Полтавський університет економіки і торгівлі
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп’ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)
«_____»_____2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

«ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕРАКТИВНИХ ТЕСТІВ З МОВИ ПРОГРАМУВАННЯ KOTLIN ДЛЯ МОБІЛЬНИХ ПРИСТРОЇВ ANDROID»

зі спеціальності 122 «Комп’ютерні науки»
освітня програма «Комп’ютерні науки»
ступеня бакалавра

Виконавець роботи Прокопчук Руслан Вікторович
_____ «___»_____2026 р.
(підпис)

Науковий керівник к., ф.-м. н., доц., Черненко Оксана Олексіївна
_____ «___»_____2026 р.
(підпис)

Рецензент

ПОЛТАВА 2026

ЗАТВЕРДЖУЮ
Завідувач кафедри _____Олена ОЛЬХОВСЬКА
«____»_____ 202_ р.

ЗАВДАННЯ ТА КАЛЕНДАРНИЙ ГРАФІК
ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Програмна реалізація інтерактивних тестів з мови програмування Kotlin для мобільних пристроїв Android»

зі спеціальності 122 «Комп'ютерні науки»

освітня програма «Комп'ютерні науки»

ступеня бакалавр

Прізвище, ім'я, по батькові Прокопчук Руслан Вікторович

Затверджена наказом ректора № №213-Н від «01» жовтня 2025 р.

Термін подання студентом роботи «___» _____ 20__ р.

Вихідні дані до кваліфікаційної роботи: публікації та лекційний матеріал з теми, навчальні тренажери в дистанційних курсах з комп'ютерних наук.

Зміст пояснювальної записки (перелік питань, які потрібно розробити)

ВСТУП

1. ПОСТАНОВКА ЗАДАЧІ
2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1 Огляд існуючих додатків та платформ інтерактивних тестів

2.2 Аналіз методик тестування знань з програмування: переваги та недоліки різних підходів

2.3 Дослідження специфіки вивчення Kotlin: ключові концепції та типові складнощі

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1 Порівняльний аналіз архітектурних підходів: BaaS проти традиційної клієнт-серверної архітектури

3.2 Проектування архітектури на засадах Clean Architecture та паттерну MVVM

3.3 Розробка схеми даних у NoSQL системі Firestore: стратегії денормалізації та безпеки.

4. ПРАКТИЧНА ЧАСТИНА

4.1 Проектування структури додатку: навігація, модулі та компоненти системи

4.2 Розробка концепції користувацького інтерфейсу та сценаріїв взаємодії

4.3 Реалізація хмарної інфраструктури та сервісів Firebase

4.4 Технічна реалізація мобільного застосунку

4.5 Опис роботи застосунку для проходження тестувань

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Розділ	ПІП, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі	Черненко О. О.		
Інформаційний огляд	Черненко О. О.		
Теоретична частина	Черненко О. О.		
Практична реалізація	Черненко О. О.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ		
2. Вивчення методичних рекомендацій та стандартів та звіт керівнику		
3. Постановка задачі		
4. Інформаційний огляд джерел бібліотек та інтернету		
5. Теоретична частина		
6. Практична частина		
7. Закінчення оформлення		
8. Доповідь студента на кафедрі		
9. Доробка (за необхідністю), рецензування		

Дата видачі завдання «____»_____ 202_ р.
Здобувач вищої освіти Руслан Прокопчук
Науковий керівник к. ф.-м. н., доц. Оксана ЧЕРНЕНКО

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на

_____
(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 202_ р.
Секретар ЕК_____ _____
(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«____» _____ 202_ р.

Погоджено
Науковий керівник _____
к. ф.-м. н., доц. Оксана ЧЕРНЕНКО
«____» _____ 202_ р.

План

кваліфікаційної роботи ступеня бакалавра
зі спеціальності 122 Комп'ютерні науки
освітня програма 122 Комп'ютерні науки
Прокопчук Руслан Вікторович
Прізвище, ім'я, по батькові
на тему «Програмна реалізація інтерактивних тестів з мови програмування Kotlin для мобільних пристроїв Android»

ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
2. ІНФОРМАЦІЙНИЙ ОГЛЯД
2.1 Огляд існуючих додатків та платформ інтерактивних тестів
2.2 Аналіз методик тестування знань з програмування: переваги та недоліки різних підходів
2.3 Дослідження специфіки вивчення Kotlin: ключові концепції та типові складнощі
3. ТЕОРЕТИЧНА ЧАСТИНА
3.1 Порівняльний аналіз архітектурних підходів: BaaS проти традиційної клієнт-серверної архітектури
3.2 Проектування архітектури застосунку на засадах Clean Architecture та паттерну MVVM
3.3 Розробка схеми даних у NoSQL системі Firestore: стратегії денормалізації та безпеки.
4. ПРАКТИЧНА ЧАСТИНА
4.1 Проектування структури додатку: навігація, модулі та компоненти системи
4.2 Розробка концепції користувацького інтерфейсу та сценаріїв взаємодії
4.3 Реалізація хмарної інфраструктури та сервісів Firebase
4.4 Технічна реалізація мобільного застосунку
4.5 Опис роботи застосунку для проходження тестувань
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А

Здобувач вищої освіти Руслан ПРОКОПЧУК

«____» _____ 202_ р.

РЕФЕРАТ

Записка: 49 сторінок, в тому числі з яких: основна частина – 49 сторінок, літературних джерел – 24 назви, рисунків – 27.

ТЕСТУВАННЯ, МОВА ПРОГРАМУВАННЯ, KOTLIN, ANDROID, JETPACK COMPOSE, FIREBASE, FIRESTORE

Мета роботи – Проєктування та програмна реалізація мобільного застосунку для пристрої на операційній системі Android для проходження інтерактивних тестів з мови програмування Kotlin

Об'єкт розробки – Мобільний застосунок для перевірки знань з мови програмування Kotlin

Методи дослідження та інформаційне забезпечення – аналіз існуючих методик тестування та перевірки знань на їхню ефективність у випадку вивчення мов програмування; зручність графічного інтерфейсу відомих інструментів; використання server-less архітектури. Розроблено адаптивну тему яка підлаштовується під кожен конкретний пристрій, авторизація через Google аккаунт, можливість вибору необхідного тестування, відображення денного прогресу, та "XP" які накопичуються за успішні проходження тестувань.

Результати дослідження. Розроблено Android застосунок, який дає змогу користувачеві проходити інтерактивні тести, які поділені за темами, вони включаються у себе завдання різних типів, такі як завдання з однією відповіддю, декількома, та задачі Парсона, що дає змогу перевіряти як теоретичні знання, так і вміння написання коду. Застосунок має декілька видів статистики, щоб заохочувати користувача виконувати завдання.

Рекомендації щодо використання результатів дослідження.

Даний інструмент рекомендовано використовувати як додатковий під час вивчення мови програмування Kotlin для закріплення знань, або перевірки чи перевірки вже засвоєних знань. Він може стати у нагоді людям які планують

вивчати Kotlin або Android розробку в цілому, надаючи змогу вдосконалювати свої знання незалежно від місця знаходження.

ЗМІСТ

ВСТУП

Актуальність. Мобільні технології кардинально змінили підходи до освіти в галузі програмування [1, 2, 3]. Сьогодні студенти та початківці розробники потребують гнучких інструментів, які дозволяють навчатися у власному темпі та отримувати миттєвий зворотний зв'язок.

Kotlin, будучи офіційною мовою для Android-розробки, вимагає систематичного підходу до вивчення, проте традиційні методи часто не враховують специфіку сприйняття інформації через мобільні пристрої.

Створення спеціалізованого додатку для тестування знань відкриває нові можливості для ефективного засвоєння матеріалу, оскільки поєднує теоретичну базу з практичними завданнями у форматі, звичному для сучасного користувача смартфонів.

Розробка інтерактивного тестового середовища має кілька принципових переваг порівняно з класичними освітніми платформами. Насамперед, це природна інтеграція в повсякденне життя користувача. Людина може присвятити навчанню навіть короткі проміжки часу під час поїздки чи перерви, що значно підвищує регулярність занять та загальну ефективність освітнього процесу. Система адаптивного тестування здатна аналізувати типові помилки конкретного користувача та формувати індивідуальну траєкторію навчання, концентруючись на проблемних темах.

Метою проекту є створення комплексного мобільного рішення, яке трансформує процес вивчення Kotlin із пасивного читання документації в активну взаємодію з матеріалом через різноманітні форми тестування. Робота охоплює дослідження когнітивних особливостей засвоєння програмних концепцій, формування банку завдань з поступовим наростанням складності, побудову гнучкої архітектури додатку, яка забезпечить швидку роботу навіть на пристроях різної потужності.

Предметом розробки є Android-додаток, архітектура якого передбачає модульну систему тестів, механізм градації складності завдань, візуалізацію прогресу навчання через графіки та діаграми, детальні пояснення до кожного питання з прикладами правильного використання конструкцій мови, а також локальне збереження даних для можливості роботи без постійного інтернет-з'єднання.

Очікуваним результатом є функціональний освітній інструмент, який не просто перевіряє знання, а формує глибоке розуміння принципів Kotlin через контекстуальні завдання та детальний аналіз помилок. Додаток покликаний стати супутником розробника на всіх етапах вивчення мови, від знайомства з базовим синтаксисом до опанування просунутих концепцій корутин та функціонального програмування.

Проект ілюструє потенціал мобільних технологій у створенні персоналізованого освітнього досвіду, який враховує індивідуальні особливості кожного учня та адаптується до його потреб.

Обсяг пояснювальної записки до кваліфікаційної роботи складає: 49 стор., в т.ч. основна частина - 49 стор., додатки - 0 стор., джерела - 24 назв. [4]

РОЗДІЛ 1. ПОСТАНОВКА ЗАВДАННЯ

Даний проект присвячений створенню системи інтерактивних тестів для вивчення мови програмування Kotlin, орієнтованих на використання в мобільному середовищі Android. На цьому етапі основний фокус зосереджено на розробці змістовної частини тестування та проектуванні архітектури майбутнього додатку, що створить міцний фундамент для наступного етапу програмної реалізації.

Першим важливим напрямком роботи є формування повноцінної методологічної бази тестових матеріалів. Це передбачає детальне дослідження синтаксису та особливостей Kotlin, структурування навчального матеріалу за тематичними блоками та рівнями складності.

Необхідно розробити систему градації знань, яка дозволить користувачам поступово просуватися від елементарних концепцій до складних конструкцій. Кожен тематичний блок має містити достатню кількість питань для об'єктивної оцінки розуміння матеріалу, при цьому завдання повинні бути сформульовані таким чином, щоб стимулювати критичне мислення та практичне застосування знань.

Другим ключовим завданням є розробка різноманітних форматів тестових питань, що виходять за рамки стандартного множинного вибору. Планується створити завдання на аналіз коду, де користувач має ідентифікувати логічні або синтаксичні помилки у запропонованих фрагментах програм.

Також передбачається розробка інтерактивних вправ на доповнення коду, які вимагатимуть точного знання синтаксичних конструкцій та їх контекстного використання. Окрему увагу буде приділено створенню ситуаційних завдань, що імітують реальні сценарії розробки та вимагають вибору оптимального підходу серед кількох можливих варіантів.

До кожного питання необхідно підготувати детальні пояснення правильних відповідей із прикладами коду, що перетворює процес тестування на повноцінний навчальний досвід.

Третім напрямком роботи є проектування архітектури мобільного додатку з урахуванням специфіки освітнього контенту. Необхідно визначити оптимальну структуру навігації, яка забезпечить інтуїтивний доступ до різних розділів тестування.

Важливо спроектувати систему відстеження прогресу, що візуалізуватиме досягнення користувача та мотивуватиме до подальшого навчання. Слід продумати механізм збереження результатів тестування та аналітику помилок, що дозволить системі формувати персоналізовані рекомендації щодо повторення проблемних тем.

Архітектура має передбачати модульність, щоб у майбутньому можна було легко розширювати банк питань та додавати нові тематичні блоки без переробки всієї системи.

Четвертим завданням є розробка концепції користувацького інтерфейсу, адаптованого для мобільних пристроїв різних розмірів. Інтерфейс має забезпечувати комфортне читання завдань та фрагментів коду на екранах смартфонів, зручне введення відповідей через сенсорне управління та швидкий доступ до додаткових матеріалів та підказок.

Необхідно продумати візуальне оформлення, яке не відволікатиме від навчального процесу, але водночас зробить взаємодію з додатком приємною.

Особливу увагу слід приділити проектуванню екранів результатів тестування, де користувач зможе детально переглянути свої відповіді, ознайомитися з поясненнями та зрозуміти власні помилки.

Результатом цього етапу має стати повноцінна концептуальна модель освітнього додатку, яка включатиме структурований набір тестових завдань, детальну специфікацію архітектури системи, опис користувацьких сценаріїв взаємодії з додатком та технічне завдання для наступного етапу програмної реалізації. Така підготовча робота забезпечить створення якісного освітнього інструменту, який ефективно допомагатиме у вивченні Kotlin та розвитку практичних навичок програмування для Android.

РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1 Огляд існуючих додатків та платформ інтерактивних тестів

При розробці мобільного додатку для тестування знань з мови програмування Kotlin важливо проаналізувати існуючі рішення на ринку. Це дозволяє виявити їхні сильні та слабкі сторони, а також запозичити успішні механіки взаємодії з користувачем та гейміфікації. Для аналізу було обрано платформи: Sololearn, Moodle, Mimo та Duolingo.

1. Sololearn - є однією з найпопулярніших мобільних платформ для вивчення коду з великою спільнотою користувачів (див. рис. 2.1). Платформа пропонує виконувати тести за різними форматами.

Рисунок 2.1 – Вигляд застосунку Sololearn

Основним методом перевірки знань є короткі вікторини після кожного теоретичного блоку. Використовуються такі типи завдань:

1.1 Fill in the blanks: вставка пропущених операторів або ключових слів у фрагмент коду.

1.2 Multiple Choice: вибір правильного варіанту відповіді з кількох запропонованих.

1.3 Reordering: розташування рядків коду в правильній послідовності для виконання алгоритму.

Особливості для Kotlin: Курс Kotlin присутній, але часто базується на контенті, створеному спільнотою. Є вбудований мобільний компілятор, що дозволяє запускати код прямо на смартфоні.

Гейміфікація: Використовує систему XP (досвіду), таблицю лідерів та щоденні "страйки" які мотивують проходити матеріал кожного дня. Унікальною фішкою є режим Code Challenges — дуелі з реальними гравцями в реальному часі, де потрібно швидше відповісти на тестові запитання.

Недоліки: Часто зустрічається застарілий контент у коментарях або курсах спільноти; безкоштовна версія має агресивну рекламу.

1.2. Mimo

Mimo позиціонує себе як найбільш дружній додаток для новачків (див. рис. 2.2), фокусуючись на Web-розробці (HTML/CSS/JS) та Python.

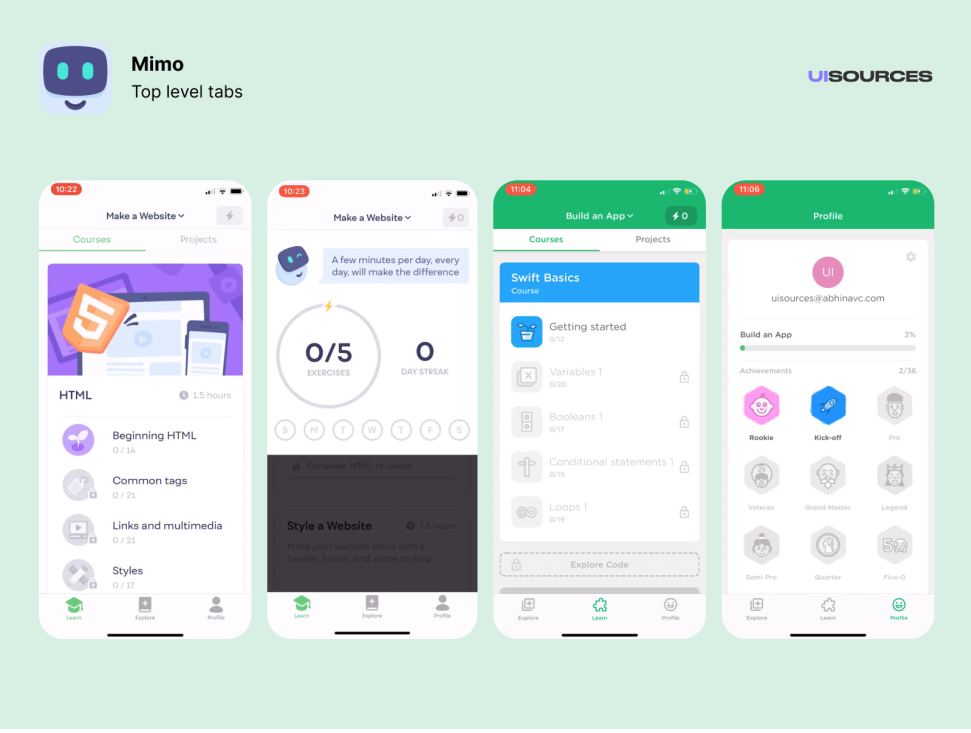


Рисунок 2.2 – Вигляд додатку Mimo

Формат тестів: Інтерфейс максимально спрощений. Тести інтегровані безпосередньо в процес навчання у вигляді мікро-вправ. Користувач рідко пише код з нуля, частіше обирає блоки або дописує змінні.

Графічний інтерфейс: Це приклад ідеального мобільного інтерфейсу — мінімум тексту, максимум інтерактивних елементів, великі кнопки, приємна анімація.

Гейміфікація: Схожа на Duolingo — система "сердець", які втрачаються при помилках, та монети за успішне проходження.

Релевантність для проекту: Хоча Mimo слабко підтримує Kotlin, його інтерфейс та подача матеріалу є чудовим прикладом для наслідування при проектуванні UI власного додатку.

1.3. Duolingo

Хоча Duolingo не є платформою для програмування (див. рис. 2.3) (за винятком базових курсів математики), його механіки утримання користувачів є галузевим стандартом.

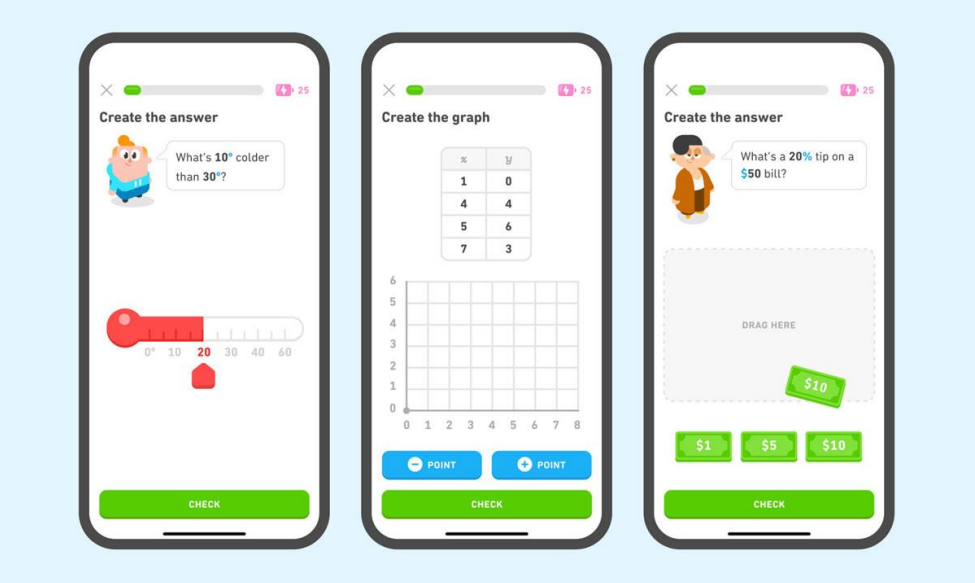


Рисунок 2.3 – Вигляд додатку Duolingo

Streak Freeze: Можливість "заморозити" прогрес, якщо пропустив один день.

Звуковий супровід: Характерні звуки правильної/неправильної відповіді створюють миттєвий зворотний зв'язок.

Прогрес-бар: Візуалізація наближення до мети уроку в реальному часі.

1.4 Moodle

LMS Moodle який використовується в університетах пропонує також можливість використання різних типів тестів таких як обрання однієї правильної відповіді, обрання декількох правильних відповідей, встановити відповідності (див. рис. 2.4), ручне введення відповіді.

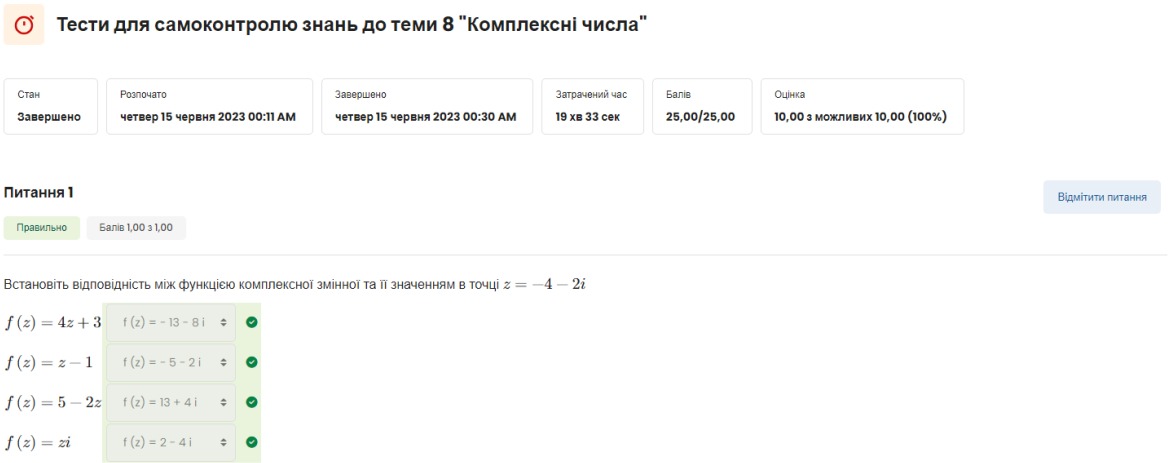


Рис 2.4 – Вигляд питання у тестуванні на платформі Moodle

Всі ці види можна використовувати і з метою навчанню програмуванню, обрання відповідей чудово підходить для засвоювання теоретичних знань, а введення відповіді вручну для того щоб ввести пропущений код або ввести його з нуля.

2.2 Аналіз методик тестування знань з програмування: переваги та недоліки різних підходів

Оцінювання професійної компетентності у сфері розробки програмного забезпечення є багатогранним процесом, який вимагає підходу з ураховунням різних моментів.

На відміну від гуманітарних дисциплін, перевірка знань з мови програмування, такої як Kotlin, охоплює не лише знання синтаксичних правил, але й розуміння алгоритмічних структур, парадигм об'єктно-орієнтованого та

функціонального програмування, а також навички роботи з інструментарієм платформи Android.

У сучасній практиці електронного навчання виділяють кілька ключових методологічних підходів до автоматизованого тестування, кожен з яких має свої специфічні переваги та обмеження, особливо в контексті мобільного використання.

Однією з найбільш розповсюджених методик є тестування закритого типу[5], де користувачеві пропонується обрати правильну відповідь із запропонованого переліку (див. рис. 2.5). До беззаперечних переваг цього методу в контексті мобільної розробки належить низький поріг входження та зручність інтерфейсу: користувачеві не потрібно використовувати віртуальну клавіатуру для введення коду, що суттєво покращує досвід користування на невеликих екранах.

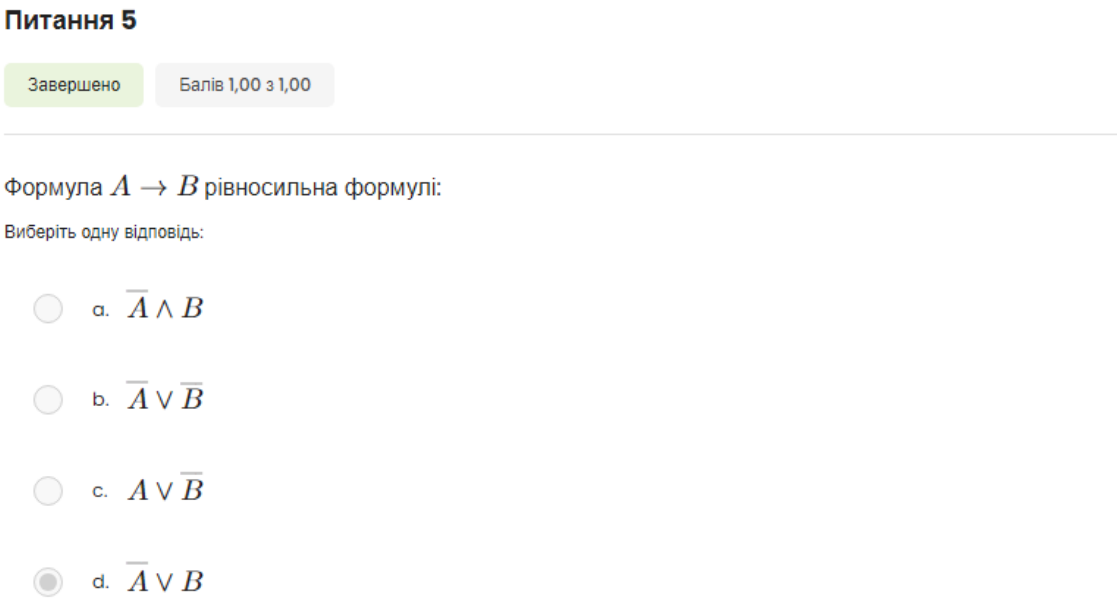


Рисунок 2.5 – Приклад тестування з вибором правильної відповіді

З педагогічної точки зору, ця методика ефективна для перевірки декларативних знань — термінології, базового синтаксису Kotlin, особливостей типізації або запам'ятовування методів стандартних бібліотек.

Однак, ця методика має суттєвий недолік, відомий як «ефект впізнавання». Студент може обрати правильний варіант методом виключення або вгадування, не

маючи при цьому глибокого розуміння предмета. Таке тестування перевіряє здатність розпізнати правильний код, але не здатність його написати, що створює ризик формування "ілюзії компетентності".

Альтернативним та більш прогресивним підходом є методика проблем Парсона [6]. Суть методу полягає в тому, що учневі надаються вже готові рядки коду, які необхідно розташувати у правильній логічній послідовності, щоб отримати працюючий алгоритм. Для вивчення Kotlin, де порядок операцій та вкладеність блоків (наприклад, у лямбда-виразах або корутинах) мають критичне значення, цей метод є надзвичайно ефективним.

Перевагою задач Парсона є зниження розумового навантаження, не пов'язаного безпосередньо з логікою програми: студент не витрачає час на пошук синтаксичних помилок, пропущених дужок чи крапок з комою, а фокусується на побудові алгоритму.

В умовах мобільного додатку цей метод реалізується через механіку перетягування (див. рис. 2.6), що є ідеальним для сенсорних екранів. Водночас, недоліком можна вважати обмеженість варіативності рішень, оскільки користувач змушений діяти в межах наперед визначених фрагментів коду, що не стимулює творчий підхід до розв'язання задач.

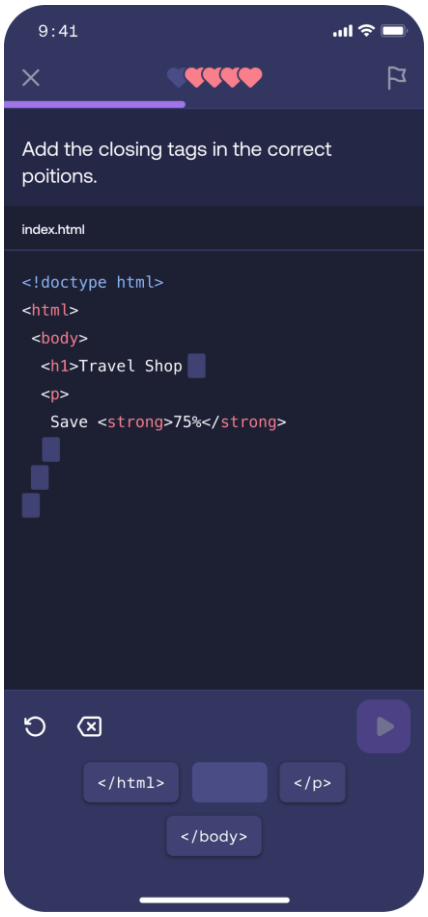


Рисунок 2.6 – Приклад тесту з перетягуванням блоків

Ще однією важливою методикою є завдання на доповнення коду (Fill in the blanks) (див. рис. 2.7)

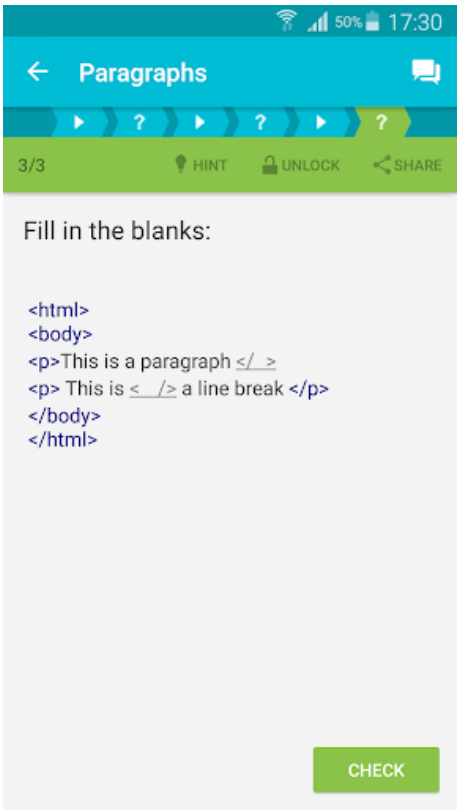


Рисунок 2.6 – Вигляд режиму Fill in the blanks

Цей підхід займає проміжне місце між пасивним вибором відповіді та активним написанням коду. Користувачеві демонструється фрагмент програми з пропущеними ключовими елементами — операторами, назвами змінних, типами даних або викликами функцій.

Для мови Kotlin це особливо актуально при вивченні специфічних конструкцій, таких як Null Safety (оператори ?., !!, ?:) або Scope Functions (let, run, apply). Головною перевагою є необхідність активного відтворення знань з пам'яті, що сприяє кращому засвоєнню матеріалу порівняно з простим тестуванням.

Проте, реалізація такого методу на мобільних пристроях вимагає обережності: необхідність ручного введення тексту може призвести до пригнічення мотивації користувача через помилки автокорекції або незручність віртуальної клавіатури, тому часто введення замінюють вибором готового блоку.

Окремо варто виділити методику аналізу та налагодження коду[7, 8]. В рамках цього підходу студенту пропонується проаналізувати фрагмент коду та передбачити результат його виконання або знайти помилку. Це розвиває критично важливу

навичку "читання коду", яка у професійній діяльності розробника займає навіть більше часу, ніж написання нового.

Перевагою є можливість перевірки глибинного розуміння механізмів роботи платформи Android без необхідності писати великі обсяги тексту. Недоліком є висока інтелектуальна складність для новачків, що може призвести до втрати мотивації на ранніх етапах навчання.

Узагальнюючи аналіз методик, можна зробити висновок, що для мобільного додатку, орієнтованого на вивчення Kotlin, використання лише одного методу тестування є недостатнім. Найбільш ефективною стратегією є комбінований підхід.

Використання тестів закритого типу доцільне для перевірки теоретичного базису, задачі Парсона оптимально підходять для закріплення алгоритмічного мислення в умовах сенсорного інтерфейсу, а завдання на доповнення коду дозволяють перевірити знання синтаксису.

Така гібридна модель дозволяє нівелювати недоліки окремих методик та забезпечити комплексну перевірку знань, адаптовану до технічних обмежень мобільних пристроїв.

2.3. Дослідження специфіки вивчення Kotlin: ключові концепції та типові складнощі

Мова програмування Kotlin, яка на сьогодні є офіційним стандартом для розробки під операційну систему Android [9], характеризується унікальною парадигмою, що поєднує об'єктно-орієнтований та функціональний підходи.

Специфіка вивчення Kotlin суттєво відрізняється від класичних мов, таких як Java або C++, оскільки вона вимагає від розробника не лише засвоєння синтаксичних конструкцій, але й зміни способу мислення в бік лаконічності та безпеки коду. Розуміння цих особливостей є критичним для розробки ефективних тестових завдань, оскільки тести повинні перевіряти глибинне розуміння механізмів мови, а не лише поверхневі знання.

Фундаментальною концепцією, що визначає специфіку Kotlin [10] і водночас становить одну з найбільших труднощів для початківців, є система обробки нульових посилань, відома як Null Safety [11]. На відміну від Java, де помилка NullPointerException виникає під час виконання програми, Kotlin переносить вирішення цієї проблеми на етап компіляції. Це досягається через суворий поділ типів даних на ті, що можуть приймати значення null (nullable), і ті, що не можуть (non-nullable).

Для студента, який раніше працював з мовами зі слабкою типізацією або старими версіями Java, необхідність постійного явного контролю за станом змінних створює додаткове навантаження. Складність полягає у правильному використанні операторів безпечного виклику, евіс-оператора та тверджень, а також у розумінні того, коли використання !! (not-null assertion operator) є виправданим, а коли це свідчить про поганий дизайн коду. Тестові завдання в цьому контексті повинні фокусуватися на здатності учня прогнозувати поведінку компілятора при роботі з потенційно пустими об'єктами.

Kotlin позиціонується як прагматична та лаконічна мова[12], що дозволяє значно скоротити обсяг шаблонного коду. Використання класів даних (data classes), властивостей замість полів з геттерами/сеттерами, а також функцій розширення робить код виразним, але часто приховує реальну логіку виконання від новачка.

Значний виклик при вивченні Kotlin для Android-розробки становить тема асинхронного програмування, реалізована через механізм корутин (Coroutines). Це концептуально складніша тема [13], ніж класичні потоки. Корутини вводять поняття «призупинення» виконання функції без блокування потоку, що є неочевидним для тих, хто звик до лінійного виконання коду. Типовою помилкою студентів є неправильне керування життєвим циклом корутин, що може призводити до витоків пам'яті в мобільних додатках. Відповідно, тести з цієї теми мають перевіряти вміння правильно обирати контекст виконання та обробляти винятки в асинхронних блоках.

Специфіка Kotlin полягає у широкому використанні елементів функціонального програмування [14], таких як лямбда-вирази та функції вищого

порядку, особливо при роботі з колекціями. Перехід від імперативного стилю обробки даних (цикли for, while) до декларативного (ланцюжки викликів map, filter, reduce) часто викликає труднощі у новачків, яким важко візуалізувати потік даних через низку перетворень.

Студенти можуть писати код, який працює, але є неефективним з точки зору продуктивності, створюючи зайві проміжні об'єкти. Тому важливо, щоб процес навчання та тестування охоплював не лише коректність результату, але й оптимальність обраного алгоритмічного підходу.

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1 Порівняльний аналіз архітектурних підходів: BaaS проти традиційної клієнт-серверної архітектури

При виборі архітектурного фундаменту для сучасної мобільної системи, особливо такої динамічної, як навчальний тренажер, ми фактично обирається спосіб, у який додаток буде обмінюватися даними.

Традиційна клієнт-серверна модель, яка десятиліттями була стандартом індустрії, передбачає створення окремого проміжного шару - бекенду. У такому сценарії розробнику доводиться не лише проектувати мобільний інтерфейс, а й власноруч будувати всю логіку системи яку користувач навіть не бачить: налаштовувати сервери, писати API-ендпоінти та вручну реалізовувати протоколи на кшталт WebSockets для того, щоб дані на екрані оновлювалися без примусового перезавантаження. Це надійний, але досить «важкий» шлях, який часто зміщує фокус із розробки унікального користувацького досвіду на рутинну підтримку інфраструктури.

На противагу цьому, концепція Backend-as-a-Service, втілена в Google Firebase, пропонує принципово іншу парадигму, де замість розробки серверної частини з нуля, використовуються високотехнологічні рішення готової платформи, що значно впливає на швидкість як розробки, та і роботи системи в цілому.

Для даного проекту це є стратегічним рішенням, оскільки Firebase дозволяє мобільному додатку взаємодіяти з хмарним сховищем Firestore напряму, оминаючи зайві посередницькі ланки. Це не просто спрощує код, а докорінно змінює те, як користувач відчуває програму. Завдяки нативній підтримці реактивних потоків даних, можлива реалізація інтерфейсу який відгукує його дії.

Коли студент завершує тест, його новий XP-рейтинг або отримана галочка за тему не просто записуються в базу - вони миттєво з’являються на всіх екранах

завдяки механізму SnapshotListeners, створюючи відчуття безперервного ігрового процесу.

З технічної точки зору, використання BaaS також дозволяє переосмислити підхід до безпеки та масштабування. Замість написання складних серверних перевірок, використовуються декларативні правила безпеки безпосередньо в консолі Firebase, які на рівні ядра бази даних контролюють, щоб кожен користувач мав доступ лише до свого прогресу.

Водночас гнучка NoSQL-структура Firestore дає змогу легко оперувати різними типами завдань — від класичних тестів до складних алгоритмічних задач Парсона без необхідності постійної перебудови схеми реляційних таблиць.

Таким чином, використання Firebase стає не просто способом прискорити розробку, а технічним інструментом, що дозволив зосередитися на головному - створенні інтерактивного навчального середовища, яке реагує на дії користувача миттєво і безвідмовно.

3.2 Проектування архітектури застосунку на засадах Clean Architecture та паттерну MVVM

Фундаментом розробки інтерактивної системи тестування став комплексний архітектурний підхід, що базується на синтезі принципів Clean Architecture [15] та шаблону проектування Model-View-ViewModel. Такий вибір зумовлений необхідністю побудови системи, що характеризується високим рівнем масштабованості, незалежністю від зовнішніх фреймворків та можливістю ефективного юніт-тестування окремих модулів.

Основною ідеєю Clean Architecture у межах даного проекту є поділ програмного забезпечення на незалежні рівні, кожен з яких має чітко визначену зону відповідальності та взаємодіє з іншими через суворо регламентовані інтерфейси.

Центральне місце в ієрархії системи посідає доменний рівень, який є абсолютно автономним і не містить посилань на деталі реалізації, такі як бази даних

чи особливості Android-платформи [16, 17]. Він інкапсулює ключову бізнес-логіку застосунку, включаючи сутності - наприклад, моделі технологій та питань - та сценарії використання Use Cases, що описують конкретні дії користувача в межах системи. Такий підхід забезпечує стабільність ядра програми: будь-які зміни в зовнішніх сервісах, наприклад, перехід на іншу систему зберігання даних, не впливають на логіку перевірки відповідей чи алгоритми розрахунку досвіду користувача.

Рівень даних відповідає за взаємодію із зовнішніми джерелами інформації, зокрема хмарною базою даних Firestore. Реалізація цього шару базується на патерні «Репозиторій», який виступає абстрактним посередником між джерелами даних та доменним рівнем.

Репозиторій приховує складність мережевих запитів та специфіку набору інструментів розробки Firebase, перетворюючи «сирі» документи з хмари у чисті доменні моделі, з якими зручно працювати іншим компонентам системи.

Презентаційний рівень реалізований за допомогою паттерну MVVM, який ідеально інтегрується з декларативним підходом Jetpack Compose. ViewModel відіграє роль зв’язку, який отримує дані від доменного рівня, обробляє їх та перетворює у стан інтерфейсу (UI State). Використання реактивних потоків даних StateFlow дозволяє ViewModel асинхронно транслювати зміни стану до UI-компонентів [13], забезпечуючи коректне відображення таймерів, прогрес-барів та списків питань у реальному часі.

Такий розподіл обов'язків гарантує, що інтерфейс залишається простим і відповідає лише за візуалізацію даних та передачу дій користувача, тоді як вся логіка керування станом зосереджена у ViewModel.

Зв’язуючою ланкою між усіма рівнями виступає механізм впровадження залежностей Dependency Injection на базі бібліотеки Hilt. Це дозволяє автоматизувати створення та життєвий цикл компонентів, зменшуючи жорстку зв'язність між ними та спрощуючи заміну реальних репозиторіїв на їхні тестові аналоги під час перевірки системи. У результаті взаємодія Clean Architecture та MVVM забезпечує створення надійного програмного продукту, де технічні

складності інтеграції з хмарними сервісами чітко відокремлені від освітньої логіки та користувацького інтерфейсу.

3.3 Розробка схеми даних у NoSQL системі Firestore: стратегії денормалізації та безпеки

Проектування структури даних у системі Cloud Firestore вимагає перегляду традиційних підходів до моделювання [18], притаманних реляційним базам даних.

Основною особливістю обраної NoSQL-платформи є її документо-орієнтована природа, де дані організовуються в ієрархічні структури колекцій та документів. На відміну від реляційних систем, де акцент робиться на мінімізації дублювання через нормалізацію, у Firestore пріоритет надається швидкості зчитування та масштабованості.

Це досягається за рахунок стратегій денормалізації, коли дані групуються за принципом їхнього безпосереднього використання в інтерфейсі застосунку, що дозволяє уникнути високовартісних операцій об'єднання на боці клієнта.

В основі архітектури даних застосунку лежать три ключові колекції: technologies, questions та users. Колекція technologies виконує роль метаданих, зберігаючи інформацію про назви мов програмування, посилання на графічні ресурси та загальний опис, що необхідно для формування головного екрана Dashboard.

Колекція questions є найбільш об’ємною та містить банк тестових завдань, де кожен документ ідентифікується за унікальним id та містить атрибути techId та topic для ефективної фільтрації. Особливу увагу приділено поліморфній структурі питань, де в межах одного типу даних можуть співіснувати як класичні тести з вибором відповіді (multiple_choice), так і алгоритмічні завдання (parsons_puzzle), що вимагають специфічних полів, таких як масиви блоків коду та правильна послідовність їх впорядкування.

Для забезпечення високої продуктивності мобільного інтерфейсу була застосована стратегія денормалізації стосовно профілів користувачів. Замість того,

щоб щоразу обчислювати загальний прогрес шляхом запитів до всієї історії тестування, критичні показники, як-от totalXP та масив ідентифікаторів успішно пройдених питань passedQuestions, зберігаються безпосередньо в основному документі користувача в колекції users. Це дозволяє презентаційному рівню застосунку миттєво отримувати актуальний стан навчання одним запитом.

Детальна ж історія активності винесена в окрему підколекцію progress, що забезпечує логічну ізоляцію великих обсягів даних від основної інформації профілю, запобігаючи перевищенню лімітів розміру документа.

Невід'ємною частиною проектування схеми NoSQL є розробка декларативних правил безпеки Security Rules, які в архітектурі BaaS виконують роль серверного захисту даних. Оскільки доступ до бази здійснюється безпосередньо з клієнтського боку, правила безпеки регламентують права доступу на основі ідентифікатора авторизованого користувача request.auth.uid.

Це гарантує цілісність системи: користувач має повний доступ до читання та запису у власному профілі та підколекціях прогресу, проте обмежений лише правом читання для глобальних колекцій технологій та питань.

Такий підхід не лише захищає конфіденційну інформацію, а й виступає додатковим рівнем валідації типів даних на боці сервера, забезпечуючи стабільну роботу системи в умовах відсутності традиційного серверного бекенду.

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1 Проектування структури додатку: навігація, модулі та компоненти системи

Проектування програмної архітектури мобільного додатку є фундаментальним етапом розробки, який визначає не лише стабільність та швидкодію кінцевого продукту, але й можливості його подальшого масштабування та підтримки. Для реалізації системи інтерактивного тестування було обрано сучасний архітектурний підхід, рекомендований компанією Google, який базується на принципах «чистої архітектури» [15] та використанні патерну MVVM (див. рис. 4.1) [19].

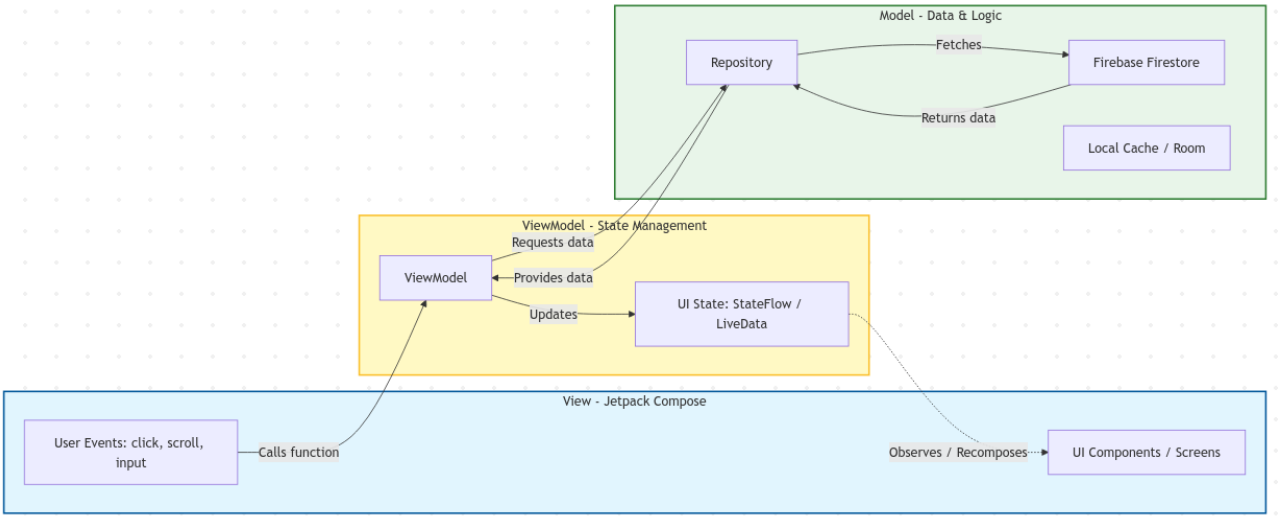


Рисунок 4.1 – Діаграма архітектури MVVM

Цей вибір обумовлений необхідністю чіткого розмежування відповідальності між компонентами системи: логіка роботи з даними, бізнес-правила та інтерфейс користувача повинні функціонувати незалежно одне від одного, що дозволяє локалізувати потенційні помилки та спрощує процес тестування окремих модулів.

Центральним елементом обраної архітектури є патерн MVVM, який ідеально інтегрується з декларативним UI-фреймворком Jetpack Compose [20]. Взаємодія між компонентами будується на основі односпрямованого потоку даних [13]. Нижній

рівень архітектури Model відповідає за абстракцію даних. У розроблюваному додатку цей шар реалізовано через патерн Repository, який виступає єдиною точкою доступу до даних, інкапсулюючи логіку взаємодії з хмарною базою Firebase Firestore. Репозиторій отримує запити від верхніх рівнів, завантажує необхідні тестові завдання або відправляє результати користувача на сервер, повертаючи вже оброблені об'єкти даних, очищені від мережевих деталей.

Сполучною ланкою системи виступає компонент ViewModel. Його ключова роль полягає у збереженні та управлінні станом інтерфейсу UI State. Оскільки мобільні пристрої часто змінюють свою конфігурацію (наприклад, при повороті екрану або зміні теми оформлення), звичайні контролери можуть втрачати дані. ViewModel вирішує цю проблему, зберігаючи життєвий цикл незалежно від відображення.

Саме тут зосереджена основна логіка презентації: ViewModel отримує «сирі» дані з репозиторію, перетворює їх у зручний для відображення формат та реагує на дії користувача (наприклад, натискання кнопки «Перевірити»), ініціюючи відповідні бізнес-процеси.

Верхній рівень View реалізований засобами Jetpack Compose і є повністю пасивним. Його завдання зводиться виключно до візуалізації стану, отриманого від ViewModel. Завдяки декларативній природі Compose, розробнику не потрібно вручну оновлювати елементи інтерфейсу при зміні даних - інтерфейс автоматично перемальовується відповідно до актуального стану моделі. Такий підхід робить код інтерфейсу лаконічним, читабельним та стійким до помилок, пов'язаних з неузгодженістю стану відображення та реальних даних.

Організація навігації в додатку базується на концепції Single Activity Architecture [21]. Замість створення окремих важких вікон (Activities) для кожного екрану, додаток використовує один контейнер, всередині якого відбувається динамічна заміна контенту за допомогою бібліотеки Navigation Compose. Граф навігації проектує шлях користувача як послідовність переходів між вузловими точками: від екрану авторизації до головного меню Dashboard, далі до налаштувань параметрів тестування, безпосередньо ігрової сесії та фінального екрану результатів.

Така структура забезпечує плавні анімації переходів та ефективне використання системних ресурсів.

З точки зору організації кодової бази, проект структуровано за модульним принципом, що відображає логічні шари додатку. Виділяються окремі пакети для моделей даних, бізнес-логіки, елементів інтерфейсу та допоміжних утиліт. Така гранулярність дозволяє розробнику фокусуватися на конкретній задачі, не втручаючись у роботу суміжних компонентів, що є критично важливим для підтримки чистоти та якості коду в довгостроковій перспективі.

4.2 Розробка концепції користувацького інтерфейсу та сценаріїв взаємодії

Візуальна концепція додатку базується на настановних принципах системи Material Design 3 (MD3) [22]. Цей вибір забезпечує уніфікацію інтерфейсних рішень зі стандартами платформи Android, що гарантує інтуїтивну зрозумілість навігації для користувача. Пріоритетною схемою оформлення обрано темну тему, оскільки цільовою аудиторією є розробники, для яких темний фон є звичним робочим середовищем, що знижує зорове навантаження при читанні коду.

Організація навігації та структура екранів Навігаційна модель додатку реалізована через нижню панель (Bottom Navigation Bar), яка забезпечує швидкий доступ до трьох ключових розділів: «Навчання», «Тести», «Статистика» та «Профіль».

1. Головний екран (Dashboard) (див. рис. 4.2): Спроектований як стрічка карток з темами. Кожна картка містить візуальний індикатор прогресу Linear Progress Indicator, що дозволяє миттєво оцінити завершеність модуля.

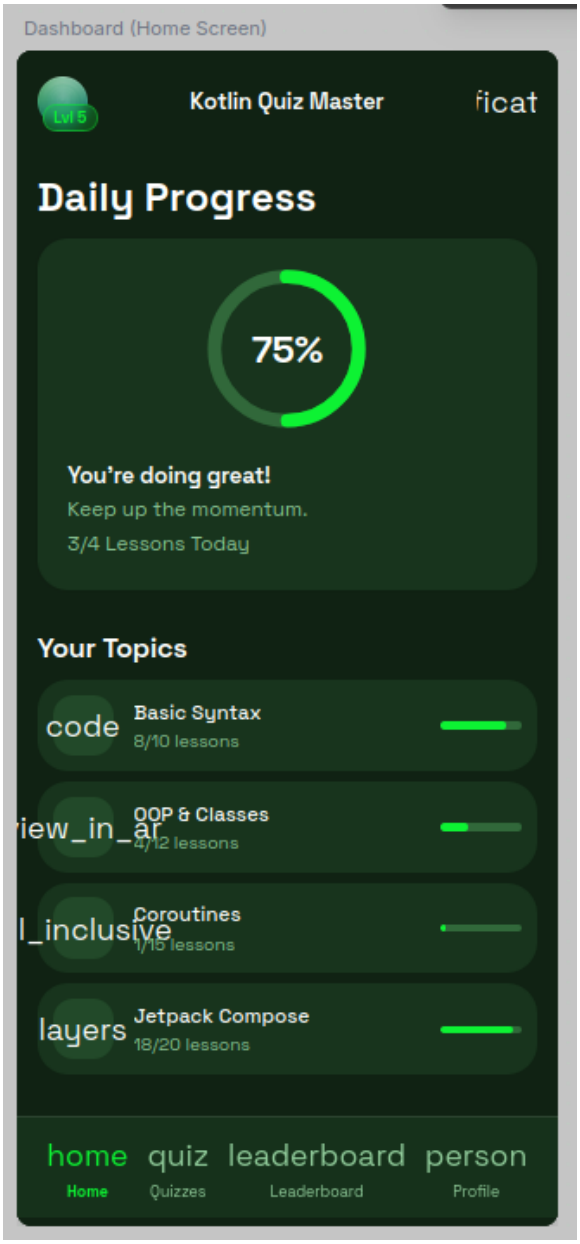


Рисунок 4.2 – Екран Dashboard

2. Екран тестування (див. рис. 4.3): Це центральний елемент взаємодії. Щоб мінімізувати відволікаючі фактори, на цьому екрані приховуються системні панелі. Верхню частину займає таймер і шкала прогресу. Центральну зону відведено під блок коду, для якого використовується моноширинний шрифт (наприклад, JetBrains Mono) із синтаксичним підсвічуванням ключових слів Kotlin.

Рисунок 4.3 – Екран тестування

3. Зона взаємодії**:** Кнопки з варіантами відповідей розташовані в нижній третині екрану — в так званій «зоні комфорту великого пальця», що дозволяє зручно проходити тести однією рукою на пристроях з великою діагоналлю.

Сценарії зворотного зв'язку: Система реагує на дії користувача за допомогою візуальних та тактильних сигналів.

- При виборі відповіді: Картка варіанту миттєво змінює колір контейнера на зелений або червоний, що супроводжується короткою вібрацією різної тональності.
- Екран завершення тестування(див. рис. 4.4): по завершенню тестування користувач побачить екран з підсумком на якому буде відсоток правильних відповідей, нараховані «XP», пропозиції перейти на наступний тест або переглянути помилки (якщо вони були).

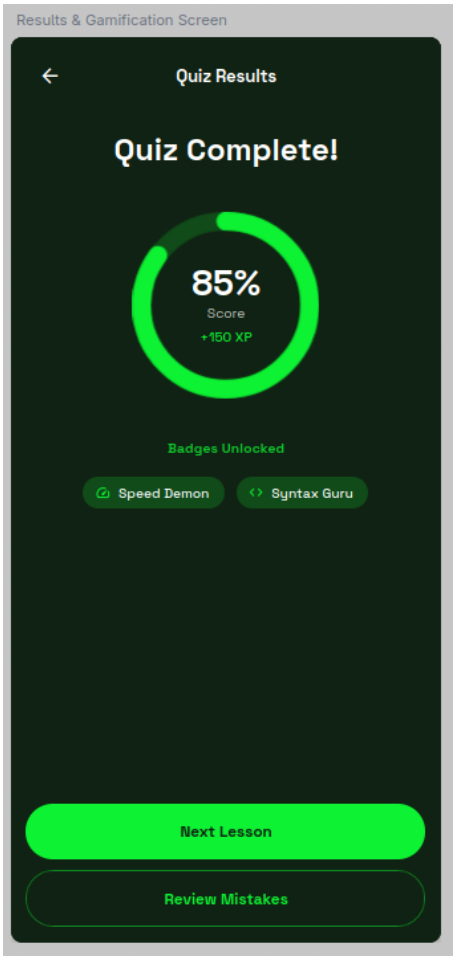


Рисунок 4.4 – Екран завершення тестування

- Робота над помилками: У випадку неправильної відповіді використовується компонент Modal Bottom Sheet (висувна панель знизу). Це дозволяє показати пояснення помилки поверх основного екрану, не розриваючи контекст завдання і не змушуючи користувача переходити на нову сторінку.

Адаптивність та доступність Усі інтерактивні елементи спроектовані з дотриманням вимог доступності: мінімальний розмір області натискання становить 48x48 dp, що запобігає помилковим клікам. Інтерфейс підтримує динамічне масштабування тексту, гарантуючи читабельність коду навіть для користувачів із вадами зору.

4.3 Реалізація хмарної інфраструктури та сервісів Firebase

Процес впровадження хмарної інфраструктури розпочато з конфігурування проекту в консолі Google Firebase [23] (див. рис. 4.5) та його інтеграції з Android-застосунком через ідентифікацію пакетів та налаштування файлів конфігурації сервісів.

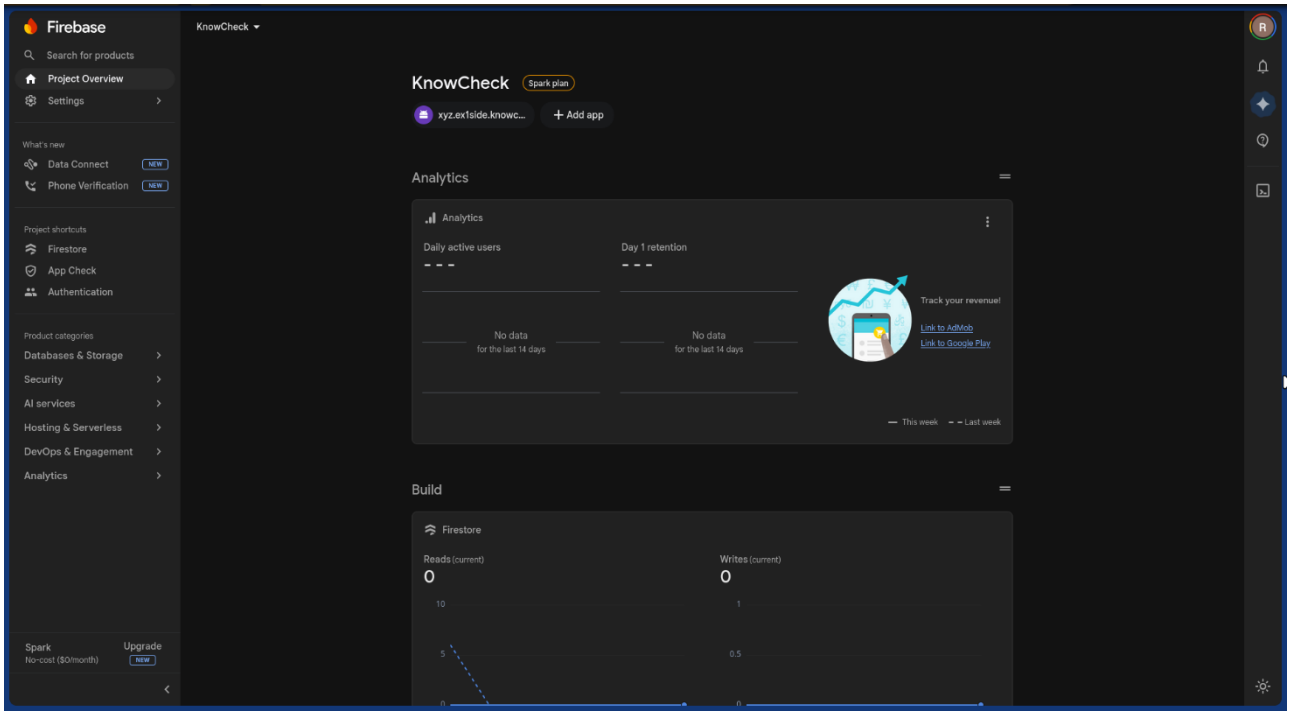


Рис. 4.5 – Головна панель Firebase

Основним критерієм при виборі сервісів була необхідність забезпечення безперебійної автентифікації та швидкої синхронізації даних між сервером і клієнтом.

У межах реалізації модуля авторизації було налаштовано сервіс Firebase Authentication, що забезпечує делегування процесу перевірки особистості провайдеру Google Identity. Такий підхід дозволив інтегрувати систему захищеного входу без необхідності збереження паролів на боці застосунку, що підвищує рівень конфіденційності даних користувачів, а також зручності оскільки це пришвидшує процес реєстрації та кожен подальший вхід у обліковий запис, без необхідності запам’ятовувати пароль.

Процес авторизації синхронізовано з базою даних Firestore (див. рис. 4.6):

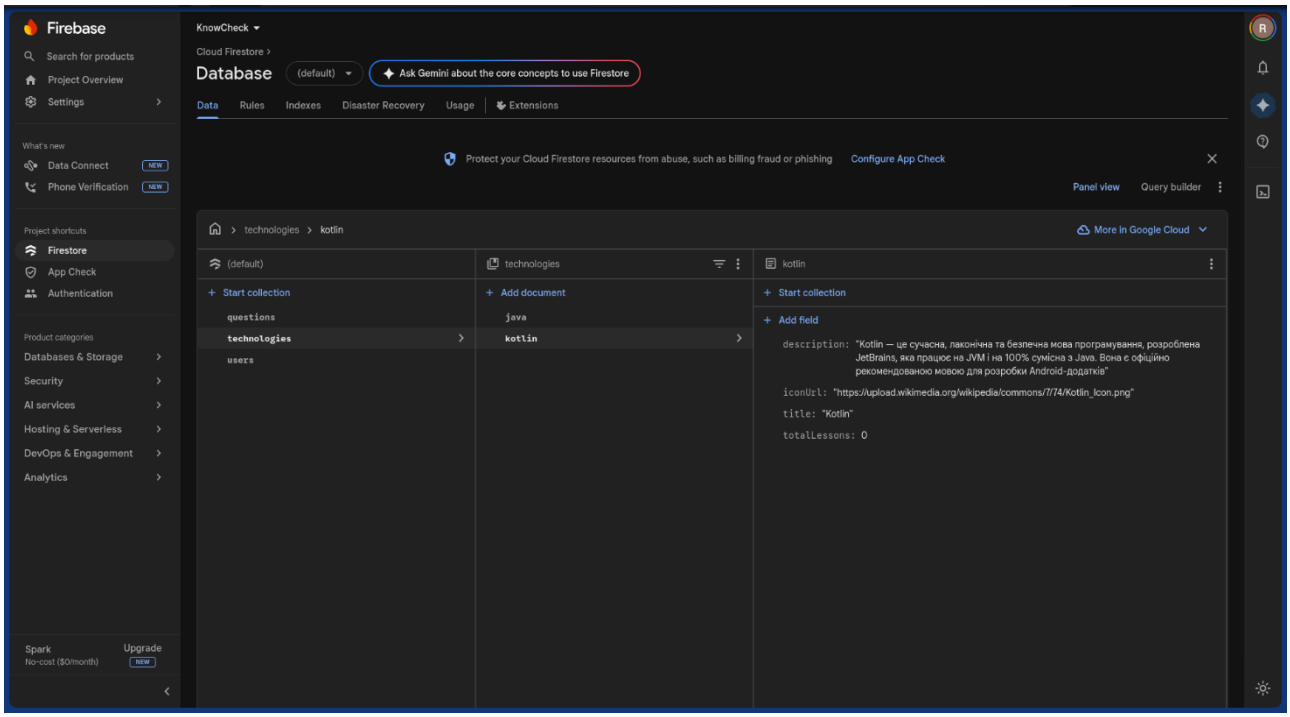


Рис. 4.6 – Вигляд панелі Firestore

При першому вході автоматично ініціалізується документ користувача в колекції users (див. рис. 4.7), де зберігаються його персональні налаштування та початкові показники прогресу.

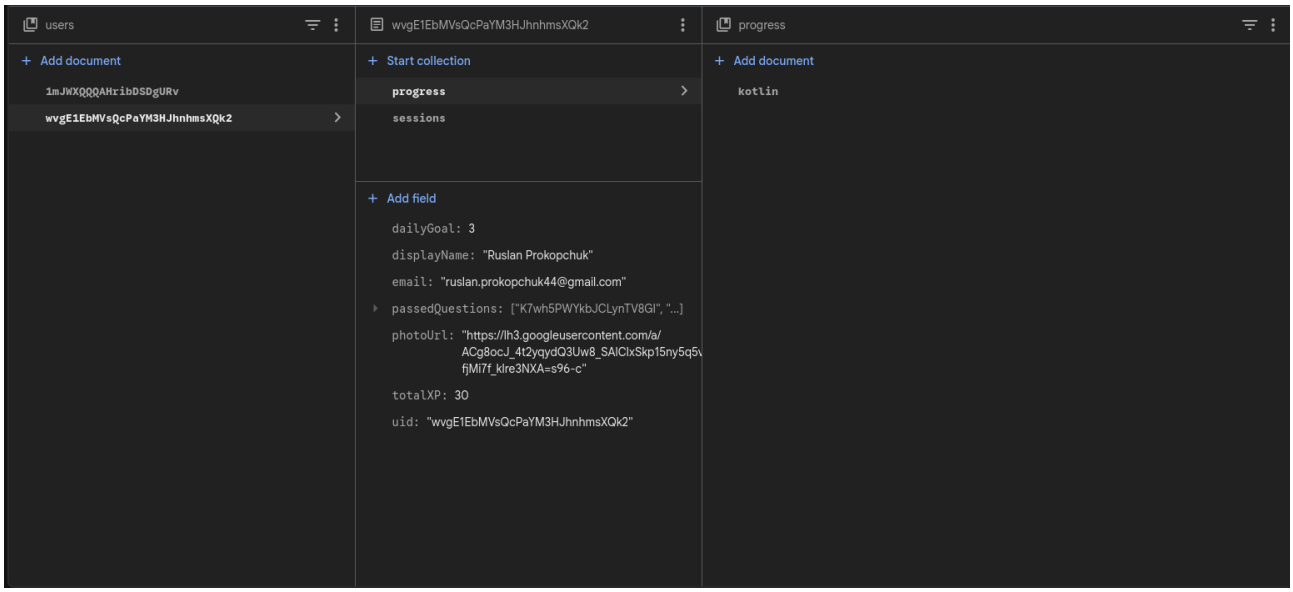


Рис. 4.7 – Вигляд колекції users у панелі Firestore

Центральним компонентом інфраструктури є розгортання бази даних Cloud Firestore.

Для забезпечення цілісності даних на боці хмарного сервера прописано декларативні правила безпеки Security Rules (див. рис. 4.8).

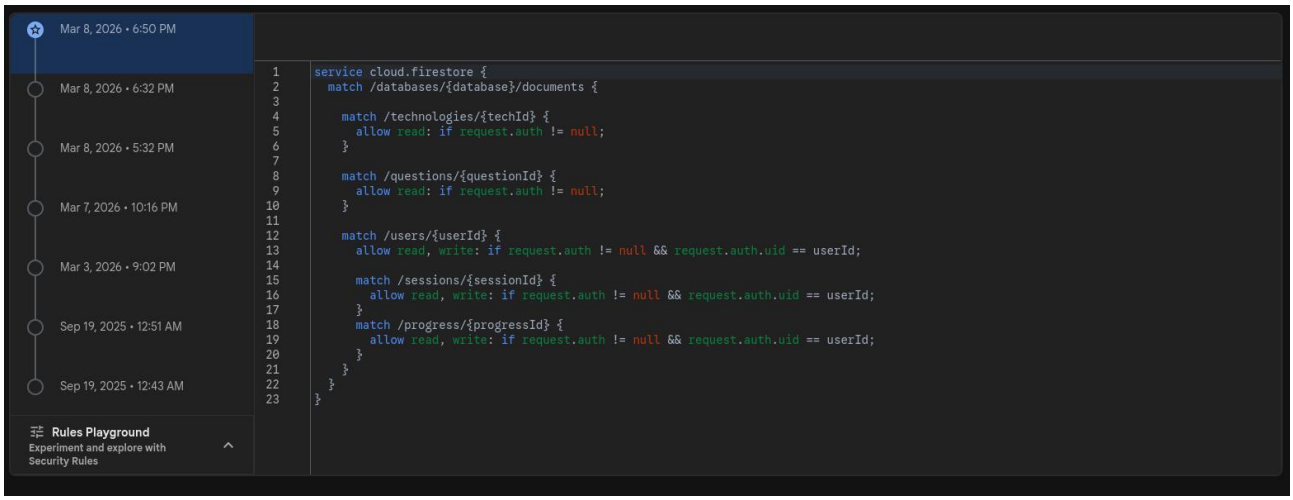


Рис 4.8 – Редактор Security Rules

Зокрема, встановлено обмеження, які дозволяють виконувати операції запису та модифікації даних у колекції users та підколекціях progress виключно авторизованим власникам цих документів за їхніми унікальними ідентифікаторами (UID).

Така конфігурація інфраструктури (див. рис. 4.9) виключає можливість несанкціонованого доступу до приватних результатів тестування та гарантує стабільну роботу системи в умовах відсутності традиційного серверного бекенду.

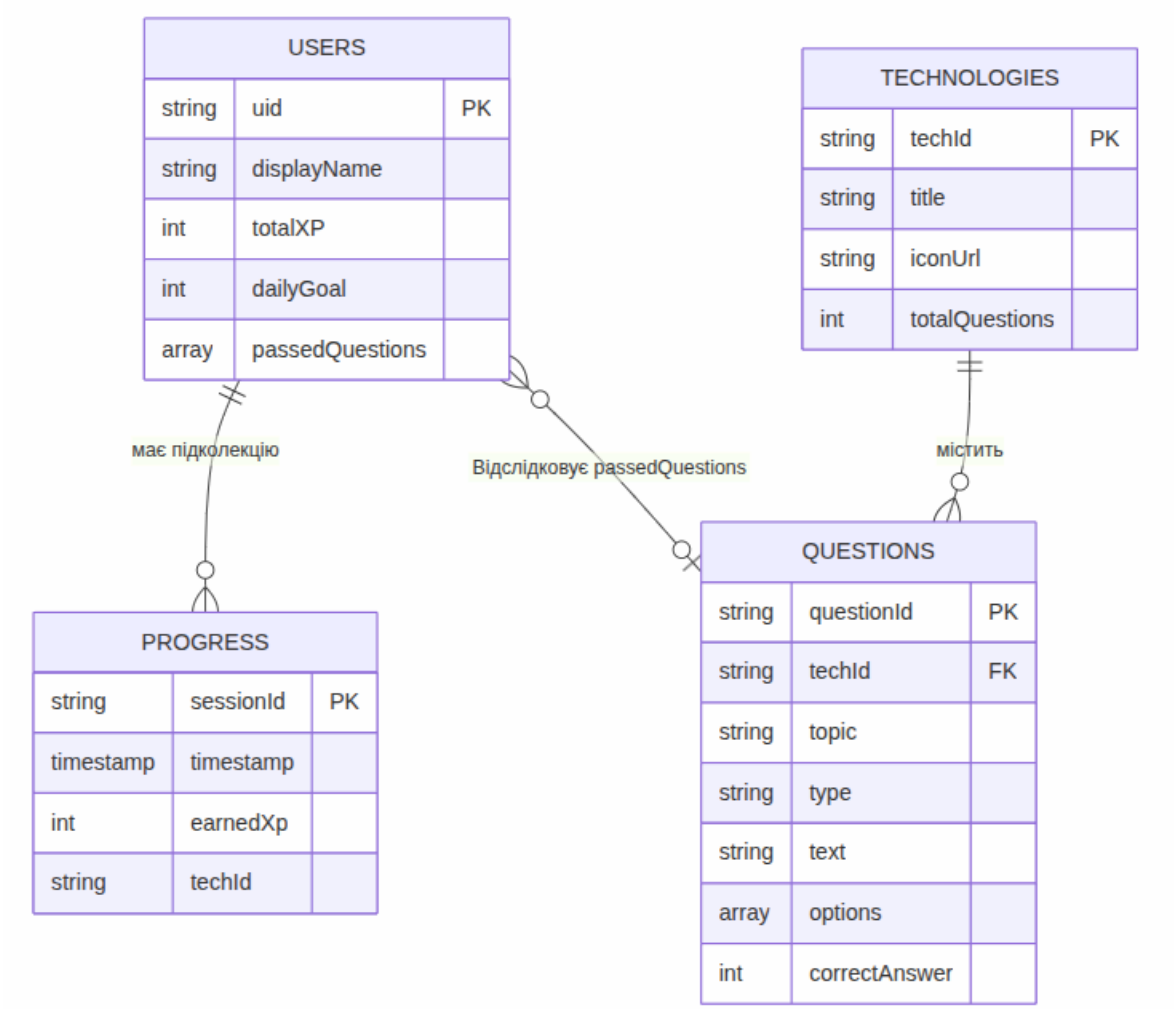


Рис. 4.9 – Структура бази даних Firebase

4.4 Технічна реалізація мобільного застосунку

Головним інструментом розробки Android застосунку є Android Studio (див. рис. 4.10) – офіційна IDE від Google яка містить всі необхідні можливості для розробки та тестування застосунків під час розробки.

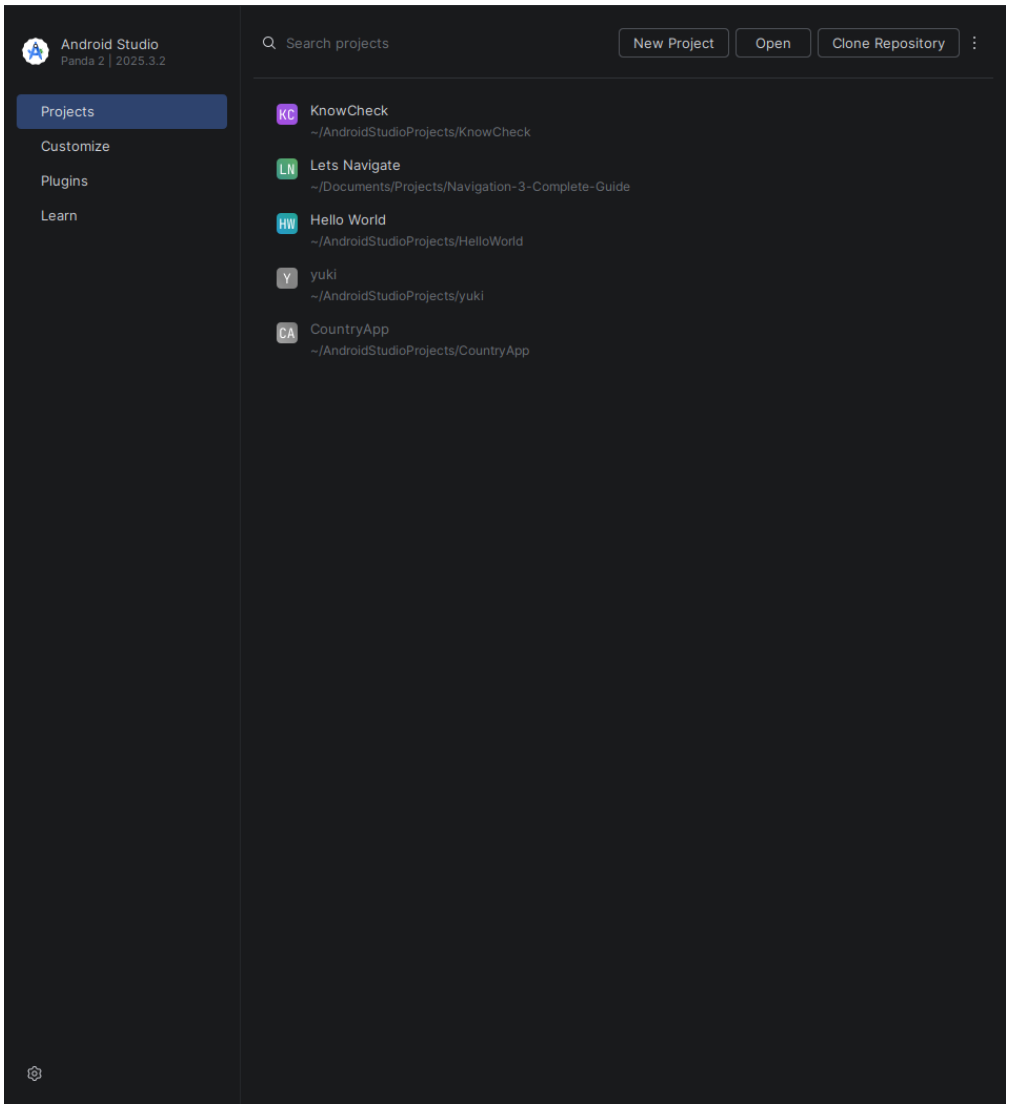
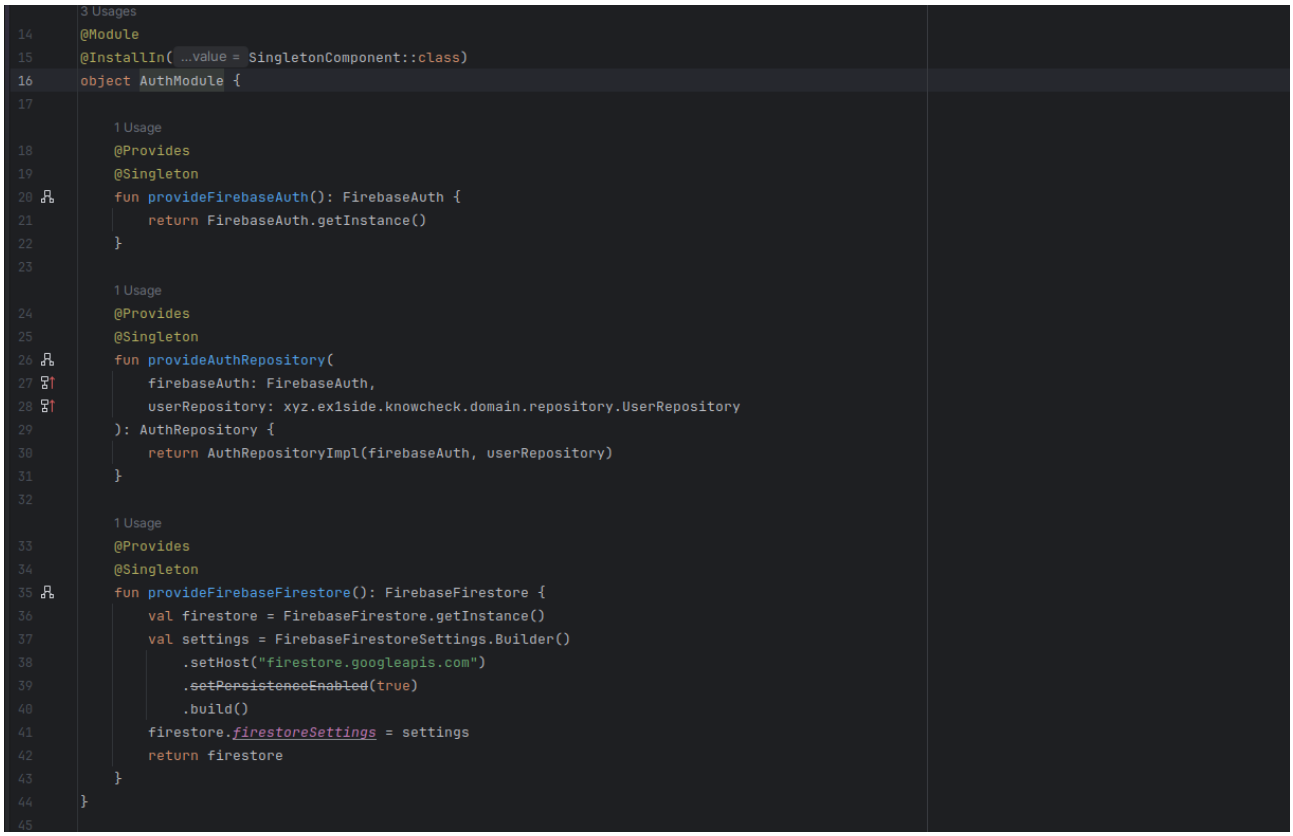


Рис 4.10 – Головний екран Android Studio

Технічну реалізації системи тестування знань було розділено на кілька послідовних етапів, кожен з яких спрямований на забезпечення стабільності, реактивності та масштабованості програмного продукту.

Побудова архітектурного каркаса. На початковому етапі розробки було впроваджено систему впровадження залежностей (Dependency Injection) за допомогою бібліотеки Dagger Hilt (див рис. 4.11). Це дозволило реалізувати централізоване керування життєвим циклом об'єктів та забезпечити надання екземплярів Firestore SDK у репозиторії без жорсткої прив’язки до контексту.



```
3 Usages
@Module
@InstallIn( ...value = SingletonComponent::class)
object AuthModule {

    1 Usage
    @Provides
    @Singleton
    fun provideFirebaseAuth(): FirebaseAuth {
        return FirebaseAuth.getInstance()
    }

    1 Usage
    @Provides
    @Singleton
    fun provideAuthRepository(
        firebaseAuth: FirebaseAuth,
        userRepository: xyz.ex1side.knowcheck.domain.repository.UserRepository
    ): AuthRepository {
        return AuthRepositoryImpl(firebaseAuth, userRepository)
    }

    1 Usage
    @Provides
    @Singleton
    fun provideFirebaseFirestore(): FirebaseFirestore {
        val firestore = FirebaseFirestore.getInstance()
        val settings = FirebaseFirestoreSettings.Builder()
            .setHost("firestore.googleapis.com")
            .setPersistenceEnabled(true)
            .build()
        firestore.firestoreSettings = settings
        return firestore
    }
}
```

Рис 4.11 – Приклад модулю ін’єкції авторизації написаного з використанням Dagger Hilt

На рівні даних реалізовано репозиторії, які за допомогою Kotlin Coroutines виконують асинхронні запити до Firestore, перетворюючи результати в реактивні потоки StateFlow.

Структуру проекту було розділено на три функціональні шари [24] згідно з принципами Clean Architecture: data, domain та presentation (див. рис 4.12). Така декомпозиція (див. рис. 4.13) дозволила ізолювати бізнес-правила (Use Cases) від деталей реалізації інтерфейсу та мережевої взаємодії.

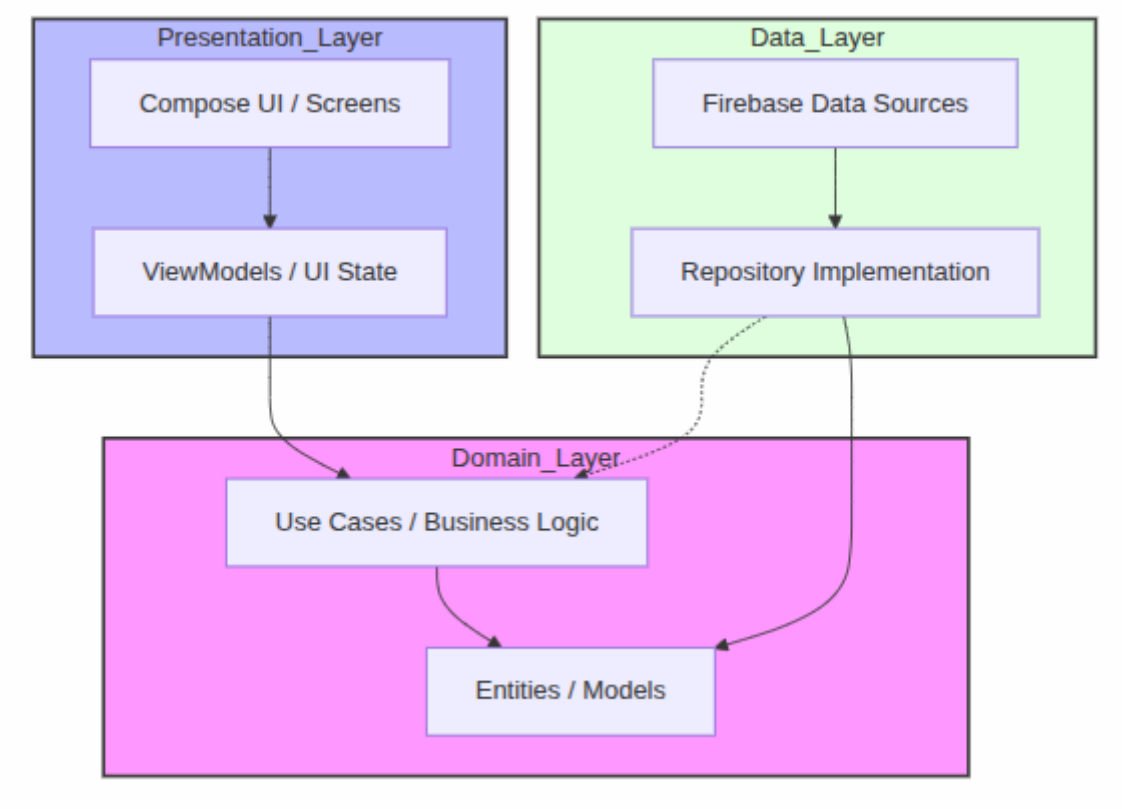


Рис 4.12 – Діаграма взаємодії шарів застосунку

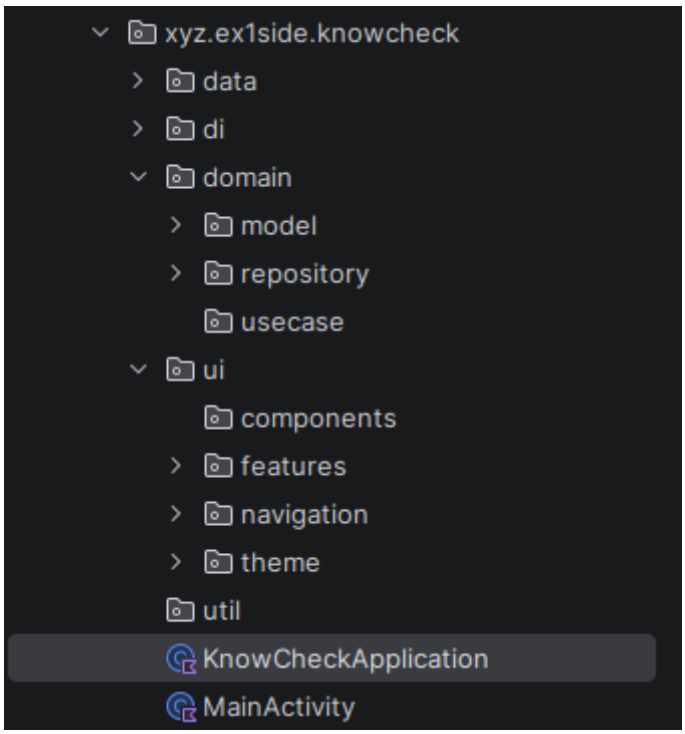


Рис 4.13 – Структура застосунку

У межах доменного рівня було спроектовано базові сутності, що описують структуру питань, технологій та профілю користувача. Для кожного функціонального сценарію (отримання списку тем, оновлення прогресу, розрахунок XP) було розроблено окремі класи UseCase. Це дозволило інкапсулювати логіку прийняття рішень (наприклад, перевірку на унікальність питання перед нарахуванням балів) у незалежні модулі, що піддаються автоматизованому тестуванню.

Реалізація рівня даних базувалася на створенні репозиторіїв, які виступають мостом між Firestore та бізнес-логікою. Для отримання даних у реальному часі було застосовано механізм callbackFlow, який трансформує подієві слухачі Firebase у реактивні потоки Kotlin Flow. На цьому етапі було впроваджено мапінг - перетворення «сирих» документів Firestore у типобезпечні об'єкти Kotlin, що зводить до нуля ризики виникнення помилок при зміні схеми даних.

Інтерфейс застосунку реалізовано за допомогою Jetpack Compose, де кожен екран є функцією стану UI State (див. рис. 4.14). Управління станом здійснюється через ViewModels, які за допомогою оператора StateFlow транслюють зміни даних від репозиторіїв до UI-компонентів. Для забезпечення плавної навігації було сконструйовано навігаційний граф, що регламентує переходи між екранами та безпечну передачу аргументів (наприклад, ID обраної технології) через маршрутизацію.

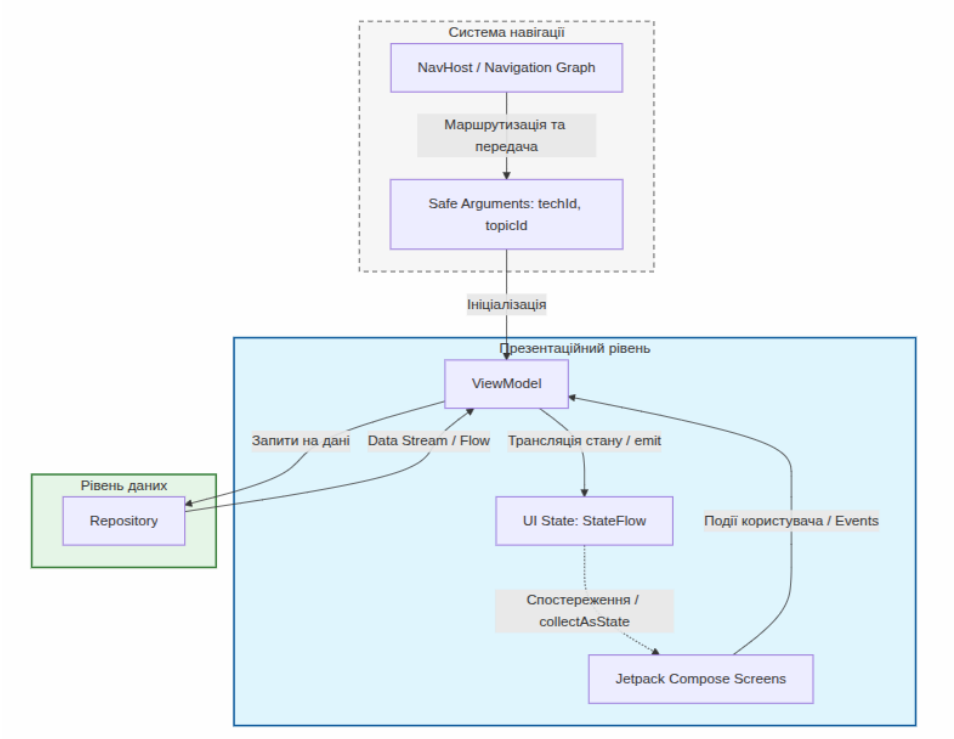


Рис. 4.14 – Діаграма взаємодії між навігацією, станом та шарами архітектури

На фінальному етапі реалізовано специфічні модулі, зокрема Parsons Puzzle. Алгоритм перевірки базується на динамічному перерахунку індексів у масиві блоків при кожній дії користувача. Також впроваджено систему автоматичного оновлення щоденного прогресу: при кожному успішному завершенні сесії виконується запит update до документа користувача, що ініціює миттєву рекомпозицію індикаторів на Dashboard.

4.5 Опис роботи застосунку для проходження тестувань

Застосунок пропонує інтуїтивно зрозумілий інтерфейс, коли користувач вперше відкриває застосунок, йому потрібно виконати вхід через обліковий запис Google. Для цього використовується Google One Tap SignIn (див. рис. 4.15), застосунок запропонує обрати один з наявних на пристрої облікових записів.

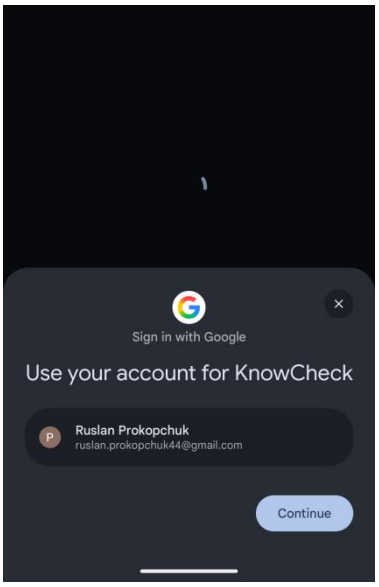


Рисунок 4.15 – Вхід у обліковий запис

Реалізований алгоритм автентифікації працює за принципом безшовної інтеграції: якщо користувач уже зареєстрований у системі, застосунок автоматично виконує вхід до існуючого профілю. У випадку першого звернення до сервісу, система ініціює фонове створення нового облікового запису в базі даних Firebase. Такий підхід дозволяє повністю автоматизувати процес ідентифікації, позбавляючи користувача необхідності проходити складні етапи реєстрації та мінімізуючи когнітивне навантаження під час взаємодії з інтерфейсом.

Після авторизації буде відкрито перенаправлено на екран Dashboard, на цьому екрані можна побачити денний прогрес, та останні мови з яких виконувалися тестування (див. рис. 4.16).

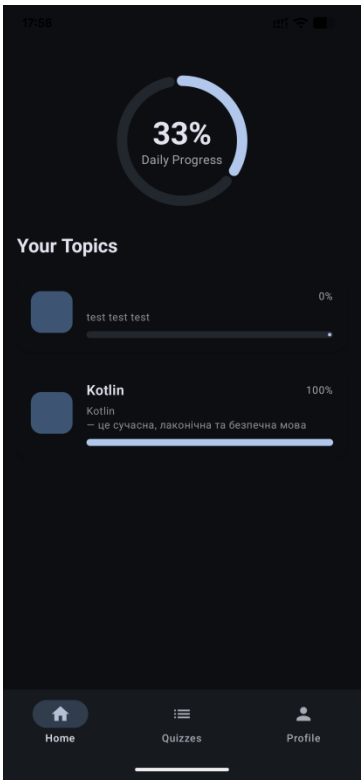


Рисунок 4.16 - Вигляд екрану Dashboard

У нижній панелі можна перемикатися між панелями як домівка (див. рис. 4.16), екраном вибору мови програмування, та профілем користувача.

Після остаточного вибору конкретного технологічного стека (наприклад, певної мови програмування) програмне забезпечення виконує перенаправлення користувача до інтерфейсу вибору тематичних модулів (див. рис. 4.17). Перелік доступних тем формується динамічно шляхом автоматичного завантаження даних із хмарної бази даних Firebase.

Для кожного елемента списку на екрані відображається детальна інформація, що включає назву теми та загальну кількість запитань у відповідному тесті. Крім того, реалізовано механізм відстеження прогресу: якщо тестування за певною темою було завершене раніше, система автоматично маркує її як пройдену, що дозволяє користувачу зручно орієнтуватися у навчальному процесі.

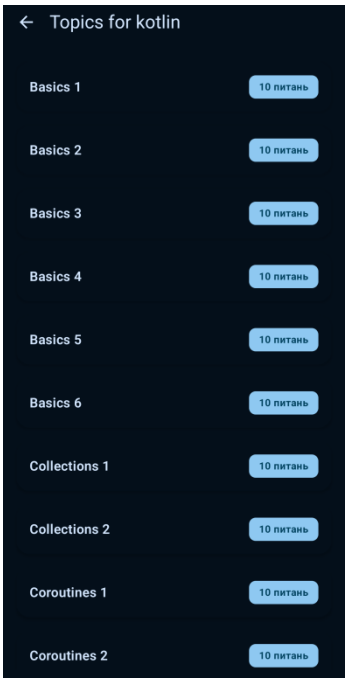


Рисунок 4.17 – Вибір теми тестування

Після обрання теми, користувач переходить безпосередньо до тестування, тести включають питання з однією або декількома варіантами відповідей, а також задачі Парсона на відновлення правильної послідовності коду (див. рис. 4.18).

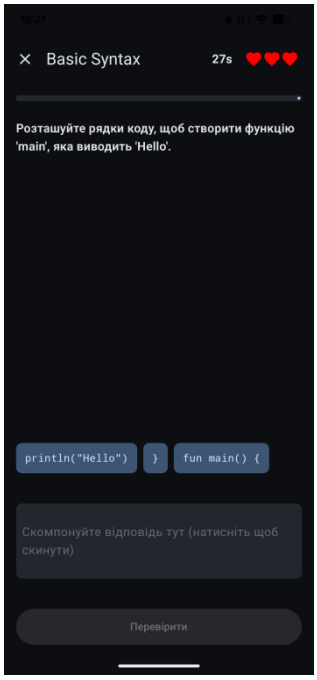


Рисунок 4.18 – Вигляд задачі Парсона

У разі якщо користувач відповів неправильно, то він може переглянути правильну відповідь та пояснення матеріалу (див. рис. 4.19)

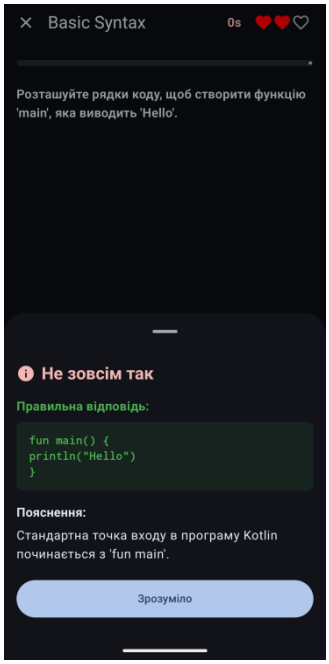


Рисунок 4.19 – Демонстрація функціоналу пояснення відповіді

Після закінчення тестування йде перенаправлення на екран завершення тестування, на ньому відображається успіх проходження у відсотковому відношенні та порада щодо повторного тестування, також якщо дано неправильні відповіді, то можна переглянути пояснення одразу для всіх запитань (див. рис. 4.20).

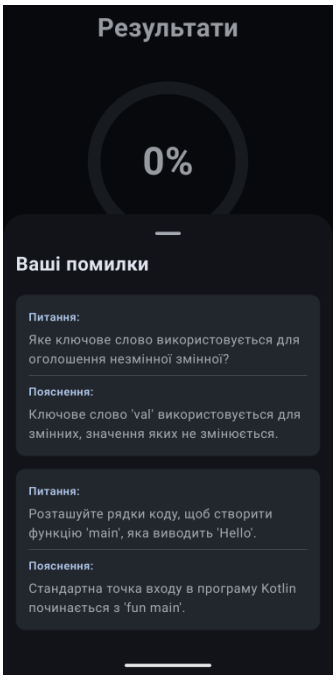


Рисунок 4.20 – Екран завершення з вікном пояснення

Таким чином, розроблений користувацький інтерфейс та функціональна логіка застосунку забезпечують цілісний, інтуїтивно зрозумілий та орієнтований на користувача (UX-centric) процес навчання. Завдяки інтеграції сучасних інструментів, таких як Google One Tap SignIn та хмарна інфраструктура Firebase, вдалося мінімізувати бар'єри на етапі входу та повністю автоматизувати управління обліковими записами.

Динамічне формування контенту, гнучка система відстеження індивідуального прогресу на панелі Dashboard, а також впровадження інтерактивних занять (зокрема, задач Парсона) підвищують рівень залученості користувачів. Окремою перевагою архітектури є розвинена система зворотного зв'язку, яка надає розгорнуті пояснення до помилок як під час, так і після завершення тестування. Це підтверджує, що спроєктована система не лише виконує функцію контролю знань, а й слугує повноцінним інструментом для ефективного самостійного навчання та аналізу помилок.

ВИСНОВКИ

За результатами дослідження можна зробити наступні висновки:

1. Аналіз предметної області: Проведено огляд ринку існуючих освітніх платформ (Sololearn, Moodle, Mimo, Duolingo). Виявлено, що більшість рішень або мають надто високий поріг входження, або, навпаки, пропонують поверхневі тести, не адаптовані до специфіки мобільної розробки. Це підтвердило актуальність створення спеціалізованого додатку для поглибленого вивчення Kotlin.

2. Методологічна база: Розроблено структуру навчального матеріалу та методику оцінювання знань. Сформовано банк тестових завдань різних типів та рівнів складності. Запропонована система оцінювання враховує не лише правильність відповідей, але й швидкість реакції, що стимулює розвиток професійних навичок.

3. Технологічний стек: Обґрунтовано вибір концепції Modern Android Development як технологічної основи проекту. Використання мови Kotlin забезпечує надійність коду, декларативний фреймворк Jetpack Compose дозволяє створити динамічний та адаптивний інтерфейс, а хмарна платформа Firebase вирішує питання зберігання даних та синхронізації прогресу користувачів без необхідності розгортання власного сервера.

4. Архітектурні рішення: Спроектовано архітектуру додатку на основі патерну MVVM (Model-View-ViewModel) та принципів Clean Architecture. Це забезпечує чітке розмежування логіки додатку та інтерфейсу, що спрощує підтримку та майбутнє масштабування продукту. Розроблено структуру бази даних Cloud Firestore, оптимізовану для зберігання різнорідних тестових завдань.

5. UI/UX Дизайн: Розроблено концепцію користувацького інтерфейсу в стилі Material Design 3, яка враховує ергономіку мобільних пристроїв. Акцент зроблено на використанні темної теми та синтаксичного підсвічування коду, що створює комфортні умови для тривалого навчання.

Практична цінність роботи полягає у створенні готової специфікації та архітектурного проекту, які можуть бути використані для безпосередньої програмної реалізації додатку. Запропоноване рішення здатне підвищити ефективність самостійної підготовки Android-розробників.

Перспективи подальшого розвитку проекту вбачаються у впровадженні елементів штучного інтелекту для генерації унікальних тестових завдань, додаванні режиму змагань (PvP) між користувачами та інтеграції вбудованого компілятора коду для виконання практичних задач безпосередньо в додатку.

Апробація результатів роботи. Основні положення та результати дослідження пройшли апробацію на XLVIX Міжнародній науковій конференції студентів та аспірантів «Актуальні питання розвитку науки та забезпечення якості освіти у XXI столітті». За матеріалами роботи опубліковано тези доповіді у відповідному збірнику матеріалів.

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Firebase Documentation : офіційна документація платформи Firebase. URL: https://firebase.google.com/docs (дата звернення: 05.04.2026).
2. Kotlin Language Specification : технічна специфікація мови Kotlin. URL: https://kotlinlang.org/spec (дата звернення: 10.04.2026).
3. Material Design 3 Guidelines : офіційна документація Google Design. URL: https://m3.material.io (дата звернення: 14.04.2026).
4. Ольховська О. В., Черненко О. О. Методичні рекомендації до виконання кваліфікаційної роботи — Полтавський університет економіки і торгівлі, 2025. — 67 с.
5. Лавріщева К. М. Інженерія програмного забезпечення : підручник. Київ : Академперіодика, 2016. 304 с.
6. Жуковський С. С., Вакалюк Т. А. Об’єктно-орієнтоване програмування : навч.-метод. посіб. Житомир : Вид-во ЖДУ ім. І. Франка, 2019. 112 с.
7. Ковалюк Т. В. Основи програмування. Київ : Видавнича група BHV, 2005. 384 с.
8. Мартін Р. Чиста архітектура. Мистецтво розробки програмного забезпечення / пер. з англ. М. В. Сидоренка. Харків : Фабула, 2019. 336 с.
9. Морзе Н. В. Методика навчання інформатики : навч. посіб. Київ : Навчальна книга, 2004. 256 с.
10. Пасічник В. В., Резніченко В. А. Організація баз даних та знань. Київ : Видавнича група BHV, 2006. 384 с.
11. Anderson L. W., Krathwohl D. R. A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives. New York : Longman, 2001. 352 p.
12. Bailey A., Meiring D., Castro D. Android Development with Kotlin. Birmingham : Packt Publishing, 2023. 450 p.
13. Crispin L., Gregory J. Agile Testing: A Practical Guide for Testers and Agile Teams. Boston : Addison-Wesley, 2009. 576 p.

14. Dunn M. Robust Android Architecture with Kotlin. London : Packt Publishing, 2024. 312 p.
15. Gamma E., Helm R., Johnson R. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.
16. Garg V. Mastering Jetpack Compose. London : Packt Publishing, 2023. 344 p.
17. Griffiths D., Griffiths D. Head First Kotlin: A Brain-Friendly Guide. Sebastopol : O'Reilly Media, 2019. 480 p.
18. Freeman E., Robson E., Bates B. Head First Design Patterns. 2nd ed. Sebastopol : O'Reilly Media, 2020. 672 p.
19. Meier R., Donovan I. Professional Android. 4th ed. Indianapolis : Wrox, 2018. 1056 p.
20. Myers G. J., Sandler C. The Art of Software Testing. 3rd ed. Hoboken : John Wiley & Sons, 2011. 240 p.
21. Skeen J., Greenhalgh D. Kotlin Programming: The Big Nerd Ranch Guide. 2nd ed. Atlanta : Big Nerd Ranch Guides, 2021. 592 p.
22. Parsons D., Haden P. Parson's programming puzzles: a fun and effective learning tool for first programming courses. Proceedings of the 8th Australasian Conference on Computing Education. 2006. Vol. 52. P. 157–163.
23. Android Developers : офіційна документація Google для розробників. URL: https://developer.android.com (дата звернення: 16.04.2026).
24. Guide to App Architecture : офіційна документація Android Developers. URL: https://developer.android.com/topic/architecture (дата звернення: 24.04.2026).