ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка мобільного застосунку для вивчення програмування з елементами ігрофікації»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Владислав СПИРИДОНОВ
(підпис)

Виконав: Здобувач вищої освіти групи ПД-44

Владислав СПИРИДОНОВ

Керівник: Віктор ЛАВРЕНЕНКО
викладач кафедри ІПЗ

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

_____Спиридонову Владиславу Сергійовичу_____

1. Тема кваліфікаційної роботи: «Розробка мобільного застосунку для вивчення програмування з елементами ігрофікації»
керівник кваліфікаційної роботи викладач кафедри ІПЗ Віктор ЛАВРЕНЕНКО, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи розробки, методики розробки освітніх застосунків з елементами взаємодії користувача, документація технологій Jetpack Compose, Kotlin, Firebase та AI-моделей для автоматичного рев'ю коду та методи гейміфікації у навчанні.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд мобільного застосунку для вивчення програмування та аналіз існуючих аналогів, формування вимог до програмного забезпечення.
2. Дослідження існуючих аналогів для визначення їхніх переваг і недоліків та формування вимог до нашого застосунку.

3.Пректування архітектури, моделі даних і користувацького інтерфейсу мобільного застосунку.
4. Описання засобів реалізації застосунку та проведення тестування застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз існуючих рішень.
2. Вимоги.
3. Програмні засоби реалізації.
4. Опис програми для вивчення програмування.
5. Діаграма варіантів використання.
6. Структурна схема екранів мобільного застосунку.
7. Екранні форми.
8. Демонстрація роботи програми для вивчення програмування з елементами ігрофікації
9. Апробація результатів дослідження.

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд методів до створення мобільних застосунків для програмування з інтеграцією AI	14.03-20.03.2025	
4	Проектування додатку для вивчення програмування з квестами, AI-код-рев'ю та досягненнями	21.03-30.03.2025	
5	Програмна реалізація додатку	31.03-15.04.2025	
6	Тестування застосунку	16.04-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Владислав СПИРИДОНОВ
(підпис)

Керівник
кваліфікаційної роботи _____ Віктор ЛАВРЕНЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 66 стор., 4 табл., 27 рис., 20 джерел.

Мета роботи - Оптимізація процесу вивчення програмування за рахунок використання принципів гейміфікації та методів штучного інтелекту, розробка мобільного застосунку для покращення засвоєння матеріалу та підвищення мотивації користувачів.

Об'єкт дослідження - Процес навчання програмуванню.

Предмет дослідження - Програмне забезпечення для вивчення мов програмування.

Короткий зміст роботи: У роботі розглянуто ключові аспекти навчання програмуванню із застосуванням сучасних методів гейміфікації, зокрема мобільних освітніх застосунків, інтеграції чат-ботів та автоматизованого AI-код-рев'ю. Проаналізовано популярні технології для створення мобільних додатків, такі як: Ollama, Firebase, Jetpack Compose.

Розроблено архітектуру мобільного застосунку Kvantor, який забезпечує поетапне опанування мов програмування у форматі теоретичних матеріалів, тестових завдань, вправ із написання коду та взаємодії з AI у вигляді чат-бота. Програмно реалізовані ключові модулі: Реєстрація та авторизація, подання навчального матеріалу, збереження прогресу, інтегрування штучного інтелекту Ollama для AI-код-рев'ю та чат-бота, створеного для підтримки користувача в процесі вивчення програмування.

В роботі використано Jetpack Compose для побудови інтерфейсу користувача, Firebase Authentication і Firestore для взаємодії з базою даних, Штучний інтелект, що використовується для здійснення аналізу програмного коду та забезпечення персоналізованої взаємодії з користувачем через чат-бот. Сферою використання застосунку є підвищення мотивації студентів до вивчення мов програмування шляхом застосування гейміфікованого підходу та інтерактивних елементів з інтеграцією автоматизованого зворотного зв'язку та

персоналізованої взаємодії через чат-бот. Структура програмного рішення створена методом модулів, що надає можливість без внесення змін до основного коду, адаптувати контент під користувачів різного рівня підготовки та створювати нові курси.

З метою мінімізації затримок при взаємодії з AI-моделлю реалізовано механізм локального кешування відповідей у пам’яті пристрою, що забезпечує оперативний доступ до раніше згенерованих результатів. При відновленні мережевого з’єднання використовується алгоритм синхронізації з пріоритетною обробкою критично важливих даних, що гарантує актуальність та цілісність інформації. Передбачено базові засоби шифрування та модерації контенту, що узгоджується з чинними стандартами GDPR. Отримані результати відкривають перспективи подальших досліджень у сфері гейміфікованого навчання та розвитку Kvantor.

У майбутньому планується запуск iOS-версії застосунку на SwiftUI, що розширить аудиторію. Крім того, запровадження адаптивної аналітичної панелі, яка дасть змогу викладачам відстежувати прогрес студентів у реальному часі й коригувати послідовність засвоєння матеріалу на основі об’єктивних даних.

КЛЮЧОВІ СЛОВА: МОБІЛЬНИЙ ЗАСТОСУНОК, ПРОГРАМУВАННЯ, ГЕЙМІФІКАЦІЯ, ШТУЧНИЙ ІНТЕЛЕКТ, ЧАТ-БОТ, AI-КОД-РЕВ’Ю, JETPACK COMPOSE, FIREBASE, OLLAMA, ОСВІТА, KVANTOR.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ - Програмне забезпечення

AI - Artificial Intelligence (Штучний інтелект)

IDE - Integrated Development Environment (Інтегроване середовище розробки)

UI - User Interface (Інтерфейс користувача)

UX - User Experience (Користувацький досвід)

API - Application Programming Interface (Інтерфейс прикладного програмування)

JSON - JavaScript Object Notation (Формат обміну даними)

MVP - Minimum Viable Product (Мінімально життєздатний продукт)

AI-код-рев'ю - Автоматизована перевірка коду за допомогою AI

Firebase - Хмарна платформа Google для бекенд-рішення

UI/UX - Сфера дизайну інтерфейсів та взаємодії користувача

CRUD - Create, Read, Update, Delete (Базові операції з даними)

DB - Database - база даних

KVANTOR - авторський мобільний застосунок, створений у межах кваліфікаційної роботи, що забезпечує навчання програмуванню з використанням гейміфікованих механік, інтерактивних завдань та інструментів штучного інтелекту.

ВСТУП

Актуальність: Швидка цифровізація бізнес-процесів стимулює зростаючий попит на ІТ-фахівців, тоді як класичні освітні програми часто не встигають за темпами змін вимог ринку: вони потребують значних часових і ресурсних затрат, обмежують гнучкість навчального графіка та недостатньо орієнтовані на практичне застосування знань. Сучасні користувачі прагнуть інтегрувати навчання в свій насичений розклад - під час поїздок, між зустрічами або перервами на роботі - тому мобільні платформи з короткими адаптивними уроками дедалі популярніші. Формат microlearning дозволяє виконувати невеликі навчальні сесії тривалістю 5-7 хвилин, забезпечуючи неперервний розвиток навичок без необхідності виділяти великі проміжки часу.

Поряд із зручністю «навчися де завгодно» важливою стає мотивація учнів: гейміфікація - система рівнів, бейджів та рейтингових таблиць - перетворює навчальний процес на захопливу гру з миттєвим відгуком, підтримує конкурентний дух і сприяє глибшому засвоєнню матеріалу. Дослідження показують, що інтеграція ігрових механізмів підвищує ефективність навчання та мотивацію, адже користувачі бачать власний прогрес і отримують постійне позитивне підкріплення.

У світлі цих тенденцій розробка мобільного додатка для вивчення програмування стає вкрай своєчасною відповіддю на реальні потреби сучасного суспільства. Поєднуючи переваги інтерактивного формату, можливості live-coding із автоматичними тестами та AI-рев'ю, а також елементи гейміфікації, платформа забезпечить доступне, гнучке й водночас глибоке опанування навичок програмування для широкого кола користувачів.

Об'єкт дослідження: Процес навчання програмуванню.

Предметом дослідження є: Програмне забезпечення для вивчення мов програмування.

Метою роботи є оптимізація процесу вивчення програмування за рахунок використання принципів гейміфікації та методів штучного інтелекту, розробка

мобільного застосунку для покращення засвоєння матеріалу та підвищення мотивації користувачів.

Методи дослідження: У ході розробки мобільного додатка було використано поєднання теоретичних та практичних підходів. Спершу був проведен аналіз наукових джерел й нормативних документів у сфері ІТ-освіти щодо гейміфікації, освіти у мобільному форматі, та інтеграції мовних моделей. Це дало змогу сформувати функціональні та нефункціональні вимоги до системи. Далі було проаналізовано популярні освітні додатки з метою виявити сильні сторони та недоліки конкурентів. На основі отриманих висновків створено робочий прототип у середовищі Android Studio з використанням Jetpack Compose та підключенням моделі ШІ Ollama для code-review та чат боту. Прототип проходив внутрішнє функціональне тестування безпосередньо розробником: Було перевірено валідність авторизації через Firebase, злагодженість взаємодії компонентів бази даних та робота AI-компонентаТаким чином, застосовані методи дозволили здійснити попередню оцінку життєздатності розробленої концепції та водночас забезпечили комплексне обґрунтування архітектурних рішень на етапі до залучення кінцевих користувачів.

Практична значущість результатів полягає в можливості використовувати застосунок для вивчення мов програмування на своїх телефонах.

1 АНАЛІЗ СУЧАСНИХ ЗАСОБІВ ДЛЯ ВИВЧЕННЯ МОВ ПРОГРАМУВАНЯ

1.1 Мобільний додаток KVANTOR для вивчення мов програмування

KVANTOR - це мобільна платформа, що дозволяє початківцям крок за кроком опановувати базові концепції програмування. Застосунок орієнтований на початківців і надає можливість опанування теоретичного та практичного матеріалу та у зручному мобільному форматі. Концепція застосунку передбачає поетапне подання навчального матеріалу, що логічно переходить від ознайомлення з теоретичними основами до виконання тестових завдань і самостійного написання коду.

1.1.1 Концепція та цільова аудиторія

KVANTOR побудовано навколо ідеї об’єднати лаконічні теоретичні пояснення, інтерактивні вправи та індивідуальний AI-фідбек у єдиний освітній потік. В таблиці 1.1 наведено основні компоненти та функціональні блоки платформи, які забезпечують цю єдність. Ми застосовуємо так званий мікроформат: кожна навчальна сесія триває 5-7 хвилин, що дозволяє вставити заняття в будь-який вільний момент - наприклад, під час поїздок чи перерв між справами. Теорія викладається невеликими блоками, одразу після яких користувач переходить до практичних завдань: тестів і написання малого фрагмента коду. Такий підхід спирається на дані досліджень, які доводять, що матеріал краще засвоюється, якщо його одразу застосовувати на практиці. Для персоналізації досвіду ми інтегрували локальну LLM-модель Phi-3, яка імітує код-рев’ю: користувачі отримують детальний діалоговий зворотний зв’язок, ніби їхній код переглядає реальний ментор. Завдяки цьому KVANTOR перетворюється з простого додатка на адаптивне середовище, що підлаштовується під індивідуальний темп і стиль навчання кожного.

Таблиця 1.1

Орієнтація KVANTOR на різні цільові групи користувачів

Цільова група	**Її потреби**	**Як KVANTOR це задовольняє**
Студенти IT-фаху	Швидко повторити матеріал, багато практики	Мікроуроки + живий редактор коду
Студенти не-IT	Низький “поріг входу”, пояснення без жаргону	Побутові приклади, візуальні пазли, поступовий перехід до чистого коду
Самоучки-дорослі	Поєднати навчання з роботою, бачити чіткий прогрес	Гейміфікація (XP, досягнення, streak), push-нагадування про Daily Goal

1.2 Аналіз існуючих освітніх рішень

У сучасному освітньому середовищі для опанування програмування виділяють кілька основних підходів, які відрізняються за форматом, інструментарієм та глибиною подачі матеріалу. Перший із них - традиційні академічні курси, що пропонуються в університетах і спеціалізованих IT-центрах. Вони базуються на класичній лекційно-практичній моделі: студенти відвідують заняття, виконують лабораторні роботи в середовищі IDE на комп’ютері та проходять контрольні тестування. Такий формат забезпечує глибоке занурення в фундаментальні концепції, однак вимагає витрат значного часу на відвідування аудиторій і не завжди відповідає динаміці сучасного ринку праці. [1]

Другим напрямом є великі MOOC-платформи (Coursera, edX, Udemy та ін.), які надають структуровані курси з відеолекціями, інтерактивними вправами й автоматизованою перевіркою домашніх завдань. Студенти можуть навчатися у власному темпі, обирати зручний графік і отримувати сертифікати після успішного завершення модулів. Такий формат поєднує академічну строгость із гнучкістю доступу, але залежить від якості перекладу та локалізації матеріалів, а також потребує високої самоорганізації слухачів.

Третій підхід охоплює інтерактивні веб-сервіси та онлайн IDE (Codecademy, freeCodeCamp, Code.org тощо), які дозволяють безпосередньо в браузері писати й виконувати код. Навчальні вправи поділено на короткі кроки з миттєвою зворотною реакцією системи та підказками, що сприяє швидкому закріпленню синтаксису і базових алгоритмів. Ця модель особливо ефективна для новачків, оскільки знижує бар'єр входу та дає відчуття «гри» під час навчання. Водночас обсяг теоретичного матеріалу може бути обмеженим, і платформи часто орієнтовані на одну-дві мови програмування.

Четвертий напрям - мобільні та змагальні рішення, що поєднують елементи мікронавчання й гейміфікації. Сервіси такого типу пропонують короткі уроки (5-10 хвилин), щоденні челенджі, внутрішню валюту й таблиці лідерів, стимулюючи регулярну практику. Вони ідеально вписуються в щільний графік сучасних користувачів і мотивують через рівні, бейджі та соціальні змагання. Хоча мобільні формати не замінюють глибокі курси, вони є чудовим доповненням до інших освітніх рішень, дозволяючи підтримувати навичку щоденними вправами.

У статті Patience Davies "Just where did 'claim stories' (22 times more memorable) come from?" на платформі LinkedIn детально розбирається походження широко поширеного твердження про те, що історії запам'ятовуються у 22 рази краще за факти. Авторка прослідковує маршрут цієї цитати через численні публікації та маркетингові матеріали й доходить висновку, що немає жодних наукових доказів або первинних досліджень, які б підтверджували це число. Davies показує, як «лабораторний міф» перетворюється на факт завдяки повторюваному ресерчу й цитуванню без перевірки джерел, закликаючи освітян і маркетологів бути обережними з подібними гучними заявами та шукати першоджерела перед застосуванням у власних матеріалах. [5]

1.3 Особливості ігрофікації в освітніх застосунках

У сучасних навчальних додатках ігрові елементи все частіше виступають основним інструментом для залучення користувачів і підтримки їхньої мотивації. Замість звичайної послідовності теорії та вправ платформи додають системи очок

і рівнів, які відображають прогрес у реальному часі та дають відчуття досягнення після кожної виконаної вправи. Щоб відзначити ключові етапи навчання - завершення курсу або підтримку регулярності занять - використовуються візуальні бейджі й нагороди, які «закріплюють» успіхи в пам’яті користувача.

Додатковий шар змагання вводять таблиці лідерів, де учасники можуть порівняти свої показники з результатами інших та отримати додатковий стимул підвищувати власні навички. Віртуальна валюта, яку «заробляють» за виконані завдання, дозволяє обмінювати здобуті бали на підказки або додаткові спроби, що додає елемент стратегії у процес навчання. Так само прогрес-бари чітко демонструють, скільки ще залишилося до наступного рівня, і підтримують відчуття поступової динаміки.

Окрему увагу сучасні платформи приділяють сюжетним лініям: навчальні курси побудовані як пригоди, де користувач виконує місії, проходить «рівні» та взаємодіє з персонажами. Це особливо привабливо для молодших студентів, адже оволодіння новими концепціями перетворюється на захопливу подорож. Нарешті, персоналізація профілю - можливість вибрати аватар, задати псевдонім або оформити власне «живе» відображення прогресу - робить навчання більш особистим і сприяє емоційному залученню. Таким чином, гейміфікація охоплює все від системи балів і лідербордів до сюжетної подачі й індивідуальних налаштувань, забезпечуючи стійку мотивацію учнів і високу ефективність у засвоєнні матеріалу.

Багато освітніх платформ доповнюють базові вправи внутрішньою валютою або ресурсами, які користувач може «витратити» на підказки чи додаткові спроби при виконанні завдань. Така система заохочення створює відчуття реальної винагороди за кожен крок уперед і стимулює продовжувати навчання.

Крім того, персоналізація інтерфейсу - вибір аватару, налаштування профілю та отримання індивідуальних рекомендацій - надає відчуття власної участі в процесі, немов гравець «прокачує» свого персонажа, проходячи освітні етапи. Відстеження успіхів у вигляді індикаторів прогресу та графічної інфографіки,

наприклад, таймера послідовних днів занять (streak), відсотка проходження курсу або діаграм із досягненнями, допомагає користувачеві візуалізувати власний розвиток і підсилює мотивацію до регулярної практики.

Деякі платформи переносять навчання в формат інтерактивної історії: користувач рухається по «рівнях» та виконує завдання як персонаж гри, що допомагає краще запам’ятовувати матеріал - дослідження свідчать про збільшення ефективності навчання до 63 % порівняно з традиційним викладом (лише 5 %). Загалом гейміфікація довела свою дієвість у мета-аналізі 41 дослідження з понад 5000 учасників: впровадження ігрових механік суттєво підвищує успішність та мотивацію (ефект $g \approx 0.82$) [4] .

Одним із найяскравіших прикладів є Duolingo, яке перетворило нудне заучування на захопливу гру, і зараз цей підхід активно адаптують для програмування. Зокрема, Mimo виводить уроки з Python, JavaScript, HTML та інших мов у форматі коротких ігрових завдань - за що його часто називають «Duolingo для кодування». У наступних розділах ми докладніше розглянемо, як провідні мобільні застосунки реалізують ці та інші гейміфіковані елементи.

2 ДОСЛІДЖЕННЯ ІСНУЮЧИХ АНАЛОГІВ

Був проведен огляд трьох провідних мобільних застосунків для вивчення програмування - Grasshopper, Mimo і SoloLearn - щоб сформувати уявлення про їхні можливості й вимоги, які слід реалізувати в KVANTOR. Зокрема, Mimo повідомляє про базу понад 25 млн користувачів, а в SoloLearn зареєстровано понад 40 млн учнів; Grasshopper, хоч і закрив оновлення у 2023 році, усе ще слугує зразком простоти інтерфейсу для початківців. Кожен із цих продуктів демонструє свій підхід до поєднання теорії, практики та ігрових елементів, і аналіз їхнього функціоналу допоможе нам виокремити найважливіші характеристики для майбутнього розвитку KVANTOR.

2.1 Mimo

Mimo [2] входить до числа найпопулярніших мобільних додатків для вивчення кодування, пропонуючи понад десять навчальних треків - від Python і JavaScript до Swift, C# та SQL. Платформа дозволяє обрати спеціалізацію (наприклад, Front-end, Full-stack або Python) і проходити уроки у форматі коротких «порцій» теорії зразу ж після яких користувач вирішує практичні вправи: доповнює фрагменти коду, виправляє готові блоки або створює прості проєкти. Увесь процес відбувається без виходу із застосунку - у вбудованому редакторі з середовищем виконання - що робить навчання доступним у будь-якому місці та в будь-який час. Крім базових завдань, Mimo пропонує проєкти підвищеної складності, наприклад розробку простого веб-сайту або скрипта, що дає можливість одразу поповнювати власне портфоліо. Заохочуючи регулярність занять, платформа інтегрує елементи гейміфікації: за виконані вправи нараховуються бали досвіду, відкриваються досягнення та збільшується рівень прогресу, а короткі уроки (bite-sized lessons) у середньому займають не більше 5-7 хвилин, що ідеально вписується у щоденний графік користувача..

У Mimo за кожне виконане завдання та завершений проєкт нараховуються очки досвіду (XP), що відображають прогрес користувача. Щоб підтримувати інтерес, платформа регулярно проводить тематичні челенджі й вікторини з бонусними балами, а також формує таблицю лідерів, до якої потрапляють найактивніші учасники. Особливо мотивує функція «щоденного стрику» (daily streak): лічильник фіксує кількість днів, протягом яких користувач хоча б раз заходив у застосунок і проходив урок, що допомагає сформувати звичку до регулярного навчання.

На самому початку роботи Mimo пропонує скоротений тест для оцінки початкового рівня знань, аби адаптувати подальші уроки під індивідуальні потреби. Якщо під час проходження курсу виявляються складнощі, система автоматично надає додаткові теоретичні матеріали й дає можливість повторити теми через нові запитання. Такий підхід забезпечує поступове закріплення знань і знижує ризик фрустрації при вивченні коду.

Mimo є кросплатформовим рішенням - застосунок працює на Android, iOS та в браузері, що дозволяє переходити між пристроями без перерви в навчанні. Базовий контент доступний безкоштовно, а підписка Mimo Pro відкриває весь перелік курсів, розширений глосарій і необмежену практику, реалізуючи популярну freemium-модель. Окрім цього, у додаток інтегровано AI-помічника: якщо користувач застряг на якомусь рівні, він може звернутися до вбудованого чат-бота, який пояснить помилку або навіть запропонує фрагмент коду для вирішення завдання. Весь інтерфейс зі середовищем редагування коду та вікном AI Assistant наведено на рисунку 1.1, і він демонструє, як паралельно відображаються текст умови завдання, чат-помічник і результат виконання коду. Такий набір функцій робить Mimo гнучким інструментом як для ознайомлення з основами, так і для поглибленої практики.

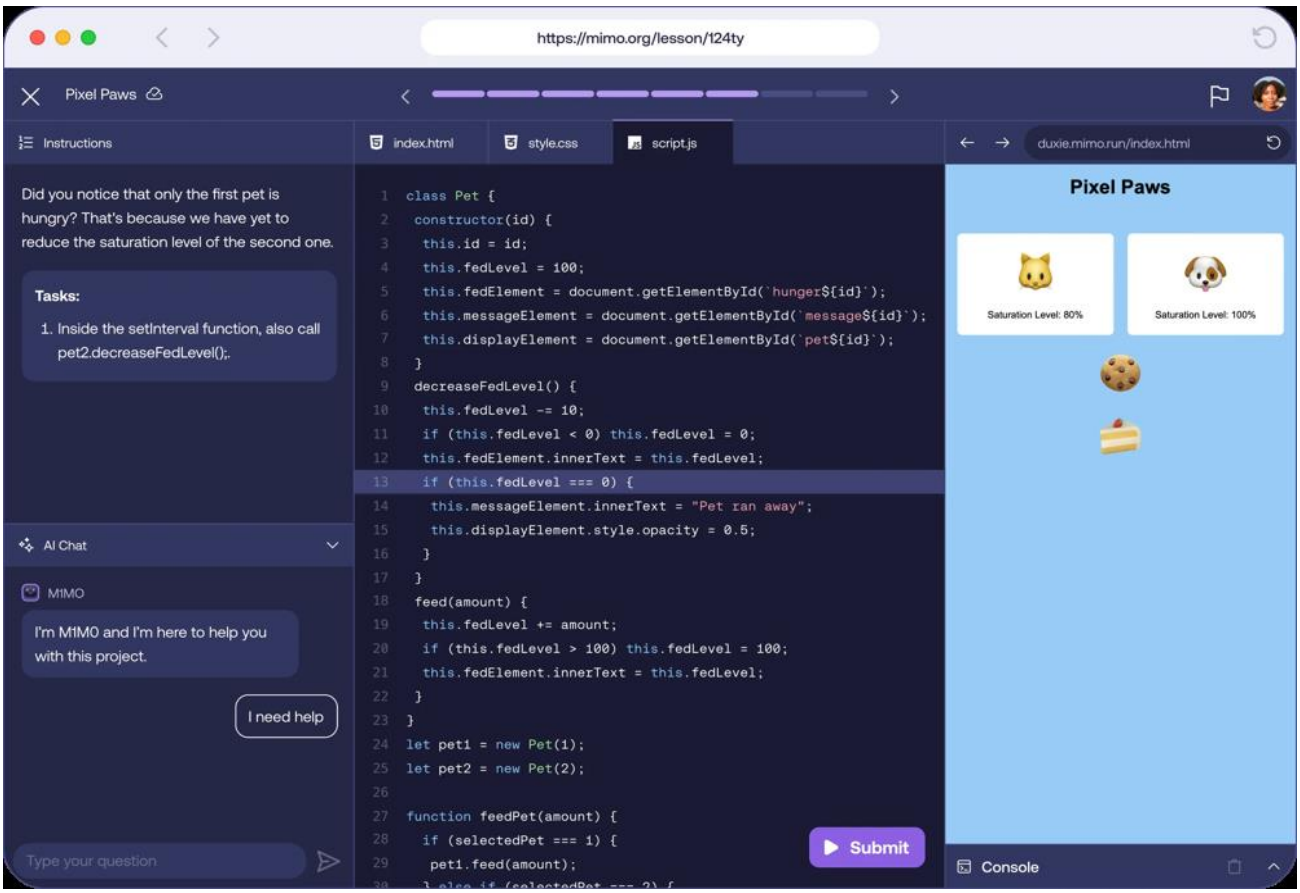


Рис. 1.1 Інтерфейс застосунку Mimo.

Mimo демонструє, як поєднання ігрових механік та широкого освітнього контенту може зробити вивчення коду захопливим і результативним. Платформа охоплює численні мови та технології, пропонуючи короткі практичні уроки з інтерактивними вправами, систему досягнень і візуальні індикатори прогресу, що підтримують постійну мотивацію. Завдяки вбудованому AI-асистентові користувачі можуть миттєво отримувати пояснення та підказки прямо під час роботи з кодом. До переваг Mimo належать лаконічний формат bite-sized lessons, сервіси для вибору навчального треку та гнучка freemium-модель із можливістю розблокувати повний набір курсів за підпискою. Серед недоліків - обмежений доступ до частини контенту без оплати та доволі прості вправи, які можуть не задовольнити тих, хто шукає глибшого занурення в складніші теми. Незважаючи на це, для новачків і користувачів середнього рівня Mimo залишається одним із найзручніших інструментів на ринку.

2.2 Grasshopper

З розвитком курсу завдання ускладнюються, а користувачу відкриваються нові концепції. Щоб визначити рівень підготовки, користувач починає з базового курсу “The Fundamentals”, у якому знайомиться з такими основами програмування, як змінні, функції, масиви, цикли тощо. Кожне нове поняття супроводжується практичними вправами. Наприклад, на початковому етапі застосунок пропонує зібрати прапор Франції, розміщуючи кольорові прямокутники в правильному порядку. Для цього необхідно викликати відповідні функції малювання фігур у заданій послідовності. Застосунок Grasshopper активно застосовує гейміфікацію в дещо простішій формі. В застосунку є досягнення в яких проказується, скільки нових концепцій опановано, скільки ключових слів JavaScript використано, скільки днів триває поточна серія занять без перерви та інша цінна для корситувача інформація Grasshopper від Google створено з метою познайомити новачків із кодуванням через серію інтерактивних вправ у вигляді міні-ігор. У центрі уваги - вивчення JavaScript: замість написання всього коду вручну користувач отримує готові блоки, які потрібно правильно впорядкувати, щоб вирішити завдання. Перший модуль «The Fundamentals» знайомить зі змінними, функціями, масивами та циклами у формі пазлів - наприклад, на початковому рівні потрібно розмістити кольорові фрагменти, щоб «намалювати» прапор Франції, викликаючи відповідні функції малювання. З кожним новим завданням головоломки стають складнішими, пропонуючи поступове занурення в більш просунуті концепції.

Щоб підтримувати інтерес, Grasshopper використовує просту систему досягнень: додаток відстежує, скільки днів поспіль користувач виконує хоча б одну вправу (streak), скільки нових ключових слів освоєно і які головоломки пройдено. Змагання у табличці лідерів та нагороди за серії днів без перерви стимулюють повертатися в додаток щодня. Таким чином, через «coding puzzles» і базову гейміфікацію Grasshopper дарує користувачам перший позитивний досвід роботи з кодом без зайвої складності синтаксису.

.

Рис. 1.2 Головний екран застосунку Grasshopper.

Grasshopper підтримує мотивацію користувача за рахунок лічильника безперервних днів занять (streak) і накопичення балів, що відображаються як значок із трофеєм та числовий індикатор у профілі на рисунку 1.2. Якщо серія переривається, застосунок надсилає нагадування, щоб повернути учня до навчання, а після завершення чергового розділу або отримання нової відзнаки користувач бачить вітальне повідомлення, яке посилює відчуття досягнення. Хоча оновлення платформи припинилися у 2023 році, вона демонструє ефективне поєднання мінімалістичного інтерфейсу й базових ігрових механік для формування першого досвіду роботи з JavaScript.

При цьому обмеженість підтримки лише однієї мови та відсутність просунутих рівнів складності можуть стати вузьким місцем для довготривалого залучення. Ураховуючи це, в KVANTOR ми прагнемо зберегти інтуїтивність початкового етапу, але відразу передбачити можливість масштабування курсу та підтримку кількох мов і рівнів майстерності.

2.3 SoloLearn

SoloLearn - це одна з найпопулярніших мобільних платформ для вивчення програмування, що пропонує курси з Python, Java, C++, JavaScript та низки інших мов. Завдяки ігровим елементам, таким як система балів, досягнень і лідербордів, додаток підтримує високий рівень залученості, а можливість створювати власні проєкти та ділитися ними з спільнотою допомагає вдосконалювати практичні навички. [3]

У наукових дослідженнях SoloLearn визнають ефективним інструментом дистанційного мобільного навчання. Зокрема, у публікації «Європейські орієнтири розвитку України» відзначається його популярність під час пандемії COVID-19 завдяки адаптивним урокам, інтерактивності та доступності для широкої аудиторії.

На відміну від Mimo, де завдання складаються здебільшого з тестів і коротких практичних вправ, основний акцент SoloLearn зроблено на повноцінному кодуванні в інтегрованому редакторі - користувачі можуть писати, запускати і відлагоджувати власні програми безпосередньо в додатку. Такий підхід сприяє глибшому зануренню в процес розробки та розвитку аналітичного мислення.

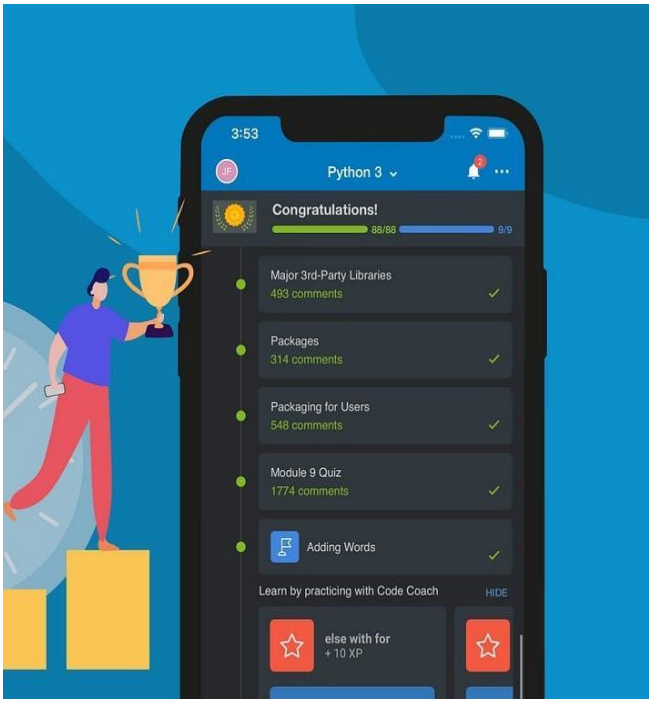


Рис. 1.3 Елементи гейміфікації в інтерфейсі застосунку SoloLearn

Ще одним важливим елементом SoloLearn є розвинена система ігрових відзнак і відображення прогресу: інтерфейс містить шкалу виконання курсу, різноманітні бейджі (наприклад, трофеї за завершені модулі або медалі за досягнення щоденного стрику) та бонусні завдання з додатковим досвідом (XP), як показано на рисунку 1.3 Завдяки цим візуальним підкріпленням створюється ігрова атмосфера, яка мотивує користувачів повертатися до уроків і регулярно підвищувати свій рівень знань, забезпечуючи постійне відчуття прогресу й позитивного підкріплення.

У 2023 році в SoloLearn з’явився Kodie - інтерактивний AI-асистент на базі OpenAI, який виступає в ролі персонального ментора для коду. Тепер, коли користувач не може розібратися з помилкою, достатньо звернутися до Kodie з питанням на кшталт «Чому не працює мій цикл?», і асистент зрозуміло пояснить причину збою в коді. Окрім того, штучний інтелект аналізує результати виконаних вправ і формує адаптивні завдання та тести відповідно до сильних і слабких сторін учня. Таким чином, кожен отримує індивідуально підібраний набір практичних вправ, що дозволяє більш ефективно закріпити матеріал і прискорити прогрес у навчанні.

2.4 Позиціювання KVANTOR серед аналізованих аналогів

Порівняльний аналіз трьох платформ - Grasshopper, Mimo та SoloLearn - показав, що кожен додаток займає свою нішу: Grasshopper пропонує максимально простий вступ до JavaScript через головоломки, Mimo поєднує короткі теоретичні блоки з bite-sized вправами та AI-асистентом, а SoloLearn робить ставку на повноцінне кодування в браузері й інтегрованого редактора. Вивчивши їхні особливості, ми визначили низку ключових факторів успіху: продумана студентська траєкторія (від бази до проєктів), гейміфікація для підтримки мотивації, адаптивність завдань під рівень учня та наявність миттєвого зворотного зв’язку. Ці висновки лягли в основу створення KVANTOR - ми спроєктували платформу таким чином, щоб поєднати інтуїтивний стартовий досвід Grasshopper,

гнучкість і різноманітність курсів Mimo та глибину кодування SoloLearn, додавши власні механіки адаптивного AI-рев'ю та масштабовану архітектуру для подальшого розвитку.

Таблиця 1.2

Порівняння ключових параметрів та позиціювання KVANTOR

Ключовий критерій	**Сильна риса аналога**	**Виявлені обмеження**	**Як це враховано у KVANTOR**
Підтримка мов	SoloLearn - 20 мов (широка лінійка)	Grasshopper - тільки JS; Mimo/Grasshopper не дають гнучкого додавання нових курсів	Стартуємо з Python і JavaScript, але модульна архітектура дозволяє додати будь-яку мову без переробки ядра
Формат уроків	Mimo - короткі bite-sized уроки	SoloLearn має плутану навігацію, частина уроків занадто довга	Чіткий мікроформат (5-7 хвилин), одноманітну теорію одразу підкріплює код-вправа
AI-підтримка	SoloLearn (PRO) - Kodie	Mimo - чат-підказка тільки в проєктах; Grasshopper - відсутня	Локальна LLM Phi-3 (через Ollama) дає code-review кожному відкритому питанню
Гейміфікація	Mimo / SoloLearn - XP, бейджі, streak	Grasshopper - мінімальна, SoloLearn - рейтинги «з'їдають» новачків	Досягнення (ачівки); інші елементи гейміфікації - у розробці (наприклад, XP, щоденні цілі, сезонні рейтинги)
Модель вартості	Повністю безкоштовний	Mimo/SoloLearn – freemium з рекламою	Повністю безкоштовний
Офлайн-режим	-	У всіх трьох – лише часткове кешування	Повноцінний офлайн: уроки й автотести працюють без мережі, синхронізація при першому підключенні

Огляд трьох платформ виявив у кожної свої сильні сторони та водночас обмеження. Grasshopper ідеально підходить для початкового знайомства з JavaScript, але через вузьку спеціалізацію й дефіцит контенту його ефективність

рано чи пізно вичерпується. Mimo та SoloLearn охоплюють ширший спектр мов і форматів завдань, що підкреслює важливість гнучкої, масштабованої архітектури.

KVANTOR поєднує ці підходи: стартові курси з Python і JavaScript доповнені можливістю просто й швидко додавати нові мови без зміни ядра платформи. Наше дослідження підтвердило, що для формування реальних навичок недостатньо лише теоретичних питань чи простих тестів - необхідний повноцінний live-coding з автоматизованими перевірками та глибинним code-review. Тому вже з перших уроків KVANTOR пропонує режим написання коду в реальному часі з автотестами та детальним аналізом, який забезпечує локально розгорнута LLM-модель Phi-3. На відміну від обмеженого чат-помічника в Mimo чи платного Kodie у SoloLearn, наш AI не потребує зовнішніх квот і гарантує миттєвий персональний фідбек.

Таблиця 1.2 порівнює ключові параметри платформи KVANTOR з дослідженими аналогами і демонструє, як наше рішення поєднує інтуїтивність, масштабованість та інноваційний AI-підхід.

2.5 Формування вимог до програмного забезпечення

Наразі в Kvantor доступні курси з Python та JavaScript, розроблені для абсолютних новачків. Архітектура застосунку побудована таким чином, щоб у майбутньому забезпечити просте додавання нових курсів без необхідності кардинальних змін у системі На підставі аналізу функціональності Grasshopper, Mimo та SoloLearn сформульовано базові вимоги до мобільного додатку Kvantor. По-перше, він має створювати інтерактивне середовище, де кожен урок поєднує короткі теоретичні блоки з практичними завданнями та живим кодуванням із автоматичними перевірками. По-друге, важливо впровадити елементи гейміфікації - бали досвіду, досягнення, щоденні серії занять і рейтингові таблиці - щоб заохочувати регулярну активність і підтримувати високий рівень мотивації. По-третє, AI-компонент повинен працювати локально, надаючи користувачам миттєвий персоналізований фідбек і адаптуючи завдання під їхній рівень знань без зовнішніх обмежень.

З технічної точки зору, клієнтська частина на Android базується на поєднанні MVVM та Clean Architecture, а сервіс штучного інтелекту винесено в окремий Flask-модуль. Така модульність забезпечує незалежність розвитку й оновлення обох компонентів. Наразі платформа охоплює два стартові курси - Python та JavaScript для абсолютних новачків, - а її архітектура дозволяє легко додати нові мови та напрямки без зміни «серця» системи. Таким чином, Kvantor поєднує гнучкість у розширенні зі злагодженим робочим процесом, що сприяє ефективному засвоєнню матеріалу.

Кожен курс у KVANTOR розділений на тематичні модулі, кожен із яких містить три типи контенту: блок теорії з ілюстраціями коду, інтерактивні вправи (вибір правильної відповіді чи вставка фрагмента коду) та практичні завдання, де користувачу пропонується написати власний код. Така структура дає змогу безболісно додавати нові модулі в майбутньому, зберігаючи єдину логіку навігації.

Для завдань із відкритим кодом реалізовано автоматичну валідацію: рішення користувача запускається на прихованих тестах, що дозволяє миттєво побачити, чи правильний результат. Крім того, інтегрований AI-помічник аналізує надісланий код, виявляє помилки та пропонує пояснення і поради щодо покращення реалізації. Такий підхід поєднує швидкий фідбек із глибоким аналізом, сприяючи більш ефективному навчанню.

У KVANTOR передбачено низку гейміфікаційних механік, які заохочують користувачів повертатися до занять і відчувати прогрес. Зокрема, система досягнень автоматично видає «ачівки» за ключові етапи: перші виконані код-завдання, завершення модулів або серії днів активного навчання. У профілі ці відзнаки відображаються як візуальні позначки, що демонструють індивідуальний розвиток та мотивують рухатися далі. Якщо під час тестів виникають складнощі, можна скористатися підказками, які не лише допомагають обрати правильну відповідь, а й пояснюють сутність теми, що сприяє глибшому розумінню матеріалу.

Додатково вбудована внутрішня валюта - монети, які користувачі накопичують за успішно виконані вправи й завдання. Ці монети можна витратити на додаткові спроби чи нові підказки, що додає елемент стратегії у навчальний

процес і заохочує до регулярної активності. У майбутньому планується розширення гейміфікації: запровадження щоденних цілей, системи досвіду (XP), сезонних рейтингів і соціальних викликів між учасниками, аби зробити процес навчання максимально динамічним та інтерактивним.

У магазині KVANTOR накопичені монети використовуються як внутрішня валюта: їх можна обміняти на додаткові «життя» для повторного проходження завдання після помилки або на підказки під час тестів, що додає стратегічний елемент у процес навчання та дозволяє користувачеві контролювати свій прогрес. У перспективі асортимент магазину можна буде розширити косметичними предметами для персоналізації інтерфейсу або спеціальними бонусами за активність, щоб зробити систему ще більш привабливою та різноманітною.

Профіль користувача у KVANTOR передбачає налаштування нікнейму та вибір аватара з готової галереї, а на головній сторінці кабінету відображаються вже отримані та майбутні досягнення. Завдяки цьому кожен учень бачить свій індивідуальний шлях розвитку й може підкреслити власну унікальність у межах освітнього середовища.

Для надійного зберігання всіх даних - від прогресу й профілів до результатів автотестів та медіаконтенту - ми обрали екосистему Firebase [10] від Google. Аутентифікацію користувачів забезпечує Firebase Authentication (email, Google Account тощо), а базою даних слугує Cloud Firestore, що підтримує офлайн-кешування та дозволяє оновлювати уроки без випуску нової версії додатка. Для запуску автотестів і перевірки коду використовуються Cloud Functions, які виконуються в ізольованому середовищі, а великий обсяг файлів (наприклад, аватарки) зберігається у Firebase Storage. Завдяки такому хмарному бекенду дані синхронізуються між пристроями, а платформа легко масштабується під навантаження великої кількості користувачів.

Інтерфейс та технології розробки. Клієнтська частина Kvantor розробляється для платформи Android із використанням фреймворку Jetpack Compose. [11] Це сучасний інструмент, який дозволяє створювати гнучкий, візуально привабливий і зручний інтерфейс із декларативною розміткою.

3 ПРОЄКТУВАННЯ

3.1 Вибір та обґрунтування архітектурного рішення

У KVANTOR застосовано розділення клієнтської й серверної частин із чітким виділенням відповідальностей: користувацький інтерфейс на Android базується на MVVM у межах Clean Architecture, а всі AI-функції винесено в окремий Flask-мікросервіс. Така двокомпонентна модель дозволяє паралельно розвивати мобільний додаток і сервіс обробки коду без взаємного ризику порушити роботу одного з них.

Шар даних відповідає за завантаження та збереження інформації (Firestore, локальне кешування).

Бізнес-логіка винесена у Use Cases і ViewModel, що гарантує незалежність від Android API та спрощує написання юніт-тестів.

UI-рівень зосереджено на декларативних Composable, що взаємодіють із ViewModel через зрозумілі потоки даних (Kotlin Flow).

Таке розмежування робить систему стійкою до змін: оновлення будь-якого шару чи сервісу (наприклад, заміна Flask-мікросервісу на інший механізм AI) не потребує переробки решти коду.

MVC (Model-View-Controller): проста реалізація, але в Android призводить до надмірного зв’язку між Activity та View, що ускладнює модульне тестування та підтримку.

MVP (Model-View-Presenter): кращий розподіл логіки, але за рахунок «важких» Presenter-класів і значної кількості шаблонного коду.

Обидва підходи виявилися недостатньо гнучкими для нашого сценарію, тому було прийнято рішення поєднати переваги MVVM і Clean Architecture.

Модульність - кожен шар ізольований і легко тестується.

Масштабованість - нові функції або курси додаються без зміни базового коду.

Швидкість розробки - автономна робота над AI-сервісом дозволяє паралельно впроваджувати й удосконалювати його алгоритми.

Стабільність - оновлення одного з компонентів не впливають на інші, що гарантує безперебійну роботу платформи.

Таким чином, обрана архітектура оптимально поєднує необхідну гнучкість, можливості для автоматизованого тестування та простоту майбутнього розвитку як мобільного інтерфейсу, так і AI-модуля.

Під час аналізу також розглядали патерн MVP (Model-View-Presenter), який розділяє представлення та бізнес-логіку через окремий Presenter і тим самим полегшує тестування. Проте на практиці такий підхід породжує надлишок шаблонного коду для зв’язку View із Presenter і може призвести до появи «fat presenters» зі зростанням складності проєкту. [6]

Отже, поєднання Clean Architecture із MVVM забезпечує оптимальний баланс: кожен компонент ізольований і легко тестується, а нові сервіси додаються без змін у клієнтському коді. Це архітектурне рішення відповідає всім ключовим вимогам KVANTOR - модульності, масштабованості, чіткому розподілу обов’язків і простоті інтеграції штучного інтелекту при збереженні можливості подальшого розширення системи. Основними критеріями вибору стали тестованість, гнучкість і легкість підтримки. Детальне наведено в таблиці 3.1.

Таблиця 3.1

Порівняння архітектурних підходів за ключовими критеріями

Критерій	**Clean Architecture + MVVM**	**MVC**	**MVP**
Тестованість	Висока (ізольовані тести)	Низька (залежність від Android)	Середня (легше тестувати, ніж MVC)
Модульність	Висока (чітке розділення шарів)	Низька (тісний зв'язок компонентів)	Середня (краще, ніж MVC)

Продовження таблиці 3.1

Порівняння архітектурних підходів за ключовими критеріями

Критерій	**Clean Architecture + MVVM**	**MVC**	**MVP**
Масштабованість	Висока (легке оновлення компонентів)	Низька (складність розширення)	Середня (обмежена масштабованість)
Складність реалізації	Середня (потрібна додаткова організація)	Низька (проста структура)	Висока (багато шаблонного коду)
Підтримка	Висока (легкість внесення змін)	Низька (складність модифікації)	Середня (обмежена гнучкість)

Обираючи поєднання Clean Architecture і MVVM, ми досягли необхідного балансу між модульністю, тестованістю та масштабованістю. Така структура дозволяє незалежно вдосконалювати мобільний клієнт та AI-сервіс: логіку інтерфейсу і бізнес-вимоги у ViewModel відділено від Android-специфіки, а алгоритми аналізу коду й генерації фідбеку винесені у Flask-мікросервіс. Це не лише підвищує швидкодію застосунку, позбавляє його від зайвих залежностей і полегшує підтримку, але й дає змогу оперативно оновлювати й оптимізувати нейронні моделі без ризику порушити стабільність клієнта. У порівнянні з класичними MVC та MVP, наша архітектура показує вищу гнучкість у розвитку нових функцій і простоту інтеграції складних сервісів, що є вирішальним для проєкту рівня KVANTOR.

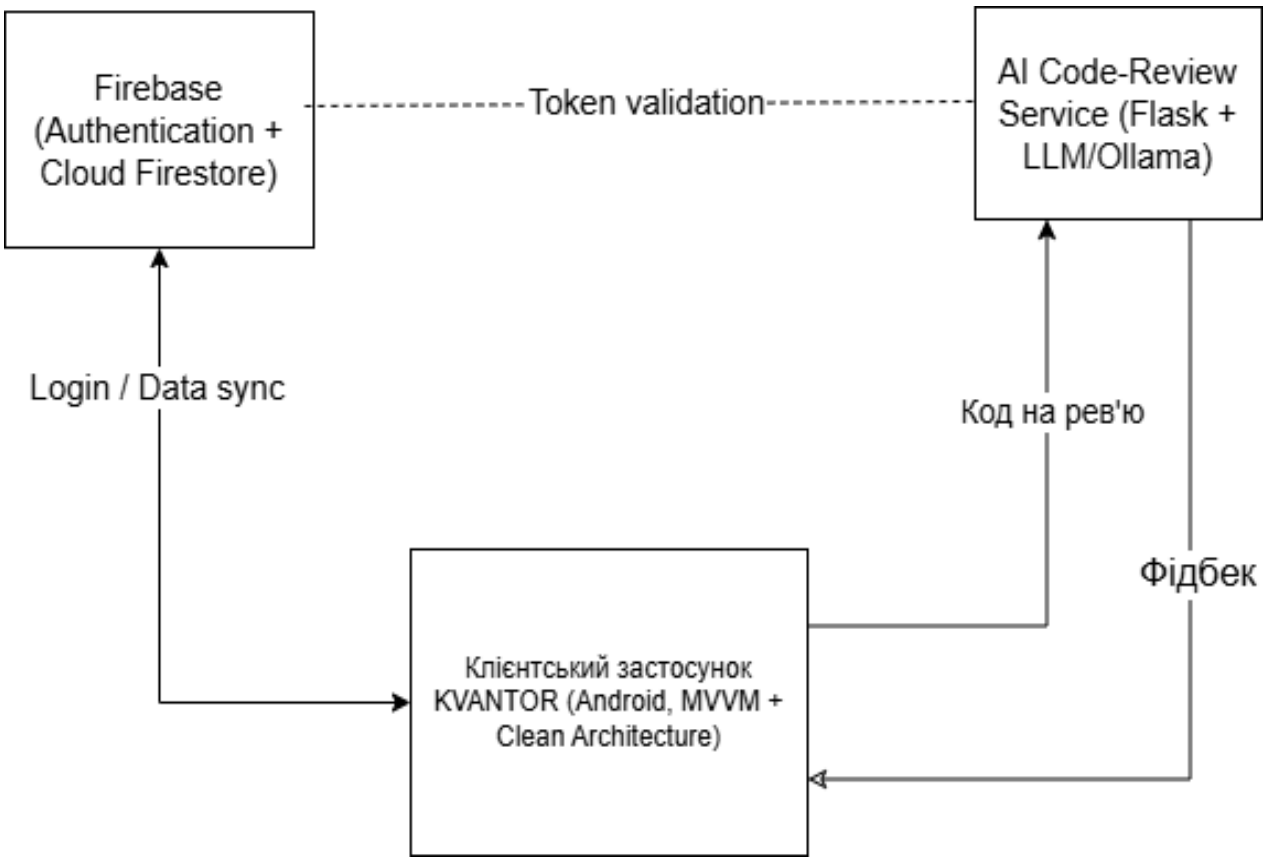


Рис. 3.1 Структура системи KVANTOR: взаємодія клієнтського додатку з Firebase та AI-сервісом

На рисунку 3.1 наведено логічну взаємодію трьох основних компонентів системи KVANTOR: клієнтського Android-додатка, Firebase та AI-сервісу. Додаток, реалізований із використанням MVVM та Clean Architecture, спочатку ініціює автентифікацію через Firebase Authentication, а потім синхронізує прогрес і метадані з Cloud Firestore. Коли користувач надсилає фрагмент коду, клієнт формує запит до Flask-мікросервіса з локальною LLM-моделлю Ollama, передаючи разом із кодом Firebase-токен. AI-сервіс верифікує токен через Firebase Admin SDK, аналізує код на прихованих тестах та повертає структурований фідбек рекомендації, виявлені помилки та оцінку якості рішення. Після цього додаток оновлює досягнення в Firestore та відображає користувачеві результати перевірки. Такий поділ ролей робить Firebase єдиним центром автентифікації й зберігання даних, а AI-компонент незалежним сервісом, що спрощує його масштабування й оновлення без необхідності перекомпіляції клієнтського коду.

3.2 Клієнтська архітектура застосунку

На рисунку 3.2 показано, як мобільний клієнт взаємодіє із зовнішніми сервісами і користувачами. Типовий користувач передає додаткові дані реєстрації, прогрес та аватари у клієнт, після чого ті синхронізуються з Firebase Cloud. Firebase повертає токен авторизації, дані профілю й зображення; на його базі AI Code-Review Server виконує валідацію ID-токена (штрихова лінія) перед обробкою запитів. Під час розв’язання завдань додаток надсилає код на рев’ю до AI-сервісу, отримує зворотний AI-фідбек і відразу відображає його в інтерфейсі. Окремо адміністратор може через клієнтський застосунок запитувати зведені звіти, що дозволяє відстежувати статистику користувачів без прямого доступу до внутрішніх сервісів У KVANTOR інтерфейс користувача побудовано на декларативних принципах, що дозволяють динамічно оновлювати UI відповідно до стану програми. Всі екрани та навігаційні шляхи реалізовані за допомогою сучасних компонентів Android UI, а вся обробка подій і бізнес-логіка виконується у ViewModel, що гарантує чітке розділення представлення та логіки. На доменному рівні сценарії використання оформлено у вигляді строго типізованих Use Cases із підтримкою асинхронних потоків даних на базі Kotlin Flow. Репозиторії, з яких ViewModel отримують дані з хмарного бекенду, працюють із DTO-об’єктами та впроваджуються через Dagger Hilt, що спрощує як модульне тестування, так і масштабування проєкту.

На рисунку 3.2 показано, як клієнтський додаток взаємодіє із зовнішніми сервісами та користувачем. Після введення реєстраційних даних, прогресу та обраного аватара інформація синхронізується з Firebase Cloud, який повертає токен авторизації, профільні дані й зображення. Цей токен потім використовується AI Code-Review Server для перевірки запитів (штрихова лінія). Коли користувач надсилає код на рев’ю, клієнт відправляє запит до AI-сервісу, отримує структурований фідбек і відразу відображає його в інтерфейсі. Адміністратор може створювати зведені звіти без прямого доступу до внутрішніх сервісів, використовуючи ті ж клієнтські інструменти.

Таке чітке розділення відповідальностей забезпечує декілька важливих переваг. По-перше, зміну будь-якого бекенда - наприклад, заміну Firebase чи AI-сервера - можна виконати без правок у частині інтерфейсу, адже клієнт працює з абстракціями. По-друге, використання єдиного механізму аутентифікації через Firebase підвищує загальний рівень безпеки: кожен запит до AI-сервісу автоматично верифікується через виданий токен. По-третє, завдяки вбудованому кешуванню Firestore основний навчальний контент доступний офлайн, а мережеві операції (наприклад, відправка коду на рев'ю чи формування звітів) виконуються лише за наявності інтернет-з'єднання. В результаті клієнтська архітектура залишається модульною, легко тестується і готова до масштабування.

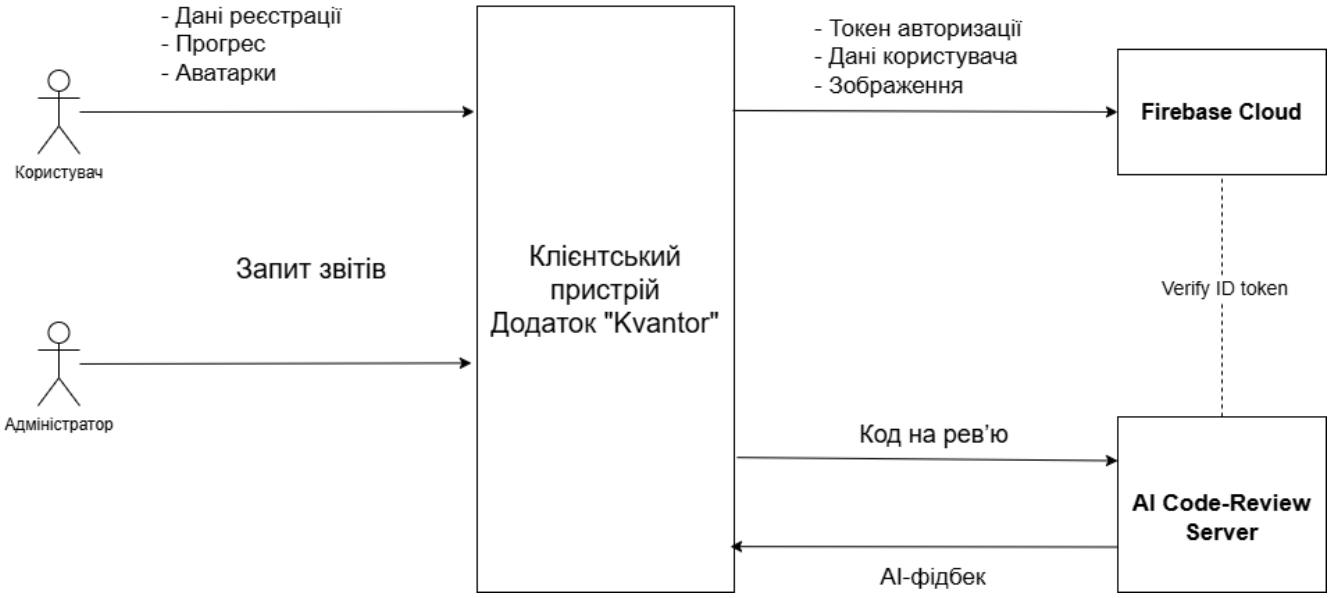


Рис. 3.2 Структура взаємодії клієнтського додатку KVANTOR із Firebase Cloud та AI Code-Review Server

3.3 Архітектура серверної частини й AI-інтеграція

У KVANTOR усі операції з аналізу коду винесено в окремий мікросервіс AI Code-Review на базі Flask. Після старту сервер піднімає HTTP-службу та встановлює з'єднання з локальним демоном Ollama [7] , у якому запущена квантизована модель phi-3.

Параметри підключення - адреса API, назва моделі, обмеження на кількість токенів - налаштовуються через змінні середовища, тому змінити модель або її розташування можна без правок у програмному коді.

Ollama - це легкий локальний демон для розгортання великих мовних моделей офлайн. Він дозволяє зберігати й запускати квантовані моделі без потреби у віддалених сервісах, забезпечуючи швидку генерацію відповідей і мінімальні затримки. Завдяки простому HTTP-API та підтримці контейнеризації, Ollama легко інтегрується до будь-яких бекенд-сервісів, зберігаючи дані в рамках локальної інфраструктури та гарантує конфіденційність користувацького коду.

Клієнтський додаток надсилає запити до двох REST-ендоінтів:

/review отримує JSON із полями task та code, будує оптимізований prompt і повертає структурований текст фідбеку;

/ask працює як загальний чат-інтерфейс, повертаючи лаконічні відповіді помічника.

Для обох маршрутів використовується спільний об’єкт requests.Session, який підтримує TCP-з’єднання відкритим, що зменшує затримки при серійному опитуванні. Тайм-аут на очікування відповіді встановлено на 60 секунд, щоб уникнути блокування мобільного інтерфейсу у разі довгої обробки. Така архітектура робить AI-сервіс незалежним, легко масштабованим і готовим до інтеграції нових моделей без зміни клієнтського коду.

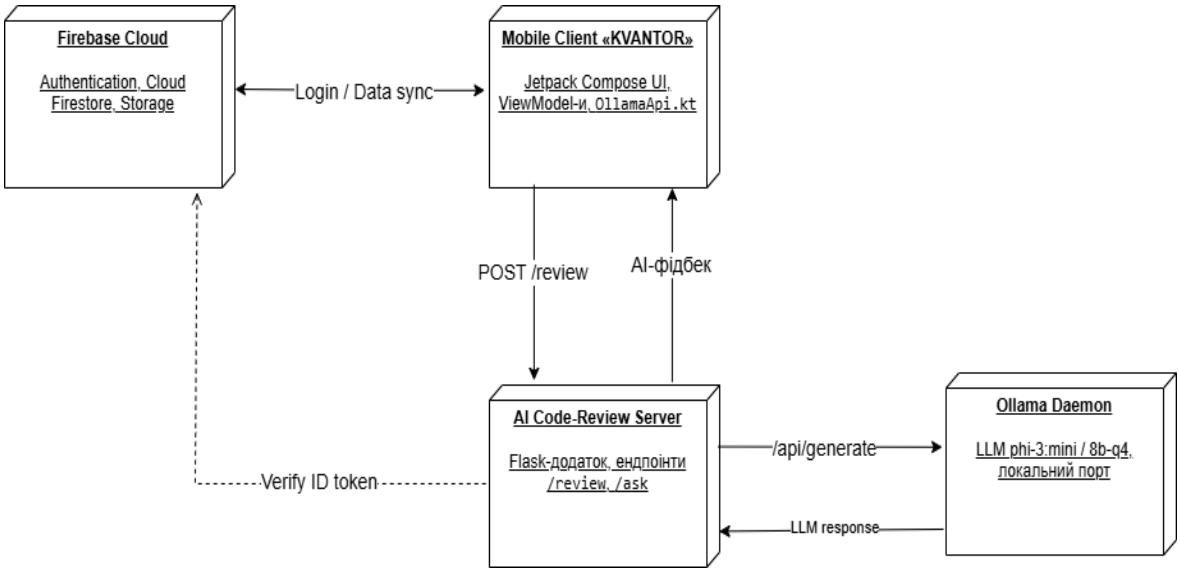


Рис. 3.3 Логічна схема мікросервісу AI Code-Review та його взаємодії з Firebase

На рисунку 3.3 схематично представлено внутрішню будову AI Code-Review сервісу та його взаємодію з Firebase Cloud: перед обробкою кожного запиту Flask-сервер може верифікувати ID-токен через Firebase Authentication, щоб переконатися в автентичності клієнта. У режимі розробки ця перевірка відключена, але в продакшн середовищі вона буде обов’язковою.

Для швидкого тестування в репозиторії є бат-файл, який автоматично послідовно піднімає Ollama з вказаною моделлю, запускає Flask-сервер і відправляє тестові POST-запити на обидва ендпоінти (/review та /ask). Такий підхід спрощує налагодження: розробник одразу бачить відповідь моделі без зайвих дій. У подальшому сервіс легко контейнеризується - Flask і Ollama можна винести в окремі Docker-контейнери або розгорнути на хмарі, просто змінивши URL-адреси в середовищних змінних.

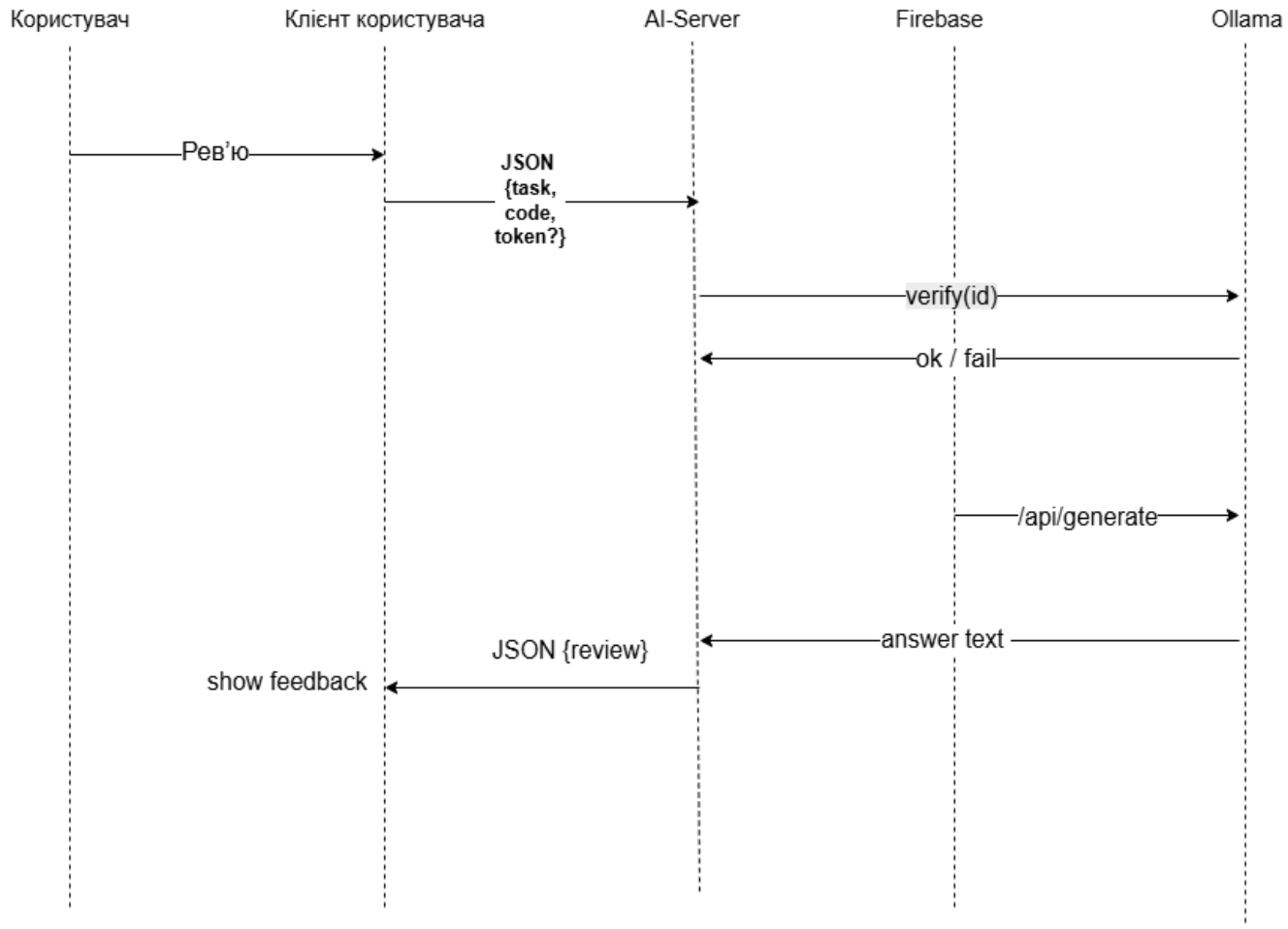


Рис. 3.4 Послідовність обробки запиту рецензії коду між клієнтом, AI-сервером та мовною моделлю

Послідовність обробки запиту рецензії подана на рисунку 3.4 Коли користувач натискає «Отримати рецензію» у LessonActivity, клієнтські ViewModel-и формують HTTP POST / review із JSON-пакетом {task, code, token}. AI-сервер приймає запит, за наявності токена звертається до Firebase Admin SDK для перевірки verify(id) та отримує відповідь «OK» або «Fail». Після успішної верифікації сервер обрізає код до допустимої довжини, конструює prompt і надсилає його демону Ollama через /api/generate. LLM phi-3 повертає текстову відповідь (дві строки: коротка порада та приклад поліпшеного фрагмента), яку Flask загортає у JSON {review} і відправляє назад мобільному клієнту. ViewModel оновлює стан екрана; інтерфейс негайно відображає AI-фідбек, тоді як усі обчислення виконуються на сервері без блокування UI.

3.4 Організація даних та реалізація правил безпеки в Firestore

У застосунку база даних Cloud Firestore виконує одразу кілька ролей - це сховище навчального контенту, журнал прогресу та «двигун» елементів гейміфікації. Документна модель Firestore органічно поєднується з офлайн-кешем Android-клієнта: після першого завантаження матеріали потрапляють у локальний кеш і залишаються доступними навіть за відсутності мережі, а запис змін відбувається асинхронно при відновленні зв’язку. Таким чином, вибір Firestore забезпечує і швидкий доступ із мінімальною кількістю операцій читання, і плавну роботу застосунку в умовах нестабільного мобільного інтернету.

Навчальний контент структуровано у дві колекції - Python та JavaScript. Кожен документ відповідає окремому модулю курсу й містить метадані (id, title, description) та масив pages[], де сторінки-об’єкти йдуть у потрібній послідовності («theory», «test», «coding» тощо). Зберігаючи всі сторінки безпосередньо усередині документа, ми навмисне відмовилися від вкладених підколекцій: один запит дістає повний урок, а значить економить батарею, мережу й квоту Firestore Reads. У разі редагування контенту модуль оновлюється цілим документом через адміністративний бекенд - клієнтських прав на запис у цих колекціях не існує.

Другий «полюс» схеми - колекція users, де ключем кожного документа є UID, виданий Firebase Authentication. Щойно новий студент завершує реєстрацію, створюється його профіль, у якому одномоментно зберігається все, що формує ігрову мотивацію: coins, lives, hints, поточний стрік (streakDays) та вибраний аватар. Тут само лежить денормалізований об’єкт progress - мапа, у якій ключами виступають id модулів, а значення містить статус виконання. Ще простіше представлено achievements: це масив рядків-ідентифікаторів того, що вже заробив гравець («JS_SAMURAI», «PY_MASTER», «Перший крок» тощо). Єдина «ціна» такої денормалізації - більший розмір профільного документа, зате клієнту досить одного читання, щоб намалювати екран статистики, магазину та панель досягнень; окрема колекція progress або achievements просто не потрібна.

Безпечний доступ гарантують правила Firestore, побудовані за принципом least privilege. Кожен автентифікований користувач має право читати контент модулів, але не може його змінювати; створення та оновлення навчальних матеріалів дозволено виключно серверному обліковому запису адміністратора. З іншого боку, документ профілю може читати й оновлювати лише сам власник - поле request.auth.uid перевіряється на повну відповідність UID документа. Такі правила не допускають перегляду чужих досягнень чи монет та одночасно не створюють зайвого навантаження на клієнтську логіку: більшість перевірок виконується безпосередньо у сервісі Firestore ще до того, як запит дістанеться застосунку.

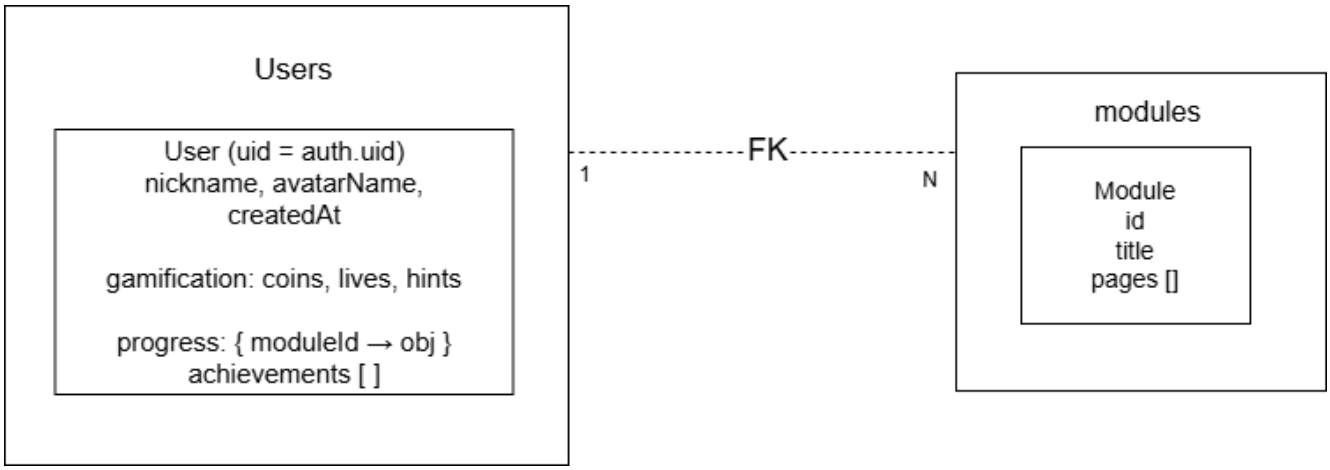


Рис. 3.5 Логічна схема даних Firestore у KVANTOR

На рисунку 3.5 подано узагальнену логічну схему бази даних Firestore. Прямокутник users позначає колекцію профілів; вкладений блок демонструє, що в одному документі зберігаються як облікові поля (UID, нікнейм, дата створення), так і всі атрибути гейміфікації — caldo монет, кількість «життів», підказок, а також денормалізовані структури progress (мапа moduleId → об’єкт стану) й achievements (масив отриманих відзнак). Праворуч розміщено колекцію modules, де кожен документ містить метадані уроку та масив pages[] із повним змістом модуля. Пунктирна стрілка «1 → N» показує, що ключі з об’єкта progress напряму посилаються на id документів modules: один користувач може мати записи про багато модулів, а один модуль фігурує у прогресі багатьох користувачів. Таке денормалізоване посилання зменшує кількість операцій читання, спрощує офлайн-кешування й полегшує масштабування — щоби додати курс з нової мови, достатньо створити ще одну колекцію modules без змін у структурі профілю чи правилах безпеки.

3.5 Зв’язок між компонентами

У розробці KVANTOR використано компактний, але достатньо різноманітний стек. Клієнтська частина написана повністю мовою Kotlin, а інтерфейс створено за допомогою Jetpack Compose і Navigation Component, тож усі екрани - від реєстрації й авторизації до проходження модуля та магазину – описуються декларативно й швидко перебудовуються під зміну стану. За асинхронні операції відповідають coroutines і Flow: саме вони дозволяють без пауз оновлювати уроки з Firestore, отримувати результати AI-рев’ю та не блокувати UI. Під капотом Clean Architecture зв’язки між шаром даних і ViewModel-ами роз’єднує Dagger Hilt, завдяки чому підмінні реалізації легко підставити на тестах або під час переходу на іншу бекенд-службу.

Уся аутентифікація концентрується у Firebase Authentication: після реєстрації чи входу клієнт отримує токен, який додається до кожного чутливого запиту. Навчальний контент і гейміфікаційні показники зберігаються у Cloud Firestore. Тут лежать окремі колекції модулів для Python і JavaScript, а також документи

користувачів, усередині яких денормалізовано зберігаються монети, «життя», підказки, мапа progress за ідентифікаторами уроків і масив achievements. Firestore підтримує офлайн-кеш, тому сторінка теорії чи тест відкривається навіть без зв’язку, а всі накопичені зміни синхронізуються, щойно з’являється мережа. Статичні файли (аватарки) зберігає Firebase Storage, отже до них можна звертатися за тією ж автентифікаційною сесією.

Шар штучного інтелекту винесений окремо від мобільного клієнта. Легкий Flask-сервер приймає POST-запит /review із кодом користувача та, при ввімкненій перевірці, верифікує одержаний ID-токен через Firebase Admin. Далі сервер формує короткий prompt і надсилає його на локальний демон Ollama, де розгорнута LLM phi-3-mini у квантованій 8-бітній конфігурації. Відповідь моделі - одна порада та приклад поліпшеного рядка коду - повертається тим самим маршрутом, а ViewModel миттєво відображає її в інтерфейсі. Така побудова зберігає мобільний застосунок легким: складна обробка йде на сервері, а клієнт отримує лише вже готовий текст.

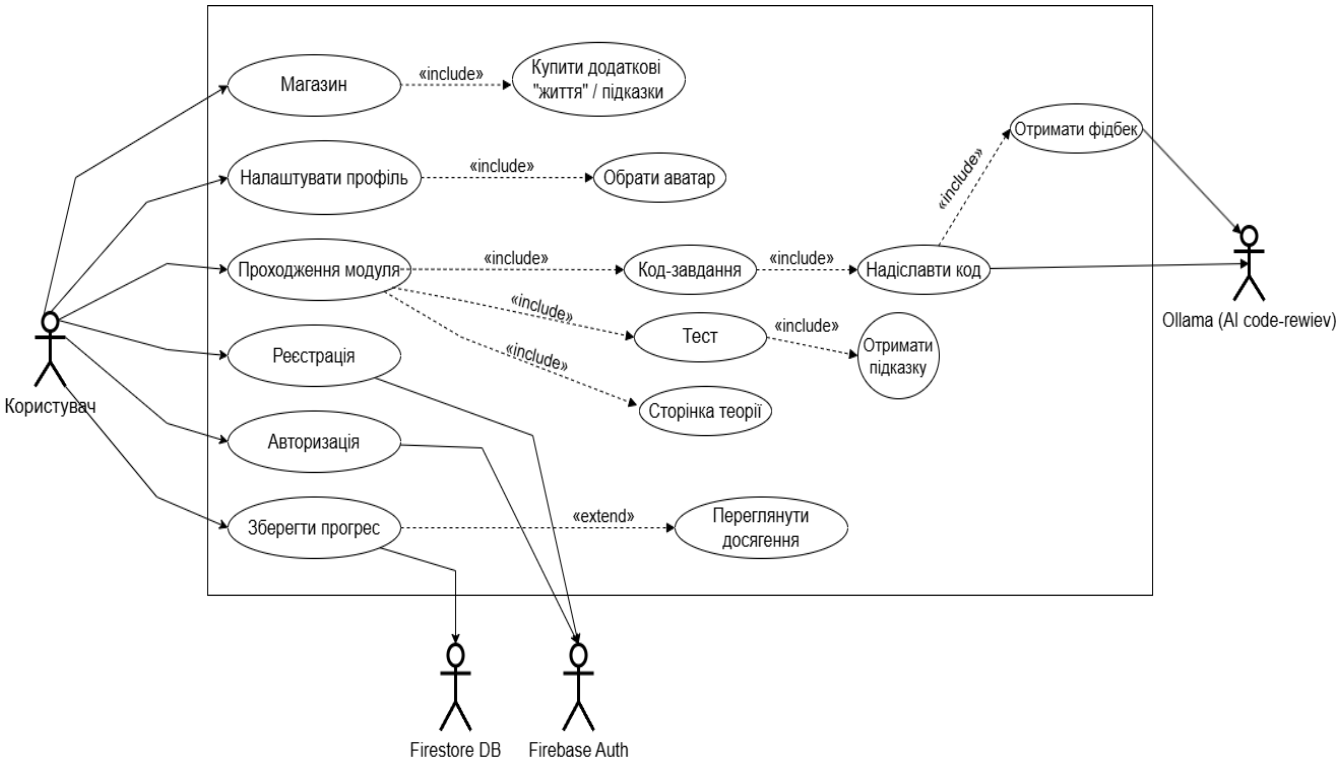


Рис. 3.6 Діаграма прецедентів, що ілюструє порядок дій користувача в Android-додатку, взаємодію з Firebase (Auth, Firestore) та AI-сервісом для коду-рев’ю

На рисунку 3.6 показано, як саме ці технології «розкладаються» по основних варіантах використання. Ліворуч актор «Користувач» ініціює дії в Android-клієнті: реєструється, проходить модулі, купує додаткові «життя», надсилає код на рецензію. Прецеденти «Реєстрація» та «Авторизація» звертаються до Firebase Auth, після чого будь-які звернення до Firestore або AI-сервера передають той самий токен.

Прецедент «Проходження модуля» включає сторінки теорії, тести й код-завдання; коли користувач натискає «Надіслати код», клієнт надсилає HTTP-запит на Flask, а той, сформувавши prompt, викликає Ollama; зворотний AI-фідбек повертається у вигляді JSON і відразу ж потрапляє у Compose-UI. Усі зміни прогресу та досягнень записуються в документ користувача, а магазин дістає поточну кількість монет із того самого джерела - додаткові колекції не потрібні, що скорочує кількість запитів і спрощує кешування.

Таким чином Kotlin і Jetpack Compose відповідають за швидкий і реактивний інтерфейс, Firebase - за безпечне зберігання даних і автентифікацію, а зв’язка Flask + Ollama надає можливість отримувати миттєвий інтелектуальний фідбек, залишаючи клієнт мінімальним за вагою й енергоспоживанням.

Обрана архітектура дає змогу незалежно масштабувати всі компоненти: додати нову мову програмування - це лише ще одна колекція модулів; оновити модель LLM - досить перезапустити Ollama на сервері; а розширити гейміфікацію можна, додавши нові поля до документа users без зміни структури решти системи.

4 ЗАСОБИ РЕАЛІЗАЦІЇ ЗАСТОСУНКУ ТА ТЕСТУВАННЯ

4.1 Технологічний стек та середовище розробки

Android Studio [8] з Gradle KTS і Version Catalog (libs.versions.toml) дають централізоване керування залежностями та пришвидшують збірку. Плагіни (компіляція Android, Kotlin, Compose) підключено через alias(libs.plugins…), що уніфікує їх версії. Єдиний модуль :app у settings.gradle.kts мінімізує залежності та оптимізує інкрементальну збірку.

Compose BOM гарантує сумісність версій UI-бібліотек (activity-compose, ui-tooling, material3) без ручного узгодження. Ін’єкцію залежностей і управління життєвим циклом ViewModel-ів забезпечують lifecycle-runtime-ktx, lifecycle-viewmodel-ktx і відповідні Compose-адаптери, що дозволяє відокремлювати UI-складову від бізнес-логіки й спрощує модульне тестування без прямої залежності від Android-фреймворку.

Retrofit (з Gson) і OkHttp-логер використовуються для зручного серіалізування JSON та моніторингу HTTP-трафіку. Асинхронні завдання обробляються через Kotlin Coroutines, а Compose увімкнено через buildFeatures.compose для швидкого HMR-редагування інтерфейсів. Всі налаштування Java- і Kotlin-опцій спрямовані на сумісність із версією JVM 11, що вказано в compileOptions та kotlinOptions, а підтримка Compose активується через buildFeatures.compose = true. Завдяки такому середовищу розробки команда отримує гнучкий інструментарій для «гарячого» редагування макетів, швидкої збірки й простого додавання нових залежностей, що суттєво скорочує час від задуму до робочого прототипу.

4.2 Організація збірки та тестування

Проєкт KVANTOR має монолітну структуру Android-модулю :app, підключену у settings.gradle.kts. Для централізованого керування версіями бібліотек використано Version Catalog (libs.versions.toml), який зберігає

налаштування AGP, Kotlin, Compose BOM та ключові залежності (Firebase, Retrofit, Coroutines тощо). Плагіни підключаються через alias(libs.plugins…) у build.gradle.kts - це забезпечує єдину точку зміни версій плагінів com.android.application, org.jetbrains.kotlin.android і org.jetbrains.kotlin.plugin.compose.

Конфігурація збірки передбачає дві стандартні build-types - debug і release. У release активовано R8/ProGuard з набором правил із proguard-android-optimize.txt та власним файлом proguard-rules.pro. Версія застосунку (code та name) прописані у defaultConfig, їх можна автоматизувати через віддалений скрипт або CI, але наразі вони фіксовані. Підпис release-APK здійснюється сертифікатом, ключі якого зберігаються у зашифрованих властивостях Gradle (gradle.properties) і не комітяться в репозиторій.

Для інтеграції з Firebase Auth і Firestore у app застосовано плагін com.google.gms.google-services, який підключається наприкінці секції plugins. Це гарантує автоматичне зчитування конфігурації з google-services.json на етапі збірки. HTTP-клієнт (Retrofit + Gson), корутини, Lifecycle та Compose-бібліотеки також входять до залежностей, тому повна збірка APK триває командою ./gradlew clean assembleRelease або /gradlew assembleDebug

CI/CD без сторонніх сервісів реалізовано локальними shell-скриптами. Наприклад, build_release.sh викликає assembleRelease, копіює готовий APK у папку release-candidates/ з часовою міткою, а потім запускає deploy.sh. Останній піднімає серверну частину через Docker Compose (контейнери flask-api і ollama-phi3), використовуючи docker-compose.yml у корені проекту. Завдяки цьому мобільний клієнт і сервер працюють в ідентичних середовищах, а ручний деплой в Google Play Console зводиться до завантаження APK зі сформованої папки.

4.3 Система контролю версій

Розробка програмного забезпечення неможлива без використання системи контролю версій (СКВ), особливо коли йдеться про дипломний проєкт з довготривалим циклом розробки та потенційно кількома співрозробниками. У

випадку Kvantor було обрано розподілену систему контролю версій Git з розміщенням репозиторію на платформі GitHub. [9] Git зарекомендував себе як надзвичайно швидка та гнучка СКВ з відкритим кодом, що дозволяє командам експериментувати з кодом без страху порушити основну гілку розробкиabout.gitlab.com. Кожен розробник має повну копію історії проєкту на своєму комп'ютері, тому може працювати офлайн і виконувати коміти автономно. Це є великою перевагою Git над централізованими системами (як-от SVN): навіть без доступу до інтернету можна зафіксувати зміни, переглянути історію, створити гілку для нової фічі.

У процесі розробки Kvantor Git використовувався для ведення журналу змін: кожна реалізована функція або виправлена помилка оформлювалась окремим commit з описом. Такий підхід не лише допомагає відстежувати, хто і коли вніс певні зміни, але й дозволяє при потребі повернутися до попередньої версії (наприклад, якщо новий код спричинив небажану поведінку, можна виконати revert проблемного коміту).

Також Git полегшує паралельну роботу над проєктом через механізм гілкування (branching). У нашому випадку основна гілка main містила стабільну версію, а для розробки великих підфункцій створювалися тимчасові гілки (наприклад, feature/ai-review для роботи над інтеграцією AI). Після завершення роботи та тестування зміни об'єднувалися (merge) назад у головну гілку. Git ефективно зливає зміни навіть якщо кілька розробників працювали над різними модулями, і здатен автоматично виявляти конфлікти, якщо правки стосувались одних і тих самих рядків коду.

GitHub як платформа для хостингу репозиторію додав свої переваги. По-перше, це віддалене резервне копіювання: кожен пуш на GitHub гарантує, що копія коду зберігається у хмарі і доступна з будь-якого місця. По-друге, GitHub надав інструменти для командної роботи: система Pull Request для огляду коду (код-рев'ю), трекер задач та помилок (Issues) для планування роботи, і базову безперервну інтеграцію (GitHub Actions) для автоматичного запуску тестів чи збірки. У проєкті Kvantor було задіяно Issues для відстеження прогресу розробки:

кожен модуль або задача дипломної роботи відображався як Issue, який закривався після завершення реалізації. Це допомогло структурувати роботу і мати історію вирішених проблем, що може бути корисним при написанні звіту або аналізі проектних ризиків.

Альтернативними платформами для розміщення Git-репозиторію могли б бути GitLab чи Bitbucket. GitLab привабливий тим, що є відкритим ПЗ і може бути розгорнутий на власному сервері, а також має розширені можливості CI/CD «з коробки». Bitbucket від Atlassian, своєю чергою, дозволяє приватні репозиторії для команд до 5 осіб безкоштовно та інтегрується з іншими продуктами Atlassian (Jira, Trello). Втім, у контексті нашого проєкту ці переваги не були вирішальними. GitHub обрано завдяки його популярності та простоті: практично кожен розробник знайомий із GitHub, а для навчального або відкритого проєкту це своєрідний «стандарт». До того ж, GitHub нині теж підтримує безкоштовно необмежену кількість приватних репозиторіїв і має достатні можливості CI через GitHub Actions. Таким чином, ми не мали вагомих причин ускладнювати інфраструктуру іншим хостингом – GitHub повністю задовольнив потреби Kvantor.

Не можна оминути увагою й альтернативні системи контролю версій, хоча б для обґрунтування вибору Git. Історично склалося, що до Git розробники широко використовували Subversion (SVN) – централізовану СКВ. У порівнянні з Git, SVN вимагає постійного зв’язку з центральним сервером для більшості операцій і менш гнучкий у плані гілкування та злиття гілок. Для невеликого проекту з одним розробником різниця могла би здатись незначною, але навіть у таких умовах Git забезпечує більш швидку роботу (коміти та дифи локально) і надійність (наявність повної копії дозволяє відновити проект з локального репозиторію у разі збою сервера). В таблиці порівняння VCS можна було б вказати, що Git перевершує SVN за такими параметрами: офлайн-робота (Git дозволяє майже всі дії офлайн, SVN – ні), злиття гілок (Git оптимізований для merge, SVN часто породжує складнощі), швидкість (локальні операції Git дуже швидкі, на відміну від мережевих SVN), розповсюдженість (Git став де-факто стандартом, велика спільнота, багато

інструментів навколо). У результаті вибір Git для Kvantor був очевидним як з точки зору індустріальних стандартів, так і з міркувань зручності та надійності.

Отже, система контролю версій Git у поєднанні з хостингом на GitHub стала невід'ємною частиною засобів реалізації проєкту Kvantor. Вона забезпечила впорядкованість роботи з кодом, дозволила безпечно експериментувати з новими функціями, спростила командну взаємодію та створила історію розвитку проєкту, що може бути проаналізована й у дипломній роботі. Використання Git підкреслює професійний підхід до розробки: навіть навчальний застосунок створено за сучасними практиками інженерії програмного забезпечення, що гарантує його підтримуваність та розширюваність у майбутньому.

4.4 Архітектура користувацького інтерфейсу та опис екранних модулів.

Узагальнена архітектура інтерфейсу «Kvantor» розподілена на автономні екранні модулі, пов'язані між собою простою навігаційною схемою на рисунку 4.1. Перший рівень складається з екранів реєстрації та авторизації, через які користувач потрапляє на головне меню. З головного меню доступні вибір курсу (Python, JavaScript, AI-помічник), профіль користувача та магазин, кожен із яких реалізований як окремий компонент із власним набором підекранів (теорія, тестування, написання коду, AI-рев'ю) і спільним шаром авторизованих запитів до Firebase.

Кожний модуль інкапсулює свою логіку й UI-компоненти, що спрощує автоматизоване тестування і масштабування інтерфейсу. Наприклад, головне меню містить три тематичні кнопки, перемикач теми й аватар, а екран AI-помічника - поле вводу, область повідомлень і індикатор відповіді. Такий поділ забезпечує швидке розширення функціоналу без порушення цілісності дизайну.

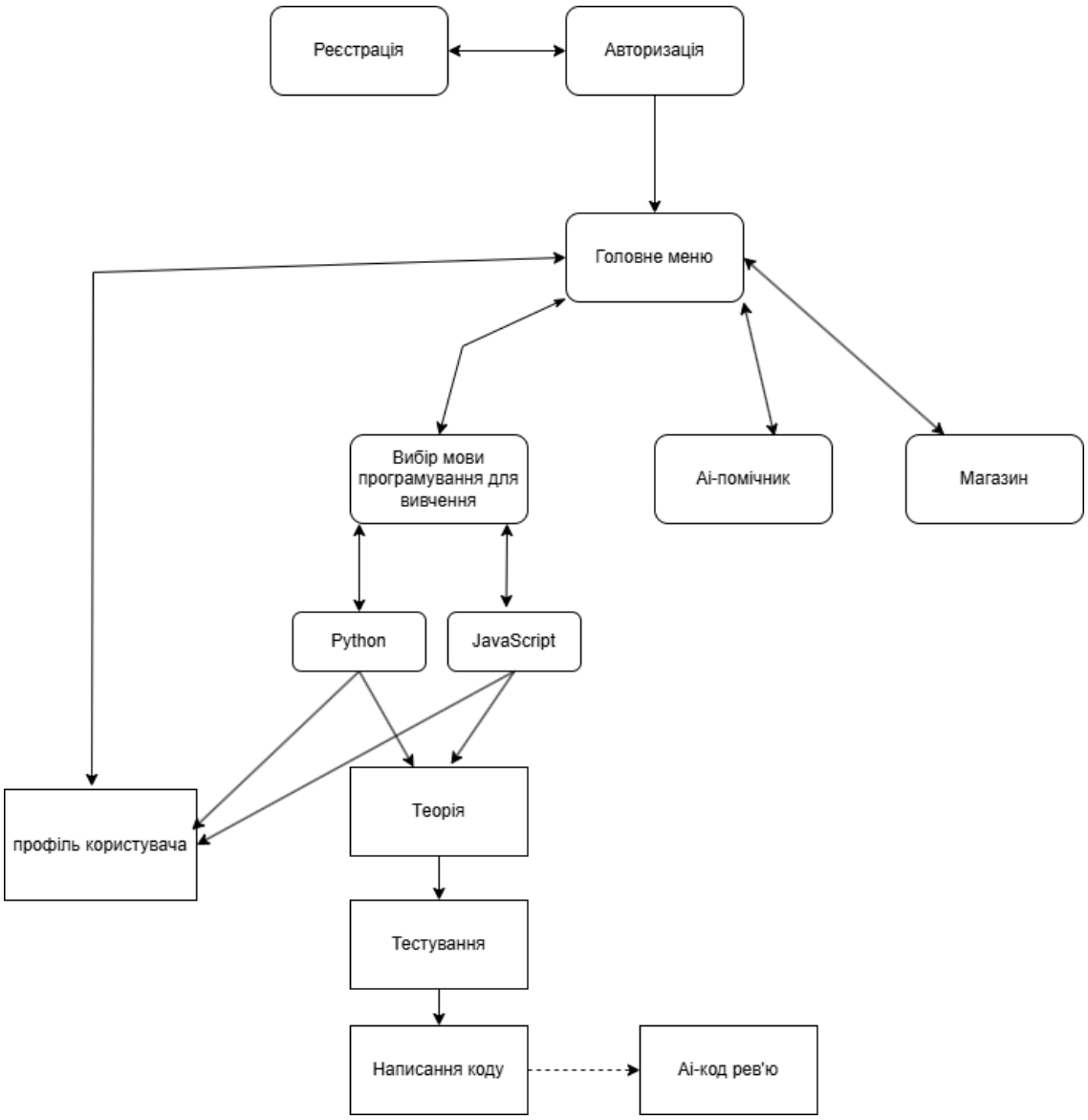


Рис. 4.1 Структурна схема екранів мобільного застосунку “Kvantor”

4.5 Екран аутентифікації

Екран аутентифікації в Kvantor покликаний забезпечити зручний та безпечний вхід у систему. Для цього реалізовано дві активності – AuthActivity та RegisterActivity, оформлені в єдиному стилі через тему KvantorTheme із кастомним шрифтом Rubik. AuthActivity відображає поля для введення електронної пошти й паролю, причому поле паролю оснащене іконкою “око” для перемикання видимості символів. Після натискання кнопки «Увійти» спрацьовує стандартний метод FirebaseAuth.signInWithEmailAndPassword, а в разі помилки користувач бачить повідомлення через Snackbar або Toast. Якщо користувач ще не

зареєстрований, натискання «Реєстрація» переводить до RegisterActivity, яка в аналогічному інтерфейсі створює обліковий запис за допомогою FirebaseAuth.createUserWithEmailAndPassword і повертає на екран входу.

Авторизація через Google здійснюється за допомогою GoogleSignInClient: при натисканні «Продовжити з Google» відкривається стандартний діалог вибору акаунту, а результат обробляється в onActivityResult, де отримані облікові дані передаються в FirebaseAuth.signInWithCredential. Після успішного входу - незалежно від способу - викликається функція navigateBasedOnUserProfile(), яка звертається до Firestore, перевіряючи наявність заповненого профілю. Якщо запису ще немає, користувач потрапляє на екран налаштування профілю (ProfileSetupActivity), інакше - одразу до вибору курсу (CourseSelectionActivity) рисунок 4.2, 4.3).

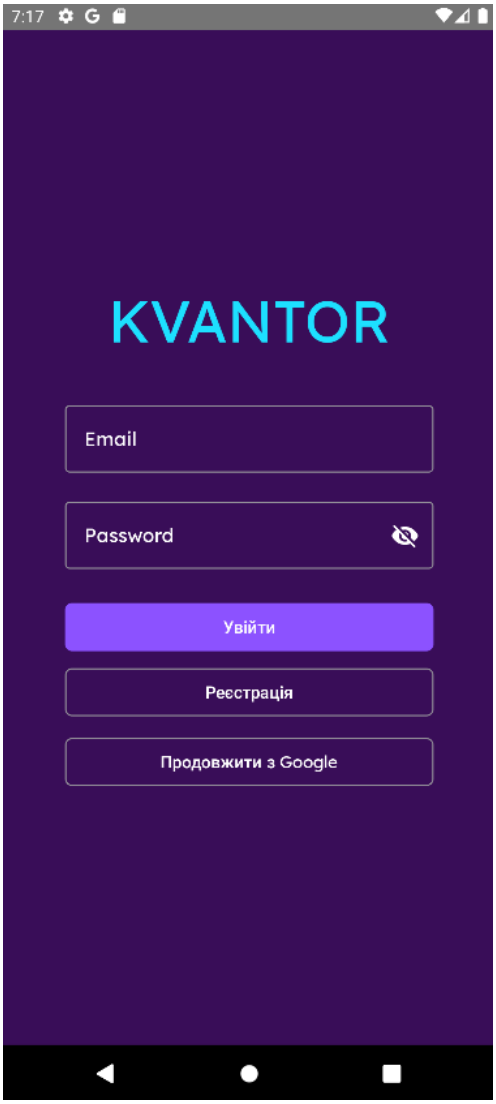


Рис. 4.2 Екран аутентифікації користувача в додатку “Kvantor“

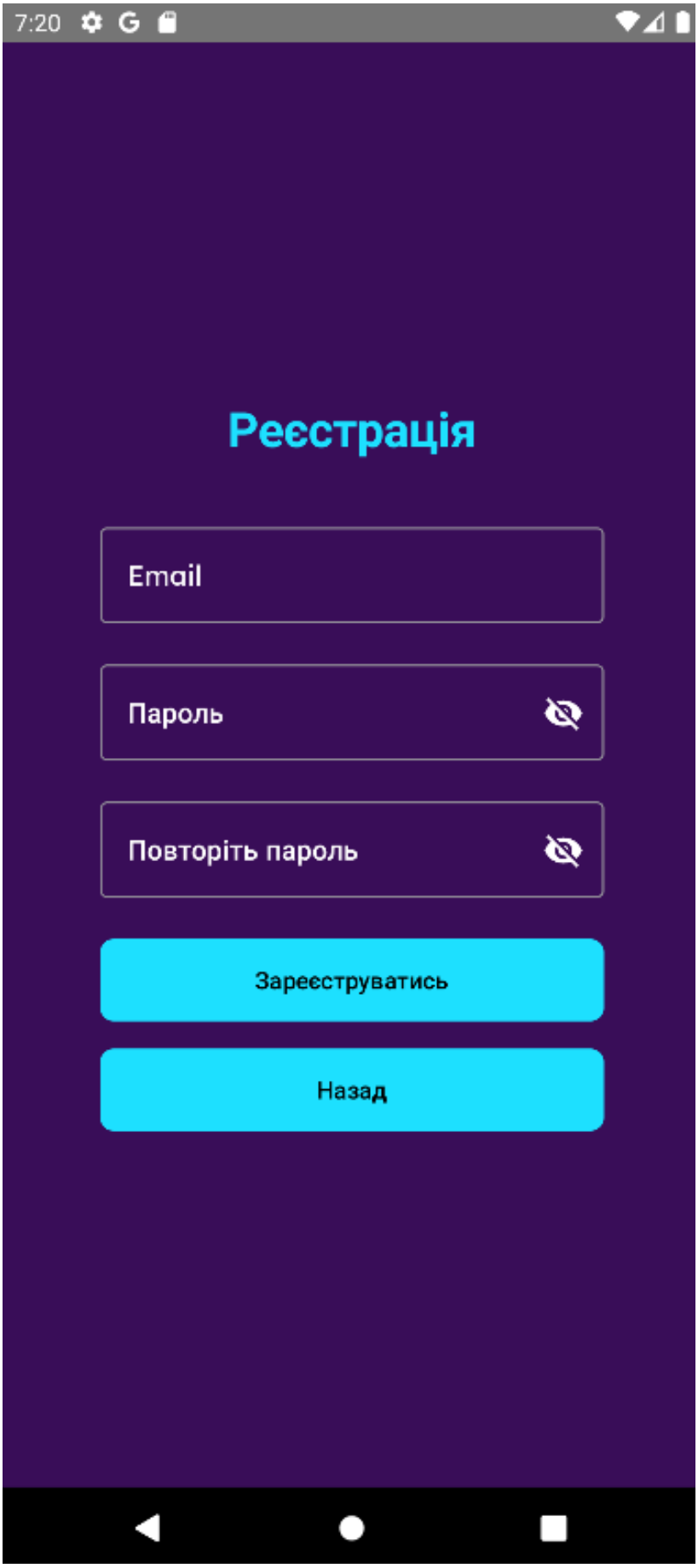


Рис. 4.3 Екран реєстрації нового користувача в додатку „Kvantor“

4.6 Екран налаштування профілю

Після успішної аутентифікації користувача на AuthActivity відкривається ProfileSetupActivity, яка в методі onCreate через setContent { KvantorTheme { ProfileSetupScreen { nickname, avatarName → … } } } відображає екран налаштування профілю зображений на рисунку 4.4. При натисканні кнопки «Продовжити» у ProfileSetupScreen викликається корутина в lifecycleScope, яка через UserBootstrapper.createUserSkeleton(uid, nickname, avatarName) створює в Firestore каркас профілю та ініціалізує початкові ачівки. Після успіху з'являється toast-повідомлення про першу ачівку й відбувається перехід до вибору курсу (CourseSelectionActivity).

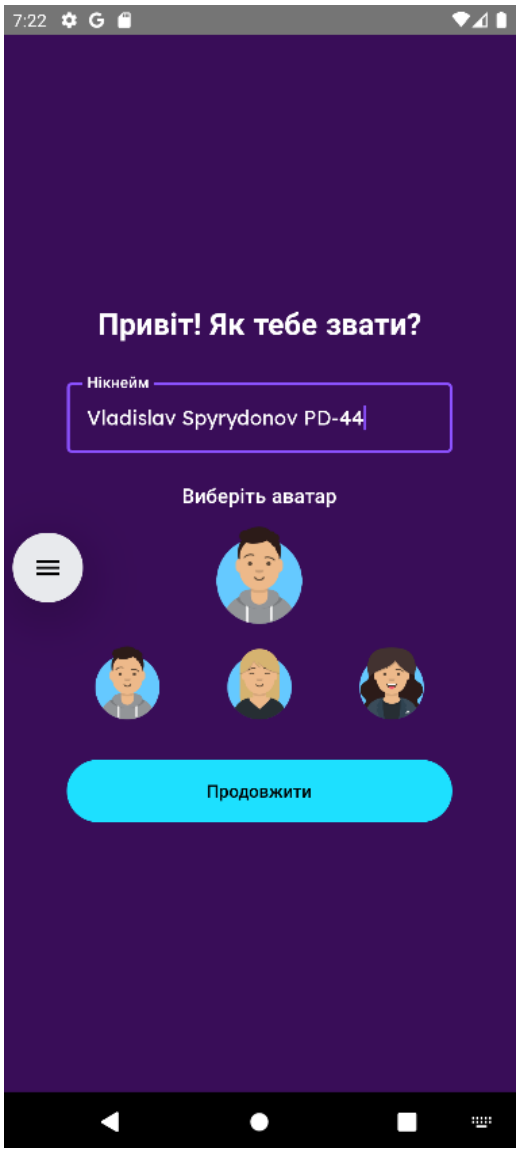


Рис. 4.4 Екран налаштування профілю користувача в додатку "Kvantor"

У ProfileSetupScreen використано Column із фоновим кольором та відступами, всередині якого спочатку виводиться заголовок-питання «Привіт! Як тебе звати?» та OutlinedTextField для введення нікнейму. Далі з’являється текст «Виберіть аватар» та велика кружечкова іконка обраного аватара, що при кліку розгортає ряд із трьома варіантами (avatar1, avatar2, avatar3), з яких користувач може обрати один, дивіться на рисунку 4.4. Після вибору аватара та введення нікнейму кнопка «Продовжити» перевіряє, чи нікнейм не порожній (інакше показує тост «Введи нікнейм»), і за допомогою колбеку onContinue(nickname, selectedAvatar) передає ці дані в Activity. Так забезпечується збереження налаштувань профілю та гнучке управління переходами в подальші розділи додатку.

4.7 Екран вибору курсу

Вибір курсу реалізовано в CourseSelectionActivity, яка при створенні викликає setContent { KvantorTheme { CourseSelectionScreen(...) } }. Activity зберігає стан теми (світла/темна) і передає його в Composable, а також обробляє натискання на курс через метод openCourse(courseId: String): в debug-зборці миттєво відкривається потрібна Activity, а в release-версії - спочатку оновлюється поле selectedCourse в документі користувача Firestore, і лише після успіху відбувається перехід, дивись рисунок 4.5.

Компонент CourseSelectionScreen використовує Column із вертикальним скролом і відступами для адаптації під різні розміри екрана. У верхньому рядку (Row) розташовані IconToggleButton для зміни теми, кнопка-кошик для переходу в ShopActivity й клікабельний аватар користувача (довантажується з Firestore іконка профілю). Далі виводиться заголовок «Обері курс» із великим ікон-скелетом та трьома змодельованими кнопками курсів.

Кожна кнопка курсу - це CourseButton, яка через MutableInteractionSource відслідковує натискання й анімує масштаб, а при кліку викликає переданий onSelect. Для курсів Python і JavaScript логіку натискання обробляє onSelect, тоді як для AI-помічника відразу викликається AiAssistantActivity. Колірна схема та стилі

текстів беруться з MaterialTheme.colorScheme, що забезпечує єдину візуальну консистентність із рештою додатку.

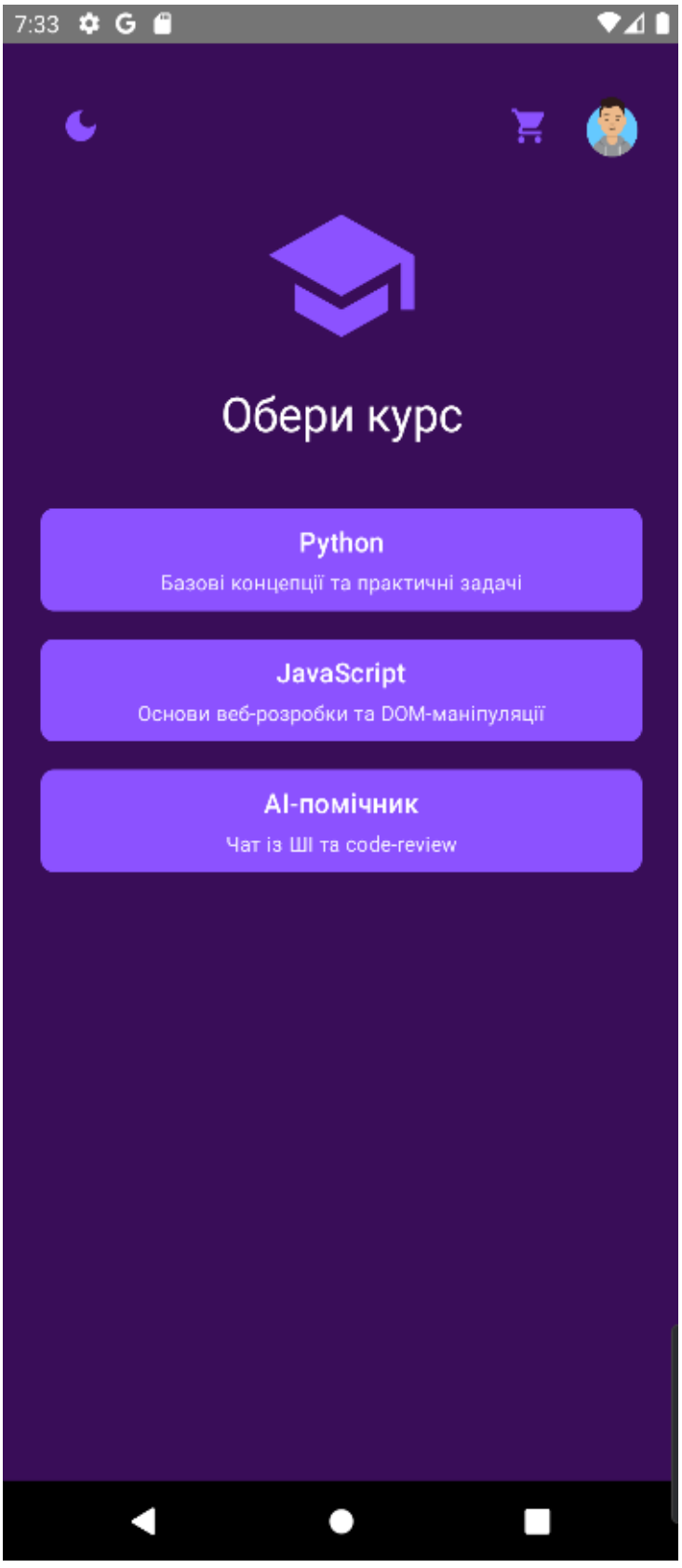


Рис. 4.5 Екран вибору курсу в додатку “Kvantor”

4.8 Екран профілю

У ProfileActivity в методі onCreate через setContent { KvantorTheme { ProfileScreen() } } відображається ProfileScreen, який у LaunchedEffect(uid) асинхронно завантажує з Firestore документ користувача (поле nickname і назву аватара), а також підколекцію achievements на рисунку 4.6. Дані зберігаються в стані (mutableStateOf): рядок нікнейму, ресурс іконки аватара (avatarResId) та список Achievement(id, unlocked). Кількість розблокованих досягнень обчислюється як achievements.count { it.unlocked }.

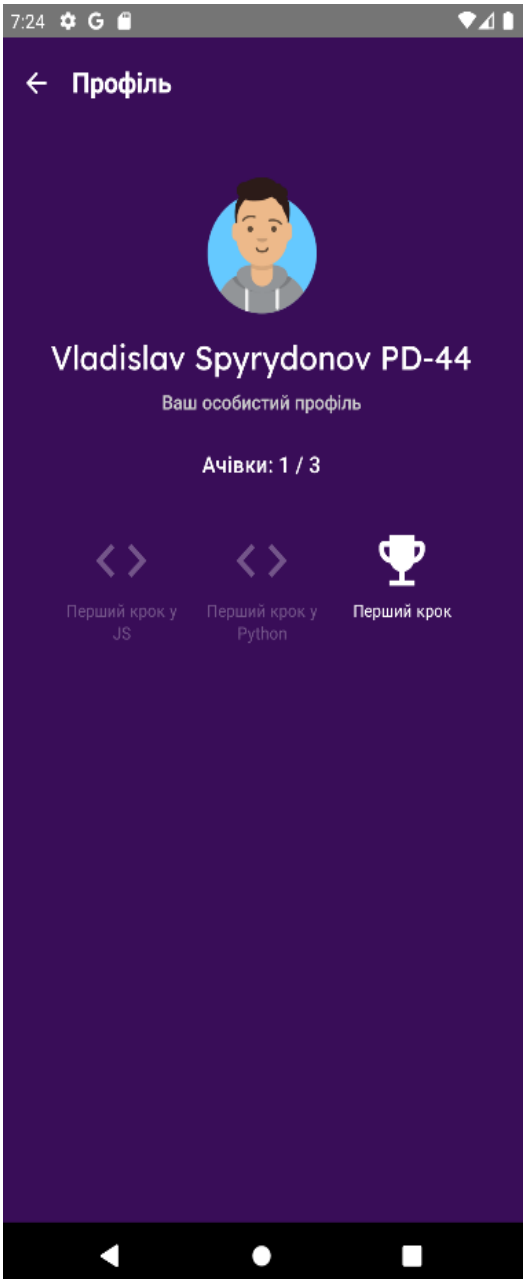


Рис. 4.6 - Екран профілю користувача в додатку “Kvantor”

У верхній частині екрану використано TopAppBar із заголовком «Профіль» та кнопкою «Назад», що викликає finish(). Після нього іде Column із відступами: великий круглий аватар, нікнейм користувача, підзаголовок «Ваш особистий профіль» та текстовий лічильник “Ачівки: N / M”.

Для відображення самих досягнень застосовано LazyVerticalGrid(columns = GridCells.Fixed(3)), де для кожного елемента Achievement обирається іконка й текстова мітка за id. Якщо досягнення unlocked == false, його колірна прозорість зменшена (alpha = 0.3f), що дає візуальний зворотний зв’язок про незавершені цілі. Цей підхід забезпечує гнучке й масштабоване відображення будь-якої кількості ачівок.

4.9 Екран курсу Python та JavaScript

Обидві активності – MainActivity для Python і JavaScriptMainActivity для JS – задають свій інтерфейс через setContent { KvantorTheme { … } }. У верхній панелі розташовані кнопка «Назад» (повернення до вибору курсу), перемикач теми (Python-екран) або іконка нічного режиму (JS-екран), а також аватарка користувача, підвантажена з Firestore і клікабельна для переходу до ProfileActivity. Під цією панеллю великим шрифтом виводиться назва обраного курсу. Список доступних модулів формується як колекція пар назва → опис, яку Composable-метод ітерує у вертикальному скрол-стовпці. Кожен пункт упаковано в кольоровий контейнер (background(accent)), що реагує на натиск (розгортає опис за допомогою AnimatedVisibility), а шрифт Rubik і кольорова палітра з MaterialTheme.colorScheme гарантують єдину стилістику. Нижче - дві дружні за дизайном кнопки: у Python-версії вони мають ефект стискання при натиску, цей ефект зображений на рисунку 4.7, у JS-версії - фіксований розмір і назву “Почати з початку” та “Продовжити курс”, дивись рисунок 4.8.

Натиск на «Почати курс з початку» ініціює запис { moduleIndex: 0, pageIndex: 0 } у полі progress.python або progress.javascript документа користувача в Firestore, після чого запускається LessonActivity з параметром courseType. У свою чергу LessonActivity і LessonViewModel завантажують модулі з відповідних колекцій (modules чи modules_js), відновлюють останню позицію користувача,

відображають контент (теорію, тестові запитання, практичні завдання та фінальний екран) і підтримують гейміфікаційні елементи (життя, підказки, монети), що робить архітектуру обох курсів уніфікованою та легко масштабованою.

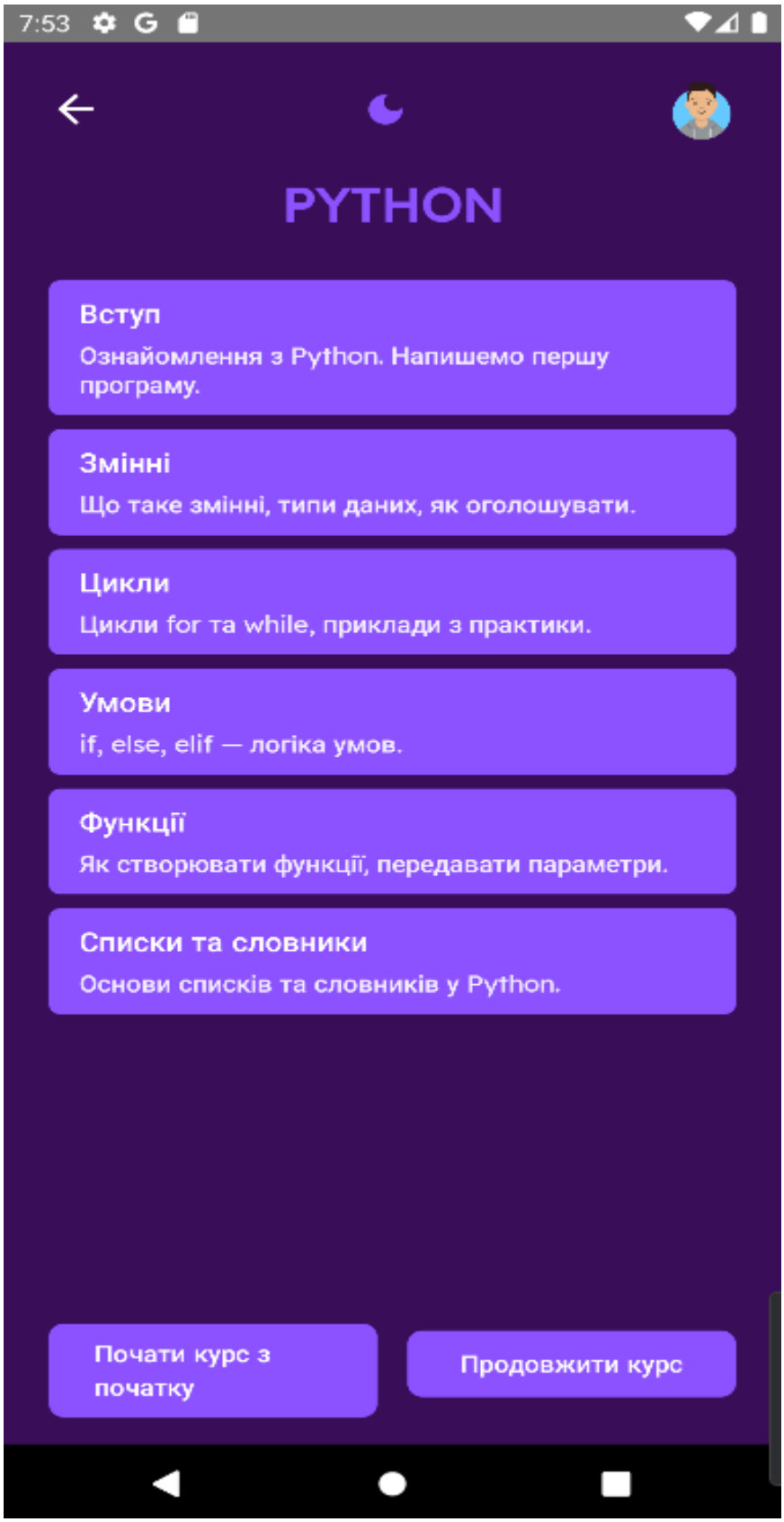


Рис. 4.7 Екран курсу Python

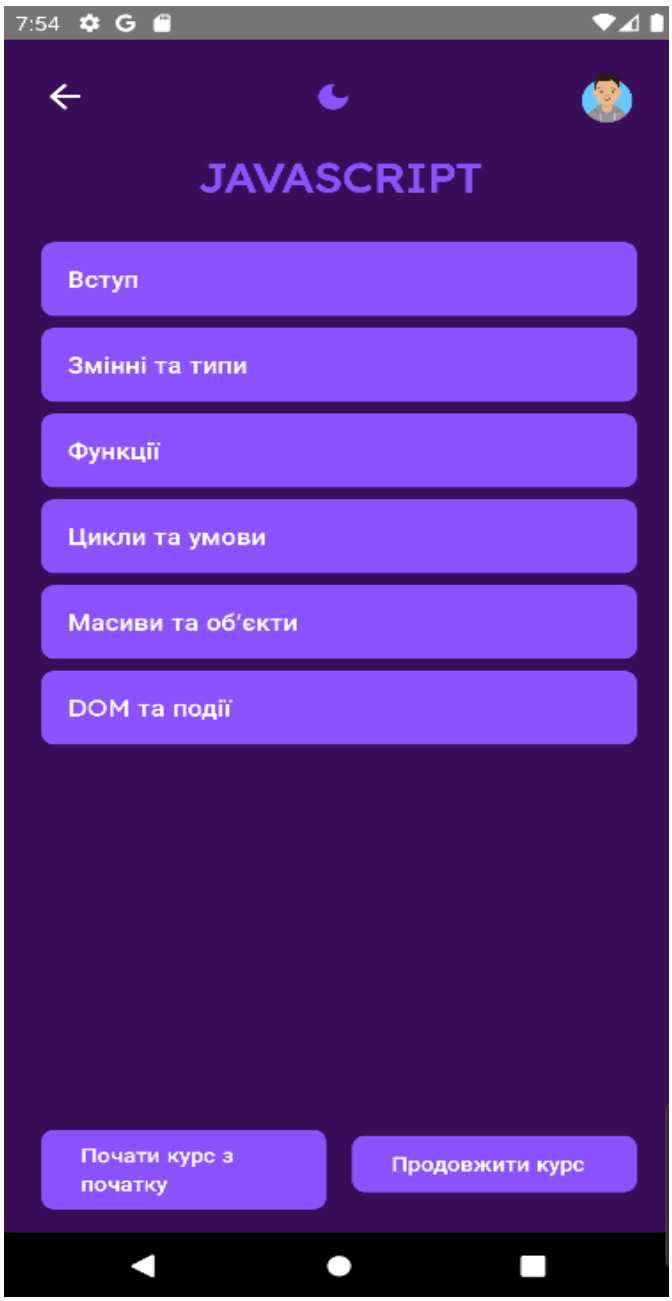


Рис. 4.8 Екран курсу JavaScript

4.10 Екран AI-помічника

При натисканні кнопки «AI-помічник» у меню курсів запускається AiAssistantActivity, яка через setContent відображає AiAssistantScreen(vm: AiAssistantViewModel) на рисунку 4.9. У верхній частині екрану - TopAppBar з заголовком “AI-помічник” та кнопкою «Назад», що викликає finish(). Основний простір займає зверстаний у LazyColumn чат: повідомлення від користувача

вирівнюються праворуч, а відповіді ШІ - ліворуч, кожне в Surface із тоном (primaryContainer для користувача, secondaryContainer для ШІ).

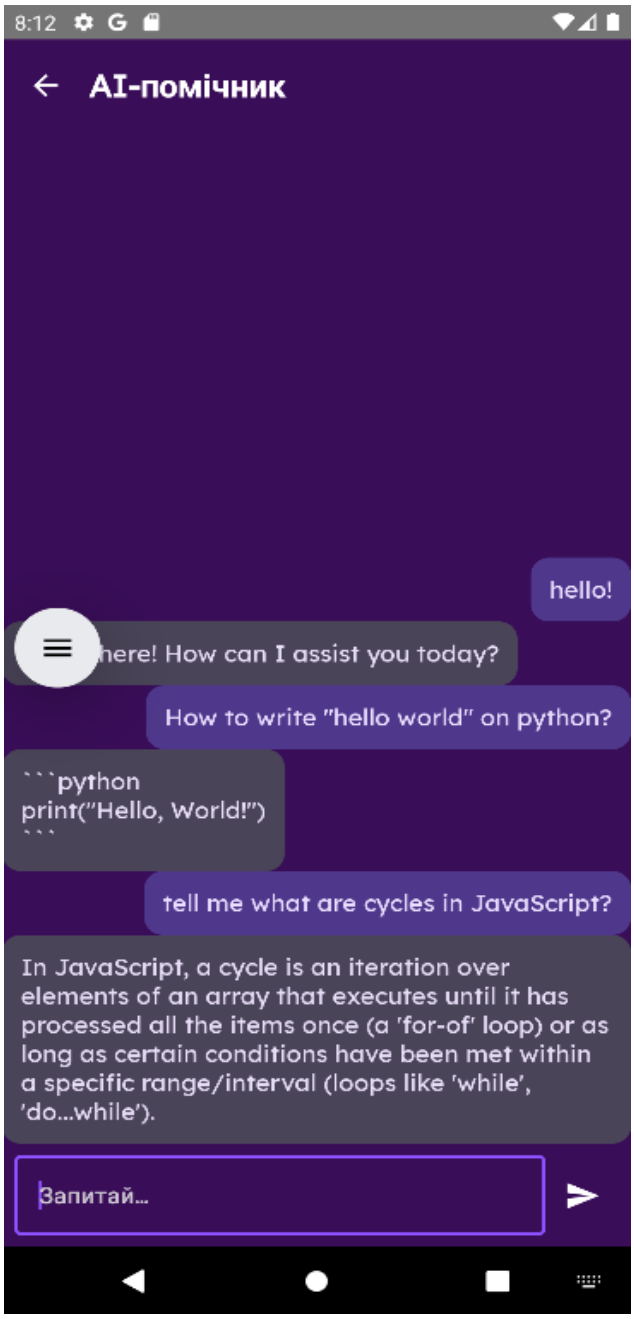


Рис. 4.9 Інтерфейс AI-помічника

Під списком повідомлень розташований рядок вводу: OutlinedTextField із плейсхолдером “Запитай…” і IconButton із іконкою відправки (Send). При натиску цієї кнопки, якщо рядок не порожній, викликається vm.send(input.trim()), після чого

поле очищується. Комунікація із бекендом ШІ відбувається через AiAssistantViewModel, що підтримує chat: StateFlow<List<ChatMessage>>.

Функція MessageBubble малює окремі бульбашки повідомлень: перевіряє роль (msg.role), вибирає колір фону й вирівнювання, а потім обгортає текст у Surface зі скругленою формою й відступами. Така архітектура робить інтерфейс читабельним, уніфікованим і легко розширюваним (наприклад, для підтримки зображень чи кнопок у повідомленнях).

4.11 Екран магазину

При відкритті ShopActivity з меню курсів за допомогою setContent завантажується LessonViewModel, який надає поточні значення кількості життів, підказок і внутрішньої валюти. У верхній частині екрана розміщено TopAppBar із заголовком «Магазин» та кнопкою «Назад», що завершує активність.

Нижче розташована горизонтальна панель балансу, де відображається число життів, кількість доступних підказок і загальна сума монет. Під нею наведено короткі пояснення значків: «життя» - блокують проходження тестів за відсутності лічильника, «підказки» - дозволяють отримати допомогу при складних запитаннях, «монети» - заробляються за правильні відповіді та використовуються для покупок у магазині, дивись рисунок 4.10 - Екран магазину: відображення балансу життів, підказок та монет, пояснювальні рядки та кнопки купівлі життя або підказки).

Далі наведено перелік товарів із кнопками для придбання додаткового життя або підказки за встановлену кількість монет. Кожна кнопка активується тільки за наявності достатньої кількості монет і при натисканні викликає методи vm.buyLife() або vm.buyHint(). Ці методи через GameManager ініціюють транзакцію у Firestore, оновлюють баланс монет і, у разі успішного завершення, збільшують лічильник життів або підказок.

Таке рішення з централізованим керуванням станом гейміфікаційних елементів у єдиному LessonViewModel забезпечує однорідний стиль інтерфейсу та спрощує його підтримку.

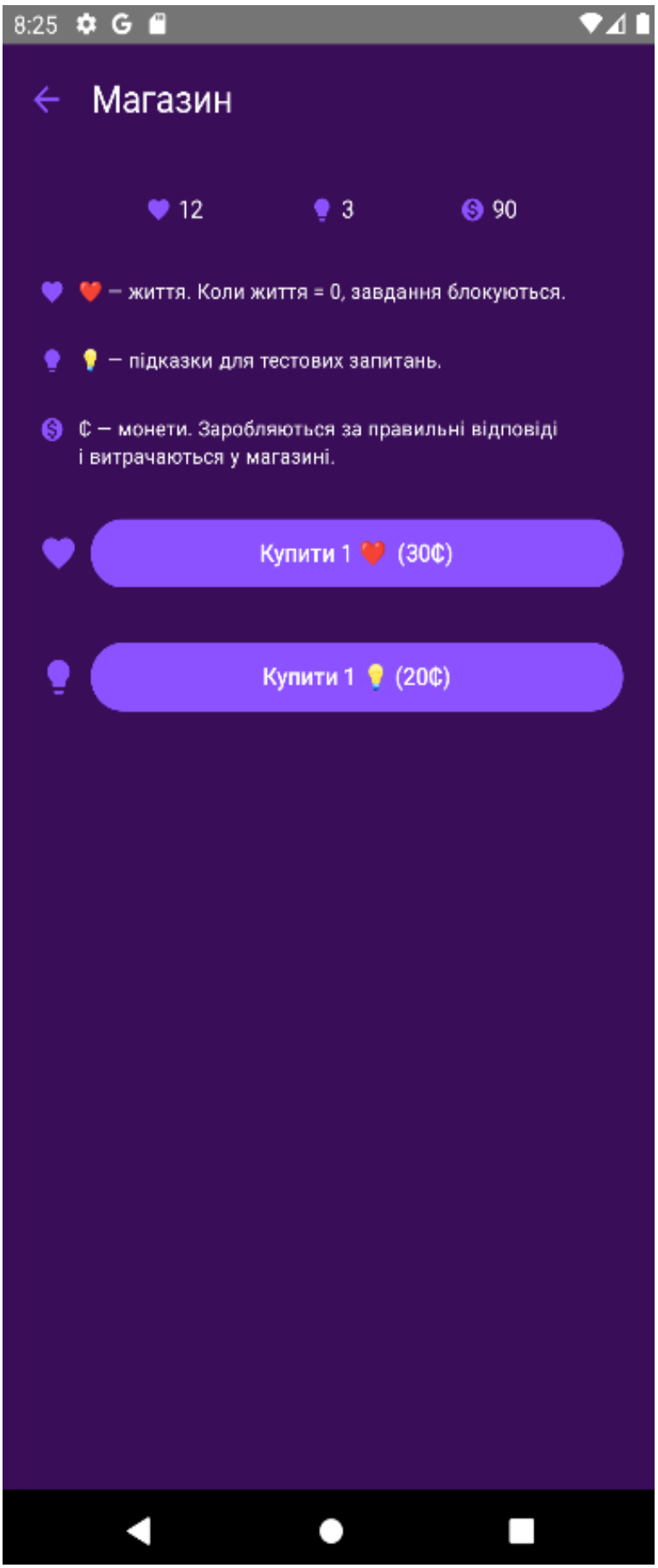


Рис. 4.10 Екран магазину: відображення балансу життів, підказок та монет, пояснювальні рядки та кнопки купівлі життя або підказки.

4.12 Екран уроку

Екран LessonActivity складається з однієї Activity, одного LessonViewModel і набору Composable-компонентів, причому вся логіка взаємодії користувача, збереження прогресу та елементів гейміфікації (життя, підказки, внутрішня валюта) інкапсулюється у LessonViewModel. На рисунках нижче продемонстровано три різні типи сторінок усередині модуля.

У рамках екрану уроку розрізняють три типи сторінок:

На рисунку 4.11 (Теоретична сторінка): відображається одноекранна картка з піктограмою, заголовком «Модуль: …» та текстом теоретичного матеріалу. Кнопка «Далі» завжди активна, оскільки перевірка на цьому етапі не потрібна.

На рисунку 4.12 (Тест із можливістю підказки): користувач бачить запитання з чотирма варіантами відповіді. Якщо в користувача залишилися підказки, активна кнопка «Показати підказку (n)»; після натискання лічильник підказок зменшується на одиницю, і внизу з’являється повідомлення зі змістом підказки (поле hint об’єкта Page.Test). Після вибору відповіді стає доступною кнопка «Перевірити», яка в разі правильної відповіді зараховує користувачеві бонусні одиниці валюти, а в разі помилки - зменшує лічильник життів та виводить відповідне сповіщення.

На рисунку 4.13 (Відкрите код-завдання): під теоретичною частиною або тестом розміщено багаторядкове текстове поле для введення коду та кнопку «Надіслати». Після відправлення код надходить на серверну функцію, яка перевіряє його з використанням прихованих тестів, повертає очікуваний результат (expectedCode) та рецензію ШІ (aiReview). У випадку невірного рішення під редактором відображається повідомлення з уточненим зразком коду та розгорнута рецензія штучного інтелекту; якщо завдання виконано правильно, користувач отримує визначену кількість внутрішньої валюти.

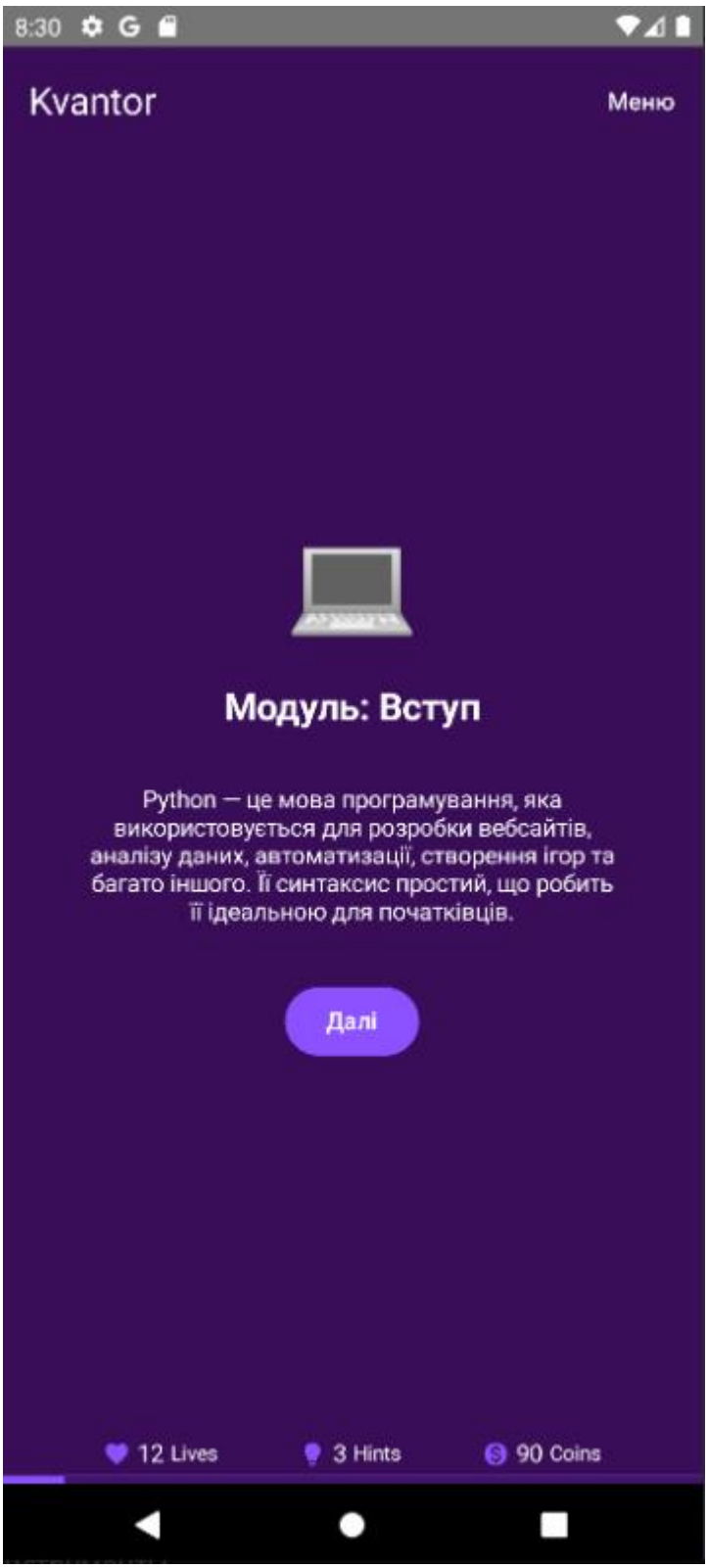


Рис. 4.11 сторінка Page.Theory

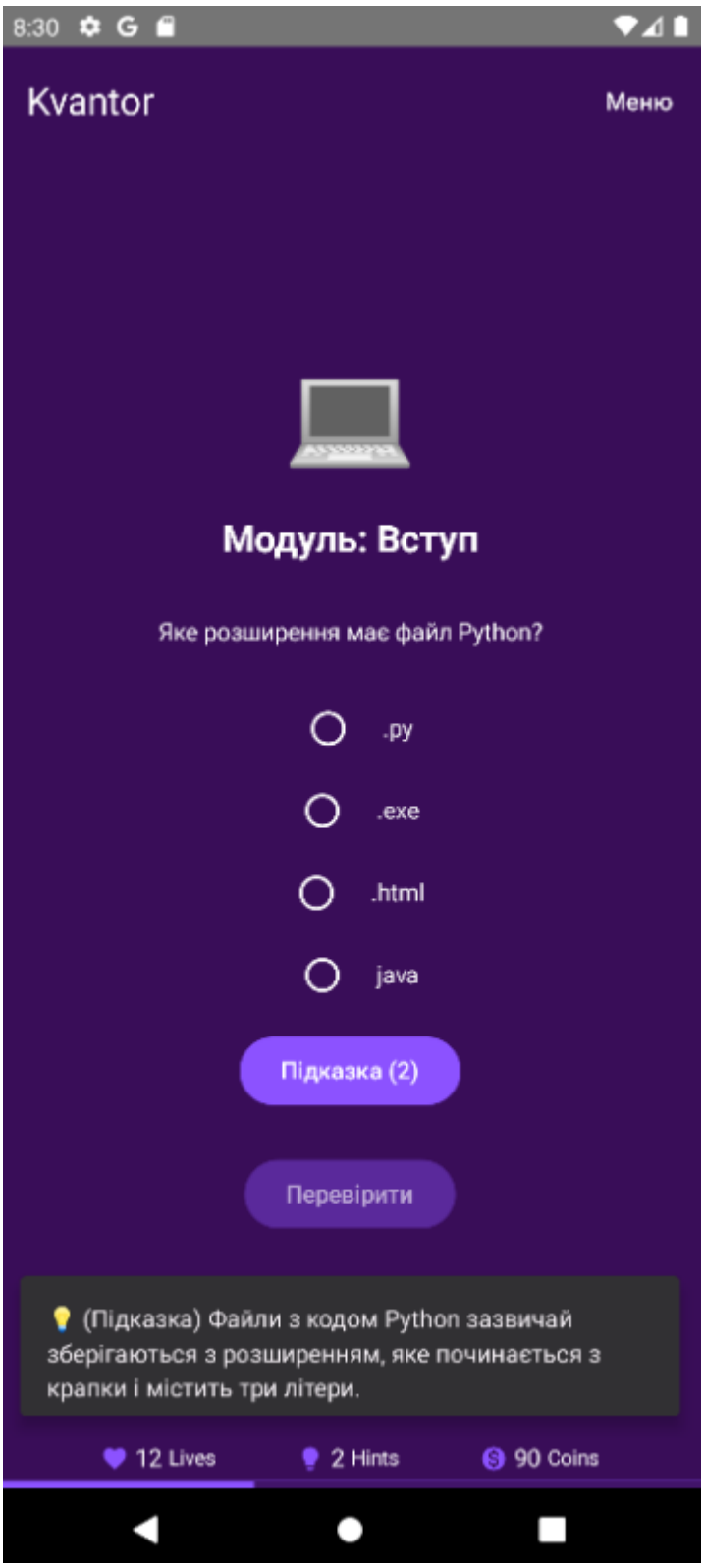


Рис. 4.12 сторінка Page.Test після використання підказки

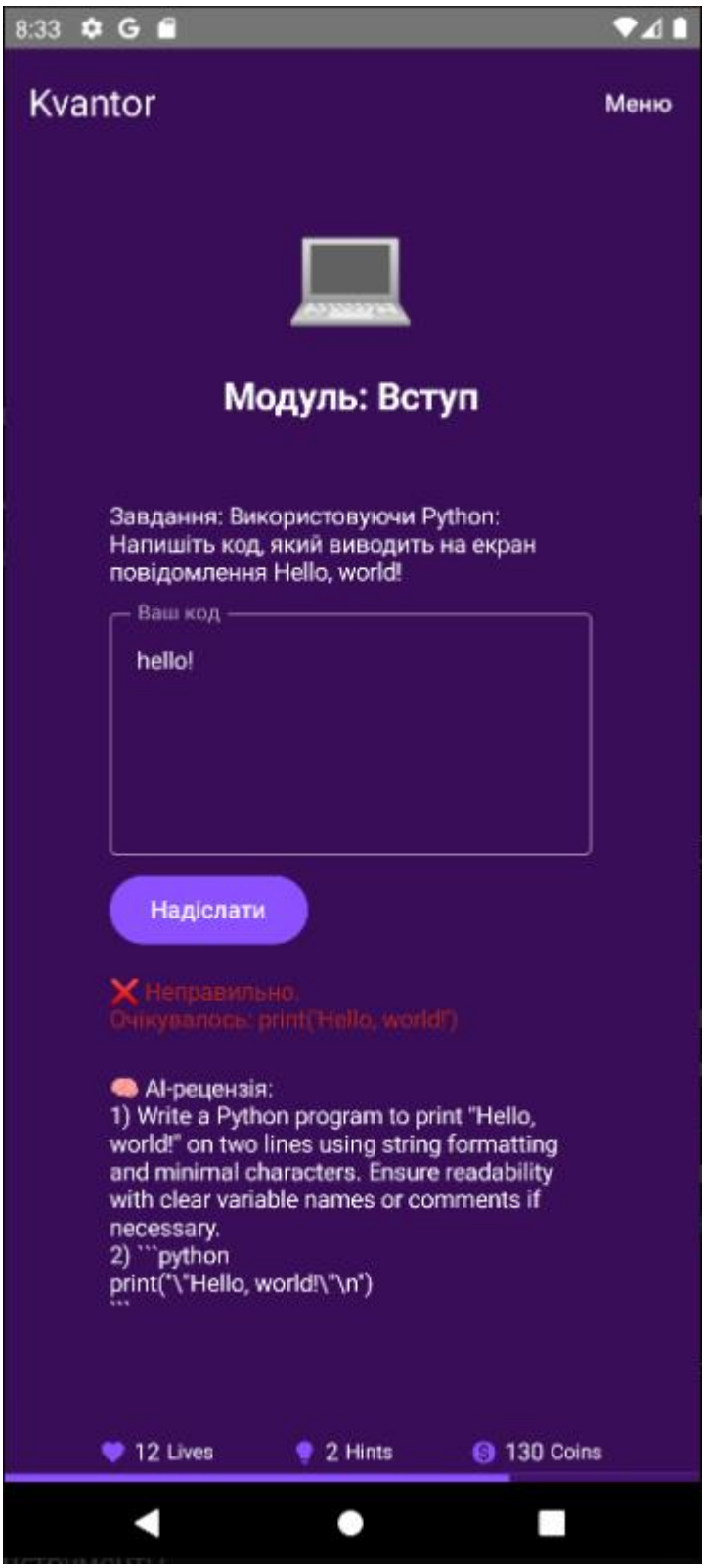


Рис. 4.13 сторінка Page.CodingTask із AI code-review

Після завантаження даних кожен документ Module містить масив pages, де для кожної сторінки зберігаються поля type (значення: "THEORY" | "TEST" | "TASK" | "FINAL") та payload (наприклад, текст теорії, список варіантів відповіді,

підказка тощо). Одразу після ініціалізації ViewModel викликається метод restoreProgress(), який встановлює внутрішні лічильники _currentModuleIndex і _currentPageIndex згідно з даними користувацького профілю.

Інтерфейс LessonScreen поділено на п’ять основних областей. По-перше, верхня панель (Top-bar) реалізована за допомогою TopAppBar із заголовком «Kvantor» та кнопкою «Меню». При натисканні на неї виконується viewModel.saveProgress() і відбувається повернення до вибору курсу. По-друге, PageContainer - це контейнер типу Box, який у залежності від поточного типу сторінки динамічно рендерить одну з трьох вкладених Composable - для теоретичного матеріалу, тесту або відкритого завдання. По-третє, горизонтальна панель статусу (Bottom-status-bar) складається з компонентів StatusChip, що відображають кількість життів, підказок і внутрішньої валюти; відлік часу до відновлення нового життя (timeToNextLife) показується окремо й оновлюється щосекунди через StateFlow у ViewModel. По-четверте, безпосередньо нижче розміщено індикатор прогресу (LinearProgressIndicator), де рівень заповнення обчислюється як

і стилізується акцентним кольором. І, нарешті, для відображення поточних повідомлень користувачу (наприклад, про відсутність ресурсів чи текст підказки) використовується SnackbarHost, керований локальним станом snackState.

Обробка подій від ViewModel (UiEvent) реалізована таким чином:

- При емісії UiEvent.NoLives (checkAnswer() або next() при відсутності життів) показується сповіщення «У вас закінчилися життя. Спробуйте пізніше».
- При UiEvent.NoHints (requestHint() без доступних підказок) відображається повідомлення «Підказки закінчилися».
- При UiEvent.NoCoins (у buyLife() чи buyHint() за браку валюти) з’являється повідомлення «Недостатньо коштів».
- Коли доступна підказка, ViewModel встановлює стан showHint; у LaunchedEffect(hint) UI відразу показує сповіщення з текстом підказки, після чого викликається clearHint().

У документі users/{uid} зберігаються поля прогресу (progress.{courseType}.moduleIndex, progress.{courseType}.pageIndex), які оновлюються методом saveProgress(); значення ресурсів (lives, hints, coins, lastLifeTS) змінюються через транзакції в GameManager; а підколекція achievements доповнюється новими відмітками за допомогою AchievementManager.unlockAchievement(), що виконується у Cloud Functions. Така організація даних забезпечує можливість офлайн-роботи: клієнт підтримує локальний кеш і автоматичний ретрай у разі помилок мережі, а всі лічильники синхронізуються через snapshotListener Firestore.

4.13 Тестування застосунку

У процесі розроблення дипломного застосунку «Kvantor» особливу увагу було приділено автоматизованому тестуванню, адже саме воно зменшує ризик регресій, пришвидшує цикл «код – перевірка – реліз» і дає впевненість, що ключові сценарії залишаються працездатними після будь-яких змін. Існує кілька рівнів перевірок: unit-тести фокусуються на логіці окремих функцій, UI (або widget)-тести перевіряють правильність розмітки одного екрана, інтеграційні тести охоплюють ланцюжок із кількох екранів і взаємодію з базою чи віддаленими сервісами, а end-to-end тести проганяють повний шлях користувача разом із бек-ендом.

Для роботи було обрано саме UI та інтеграційний рівні як оптимальний компроміс між швидкістю виконання та глибиною покриття, адже unit-тести не дають інформації про роботу інтерфейсу, а повні end-to-end перевірки занадто повільні і складні для CI. Загальний обсяг реалізованих тестів становить близько двадцяти сценаріїв, включно з трьома інтеграційними перевірками вибору курсу та одним UI-тестом для головного екрану, що забезпечує швидкість виконання одного циклу перевірок у середньому менше ніж п’ять секунд.

Така кількість і рівень деталізації тестів дозволяють утримувати високий рівень якості інтерфейсу та бізнес-логіки без надмірних витрат на їх підтримку, а також сприяють своєчасному виявленню дефектів ще на ранніх етапах розробки.

Крім того, автоматизовані тести були інтегровані в конвеєр безперервної інтеграції (CI), що забезпечило їх прогін при кожному коміті та запобігло накопиченню критичних помилок у коді. Використання емулятора Android 11 і команди ./gradlew connectedDebugAndroidTest дозволило запускати перевірки на виділених віртуальних пристроях, що спростило масштабування тестового середовища та підвищило стабільність результатів. Таким чином, ця стратегія, гарно показала себе при тустуванні і спрощує процес валідації нового функціоналу. Нижче наведені приклади коду для тестування нашого застосунку на рисунках (4.14 – 4.16).

```
18      /** PYTHON */
19      @Test fun pythonButton_opensPython() {
20          compose.onNodeWithTag("btn_python").performClick()
21
22          compose.waitUntil(TIMEOUT) {
23              compose.onAllNodes(
24                  hasTestTag("python_header") or hasText("Python"),
25                  useUnmergedTree = true
26              ).fetchSemanticsNodes().isNotEmpty()
27          }
28
29          compose.onNode(
30              hasTestTag("python_header") or hasText("Python"),
31              useUnmergedTree = true
32          ).assertExists()
33      }
34
35      /** JAVASCRIPT */
36      @Test fun jsButton_opensJavaScript() {
37          compose.onNodeWithTag("btn_js").performClick()
38
39          compose.waitUntil(TIMEOUT) {
40              compose.onAllNodes(
41                  hasTestTag("js_header") or hasText("JavaScript"),
42                  useUnmergedTree = true
43              ).fetchSemanticsNodes().isNotEmpty()
44          }
45
46          compose.onNode(
47              hasTestTag("js_header") or hasText("JavaScript"),
48              useUnmergedTree = true
49          ).assertExists()
50      }
```

Рис. 4.14 Перший фрагмент коду integration-тесту CourseSelectionNavigationTest

```
    /** AI-помічник */
    @Test fun aiButton_opensAiHelper() {
        compose.onNodeWithTag("btn_ai").performClick()

        compose.waitUntil(TIMEOUT) {
            compose.onAllNodes(
                hasTestTag("ai_header") or hasText("AI"),
                useUnmergedTree = true
            ).fetchSemanticsNodes().isNotEmpty()
        }

        compose.onNode(
            hasTestTag("ai_header") or hasText("AI"),
            useUnmergedTree = true
        ).assertExists()
    }
}
```

Рис. 4.15 Другий фрагмент коду integration-тесту CourseSelectionNavigationTest

```
@RunWith(AndroidJUnit4::class)
class HomeScreenElementsTest {

    @get:Rule
    val compose = createAndroidComposeRule<CourseSelectionActivity>()

    /** на головному екрані видно ВСІ базові елементи */
    @Test fun allCoreWidgetsAreVisible() {
        val tags = listOf(
            "toggle_theme",
            "btn_shop",
            "avatar",
            "btn_python",
            "btn_js",
            "btn_ai"
        )

        tags.forEach { tag ->
            compose.onNodeWithTag(tag, useUnmergedTree = true)
                .assertExists("Елемент із тегом <$tag> відсутній!")
                .assertIsDisplayed()
        }
    }
}
```

Рис. 4.16 Фрагмент коду widget-тесту HomeScreenElementsTest

Практична реалізація складається з двох файлів.

Перший, CourseSelectionNavigationTest, містить три методи тестування: pythonButton_opensPython, jsButton_opensJavaScript та aiButton_opensAiHelper. У кожному з них симулюється натискання відповідної кнопки - Python, JavaScript або AI-помічник, далі за допомогою waitUntil очікується асинхронне оновлення даних у Firebase Firestore, а потім перевіряється поява відповідного заголовка (“Python”, “JavaScript” або “AI”) на новому екрані.

Другий файл, HomeScreenElementsTest, перевіряє, що після запуску CourseSelectionActivity на головному екрані відображаються всі необхідні елементи інтерфейсу: заголовок, три кнопки курсів, перемикач теми та аватар. Обидва набори тестів реалізовано з використанням Jetpack Compose Testing API та виконуються командою. /gradlew connectedDebugAndroidTest на емуляторі Android 11.

Повний прогін, що охоплює один UI-кейс і три інтеграційні підкейси, триває близько чотирнадцяти секунд і завершується статусом BUILD SUCCESSFUL, що підтверджено на рисунках 4.17 - 4.18.

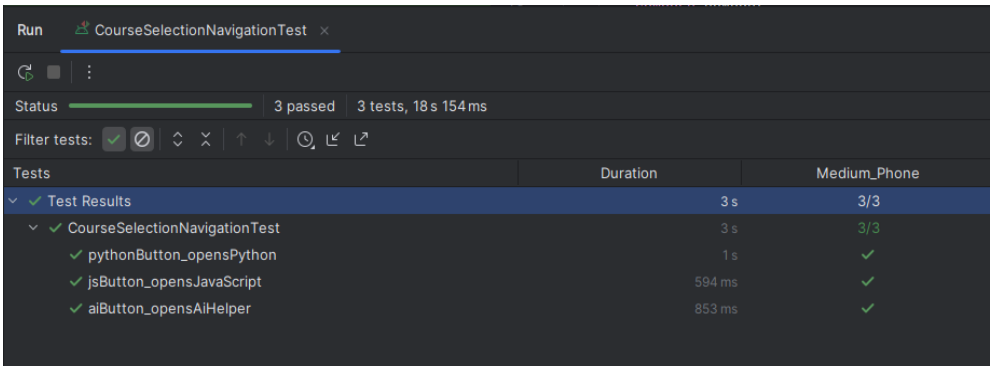


Рис. 4.17 Результати виконання CourseSelectionNavigationTest

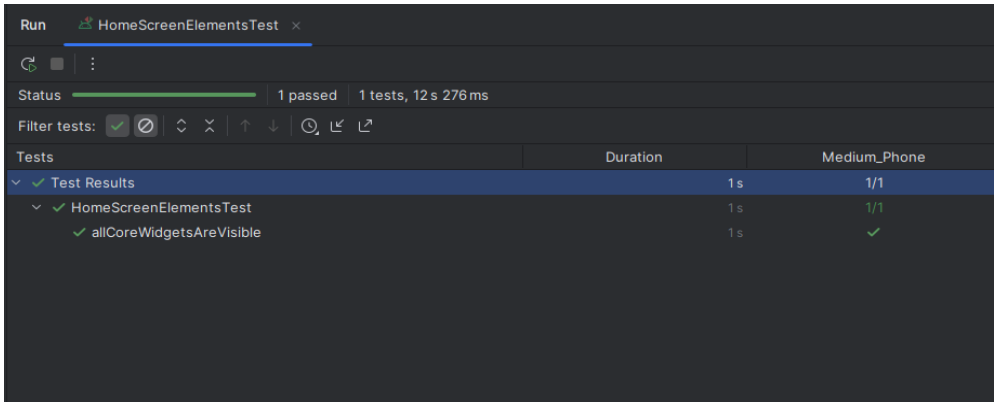


Рис. 4.18 Результати виконання HomeScreenElementsTest

Щоб продемонструвати місце наших перевірок у загальній «піраміді» тестування, на мою думку доцільно навести порівняльну характеристику різних рівнів на таблиці 4.1. Ця таблиця наочно ілюструє співвідношення швидкості виконання, глибини перевірок та простоти підтримки для кожного рівня тестування., глибиною перевірки та зручністю підтримки кожного типу тестів, а також підкреслює, як наші UI-тести та інтеграційні сценарії вписуються в загальну стратегію забезпечення якості.

Таблиця. 4.1

Порівняльна характеристика різних рівнів автоматизованого тестування застосунку

Рівень тестування	**Область перевірки**	**Середній час виконання**	**Сильні сторони**	**Обмеження**
Unit-тести	окремі функції, класи	< 100	максимальна швидкість, проста підтримка	не охоплюють UI та інтеграції
UI (widget)-тести HomeScreenElementsTest	верстка одного екрана	≈ 1 с	швидко ловлять злам розмітки	не перевіряють навігацію
Інтеграційні тести CourseSelectionNavigationTest	кілька екранів, робота з Firestore	≈ 1 с на сценарій	перевіряють реальний користувацький флоу	потребують моків сервісів
End-to-end тести	повний шлях через бек-енд	≥ 30 с	найвища впевненість	повільні, складні у CI

Таким чином, обраний набір перевірок забезпечує усвідомлений баланс: UI-тест дає блискавичний зворотний зв’язок щодо цілісності інтерфейсу, а інтеграційні тести гарантують коректність навігації та взаємодії з хмарною базою даних без суттєвих витрат часу; разом вони закривають критично важливі сценарії користувача і підтверджують готовність застосунку до подальших ітерацій розробки та розгортання.

ВИСНОВКИ

1. З’ясовано потребу в мобільному рішенні для ефективного вивчення мов із інтеграцією штучного інтелекту - як для теоретичних уроків, так і для автоматизованих тестів і code-review.

2. Проведено порівняльний аналіз таких сервісів, як Mimo, Grasshopper та SoloLearn; виявлено їхні обмеження (наприклад, жорсткі сценарії тестування без гнучких підказок), які стали нашими конкурентними перевагами.

3. Сформульовано набір функціональних вимог (реєстрація, вибір курсу, AI-асистент, гейміфікація тощо) та нефункціональних.

4. Проаналізовано доступні інструменти та бібліотеки, після чого зупинилися на Jetpack Compose для UI, Firebase для бекенду, а також власному AI-модулі для підтримки code-review.

5. Виконано розробку всіх екранів - від логіна і налаштування профілю до модульних уроків із теорією, тестами й відкритими завданнями - та забезпечено інтеграцію з Firestore, коректне збереження прогресу й гейміфікацію.

6. Проведено ручне і автоматизоване тестування на різних пристроях і конфігураціях: UI-тести з імітацією натискань, перевірка транзакцій у Firestore, функціональне тестування AI-асистента та навантажувальне тестування прогрес-бару й мережевих запитів.

7. Робота пройшла апробацію на наступних наукових конференціях:

Спиридонов В.С., Лаврененко В.В. Огляд сучасних технологій для розробки мобільних освітніх застосунків. // VI Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT». Збірник тез. 15 квітня 2025 року, ДУІКТ, м.Київ (подано до друку).

Спиридонов В.С., Лаврененко В.В. Актуальність використання гейміфікації у розробці мобільних додатків для вивчення програмування. // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». Збірник тез. 24 квітня 2025 року, ДУІКТ, м.Київ, С. 231-232.

ПЕРЕЛІК ПОСИЛАНЬ

1 Росмансях Ю. Особливості, фреймворки та переваги гейміфікованого мікронавчання / Y. Rosmansyah // Proceedings of the 3rd International Conference on Modern Educational Technology (ICMET ’21), 17–19 травня 2021 р., Джакарта, Індонезія. – New York: ACM, 2021. – С. 130–135. – DOI: 10.1145/3468978.3469000.

2. Mimo. Learn to code | Mimo: Python, JavaScript, HTML, CSS, and more. URL: https://mimo.org (дата звернення: 26.05.2025).

3. Sololearn: learn to code. SoloLearn: Programmieren lernen. URL: https://www.sololearn.com/en/ (дата звернення: 26.05.2025).

4. Gamification in learning apps: examples and features overview - riseapps. Riseapps: AI Software Solutions since 2016. URL: https://riseapps.co/gamification-in-learning-apps (дата звернення: 26.05.2025).

5. Examining the effectiveness of gamification as a tool promoting teaching and learning in educational settings: a meta-analysis [Електронний ресурс] // PLOS ONE. – 2023. – Режим доступу: https://pmc.ncbi.nlm.nih.gov/articles/PMC10591086/#:~:text=The%20integration%20of%20gamification%20into,The%20research (дата звернення: 18.05.2025).

6. Patience-Davies . Just where did the claim that stories are 22 times more memorable than facts alone come from?. LinkedIn: Log In or Sign Up. URL: https://www.linkedin.com/pulse/just-where-did-claim-stories-22-times-more-memorable-patience-davies-ud7le/ (date of access: 26.05.2025).

7. Рібіш М., Шмідт Д. Застосування Clean Architecture та MVVM для модульних Android-додатків / M. Riebisch, D. Schmidt // Proceedings of the International Conference on Mobile Software Engineering and Systems. – Cham: Springer, 2024. – С. 123–138. – DOI: 10.1007/978-3-031-78338-8_9.

8. Ollama Search. *Ollama*. URL: https://ollama.com/search (дата звернення: 26.05.2025).

9. Download Android Studio & App Tools - Android Developers. *Android Developers*. URL: https://developer.android.com/studio (дата звернення: 26.05.2025).

10. GitHub · Build and ship software on a single, collaborative platform. *GitHub*. URL: https://github.com (дата звернення: 26.05.2025).

11. Firebase | Google's Mobile and Web App Development Platform. *Firebase*. URL: https://firebase.google.com (дата звернення: 26.05.2025).

12. Jetpack Compose UI App Development Toolkit - Android Developers. Android Developers. URL: https://developer.android.com/jetpack/compose (дата звернення: 26.05.2025).

13. Gjerde M., Sørensen C., Nordby K. Оцінка архітектури MVVM в Android-додатках: переваги та компроміси / M. Gjerde, C. Sørensen, K. Nordby // ACM Transactions on Software Engineering and Methodology. – 2020. – Vol. 29, No. 4. – Art. 30. – DOI: 10.1145/3402102.

14. Butler M., Lee S., Park J. Автоматизоване рев'ю коду з глибинним навчанням: огляд та напрямки розвитку / M. Butler, S. Lee, J. Park // IEEE Transactions on Software Engineering. – 2022. – Vol. 48, No. 3. – P. 865–892. – DOI: 10.1109/TSE.2021.3068507.

15. Patadia M. Що таке декларативний UI в Jetpack Compose? [Електронний ресурс] / M. Patadia // Medium. – 2021 May 15 [цит. 2025 May 27]. – Available from: https://meetpatadia9.medium.com/what-is-declarative-ui-in-jetpack-compose-e7a64b0616d5.

16. Chen M., Lee T. Зручність мобільних шаблонів аутентифікації: емпіричне дослідження / M. Chen, T. Lee // Proceedings of the 2018 ACM on Interactive Surfaces and Spaces (ISS '18). – 2018. – P. 45–54. – DOI: 10.1145/3279779.3279791.

17. Горєльов В. О. Застосування Android Studio для розробки мобільних застосунків [Електронний ресурс] / V. O. Горєльов // Сайт кафедри комп'ютерних наук IT-Центру. – 2017. – Режим доступу: https://itcm.comp-sc.if.ua/2017/Goryelov.pdf (дата звернення: 27.05.2025).

18. Esmaeel H. Застосування інструментів Android Studio SDK [Електронний ресурс] / H. Esmaeel // ResearchGate. – 2019. – Режим доступу: https://www.researchgate.net/profile/Hana-Esmaeel/publication/331673953_Apply_Android_Studio_SDK_Tools/links/5db4b1594

585155e27074ed0/Apply-Android-Studio-SDK-Tools.pdf (дата звернення: 27.05.2025).

19. Ivanov A., Petrenko B. Аналіз продуктивності Android-додатків на Kotlin / A. Ivanov, B. Petrenko // Proc. of the 2018 IEEE 15th Int. Conf. on Software Testing, Verification and Validation Workshops (ICSTW 2018). – 2018. – P. 112–119. – DOI: 10.1109/ICSTW.2018.8353979.

20. Guspiel M. Архітектурні патерни в розробці Android: MVVM і Clean Architecture [Магістерська робота] / M. Guspiel // Metropolia University of Applied Sciences. – 2019. – Режим доступу: https://www.theseus.fi/bitstream/handle/10024/852201/Guspiel_Michal.pdf?sequence=2&isAllowed=y (дата звернення: 27.05.2025).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка мобільного застосунку для вивчення програмування з елементами ігрофікації

Виконав студент 4 курсу
групи ПД-44
Спиридонов Владислав Сергійович
Керівник роботи
викладач кафедри ІПЗ Лавреченко Віктор Вікторович

Київ – 2025

1

Мета, об’єкт та предмет дослідження

Мета роботи: оптимізація процесу вивчення програмування за рахунок використання принципів гейміфікації та методів штучного інтелекту, розробка мобільного застосунку для покращення засвоєння матеріалу та підвищення мотивації користувачів.

Об’єкт дослідження: процес навчання програмуванню.

Предмет дослідження: програмне забезпечення для вивчення мов програмування.

2

Задачі кваліфікаційної роботи

1. Провести аналіз мобільних застосунків для вивчення програмування.
2. Оцінити наявні альтернативи.
3. Сформулювати функціональні та нефункціональні характеристики програмного забезпечення.
4. Дослідити та вибрати інструменти для розробки.
5. Програмно реалізувати застосунок.

3

Аналіз існуючих рішень

Показник	SoloLearn	Mimo	Codecademy	**Kvantor**
Платформа	Веб, мобільний	Мобільний	Веб, мобільний	Мобільний
AI-асистент	-	-	-	+
Автоматичне рев'ю коду	-	-	-	+
Система досягнень і мотивації	+	-	+	+
Інтерактивні код-завдання	+	+	+	+
Навчання через практику (code first)	-	-	+	+
Підтримка кількох мов програмування	+	+	+	+

4

Вимоги

Функціональні вимоги:

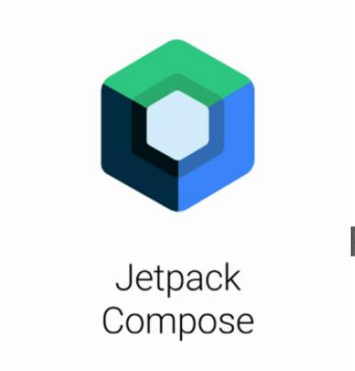
1. Реєстрація та автентифікація користувачів з можливістю збереження прогресу.
2. Вибір мови програмування та складності навчальних модулів.
3. Інтерактивне проходження теоретичних та практичних завдань у форматі квестів.
4. Використання AI-код-рев'ю для перевірки написаного коду та надання персоналізованих рекомендацій.
5. Система досягнень за успішне проходження завдань.

Нефункціональні вимоги:

1. Підтримка всіх версій Android починаючи з Oreo (API 26).
2. Інтуїтивно зрозумілий та адаптивний інтерфейс користувача.
3. Інтеграція з Firebase Analytics для збору метрик активації уроків та помилок AI-рев'ю.
4. Забезпечення захисту персональних даних та безпечна автентифікація через Firebase Authentication.
5. Прогрес і кеш оновлюються з сервером не рідше ніж раз на 1 хвилину.

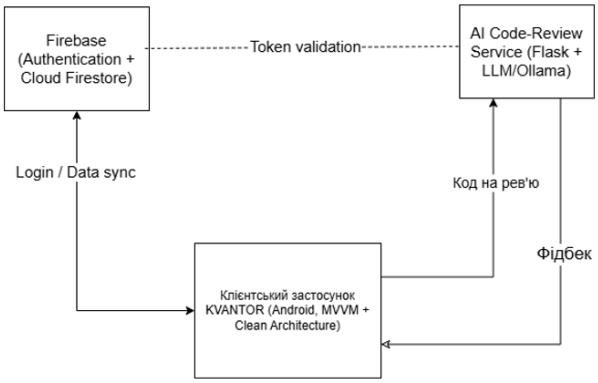
5

Програмні засоби реалізації



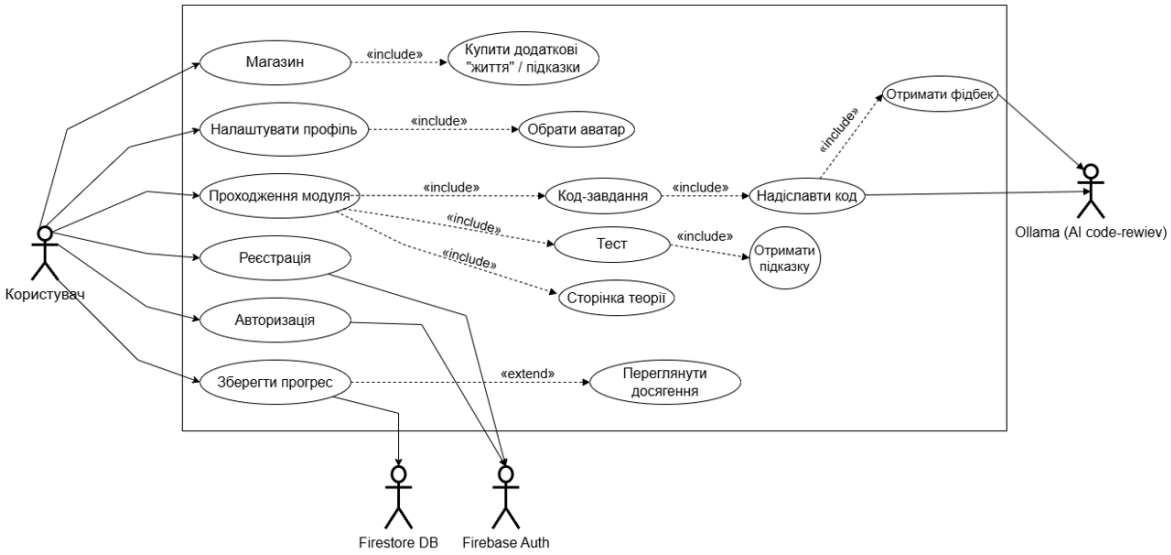
6

Опис програми для вивчення програмування



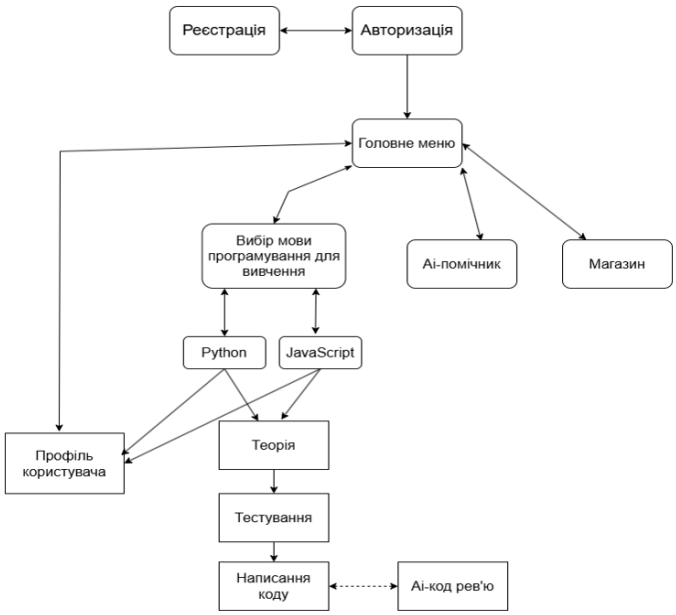
7

Діаграма варіантів використання



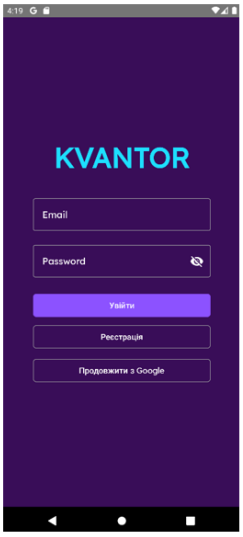
8

Структурна схема екранів мобільного застосунку

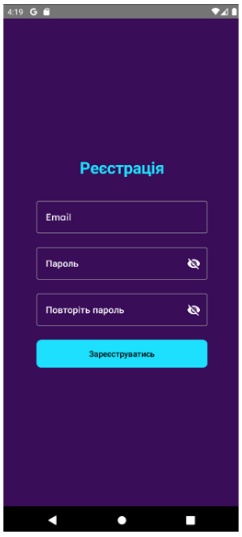


9

Екранні форми

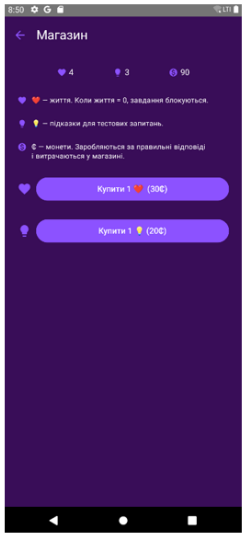


Екран авторизації



Екран реєстрації

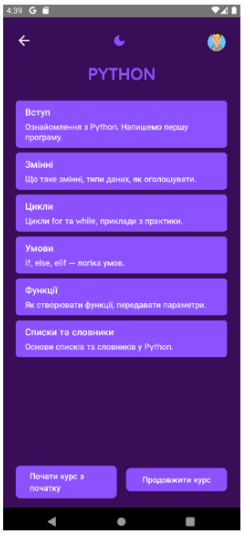
Головне меню



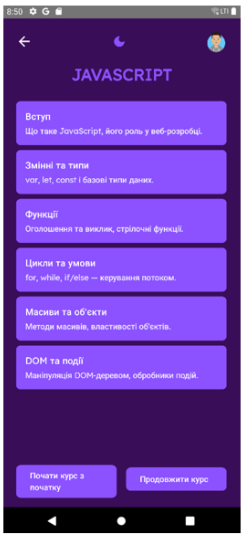
Магазин

10

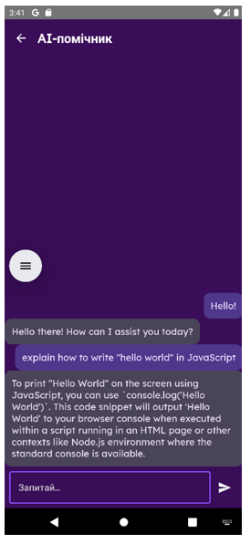
Екранні форми



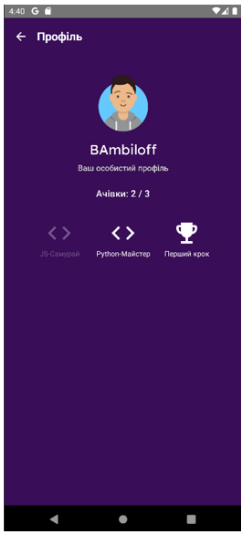
Модуль Python



Модуль JavaScript



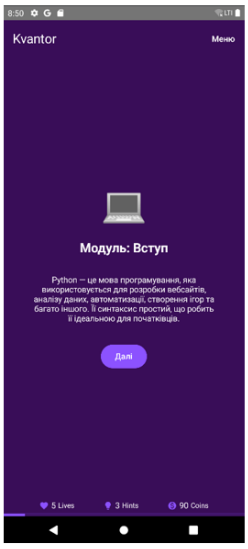
Ai-помічник
(чат бот)



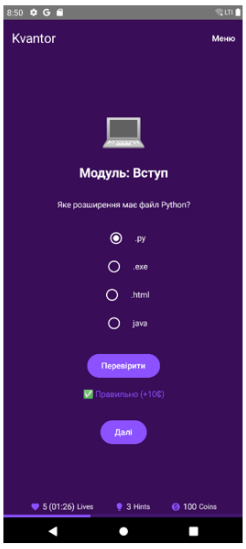
Особистий профіль

11

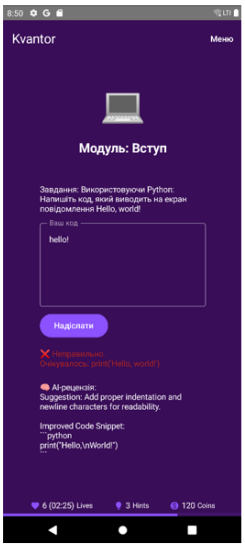
Екранні форми



Екран теорії



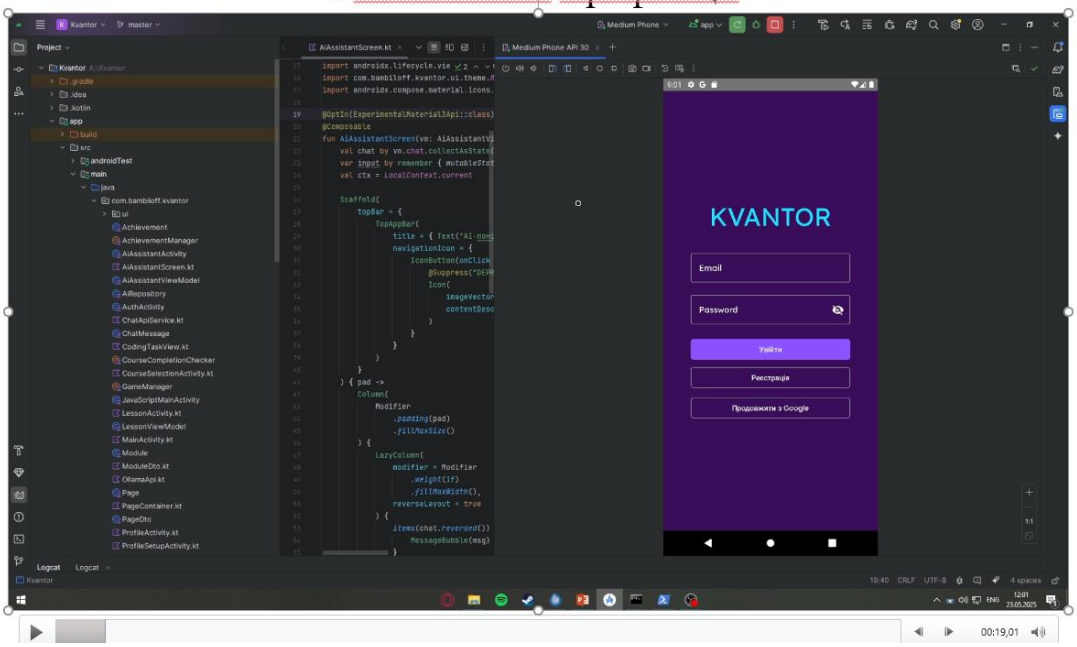
Екран теоретичних питань



Екран відкритих питань
та Ai код-рев'ю

12

Демонстрація роботи програми для вивчення програмування з елементами ігрофікації



13

Апробація результатів дослідження

1. Спиридонов В.С., Лаврененко В.В. Огляд сучасних технологій для розробки мобільних освітніх застосунків. // VI Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT». Збірник тез. 15 квітня 2025 року, ДУІКТ, м.Київ (подано до друку).

2. Спиридонов В.С., Лаврененко В.В. Актуальність використання гейміфікації у розробці мобільних додатків для вивчення програмування. // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». Збірник тез. 24 квітня 2025 року, ДУІКТ, м.Київ, сторінка 231-232.

14

Висновки

1. Підтвердив високу потребу в мобільних інструментах з ігрофікацією та миттєвим фідбеком для навчання програмуванню.
2. Виявив обмеження існуючих рішень, що визначило конкурентні переваги kvantor.
3. Сформулював компактний перелік функціональних і нефункціональних вимог, який ліг в основу архітектури й тестування.
4. Підібрано технологічний стек з урахуванням продуктивності, сумісності зі штучним інтелектом і зручності підтримки.
5. Реалізовав застосунок.
6. Проведено комплексне тестування

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

Вихідний код файлу CourseSelectionActivity.kt

```
package com.bambiloff.kvantor

import android.annotation.SuppressLint
import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.core.animateFloatAsState
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import
androidx.compose.foundation.interaction.MutableInteractionSo
urce
import
androidx.compose.foundation.interaction.collectIsPressedAsSta
te
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.rememberScrollState
import
androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.graphicsLayer
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.testTag
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.unit.dp
import com.bambiloff.kvantor.ui.theme.KvantorTheme
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.firestore.FirebaseFirestore
import com. google. firebase. firestore. BuildConfig

class CourseSelectionActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            var isDarkTheme by remember { mutableStateOf(true)
}

            KvantorTheme(darkTheme = isDarkTheme) {
                CourseSelectionScreen(
                    onSelect      = ::openCourse,
                    isDarkTheme   = isDarkTheme,
                    onToggleTheme = { isDarkTheme = it }
                )
            }
        }
    }


    private fun openCourse(courseId: String) {


        if (BuildConfig.DEBUG) {
            launchTarget(courseId)
            return
        }


        val uid = FirebaseAuth.getInstance().currentUser?.uid ?:
return
        FirebaseFirestore.getInstance()
            .collection("users")
            .document(uid)
            .update("selectedCourse", courseId)
            .addOnSuccessListener { launchTarget(courseId) }
    }


    private fun launchTarget(courseId: String) {
        val target = when (courseId) {
            "python"     -> MainActivity::class.java
            "javascript" -> JavaScriptMainActivity::class.java
            else         -> MainActivity::class.java
        }
        startActivity(Intent(this, target))

        if (!BuildConfig.DEBUG) {
            finish()
        }
    }
}

@SuppressLint("DiscouragedApi")
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun CourseSelectionScreen(
    onSelect: (String) -> Unit,
    isDarkTheme: Boolean,
    onToggleTheme: (Boolean) -> Unit
) {
    val cs  = MaterialTheme.colorScheme
    val ctx = LocalContext.current


    var avatarResId by remember {
mutableStateOf(R.drawable.default_avatar) }

LaunchedEffect(FirebaseAuth.getInstance().currentUser?.uid)
{
        val uid = FirebaseAuth.getInstance().currentUser?.uid ?:
return@LaunchedEffect
        FirebaseFirestore.getInstance()
            .collection("users")
            .document(uid)
            .get()
            .addOnSuccessListener { doc ->
                val name = doc.getString("avatarName") ?:
"default_avatar"
                val id   = ctx.resources.getIdentifier(name,
"drawable", ctx.packageName)
                avatarResId = if (id != 0) id else
R.drawable.default_avatar
            }
    }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .background(cs.background)
```

```
            .verticalScroll(rememberScrollState())
            .padding(24.dp),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {

        Row(
            Modifier
                .fillMaxWidth()
                .padding(bottom = 16.dp),
            horizontalArrangement = Arrangement.SpaceBetween,
            verticalAlignment     = Alignment.CenterVertically
        ) {
            IconToggleButton(
                modifier = Modifier.testTag("toggle_theme"),
                checked         = isDarkTheme,
                onCheckedChange = onToggleTheme
            ) {
                Icon(
                    imageVector       = if (isDarkTheme)
Icons.Default.DarkMode else Icons.Default.LightMode,
                    contentDescription = "Перемкнути тему",
                    tint              = cs.primary
                )
            }

            Row(verticalAlignment = Alignment.CenterVertically)
{
                IconButton(
                    modifier = Modifier.testTag("btn_shop"),
                    onClick = {
                        ctx.startActivity(Intent(ctx,
ShopActivity::class.java))
                    }
                ) {
                    Icon(
                        Icons.Default.ShoppingCart,
                        contentDescription = "Магазин",
                        tint               = cs.primary
                    )
                }

                Spacer(Modifier.width(8.dp))

                Image(
                    painter           = painterResource(id =
avatarResId),
                    contentDescription = "Профіль",
                    modifier          = Modifier
                        .size(36.dp)
                        .clip(RoundedCornerShape(8.dp))
                        .testTag("avatar")
                        .clickable {
                            ctx.startActivity(Intent(ctx,
ProfileActivity::class.java))
                        }
                )
            }
        }


        Icon(
            imageVector       = Icons.Default.School,
            contentDescription = null,
            tint              = cs.primary,
            modifier          = Modifier.size(96.dp)
        )
        Spacer(Modifier.height(16.dp))
        Text(
            text  = "Обери курс",
            style = MaterialTheme.typography.headlineMedium,
            color = cs.onBackground
        )
        Spacer(Modifier.height(32.dp))


        CourseButton(
            title       = "Python",
            description = "Базові концепції та практичні задачі",
            onClick     = { onSelect("python") },
            cs          = cs,
            tag         = "btn_python"
        )
        Spacer(Modifier.height(8.dp))
        CourseButton(
            title       = "JavaScript",
            description = "Основи веб-розробки та DOM-
маніпуляції",
            onClick     = { onSelect("javascript") },
            cs          = cs,
            tag         = "btn_js"
        )
        Spacer(Modifier.height(8.dp))
        CourseButton(
            title       = "AI-помічник",
            description = "Чат із ШІ та code-review",
            onClick     = {
                ctx.startActivity(Intent(ctx,
AiAssistantActivity::class.java))
            },
            cs          = cs,
            tag         = "btn_ai"
        )
    }
}


@Suppress("SameParameterValue")
@Composable
private fun CourseButton(
    title: String,
    description: String,
    onClick: () -> Unit,
    cs: ColorScheme,
    tag: String
) {
    val interaction = remember { MutableInteractionSource() }
    val pressed by interaction.collectIsPressedAsState()
    val scale  by animateFloatAsState(if (pressed) 0.95f else 1f)

    Button(
        onClick           = onClick,
        interactionSource = interaction,
        modifier          = Modifier
            .fillMaxWidth()
            .testTag(tag)
            .graphicsLayer { scaleX = scale; scaleY = scale }
            .padding(vertical = 4.dp),
        shape  = RoundedCornerShape(8.dp),
        colors = ButtonDefaults.buttonColors(
            containerColor = cs.primary,
            contentColor   = cs.onPrimary
        )
    ) {
        Column(horizontalAlignment =
Alignment.CenterHorizontally) {
            Text(title, style =
MaterialTheme.typography.titleMedium)
            Spacer(Modifier.height(4.dp))
            Text(
                text  = description,
```

```
            style = MaterialTheme.typography.bodySmall,
            color = cs.onPrimary.copy(alpha = .9f)
        )
```

Вміст файлу JavaScriptMainActivity.kt

```
package com.bambiloff.kvantor

import android.app.Activity
import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.OnBackPressedCallback
import androidx.activity.compose.setContent
import androidx.compose.animation.AnimatedVisibility
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.rememberScrollState
import
androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.DarkMode
import androidx.compose.material.icons.filled.LightMode
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.bambiloff.kvantor.ui.theme.KvantorTheme
import com.bambiloff.kvantor.ui.theme.Rubik
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.firestore.FirebaseFirestore
import androidx.compose.ui.text.style.TextAlign
import androidx. compose. ui. platform. testTag


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


        onBackPressedDispatcher.addCallback(
            this,
            object : OnBackPressedCallback(true) {
                override fun handleOnBackPressed() {
                    startActivity(
                        Intent(this@JavaScriptMainActivity,
CourseSelectionActivity::class.java)

.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)
                    )
                    finish()
                }
            }
        )

        setContent {
            var dark by remember { mutableStateOf(true) }
            KvantorTheme(darkTheme = dark) {
JavaScriptMenu(dark, onToggle = { dark = it }) }
                }
            }
        }
    }
}


@Composable
fun JavaScriptScreen() {
    Text(
        "Курс JavaScript",
        modifier = Modifier.testTag("js_header")
    )
}


@Composable
private fun JavaScriptMenu(dark: Boolean, onToggle:
(Boolean) -> Unit) {

    /* базові кольори (повністю як у Python-екрані) */
    val bg      = if (dark) KvBg else Color(0xFFF5F5F5)
    val accent  = KvAccent
    val textClr = if (dark) KvTextColor else
Color(0xFF1A1A1A)

    val context = LocalContext.current
    val uid  = FirebaseAuth.getInstance().currentUser?.uid
    val db   = FirebaseFirestore.getInstance()


    var avatar by remember {
mutableStateOf(R.drawable.default_avatar) }
    LaunchedEffect(uid) {
        uid?.let { u ->
            db.collection("users").document(u).get()
                .addOnSuccessListener { d ->
                    avatar = when (d.getString("avatarName")) {
                        "avatar1" -> R.drawable.avatar1
                        "avatar2" -> R.drawable.avatar2
                        "avatar3" -> R.drawable.avatar3
                        "avatar4" -> R.drawable.avatar4
                        else      -> R.drawable.default_avatar
                    }
                }
        }
    }


    val topics = listOf(
        "Вступ"           to "Що таке JavaScript, його роль
у веб-розробці.",
        "Змінні та типи"    to "var, let, const і базові типи
даних.",
        "Функції"         to "Оголошення та виклик, стрілочні
функції.",
        "Цикли та умови"    to "for, while, if/else — керування
потоком.",
        "Масиви та об'єкти" to "Методи масивів, властивості
об'єктів.",
        "DOM та події"      to "Маніпуляція DOM-деревом,
обробники подій."
    )


    Box(Modifier.fillMaxSize().background(bg)) {
        Column(
            Modifier
```

```
            .fillMaxSize()
            .padding(horizontal = 24.dp, vertical = 16.dp)
            .verticalScroll(rememberScrollState()),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {

        Row(
            Modifier.fillMaxWidth().padding(bottom = 16.dp),
            horizontalArrangement =
Arrangement.SpaceBetween,
            verticalAlignment     = Alignment.CenterVertically
        ) {
            Icon(
                painter =
painterResource(R.drawable.ic_arrow_back),
                contentDescription = null,
                tint = textClr,
                modifier = Modifier.size(28.dp).clickable {
                    context.startActivity(
                        Intent(context,
CourseSelectionActivity::class.java)

.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)
                    )
                    (context as Activity).finish()
                }
            )

            IconToggleButton(dark, onToggle) {
                Icon(
                    if (dark) Icons.Default.DarkMode else
Icons.Default.LightMode,
                    null,
                    tint = accent
                )
            }

            Image(
                painter = painterResource(id = avatar),
                contentDescription = null,
                modifier = Modifier
                    .size(36.dp)
                    .clip(RoundedCornerShape(8.dp))
                    .clickable {
                        context.startActivity(Intent(context,
ProfileActivity::class.java))
                    }
            )
        }


        Text(
            "JAVASCRIPT",
            fontSize   = 28.sp,
            fontWeight = FontWeight.Bold,
            fontFamily = Rubik,
            color      = accent
        )

        Spacer(Modifier.height(24.dp))


        topics.forEach { (title, descr) ->
            var expand by remember { mutableStateOf(false) }
            Column(
                Modifier.fillMaxWidth()
                    .padding(vertical = 4.dp)
                    .background(accent,
RoundedCornerShape(8.dp))
                    .clickable { expand = !expand }
                    .padding(16.dp)
            ) {
                Text(title, color = Color.White, fontSize = 16.sp,
fontFamily = Rubik)
                AnimatedVisibility(expand) {
                    Text(
                        descr,
                        color = Color.White.copy(alpha = .9f),
                        fontSize = 14.sp,
                        fontFamily = Rubik,
                        modifier = Modifier.padding(top = 6.dp)
                    )
                }
            }
        }

        Spacer(Modifier.weight(1f))


        Row(
            Modifier.fillMaxWidth(),
            horizontalArrangement =
Arrangement.spacedBy(16.dp)
        ) {
            BasicPurpleButton(
                text = "Почати курс з початку",
                modifier = Modifier.weight(1f)
            ) {
                uid?.let { user ->
                    db.collection("users").document(user)
                        .update(
                            "progress.javascript",
                            mapOf("moduleIndex" to 0, "pageIndex" to
0)
                        )
                }
                context.startActivity(
                    Intent(context, LessonActivity::class.java)
                        .putExtra("courseType", "javascript")
                )
            }

            BasicPurpleButton(
                text = "Продовжити курс",
                modifier = Modifier.weight(1f)
            ) {
                context.startActivity(
                    Intent(context, LessonActivity::class.java)
                        .putExtra("courseType", "javascript")
                )
            }
        }
    }
}


@Composable
private fun BasicPurpleButton(
    text: String,
    modifier: Modifier = Modifier,
    onClick: () -> Unit
) {
    Button(
        onClick = onClick,

            modifier = modifier,
        shape    = RoundedCornerShape(8.dp),
        colors   = ButtonDefaults.buttonColors(
            containerColor = KvAccent,
```

```
            contentColor   = Color.White
        )
    ) {
        Text(
            text,
            fontFamily = Rubik,
```

Вміст файлу LessonActivity.kt

```
            package com.bambiloff.kvantor

import android.annotation.SuppressLint
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.AnimatedVisibility
import androidx.compose.animation.fadeIn
import
androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Code
import androidx.compose.material.icons.filled.Favorite
import androidx.compose.material.icons.filled.Lightbulb
import androidx.compose.material.icons.filled.MonetizationOn
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider
import androidx.lifecycle.viewmodel.compose.viewModel
import com.bambiloff.kvantor.ui.*
import com.google.firebase.auth.FirebaseAuth
import kotlinx.coroutines.flow.collectLatest


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

        val courseType = intent.getStringExtra("courseType") ?:
"python"
        val uid        = FirebaseAuth.getInstance().currentUser?.uid
?: return

        val factory = object : ViewModelProvider.Factory {
            @Suppress("UNCHECKED_CAST")
            override fun <T : ViewModel> create(modelClass:
Class<T>): T =
                LessonViewModel(courseType).apply {
loadModules() } as T
        }

        setContent {
            val vm: LessonViewModel = viewModel(factory =
factory)
            LessonScreen(
                viewModel    = vm,
                courseType   = courseType,
                uid          = uid,
                onBackToMenu = { finish() }
            )
            modifier = Modifier.fillMaxWidth(),
            textAlign = TextAlign.Left
        )
    }
}
        }
    }
}


@SuppressLint("DefaultLocale")
@OptIn(ExperimentalMaterial3Api::class,
ExperimentalFoundationApi::class)
@Composable
fun LessonScreen(
    viewModel: LessonViewModel,
    courseType: String,
    uid: String,
    onBackToMenu: () -> Unit
) {

    val modules       by viewModel.modules.collectAsState()
    val currentModIdx  by
viewModel.currentModuleIndex.collectAsState()
    val currentPageIdx by
viewModel.currentPageIndex.collectAsState()

    val lives         by viewModel.lives.collectAsState()
    val hints         by viewModel.hints.collectAsState()
    val coins         by viewModel.coins.collectAsState()
    val showHint      by viewModel.showHint.collectAsState()
    val timeLeft   by
viewModel.timeToNextLife.collectAsState()

    val livesLabel = if (timeLeft > 0 && lives < 10)
        "$lives (${String.format("%02d:%02d", timeLeft/60,
timeLeft%60)})"
    else "$lives"

    val snack = remember { SnackbarHostState() }



    LaunchedEffect(Unit) {
        viewModel.events.collectLatest {
            when (it) {
                LessonViewModel.UiEvent.NoLives ->
                    snack.showSnackbar("У вас закінчились життя.
Спробуйте пізніше")
                LessonViewModel.UiEvent.NoHints ->
                    snack.showSnackbar("Підказок більше нема")
                LessonViewModel.UiEvent.NoCoins ->      // ←
нова гілка
                    snack.showSnackbar("Недостатньо монет для
покупки")
            }
        }
    }


    val pageCount =
modules.getOrNull(currentModIdx)?.pages?.size ?: 1
    val progress  = ((currentPageIdx +
1).coerceAtMost(pageCount)).toFloat() / pageCount

    Scaffold(
        snackbarHost = { SnackbarHost(snack) },
```

```
        topBar = {
            TopAppBar(
                title = { Text("Kvantor", color = KvTextColor) },
                actions = {
                    TextButton(
                        onClick = {
                            viewModel.saveProgress()
                            onBackToMenu()
                        }
                    ) {
                        Text("Меню", color = KvTextColor)
                    }
                },
                colors =
TopAppBarDefaults.topAppBarColors(containerColor =
KvBg)
            )
        },


        bottomBar = {
            Column {

                Row(
                    Modifier
                        .fillMaxWidth()
                        .background(KvBg)
                        .padding(horizontal = 12.dp, vertical = 4.dp),
                    horizontalArrangement =
Arrangement.SpaceEvenly,
                    verticalAlignment = Alignment.CenterVertically
                ) {
                    StatusChip(Icons.Default.Favorite,       livesLabel,
"Lives")
                    StatusChip(Icons.Default.Lightbulb,
hints.toString(),  "Hints")
                    StatusChip(Icons.Default.MonetizationOn,
coins.toString(),  "Coins")
                }


                HorizontalDivider(color = KvAccent.copy(.3f),
thickness = 1.dp)
                LinearProgressIndicator(
                    progress   = { progress },
                    modifier   = Modifier.fillMaxWidth().height(4.dp),
                    color      = KvAccent,
                    trackColor = KvAccent.copy(.1f)
                )
            }
        },
        containerColor = KvBg
    ) { padding ->
        PageContainer(Modifier.padding(padding)) {
            when {
                modules.isEmpty() -> Box(
                    Modifier.fillMaxSize(),
                    contentAlignment = Alignment.Center
                ) { CircularProgressIndicator(color = KvAccent) }

                currentModIdx < modules.size ->
LessonModuleContent(
                    module       = modules[currentModIdx],
                    pageIndex    = currentPageIdx,
                    isLastModule = currentModIdx ==
modules.lastIndex,
                    courseType   = courseType,
                    uid          = uid,
                    vm           = viewModel,
                    onNext       = viewModel::next,
                    onBackToMenu = onBackToMenu
                )

                else -> CourseFinishedScreen(onBackToMenu)
            }


            showHint?.let { hint ->
                LaunchedEffect(hint) {
                    snack.showSnackbar("💡 $hint")
                    viewModel.clearHint()
                }
            }
        }
    }
}


@Composable
fun LessonModuleContent(
    module: Module,
    pageIndex: Int,
    isLastModule: Boolean,
    courseType: String,
    uid: String,
    vm: LessonViewModel,          // отримали
    onNext: () -> Unit,
    onBackToMenu: () -> Unit
) {
    val page = module.pages.getOrNull(pageIndex)
    var done by remember(pageIndex) { mutableStateOf(false) }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .verticalScroll(rememberScrollState())
            .padding(24.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        /* HERO-іконка */
        AnimatedVisibility(visible = true, enter = fadeIn()) {
            if (courseType == "javascript") {
                Icon(Icons.Filled.Code, null, tint = KvAccent,
modifier = Modifier.size(72.dp))
            } else Text("💻", fontSize = 64.sp)
        }

        Spacer(Modifier.height(16.dp))

        Text(
            text      = "Модуль: ${module.title}",
            style     =
MaterialTheme.typography.titleLarge.copy(fontWeight =
FontWeight.Bold),
            color     = KvTextColor,
            textAlign = TextAlign.Center
        )
        Spacer(Modifier.height(32.dp))

        /* ——— контент сторінки ——— */
        when (page) {
            is Page.Theory     -> Text(page.text, color =
KvTextColor, textAlign = TextAlign.Center)
                .also { done = true }

            is Page.Test       -> TestPage(page, vm) { done = true }
            is Page.CodingTask -> CodingTaskView(page) { done
```

```
= true }

        is Page.Final     -> {
            LaunchedEffect(uid, courseType) {
                val achId = if (courseType == "python")
"PY_MASTER" else "JS_SAMURAI"
                AchievementManager.unlockAchievement(uid,
achId)
            }

            Text(page.message, color = KvTextColor, textAlign
= TextAlign.Center)
            done = true
            Spacer(Modifier.height(32.dp))
            KvantorButton(
                text    = if (isLastModule) "Повернутися в меню"
else "До наступного модуля",
                onClick = if (isLastModule) onBackToMenu else
onNext
            )
        }

        null -> {}
    }

    if (done && page !is Page.Final) {
        Spacer(Modifier.height(32.dp))
        KvantorButton("Далі", onClick = onNext)
    }
  }
}

/* ———————————— Фінальний екран
———————————— */
@Composable
fun CourseFinishedScreen(onBackToMenu: () -> Unit) {
  Column(
    Modifier.fillMaxSize(),
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.Center
  ) {
    Text(
      "🎉 Вітаємо!\nВи завершили всі модулі.",
      style     = MaterialTheme.typography.titleLarge,
      color     = KvTextColor,
      textAlign = TextAlign.Center
    )
    Spacer(Modifier.height(32.dp))
    KvantorButton("Повернутися в меню", onClick =
onBackToMenu)
  }
}

/* ———————————— Сторінка-тест
———————————— */
@Composable
fun TestPage(
  test: Page.Test,
  vm: LessonViewModel,
  onDone: () -> Unit
) {
  var selected by remember(test) { mutableStateOf(-1) }
  var checked  by remember(test) { mutableStateOf(false) }

  Text(test.question, color = KvTextColor, textAlign =
TextAlign.Center)
  Spacer(Modifier.height(24.dp))

  test.answers.forEachIndexed { idx, ans ->
    Row(verticalAlignment = Alignment.CenterVertically) {
      RadioButton(
        selected = selected == idx,
        onClick  = { selected = idx; checked = false },
        colors   = RadioButtonDefaults.colors(
          selectedColor   = KvTextColor,
          unselectedColor = KvTextColor
        )
      )
      Spacer(Modifier.width(8.dp))
      Text(ans, color = KvTextColor)
    }
  }

  /* ---- ПІДКАЗКА ---- */
  if (test.hint != null) {
    Spacer(Modifier.height(8.dp))
    KvantorButton(
      text    = "Підказка
(${vm.hints.collectAsState().value})",
      enabled = vm.hints.collectAsState().value > 0,
      onClick = { vm.requestHint(test) }   // ← явне ім’я
параметра
    )
  }

  /* ---- Перевірка ---- */
  Spacer(Modifier.height(24.dp))
  KvantorButton(
    text    = "Перевірити",
    enabled = selected != -1,
    onClick = {
      vm.checkAnswer(test, selected)
      checked = true
      onDone()
    }
  )

  /* ---- Результат ---- */
  val correct = remember(checked) { selected ==
test.correctAnswerIndex }
  if (checked) {
    Spacer(Modifier.height(16.dp))
    Text(
      if (correct) "✅ Правильно (+10₡)" else "❌
Неправильно (-1 ❤️)",
      color     = if (correct) KvAccent else
KvAccent.copy(.7f),
      textAlign = TextAlign.Center
    )
  }
}

/* ———————————— Допоміжне: статус-чіп
———————————— */
@Composable
private fun StatusChip(icon: ImageVector, valueText: String,
label: String) = Row(
  verticalAlignment     = Alignment.CenterVertically,
  horizontalArrangement = Arrangement.spacedBy(4.dp)
) {
  Icon(icon, null, tint = KvAccent, modifier =
Modifier.size(16.dp))
  Text(valueText, color = KvTextColor, fontSize = 14.sp)
  Text(label,     color = KvTextColor, fontSize = 12.sp)
}
```

Вміст файлу LessonViewModel

```
            // File:
app/src/main/java/com/bambiloff/kvantor/LessonViewModel.k
t
package com.bambiloff.kvantor

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.google.firebase.Timestamp
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.firestore.FirebaseFirestore
import com.google.firebase.firestore.SetOptions
import kotlinx.coroutines.delay
import kotlinx.coroutines.flow.*
import kotlinx.coroutines.launch
import kotlinx.coroutines.tasks.await

class LessonViewModel(
    private val courseType: String = "python"   // "python" або
"javascript"
) : ViewModel() {

    /* ---------------- Firebase ---------------- */
    private val db   = FirebaseFirestore.getInstance()
    private val auth = FirebaseAuth.getInstance()

    /* ---------------- Data ---------------- */
    private val _modules           =
MutableStateFlow<List<Module>>(emptyList())
    val          modules: StateFlow<List<Module>> = _modules

    private val _currentModuleIndex = MutableStateFlow(0)
    val          currentModuleIndex: StateFlow<Int> =
_currentModuleIndex

    private val _currentPageIndex   = MutableStateFlow(0)
    val          currentPageIndex:  StateFlow<Int> =
_currentPageIndex

    /* ----------  Gamification  ---------- */
    private val _lives     = MutableStateFlow(0)
    val          lives:    StateFlow<Int> = _lives

    private val _hints     = MutableStateFlow(0)
    val          hints:    StateFlow<Int> = _hints

    private val _coins     = MutableStateFlow(0)
    val          coins:    StateFlow<Int> = _coins

    private val _showHint  = MutableStateFlow<String?>(null)
    val          showHint: StateFlow<String?> = _showHint

    /* ----------  last life timestamp & таймер ---------- */
    private val _lastLifeTS     =
MutableStateFlow<Timestamp?>(null)
    private val _timeToNextLife = MutableStateFlow(0L)      //
сек
    val          timeToNextLife: StateFlow<Long> =
_timeToNextLife

    /* (можна реагувати у UI) */
    sealed interface UiEvent {
        object NoLives  : UiEvent
        object NoHints  : UiEvent
        object NoCoins  : UiEvent
    }
    private val _events = MutableSharedFlow<UiEvent>()
    val          events = _events.asSharedFlow()

    /* -------- константи -------- */
    private companion object {
        const val LIFE_COST        = 30
        const val HINT_COST        = 20
        const val MAX_LIVES        = 10
        const val RESTORE_INTERVAL = 2 * 60   // 10 хвилин
= 600 с
    }

    /* ---------------- Current module helper ---------------- */
    val currentModule = combine(_modules,
_currentModuleIndex) { list, idx ->
        list.getOrNull(idx)
    }.stateIn(viewModelScope, SharingStarted.Eagerly, null)

    /* ----------------------------------------------------------------
*/
    init {
        auth.currentUser?.uid?.let { uid ->
            /* ---- 1. live listener на документ користувача ---- */
            db.collection("users").document(uid)
                .addSnapshotListener { snap, _ ->
                    snap ?: return@addSnapshotListener
                    _lives.value      = (snap.getLong("lives") ?:
0).toInt()
                    _hints.value      = (snap.getLong("hints") ?:
0).toInt()
                    _coins.value      = (snap.getLong("coins") ?:
0).toInt()
                    _lastLifeTS.value =
snap.getTimestamp("lastLifeTS")
                }

            /* ---- 2. одноразова ініціалізація ---- */
            viewModelScope.launch {
                UserBootstrapper.ensureStats(uid)
                GameManager.maybeRestoreLife(uid)
            }

            /* ---- 3. тікер 1 сек: рахуємо до наступного ❤️ ----
*/
            viewModelScope.launch {
                while (true) {
                    delay(1_000)

                    val livesNow = _lives.value
                    if (livesNow >= MAX_LIVES) {
                        _timeToNextLife.value = 0
                        continue
                    }

                    val last = _lastLifeTS.value
                    if (last == null) { _timeToNextLife.value = 0;
continue }

                    val passed = (Timestamp.now().seconds -
last.seconds).toInt()
                    val left   = (RESTORE_INTERVAL -
passed).coerceAtLeast(0)
                    _timeToNextLife.value = left.toLong()

                    if (left == 0) {
                        GameManager.maybeRestoreLife(uid,
MAX_LIVES)
                    }
                }
            }
        }
    }

    /* ---------------- Modules loading ---------------- */
    fun loadModules() {
```

```
        viewModelScope.launch {
            val collection = if (courseType == "javascript")
"modules_js" else "modules"
            try {
                val snapshot = db.collection(collection).get().await()
                _modules.value = snapshot.documents
                    .mapNotNull {
it.toObject(ModuleDto::class.java)?.toModule() }
                    .sortedBy { it.id }

                restoreProgress()          // відновлюємо позицію
            } catch (e: Exception) {
                e.printStackTrace()
            }
        }
    }

    /* ---------- Progress save / restore ---------- */
    fun saveProgress() {
        auth.currentUser?.uid?.let { uid ->
            val progress = mapOf(
                "moduleIndex" to _currentModuleIndex.value,
                "pageIndex"   to _currentPageIndex.value
            )
            db.collection("users").document(uid)
                .set(
                    mapOf("progress" to mapOf(courseType to
progress)),
                    SetOptions.merge()
                )
        }
    }

    private fun restoreProgress() {
        viewModelScope.launch {
            auth.currentUser?.uid?.let { uid ->
                try {
                    val doc  =
db.collection("users").document(uid).get().await()
                    @Suppress("UNCHECKED_CAST")
                    val root = doc.get("progress") as? Map<String,
Map<String, Long>>
                    val cur  = root?.get(courseType)

                    val mIdx = (cur?.get("moduleIndex") ?: 0L).toInt()
                    val pIdx = (cur?.get("pageIndex")   ?: 0L).toInt()

                    _currentModuleIndex.value =
                        mIdx.coerceIn(0,
_modules.value.lastIndex.coerceAtLeast(0))

                    val pagesInModule = _modules.value
                        .getOrNull(_currentModuleIndex.value)
                        ?.pages?.lastIndex ?: 0

                    _currentPageIndex.value =
                        pIdx.coerceIn(0, pagesInModule)

                } catch (e: Exception) {
                    e.printStackTrace()
                }
            }
        }
    }

    /* ---------- Gamification helpers ---------- */

    fun checkAnswer(page: Page.Test, userAnswerIndex: Int) =
viewModelScope.launch {
        val uid = auth.currentUser?.uid ?: return@launch
        if (userAnswerIndex == page.correctAnswerIndex) {
            GameManager.addCoins(uid, 10)
        } else {
            val ok = GameManager.spendLife(uid)
            if (!ok) _events.emit(UiEvent.NoLives)
        }
    }

    fun requestHint(page: Page.Test) = viewModelScope.launch
{
        val uid = auth.currentUser?.uid ?: return@launch
        val ok  = GameManager.spendHint(uid)
        if (ok) _showHint.value = page.hint else
_events.emit(UiEvent.NoHints)
    }

    fun clearHint() { _showHint.value = null }

    fun buyLife() = viewModelScope.launch {
        val uid = auth.currentUser?.uid ?: return@launch
        val ok  = GameManager.buy(uid, LIFE_COST) {
GameManager.addLives(uid, 1) }
        if (!ok) _events.emit(UiEvent.NoCoins)
    }

    fun buyHint() = viewModelScope.launch {
        val uid = auth.currentUser?.uid ?: return@launch
        val ok  = GameManager.buy(uid, HINT_COST) {
GameManager.addHints(uid, 1) }
        if (!ok) _events.emit(UiEvent.NoCoins)
    }

    /* ---------- Mark module completed ---------- */
    private fun markModuleCompleted(moduleId: String) { /*
без змін */ }

    /* ---------- Навігація ---------- */
    fun next() {
        if (_lives.value == 0) {
            viewModelScope.launch {
_events.emit(UiEvent.NoLives) }
            return
        }
        viewModelScope.launch {
            val module = currentModule.value ?: return@launch
            val lastModuleIndex = _modules.value.lastIndex
            if (_currentPageIndex.value < module.pages.lastIndex)
{
                _currentPageIndex.value += 1
            } else {
                markModuleCompleted(module.id)
                if (_currentModuleIndex.value < lastModuleIndex) {
                    _currentModuleIndex.value += 1
                    _currentPageIndex.value = 0
                } else {
                    auth.currentUser?.uid?.let { uid ->

CourseCompletionChecker.checkCourseCompleted(uid,
courseType)
                    }
                }
            }
            saveProgress()
        }
    }
}
```

```
package com.bambiloff.kvantor

import android.app.Activity
import android.content.Intent
import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.AnimatedVisibility
import androidx.compose.animation.core.animateFloatAsState
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import
androidx.compose.foundation.interaction.MutableInteractionSo
urce
import
androidx.compose.foundation.interaction.collectIsPressedAsSta
te
import androidx.compose.foundation.isSystemInDarkTheme
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.rememberScrollState
import
androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.DarkMode
import androidx.compose.material.icons.filled.LightMode
import androidx.compose.material.icons.filled.School
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.graphicsLayer
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.bambiloff.kvantor.ui.theme.KvantorTheme
import com.bambiloff.kvantor.ui.theme.Rubik
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.firestore.FirebaseFirestore
import com.google.firebase.firestore.SetOptions
import androidx.compose.ui.platform.testTag


class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            var isDarkTheme by remember { mutableStateOf(true)
}

            KvantorTheme(darkTheme = isDarkTheme) {
                PythonCourseScreen(
                    isDarkTheme   = isDarkTheme,
                    onToggleTheme = { isDarkTheme = it }
                )
            }
        }
    }
}

@Composable
fun PythonScreen() {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .testTag("python_header")   // ← ЯКІР ДЛЯ ТЕСТУ
    ) {
        Text("Python: перший урок")
    }
}

@Suppress("DiscouragedApi")
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun PythonCourseScreen(
    isDarkTheme: Boolean,
    onToggleTheme: (Boolean) -> Unit
) {
    val context = LocalContext.current
    val uid     = FirebaseAuth.getInstance().currentUser?.uid
    val db      = FirebaseFirestore.getInstance()
    val cs      = MaterialTheme.colorScheme

    var avatarResId by remember {
mutableStateOf(R.drawable.default_avatar) }
    LaunchedEffect(uid) {
        uid?.let { user ->
            db.collection("users").document(user).get()
                .addOnSuccessListener { doc ->
                    val name = doc.getString("avatarName") ?:
"default_avatar"
                    val id   = context.resources.getIdentifier(
                        name, "drawable", context.packageName
                    )
                    avatarResId = if (id != 0) id else
R.drawable.default_avatar
                }
        }
    }

    Box(
        Modifier
            .fillMaxSize()
            .background(cs.background)
            .testTag("python_header")
    ) {
        Column(
            Modifier
                .fillMaxSize()
                .padding(horizontal = 24.dp, vertical = 16.dp)
                .verticalScroll(rememberScrollState()),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Row(
                Modifier
                    .fillMaxWidth()
                    .padding(bottom = 16.dp),
                horizontalArrangement =
Arrangement.SpaceBetween,
                verticalAlignment     = Alignment.CenterVertically
            ) {
                Icon(
                    painter =
painterResource(R.drawable.ic_arrow_back),
                    contentDescription = "Назад",
                    tint = cs.onBackground,
                    modifier = Modifier
                        .size(28.dp)
                        .clickable {
                            context.startActivity(
                                Intent(context,
CourseSelectionActivity::class.java)

.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)
                            )
```

```
                (context as Activity).finish()
            }
        )

        IconToggleButton(
            checked = isDarkTheme,
            onCheckedChange = onToggleTheme
        ) {
            Icon(
                imageVector = if (isDarkTheme)
Icons.Default.DarkMode else Icons.Default.LightMode,
                contentDescription = null,
                tint = cs.primary
            )
        }

        Image(
            painter = painterResource(id = avatarResId),
            contentDescription = "Профіль",
            modifier = Modifier
                .size(36.dp)
                .clip(RoundedCornerShape(8.dp))
                .clickable {
                    context.startActivity(
                        Intent(context, ProfileActivity::class.java)
                    )
                }
        )
    }

    Text(
        text       = "PYTHON",
        fontSize   = 28.sp,
        fontWeight = FontWeight.Bold,
        fontFamily = Rubik,
        color      = cs.primary
    )

    Spacer(Modifier.height(24.dp))

    // Список тем з фоном 0xFF8C52FF
    val topics = listOf(
        "Вступ" to "Ознайомлення з Python. Напишемо
першу програму.",
        "Змінні" to "Що таке змінні, типи даних, як
оголошувати.",
        "Цикли" to "Цикли for та while, приклади з
практики.",
        "Умови" to "if, else, elif — логіка умов.",
        "Функції" to "Як створювати функції, передавати
параметри.",
        "Списки та словники" to "Основи списків та
словників у Python."
    )

    topics.forEach { (title, description) ->
        var expanded by remember { mutableStateOf(false) }
        Column(
            Modifier
                .fillMaxWidth()
                .padding(vertical = 4.dp)
                .clip(RoundedCornerShape(6.dp))
                .background(cs.primary)            // тепер
#8C52FF
                .clickable { expanded = !expanded }
                .padding(horizontal = 16.dp, vertical = 10.dp)
        ) {
            Text(
                text       = title,
                fontSize   = 16.sp,
                fontWeight = FontWeight.Medium,
                fontFamily = Rubik,
                color      = cs.onPrimary          // білий текст
            )
            AnimatedVisibility(expanded) {
                Text(
                    text       = description,
                    fontSize   = 14.sp,
                    fontFamily = Rubik,
                    color      = cs.onPrimary.copy(alpha = .9f),
                    modifier   = Modifier.padding(top = 6.dp)
                )
            }
        }
    }

    Spacer(Modifier.height(32.dp))
    Spacer(Modifier.weight(1f))

    Row(
        Modifier.fillMaxWidth(),
        horizontalArrangement =
Arrangement.spacedBy(16.dp)
    ) {
        val interaction = remember {
MutableInteractionSource() }
        val pressed by interaction.collectIsPressedAsState()
        val scale by animateFloatAsState(if (pressed) 0.95f
else 1f)

        Button(
            onClick = {
                uid?.let { user ->
                    db.collection("users")
                        .document(user)
                        .set(
                            mapOf("progress" to mapOf("python" to
mapOf("moduleIndex" to 0, "pageIndex" to 0))),
                            SetOptions.merge()
                        )
                }
                context.startActivity(
                    Intent(context,
LessonActivity::class.java).putExtra("courseType", "python")
                )
            },
            modifier         = Modifier
                .weight(1f)
                .graphicsLayer { scaleX = scale; scaleY = scale
},
            interactionSource = interaction,
            shape            = RoundedCornerShape(8.dp),
            colors           = ButtonDefaults.buttonColors(
                containerColor = cs.primary,
                contentColor   = cs.onPrimary
            )
        ) {
            Text("Почати курс з початку", fontFamily =
Rubik)
        }

        val interaction2 = remember {
MutableInteractionSource() }
        val pressed2 by
interaction2.collectIsPressedAsState()
        val scale2 by animateFloatAsState(if (pressed2) 0.95f
else 1f)

        Button(
            onClick = {
```

```
                context.startActivity(
                    Intent(context,
LessonActivity::class.java).putExtra("courseType", "python")
                )
            },
            modifier         = Modifier
                .weight(1f)
                .graphicsLayer { scaleX = scale2; scaleY =
scale2 },
            interactionSource = interaction2,
            shape            = RoundedCornerShape(8.dp),
                colors           = ButtonDefaults.buttonColors(
                    containerColor = cs.primary,
                    contentColor   = cs.onPrimary
                )
            ) {
                Text("Продовжити курс", fontFamily = Rubik)
            }
        }
    }
}
```

PageContainer.kt

```
        package com.bambiloff.kvantor

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp

/* кольори бренду */
val KvBg       = Color(0xFF390D58)
val KvAccent   = Color(0xFF8C52FF)
val KvTextColor = Color.White

/* універсальна кнопка */
@Composable
fun KvantorButton(
    text: String,
    enabled: Boolean = true,
    onClick: () -> Unit,
    modifier: Modifier = Modifier
) = Button(
        onClick  = onClick,
        enabled  = enabled,
        modifier = modifier,
        colors   = ButtonDefaults.buttonColors(
            containerColor         = KvAccent,
            contentColor           = Color.White,
            disabledContainerColor = KvAccent.copy(.4f),
            disabledContentColor   = Color.White.copy(.6f)
        ),
        shape = ButtonDefaults.filledTonalShape
    ) { Text(text) }

    /* обгортка сторінки */
    @Composable
    fun PageContainer(content: @Composable BoxScope.() ->
    Unit) = Box(
        modifier = Modifier
            .fillMaxSize()
            .background(KvBg)
            .padding(24.dp),
        contentAlignment = Alignment.Center,
        content = content
    )
```