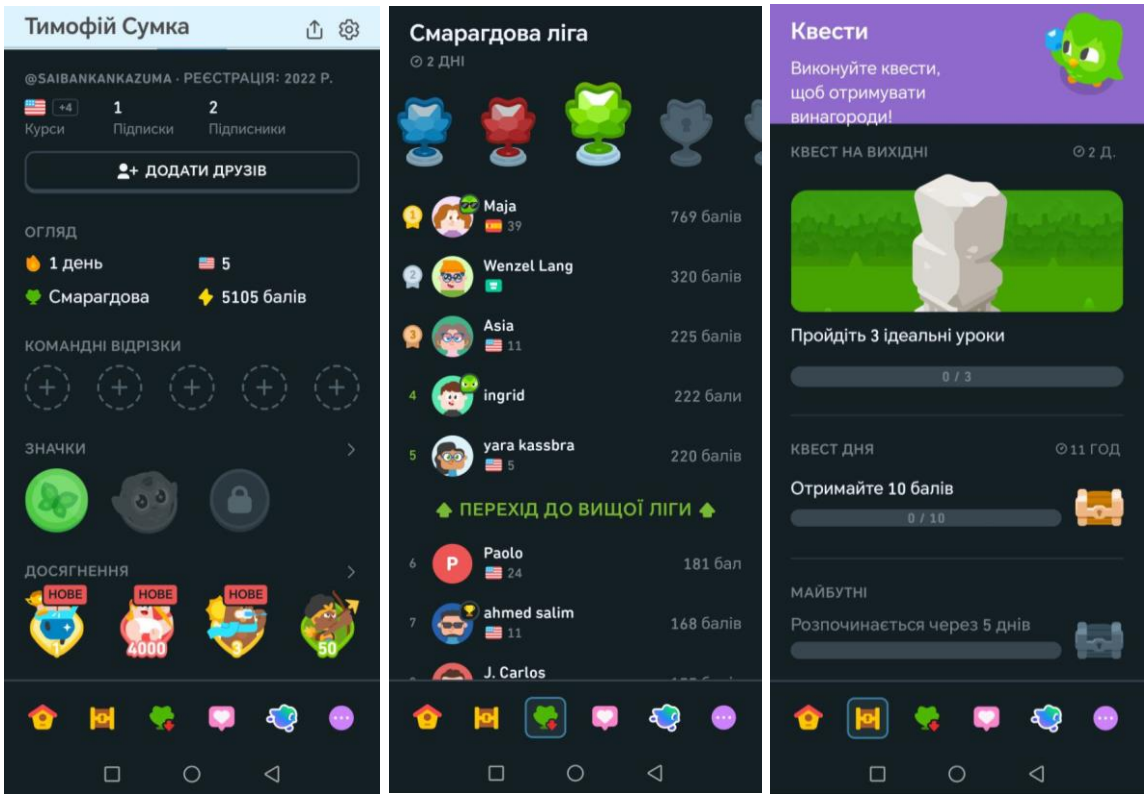



Рисунок 1.8 - Вікна гейміфікації застосунку Duolingo

У результаті проведеного аналізу інтерфейсу та функціональних можливостей застосунку було наведено у (таблиці 1.2).

Таблиця 1.2 - Переваги та недоліки Duolingo

| **Переваги** | **Недоліки** |
|---|---|
| – високий Retention завдяки пропрацьованій гейміфікації та Push-повідомленням;<br>– велике різноманіття інтерактивних завдань;<br>– соціальна інтеграція;<br>– кросплатформеність. | – повна відсутність можливості вивчати власні слова;<br>– обмеженість заготовленими курсами контенту;<br>– фокус лише на базові рівні знання. |

2.Anki (AnkiDroid Open Source Team) [12] - найрозвинутіший інструмент для запам’ятовування великих обсягів даних, що базується на відкритому коді та

особливій реалізації системи SRS. На відміну від інших конкурентів, Anki надає повний контроль над структурою даних та параметрами алгоритму SRS, та використовує набори слів (карток) створених користувачами на веб-платформі.

Аналізуючи інтерфейс (рисунок 1.9), можна визначити такі особливості:

– редактор карток на базі HTML;

– таблична структура даних;

– наявність розширень написаних на Python;

– деталізована статистика протягом всього застосунку.

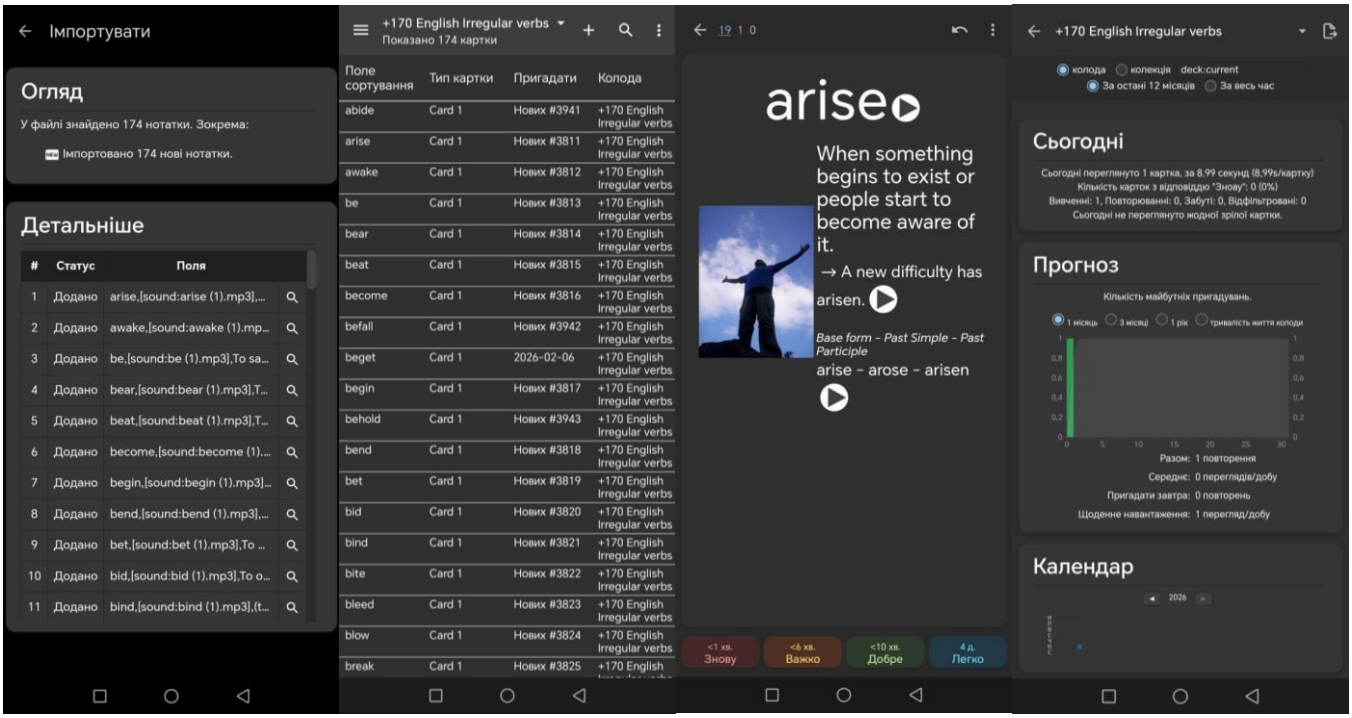



Рисунок 1.9 - Інтерфейс застосунку Anki

Головною перевагою Anki є акцент на реалізацію методики SRS. Станом на версію Anki 23.10, застосунок пропонує два доступні алгоритми повторень: перший базується на класичному алгоритмі SuperMemo 2, а другий - FSRS [13], більш сучасний та передовий варіант реалізації SRS, що забезпечує точніше планування повторень. Ще однією характерною особливістю є відкрита архітектура застосунку, яка дозволяє імпортувати та експортувати колоди карток між користувачами через веб-платформу проєкту. На основі цього, для Anki було

визначено основні переваги та недоліки, що відображені у (таблиці 1.3).

Таблиця 1.3 - Переваги та недоліки Anki

| **Переваги** | **Недоліки** |
|---|---|
| – повна автоматизація SRS та найточніша реалізація алгоритму SM-2;<br>– деталізована індивідуальна статистика;<br>– підтримка імпорту/експорту форматів CSV, TXT, JSON;<br>– повна працездатність без доступу до мережі з наступною синхронізацією через AnkiWeb;<br>– відсутність реклами та відкритий вихідний код. | – застарілий UI/UX;<br>– відсутність вбудованої підтримки мотивації;<br>– суб'єктивна оцінка запам'ятовування інформації користувачем;<br>– складність конфігурації для нових користувачів ;<br>– відсутність вбудованого контенту. |

3. Memrise (Memrise Ltd.) [14] - система, що представляє собою гібридне рішення, яке поєднує алгоритм інтервальних повторень із багатим медіа-контентом. Проєкт спеціалізується на асоціативному навчанні та зануренні в мовне середовище через візуальні образи та справжніх носіїв мови, орієнтуючись на природне засвоєння мови в ігровому форматі.

Аналізуючи основні вікна інтерфейсу Memrise (рисунок 1.10), можна визначити такі його особливості:

– використання коротких відеороликів вимови з носіями мови;

– типи запитань автоматично змінюються залежно від прогресу користувача (вибір варіанту, набір тексту, аудіювання);

– велика мовна база;

– навчальний процес візуалізується як ріст рослини, який прогресує за рахунок успішного виконання вправ та тестів.

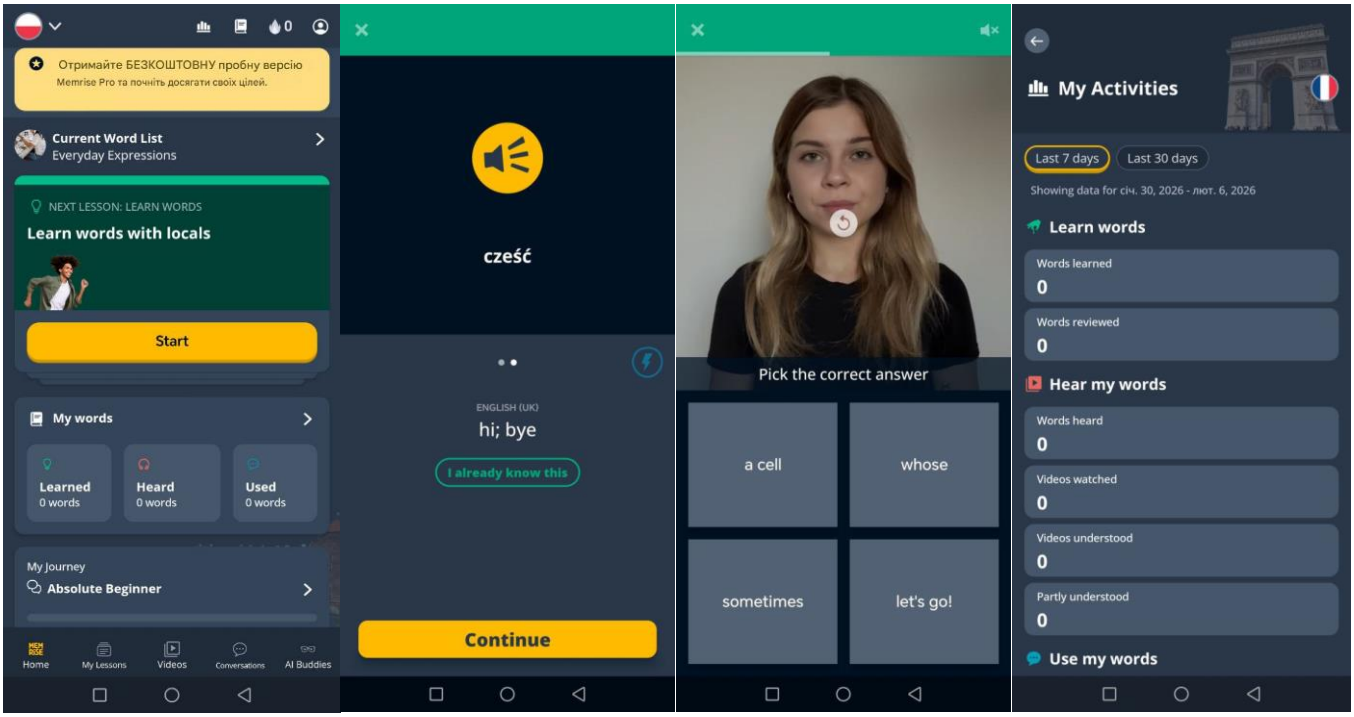



Рисунок 1.10 - Інтерфейс застосунку Memrise

Основним акцентом застосунку є використання медіаконтенту, де носії мови демонструють живу вимову. Це ефективно вирішує проблему сухого вивчення слів, додаючи необхідний візуальний та аудіо контекст. Memrise добре реалізує динамічну зміну типів завдань, адаптуючись до індивідуального темпу засвоєння матеріалу користувачем. Проте останні оновлення платформи суттєво обмежили можливості створення власних курсів.

На основі цього дослідження, для Memrise було визначено основні переваги та недоліки, що відображені у (таблиці 1.4).

Таблиця 1.4 - Переваги та недоліки Memrise

| **Переваги** | **Недоліки** |
|---|---|
| – відео з реальними носіями мови;<br>– більш серйозний підхід до лексики, ніж у Duolingo, але значно цікавіший за Anki;<br>– слова подаються в контексті; | – більшість передових функцій та режимів повторення доступні лише за передплатою;<br>– складний процес створення та використання власних курсів; |

Продовження таблиці 1.4

| | |
|---|---|
| – адаптований під швидкі сесії інтерфейс. | – алгоритм SRS менш гнучкий у налаштуванні. |

Проведений аналіз існуючих рішень показав, що ринок мобільних освітніх застосунків можна умовно розділити на дві категорії: продукти, що які успішно борються з дефіцитом уваги, але обмежують користувача заздалегідь прописаним контентом, та інструменти, що надають повну свободу у роботі з даними та алгоритмами, проте мають низьке утримання.

Таким чином, головною цілю роботи є об’єднати технічну гнучкість (імпорт власних даних, свобода дій, сучасний алгоритм FSRS) з психологічними методами утримання (гейміфікація, сповіщення, тощо). Це дозволить створити інструмент, який буде однаково ефективним як для швидких сесій мікронавчання, так і для серйозної підготовки, забезпечуючи високий рівень утримання за рахунок сучасних методів гейміфікації.

## 1.3 Визначення функціональних та нефункціональних вимог до програмного забезпечення

Провівши аналіз предметної області та дослідивши наявні програмні рішення, можна зробити висновок, що потрібне поєднання алгоритмічного підходу до навчання з високим рівнем ігрової залученості.

Функціональні вимоги до системи визначено за обов’язковими вимогами ринку, та потребами типового користувача, головний акцент якого, полягає у мікронавчанні, швидкого зворотного зв'язку та персоналізації контенту:

Робота з даними:

– можливість керувати (створювати, редагувати та видаляти) власними словниками, категоріями та словами;

– функція пошуку слів у зовнішніх джерелах;

– підтримка пошуку перекладу слова у контексті;

– підтримка імпорту слів із зовнішніх файлів форматів CSV та JSON;

– можливість поширення створених списків між користувачами;

Модуль навчання:

– реалізація алгоритму інтервальних повторень (SRS);

– вправа Flash-Cards для вивчення слів;

– автоматичне планування сесій повторення на основі індивідуального плану та налаштувань сповіщень;

– автоматична генерація тестів на основі користувацьких даних;

Гейміфікація та утримання:

– система «ударного режиму» (Streak);

– інтерактивні вправи;

– система нагород за досягнення певних етапів навчання;

– завдання «слово дня»;

Аналітика:

– візуалізація статистики прогресу;

– push-сповіщення;

За нефункціональні вимоги було взято загальні фактори, що забезпечують стабільність роботи, та еталонні критерії якості застосунку, визначені офіційною спільнотою Android Developers [15] :

– час відгуку інтерфейсу та завантаження даних не перевищує 200 мс;

– швидкість анімації має складати не менше 60 FPS (16 мс на кадр);

– запуск аудіо має відбуватися протягом 1 секунди після запиту;

– цілісність локальної бази даних (Room) та запобігання втраті даних при раптовому завершенні роботи або критичних помилках;

– відсутність помилок типу Android Not Responding та аварійних завершень під час виконання фонових операцій;

– збереження та відновлення стану інтерфейсу при перемиканні між застосунками або зміні орієнтації екрана;

– інтерфейс має бути мінімалістичним та інтуїтивно зрозумілим, побудованим на базі Material Design;

– підтримка темної та світлої тем з дотриманням контрастності тексту;

– розмір елементів керування повинен бути не менше 48 dp;

– адаптивність під різні екрани Android;

Основними акторами при взаємодії із застосунком, є сам користувач, та зовнішній сервер, що приймає та виконує запити до API (таблиця 1.5).

Таблиця 1.5 - Опис акторів системи

| **Актор** | **Короткий опис** |
|---|---|
| Користувач | Вивчає нові слова, проходить сесії повторення, налаштовує власні словники та стежить за прогресом. |
| Зовнішній API | Надає дані з зовнішніх джерел (БД, API-сервіси) про переклади, транскрипції та приклади вживання слів. |

Для візуалізації архітектурних рішень та логіки взаємодії об'єктів використано уніфіковану мову моделювання UML, що дозволяє описати процеси та структуру даних перед етапом програмної реалізації. Загальний процес використання системи користувачем представлено на діаграмі варіантів використання (Додатку А.2) [16], він складається з таких варіантів використання:

Керування даними:

– CRUD-операції над словниками, категоріями, словами;

– Імпорт групи слів через файл/код /спільноту;

– Запит на пошук слова в зовнішніх джерелах;

– Запит на переклад слова в контексті;

Навчання:

– проходження щоденних повторень;

– вивчення слів;

– генерація тестів;

– форма Flash-карток;

– виконання вправи «Слово дня» ;

Гейміфікація:

– перегляд статистики;

– виконання завдань;

– отримання нагород;

– підтримання ударного режиму;

Головним з цих сценаріїв є повторення слів за розкладом інтервалу з залученням системи SRS, оскільки він реалізує головну логіку застосунку, на який робиться акцент, та безпосередньо впливає на ефективність засвоєння лексики користувачем. Детальніший опис роботи, логіки, тригерів та передумов цього процесу представлено в (таблиці 1.6).

Таблиця 1.6 - Специфікація варіанту використання «Щоденне повторення (SRS)»

| **Параметр** | **Опис** |
|---|---|
| Передумова | Користувач авторизований, у системі наявні слова, термін повторення яких настав згідно з алгоритмом SRS. |
| Тригер | Вибір кнопки «Почати повторення» на головному екрані застосунку. |
| Основний сценарій | 1. Система відображає завдання відповідного типу згідно з поточним рівнем вивчення слова.<br>2. Користувач дає відповідь, обираючи варіант, вписуючи текст, складаючи речення або озвучуючи слово (в залежності від типу завдання)<br>3. Користувач використовує наданий функціонал для полегшення прийняття рішення (підказки, друга спроба)<br>4. Система автоматично визначає оцінку та перераховує дату наступного показу слова через математичний модуль FSRS |

## 1.4 Висновки до першого розділу. Постановка задачі

У межах першого розділу було проведено комплексне дослідження предметної області мобільних застосунків для вивчення іноземних мов на платформі Android. Було проаналізовано динаміку ринку мобільних операційних систем, що визначило доцільність вибору Android як пріоритетного середовища. Вивчення популярних категорій Google Play виявило стабільно високий попит у секторі «Education» та категорії вивчення іноземних мов.

Для виявлення ключових проблем навчання, було детально вивчено психологічні аспекти, що перешкоджають засвоєнню знань: втрата мотивації та низька ефективність традиційних методів запам’ятовування. Ефективним рішенням цих проблем, на основі наукових досліджень та існуючих практик, було визначено використання алгоритму інтервальних повторень, та впровадження механік гейміфікації, для підтримки залученості користувачів.

Проведений порівняльний аналіз наявного ПЗ, наприклад таких головних аналогів як Duolingo, Anki та Memrise, дозволив визначити їхні переваги та недоліки. На основі отриманих даних було сформовано перелік функціональних та нефункціональних вимог, що лягли в основу технічних вимог проєкту.

Отримавши список визначених функціональних та технічних вимог до системи, можна сформулювати конкретні задачі проєктування, які мають бути реалізовані в подальшому під час виконання кваліфікаційної роботи:

– описати декомпозицію проєкту на рівні модулів, та визначити інтерфейси їх взаємодії;

– розробити схему серверної та локальної бази даних для збереження словників, історії повторень та прогресу користувача;

– спроєктувати та реалізувати логіку алгоритму FSRS у програмний код;

– розробити макети інтерфейсу на основі Material Design;

– вибрати сучасні технології та бібліотеки для реалізації архітектури;

– реалізувати модулі керування контентом (CRUD операції);

– впровадити ігрові механіки: систему «Streak», нагороди, завдання;

– спроєктувати сервер для взаємодії з зовнішніми API, для отримання транскрипцій, перекладів та прикладів вживання слів;

– визначити методи тестування та провести перевірку модулів на відповідність вимогам якості Android Developers;

– сформулювати підсумкові висновки на основі отриманих результатів виконання кваліфікаційної роботи та визначити можливі напрями подальшого розвитку системи.

# 2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 2.1 Вибір типу архітектури та шаблонів проєктування

Етап проєктування визначає здатність системи до масштабування, тестування та супроводу. Для розроблюваної мобільної інформаційної системи, яка базується на математичній моделі інтервальних повторень, та потребує асинхронної взаємодії із зовнішніми API, вибір архітектурного патерну має задовольнити такі визначені критерії:

– синхронізація станів гейміфікації, оскільки вони мають оновлюватися в реальному часі після кожної відповіді користувача в тестах;

– підтримка локальної БД, яка синхронізується з хмарою при появі мережі.

– розділення бізнес-логіки та інтерфейсу, оскільки математична модель має бути ізольована від інтерфейсу;

– інтерфейс повинен миттєво реагувати на зміну стану навчання;

– можливість окремого тестування алгоритмів планування повторень.

Згідно з дослідженнями факультету інформаційних технологій та математики Тетовського університету [17], станом на 2025 рік найбільш розповсюдженими та актуальними архітектурними патернами у сучасній розробці залишаються MVC, MVP та MVVM.

1) Model-View-Controller (MVC) (рисунок 2.1). Є класичним підходом до розробки ПЗ, який був першим адаптований для Android-платформи. Основна ідея полягає у розбитті додатка на три модулі, що дозволяє відокремити зберігання даних від їх візуалізації:

– model (Модель) - шар даних та бізнес-логіки, що відповідає за отримання та обробку даних, та не має знань про інтерфейс;

– view (Представлення) - шар візуалізації, який відображає інтерфейс програми, за допомогою Jetpack Compose або XML-лейаути;

– controller (Контролер) - головний шар обробки дій користувача на View, та оновлення даних у Model. В Android, це зазвичай Activity або Fragment.

В контексті мобільної архітектури, Controller зазвичай реалізується через класи Activity або Fragment. Головною проблемою MVC є ризик виникнення антипатерну «god pattern», коли внаслідок наростання функцій, бізнес-логіка починає надмірно накопичуватися всередині контролера. Це призводить до створення тісної зв'язності між компонентами та перетворення класу на «роздувальник» (bloater), що ускладнює підтримку та тестування коду [18]. Для логіки алгоритму SRS, це призводить до змішення його з кодом керування екраном, що робить систему нестійкою до помилок.

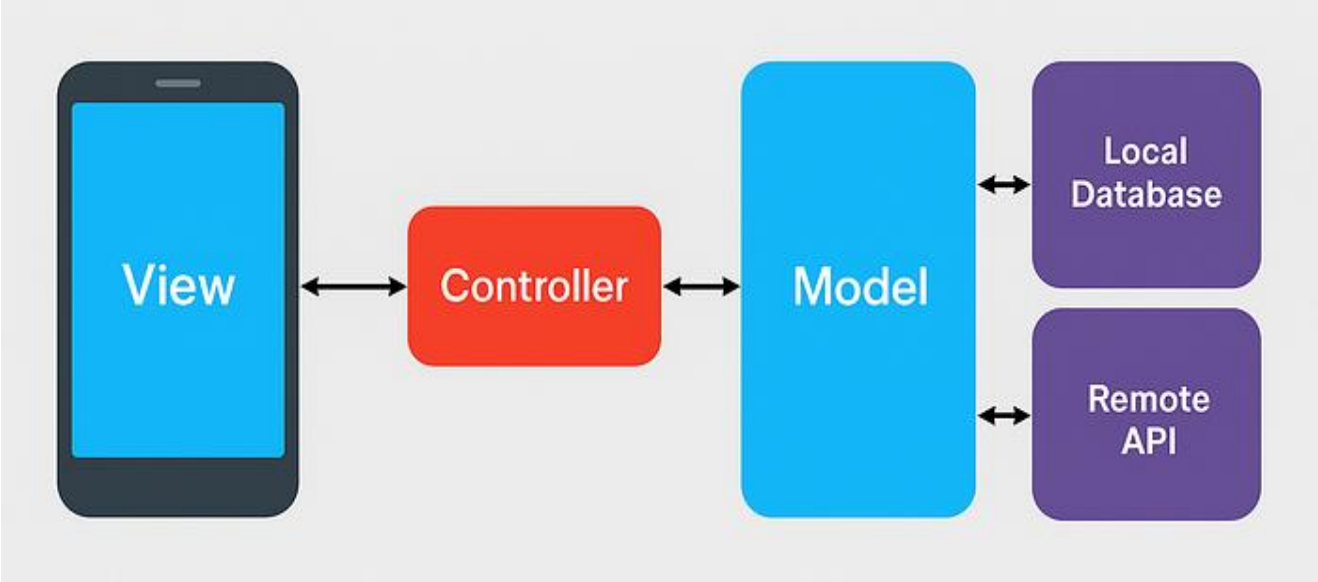



Рисунок 2.1 - Приклад архітектури MVC

2) Model-View-Presenter (MVP) (рисунок 2.2). MVP став заміною MVC, що мав на меті вирішити проблему перевантажених контролерів шляхом чіткого розподілу відповідальності. Головною зміною є введення інтерфейсів, які однозначно визначають, як саме взаємодіє бізнес-логіка та інтерфейс.

– Model (Модель) - шар даних, аналогічний до MVC;

– View (Представлення) - шар візуалізації. Як правило, це Activity або Fragment, які реалізують спеціально визначений інтерфейс;

– Presenter (Презентер) - центральний вузол, аналог контролера, який отримує дані з Model, форматує їх та View відображає їх через View.

Ключовою перевагою MVP над MVC, є повна ізоляція Presenter від Android API, що дозволяє легко проводити Unit-тестування. Проте, як зазначається у дослідженні 2025 року, MVP вимагає ручного зв’язування між компонентами. Це робить код занадто великим, та створює затримки при додаванні нових функцій через необхідність постійного оновлення інтерфейсів контрактів.

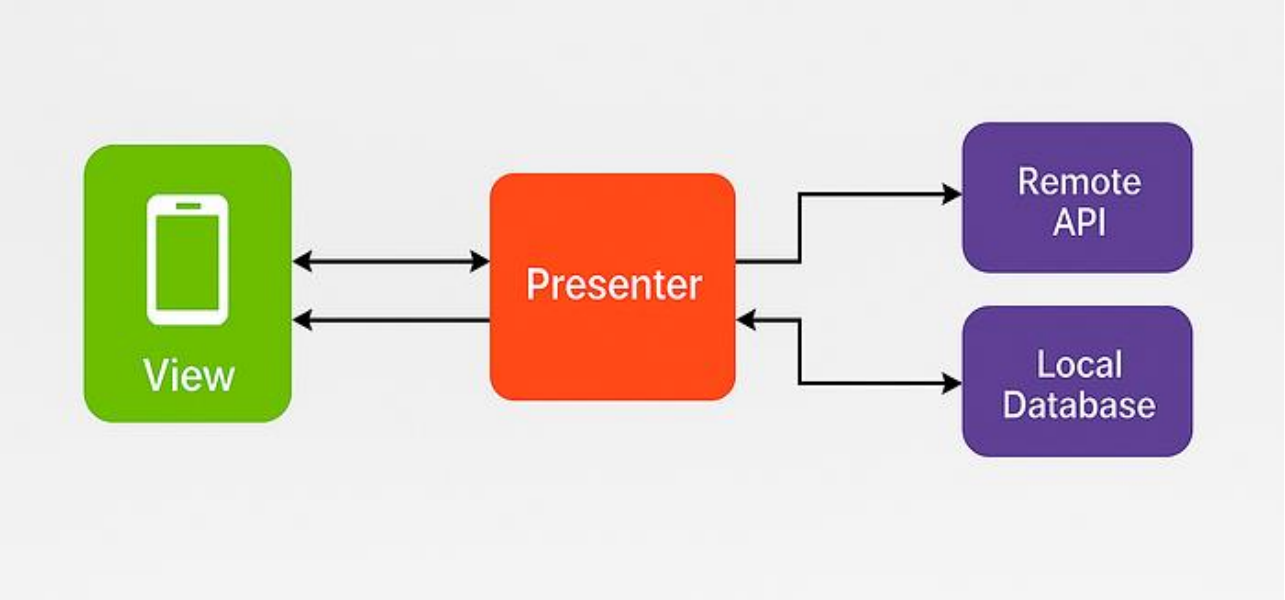



Рисунок 2.2 - Приклад архітектури MVP

3) Model-View-ViewModel (MVVM) (рисунок 2.3). Цей патерн архітектури став золотим стандартом завдяки ініціативі Google Modern Android Development, та базується на реактивному підході та механізмі спостереження за даними.

– Model - (Модель) - шар даних. Містить також Use Cases бізнес-логіки;

– View (Представлення) - декларативний інтерфейс, що автоматично реагує на зміни даних у ViewModel без необхідності явного оновлення елементів;

– ViewModel (Модель представлення) - посередник, який надає дані для UI через механізми спостереження за даними.

На відміну від Презентера у MVP, ViewModel не знає про існування конкретного View, що забезпечує високий рівень незалежності та спрощує повторне використання компонентів. Завдяки зв'язуванню даних, складність обробки динамічних змін інтерфейсу значно знижується.

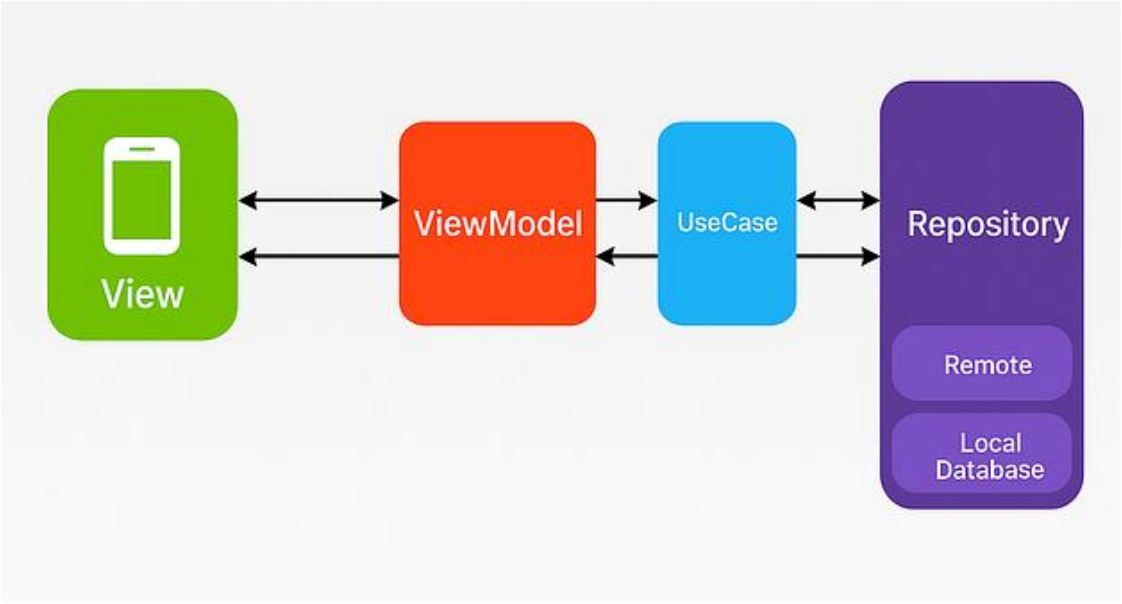



Рисунок 2.3 - Приклад архітектури MVVM

За результатом аналізу, для остаточно вибору архітектури, було проведено порівняння за визначеними критеріями (таблиця 2.1).

Таблиця 2.1 - Порівняльний аналіз архітектурних патернів

| **Критерій** | **MVC** | **MVP** | **MVVM** |
|---|---|---|---|
| Розділ обов'язків | Слабкий (перевантажений controller) | Середній (контракти) | Високий (ViewModel) |
| Автоматичне оновлення UI | Відсутнє (ручне оновлення View) | Середній (виклики методів інтерфейсу) | Високий (автоматичне оновлення) |
| Тестування | Складне (виділити логіку) | Легке (Presenter ізольований) | Дуже легке (ViewModel) |
| Синхронізація даних | Складна (вручну) | Середня (розширення інтерфейсів) | Висока (єдине джерело істини у ViewModel) |
| Масштабованість | Обмежена | Середня | Висока |

На основі проведеного порівняльного аналізу архітектурних патернів, для реалізації системи було обрано MVVM. Він найкраще відповідає визначеним функціональним та нефункціональним вимогам:

– реактивний підхід автоматично оновить екрани, що особливо важливо для прогрес-барів та гейміфікації;

– дозволяє повністю ізолювати математичну модель інтервальних повторень, та допомагає легко її тестувати;

– оскільки процес навчання передбачає часту зміну станів, механізм спостереження за даними забезпечує миттєву реакцію UI без затримок;

– MVVM ідеально інтегрується з бібліотекою Room для локального зберігання даних та репозиторіями для хмарної синхронізації через Firebase.

Таким чином, архітектура MVVM є оптимальним рішенням, оскільки вона є актуальною, забезпечує високу гнучкість при розробці складного функціоналу та дозволяє реалізувати асинхронну взаємодію з API. Це дозволяє підготувати систему для подальшого масштабування та доопрацювання.

## 2.2 Опис декомпозиції

### 2.2.1 Загальна декомпозиція

Загальну структуру проєкту було розбито на такі системи та підсистеми, де кожна відповідає за окремий функціонал:

– система автентифікації та профілю, включає в себе функції реєстрації, входу, відновлення доступу та початкового налаштування застосунку (вибір мови, ролі, сповіщень, тощо);

– система керування матеріалами є центральною для роботи з головними даними (словники, категорії та слова);

– система навчання (SRS) поєднує алгоритм SM-2/FSRS, тестовий модуль з різними типами завдань та планувальник щоденних повторень;

– система гейміфікації відстежує щоденну активність (Streak) та відповідає за завдання, систему рівнів та досягнень;

– система спільноти представляє з себе модуль, що обробляє публікації користувачів, обмін словами та перегляд активності інших;

– система аналітики та статистики потрібна для збору даних про навчання, що відповідає за візуалізацію прогресу та розрахунок статистики;

– система синхронізації забезпечує роботу локальної БД, запити до зовнішніх API та синхронізацію даних між локальним сховищем і хмарою;

### 2.2.2 Модульна декомпозиція

Розбивши проєкт на системи, модульна декомпозиція передбачає їх розбиття на менші, окремі модулі або компоненти, розподілені за архітектурними шарами. Це робиться для відокремлення бізнес-логіки від особливостей реалізації бази даних чи інтерфейсу, та визначення конкретних обов’язків.

Головним в системі є модуль сценаріїв використання, який керує виконанням загальних дій користувача (наприклад, додавання нового слова чи старт сесії). Окремо йде математичний модуль, що містить в собі алгоритм SRS та його механізми. Він є повністю автономним, отримує на вхід дані про успішність відповіді, дані слова та повертає нові розраховані характеристики навчання, що ізолює всі формули в одному місці.

Важливою частиною є модуль сесії навчання, який керує процесом тестування. Він тримає дані про слова, формує список слів для повторення на день, та реалізує логіку підказки та роботи над помилками, дозволяючи повторно відповісти після помилки. Він працює в парі з генератором завдань, що пропонує різні варіанти вправ на основі складності та прогресу слова. Так, програма може автоматично підбирати складність завдання, орієнтуючись на те, наскільки слово запам’яталось на цей момент, завдяки EaseFactor.

Оскільки в системі присутні два режими роботи: онлайн та офлайн, для зручності було визначено єдиний модуль керування даними, який буде реалізувати архітектурний патерн Repository. Він реалізує принцип єдиного джерела істини, абстрагуючи логіку вибору між локальним сховищем та

зовнішніми в єдиний потік даних [19]. Внутрішня логіка модуля самостійно визначає пріоритетність джерел: в першу чергу, дані записуються в локальне сховище, щоб забезпечити офлайн-доступ та швидке відображення, а при появі мережі, модуль автоматично запускає фонову синхронізацію з хмарним сховищем. Це дозволить звертатись завжди до одного і того самого модуля, без визначення яке саме сховище використати.

Система гейміфікації, в свою чергу, розділена одразу на декілька модулів, для кращого утримання користувача:

– модуль активності фіксує щоденну активність користувача в системі, та відповідає за керування ударного режиму;

– модуль прогресу та рівнів нараховує досвід за виконання тестів, та відстежує рівень прогресу користувача;

– модуль квестів керує списком щоденних завдань та видає досягнення;

Всі зовнішні запити обробляє модуль мережевої взаємодії, а імпорт даних ззовні (CSV/JSON, код поширення, спільнота), забезпечує модуль імпорту та обробки даних. Він відповідає за розпізнавання формату, перевірку даних на правильне форматування, помилки та їх подальше перетворення в коректний тип даних для особистого словника користувача.

У результаті модульної декомпозиції, було визначено такі модулі системи:

– модуль профілю та налаштувань;

– модуль сценаріїв роботи;

– модуль математичного алгоритму SRS;

– модуль логіки навчальних сесій;

– модуль генерації тестових завдань;

– модуль управління БД;

– модуль мережевої взаємодії (API);

– модуль імпорту даних;

– модулі гейміфікації (XP, Streak, Achievements, Tasks);

– модуль аналітики.

### 2.2.3 Декомпозиція моделі переходів станів

Декомпозиція моделі переходів станів потрібна щоб описати динамічну поведінку описаних до цього компонентів. Розроблювана система є дуже динамічною, оскільки має два режими роботи, систему гейміфікації протягом всієї сесії, та навчальні сесії, тому потрібно визначити їх основні стани взаємодії.

Оскільки до цього було визначено, що проєкт розробляється на реактивному підході (MVVM), зміна стану одного об'єкта повинна автоматично ініціювати реакцію в інших модулях, як наприклад оновлення графіків аналітики, статистики або коригування показників гейміфікації.

Процес навчальної сесії описує проходження користувачем підготовлених завдань, в залежності від прогресу їх вивчення. Цей процес включає відбір слів, валідацію та оновлення SM2 даних (таблиця 2.2). Вона може бути двох типів: загальна та повторення. Різниця полягає в тому, що сесія повторення додатково фільтрує слова по даті наступного повторення.

Таблиця 2.2 - Опис станів системи навчальної сесії

| **Стан** | **Опис** |
|---|---|
| Початок сесії | Система витягує слова, за датою наступного повторення та формує чергу на сьогодні. Після цього, на екрані відображається набір та кількість слів, готових до вивчення. |
| Відображення завдання | Демонструється завдання, тип якого залежить від EaseFactor слова. |
| Коригування черги | Якщо відповідь хибна, якість відповіді зменшується, обраний варіант прибирається, і дається друга спроба.<br>Якщо використано підказку, якість відповіді зменшується, і один з неправильних варіантів прибирається. |

Продовження таблиці 2.2

| | |
|---|---|
| Валідації | Система порівнює відповідь із правильно, та надає візуальний фідбек. |
| Визначення підсумків | Після завершення всіх завдань, для кожного слова розраховується нові значення алгоритму SRS та оновлюється загальна статистика. |

Слово є головною одиницею в системі, оскільки на їх основі будуються тести, картки, статистика та завдання. Враховуючи систему поступового вивчення слова, воно містить декілька станів, які визначають на якому етапі вивчення воно знаходиться (таблиця 2.3).

Таблиця 2.3 - Опис станів слова

| **Стан** | **Опис** |
|---|---|
| Нове (New) | Слово тільки додане до словника, але ще не проходило вивчення (інтервал не розраховано). |
| В активному вивченні (Learning) | Слово включене в цикл SRS, має стандартні характеристики вивчення та часто з’являється в тестах. |
| Засвоєно (Mastered) | Має високий рівень стабільності та EaseFactor, а інтервал повторення перевищує 180 днів. Слово вважається вивченим. |
| Повторення (Due) | Запланована дата повторення відповідає поточній даті. Слово автоматично додається до списку завдань на сьогодні |
| Критичний (Overdue) | Термін повторення слова настав або минув. Мають найвищий пріоритет у черзі. |

Окрім алгоритму вивчення слів, система робить великий акцент на гейміфікації та прогресу. Такі процеси зазвичай функціонують протягом всього життєвого циклу сесії, та проходять через декілька основних етапів (таблиця 2.4)

Таблиця 2.4 - Опис станів гейміфікації

| Стан | Опис |
|---|---|
| Активний Streak | Система відстежує часові мітки та сповіщає про необхідність завершення сесії для збереження прогресу. |
| Розрахунок XP | Динамічний стан нарахування балів залежно від складності завдання та якості відповіді. |
| Підвищення рівня | Динамічний стан обробка досягнення користувачем нового порогу досвіду, що запускає оновлення профілю. |
| Нарахування нагороди | Тригерний стан при виконанні якоїсь умови (досягнення або завдання) |

Оскільки система підтримує два стани роботи: онлайн та офлайн, має місце процес синхронізації даних (таблиця 2.5).

Таблиця 2.5 - Опис синхронізації даних

| Стан | Опис |
|---|---|
| Офлайн режим роботи | Дані зчитуються та записуються виключно в БД Room. Статус синхронізації позначається як «Очікування підключення». |
| Синхронізація | Процес обміну даними між локальною БД та хмарною, для забезпечення відповідності |
| Відповідність | Коли локальна база даних повністю відповідає даним з хмарної БД. |

## 2.3 Опис залежностей

Опис залежностей дозволяє визначити всі взаємодії між модулями, які було виявлено під час декомпозиції системи. Це дає побачити, як зміна в одному компоненті впливає на функціонування інших компонентів та системи в цілому, що є важливим для забезпечення стабільності та підтримуваності.

### 2.3.1 Міжмодульні залежності

Першими на розгляді є міжмодульні залежності, які описують взаємодію та обмін даними між модулями системи:

– модуль профілю та налаштувань безпосередньо звертається до системи автентифікації для керування обліковим записом, а також використовує модуль БД для збереження та оновлення персональних налаштувань користувача;

– модуль сценаріїв роботи виступає посередником, керуючи викликами між інтерфейсом та модулем керування даними;

– модуль логіки навчальних сесій є основним отримувачем даних із математичного модуля SRS, оскільки результати кожного тестування безпосередньо впливають на розрахунок майбутніх інтервалів;

– модуль генерації тестових завдань залежить від поточних параметрів слова (EaseFactor), що зберігаються в модулі тестування;

– модулі гейміфікації (XP, Streak, Achievements) підписані на події успішного завершення тестів у модулі навчальних сесій, що автоматично слугує тригером для миттєвого оновлення ігрового прогресу користувача;

– модуль анімацій реагує на зміни станів у модулі гейміфікації;

– модуль керування даними координує роботу між локальним та хмарним сховищами, забезпечуючи єдине джерело істини для решти модулів;

– модуль мережевої взаємодії отримує та надає актуальні дані для модуля імпорту, доповнюючи локальну БД новими даними;

– модуль аналітики збирає дані, що накопичуються в модулі керування даними за весь період використання застосунку, та надає по ним звіт.

### 2.3.2 Залежності всередині даних

Система має акцент на ієрархію даних та логіку їх вкладення між собою, тому вони мають наступні зв’язки:

– слова мають жорстку прив’язку до категорій, а категорії до словників, які

в свою чергу, прив’язуються напряму до користувача. Видалення або переміщення типу даних вищого рівня, призводить до зміни даних нижніх рівнів;

– статистика прогресу залежить від атрибутів конкретного слова та історії його повторень, щоб відобразити реальний прогрес у системі;

– досягнення користувача залежать від значень у модулях аналітики та гейміфікації, оскільки система автоматично відстежує накопичені показники для визначення моменту видачі відповідної нагороди.

### 2.3.3 Залежності між станами

Система також має залежності між станами, коли зміна одного компонента стає тригером для активації іншого:

– перехід слова у стан «Due» або «Overdue» створює залежність для модуля сесії, автоматично запускаючи формування окремої черги слів на повторення за датою на поточний день;

– перехід сесії у стан «Підсумок» викликає зміну станів у модулі гейміфікації, де відбувається нарахування досвіду та перевірка виконання завдань, і у модулі синхронізації, де відбувається збереження у хмарне сховище.

На основі проведеного аналізу було побудовано діаграму міжмодульних зв’язків (рисунок 2.4), що відображає архітектуру по рівням.

Діаграма демонструє структуру застосунку за принципом розподілу відповідальності, де кожен шар має чітко визначену роль. Шар представлення є найбільш залежним, оскільки він ініціює запити до бізнес-логіки через модуль сценаріїв роботи та відображає результати через модуль візуального фідбеку та інтерфейс користувача.

Шар бізнес-логіки містить модуль гейміфікації, модуль навчальних сесій, математичний модуль SM2 та модуль генерації завдань, виконуючи алгоритми навчання та будучи незалежним від конкретної реалізації бази даних.

Шар даних об'єднує модуль керування даними, модуль мережевої взаємодії, модуль імпорту та модуль аналітики, забезпечуючи роботу як з

локальною базою даних Room, так і з хмарним сховищем Firestore, та виступаючи фундаментальною основою для функціонування всього застосунку.

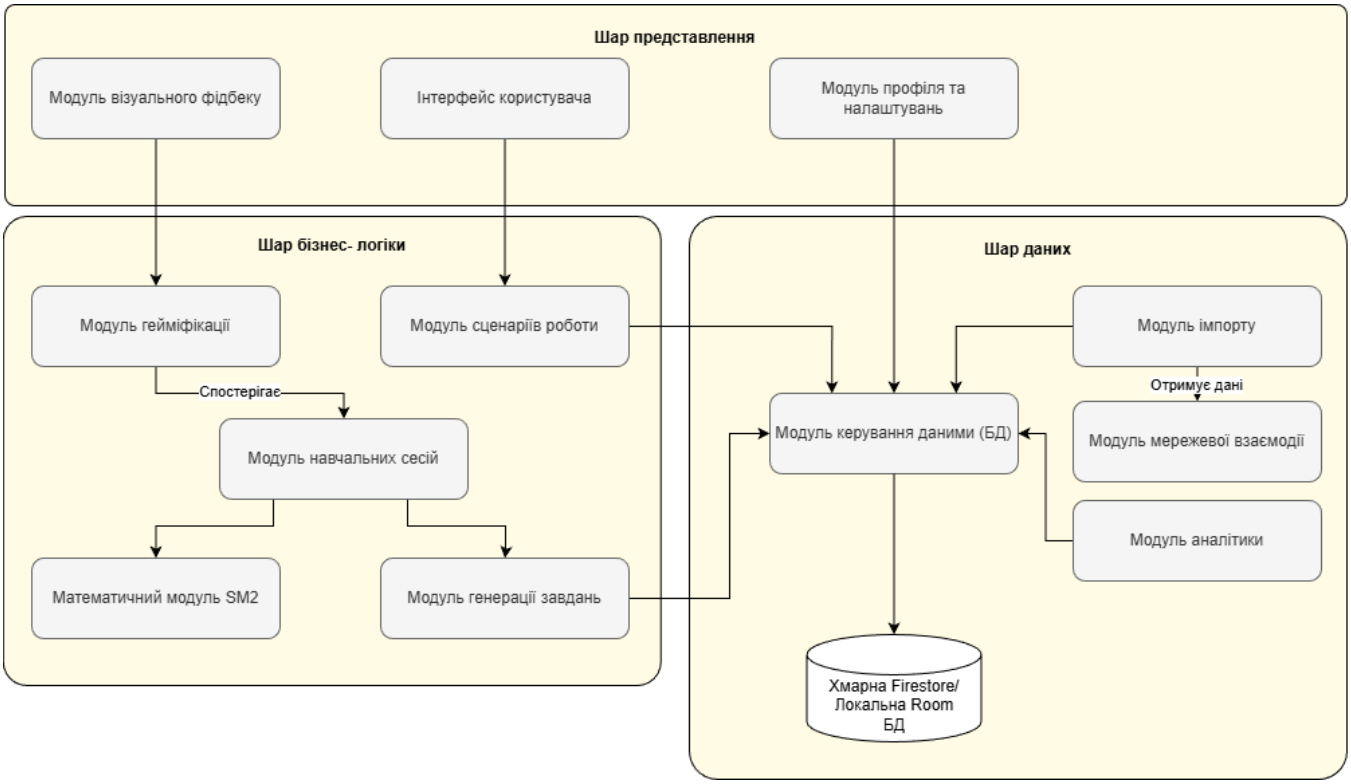



Рисунок 2.4 - Діаграма міжмодульних залежностей

## 2.4 Опис інтерфейсів

Основною роллю інтерфейсів в системі є проєктування репозиторію та контрактів для локальної і хмарної БД, щоб забезпечити розмежування між двома режимами роботи системи (онлайн та офлайн).

Для полегшення тестування та покращення загальної архітектури, ViewModel звертається не до конкретних класів, а до інтерфейсів, реалізація яких надається системою автоматично через Factory-патерн. Такий підхід дозволяє приховати складну логіку ініціалізації об'єктів від клієнтського коду, забезпечуючи слабку зв'язність між компонентами та спрощуючи підміну реалізацій під час тестування [20]. Перелік основних інтерфейсів, з яких складається архітектура системи, наведено у (таблиці 2.6).

Таблиця 2.6 - Опис основних інтерфейсів системи

| Інтерфейс | Опис |
|---|---|
| IUserRepository | Головний репозиторій. Містить основний функціонал роботи з даними в межах сесії, керує вибором між локальною та хмарною БД, виконує роль синхронізації. |
| IExternalService | Клас для роботи з зовнішніми API. Відповідає за отримання пошук даних про слово, роботу з перекладачем та отримання додаткової інформації (слово дня). |
| ISessionManager | Контролер навчальної сесії. Керує черговістю завдань на основі алгоритму SRS та верифікацією відповідей. |
| ILocalDataSource | Опис методів для роботи з локальною БД (Room). Дозволяє працювати в офлайн-режимі. |
| IRemoteDataSource | Опис методів для роботи з Firebase. Використовується для онлайн режиму з хмарним збереженням даних та синхронізацію між пристроями. |

Головним інтерфейсом для роботи з даними є IUserRepository. Він виступає головною точкою доступу до даних, та реалізує патерн Repository:

– Методи add…(),delete…(), edit…() реалізують CRUD логіку для даних, спочатку оновлюючи хмарне сховище, після чого фіксують зміни у локальній БД;

– fetchCategories() та fetchWords() реалізують Local First: система спочатку відображає дані з Room, паралельно запускаючи фонову синхронізацію з Firebase для оновлення застарілих записів;

– getReviewQueue() повертає список слів, які мають статус «Due» або «Overdue», формуючи план навчання на поточний день;

– updateSrsData() оновлює саме математичні коефіцієнти алгоритму. Це дозволяє інкапсулювати логіку SRS всередині репозиторію;

– Інтерфейс IExternalService реалізує роботу із зовнішніми джерелами:

– fetchWordDetails() повертає всю знайдену інформацію по запиту;

– getWordOfTheDay() повертає слово дня, яке визначене одне на всіх;

– getTranslation() напряму звертається до перекладачів з запитом;

## 2.5 Детальне проєктування модулів

Детальне проєктування модулів містить в собі опис реалізації абстрактних інтерфейсів та декомпозиційних схем. На основі визначених раніше міжмодульних зв'язків та описів інтерфейсів, у цьому підрозділі здійснюється деталізація роботи системи, реалізація математичного алгоритму навчання та візуалізація ключових процесів обробки даних.

У патерні MVVM, кожна модель має чітко визначену відповідальність, отже система містить в собі такі структурні елементи бізнес-логіки:

– ViewModels (WordViewModel, CategoriesViewModel, SessionViewModel…) існують для кожного фрагмента та забезпечують зв’язок між UI та логікою, зберігаючи стан екрана в об'єктах StateFlow;

– Repository (UserRepositoryImpl) є центральним об'єктом системи, який містить в собі всі методи для роботи з даними та забезпечує автоматичну взаємодію як з локальним сховищем Room, так і з хмарним сховищем Firebase;

– SRS Engine (SRSEngine, GradeCalculator) є модулем математичних розрахунків для реалізації SRS алгоритму;

Математичне ядро застосунку базується на модифікованому алгоритмі SM-2, який адаптує навчальний план під швидкість запам'ятовування конкретного користувача [21]. Алгоритм працює з трьома параметрами:

– кількістю повторень (n);

– коефіцієнтом складності (EF), з початковим значенням 2,5;

– інтервал наступного повторення у днях (I);

Особливістю реалізації алгоритму для розроблюваного ПЗ є відмова від суб’єктивної оцінки користувачем, на користь автоматичного визначення якості відповіді системою. Це реалізується шляхом аналізу факторів тестування

(кількість спроб та використання підказки) під час виконання завдань, та зменшення оцінки. Адаптована логіка роботи алгоритму містить такі кроки:

1. Отримання оцінки якості відповіді q, критерії якої базуються на оригінальній шестибальній шкалі (0-5), запропонованій у методології SuperMemo-2 [22]. Порівняння оригінальних описів алгоритму та адаптованих критеріїв наведено у (таблиці 2.7). Розрахунок автоматично виконується модулем GradeCalculator та передається до математичного ядра SRS.

Таблиця 2.7 - Критерії оцінки якості відповіді

| **Оцінка (q)** | **Опис з SM-2** | **Адаптовані критерії** |
|---|---|---|
| 5 | Ідеальна відповідь | Правильна відповідь з першої спроби, без використання підказок. |
| 4 | Правильна відповідь після певного роздуму | Правильна відповідь з першої спроби після використання підказки. |
| 3 | Правильна відповідь, згадана з труднощами | Правильна відповідь з другої спроби (після помилки у першій). |
| 2 | Неправильна відповідь, але вона здається знайомою | Неправильна відповідь після обох спроб, підказки не використовувалися. |
| 1 | Неправильна відповідь; слово згадується важко | Неправильна відповідь після обох спроб, використовувалися підказки. |
| 0 | Повна відсутність впізнавання слова | Особливі випадки |

2. Розрахунок нового значення EF за формулою (2.1), де отримане значення обмежується нижнім порогом 1.3. Це дозволяє системі плавно знижувати складність при успіхах та підвищувати її при частих помилках, запобігаючи

ситуації надто частого повторення одного й того самого слова.

$$EF' = EF + (0.1 - (4 - q) \cdot (0.08 + (5 - q) \cdot 0.02)) \quad (2.1)$$

3. Визначення нового значення інтервалу I:

– якщо n = 1 - I = 1 день;

– якщо n = 2 - I = 6 днів;

– для n > 2 - I = $I_{n-1} \cdot EF$;

Якщо оцінка q>3, слово вважається забутим, прогрес слова анулюється, а інтервал скидається до 1 дня. Використані і отриманні параметри, та загальну візуалізацію алгоритму визначення оцінки, нового інтервалу, та загального процесу роботи з відповіддю, можна побачити на діаграмі блок-схеми адаптованої роботи алгоритму SM-2 на (рисунку 2.5).

Паралельно з математичним моделюванням інтервалів повторень, для забезпечення регулярності взаємодії користувача з алгоритмом, у проєкті реалізовано підсистему гейміфікації, спроєктовану на основі фреймворку Octalysis Ю-Кай Чоу [23].


Рисунок 2.5 - Діаграма алгоритму розрахунку інтервалів навчання

Цей фреймворк визначає основні драйвери мотивації, кожен з яких відповідає за окремий психологічний аспект залученості користувача. У розробленій системі, вони реалізовані наступним чином:

– розвиток та досягнення реалізовано через прогресивне нарахування балів досвіду (XP) за успішні відповіді ($q > 3$), що візуалізує прогрес вивчення мови;

– уникнення втрат впроваджено через механіку «ударного режиму» (Streak), де щоденна практика у тестуванні стає цінністю, яку користувач прагне зберегти, уникаючи переривання ланцюга активності;

– власність та володіння реалізовано через персоналізацію навчального процесу з можливістю створення та поширення власних слів, що формує відчуття контролю над власним словниковим запасом та навчальним процесом;

– творчість та зворотний зв'язок забезпечуються через модуль візуального фідбеку та анімаційні ефекти при досягненні цілей, завдяки чому, користувач миттєво отримує валідацію своїх дій, що стимулює подальші ітерації навчання;

– епічне значення та покликання втілено через концепцію маскота-провідника, який супроводжує користувача на шляху освоєння мови, перетворюючи рутинне навчання на частину спільної «місії» з персонажем.

Такий підхід дозволяє перетворити сухий процес навчання на більш захопливий та ігровий процес, мінімізуючи ризик припинення навчання на етапах, коли складність матеріалу зростає.

Головною ціллю сервера є збір та повернення інформації про слово з зовнішніх серверів до клієнта. Цей процес передбачає послідовну перевірку джерел інформації та асинхронну взаємодію із зовнішніми сервісами. Логіку цього процесу представлено на діаграмі послідовностей у (Додатку А.3)

Алгоритм складається з таких основних етапів:

1. Користувач вводить слово у застосунку та надсилає запит пошуку.

2. Система звертається до хмарної БД у пошуках раніше отриманих даних для цього слова, щоб мінімізувати кількість запитів до зовнішніх сервісів.

3. Якщо даних немає у БД, сервер надсилає HTTP-запит до зовнішніх API

словників (Free Dictionary API, Wiktionary API), після чого намагається отримати дані методом скрапінгу сайтів словників. Якщо дані не знайдено, запит передається ШІ-моделі із заздалегідь підготовленим промптом.

4. Після отримання базової інформації, система перевіряє наявність додаткових даних, таких як аудіовимови або перекладу на потрібну користувачеві мови. За їх відсутності, сервер надсилає запити до сервісів перекладу або синтезу мовлення у відповідній ситуації.

5. Після збору всіх даних, система формує єдиний JSON-файл та передає його модулю, який перетворює його на екземпляр класу Word і зберігає у БД. У разі відсутності даних, користувач отримує текстове повідомлення про помилку.

## 2.6 Детальне проєктування даних

На цьому етапі було визначено ключові сутності системи, зв’язки між ними, та основні потоки даних. Оскільки система використовує одразу дві БД, NoSQL та SQL, проєктування діаграми сутностей було розроблено саме для останньої, оскільки вона охоплює більшу деталізацію та розбиття таблиць. Базуючись на цьому, основними сутностями системи є:

– сутність «User» містить основну інформацію про користувача, його дані, стан профілю та статистику;

– сутність «UserSettings» містить налаштування застосунку конкретного користувача, включаючи мову застосунку, сповіщення, тощо;

– сутність «Vocabulary» представляє контейнер для категорій, який містить в собі категорії, та визначає мову, теги, та основні налаштування. Має зв’язок із сутністю «User» як 1:N;

– сутність «Category» представляє контейнер для слів, містить в собі статистику, для швидкого відображення, опис та користувацькі налаштування. Має зв’язок із сутністю «Vocabulary» як 1:N;

– сутність «Word» є головною одиницею даних в проєкті, представляє з

себе слово, та детальну всеосяжну інформацію про нього. Окрім основних даних, містить посилання на таблицю з додатковими значеннями цього слова. Має зв’язок із сутністю «Category» як 1:N;

– сутність «RelatedWord» доповнює сутність «Word», представляючи з себе її більш скорочену версію, яка потрібна для додаткових значень (синоніми, антоніми, інші значення) існуючого слова;

– Сутність «Tag» виконує роль додаткових позначень для слова, визначаючи його підкатегорії, що використовуються для фільтрації, тестування та винесення головних атрибутів (рід слова, частина мова, тип, тощо.).

Детальний опис атрибутів цих сутностей та їх взаємозв’язки між собою представлено на ERD-діаграмі (рисунку 2.6). Проєктування схеми бази даних базується на принципах моделювання сутність-зв’язок, що дозволяє візуалізувати логічну структуру даних, типи зв’язків та обмеження цілісності ще до етапу фізичного створення таблиць [24].

Для реалізації вкладеної структури таких даних як синоніми та теги, використано підхід нормалізації, який визначає що:

– таблиця «RelatedWord» зберігає синоніми, антоніми та приклади слова, заміняючи великі масиви списків на окрему таблицю;

– система тегів реалізована через зв'язок «Many-to-Many» між таблицями «Tag» та «WordTag». Поле isPredefined у таблиці «Tag» позначає його як сталий тег, який визначений системою, та використовується для фільтрації або у тестуванні як характеристика слова (рід, частина мови, тощо);

Одним з основних сценаріїв роботи системи з користувачем, є процес проходження тестування. Цей сценарій має дві варіації:

– проходження тесту зі словами, термін повторення яких настав сьогодні. В результаті проходження, статистика SRS оновлюється, користувач отримує досвід за проходження та досягнення;

– проходження тесту зі всіма словами в категорії. Цей варіант потрібен для самоперевірки та практики, в його результаті статистика не оновлюється

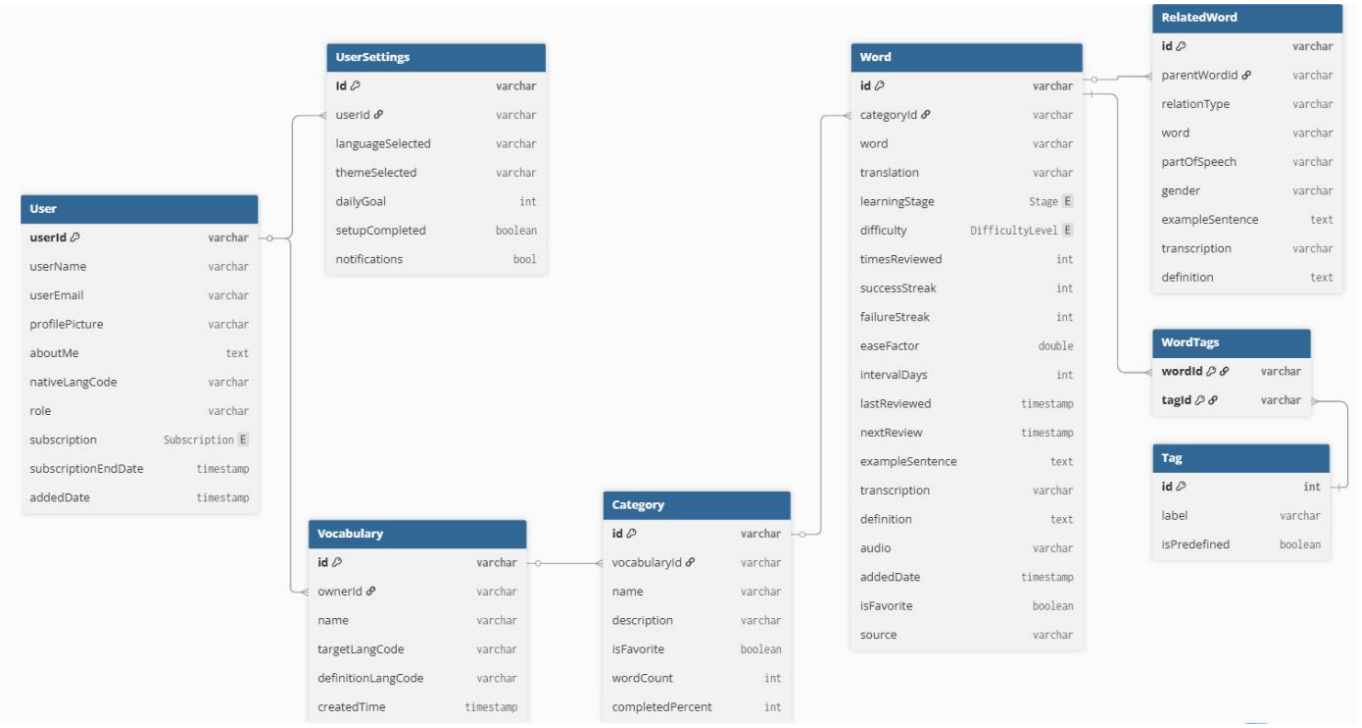



Рисунок 2.6 - Діаграма сутностей системи (ERD)

Сам процес проходження тестування, та взаємодію даних в ньому, представлено на діаграмі послідовності (рисунок 2.7).

Сесія розпочинається з ініціації користувачем повторення, після чого застосунок надсилає запит до сервера для отримання слів, дата наступного повторення яких не перевищує поточний день. Отриманий масив слів завантажується у список, після чого для кожного слова у циклі обирається тип вправи на основі його поточного значення Ease Factor.

Взаємодія з тестом починається з відображення вправи у QuizFragment, та очікування дій користувача. Після отримання відповіді, застосунок перевіряє її коректність у ProcessWordReview, та надає відповідний фідбек, позитивний при правильній відповіді або показ правильного варіанту при помилці. Результат відповіді передається до SRSModule, де на його основі формується оцінка якості (0-5), розраховуються нові параметри SM-2 та оновлюється дата наступного повторення. Оновлений документ слова зберігається у базі даних, після чого на стороні клієнта оновлюється модуль гейміфікації.

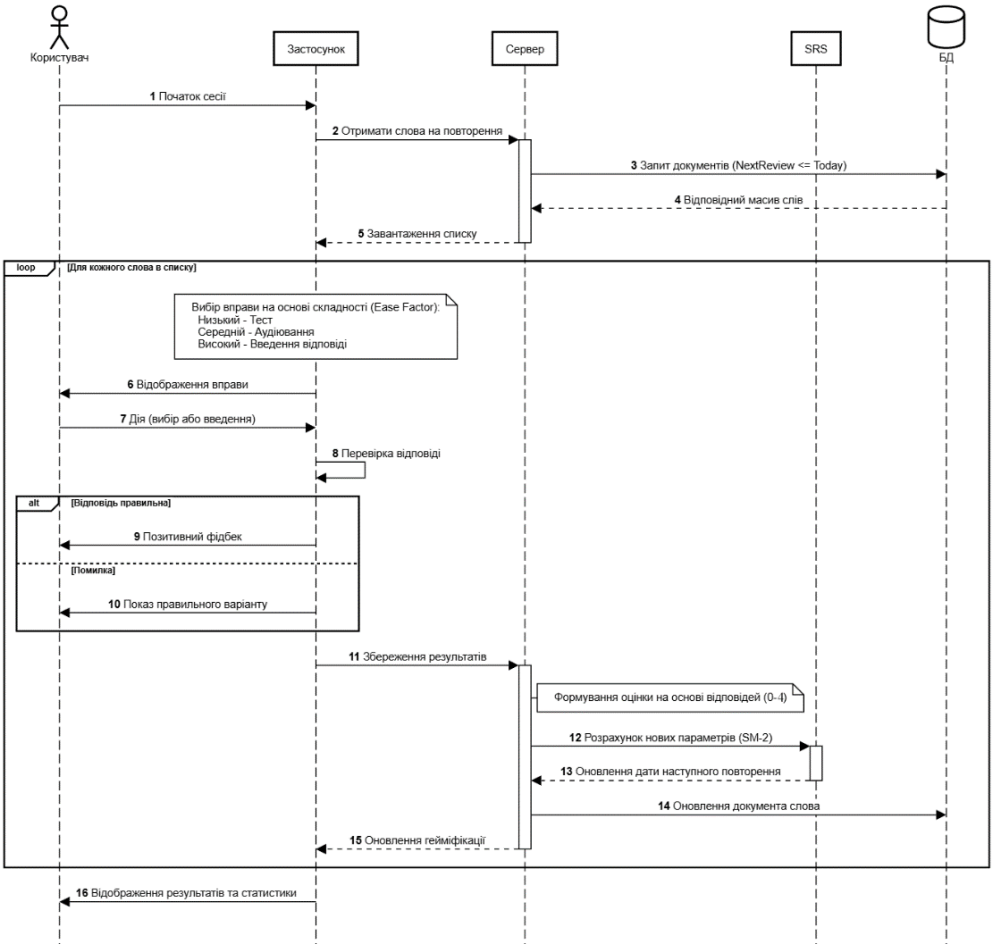



Рисунок 2.7 - Діаграма послідовності процесу тестування

## 2.7 Аналіз та вибір технологій і методів реалізації застосунку

У даному підрозділі проведено детальний порівняльний аналіз доступних мов програмування, фреймворків та методологій розробки з метою обґрунтованого відбору технологічного стеку, що найповніше відповідає поставленим функціональним та нефункціональним вимогам системи.При виборі мов програмування під платформу Android основними конкурентами є Java та Kotlin. Згідно з політикою «Kotlin-first», проголошеною Google у 2019 році, саме ця мова є стандартом для створення нових застосунків [25].

1. Java - традиційна об’єктно-орієнтована мова, що довгий час була стандартом для Android.

Переваги:

– велика база документації та готових рішень;

– стабільна робота на будь-яких версіях системи;

– широка підтримка бібліотек.

Недоліки:

– велика кількість повторюваного коду та великий синтаксис;

– відсутність вбудованої безпеки для null, що вимагає постійних перевірок на значення;

– повільніший розвиток порівняно з новими мовами.

2. Kotlin - сучасна статично типізована мова, розроблена для виправлення помилок Java та офіційно рекомендована Google як пріоритетна для Android.

Переваги:

– лаконічність синтаксису;

– повна сумісність із Java;

– механізм безпеки Null Safety;

– зручна підтримка фонових процесів;

– підтримка корутин для асинхронної роботи.

Недоліки:

– Дещо довша первинна компіляція для великих проєктів.

Як було зазначено, Kotlin має підтримку асинхронної обробки, що дозволяє легко виконувати важкі запити до API та операції з БД фоновому режимі, без затримки інтерфейсу [26]. З огляду на наведені аргумент, для забезпечення безпечної обробки навчання та відповідності сучасним стандартам, мовою програмування для проєкту було обрано Kotlin.

Окремо від мови програмування, для мобільної розробки треба обрати метод верстки екранів. Наразі існує два варіанти версти: класичний підхід (XML/Views) та сучасний декларативний (Jetpack Compose).

1. XML Views - традиційний імперативний підхід до побудови інтерфейсу, де візуальна структура описуєтьсяв окремих XML-файлах розмітки.

Переваги:

– вивчена та стабільна технологія, що використовувалась багато років;

– наявність великої кількості готових шаблонів на редакторів;

– підтримка старих версій Android.

Недоліки:

– складність поєднання стану коду та дизайну;

– велика кількість шаблонного коду.

2. Jetpack Compose - сучасний інструментарій, де інтерфейс будується за допомогою функцій на Kotlin [27]

Переваги:

– декларативний підхід до верстки;

– легка реалізація складних анімацій та переходів;

– менший обсяг коду без необхідності створення додаткових файлів.

Недоліки:

– потребує вивчення нової логіки побудови екранів, що кардинально відрізняється від класичної;

– обмежена підтримка дуже старих пристроїв.

З огляду на попередній досвід роботи з XML, та необхідність вивчати з нуля повністю відмінний підхід Jetpack Compose, було обрано першочергову реалізацію екранів за допомогою XML, з подальшим переходом на Jetpack Compose, для відповідності до сучасних стандартів.

Одним з найголовніших в проєкті є модель розробки. При розробці мобільних застосунків, найчастіше робиться вибір між каскадною та ітеративною моделями. Згідно дослідження існуючих SDLC на момент 2023 року, вибір залежить від типу проєкту, його складності, масштабів та технологій, визначаючи WaterFall та Agile як найбільш популярні варіанти у розробці [28].

1. Каскадна модель (Waterfall) - послідовний перехід від планування до тестування без повернення назад.

Переваги:

– чітке розуміння всіх етапів та термінів завершення роботи ще до початку написання коду реалізації;

– зручність коли вимоги абсолютно зрозумілі та не будуть змінюватися.

Недоліки:

– відсутність гнучкості;

– великий ризик виявити помилку в алгоритмі лише в кінці роботи.

2. Ітеративна модель (Agile) - розробка короткими циклами з постійним отриманням робочого прототипу.

Переваги:

– висока гнучкість роботи;

– можливість поступово додавати функціонал.

Недоліки:

– складність розуміння фінальних термінів.

Вибір фреймворку Ktor для реалізації серверної частини зумовлений його використанням мови Kotlin та асинхронністю процесів на основі корутин [29], що забезпечує високу продуктивність при мінімальному споживанні ресурсів.

Для реалізації проєкту обрано ітеративну модель розробки, що дозволить спочатку створити базовий MVP із ключовим функціоналом, а потім поступово розширювати та доповнювати його новим.

## 2.8 Висновки до другого розділу

В результаті проведеного аналізу, дослідження, порівняння та проєктування програмного забезпечення було сформовано повну технічну специфікацію, яка стане фундаментом для подальшої програмної реалізації.

На основі аналізу функціональних та нефункціональних вимог, було прийняте рішення використати архітектурний патерн MVVM у поєднанні з принципами Clean Architecture. Такий підхід забезпечує чіткий розподіл відповідальності між шарами представлення, бізнес-логіки та даних, що гарантує стабільну та передбачувану роботу складних математичних алгоритмів навчання, а також спрощує подальше тестування і розширення функціональності системи.

У процесі декомпозиції було визначено модульну структуру застосунку та описано ієрархію об’єктів, де головними було визначено репозиторій та менеджер навчальних сесій. Побудовано діаграми міжмодульних залежностей та послідовності дій для деталізації процесу взаємодії між компонентами під час виконання ключових сценаріїв, таких як проходження тестування та автоматичне підтягування даних із зовнішніх API.

Для детального проєктування даних, було розроблено модель бази даних (ERD), яка враховує специфіку реляційної БД Room для локального зберігання та документ-орієнтованої структури Firestore для синхронізації між пристроями.

Враховуючи великий акцент на математичну модель алгоритму SRS, окрему увагу було приділено детальному проєктуванню його логіки, в результаті чого, було розроблено модуль на основі адаптованого алгоритму SM-2, де суб’єктивну оцінку замінено на автоматизований аналіз дій користувача.

Результатом розділу стала визначена архітектура для подальшої програмної реалізації. Вона відповідає всім попередньо встановленим та сучасним вимогам до програмного забезпечення, орієнтується на високу гнучкість до майбутніх розширень та оптимізацію взаємодії даних.

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 Програмна реалізація модулів

На основі визначених у другому розділі архітектурних рішень, цей етап передбачає втілення їх у функціональний код. Перед початком роботи над проєктом, для забезпечення масштабованості та легкості тестування коду, організація файлової архітектури була спроєктована за принципами «Clean Architecture». Згідно неї, архітектура організована за функціональним призначенням компонентів (рисунок 3.1), що ізолює бізнес-логіку від деталей реалізації платформи Android [30].

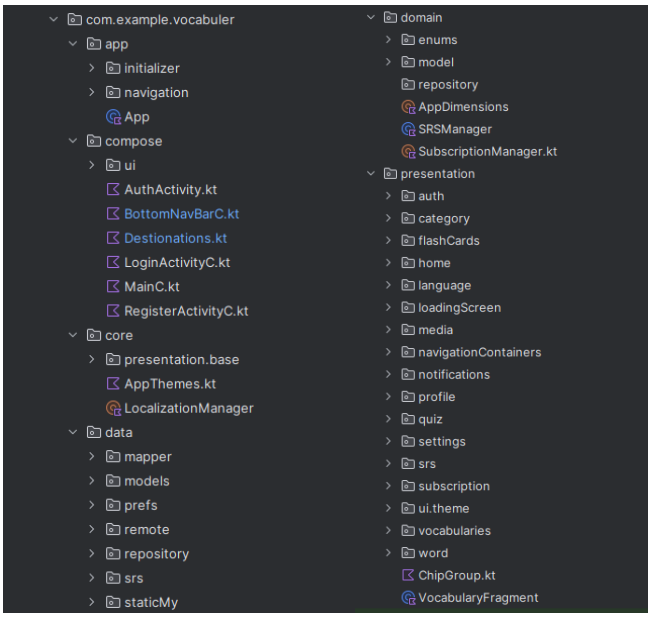



Рисунок 3.1 - Структура пакетів проєкту

Згідно цієї практики, зміна у одному шарі, не вимагають модифікацій в інших, що полегшує роботу та загальну архітектуру. Всі файли було структуровано за категоріями по принципу розподілу відповідальності:

– директорія «domain» є центральним ядром системи, що містить бізнес-правила, сутності та інтерфейси репозиторіїв. Вона повністю незалежна від Android SDK та будь-яких зовнішніх бібліотек;

– директорія «data» представляє шар для реалізації даних, який описує конкретну реалізацію Room БД, взаємодію з Firebase та імплементацію інтерфейсів визначених в domain для репозиторію, збираючи всю логіку роботи з джерелами даних в одному місці;

– директорія «presentation» реалізує шар представлення, що містить все що стосується візуалізації на екрані (Activity, Fragment, ViewModels, тощо). Вона залежить від domain, але не data;

Програмна реалізація базується на принципі архітектури Single Activity для окремих функціональних модулів. Замість використання однієї активності для всього застосунку, було використано поділ на окремі активності, кожна з яких виступає контейнером для конкретної логічної частини системи: аутентифікації, основної взаємодії користувача з системою, тестування та флеш-картками.

Це дозволяє чітко розмежовувати відповідальність між різними частинами застосунку та спрощує управління пам'яттю. Активність, що більше не потрібна, повністю вивантажується системою разом з усіма пов'язаними фрагментами (наприклад ланцюжок авторизації.

За принципом Single Activity, активність слугує єдиною стабільною точкою входу та контейнером для життєвого циклу, а зміна контенту реалізується через динамічну заміну фрагментів у межах одного контейнера. Це забезпечує передбачувану поведінку системної кнопки «Назад», спрощує передачу даних між екранами одного модуля та дозволяє ефективно використовувати Navigation Component для візуалізації графів переходу [31].

Керування переходами між екранами, та передача даних реалізовані за допомогою компонента Navigation. Всі можливі екрани, переходи між ними представлені у nav_graph, що визначає структуру взаємодії користувача з інтерфейсом [32]. Найбільшим з них є граф основної взаємодії користувача з системою, що представляє з себе головну сторінку, словники, статистику та профіль користувача (рисунок 3.2). Він візуалізує всі можливі маршрути навігації: від кожного екрану вузли вказують на доступні переходи, що унеможливлює виникнення недоступних сторінок.

Для забезпечення безпеки передачі аргументів між екранами, використано бібліотеку Safe Args, яка автоматично генерує класи для безпечної передачі аргументів [33]. Вона дозволяє визначати потрібні для переходу на екран аргументи прямо в графі, та виключає виникнення помилок під час виконання програми, пов'язаних із неправильними ключами або типами даних

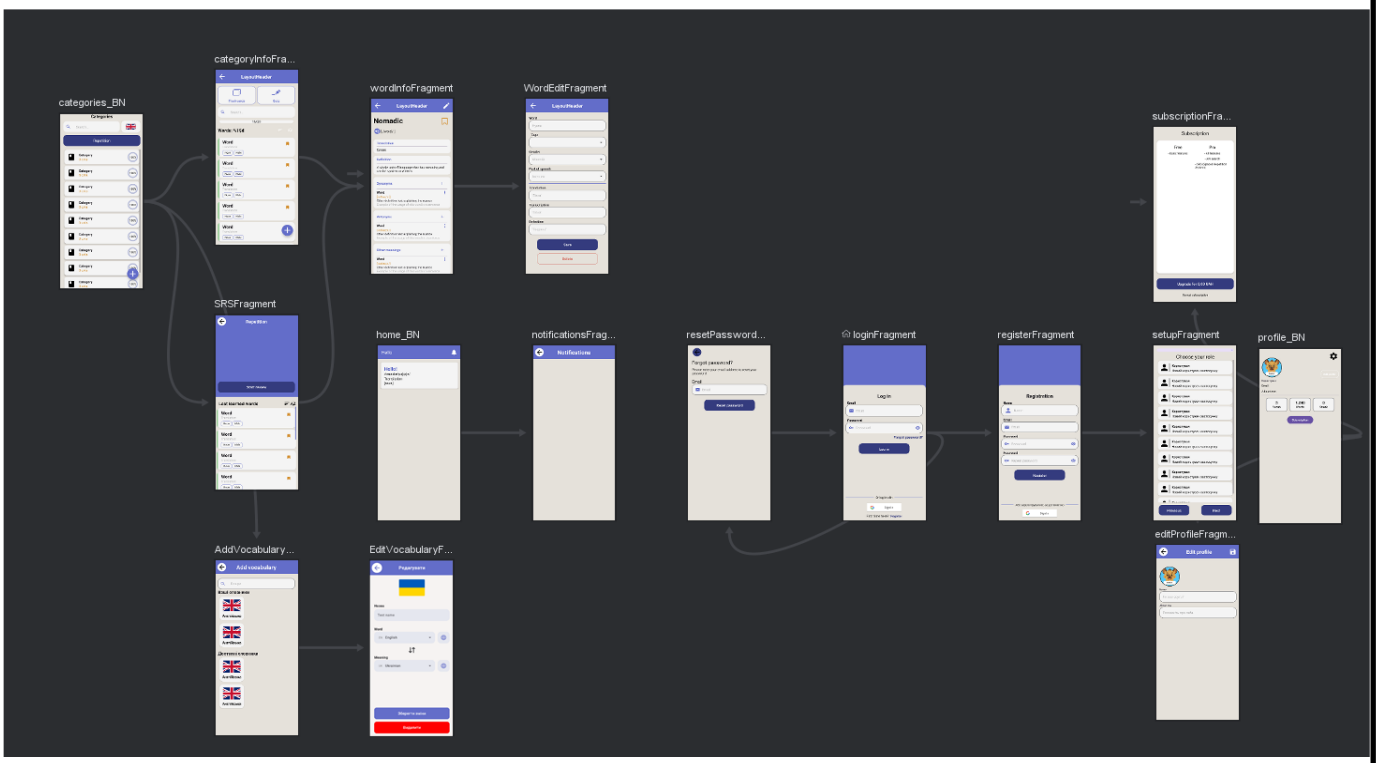



Рисунок 3.2 - Граф навігації основних екранів застосунку

Точкою входу в систему є активність LoadingActivity, вона виконує всі необхідні початкові перевірки перед завантаженням основного інтерфейсу, такі як визначення стану авторизації за рахунок перевірки токена входу, перевірка підключення до мережі, встановлюючи в якому режимі буде працювати система. Після цього, виставляються налаштування сесії, завантажуються дані та відбувається перехід на відповідну до сценарію активність:

– якщо токен авторизації існує, то навігація перемикається на екрани MainGraph, в якому містяться основні екрани системи;

– якщо токену немає, тоді навігація екранами залишається у AuthGraph, та користувач переходить на вікно авторизації.

Вікно загрузки LoadingActivity працює з LoadingViewModel, в якій відбувається вся бізнес логіка перевірок на фоні, та перехід екранів. Нижче наведено частину його коду, яка демонструє асинхронну перевірку стану авторизації та ініціалізацію синхронізації:

```
class LoadingViewModel(
  private val authRepository: AuthRepository,
  private val syncManager: SyncManager
): ViewModel() {
  private val _navigationEvent = MutableSharedFlow<NavTarget>()
  val navigationEvent = _navigationEvent.asSharedFlow()
  init {
    checkInitialState()
  }

  private fun checkInitialState() {
    viewModelScope.launch {
      val isUserAuth = authRepository.isUserAuthorized()
      if (isUserAuth) {
        syncManager.performInitialSync()
        _navigationEvent.emit(NavTarget.Main)
      } else {
        _navigationEvent.emit(NavTarget.Auth)
      }
    }
  }
}
```

Математичне ядро системи реалізовано у модулі SRSModule. Програмний код адаптованого алгоритму SM-2 відповідає за розрахунок дати наступного показу слова, інтервалу та коефіцієнту складності на основі історії успішності відповідей користувача. Як було зазначено у проєктуванні, особливістю реалізації є заміна суб'єктивної оцінки відповіді користувачем на автоматичне визначення її системою, з урахуванням різних факторів на момент відповіді. Такий підхід позбавляє користувача від необхідності самостійно оцінювати складність слова, що зменшує когнітивне навантаження, пришвидшує сесії та дає результати, що гуртуються на реальній статистиці. Код модуля наведено нижче:

```
object SRSModule {
  const val INITIAL_EF = 2.5
  private const val MIN_EF = 1.3

  fun calculateNextReview(previousInterval: Int,previousEF: Double,guessingStreak: Int,rating: SrsRating):
SrsUpdate {
    val now = Date()
    val newEF = (previousEF + (0.1 - (4 - rating.value) * 0.15)).coerceAtLeast(MIN_EF)

    var newInterval: Int
    var newRepetitions: Int

    if (rating.value >= 2) {
      newRepetitions = guessingStreak + 1
      newInterval = when (newRepetitions) {
        1 -> 1
```

```
        2 -> 6
        else -> (previousInterval * newEF).toInt()}
      if (rating.value == 2)
        newInterval = previousInterval.coerceAtLeast(1)
    } else {
      newRepetitions = 0
      newInterval = 1}
    val nextReviewMillis = now.time + TimeUnit.DAYS.toMillis(newInterval.toLong())

    return SrsUpdate(
      nextReviewDate = Date(nextReviewMillis),
      newInterval = newInterval,
      newEF = newEF,
      newRepetitions = newRepetitions,
      lastReviewDate = now
    )
  }
}
```

Для деяких параметрів є обмеження на мінімальне значення .coerceAtLeast(MIN_EF). Воно є важливим для стабільності моделі, оскільки якщо показник EaseFactor впаде нижче критичної позначки 1.3, система застрягне в стані циклу, за якого інтервали між повтореннями слів перестануть збільшуватися. Окрім того, обнуління показника newRepetitions у випадку оцінки нижче двох балів потрібен для гнучкої адаптації системи під індивідуальну швидкість забування інформації, миттєво повертаючи складне для запам'ятовування слово на початковий етап.

## 3.2 Розроблення бази даних

Підключення бази даних реалізовано через абстракцію, що дозволяє легко замінити або змінити БД при потребі. Оскільки система може працювати в двох режимах, залежно від стану підключення до інтернету, наразі оголошено два джерела даних, для кожного з яких визначено відповідний інтерфейс DataSource, який містить базові методи для роботи системи (збереження налаштувань, маніпуляції зі словами, категоріями та словниками тощо):

– UserLocalDataSource реалізує роботу з Room як локальним сховищем;

– UserRemoteDataSource реалізує роботу з Firebase як хмарним сховищем.

Головним оператором є UserRepository, який реалізує патерн Repository, і до якого звертається система, коли їй потрібно надіслати будь-який запит до БД. Це дозволяє йому керувати роботою між локальною та віддаленою БД,

реалізуючи логіку «Offline-first», коли система завжди відображає актуальні локальні дані, а фонова синхронізація оновлює їх при змінах у хмарі. При запиті до модуля він працює за наступним сценарієм:

– спочатку система звертається до локальної бази даних (Room);

– при успішному завершенні запиту, запускається паралельний фоновий запит до віддаленої бази даних (Firebase).

Основна робота ведеться з локальним сховищем для забезпечення офлайн доступу та високої швидкодії, а для синхронізації даних між пристроями використовується хмарна СКБД. Локальне сховище реалізовано за допомогою бібліотеки Room Persistence Library, яка є зручною високорівневою абстракцією над SQLite [34]. Структура бази даних включає сутності для збереження словників, категорій, слів та індивідуальних налаштувань користувача.

Між таблицями встановлено зв'язки «один-до-багатьох» із використанням зовнішніх ключів на основі ID. При видаленні словника або категорії спрацьовує правило CASCADE, що автоматично видаляє всі дочірні записи, запобігаючи появі даних без батьків. Програмне створення бази даних виконується у файлі AppDatabase. Реалізація базується на патерні Singleton, що гарантує існування лише одного екземпляра підключення до БД протягом всього життєвого циклу застосунку, щоб запобігти витокам пам'яті та конфліктам доступу.

```
@Database(
  entities = [VocabularyEntity::class, CategoryEntity::class, UserSettingsEntity::class, WordEntity::class],
  version = 5
)
@TypeConverters(Converters::class)
abstract class AppDatabase: RoomDatabase() {
  abstract fun userDao(): UserDao
  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,
          "vocabuler_database"
        )
        .fallbackToDestructiveMigration(true)
        .build()
        INSTANCE = instance
        instance
      }
    }
  }
}
```

Запити до неї реалізують об'єкти доступу до даних (DAO). Вони представляють собою інтерфейси, де визначено SQL-запити для роботи з даними. Анотації @Query дозволяють написати довільний SQL-запит, який Room перевіряє на коректність ще під час компіляції, що запобігає появі помилок. Анотація @Upsert виконує операцію вставки або оновлення запису залежно від того, чи існує він вже у базі, що спрощує логіку синхронізації даних.

```
@Query("SELECT * FROM vocabularies")
suspend fun getAllVocabularies(): List<VocabularyEntity>
@Upsert
suspend fun insertVocabularies(list: List<VocabularyEntity>)
@Query("UPDATE vocabularies SET name = :newName WHERE id = :vocabId")
suspend fun updateVocabularyName(vocabId: String, newName: String)
@Query("DELETE FROM vocabularies WHERE id = :vocabId")
```

Для реалізації хмарної БД використану Firestore, що є NoSQL базою даних документо-орієнтованого типу. Підключення до неї виконується через реєстрацію проєкту в консолі Firebase та додавання конфігураційного файлу google-services.json у Android-проєкт.

Оскільки Firestore має структуру колекцій та документів, роботу з нею винесено у допоміжний об'єкт FirebaseHelper. Він містить загальні запити у форматі «шлях до документа» та «дані у вигляді ключ-значення», що дозволяє спростити звернення до хмарної СКБД та уникнути дублювання коду при роботі з різними колекціями (словники, слова,  тощо). Всі операції виконуються асинхронно з використанням зворотних викликів onSuccess та onFailure.

```
fun addDoc(
    path: String,
    data: Map<String, Any>,
    onSuccess: (String) -> Unit,
    onFailure: (Exception) -> Unit
  ) {
    val docRef = db.collection(path).document()
    val generatedId = docRef.id

    val dataWithId = data.toMutableMap().apply {
      put("id", generatedId)
    }

    docRef.set(dataWithId)
      .addOnSuccessListener { onSuccess(generatedId) }
      .addOnFailureListener { e -> onFailure(e) }
  }

  fun deleteDoc(
    tableName: String,
    docName: String,
    onSuccess: () -> Unit,
    onFailure: (Exception) -> Unit
  ) {
```

```
    db.collection(tableName)
      .document(docName)
      .delete()
      .addOnSuccessListener { onSuccess() }
      .addOnFailureListener { e -> onFailure(e) }
  }

  fun updateDoc(
    path: String,
    docId: String,
    updates: Map<String, Any>,
    onSuccess: () -> Unit,
    onFailure: (Exception) -> Unit
  ) {
    db.collection(path).document(docId)
      .update(updates)
      .addOnSuccessListener { onSuccess() }
      .addOnFailureListener { e -> onFailure(e) }
  }
```

## 3.3 Вимоги до технічних та програмних засобів

Для стабільної роботи та повноцінного функціонування інформаційної системи необхідно дотримуватися визначених технічних вимог. Оскільки система реалізована за клієнт-серверною архітектурою, вимоги було поділено на вимоги до пристрою кінцевого користувача та вимоги до середовища сервера.

Для коректного виконання алгоритмів SRS, обробки локальної бази даних та плавної анімації інтерфейсу, мобільний пристрій користувача повинен відповідати таким мінімальним характеристикам

– процесор 4-ядерний з тактовою частотою від 1.8 ГГц;

– оперативна пам'ять не менше 2 ГБ для швидкої обробки JSON-даних;

– вільне місце в пам'яті від 150 МБ (обсяг збільшується залежно від кількості збережених слів у локальній БД);

– дисплей сенсорний, екран з роздільною здатністю від 720x1280;

– підтримка Wi-Fi або мобільного інтернету стандарту 3G/4G/5G для синхронізації з Firebase та доступу до сервера.

Програмна складова системи в основному полягає у використанні сучасних стандартів Android-розробки. Для роботи застосунку, необхідне наступне системне ПЗ та його мінімальні версії:

– операційна система Android версії 8.0 (Oreo, API level 26) або вище, для використанням бібліотек Jetpack, які потребують сучасної версії Runtime;

– сервіси Google Play, необхідні для коректної роботи автентифікації через Google акаунт та автоматичного оновлення;

– Android System WebView, оновлений до останньої версії для стабільного відображення вбудованого контенту.

Для коректного функціонування озвучення слів ПЗ потребує попередньо встановленої та активованої технології Text-to-Speech (TTS). У налаштуваннях Android для TTS повинен бути завантажений відповідний мовний пакет для цільової мови. За його відсутності система автоматично виводить повідомлення та пропонує користувачеві завантажити необхідний пакет.

Серверна частина потребує середовища Java Runtime Environment (JRE) версії 11 або вище. Сервер Для роботи рекомендується використовувати не менше 512 МБ оперативної пам'яті для JVM-процесу.

## 3.4 Тестування програмного забезпечення

### 3.4.1 Вибір та обґрунтування методів тестування застосунку

Тестування є ключовою частиною життєвого циклу розробки, потрібне для забезпечення якості, надійності та відповідності програмного продукту вимогам. В обраній ітераційній моделі розробки, тестування є критичним етапом підведення підсумків перед наступним циклом. Воно верифікує кожну нову функцію в межах поточної ітерації, запобігаючи накопиченню помилок.

Згідно з міжнародним стандартом ISTQB, процес комплексної перевірки системи включає два ключових аспекти: верифікацію, як перевірку того, чи відповідає об'єкт тестування встановленим вимогам, та валідацію, підтвердження того, що об'єкт тестування відповідає реальним потребам користувачів та інших стейкголдерів у його робочому середовищі [35].

Для комплексного тестування ПЗ було вирішено використати концепт стратегії піраміди тестування, яка є широко розповсюдженою практикою у мобільній розробці. Ця стратегія передбачає наявність великої кількості швидких

низькорівневих (модульних) тестів у основі піраміди та поступово меншої кількості складних тестів на верхніх рівнях, що забезпечує оптимальний баланс між часом тестування та повнотою покриття системи (рисунок 3.3).

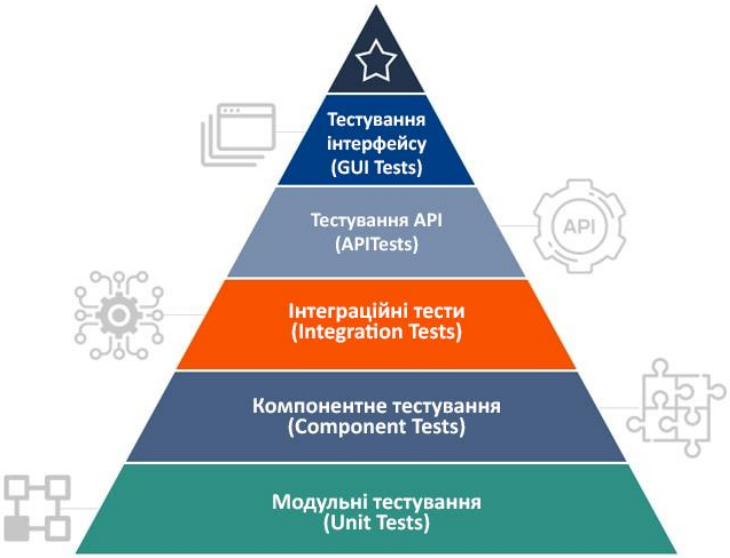



Рисунок 3.3 - Концептуальна модель «Піраміди тестування»

Першим і основним методом перевірки логіки обрано модульне тестування. Воно спрямована на перевірку математичних модулів визначення оцінки та реалізації SRS алгоритму. Для написання модульних тестів обрано фреймворк JUnit 5, який є стандартом для мови Kotlin завдяки його гнучкості та підтримці параметризованих тестів [36].

Оскільки логіка модулей часто залежить від зовнішніх компонентів, для імітації їхньої поведінки використано бібліотеку MockK. Вона дозволяє імітувати поведінку складних об'єктів, щоби ізолювати тест від залежностей, та перевіряти його стан без звернення до реальної бази даних чи мережі [37], що полегшує написання чистих модульних тестів без використання реальної інфраструктури.

Наступним етапом є тестування інтерфейсу, метою якого є імітація реальних сценаріїв взаємодії користувача з екранами застосунку для перевірки коректності відображення UI. Для автоматизації цих перевірок обрано фреймворк Espresso, який є індустріальним стандартом для платформи Android.

Його ключовою перевагою є механізм автоматичної синхронізації тестових операцій із потоком інтерфейсу, що дозволяє уникати проблем зі стабільністю перевірок час виконання анімацій або очікування асинхронних запитів [38].

Завдяки його здатності імітувати дії користувача користувача (натискання, введення тексту, переходи, тощо), Espresso дозволяє верифікувати цілісність навігації, тригери, коректність відображення елементів у списках RecyclerView та логіку взаємодії з діалоговими вікнами.

Для перевірки сумісності з різними пристроями та версіями операційної системи, використано хмарне тестування на реальних пристроях через сервіс Firebase Test Lab. Воно дозволяє запускати автоматизовані Robo-тести на сотнях фізичних пристроїв у дата-центрах Google [39]. Це дозволяє виявити UI-баги, витоки пам’яті та проблеми з продуктивністю на пристроях з низькими характеристиками, які неможливо відтворити на звичайних емуляторах.

Окрему увагу приділено API тестуванню. Для верифікації роботи Ktor-сервера та коректності отримання даних з зовнішніх сервісів, використано інструмент Insomnia. Він дозволяє створювати автоматизовані запити, та виконувати тестування отриманого результату, для перевірки структури JSON-відповідей та кодів статусів HTTP. Це дозволяє перевірити відповідність структури відповіді зі структурою на сервері.

Таким чином, для етапу тестування обрано комбінацію JUnit 5 та MockK для тестування логіки, Espresso для перевірки UI, Firebase Test Lab для апаратної перевірки та Insomnia для API. Такий набір тестів дозволяє покрити всі важливі частини системи перевірками, верифікуючи якість програмного продукту до заданих вимог та сучасних стандартів розробки.

### 3.4.2 Валідація та верифікація програмного забезпечення

Процес перевірки якості полягає у послідовній верифікації та валідації попередньо встановлених функціональних і нефункціональних вимог. Як було визначено в попередньому пункті, для повного охоплення всіх частин системи

було використано багаторівневий підхід стратегії піраміди тестування, що дозволило забезпечити перевірку системи на всіх рівнях.

Першим етапом, стало модульне тестування для головної частини системи, а саме, модулів навчання та алгоритму інтервальних повторень (SRS). Для цього, було підготовлено набір тестів, які охоплюють всі основні сценарії роботи алгоритму, для перевірки розрахунку інтервалів та інших складових, а саме:

– фіксовані значення після першого та другого тестування;
– ліміт значення параметру EaseFactor;
– визначення оцінки на основі факторів тестування;
– провал тестування.

Детальніші параметри розроблених сценаріїв наведено у (таблиці 3.1)

Таблиця 3.1 - Тестові сценарії модульного тестування

| **ID** | **Модуль** | **Опис сценарію** | **Очікуваний результат** |
|---|---|---|---|
| TC-01 | SRSModule | Розрахунок інтервалу при першому показі слова | I = 1 день, N = 1 |
| TC-02 | SRSModule | Розрахунок інтервалу при другому показі слова | I= 6 днів, N = 2 |
| TC-03 | SRSModule | Розрахунок у випадку неправильної відповіді (q < 2) | Скидання N =0 I=1 день |
| TC-04 | SRSModule | Обмеження нижньої межі EaseFactor | EF не повинно опускатися нижче 1.3 |
| TC-05 | GradeCalculator | Обчислення оцінки при ідеальній відповіді (0 помилок, відсутність підказок, миттєве виконання) | Повернення оцінки q = 5 |

Продовження таблиці 3.1

| | | | |
|---|---|---|---|
| TC-06 | GradeCalculator | Обчислення оцінки при правильній відповіді після незначного вагання (без явних помилок) | Повернення оцінки q = 4 |
| TC-07 | GradeCalculator | Обчислення оцінки при виправленні помилки (1 помилка або використання 1 базової підказки) | Повернення оцінки q = 3 |
| TC-08 | GradeCalculator | Обчислення оцінки при значних труднощах (користувач припустився кількох помилок перед вибором) | Повернення оцінки q = 2 |
| TC-09 | GradeCalculator | Обчислення оцінки при критичному рівні помилок (матеріал згадано з величезними труднощами) | Повернення оцінки q = 1 |
| TC-10 | GradeCalculator | Обчислення оцінки при повному провалі тесту (перевищення допустимого ліміту або тайм-аут) | Повернення оцінки q = 0 |

Для реалізації цих тестів, було використано фреймворк JUnit 5 та бібліотеку MockK для ізоляції логіки. Нижче наведено приклад коду модульного тесту, що перевіряє коректність ініціалізації нового слова:

```
@Test
fun `test initial interval calculation`() {
  val word = Word(id = "1", text = "hello")
  val result = srsCalculator.calculateNextReview(word, quality = 5)


  assertEquals(1, result.intervalDays)
  assertTrue(result.easeFactor > 2.5)
}
```

Другий блок сценаріїв був спрямований на детальне тестування інтерфейсу за допомогою фреймворку Espresso. Основну увагу приділено валідації вхідних даних при створенні категорій та слів. Сценарії імітували введення порожніх рядків, надто довгих назв (>50 символів) та перевірку відображення помилок або блокування кнопок у UI при різних ситуаціях.

Окремо було реалізовано сценарій перевірки навігації, що імітує повний шлях користувача від авторизації до успішного завершення навчальної сесії. Це дозволяє перевірити не тільки окремі екрани, але й коректність передачі даних між ними у реальних умовах використання. Нижче наведено код тесту для перевірки відображення вікна при створенні нової категорії:

```
@Test
fun AddCategoryFlow() {
  onView(withId(R.id.categoryNameInput))
    .perform(typeText("New category Test"), closeSoftKeyboard())
  onView(withId(R.id.btnSaveCategory)).perform(click())
  onView(withText("New category Test")).check(matches(isDisplayed()))
}
```

Для перевірки коректності інтеграції із зовнішніми сервісами через API, було розроблено тести в Insomnia, що дозволяє надсилати реальні HTTP-запити та автоматично перевіряти відповіді. Розроблені тести виконують:

– перевірку відповідності структури отриманого JSON-об'єкта;

– пінгування сервісів та окремих запитів Ktor-сервера;

– тестування обробки HTTP-помилок;

Нижче наведено приклад InsomniaTest скрипту що перевіряє отриману структуру на збіг в успішному запиті:

```
const jsonData = JSON.parse(insomnia.response.body);
insomnia.test('Перевірка структури FreeDictionary', () => {
const status = insomnia.response.status;
  insomnia.expect(status.toString()).to.be.oneOf(['200','OK']);
  insomnia.expect(jsonData).to.have.property('word');
  insomnia.expect(jsonData).to.have.property('entries').that.is.an('array');
  insomnia.expect(jsonData).to.have.property('source');
});
```

Останнім етапом стала перевірка роботи ПЗ на реальних пристроях з різними розмірами та версіями ОС Android через Firebase Test Lab. Застосунок було завантажено у форматі apk, відібрано пристрої з різними конфігураціями, та завантажено написаний завчасно скрипт, що симулювало справжню взаємодію типового користувача з системою. Для тестування було написано такі сценарії:

– реєстрація та перше налаштування профілю;

– методи CRUD для словників, категорій та слів;

– змінення користувацьких налаштувань;

– успішне проходження тесту;

– провал проходження тесту.

Самі скрипти написані на Robo Script у форматі JSON-файлу. У скрипті, де кожен крок описує тип події та ID, до якого ця подія застосовується. Нижче наведено приклад скрипту для імітації входу в систем:

```
@Test
[
 {
  "eventType": "VIEW_TEXT_CHANGED",
  "replacementText": "roboTester@gmail.com",
  "elementDescriptors": [
   { "resourceName": "logEmailInput" }
  ]
 },
 {
  "eventType": "VIEW_TEXT_CHANGED",
  "replacementText": "password123",
  "elementDescriptors": [
   { "resourceName": "logPasswordInput" }
  ]
 },
 {
  "eventType": "VIEW_CLICKED",
  "elementDescriptors": [
   { "resourceName": "btnLoginConfrm" }
  ]
 }
]
```

### 3.4.3 Аналіз результатів тестування

Завершальним етапом розробки інформаційної системи став аналіз результатів, отриманих під час верифікації та валідації визначених вимог, щоб оцінити рівень готовності системи до практичного використання.

Першим на розгляді стало модульне тестування як перший рівень піраміди тестування. Для нього було розроблено та описано основні сценарії, які перевіряють правильність роботи математичних формул, коректність обробки бізнес-логіки та точність отримання інформації з репозиторіїв. Тестування зосереджено на двох ключових модулях системи: SRSModule, що відповідає за розрахунок інтервалів повторення, та GradeCalculator, що формує оцінку якості відповіді користувача на основі визначених параметрів.

Всі розроблені тестові випадки для алгоритму SRS та модуля розрахунку оцінки були успішно пройдені без виявлення критичних проблем (рисунок 3.4), що підтверджує математичну точність реалізованих алгоритмів та коректність логіки прийняття рішень у системі навчання.

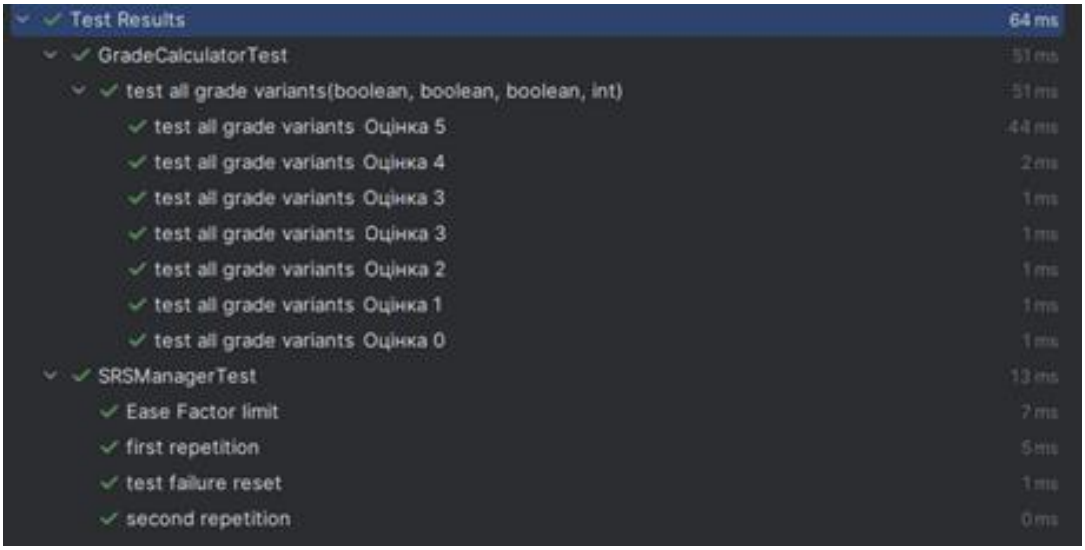



Рисунок 3.4 - Результати модульного тестування

Під час тестування інтерфейсу за допомогою Espresso було підтверджено коректність роботи валідаторів для форм введення, роботи кнопок та реакцій UI на дії користувача (рисунок 3.5). Система визначала та сповіщала про спроби збереження порожніх або некоректних даних, виводячи відповідні підказки та опрацьовувала події і їх реакції. Додаткове тестування основного графу навігації показало відсутність помилок застрягання при поверненні на минулі екрани, циклічності або переходів на пусті екрани.

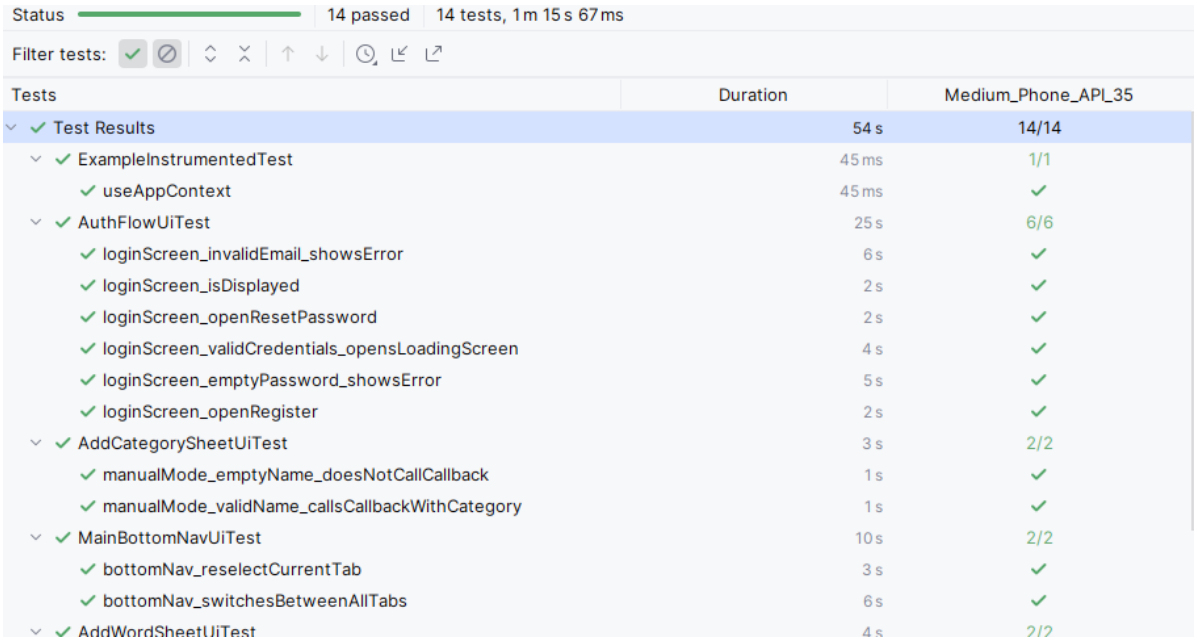



Рисунок 3.5 - Результати тестування UI

Тестування API через Insomnia успішно перевірило взаємодію Ktor-сервера, коректну обробку статусу 404 при відсутності слова у базі та відповідність очікуваної структури відповіді зовнішніх сервісів з фактичною (рисунок 3.6). Всі запити опрацьовувались за 0,1-0,2 мілісекунди та надавали коректний файл відповіді у визначеному форматі. Перевірка обробки помилок підтвердила, що при недоступності API сервер коректно повертає відповідь з детальним описом помилки, а не аварійно завершується. Додатково було успішно перевірено поведінку системи при одночасному надходженні кількох запитів.

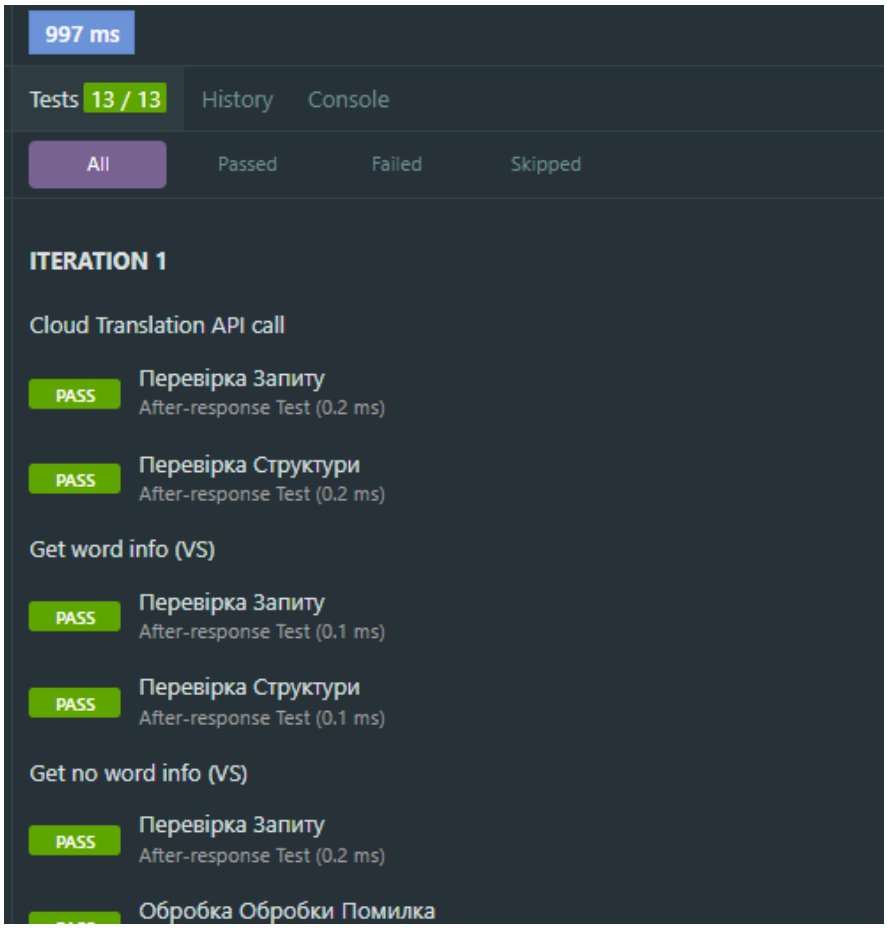



Рисунок 3.6 - Результати тестування API

Для перевірки сумісності було відібрано 3 різних пристрої, які відрізнялись версією ОС Android та розмірами екрану. Результати Robo-тестування не виявили критичних завершень, витоків пам'яті або проблем у виконанні сценаріїв, що підтверджує стабільність роботи системи на різних конфігураціях пристроїв

(рисунок 3.7). Результати показали коректне масштабування інтерфейсу на пристроях з різними розмірами екрана, а графіки споживання пам'яті не виявили аномального зростання під час сесій роботи із застосунком. Це підтверджує ефективність архітектури управління пам'яттю та відсутність проблемних місць що приводять до витоків ресурсів у фонових процесах.

Узагальнюючи результати проведеного тестування, можна стверджувати, що розроблена система успішно пройшла верифікацію та валідацію на всіх рівнях піраміди тестування. Жодних критичних проблем виявлено не було, а базова функціональність повністю відповідає визначеним вимогам.

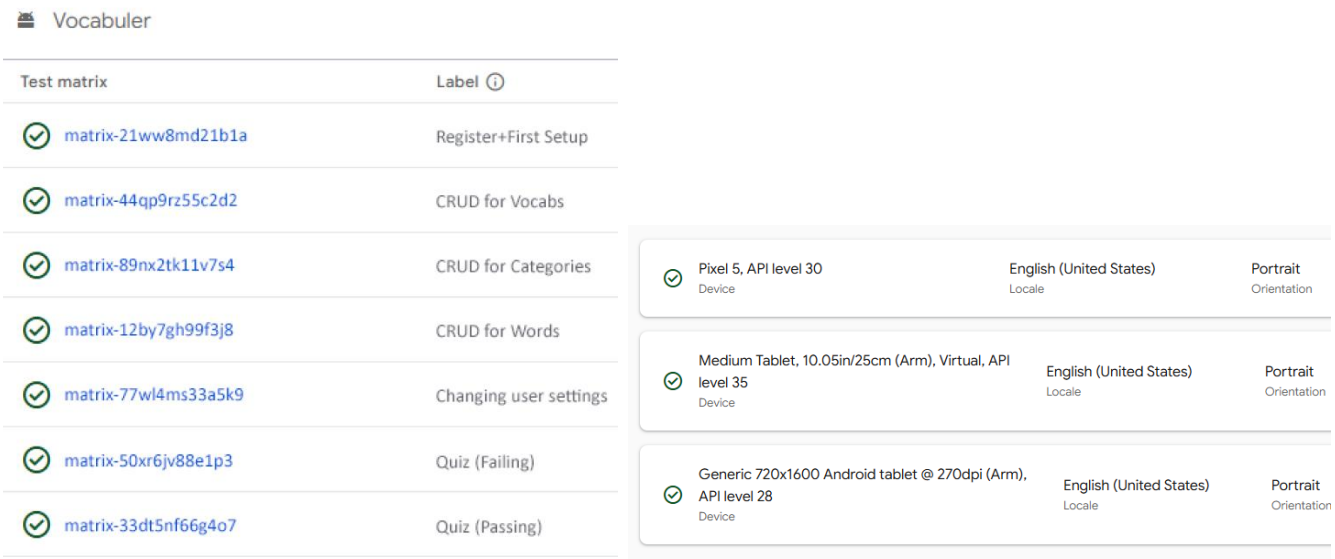



Рисунок 3.7 - Результати тестування Test Lab

## 3.5 Висновки до третього розділу

У результаті виконання третього розділу було здійснено програмну реалізацію та проведено комплексне тестування розробленої інформаційної системи. Процес імплементації базувався на раніше визначених архітектурних рішеннях, патерні MVVM та обраному технологічному стеку.

На етапі реалізації було впроваджено описані модулі, використовуючи мову програмування Kotlin, логіку взаємодії з локальною базою даних Room та

хмарним сховищем Firebase, для впровадження різних режимів роботи застосунку, та їх синхронізації між пристроями. Важливою частиною роботи була інтеграція системи запитів до зовнішніх API-сервісів через Ktor-клієнт за допомогою HTTP-запитів, що виконує функцію автоматичного отримання словникових даних та додаткової інформації.

Завершальним етапом стала верифікація та валідація розробленої системи. Воно відбувалось у декілька етапів, відповідно до піраміди тестування, що охоплює перевірками всі рівні застосунку. Модульні тести на базі JUnit 5 та MockK підтвердили коректність роботи SRS-алгоритму та математичних модулів, автоматизовані UI-тести Espresso забезпечили перевірку сценаріїв взаємодії користувача з екранами, а тестування у Firebase Test Lab підтвердило стабільність роботи ПЗ на різних пристроях з різними версіями Android. API-тестування за допомогою інструменту Insomnia дозволило верифікувати відповідність структур між системами та стійкість до помилок.

Таким чином, розроблений програмний продукт відповідає встановленим функціональним та нефункціональним вимогам, а проведені випробування довели коректність роботи реалізованих алгоритмів, інтегрованих сервісів та готовність застосунку до практичного використання.

# ВИСНОВКИ

Завершивши виконання кваліфікаційної роботи на тему «Інформаційна система для вивчення іноземних мов з використанням зовнішніх API-сервісів», можна зробити такі узагальнені підсумки:

У процесі дослідження обраної галузі визначено тенденцію до цифровізації освіти та високу затребуваність мобільних засобів для самоосвіти. Виявлено, що основними бар'єрами при вивченні іноземних мов є втрата мотивації та низька ефективність механічного запам'ятовування. Як найбільш ефективний науковий метод подолання цих проблем обґрунтовано використання системи інтервальних повторень (SRS) у поєднанні з механіками гейміфікації.

В результаті виконання дослідження предметної області було визначено основні функціональні та нефункціональні вимоги до застосунку, проаналізовано існуючі програмні рішення та виявлено їхні недоліки, такі як складність інтерфейсу або відсутність автоматизації створення слів. На основі проведеного аналізу було побудовано діаграму варіантів використання, що описує межі системи та визначає мету кваліфікаційної роботи.

Після визначення вимог і задач було проведено етап проєктування. Першим важливим кроком став розгляд архітектурних підходів. Для забезпечення масштабованості, тестованості та чистоти коду було обрано поєднання архітектури «Clean Architecture» та архітектурного патерну MVVM. Це дозволило розділити систему на три незалежні шари:

– шар представлення, відповідальний за UI та взаємодію з користувачем;

– доменний шар, що містить бізнес-логіку та сценарії використання;

– шар даних, що безпосередньо забезпечує роботу з локальною БД Room та хмарними сервісами Firebase;

На основі обраного підходу було спроєктовано модулі системи та побудовано UML-діаграми, що візуалізують структуру та поведінку ПЗ:

– діаграма класів що описує структуру основних сутностей системи (Word, Category, Dictionary, User, тощо) та зв’язки між ними;

– діаграма послідовності, що демонструє покроковий процес взаємодії застосунку із різними зовнішніми API-сервісами під час автоматичного отримання перекладу та транскрипції слова;

– діаграма діяльності, яка деталізує внутрішню логіку роботи алгоритму інтервальних повторень та прийняття рішення щодо переведення слова на наступний етап вивчення, або скидання його до початкового стану;

– діаграма станів, що відображає життєвий цикл навчальної слова, починаючи зі стану «Нове» до кінцевого стану «Засвоєне».

Окремим етапом проєктування стала розробка моделі даних. Було спроєктовано схему локальної бази даних та структуру документів у NoSQL хмарній базі Firebase Firestore для забезпечення синхронізації між пристроями.

На основі аналізу ринку мобільної розробки було обрано стек технологій:

– Kotlin як мова програмування, через її лаконічність та безпеку;

– платформа Android SDK з використанням сучасних бібліотек Jetpack;

– Kotlin Coroutines для асинхронності оновлення інтерфейсу;

– Ktor-сервер для роботи із зовнішніми API;

– хмарна платформа Firebase (Auth для авторизації, Firestore для синхронізації даних, TestLab для тестування та Functions для тригерів).

Після завершення проєктування розпочався процес реалізації. Створення модулів стало першим етапом розробки бізнес-логіки. Зокрема, було реалізовано математичний модуль SRS Calculator на основі вдосконаленого алгоритму SM-2, де система автоматично оцінює якість відповіді користувача на основі факторів тестування. Далі було створено репозиторії, що реалізують принцип «Offline-first», де дані спочатку зберігаються в локальній БД Room, а потім асинхронно синхронізуються з хмарою при наявності інтернет-підключення.

Окрему увагу було приділено інтеграції з API-сервісами, що дозволило автоматизувати процес знаходження інформації та додавання слів.

Завершальним етапом розробки була верифікація ПЗ за стратегією «піраміди тестування», використовуючи набір різних тестів, для покриття всіх

аспектів системи (JUnit 5, Espresso, Firebase Test Lab). Результати підтвердили відповідність системи визначеним вимогам, стабільність роботи алгоритмів SRS та коректність відображення інтерфейсу на різних типах пристроїв.

Впровадження розробленого ПЗ дозволить користувачам:

– підвищити ефективність запам'ятовування лексики за рахунок алгоритмічного планування повторень та нагадувань;

– скоротити час на ручне введення та створення слів завдяки автоматичній інтеграції із зовнішніми сервісами та словниками;

– підтримувати регулярність навчання через ігрові стимули.

Розроблене ПЗ може бути використане як допоміжний інструмент для самонавчання, або у сфері навчання. Можливі напрямки вдосконалення включають впровадження розпізнавання мовлення через ШІ для тренування вимови, розвиток бази можливих типів навчання, впровадження нових типів сервісів та реалізацію соціального модуля для спільного вивчення мов.

На основі цього, отримані результати кваліфікаційної роботи підтверджують досягнення поставлених завдань та мети.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ


1. Desktop vs Mobile Market Share Worldwide. StatCounter. URL: https://tinyurl.com/mv4v9u2v (дата звернення: 03.02.2026)

2. Wroblewski L. Mobile First. New York: A Book Apart, 2011. 138 с.

3. 5-Year Market Forecast: App Spending Will Climb to $270 Billion by 2025. Sensor Tower. URL: https://tinyurl.com/ywze9z67 (дата звернення: 08.02.2026)

4. Most popular Google Play categories (February 2026). AppBrain. URL: https://tinyurl.com/mpumada8 (дата звернення: 08.02.2026)

5. Education App Report 2025: Comprehensive insights into the education app industry. Business of Apps. URL: https://tinyurl.com/3be43zd8 (дата звернення: 09.02.2026)

6. Learning and Learners' Inability to Learn a Foreign Language. ResearchGate. URL: https://tinyurl.com/3fdr624m (дата звернення: 09.02.2026)

7. Застосування новітніх технологій під час реалізації прийому інтервального повторення англійської лексики у ВНЗ. URL: https://tinyurl.com/5ej7au9s (дата звернення: 09.02.2026)

8. Replication and Analysis of Ebbinghaus Forgetting Curve. PMC. URL: https://tinyurl.com/4d85fumd (дата звернення: 12.02.2026).

9. Hamari J., Koivisto J., Sarsa H. Does Gamification Work? A Literature Review of Empirical Studies on Gamification. 2014. 10 с. URL: https://tinyurl.com/2s4bh3tw (дата звернення: 12.02.2026).

10. The Use of Persona in Foreign Language Learning Facilitated by Chatbots. ResearchGate. URL: https://tinyurl.com/v79hm4pj (дата звернення: 12.02.2026)

11. Duolingo documentation. URL: https://tinyurl.com/4n82u8p3 (дата звернення: 13.02.2026)

12. Anki documentation. URL: https://tinyurl.com/3bzpe62p (дата звернення: 13.02.2026)

13. What spaced repetition algorithm does Anki use? Anki FAQs. URL: https://tinyurl.com/5n8jnhw9 (дата звернення: 13.02.2026)

14. Memrise documentation. URL: https://tinyurl.com/3833e9vt (дата звернення: 13.02.2026)

15. Core app quality guidelines. Android Developers. URL: https://tinyurl.com/2azp5249 (дата звернення: 20.02.2026)

16. Use Case Diagram - Unified Modeling Language (UML). GeeksforGeeks. URL: https://tinyurl.com/3s8ktmnt (дата звернення: 20.03.2026).

17. Ibraimi A., Memeti A., Idrizi F. MVC VS MVVM VS MVP: WHICH ARCHITECTURAL PATTERN SUITS MODERN APPLICATIONS BEST? Journal of Natural Sciences and Mathematics of UT. 2025. Vol. 10, No. 19-20. P. 306-311.

18. Занадто великий клас (Large Class / God Object). RefactoringGuru. URL: https://tinyurl.com/373yttns (дата звернення: 01.03.2026).

19. App architecture: Data layer - The Repository pattern. Android Developers. URL: https://tinyurl.com/3jc6eb55 (дата звернення: 12.03.2026).

20. Породжувальні патерни: Фабричний метод (Factory Method). RefactoringGuru. URL: https://tinyurl.com/nh38jpnv (дата звернення: 14.03.2026).

21. Application of a computer to improve the results obtained in working with the SuperMemo method. SuperMemo. URL: https://tinyurl.com/frv7z6jy (дата звернення: 14.03.2026).

22. SuperMemo: Details of SM-2 algorithm. SuperMemo. URL: https://tinyurl.com/5t3k8hv2 (дата звернення: 18.03.2026).

23. Chou Yu-kai. Actionable Gamification: Beyond Points, Badges, and Leaderboards. Milpitas: Octalysis Media, 2015. 501 с.

24. What is Entity Relationship Diagram (ERD). Visual Paradigm. URL: https://tinyurl.com/7vx8txxe (дата звернення: 19.03.2026).

25. Android's Kotlin-first approach. Android Developers. URL: https://tinyurl.com/yc4st667 (дата звернення: 19.03.2026)

26. Kotlin Coroutines Guide. Kotlin Documentation. URL: https://tinyurl.com/4tct33a9 (дата звернення: 22.03.2026).

27. Thinking in Compose. Android Developers. URL: https://tinyurl.com/yb2uekpf (дата звернення: 24.03.2026)

28. Comparative Analysis of Software Development Lifecycle Methods in Software Development: A Systematic Literature Review. ResearchGate. URL: https://tinyurl.com/mr2fftmc (дата звернення: 25.03.2026)

29. Ktor: Build asynchronous servers and clients in Kotlin. JetBrains. URL: https://ktor.io/docs/ (дата звернення: 01.04.2026).

30. Martin R. C. The Clean Architecture. The Clean Code Blog. URL: https://tinyurl.com/bddm7mez (дата звернення: 01.04.2026).

31. Principles of navigation: Single-activity architecture. Android Developers. URL: https://tinyurl.com/2ps5h247 (дата звернення: 02.04.2026)

32. Principles of Android navigation. Android Developers. URL: https://tinyurl.com/mry53j43 (дата звернення: 02.04.2026).

33. Navigation Safe Args. Android Jetpack Documentation. URL: https://tinyurl.com/m775yxxn (дата звернення: 08.04.2026).

34. Save data in a local database using Room. Android Developers. URL: https://tinyurl.com/4xye8hav (дата звернення: 15.04.2026).

35. Certified Tester Foundation Level (CTFL) Syllabus v.4.0.1 / International Software Testing Qualifications Board (ISTQB). 2024. 101 p.URL: https://tinyurl.com/yf9vh4tr (дата звернення: 24.04.2026)

36. JUnit 5 User Guide. JUnit Team. URL: https://tinyurl.com/4wnka9ez (дата звернення: 24.04.2026).

37. MockK: library for mocking in Kotlin. MockK.io. URL: https://mockk.io/ (дата звернення: 24.04.2026).

38. Espresso basics. Android Developers. URL: https://tinyurl.com/35xthzkw (дата звернення: 24.04.2026).

39. Firebase Test Lab documentation. Google Firebase. URL: https://tinyurl.com/yc89a63c (дата звернення: 24.04.2026).

40. Методичні настанови для здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 121 «Інженерія програмного забезпечення» / Л. П. Бедратюк, Г. І. Радельчук. Хмельницький: ХНУ, 2023. 60 с

ДОДАТОК А

(обов’язковий)

# ГРАФІЧНІ МАТЕРІАЛИ

| | Duolingo | AnkiDroid Flashcards | Memrise: Languages for Life | Quizlet | Drops | DuoCards | Reverso | Mondly |
|---|---|---|---|---|---|---|---|---|
| Алгоритм SRS | Не повністю | Присутня | Присутня | Присутня | Присутня | Присутня | Присутня | Присутня |
| Можливість додавання власних слів | Відсутня | Присутня | Присутня | Присутня | Відсутня | Присутня | Присутня | Відсутня |
| Гейміфікація | Присутня | Відсутня | Присутня | Не повністю | Присутня | Не повністю | Відсутня | Присутня |
| Персонаж-маскот | Присутня | Відсутня | Відсутня | Відсутня | Відсутня | Присутня | Відсутня | Відсутня |
| Офлайн режим | Не повністю | Присутня | Присутня | Присутня | Присутня | Присутня | Відсутня | Присутня |
| Розпізнавання мовлення | Присутня | Відсутня | Відсутня | Відсутня | Відсутня | Відсутня | Відсутня | Присутня |
| Ілюстрації | Присутня | Присутня | Присутня | Присутня | Присутня | Присутня | Відсутня | Присутня |
| Фокус на мікронавчанні (<5 хв) | Присутня | Відсутня | Присутня | Присутня | Присутня | Присутня | Присутня | Присутня |
| Створення власних колод/словників | Відсутня | Присутня | Присутня | Присутня | Відсутня | Присутня | Присутня | Відсутня |
| Вбудований перекладач | Відсутня | Відсутня | Відсутня | Відсутня | Відсутня | Присутня | Присутня | Відсутня |
| Соціальні функції | Присутня | Відсутня | Присутня | Присутня | Відсутня | Відсутня | Відсутня | Присутня |
| Тестування рівня знань | Присутня | Відсутня | Відсутня | Відсутня | Відсутня | Відсутня | Відсутня | Присутня |
| Навчання в контексті | Присутня | Не повністю | Присутня | Відсутня | Відсутня | Відсутня | Присутня | Присутня |
| Статистика та аналітика прогресу | Присутня | Присутня | Присутня | Присутня | Присутня | Присутня | Присутня | Присутня |
| Флеш-картки | Присутня | Присутня | Присутня | Присутня | Присутня | Присутня | Присутня | Присутня |
| Штучний інтелект | Присутня | Відсутня | Присутня | Присутня | Відсутня | Відсутня | Присутня | Присутня |
| Ударний режим (Streak System) | Присутня | Не повністю | Присутня | Присутня | Відсутня | Відсутня | Присутня | Присутня |
| Щоденні завдання | Присутня | Відсутня | Присутня | Відсутня | Присутня | Присутня | Відсутня | Присутня |
| Флеш-картки | Присутня | Відсутня | Присутня | Присутня | Присутня | Відсутня | Відсутня | Присутня |
| Нагороди та ачівки | Присутня | Відсутня | Присутня | Присутня | Присутня | Присутня | Присутня | Присутня |
| Тести та завдання | Присутня | Відсутня | Присутня | Присутня | Присутня | Присутня | Присутня | Присутня |
| Ігрові завдання | Присутня | Відсутня | Присутня | Присутня | Присутня | Відсутня | Відсутня | Присутня |
| Слово дня | Відсутня | Відсутня | Відсутня | Відсутня | Присутня | Відсутня | Відсутня | Присутня |
| Щоденний тест на повторення | Присутня | Присутня | Присутня | Відсутня | Присутня | Присутня | Присутня | Присутня |
| Сповіщення/Нагадування | Присутня | Присутня | Присутня | Присутня | Присутня | Присутня | Присутня | Присутня |
| Хмарна синхронізація | Присутня | Присутня | Присутня | Присутня | Присутня | Присутня | Присутня | Присутня |
| Імпорт слів з файлів (CSV, TXT, JSON) | Відсутня | Присутня | Не повністю | Присутня | Відсутня | Присутня | Відсутня | Відсутня |
| Поширення матеріалів між користувачами | Відсутня | Присутня | Присутня | Присутня | Відсутня | Присутня | Відсутня | Відсутня |

Рисунок А.1 - Таблиця аналізу конкурентів

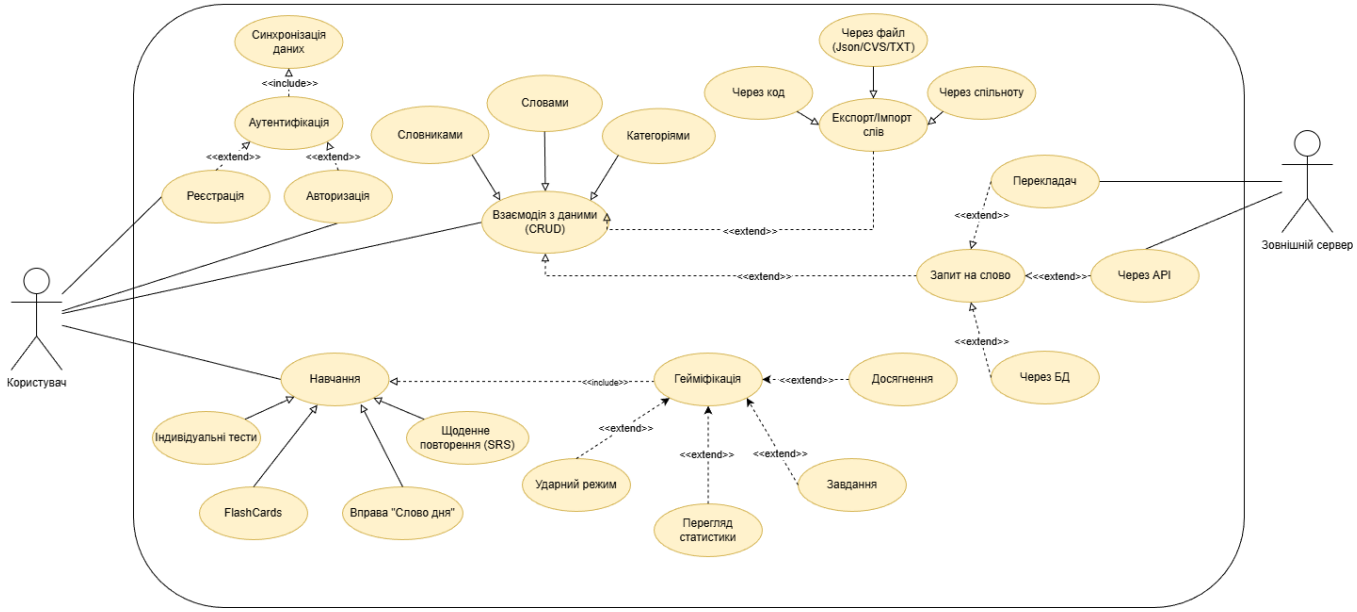



Рисунок А.2 - Діаграма використання системи

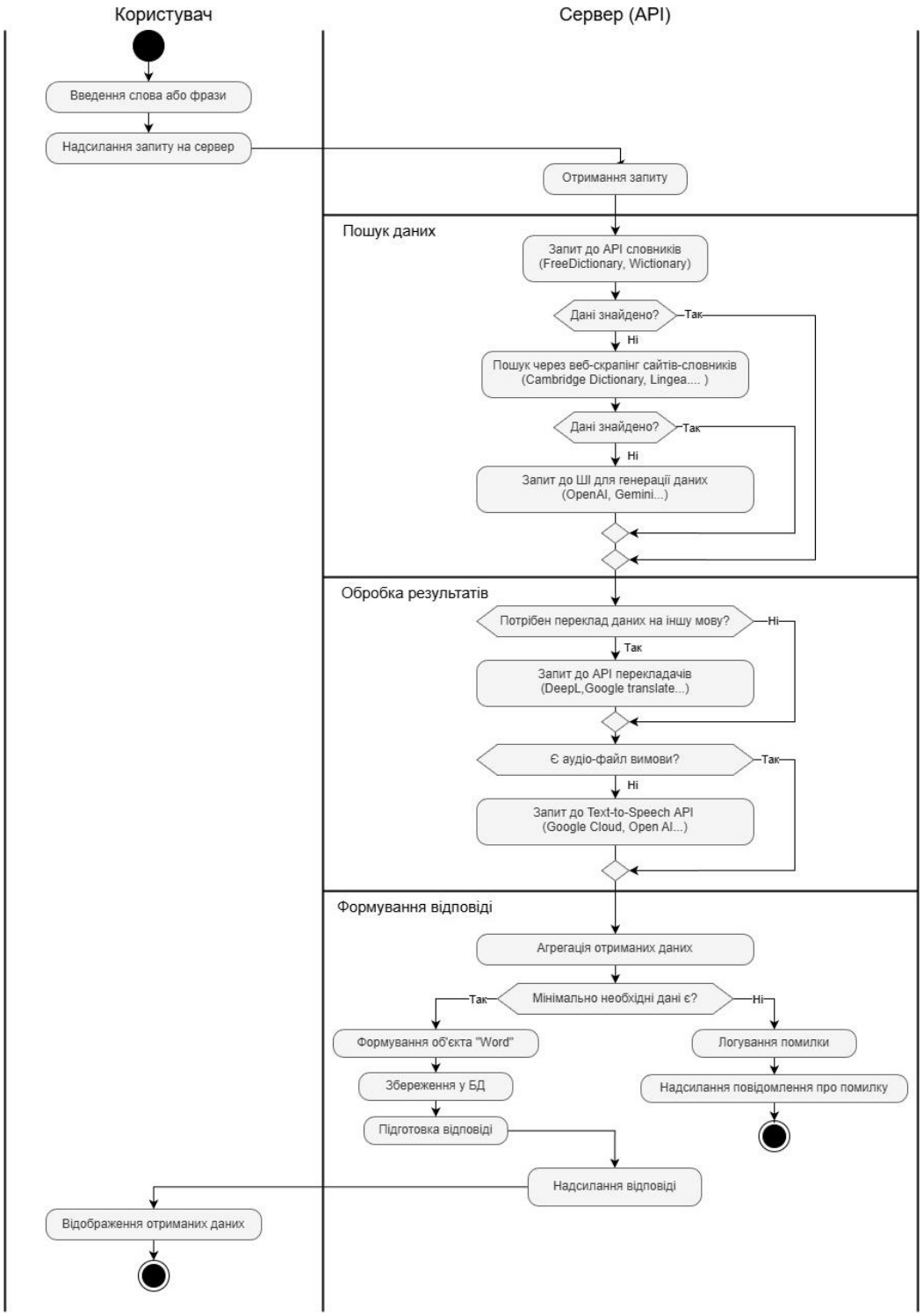



Рисунок А.3 - Діаграма станів процесу запиту інформації про слово

# ДОДАТОК Б

(обов'язковий)

# ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ

## Б.1 Код AddWordSheet.kt

```
class AddWordSheet(
  private val selectedDictionary: String,
  private val selectedCategory: String,
  private val mode: String,
  private val word: Word? = null,
  private val onWordAddingConfirmation: (Word) -> Unit
): BottomSheetDialogFragment() {

  private lateinit var binding: FragmentAddWordSheetBinding
  private var selectedFileUri: Uri? = null
  private val path = FHelper.getPath(true, selectedDictionary, selectedCategory)



  private val filePickerLauncher = registerForActivityResult(ActivityResultContracts.GetContent()) { uri: Uri?
->
    uri?.let {
      selectedFileUri = it
      binding.filePath.setText(it.toString())
    }
  }

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    parentFragmentManager.setFragmentResultListener("subscription_prompt_result", this) { _, bundle ->
      if (bundle.getString("action") == "upgrade") {
        dismiss()
        findNavController().navigate(R.id.action_global_subscriptionFragment)
      }
    }
  }

  override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View
{
    binding = FragmentAddWordSheetBinding.inflate(inflater, container, false)

    val currentVocab = VocabulariesList.list.find { it.languageCode == selectedDictionary }

    if (currentVocab != null) {
      setupTagsSpinner(currentVocab)
    } else {
      setupCustomTagsOnly()
    }

    setupClickListeners()
    return binding.root
  }

  override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    dialog?.setOnShowListener { dialogInterface ->
      val bottomSheetDialog = dialogInterface as BottomSheetDialog
      val                                          bottomSheet                                                =
bottomSheetDialog.findViewById<View>(com.google.android.material.R.id.design_bottom_sheet)
      bottomSheet?.let {
        val behavior = BottomSheetBehavior.from(it)
        behavior.state = BottomSheetBehavior.STATE_EXPANDED
        behavior.skipCollapsed = true
        it.layoutParams.height = ViewGroup.LayoutParams.MATCH_PARENT
```

```
    }
  }
  setupLayoutsByMode(mode)
}

private fun setupClickListeners() {
  binding.btnUploadFile.setOnClickListener { filePickerLauncher.launch("text/plain") }

  binding.btnConfirmManual.setOnClickListener {
    val wordText = binding.wordName.text.toString().trim()
    val translation = binding.wordTranslation.text.toString().trim()

    if (wordText.isEmpty() || translation.isEmpty()) {
      Toast.makeText(requireContext(), "Введіть текст", Toast.LENGTH_SHORT).show()
      return@setOnClickListener
    }


    saveWord(wordText, translation)
  }
}

private fun setupTagsSpinner(vocabulary: LanguagePreset) {
  val allTags = vocabulary.rules.categories.flatMap { it.tags }
  val adapterItems = allTags.map { getTagName(it) }

  val adapter = ArrayAdapter(
    requireContext(),
    android.R.layout.simple_dropdown_item_1line,
    adapterItems
  )
  binding.categoryAutoComplete.setAdapter(adapter)

  binding.categoryAutoComplete.setOnItemClickListener { parent, _, position, _ ->
    val name = parent.getItemAtPosition(position) as String
    val tag = allTags.find { getTagName(it) == name }
    tag?.let { addTagChip(it, name, vocabulary) }
    binding.categoryAutoComplete.setText("", false)
  }

  binding.categoryAutoComplete.setOnEditorActionListener { v, _, _ ->
    val text = v.text.toString().trim()
    if (text.isNotEmpty()) {
      val existingTag = allTags.find { getTagName(it).equals(text, ignoreCase = true) }
      if (existingTag != null) {
        addTagChip(existingTag, getTagName(existingTag), vocabulary)
      } else {
        addTagChip(CustomTag(text, text), text, vocabulary)
      }
      binding.categoryAutoComplete.setText("", false)
    }
    true
  }
}

private fun setupCustomTagsOnly() {
  binding.categoryAutoComplete.setOnEditorActionListener { v, _, _ ->
    val text = v.text.toString().trim()
    if (text.isNotEmpty()) {
      addTagChip(CustomTag(text, text), text, null)
      binding.categoryAutoComplete.setText("", false)
    }
    true
  }
}

private fun getTagName(tag: GrammarTag): String {
  return tag.customName ?: tag.stringResId?.let { getString(it) } ?: tag.key
}

private fun addTagChip(tag: GrammarTag, name: String, vocabulary: LanguagePreset?) {
  vocabulary?.rules?.categories?.find { cat -> cat.tags.any { it.key == tag.key } }?.let { category ->
    if (category.singleValue) {
      val categoryKeys = category.tags.map { it.key }
      for (i in 0 until binding.tagsChipGroup.childCount) {
```

```
          val chip = binding.tagsChipGroup.getChildAt(i) as? Chip
          if (categoryKeys.contains(chip?.tag as? String)) {
            binding.tagsChipGroup.removeView(chip)
            break
          }
        }
      }
    }

    if (binding.tagsChipGroup.findViewWithTag<Chip>(tag.key) == null) {
      val                                                chip                                                                =
LayoutInflater.from(requireContext()).inflate(R.layout.item_chip_edit,binding.tagsChipGroup,false) as Chip
      chip.apply {
        text = name
        this.tag = tag.key
        setOnCloseIconClickListener { binding.tagsChipGroup.removeView(this) }
      }
      binding.tagsChipGroup.addView(chip)
    }
  }

  private fun saveWord(wordText: String, translation: String) {
    val selectedTags = mutableMapOf<String, String>()
    for (i in 0 until binding.tagsChipGroup.childCount) {
      val chip = binding.tagsChipGroup.getChildAt(i) as Chip
      selectedTags[chip.tag as String] = chip.text.toString()
    }

    val newWord = Word(
      word = wordText,
      translation = translation,
      language = selectedDictionary,
      category = selectedCategory,
      tags = selectedTags,
      id = word?.id ?: ""
    )

    if (binding.apiCheckbox.isChecked) {
      addWordFromApi(wordText, translation)
    } else{
      onWordAddingConfirmation(newWord)
      dismiss()
    }
  }

  private fun addWordFromApi(wordText: String, userTranslation: String) {
    var selectedPartOfSpeech: String = "noun"
    for (i in 0 until binding.tagsChipGroup.childCount) {
      val chip = binding.tagsChipGroup.getChildAt(i) as Chip
      if (chip.tag == "partOfSpeech") {
        selectedPartOfSpeech = chip.text.toString()
      }
    }

    lifecycleScope.launch {
      try {

        val response = ServerManager.getTranslation(
          lang = selectedDictionary,
          word = wordText,
          pos = selectedPartOfSpeech,
          translation = UserSession.settings.languageSelected
        )

        if (response != null) {
          response.translation = userTranslation.ifEmpty { response.translation }
          response.category = selectedCategory


          onWordAddingConfirmation(response)
          dismiss()
        } else {
          Toast.makeText(requireContext(), "API error", Toast.LENGTH_SHORT).show()
        }
      } catch (e: Exception) {
```

```
        Log.e("API", e.message.toString())
      }
    }
  }


  private fun setupLayoutsByMode(mode: String) {
    binding.layoutManual.visibility = if (mode == "manual" || mode == "edit") View.VISIBLE else View.GONE
    binding.layoutFile.visibility = if (mode == "file") View.VISIBLE else View.GONE
    binding.layoutLink.visibility = if (mode == "link") View.VISIBLE else View.GONE

    if (mode == "edit" && word != null) {
      binding.wordName.setText(word.word)
      binding.wordTranslation.setText(word.translation)
      binding.apiSearchLayout.visibility = View.GONE
    }
  }

  private fun isPartOfSpeechSelected(): Boolean {
    for (i in 0 until binding.tagsChipGroup.childCount) {
      val chip = binding.tagsChipGroup.getChildAt(i) as? Chip
      val tagKey = chip?.tag as? String


      if (tagKey == "partOfSpeech") {
        return true
      }
    }
    return false
  }
  override fun getTheme(): Int = R.style.AppBottomSheetDialogTheme
}
```

## Б.2 Код LoadingScreen.kt

```
class LoadingScreen: AppCompatActivity() {
  private lateinit var binding: ActivityLoadingScreenBinding
  private lateinit var viewModel: LoadingScreenViewModel

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    binding = ActivityLoadingScreenBinding.inflate(layoutInflater)
    setContentView(binding.root)

    val app = application as App
    val factory = object: ViewModelProvider.Factory {
      override fun <T: ViewModel> create(modelClass: Class<T>): T {
        return LoadingScreenViewModel(app.userRepository) as T
      }
    }

    viewModel = ViewModelProvider(this, factory)[LoadingScreenViewModel::class.java]
    binding.btnRetry.setOnClickListener {
      val uid = FirebaseAuth.getInstance().currentUser?.uid
      if (uid != null) viewModel.prepareAppData(uid)
    }

    checkAuth()
    observeViewModel()


  }

  private fun observeViewModel() {
    lifecycleScope.launchWhenStarted {
      viewModel.loadingState.collect { state ->
        when (state) {
          is LoadingScreenViewModel.LoadingResult.Success -> navigateToMain()
          is LoadingScreenViewModel.LoadingResult.NeedsSetup -> navigateToSetup()
          is LoadingScreenViewModel.LoadingResult.AuthError -> navigateToAuth()
          is LoadingScreenViewModel.LoadingResult.Error -> {
            with(binding) {
```

```
                progressBar.visibility= View.GONE
                loadingText.visibility= View.GONE

                btnRetry.visibility = View.VISIBLE
                loadingHint.text = "Error: ${state.message}"
              }
            }

            else -> {}
          }
        }
      }
    }

    private fun checkAuth() {
      val uid = FirebaseAuth.getInstance().currentUser?.uid
      if (uid != null)
        viewModel.prepareAppData(uid)
      else
        navigateToAuth()
    }

    private fun navigateToMain() {
      startActivity(Intent(this, MainActivity::class.java))
      finish()
    }

    private fun navigateToSetup() {
      val intent = Intent(this, AuthActivity::class.java)
      intent.putExtra("startStep", "setup")
      startActivity(intent)
      finish()
    }

    private fun navigateToAuth() {
      startActivity(Intent(this, AuthActivity::class.java))
      finish()
    }
}
```

## Б.3 Код UserDao.kt

```
@Dao
interface UserDao {


    //====Vocabulary====

    //-Get-
    @Query("SELECT * FROM vocabularies")
    suspend fun getAllVocabularies(): List<VocabularyEntity>

    //-Set-
    @Upsert
    suspend fun insertVocabularies(list: List<VocabularyEntity>)

    //-Update-
    @Query("UPDATE vocabularies SET name = :newName WHERE id = :vocabId")
    suspend fun updateVocabularyName(vocabId: String, newName: String)

    //-Delete-
    @Query("DELETE FROM vocabularies WHERE id = :vocabId")
    suspend fun deleteVocabularyById(vocabId: String)

    @Query("DELETE FROM vocabularies")
    suspend fun clearAll()


    //====Category====

    //-Get
    @Query("SELECT * FROM categories WHERE parentVocabId = :vocabId")
    suspend fun getCategoriesByID(vocabId: String): List<CategoryEntity>
```

```
  //-Delete-
  @Query("DELETE FROM categories WHERE id = :categoryId")
  suspend fun deleteCategoryById(categoryId: String)
  //-Set
  @Upsert
  suspend fun insertCategories(list: List<CategoryEntity>)

  //Update

  @Query("UPDATE categories SET wordsCount = :total, learnedCount = :learned WHERE id = :catId")
  suspend fun updateCategoryMetadata(catId: String, total: Int, learned: Int)

  @Query("UPDATE categories SET name = :value WHERE id = :catId AND :field = 'name'")
  suspend fun updateCategoryName(catId: String, field: String, value: String)


  //====Word====

  //-Get
  @Query("SELECT * FROM words WHERE categoryId = :categoryId")
  suspend fun getWordsByCategory(categoryId: String): List<WordEntity>

  //-Set
  @Upsert
  suspend fun insertWords(words: List<WordEntity>)

  //-Delete
  @Delete
  suspend fun deleteWord(word: WordEntity)

  // -Update
  @Query("UPDATE words SET id = :newId WHERE id = :oldId")
  suspend fun updateWordId(oldId: String, newId: String)

  @Query("UPDATE words SET isFavorite = :isFav WHERE id = :wordId")
  suspend fun updateWordFavorite(wordId: String, isFav: Boolean)

  @Query("UPDATE words SET learned = :newStage WHERE id = :wordId")
  suspend fun updateWordStage(wordId: String, newStage: String)

}
```

## Б.4 Код QuizViewModel.kt

```
class QuizViewModel: ViewModel() {

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

  private val _wordList = MutableLiveData<List<Word>>()
  val wordList: LiveData<List<Word>> get() = _wordList

  private val _quizList = MutableLiveData<List<Question>>()
  val quizList: LiveData<List<Question>> get() = _quizList

  private var currentIndex = 0

  fun loadWords(words: List<Word>) {
    _wordList.value = words
    val questions = generateQuestions(words)
    _quizList.value = questions
    Log.d(TAG, "Loaded ${questions.size} questions")
  }

  private fun generateQuestions(words: List<Word>): List<Question> {
    return words.mapNotNull { word ->
      val availableTypes = questionTypeByLevel[word.difficulty]
        ?.filter { type ->
          type == QuestionType.MULTIPLE_CHOICE ||
              type == QuestionType.AUDIO_CHOICE ||
              type == QuestionType.AUDIO_ANSWER
        }
```

```
      ?.filter { type ->
        if (type == QuestionType.AUDIO_CHOICE || type == QuestionType.AUDIO_ANSWER)
          word.audio.isNotBlank()
        else true
      } ?: emptyList()

    if (availableTypes.isEmpty()) {
      return@mapNotNull null
    }

    val type = availableTypes.random()

    val target = if (word.difficulty == DifficultyLevel.HARD)
      TargetPart.WORD else TargetPart.entries.random()

    val options = if (word.difficulty == DifficultyLevel.EASY)
      getRandomOptions(target, word) else emptyList()

    val correctAnswer = if (target == TargetPart.WORD)
      word.word else word.translation

    Question(
      id = word.id,
      word = word,
      target = target,
      type = type,
      options = options,
      correctAnswer = correctAnswer,
      userAnswer = null,
      isCorrect = false,
      answered = false
    )
  }
}

fun setUserAnswer(question: Question, answer: String) {
  val updated = _quizList.value?.map {
    if (it.id == question.id) it.copy(userAnswer = answer) else it
  } ?: emptyList()
  _quizList.value = updated
  Log.d(TAG, "User selected answer: $answer for ${question.word.word}")
}

fun checkAnswer(question: Question): Boolean {
  val isCorrect = question.userAnswer == question.correctAnswer
  val updated = _quizList.value?.map {
    if (it.id == question.id) it.copy(isCorrect = isCorrect, answered = true) else it
  } ?: emptyList()
  _quizList.value = updated
  return isCorrect
}

fun nextQuestion(): Question? {
  val list = _quizList.value ?: return null
  if (currentIndex + 1 < list.size) {
    currentIndex++
    return list[currentIndex]
  }
  return null
}

fun getCurrentQuestion(): Question? {
  return _quizList.value?.getOrNull(currentIndex)
}

fun getProgress(): Float {
  val total = _quizList.value?.size ?: return 0f
  val answered = _quizList.value?.count { it.answered } ?: 0
  return if (total == 0) 0f else (answered.toFloat() / total.toFloat()) * 100
}

private fun getRandomOptions(target: TargetPart, correctWord: Word, count: Int = 4): List<String> {
  val words = _wordList.value ?: emptyList()
  val otherOptions = words.filter { it.id != correctWord.id }
```

```
    val randomOptions = when (target) {
      TargetPart.WORD -> otherOptions.map { it.word }.shuffled().take(count - 1)
      TargetPart.TRANSLATION -> otherOptions.map { it.translation }.shuffled().take(count - 1)
    }

    val correctAnswer = if (target == TargetPart.WORD) correctWord.word else correctWord.translation
    return (randomOptions + correctAnswer).shuffled()
  }
}
```

## Б.5 Код WordMapper.kt

```
object WordMapper {
  private val gson = Gson()

  fun WordFromMap(data: Map<String, Any>): Word {
    val grammarMap = (data["tags"] as? Map<String, String>)?.toMutableMap() ?: mutableMapOf()

    return Word(
      id = data["id"] as? String ?: java.util.UUID.randomUUID().toString(),
      word = data["word"] as? String ?: "",
      translation = data["translation"] as? String ?: "",
      exampleSentence = data["exampleSentence"] as? String ?: "",

      tags = grammarMap,
      category = data["category"] as? String ?: "",
      language = data["language"] as? String ?: "en",

      learned = try { Stage.valueOf(data["learned"] as String) }
      catch (e: Exception) { Stage.NEW },

      difficulty = try { DifficultyLevel.valueOf(data["difficulty"] as String) }
      catch (e: Exception) { DifficultyLevel.EASY },

      timesReviewed = (data["timesReviewed"] as? Long)?.toInt() ?: 0,
      easeFactor = (data["easeFactor"] as? Number)?.toDouble() ?: 2.5,
      intervalDays = (data["intervalDays"] as? Long)?.toInt() ?: 1,
      lastReviewed = data["lastReviewed"] as? Long,
      addedDate = data["addedDate"] as? Long ?: System.currentTimeMillis(),

      synonyms = RelatedWordfromMap(data["synonyms"]),
      antonyms = RelatedWordfromMap(data["antonyms"]),
      meanings = RelatedWordfromMap(data["meanings"]),

      isBookmarked = data["isBookmarked"] as? Boolean ?: false
    ).apply {
    }
  }

  private fun RelatedWordfromMap(rawList: Any?): MutableList<RelatedWord> {
    val list = rawList as? List<Map<String, Any>> ?: return mutableListOf()
    return list.map { item ->
      RelatedWord(
        id = item["id"] as? String ?: "",
        word = item["word"] as? String ?: "",
        exampleSentence = item["exampleSentence"] as? String ?: "",
        definition = item["definition"] as? String ?: ""
      )
    }.toMutableList()
  }

  fun Word.toMap(): Map<String, Any?> {
    return mapOf(
      "id" to id,
      "word" to word,
      "translation" to translation,
      "category" to category,
      "exampleSentence" to exampleSentence,
      "language" to language,
      "learned" to learned.name,
      "difficulty" to difficulty.name,
      "timesReviewed" to timesReviewed,
      "easeFactor" to easeFactor,
      "intervalDays" to intervalDays,
```

```
        "lastReviewed" to lastReviewed,
        "addedDate" to addedDate,
        "isBookmarked" to isBookmarked,
        "tags" to tags,
        "synonyms" to synonyms.map { it.toMap() },
        "antonyms" to antonyms.map { it.toMap() },
        "meanings" to meanings.map { it.toMap() }
      )
    }

    fun RelatedWord.toMap(): Map<String, Any> {
      return mapOf(
        "id" to id,
        "word" to word,
        "exampleSentence" to exampleSentence,
        "definition" to definition
      )
    }

    fun entityToDomain(entity: WordEntity): Word {
      val synonymsType = object: TypeToken<MutableList<RelatedWord>>() {}.type

      return Word(
        id = entity.id,
        word = entity.word,
        translation = entity.translation,
        language = entity.language,
        learned = try { Stage.valueOf(entity.learned) } catch (e: Exception) { Stage.NEW },
        difficulty = DifficultyLevel.EASY,

        synonyms = gson.fromJson(entity.synonymsJson, synonymsType) ?: mutableListOf(),
        antonyms = gson.fromJson(entity.antonymsJson, synonymsType) ?: mutableListOf(),
        meanings = gson.fromJson(entity.meaningsJson, synonymsType) ?: mutableListOf(),

        isBookmarked = entity.isFavorite,
        addedDate = entity.lastImported
      )
    }

    fun domainToEntity(word: Word, categoryId: String): WordEntity {
      return WordEntity(
        id = word.id,
        word = word.word,
        translation = word.translation,
        transcription = word.transcription,
        example = word.exampleSentence,
        learned = word.learned.name,
        isFavorite = word.isBookmarked,
        lastModified = System.currentTimeMillis(),
        lastImported = word.addedDate,

        categoryId = categoryId,
        easeFactor = word.easeFactor,
        intervalDays = word.intervalDays,
        timesReviewed = word.timesReviewed,
        lastReviewed = word.lastReviewed,

        synonymsJson = gson.toJson(word.synonyms),
        antonymsJson = gson.toJson(word.antonyms),
        meaningsJson = gson.toJson(word.meanings),
        language = word.language,
      )
    }


}
```

ДОДАТОК В

(обов’язковий)

# ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ


Рисунок В.1 - Слайд 1

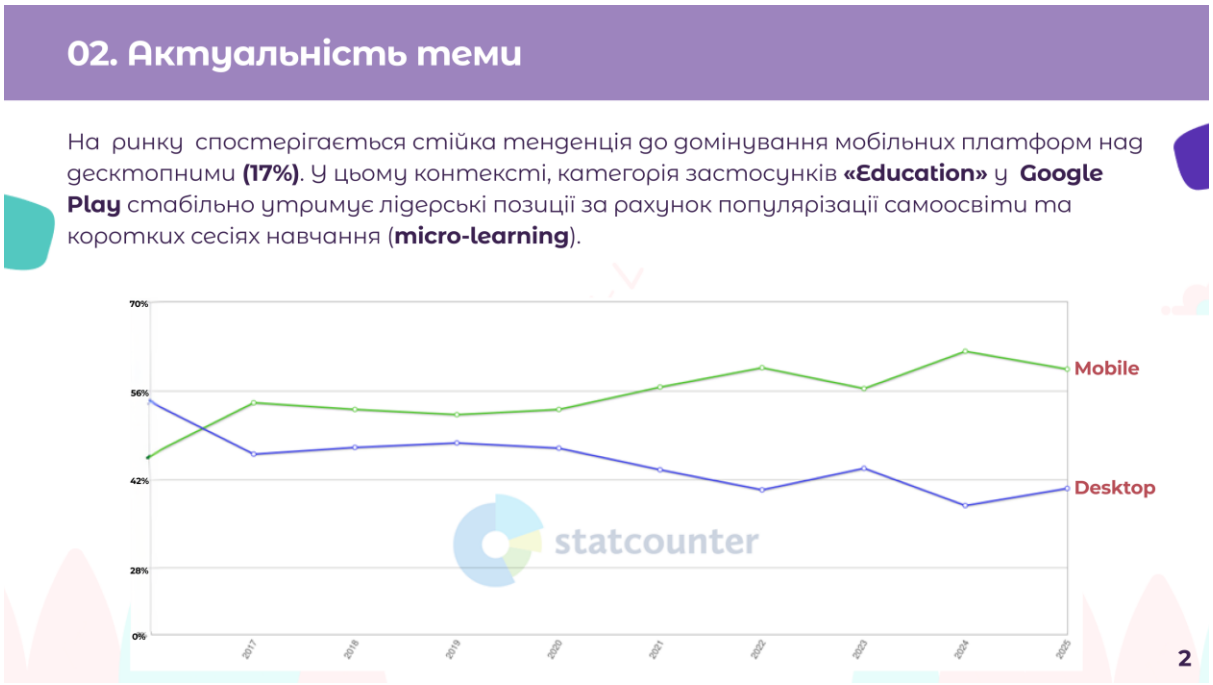
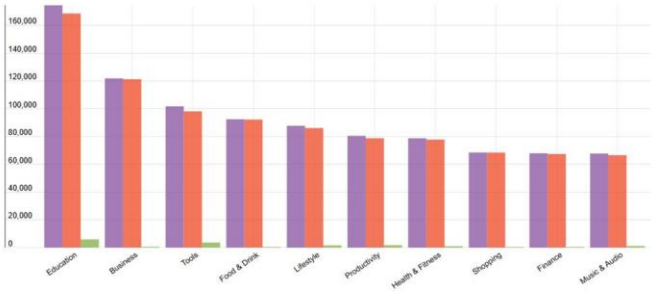


Рисунок В.2 - Слайд 2

## 02. Актуальність теми

Більшість існуючих рішень мають заздалегідь підготовлений матеріал, що обмежує персоналізацію та варіативність роботи користувача з системою.

Актуальність проекту полягає у вирішенні цієї проблеми шляхом поєднання науково обґрунтованого **алгоритму інтервальних повторень (SRS)** із механіками **гейміфікації** та особистими даними, що дозволяє адаптувати навчальний план під індивідуальні особливості кожного користувача.




3

Рисунок В.3 - Слайд 3

## 03. Мета та Завдання

**Метою роботи** є створення інформаційної системи для інтерактивного вивчення іноземних мов із використанням зовнішніх сервісів, що забезпечує **персоналізований** та **інтерактивний** навчальний процес, збереження та обробку навчальних даних у хмарному середовищі, а також інтеграцію з зовнішніми API.

Для досягнення цієї мети було визначено наступні завдання:

- виконати аналіз предметної області мобільних застосунків для вивчення іноземних мов
- проаналізувати існуючі програмні рішення та визначити їх недоліки
- встановити функціональні та нефункціональні вимоги до системи
- виконати детальне проектування архітектури
- обґрунтувати вибір технологій
- виконати програмну реалізацію
- реалізувати інтеграцію із зовнішніми сервісами через API;
- провести тестування розробленої системи
- проаналізувати отримані результати за вимогами

4

Рисунок В.4 - Слайд 4

## 03. Мета та Завдання

**Метою роботи** є створення інформаційної системи для інтерактивного вивчення іноземних мов із використанням зовнішніх сервісів, що забезпечує **персоналізований** та **інтерактивний** навчальний процес, збереження та обробку навчальних даних у хмарному середовищі, а також інтеграцію з зовнішніми API.

Для досягнення цієї мети було визначено наступні завдання:

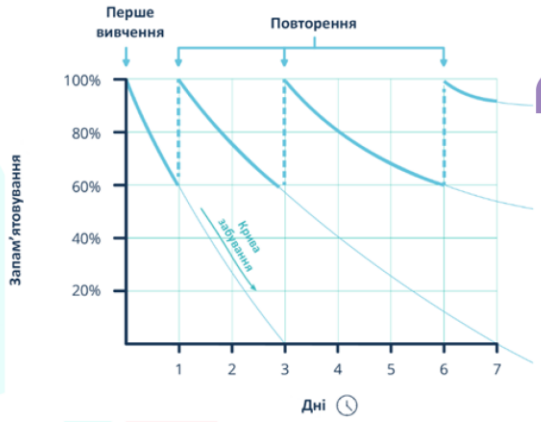
- виконати аналіз предметної області мобільних застосунків для вивчення іноземних мов
- проаналізувати існуючі програмні рішення та визначити їх недоліки
- встановити функціональні та нефункціональні вимоги до системи
- виконати детальне проектування архітектури
- обґрунтувати вибір технологій
- виконати програмну реалізацію
- реалізувати інтеграцію із зовнішніми сервісами через API;
- провести тестування розробленої системи
- проаналізувати отримані результати за вимогами

4

Рисунок В.5 - Слайд 5

## 04. Змістовий аналіз предметної області

Рішенням цього є **алгоритм інтервальних повторень - SRS.** Вона автоматично розраховує оптимальні моменти для повторення знань, базуючись на успіхах користувача що забезпечує їх надійне закріплення у довготривалій пам'яті..




Особливістю реалізації системі в проекті, є перехід від ручного введення даних до автоматизації. Це дозволяє реалізувати концепцію **Microlearning**, де користувач витрачає мінімум часу на налаштування і максимум на саме навчання

6

Рисунок В.6 - Слайд 6

## 05. Аналіз існуючих рішень

Детально проаналізовано **8 застосунків** з акцентом на
- Duolingo (Гейміфікація)
- AnkiDroid (Алгоритм SM-2)
- DuoCards (Флеш-Картки)
- Reverso (Пошук значень)

окремо поверхнево розглянуто ще **9 існуючих рішень**.
за **28 характеристиками:**
- Флеш-Картки
- Офлайн режим
- Свій контент
- Тести
- Перекладач
- ШІ
- Мікронавчання

7

Рисунок В.7 - Слайд 7

## 05. Аналіз існуючих рішень

Детально проаналізовано **8 застосунків** з акцентом на
- Duolingo (Гейміфікація)
- AnkiDroid (Алгоритм SM-2)
- DuoCards (Флеш-Картки)
- Reverso (Пошук значень)

окремо поверхнево розглянуто ще **9 існуючих рішень**.
за **28 характеристиками:**
- Флеш-Картки
- Офлайн режим
- Свій контент
- Тести
- Перекладач
- ШІ
- Мікронавчання

7

Рисунок В.8 - Слайд 8

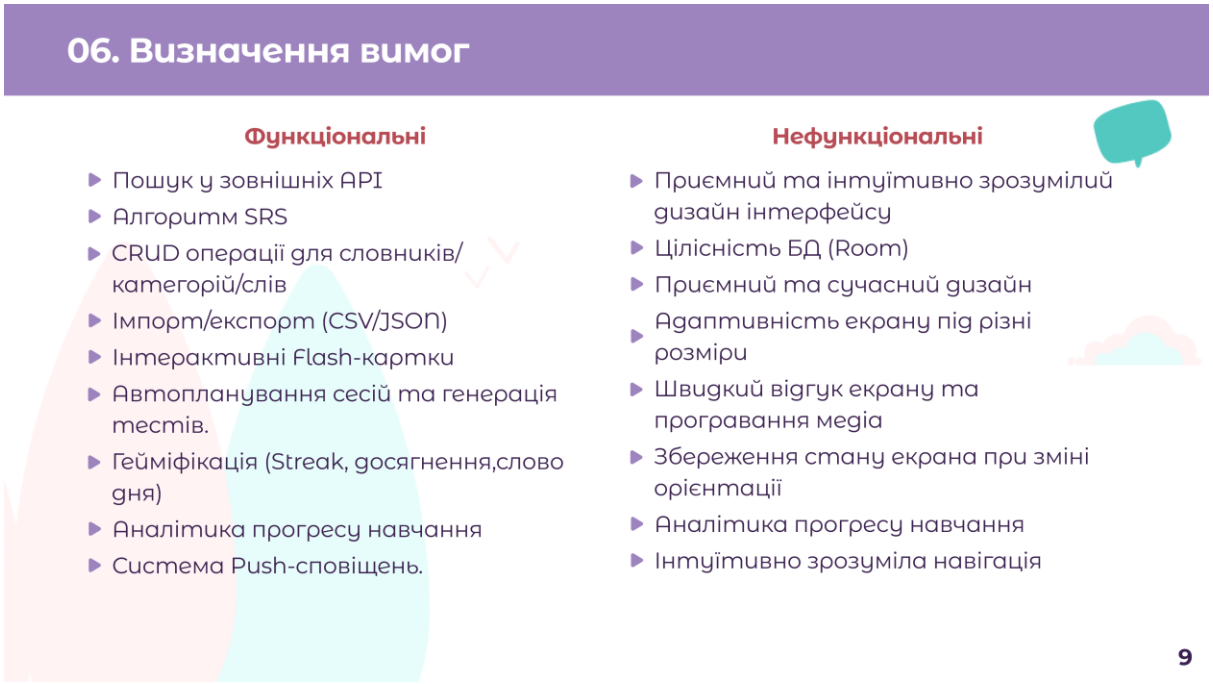



Рисунок В.9 - Слайд 9

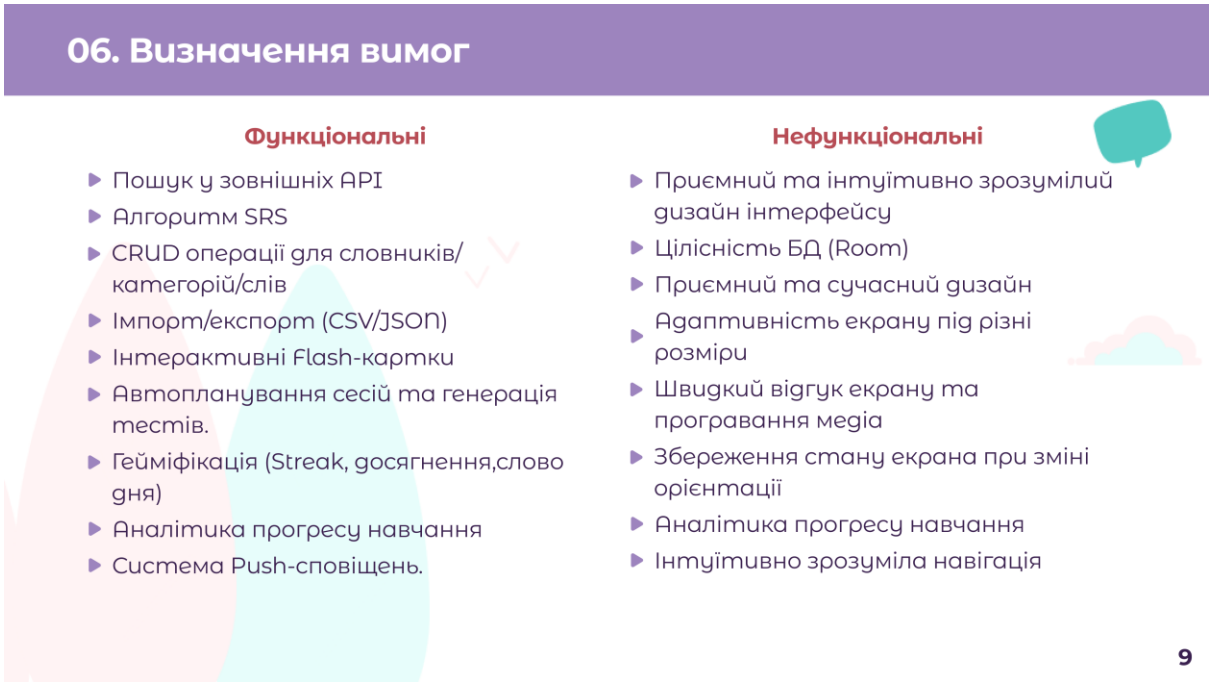



Рисунок В.10 - Слайд 10

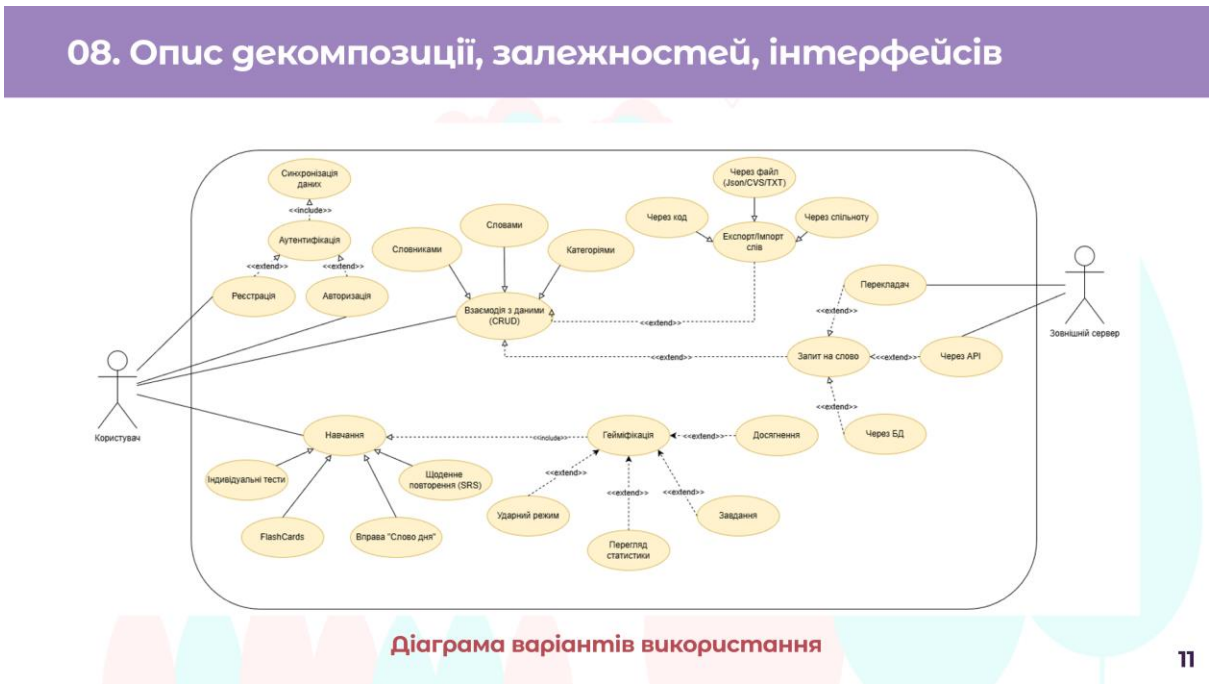



Рисунок В.11 - Слайд 11

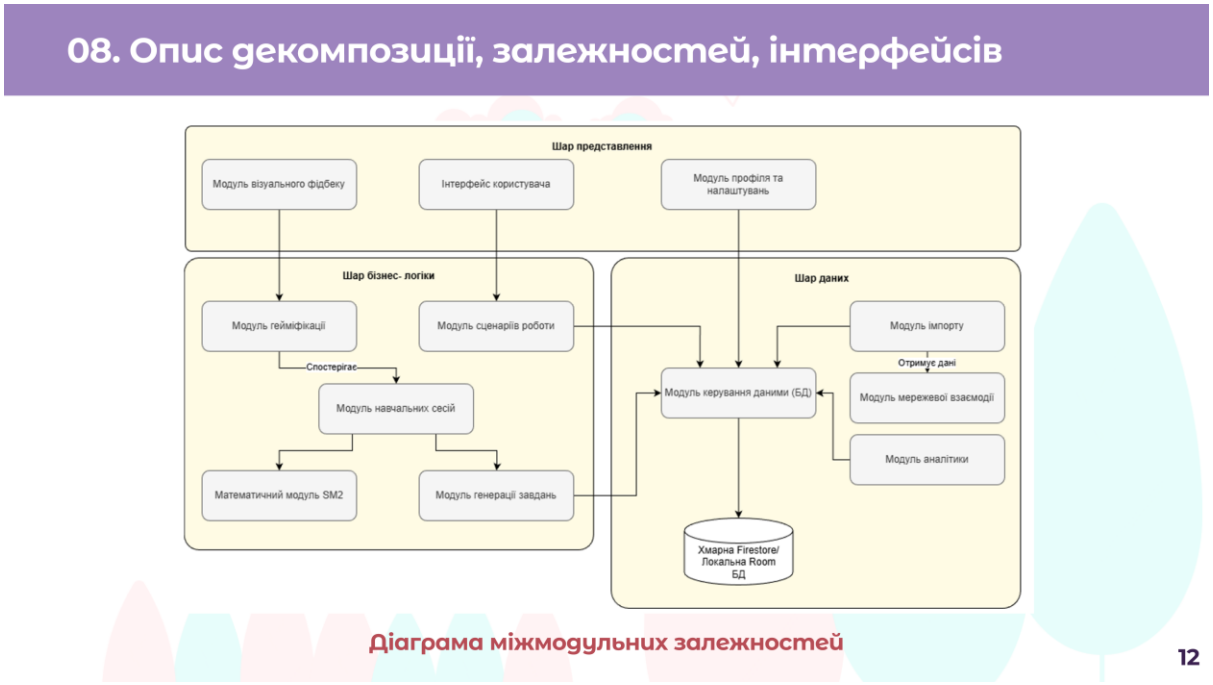



Рисунок В.12 - Слайд 12

## 08. Опис декомпозиції, залежностей, інтерфейсів

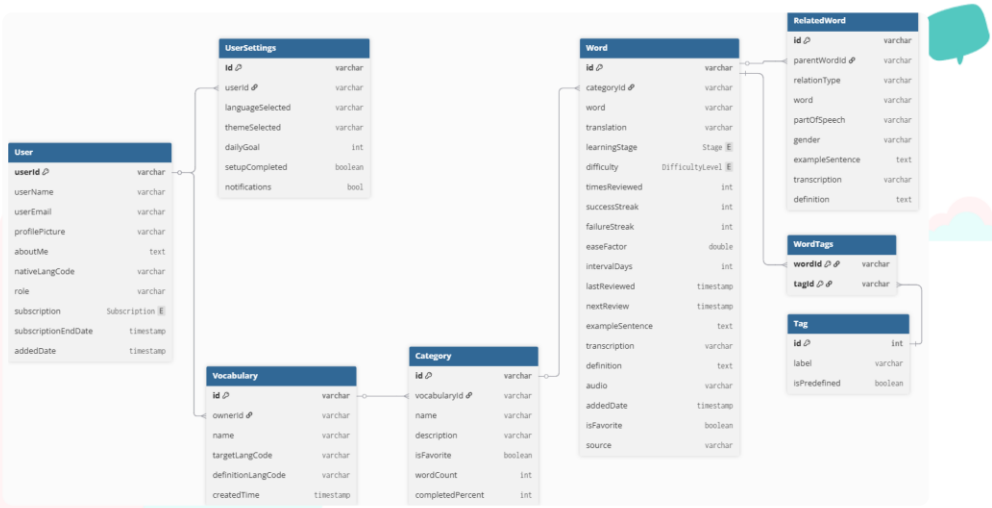
**Інтерфейси**

| | |
|---|---|
| OnWordActionListener | onBookmark(), onDelete(), onMark(), onEdit(), onWordClick() |
| OnCategoryActionListener | onDelete(), onEdit(),onCategoryClick() |
| UserRepository | Повний CRUD для Словників, Категорій та Слів : set(),update(),delete(), get(); initializeSession(), syncSettings(), updateSubscription(), clearAllData(), |
| UserRemoteDataSource | CRUD-запити для Словників, Категорій та Слів, в умовах NoSQL-БД : get(), set(), update(),delete(),add() |
| UserLocalDataSource | CRUD-запити для Словників, Категорій та Слів, в умовах SQL-БД : getAll(),update(),updateField(), ,deleteBy(), delete(), setWhere() |
| UserDao | SQL-запити до таблиць : insert(), delete(), update() |
| DictionaryApi. WictonaryApi | getWordInfo() |
| VocabulerApiService | getTranslation(), getWordInfo(), getWOTD(), |

13

Рисунок В.13 - Слайд 13

## 09. Проектування модулів і даних

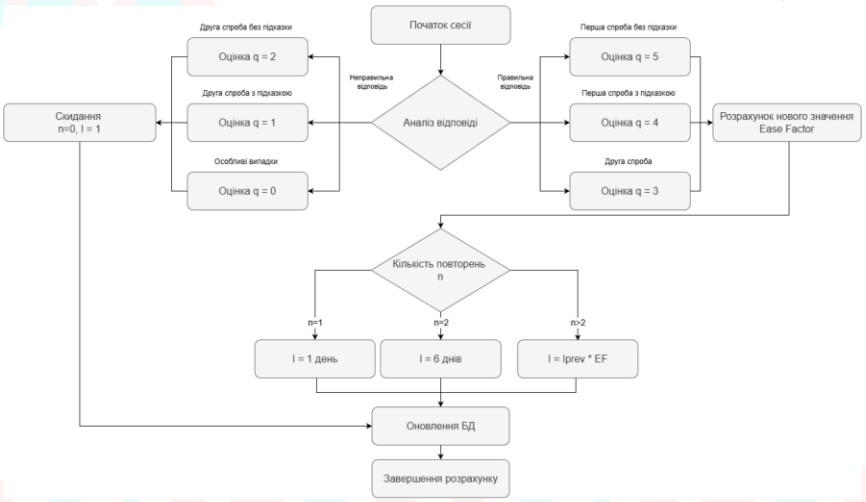



Діаграма сутностей системи (ERD)

14

Рисунок В.14 - Слайд 14

## 09. Проектування модулів і даних

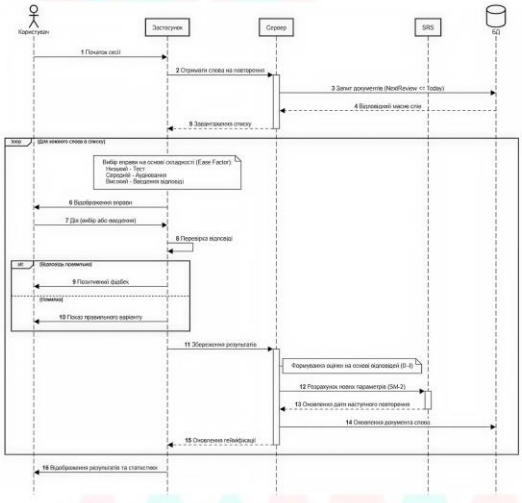


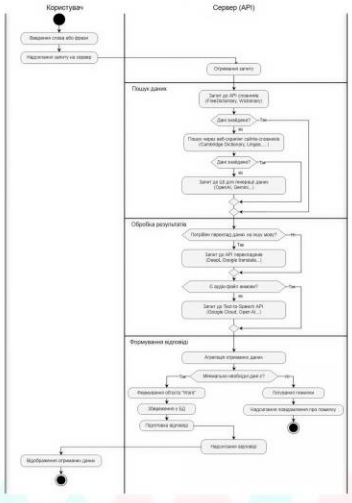

Блок-схема алгоритму розрахунку інтервалів

15

Рисунок В.15 - Слайд 15

## 09. Проектування модулів і даних




Діаграма послідовності процесу тестування

Діаграма станів процесу запиту інформації про слово

16

Рисунок В.16 - Слайд 16


Рисунок В.17 - Слайд 17

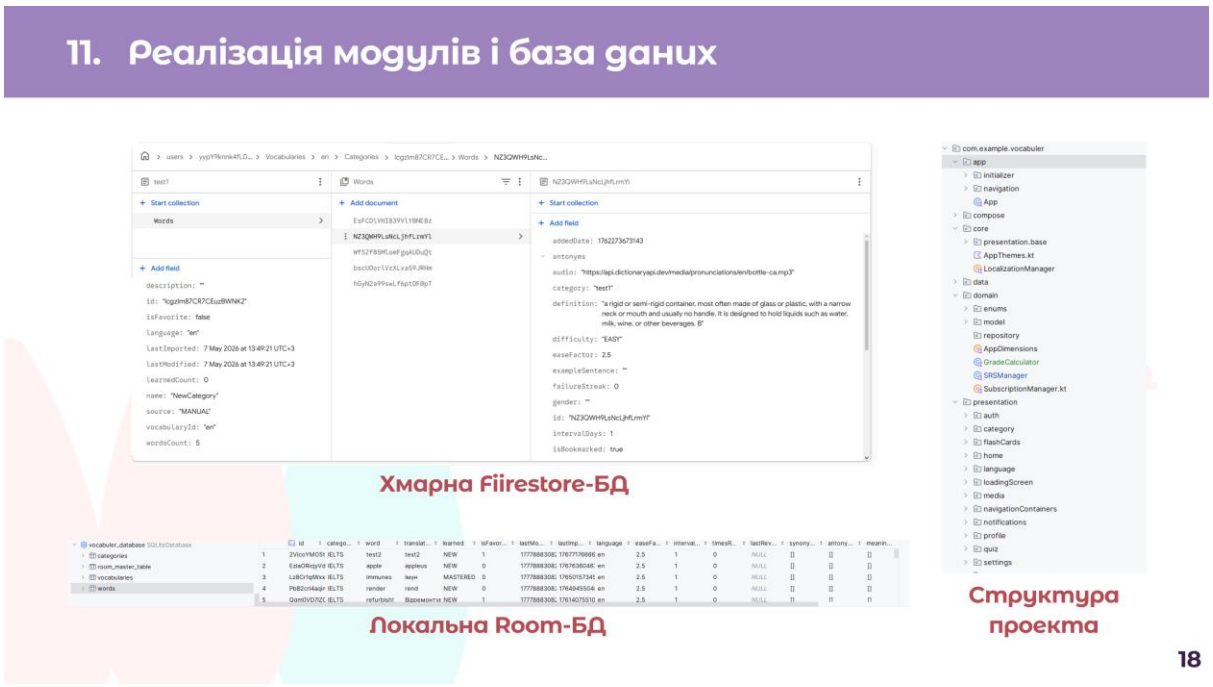



Рисунок В.18 - Слайд 18

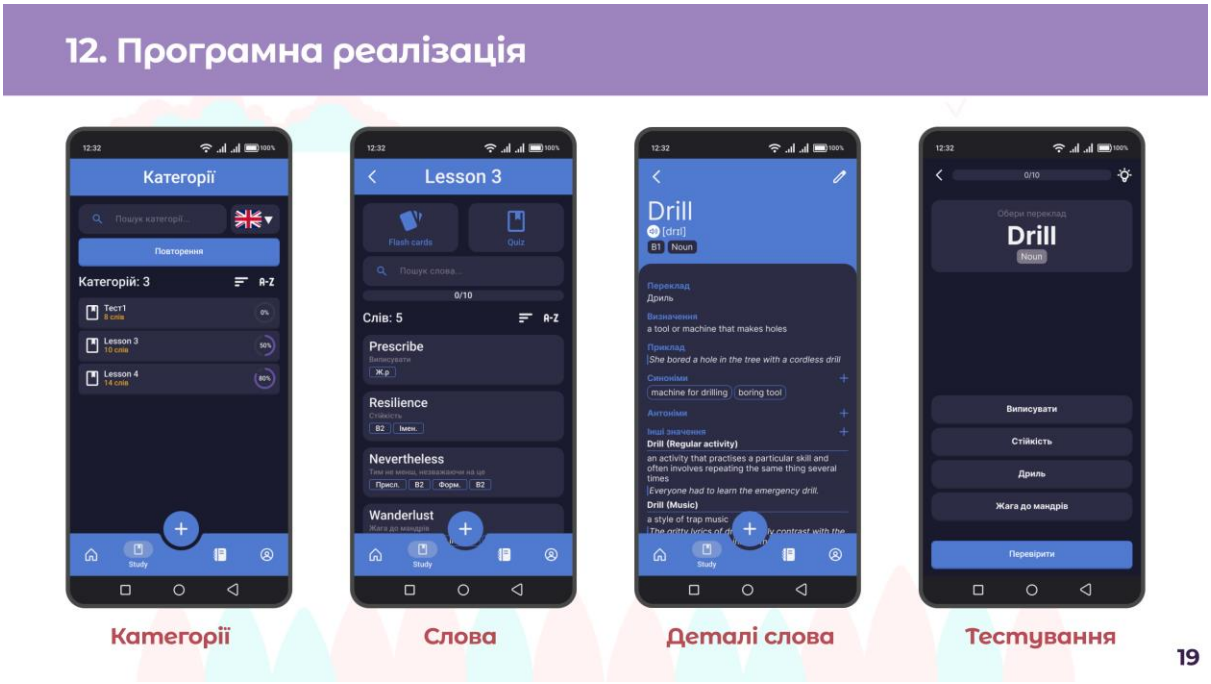



Рисунок В.19 - Слайд 19

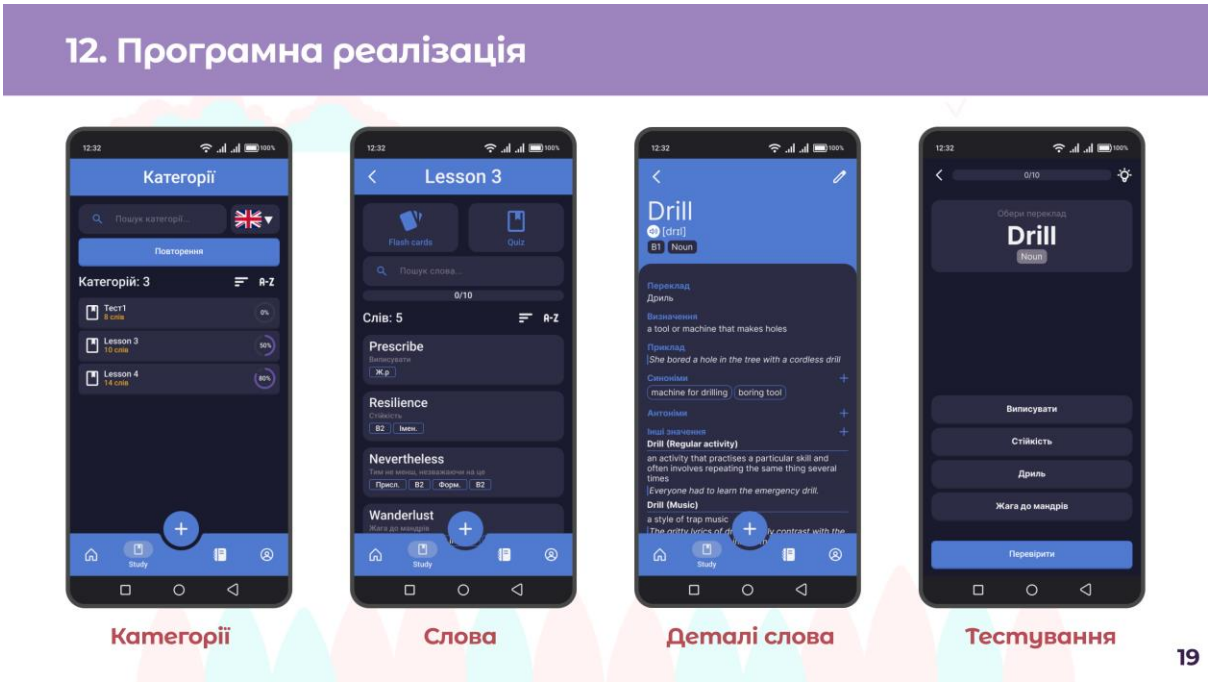



Рисунок В.20 - Слайд 20

## 14. Висновки

В межах кваліфікаційної роботи, було спроектовано та розроблено Інформаційну система для вивчення іноземних мов з використанням зовнішніх API-сервісів, для чого було виконано всі попередньо встановлені завдання:

| | |
|---|---|
| Дослідження предметної області та наявних рішень | Проведено дослідження особливостей області e-education та існуючих в ній рішень, виявлено ключові потреби користувачів та стан ринку обраної галузі |
| Визначення вимог до системи | Сформовано перелік функціональних та нефункціональних вимог до системи |
| Проектування архітектури | Створено UML-діаграми, схему бази даних, загальну архітектуру ПЗ та математичну модель алгоритму SM-2. |
| Вибір технологій | Обґрунтовано стек відповідно до спроектованої архітектури |
| Програмна реалізація | Створено функціональний мобільний застосунок з інтеграцією зовнішніх мовних API |
| Тестування системи | Проведено комплексне тестування для верифікації відповідності встановленим вимогам. |

Виконання всіх поставлених завдань, та успішне тестування результатів, свідчить про те, що мета кваліфікаційної роботи була досягнута.

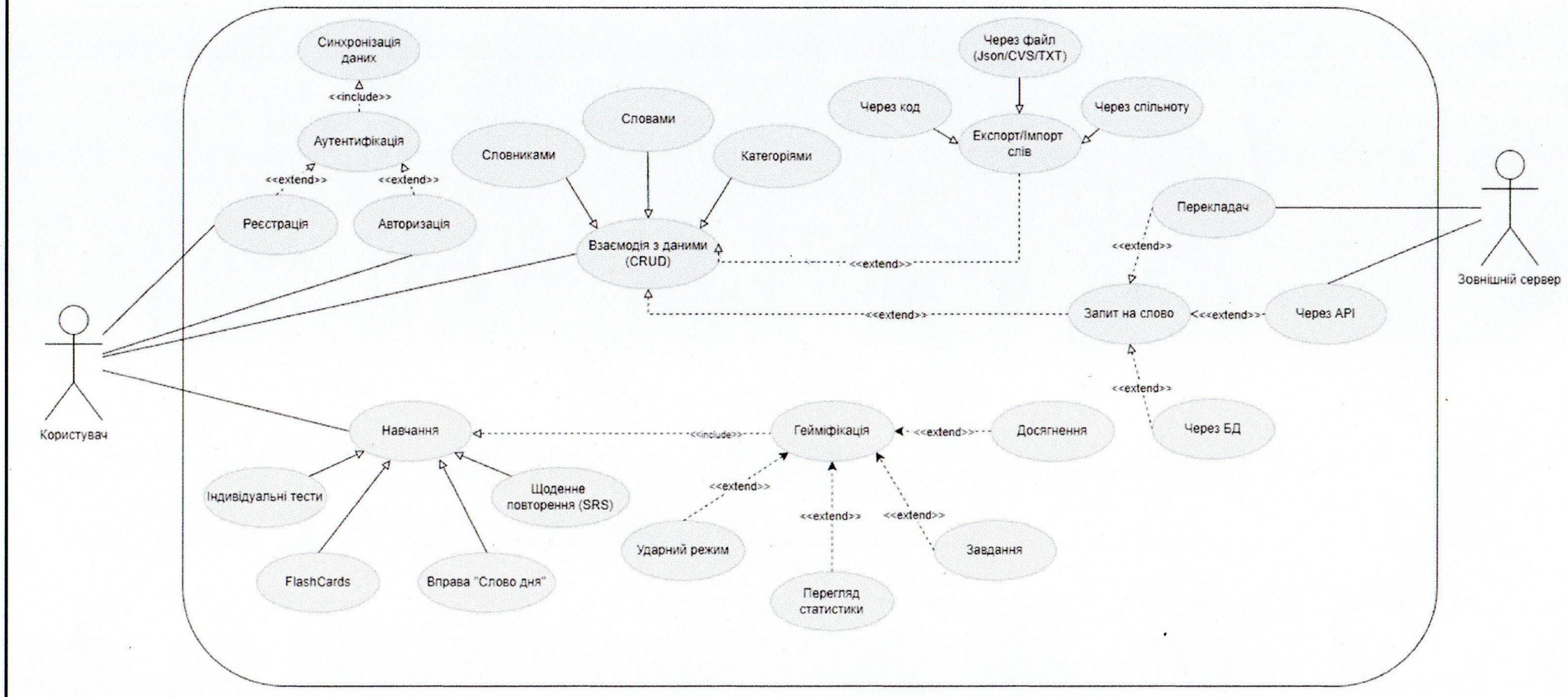
21

Рисунок В.21 - Слайд 21

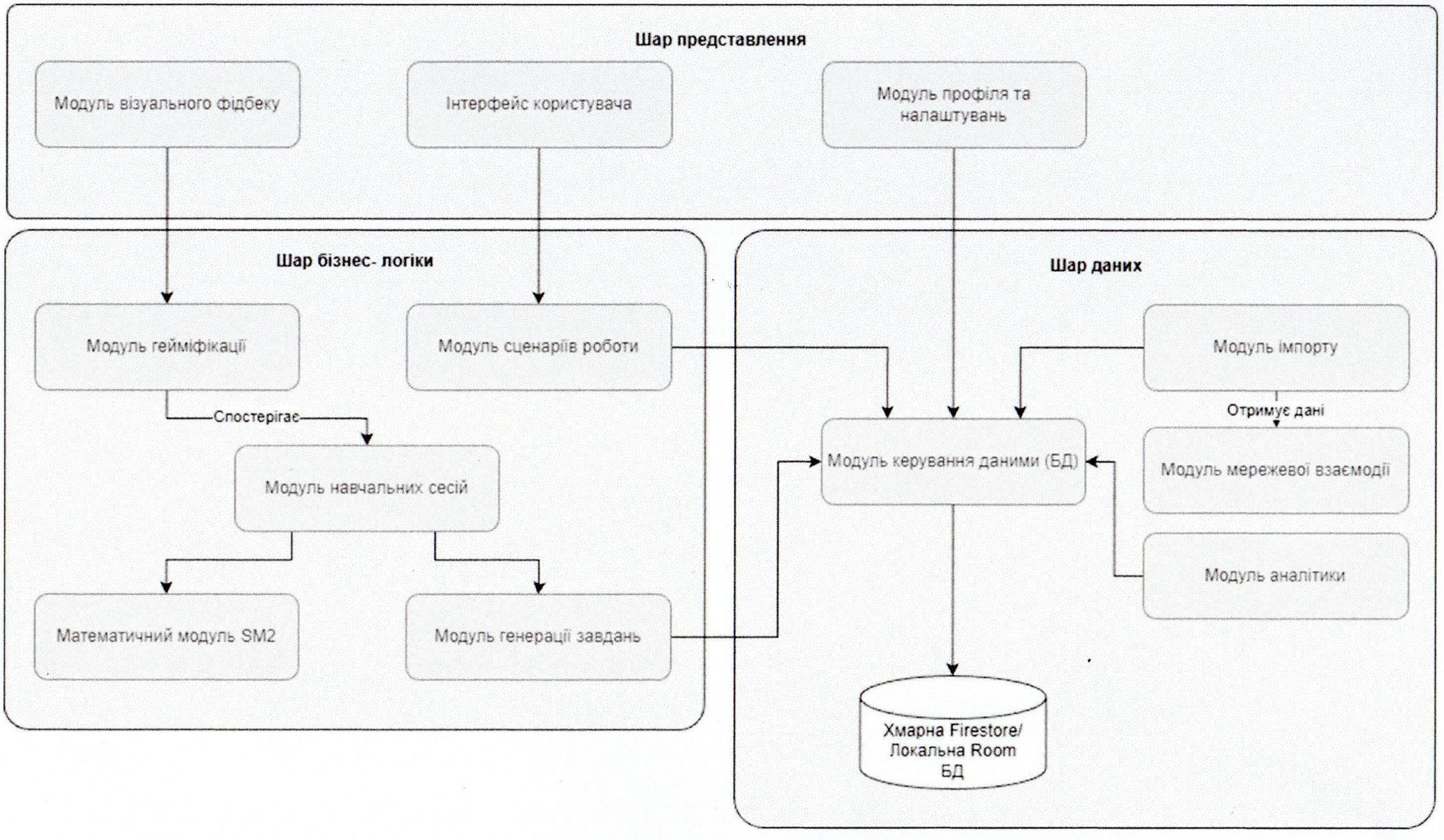

Дякую за увагу!

Рисунок В.22 - Слайд 22

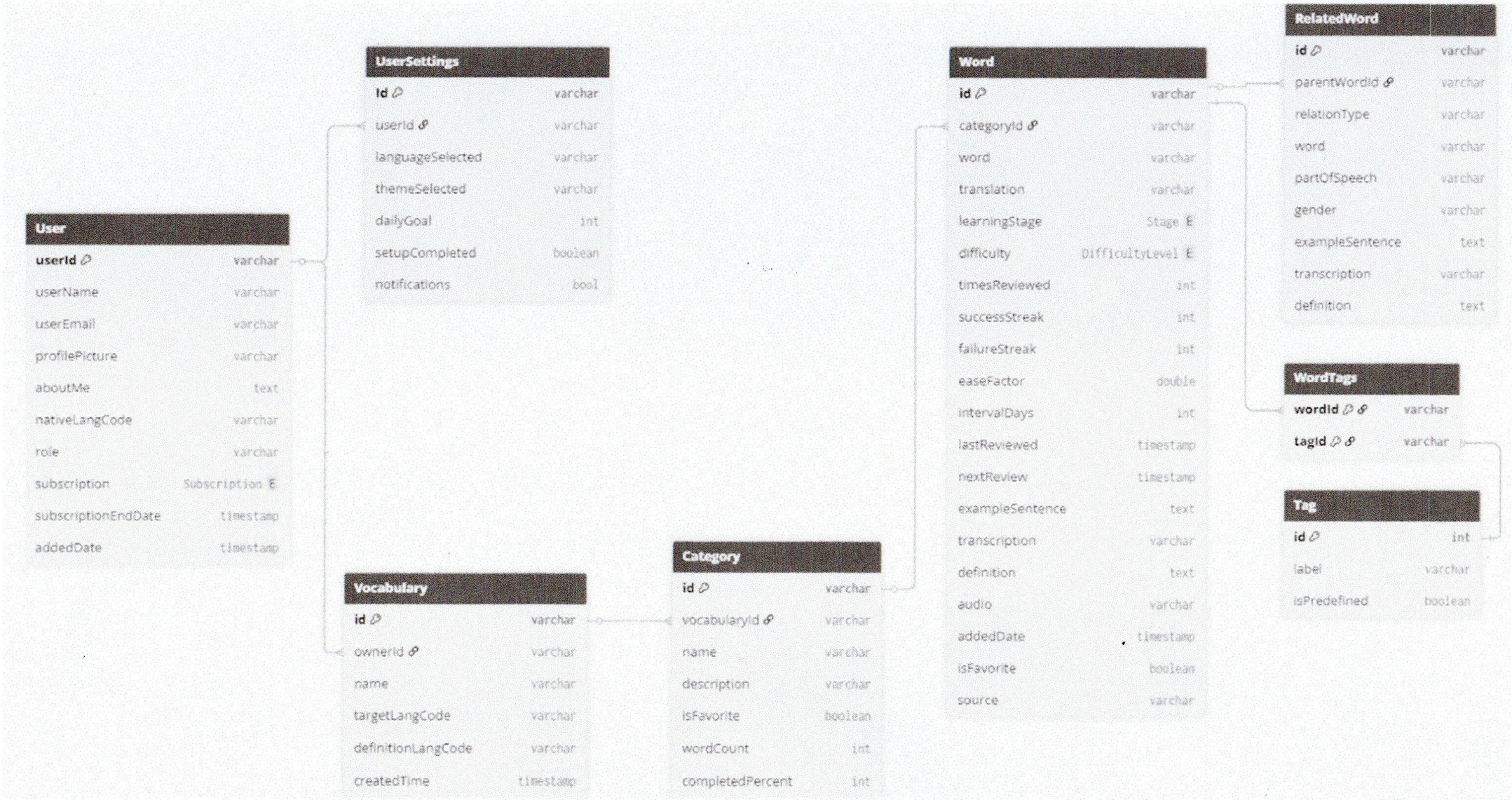
# ГРАФІЧНІ МАТЕРІАЛИ




| | | | | | КвРІПЗ.2201111.01.26.Е8 | | | |
|---|---|---|---|---|---|---|---|---|
| Змн. | Арк | № докум. | Підпис | Дата | Інформаційна система для вивчення іноземних мов з використанням API-сервісів | Літ. | Маса | Масштаб |
| *Розробив* | | Сумка Т. С. | | 27.05.26 | | | | |
| *Керівник* | | Яшина О. М. | | 27.05 | | | | |
| | | | | | Діаграма варіантів використання | Аркуш 1 | Аркушів 3 | |
| Н. Контр. | | Форкун Ю. В. | | 27.05 | | ХНУ,ІПЗ-22-1 | | |
| *Затверд.* | | Бедратюк Л. П. | | 01.06.26 | | | | |




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | КвРІПЗ.2201111.01.26.Е8 | | |
| Змн. | Арк | № докум. | Підпис | Дата | Інформаційна система для вивчення іноземних мов з використанням API-сервісів | Літ. | Маса / Масштаб |
| Розробив | | Сумка Т. С. | | 27.05.26 | | | |
| Керівник | | Яшина О. М. | | 27.05 | | | |
| | | | | | Діаграма міжмодульних зв'язків | Аркуш 2 | Аркушів 3 |
| Н. Контр. | | Форкун Ю. В. | | 27.05.26 | | ХНУ,ІПЗ-22-1 | |
| Затверд. | | Бедратюк Л. П. | | 27.05.26 | | | |




| | | | | | КвРІПЗ.2201111.01.26.Е8 | | | |
|---|---|---|---|---|---|---|---|---|
| Змн. | Арк | № докум. | Підпис | Дата | Інформаційна система для вивчення іноземних мов з використанням API-сервісів | *Літ.* | *Маса* | *Масштаб* |
| *Розробив* | | Сумка Т. С. | | 27.05.26 | | | | |
| *Керівник* | | Яшина О. М. | | 27.05.26 | | | | |
| | | | | | Діаграма класів | Аркуш 3 | Аркушів 3 | |
| Н. Контр. | | Форкун Ю. В. | | 31.05.26 | | *ХНУ,ІПЗ-22-1* | | |
| *Затверд.* | | Бедратюк Л. П. | | 01.06.26 | | | | |

# СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Сумки Тимофія Сергійовича
факультет ІТ, IVкурс, група ІПЗ-22-1

**ЗАЯВА**

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

18.05.26
дата

підпис

# Anti-Plagiarism (http://ap.km.ua) v-16.718

**Максимальне співпадіння з одним документом 1.0%**

Словники перевірки: UA, US, RU. **Помилок в документах: 13%**

| ID: 271878<br>Назва: «Інформаційна система для вивчення іноземних мов з використанням зовнішніх API-сервісів»<br>Додано в БД: 2026-05-21<br>Автора: Тимофій СУМКА<br>Керівники: канд. техн. наук, доцент Оксана ЯШИНА<br>Консультанти:<br>Опоненти: | Документ | | Сумарний збіг по Базі Даних | |
|---|---|---|---|---|
| | Символи | Лексеми | Символи | Лексеми |
| | 93136 | 651 | 1687 (2%) | 28 (4%) |

Джерело плагіату

| ID | Опис | Наявність плагіату в документі | |
|---|---|---|---|
| | | Символи | Лексеми |

**Протокол аналізу звіту подібності науковим керівником**

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

**Автор:** Тимофій СУМКА

**Співавтор:**

**Назва:** «Інформаційна система для вивчення іноземних мов з використанням зовнішніх API-сервісів»

**Науковий керівник:** канд. техн. наук, доцент Оксана Яшина

**Підрозділ:** Кафедра інженерії програмного забезпечення

**Коефіцієнт подібності 1:**2.48%

**Коефіцієнт подібності 2:**0.51%

**Мікропробіли:** 15

**Заміна букв:** 1

**Інтервали:** 0

**Білі знаки:** 37

**Дата створення звіту:** 2026-05-21 10:58:30.0

**Після аналізу Звіту подібності констатую наступне:**

☐ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

*Дата* 25.05.2026 експерт

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

**РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ**
освітнього ступеня «Бакалавр»

Дипломник Сумка Тимофій Сергійович

Тема Інформаційна система для вивчення іноземних мов з використанням зовнішніх API-сервісів

Спеціальність 121 – Інженерія програмного забезпечення

**Обсяг кваліфікаційної роботи:**

Кількість листів креслень 3; кількість сторінок записки 80

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі досліджено предметну область мобільних застосунків для вивчення іноземних мов, проаналізовано існуючі рішення, визначено їхні недоліки та обґрунтовано доцільність розробки нового ПЗ. Сформовано функціональні та нефункціональні вимоги, побудовано UML-діаграми варіантів використання. В результаті проєктування обрано патерн MVVM, спроєктовано модульну структуру, схему локальної та хмарної БД. Реалізовано мобільний Android-застосунок мовою Kotlin із використанням алгоритму інтервальних повторень SM-2, підсистеми гейміфікації та інтеграції із зовнішніми API-сервісами.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання з дотриманням всіх вимог

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі обґрунтовано актуальність теми, сформульовано мету та завдання проєктування. У першому розділі проведено аналіз ринку мобільних освітніх застосунків, вивчено психологічні аспекти засвоєння знань, зокрема методику інтервальних повторень та принципи гейміфікації за фреймворком Octalysis. Проведено порівняльний аналіз аналогів та сформовано вимоги до ПЗ. У другому розділі обрано архітектурне рішення, спроєктовано модульну декомпозицію системи, описано міжмодульні залежності та інтерфейси, розроблено модель даних. Визначено та обґрунтовано технологічний стек. У третьому розділі описано програмну реалізацію модулів, зокрема математичного ядра SRS, репозиторіїв з підтримкою Offline-first та інтеграцію із зовнішніми API-сервісами. Тестування проведено за стратегією піраміди тестування. Результати підтвердили відповідність системи вимогам та коректність роботи на різних пристроях.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є актуальною , оскільки попит на мобільні інструменти для вивчення іноземних мов стабільно зростає. Застосовано сучасний алгоритм SM-2, що підвищує ефективність засвоєння лексики. Реалізовано принцип Offline-first, що забезпечує роботу без постійного підключення до мережі. Підсистема гейміфікації на основі фреймворку Octalysis є науково обґрунтованим рішенням для підтримки мотивації. Інтеграція із зовнішніми API словників автоматизує додавання слів. Архітектура Clean Architecture забезпечує масштабованість системи, а використаний стек технологій відповідає сучасним стандартам Android-розробки.

5. Негативні сторони роботи Тестування охоплює основні сценарії, проте не включає навантажувального тестування при великій кількості слів у БД. Також у роботі відсутня підтримка розпізнавання мовлення, що обмежує можливості тренування вимови.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до тематики роботи та представлено у вигляді UML-діаграм, що наочно відображають архітектурні рішення та логіку роботи системи. Пояснювальна записка оформлена відповідно до вимог стандартів ХНУ. Структура документа логічна та послідовна, текст викладено діловим стилем.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує на позитивну оцінку. Студент продемонстрував здатність самостійно вирішувати складні інженерні задачі, від аналізу предметної області до повного циклу розробки функціонального мобільного застосунку. Матеріал пояснювальної записки структурований, логічно послідовний та демонструє достатній рівень теоретичних і практичних знань у галузі інженерії програмного забезпечення.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «добре»

РЕЦЕНЗЕНТ Лігенчук Андрій Олександрович, кандидат технічних наук, доцент кафедри комп'ютерної інженерії та інформаційних систем

"1" червня 2026 р. _____
(підпис)

**РІШЕННЯ** ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів.
Назва кваліфікаційної роботи: «Інформаційна система для вивчення іноземних мов .з використанням зовнішніх API-сервісів»
Автор: Сумка Тимофій Сергійович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Яшина Оксана Миколаївна, кандидат технічних наук, доцент
Після аналізу звіту/звітів зроблено такий висновок:

| № | Висновок | Позначка про відповідність |
|---|---|---|
| 1 | Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається до захисту. | **відповідає** |
| 2 | Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. | |
| 3 | Виявлені запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Виявлені запозичення частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат. | |
| 4 | Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту. | |
| 5 | Інше: | |

Підтвердження:
Запозичення, виявлені у роботі, є законними і не є плагіатом, оскільки:
1) у тексті кваліфікаційної роботи системою перевірки на плагіат Anti-Plagiarism виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках, у структурі змісту, назвах розділів/підрозділів, рамках форм, у назвах та URL-адресах публікацій переліку джерел посилання;
2) запозичення, виявлені у тексті роботи, є фрагментарними.
Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 1.0% з одного джерела. Загальна сумарна подібність у базі даних складає 2% за символами та 4% за лексемами. Крім того, за результатами додаткового аналізу коефіцієнт подібності 1 становить 2.48%, коефіцієнт подібності 2 - 0.51%. Виявлено мікропробілів: 15, зайвих білих знаків: 37, заміна букв: 1, маніпуляцій з інтервалами: 0. З урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 27.05.26

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Оксана ЯШИНА