НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО
В.о. завідувача кафедри
_____Олександр КОВАЛЬ
«_____»_____2025р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій»
спеціальності 121 «Інженерія програмного забезпечення»
на тему: «Додаток "Код-детектив" для вирішення логічних завдань з програмування»

Виконав:
Студент IV курсу, групи ТВ-11
Гудзовський Марк Юрійович _____
(підпис)

Керівник:
ст. викл. Дацюк О.А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант розділу «Віртуальна та додана реальність»
доц. Залєвська О.В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент:
_____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
(підпис)
«____»_____202_р.

ЗАВДАННЯ
на дипломну роботу студенту

_____Гудзовському Марку Юрійовичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи
«Додаток "Код-детектив" для вирішення логічних завдань з програмування»____
керівник роботи старший викладач._Дацюк Оксана Антонівна_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від «__»_____ 2025 року №_____
2. Строк подання студентом роботи _____
3. Вихідні дані до роботи: мови програмування Dart та Python, менеджер станів Bloc, бібліотеки інтерфейсів Django та Flutter, середовище розробки Cursor.
_____
4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити): Створення мобільного застосунку у стилі детективного розслідування, з набіром логічних завдань, пов'язаних з програмуванням, алгоритмами та енергетикою. Реалізувати систему підказок та систему рейтингу та досягнень для мотивації користувачів.
5. Перелік ілюстративного матеріалу: Схема структури бази даних, приклади інтерфейсу користувача, інструкції з деплою застосунку._____
6. Дата видачі завдання «__» _____ 202_ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	30.10.2025	виконано
2	Дослідження предметної області	31.10 - 31.12.2025	виконано
3	Постановка вимог до проектування системи	01.01 - 14.02.2025	виконано
4	Дослідження існуючих рішень	15.02 - 16.04.2025	виконано
5	Розробка програмного продукту	17.04 - 11.05.2025	виконано
6	Тестування	12.05 - 19.05.2025	виконано
7	Захист програмного продукту	16.05.2025	виконано
8	Оформлення дипломної роботи	20.05 – 30.05.2025	виконано
9	Передзахист	06.06.2025	виконано
10	Захист	20.06.2025	виконано

Студент _____ Гудзовський Марк
(підпис) (ім’я, прізвище)

Керівник роботи _____ Оксана Дацюк
(підпис) (ім’я, прізвище)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 55 сторінки, 22 рисунків, 2 додатка та 10 посилань.

Метою роботи є створення мобільного застосунку для вирішення логічних завдань з програмування, що відповідає сучасним вимогам користувачів – зручності, кросплатформеності та сучасності.

Для досягнення поставленої мети виконано такі завдання:

- проаналізовано існуючі рішення та виявлено їх основні недоліки;
- пропрацьовано ідею власного мобільного застосунку, що має виправити існуючі недоліки та покрити усі бажання користовачів;
- інтерфейс було спроектовано з урахуванням потреб різних користувачів — як студентів, так і досвідчених викладачів — із акцентом на зручність використання та інтуїтивну навігацію.
- інтегровано можливість спілкування з віртуальним асистентом, що працює завдяки моделі ШІ llama3:8b;
- розроблено сам мобільний застосунок та його завантаження на iOS та Android пристрої.

Практичне значення одержаних результатів полягає в отриманні кросплатформеного мобільного застосунку, що дозволить студентам та іншим користувачам вивчати програмування швидко та ефективно прямо в телефоні. Це дозволить хоч трошки скласти конкуренцію «думскролінгу» в інстаграмі та інших соціальних мережах.

Ключові слова: програмування, мобільний застосунок, тести, навчання, користувацький інтерфейс, штучний інтелект.

ABSTRACT

Structure and scope of the thesis. The work contains 55 pages, 22 figures, 2 appendix and 10 references.

The purpose of the work is to create a mobile application for solving logical programming problems that meets the modern requirements of users - convenience, cross-platform and modernity.

To achieve this goal, the following tasks were performed:

- existing solutions were analyzed and their main shortcomings were identified;
- the idea of our own mobile application was worked out, which should correct existing shortcomings and cover all the wishes of users;
- the interface was designed taking into account the needs of different users - both students and experienced teachers - with an emphasis on ease of use and intuitive navigation.
- the ability to communicate with a virtual assistant, which works thanks to the llama3:8b AI model, was integrated;
- the mobile application itself was developed and downloaded to iOS and Android devices.

The practical significance of the results obtained lies in obtaining a cross-platform mobile application that will allow students and other users to learn programming quickly and effectively right on the phone. This will allow us to at least somewhat compete with «dumscrolling» on Instagram and other social networks.

Keywords: programming, mobile application, tests, training, user interface, artificial intelligence

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (Application Programming Interface) — інтерфейс програмування додатків для інтеграції програмного забезпечення.

SDK (Software Development Kit) — набір інструментів, що використовуються для створення додатків.

DOM (Document Object Model) — спосіб представлення веб-сторінок у вигляді структурованого дерева.

GIT (Github) — веб-сервіс для хостингу IT-проектів та їхньої спільної розробки.

UX (User Experience) — досвід користувача, загальне враження користувача від взаємодії з веб-застосунком.

DRF (Django Rest Framework) — потужний та гнучкий інструментарій для створення веб-API.

БД – база даних

GQA (Grouped-Query Attention) — це варіант механізму уваги, який оптимізує обчислення в трансформерах.

JSON (JavaScript Object Notation) — формат обміну даними між клієнтом та сервером.

ПЗ (Програмне забезпечення) – набір програм для комп'ютера, що виконує певні завдання.

ШІ – Штучний інтелект

PDF (Portable Document Format) — формат файлів для представлення документів незалежно від апаратного та програмного забезпечення.

XML (EXtensible Markup Language) — стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками.

ВСТУП

У сучасному цифровому середовищі процес вивчення програмування зазнає кардинальних змін: замість паперових тестів і лекцій студенти обирають інтерактивні платформи, які дозволяють практикуватися одразу на реальних прикладах. Саме тут з’являється ідея додатку «Код-детектив», що об’єднує атмосферу детективного розслідування з вирішенням логічних завдань з програмування.

«Код-детектив» розроблений як мобільний додаток на Flutter з бекендом на Django і SQLite, і головна його мета — перетворити рутинні вправи на захопливу гру. Замість звичайних тестів користувач в ролі «детектива» отримує підказки та натяки до кожного завдання, а за правильне рішення здобуває очки відповідно до складності: easy — 1, medium — 3, hard — 5. А головне — весь процес навчання супроводжує можливість звернутись до вбудованого штучного інтелекту за допомогою з будь-яким запитанням та темою з програмування.

По-перше, гейміфікація підвищує мотивацію — студенти більш охоче повертаються до тренувань, щоб розкрити всі «таємниці» алгоритмів. По-друге, автоматизована система перевірки результатів миттєво оцінює рішення та оновлює рейтинг користувача в таблиці лідерів, що додає здорової конкуренції серед одногрупників.

Таким чином, «Код-детектив» не лише оптимізує процес навчання, але й формує спільноту, де кожен може розвивати аналітичне мислення та програмістські навички в інтригуючому стилі справжнього розслідування.

1 МЕТА, ПОСТАНОВКА ЗАДАЧІ ТА ОСНОВНІ ЗАВДАННЯ ПРАКТИКИ

В цьому розділі розповідається про аналіз основних проблем, які існують в інших застосунках для вивчення програмування, а також про основні шляхи вирішення цих проблем за допомогою кросплатформенності та штучного інтелекту. Розділ освітлює головні мінуси конкурентних продуктів та пропонує рішення для покращення UX.

1.1 Постановка задачі

Завдання полягає в розробці мобільного додатку «Код-детектив», який дозволяє вивчати мови програмування зі свого телефону в будь-якому місці проходячи короткі 5-хвилинні тести. Кожен тест може містити різні типи питань а також розділитися по категоріям на мови програмування, що дає змогу охопити широке коло тем та навичок.

Ключовою функцією є порівняння результатів користувачів. Серверна частина формує список лідерів, де кожен учасник бачить свій рейтинг і може змагатись з іншими, стимулюючи здорову конкуренцію і бажання «розкрити» всі алгоритмічні загадки. Також додаток дозволяє переглядати профілі інших користувачів та порівнювати результати конкретних пройдених тестів, та дивитись їх статистику.

Також передбачено інтеграцію ШІ-асистента, який допомагатиме детективам-студентам підказками або поясненнями до складних тестів. Загалом - треба розробці рішення, що об’єднує автоматизовану перевірку відповідей, соціальний компонент та ШІ-підтримку для ефективного та інноваційного навчання.

У сучасному технологічно розвиненому світі такі застосунки потрібні, щоб хоч якось конкурувати з тік-током та інстаграмом, що дозволить зробити студентів більш продуктивними та прискорити їх навчання.

1.2 Аналіз існуючих рішень та їх недоліки

Аналізуючи ринок на предмет існуючих продуктів, де користувачі могли б вивчити програмування вирішуючи цікаві логічні завдання, було виявлено два основні приклади: Codewars та Leetcode. Кожен з них представляє собою браузерний застосунок, де користувач вирішує логічні завдання з різних мов програмування для покращення своїх професійних навичок. В них дуже добре реалізовано інтеграція редактора коду прямо в браузері, а також велика база завдань.

Однак у всіх присутні дві основні проблеми:

– неможливість зручно використовувати сервіс з мобільного пристрою, бо вони представляють собою браузерну сторінку, з якої дуже проблематично програмувати на телефоні;

– дуже не цікава тематика застосунку а також процесу навчання, що дуже погано впливає на зацікавленість студентів та інших новачків у програмуванні.

Для вирішення першої проблеми можна використати фреймворк Flutter[1], який написаний мовою програмування Dart[6], та привносить дуже багато інноваційності в застосунок а саме кросплатформеність. Це означає наступне:

– додаток матиме одну код базу користувацького інтерфейсу написаного на Dart;

– додаток можна деплоїти на майже усі поширені платформи, такі як: iOS, Android, Desktop (Windows, MacOS, Linux), Web.

Завдяки такому підходу, користувачі матимуть змогу користуватись застосунком набагато частіше, а не тільки коли в них є доступ до комп’ютера.

Для вирішення другої проблеми був розроблений дизайн та тематика, що зацікавлює користувачів від зовсім малого віку до останніх курсів університету. Було вирішено розробити дуже цікавий та простий в користуванні інтерфейс, в стилі детективу для цікавого та динамічного процесу навчання. Таким чином додаток виділяється не тільки передовим функціоналом на ринку, а ще запам’ятовуєтьяс як бренд.

Висновки до розділу 1

У першому розділі було детально досліджено ключові недоліки існуючих платформ для вивчення програмування — обмежену мобільну доступність та низьку зацікавленість користувачів — і показано, як їх усунути за допомогою кросплатформеності та оригінального дизайн-рішення в стилі детективу. Обґрунтовано вибір Flutter для створення єдиної кодової бази, що працює на всіх популярних платформах, і запропоновано соціальний компонент із рейтингом та порівнянням результатів, який впроваджує здорову конкуренцію. Крім того, інтеграція ШІ-асистента забезпечує якісну підтримку під час проходження тестів. Таким чином, рішення «Код-детектив» поєднує автоматизовану перевірку, соціальну взаємодію та інтелектуальну допомогу для створення інноваційного та зручного інструменту навчання.

2 АНАЛІЗ ІНСТРУМЕНТІВ ДЛЯ СТВОРЕННЯ ТА РОЗГОРТАННЯ ПЗ

Цей розділ розкриває етап аналізу інструментів для створення та розгортання програмного продукту. Тут показано яким чином був зроблений вибір редакторів коду, архітектури системи та моделі штучного інтелекту.

2.1 Вибір середовища розробки

Під час розробки мобільного додатку «Код-детектив» було прийнято рішення використовувати два різних середовища розробки для фронтенду та бекенду. Для Flutter частини проекту було обрано Cursor.ai, а для Django[3] бекенду - PyCharm. Такий підхід дозволив максимально ефективно працювати з кожною технологією, використовуючи найкращі інструменти для відповідних завдань.

Зазвичай розробники намагаються використовувати одне середовище для всього проекту, але в моєму випадку специфіка мобільної та серверної розробки вимагала різних підходів. Cursor.ai виявився ідеальним для швидкої розробки Flutter компонентів завдяки штучному інтелекту, тоді як PyCharm забезпечив потужну підтримку Django та Python екосистеми.

2.1.1 Cursor.ai для Flutter розробки

Cursor.ai - це досить новий редактор коду, який можна назвати еволюцією VSCode з вбудованим штучним інтелектом. Розроблений спеціально для сучасної розробки, він дозволяє значно прискорити процес написання коду завдяки GPT-4 інтеграції. Особливо це корисно для Flutter розробки, де потрібно швидко створювати багато віджетів та екранів.

Головна фішка Cursor.ai - це можливість спілкування з кодом природною мовою. Можна просто описати що потрібно зробити, і редактор згенерує відповідний Dart код. Наприклад, при описі «створи Flutter віджет для відображення списку завдань з можливістю додавання нових» редактор генерує готовий StatefulWidget з усіма необхідними методами та UI елементами.

Особливо корисним виявилося те, що Cursor розуміє контекст всього Flutter проекту. Коли відбувається робота над новим віджетом, він автоматично підказує які параметри потрібні, як правильно використовувати існуючі state management рішення та які Material Design компоненти застосувати відповідно до дизайну додатку.

Також Cursor.ai має вбудовану підтримку всіх Flutter та Dart специфічних функцій - hot reload, pubspec.yaml management, Flutter команди тощо. Автокомпліт працює на рівні який важко уявити в звичайних редакторах, бо враховується не тільки синтаксис Dart, але й логіка Flutter фреймворку.

Дуже зручною функцією є можливість рефакторингу Flutter коду за допомогою ШІ. Можна виділити фрагмент коду і попросити оптимізувати його під Flutter best practices, додати error handling або переписати під новий state management паттерн. Це значно економить час на рутинних завданнях типу створення boilerplate коду для нових екранів.

Інтерфейс Cursor.ai мінімалістичний та не перевантажений, що дозволяє зосередитися на коді. Вбудована підтримка Git працює швидко та інтуїтивно. Також редактор має інтеграцію з Flutter DevTools, що дуже зручно під час розробки та дебагингу мобільного UI.

Для роботи з Flutter проектами Cursor.ai автоматично розпізнає структуру проекту та налаштовує відповідні команди для запуску на різних платформах - Android, iOS, Web та Desktop. Це особливо важливо для кросплатформенного додатку, де потрібно тестувати функціональність на різних пристроях.

Єдиний мінус який було помічено - це залежність від інтернет з'єднання для роботи ШІ функцій. Без інтернету Cursor перетворюється на звичайний VS Code, але базові функції редагування працюють нормально.

2.1.2 PyCharm для Django розробки

PyCharm[7] від JetBrains - це потужна IDE спеціально розроблена для Python розробки. Для Django проектів це один з найкращих виборів, оскільки має глибоку інтеграцію з фреймворком та розуміє всю його структуру.

Найбільша перевага PyCharm - це інтелектуальна навігація по коду. Він розуміє Django models, views, urls та templates як єдину систему. Можна легко переходити від моделі до всіх місць де вона використовується, від view до відповідного template, від URL до view функції тощо.

Дебагер в PyCharm просто неперевершений. Можна встановлювати breakpoints, переглядати стан змінних, виконувати код по кроках та навіть змінювати значення під час виконання. Це особливо корисно при розробці складної бізнес-логіки або при інтеграції з зовнішніми API для додатку.

Django specific функції PyCharm включають автокомпліт для Django ORM, підтримку Django templates з syntax highlighting, автоматичне створення Django проектів та apps, інтеграцію з Django admin панеллю та багато іншого. Для проекту «Код-детектив» це було особливо корисно при створенні моделей для завдань, користувачів та системи рейтингів.

Вбудований термінал дозволяє швидко виконувати Django команди типу python manage.py migrate для міграцій або python manage.py runserver для запуску сервера не виходячи з IDE. Також є підтримка віртуальних середовищ Python, що дуже важливо для ізоляції залежностей проекту.

PyCharm має потужну систему рефакторингу коду - можна легко перейменовувати змінні, класи, методи по всьому проекту, витягувати методи в

окремі функції, переносити код між файлами тощо. Все це робиться безпечно з урахуванням всіх залежностей.

База даних explorer дозволяє підключатися до SQLite, PostgreSQL, MySQL та інших БД прямо з IDE. Можна виконувати SQL запити, переглядати дані, редагувати схему БД. Для Django проектів це дуже зручно, бо можна одразу бачити як зміни в models відображаються в базі даних.

Інтеграція з системами контролю версій (Git, GitHub) працює бездоганно. Можна переглядати історію змін, робити commit, push, merge прямо з IDE. Також є підтримка code review процесів, що важливо при командній розробці.

Система плагінів PyCharm дозволяє розширити функціональність під конкретні потреби проекту. Є плагіни для Docker, різних баз даних, REST API розробки тощо. Для серверної частини додатку «Код-детектив» було встановлено плагіни для роботи з JSON API та документування REST endpoints.

Єдиний недолік PyCharm - це те що він досить ресурсозатратний, особливо на слабких машинах може працювати повільно. Також платна версія PyCharm Professional коштує чимало, хоча для студентів є безкоштовна ліцензія від JetBrains.

2.2 Архітектура програмної системи

Основна ідея, як вже було сказано, полягає в поєднанні Flutter з Django+SQLite бекендом. Мобільна частина містить в собі весь користувацький інтерфейс та реалізує бізнес логіку за допомогою бібліотеки BloC[2]. Серверна частина спілкується з базою даних через вбудовані механізми Django а також реалізує мікросервіс чату як зовнішній сервіс, що дозволяє зробити його модульним і використовувати для будь-якої частини проекту в майбутньому. Сам сервіс спілкується в Ollama AI API, що є безкоштовним ресурсом ідеально підходящим для нашого проекту.

Оскільки застосунок передбачає постійну взаємодію з сервером, бібліотека Dio для Flutter дозволяє дуже детально налаштовувати запити та хендлери для офлайн режиму застосунку і кешування результатів попередніх запитів. Це дозволить користувачу проходити тести навіть з поганим інтернет з’єднанням.

2.2.1 Архітектура «Clean Architecture» застосунку

Мобільний додаток «Код-детектив» було розроблено з використанням принципів чистої архітектури з чітким розділенням між рівнями презентації, домену та даних. Для управління станом було обрано паттерн BLoC з компонентами бізнес-логіки, а впровадження залежностей реалізовано через локатор сервісів. Архітектура додатку побудована за модульним принципом, де кожна функціональна область має власну структуру залежностей та компонентів.

Архітектура складається з чотирьох основних рівнів. Рівень даних містить всі джерела даних додатку - від HTTP клієнта для роботи з REST API до локального сховища для збереження налаштувань. На цьому рівні реалізовано конкретні класи джерел даних, які відповідають за отримання інформації з зовнішніх джерел, включаючи дані головної сторінки, тести та аутентифікацію. Рівень домену включає в себе основні сутності додатку такі як користувач, тест, питання та інтерфейси репозиторіїв. Цей рівень визначає бізнес-логіку додатку та не залежить від зовнішніх бібліотек чи фреймворків.

Рівень презентації відповідає за користувацький інтерфейс та управління станом. Він включає основні екрани додатку - головну сторінку з тестами та таблицею лідерів, екран для проходження тестів, сторінку перегляду профілю користувача та екран аутентифікації. Кожен екран має відповідний компонент для управління станом та бізнес-логікою. Інфраструктурний рівень забезпечує технічну основу додатку включаючи HTTP клієнт, локальне сховище даних та систему впровадження залежностей.

Для управління залежностями було реалізовано модульну систему на основі локатора сервісів з використанням архітектури на основі міксинів. Головний контейнер впровадження поєднує кілька спеціалізованих компонентів - базовий клас для основних залежностей, компонент для аутентифікації та компонент для функцій тестування та ШІ чату. Такий підхід дозволяє організувати залежності за функціональними областями та спрощує підтримку коду в майбутньому.

HTTP клієнт було налаштовано з комплексною системою перехоплювачів для забезпечення надійної роботи з API. Система автоматично додає заголовки авторизації до запитів, логує HTTP помилки для відладки, обробляє відповіді від сервера та забезпечує форматування HTTP запитів під час розробки. Конфігурація включає базову адресу з файлу налаштувань, таймаути в 150 секунд для підключення та отримання даних, стандартні JSON заголовки з налаштуваннями міждоменних запитів.

Для управління станом додатку було використано паттерн BLoC з реалізаціями компонентів стану, що забезпечує передбачувану та тестовану архітектуру. Кожна основна функціональна область має власний компонент управління станом - для головного екрану, управління тестами, аутентифікації, профілю користувача та ШІ асистента. Всі компоненти реєструються як ледачі одинаки в системі впровадження залежностей, що забезпечує єдиний екземпляр для всього додатку але створює їх тільки при першому використанні.

Головний екран додатку реалізовано як основну точку входу та панель керування застосунку. Він містить верхню панель з меню, заголовок додатку «Код-детектив», віджет для фільтрації за мовами програмування, секцію з доступними тестами, секцію таблиці лідерів та плаваючу кнопку для доступу до ШІ чату. Екран використовує анімації входу з поетапними ефектами появи та ковзання, функціональність оновлення даних при потягуванні вниз та адаптивний макет з прокручуваним контентом.

Для локального збереження даних використовується система налаштувань, яка зберігає токени аутентифікації, інформацію про сесію користувача та

налаштування додатку. Інтеграція з ШІ асистентом реалізована через кілька точок доступу - загальний ШІ чат через плаваючу кнопку дій, контекстну допомогу для окремих питань тестів та модальний інтерфейс для взаємодії з ШІ. Комунікація з бекендом відбувається через структуровані точки доступу API для аутентифікації, отримання даних тестів, збереження прогресу користувача та інтеграції з ШІ сервісами.

2.2.2 Архітектура Django бекенду

Серверна частина додатку «Код-детектив» було розроблено на фреймворку Django з використанням Django REST Framework для створення RESTful API. Архітектура серверної частини слідує стандартним принципам Django з чітким розділенням на моделі, представлення та маршрутизацію. API організовано навколо ресурсо-орієнтованих точок доступу, які обробляють дані мов програмування, тести, питання, профілі користувачів та функції ШІ асистента.

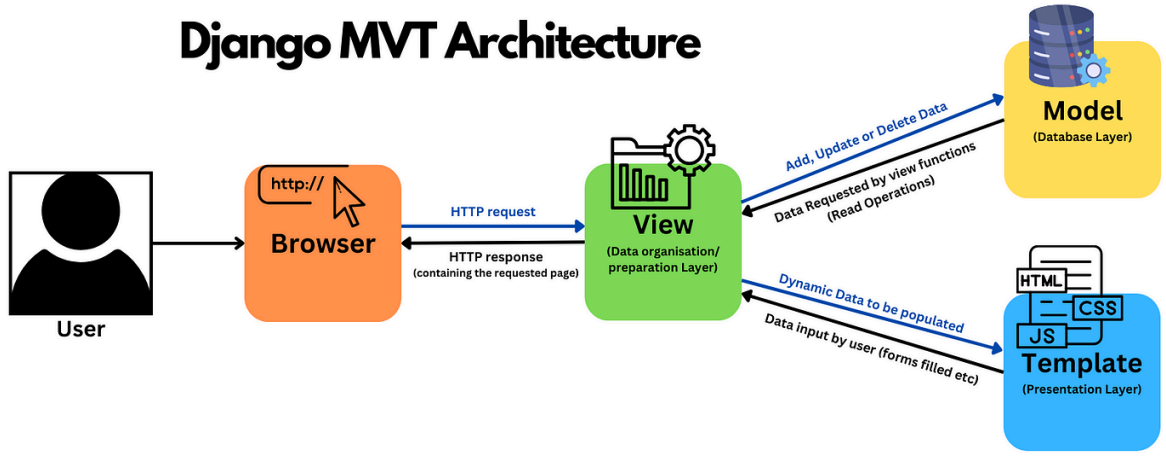


Рисунок 2.1 – MVT Архітектура Django[9]

Структура API побудована за конвенційним REST архітектурним паттерном з використанням стандартних HTTP методів та статус-кодів. Маршрутизація поєднує автоматичний роутер Django REST Framework для стандартних операцій з явними визначеннями шляхів для спеціалізованих точок доступу. Основні ресурси

включають мови програмування, тести, питання, користувацькі профілі та систему рейтингів.

Точки доступу API розділено на дві основні категорії. Перша категорія включає ViewSet endpoints для стандартних CRUD операцій - отримання списку мов програмування та доступних тестів через автоматичний роутер. Друга категорія містить спеціалізовані APIView endpoints для складної бізнес-логіки, включаючи аутентифікацію користувачів, отримання питань тесту, подання відповідей, ШІ асистент для питань, профілі користувачів, таблицю лідерів та завершені тести.

Система представлень реалізує два основні паттерни для обробки запитів. ViewSet класи використовуються для стандартних операцій читання ресурсів з додатковою логікою фільтрації. Наприклад, набір представлень тестів реалізує складну логіку фільтрації за мовою програмування та виключає тести, які користувач уже пройшов. APIView класи обробляють складну бізнес-логіку, яка не підходить для стандартних CRUD паттернів, включаючи обробку подань тестів, розрахунок очок та надання ШІ допомоги.

Рівень серіалізації трансформує дані між Python об'єктами та JSON представленнями з використанням серіалізаторів Django REST Framework. Основні серіалізатори включають моделі для мов програмування, тестів, питань та користувачів з обчислюваними полями та вкладеною серіалізацією. Система серіалізації підтримує обчислювані поля для динамічних даних, вкладену серіалізацію для складних структур даних та доступ до пов'язаних моделей для отримання додаткової інформації.

Процес подання тесту реалізовано через складний алгоритм обробки відповідей. Система валідує вхідні дані відповідей, перевіряє правильність кожної відповіді порівняно з правильною відповіддю питання, оновлює або створює записи прогресу користувача, розраховує очки з множниками складності та оновлює загальний рахунок користувача в профілі. Після завершення створюється запис про пройдений тест з результатами.

ШІ асистент інтегровано через дві основні точки доступу. Перша забезпечує допомогу для конкретних питань тестів з контекстом питання, а друга надає загальну допомогу з програмування без прив'язки до конкретного питання. Система будує контекстуальні дані для питань та використовує зовнішній сервіс чату для генерації відповідей.

Аутентифікація та дозволи реалізовано через токен-базовану систему з селективним захистом точок доступу. Публічні точки доступу включають таблицю лідерів та загальну інформацію про мови програмування та тести. Аутентифіковані точки доступу вимагають валідного токена для всіх операцій, специфічних для користувача, включаючи подання тестів, ШІ асистент та управління профілями. Система дозволів забезпечує контроль доступу на рівні представлень з використанням декораторів аутентифікації.

2.3 Розгортання проєкту

Розгортання проєкту передбачає перенесення додатку на робочі сервери для публічного доступу користувачів. Цей процес включає розгортання серверної частини на хмарній платформі та підготовку мобільного додатку для розповсюдження через пряме встановлення. Для проєкту «Код-детектив» було обрано комбінований підхід - розгортання Django бекенду на платформі Render та підготовка Flutter додатку у вигляді APK та IPA файлів.

Розгортання складається з двох основних етапів, кожен з яких має свої особливості та вимоги. Серверна частина потребує налаштування хмарного середовища з підтримкою Python та Django, конфігурації бази даних, налаштування змінних середовища та забезпечення безперервної роботи API. Мобільна частина вимагає компіляції Flutter коду для різних платформ, налаштування сертифікатів підпису та оптимізації для розповсюдження.

2.3.1 Розгортання Django на Render

Render - це сучасна хмарна платформа, яка надає простий та ефективний спосіб розгортання веб-додатків та API. Платформа підтримує автоматичне розгортання з Git репозиторіїв, що значно спрощує процес деплою та підтримки додатків. Для Django проектів Render пропонує інтегровану підтримку Python середовища з автоматичним встановленням залежностей та налаштуванням сервера.

Основною перевагою Render є його простота налаштування порівняно з традиційними хмарними провайдерами. Платформа автоматично розпізнає Django проект, встановлює необхідні залежності з requirements.txt та запускає сервер з правильними налаштуваннями. Це особливо зручно для проектів, які не потребують складної інфраструктури та можуть працювати в стандартному хмарному середовищі.

Render надає безкоштовний тарифний план, який включає достатні ресурси для розробки та тестування додатків. Платформа автоматично налаштовує HTTPS з'єднання, що забезпечує безпечну комунікацію між мобільним додатком та сервером. Також доступна інтеграція з PostgreSQL базою даних, яка автоматично конфігурується та підключається до Django додатку.

Процес розгортання включає створення нового веб-сервісу в панелі керування Render, підключення GitHub репозиторію з кодом проекту та налаштування змінних середовища. Render автоматично виявляє зміни в репозиторії та перезапускає сервіс з оновленим кодом, що забезпечує безперервну інтеграцію та розгортання.

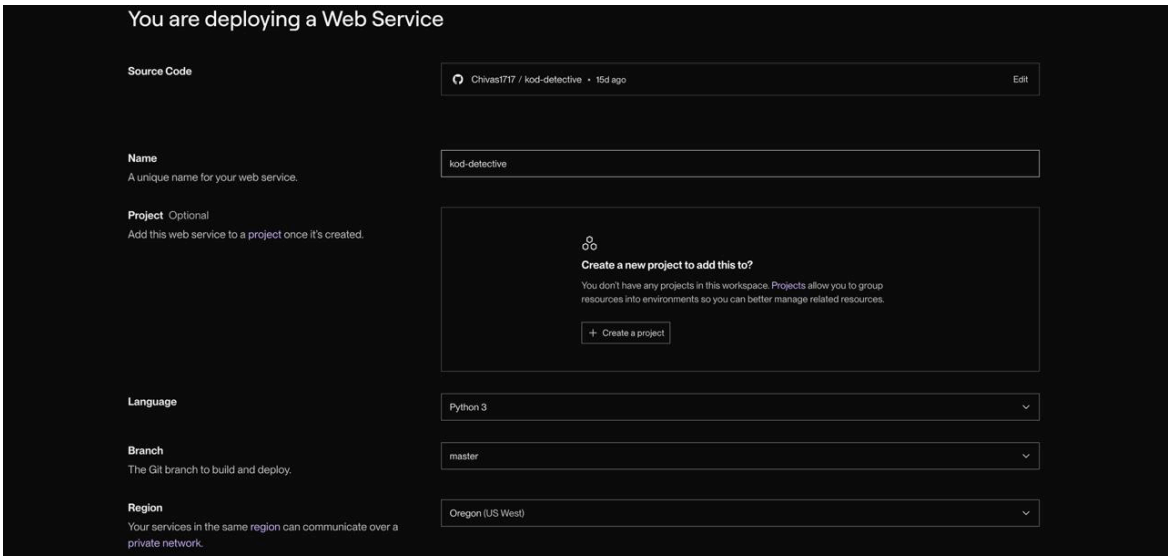


Рисунок 2.2 – Налаштування Render веб сервісу

Для проекту «Код-детектив» було налаштовано змінні середовища для підключення до бази даних, секретних ключів Django та конфігурації ШІ сервісів. Render забезпечує захищене зберігання цих змінних та їх автоматичне підставлення під час запуску додатку. Платформа також надає детальні логи роботи сервера, що спрощує діагностику та усунення проблем.

2.3.2 Створення та налаштування iOS застосунку

Мобільний додаток Flutter може бути розгорнуто як APK файл для Android пристроїв або IPA файл для iOS пристроїв. Цей підхід дозволяє розповсюджувати додаток безпосередньо користувачам без проходження процесу модерації в офіційних магазинах додатків. Для проекту «Код-детектив» було підготовлено обидва варианти для максимального охоплення цільової аудиторії.

Процес створення APK файлу для Android включає компіляцію Flutter коду в нативний Android код з використанням команди flutter build apk. Система збірки автоматично оптимізує код, ресурси та створює готовий для встановлення файл. Для забезпечення сумісності з різними Android пристроями було налаштовано мінімальну версію Android SDK та цільову версію API.

Створення IPA файлу для iOS вимагає додаткових налаштувань, включаючи реєстрацію в Apple Developer Program та налаштування сертифікатів підпису. Flutter дозволяє створювати iOS збірки на macOS системах з використанням Xcode інструментів. Процес включає налаштування bundle identifier, версії додатку та сертифікатів для розповсюдження поза App Store.

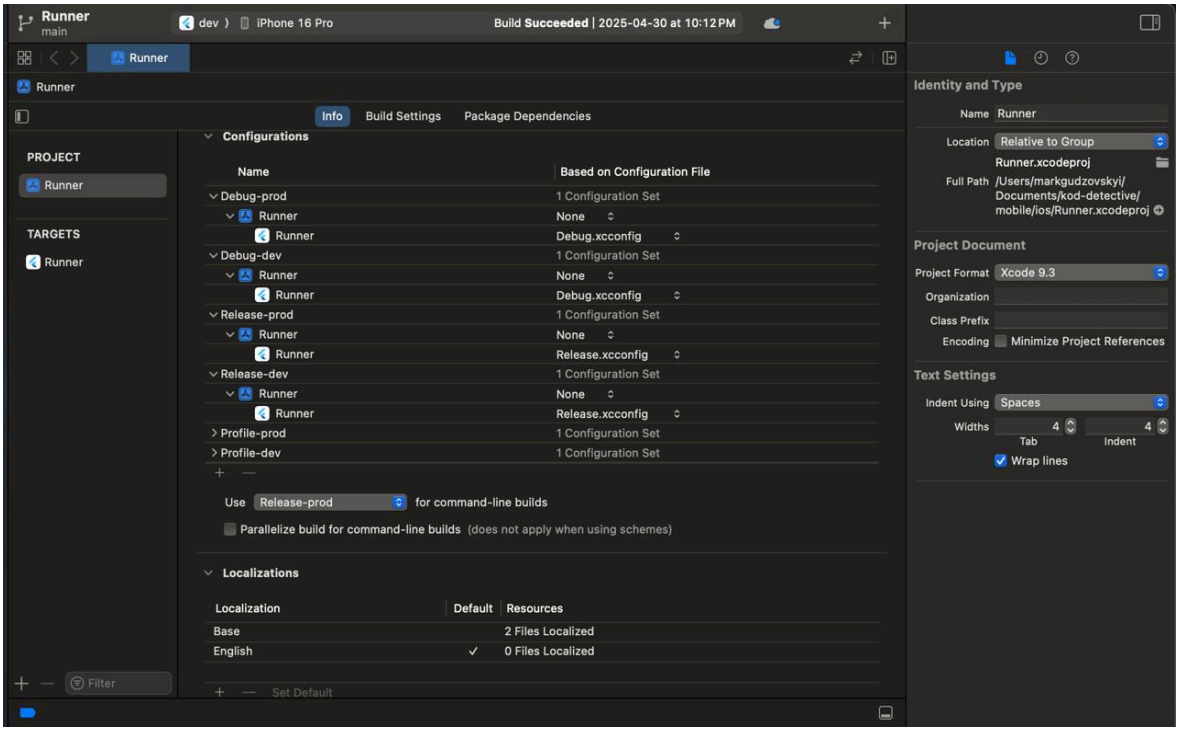


Рисунок 2.3 – Налаштування застосунку в Xcode

Обидва типи файлів було оптимізовано для розміру та продуктивності. Flutter автоматично видаляє невикористаний код та ресурси під час збірки, що зменшує розмір фінального додатку. Також було налаштовано правильні іконки додатку, splash screen та метадані для коректного відображення на пристроях користувачів.

Розповсюдження APK та IPA файлів здійснюється через веб-сайт проекту або прямі посилання для завантаження. Користувачі Android можуть встановити APK файл безпосередньо, увімкнувши встановлення з невідомих джерел в налаштуваннях безпеки. Для iOS користувачів процес більш складний і може вимагати додаткових налаштувань довіри до сертифіката розробника.

Висновки до розділу 2

У другому розділі проведено аналіз інструментів для розробки та розгортання мобільного додатку «Код-детектив». Було визначено, що Cursor.ai та PyCharm є ефективними середовищами розробки для Flutter та Django відповідно, забезпечуючи високу продуктивність завдяки штучному інтелекту та глибокій інтеграції з відповідними технологіями. Архітектура клієнтської частини на основі принципів чистої архітектури з BLoC паттерном забезпечує тестованість та підтримуваність коду, а модульна система залежностей спрощує розширення функціональності. Серверна частина на Django REST Framework надає надійну та масштабовану основу для API з токен-базованою аутентифікацією та інтеграцією ШІ сервісів. Для розгортання проєкту було обрано Render як хмарну платформу для бекенду та підготовку мобільного додатку у вигляді APK/IPA файлів, що забезпечує доступність додатку для широкої аудиторії користувачів Android та iOS пристроїв.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ТА ПРОЕКТУВАННЯ СИСТЕМИ

У цьому розділі детально розглядаються технічні аспекти реалізації мобільного додатку «Код-детектив». Основна увага приділяється аналізу Django та Django REST Framework як основи серверної частини, Flutter з мовою Dart для створення кросплатформенного мобільного інтерфейсу, а також SQLite[5] як системи управління базою даних. Розглянуто інтеграцію зовнішніх сервісів штучного інтелекту для реалізації навчального асистента, токен-базовану систему аутентифікації та використання BLoC паттерна для управління станом додатку.

3.1. Розробка серверної частини застосунку

Серверна частина проекту «Код-детектив» була спроектована як комплексна система для обслуговування мобільного додатку з підтримкою різноманітних типів навчальних завдань та інтелектуальної допомоги.

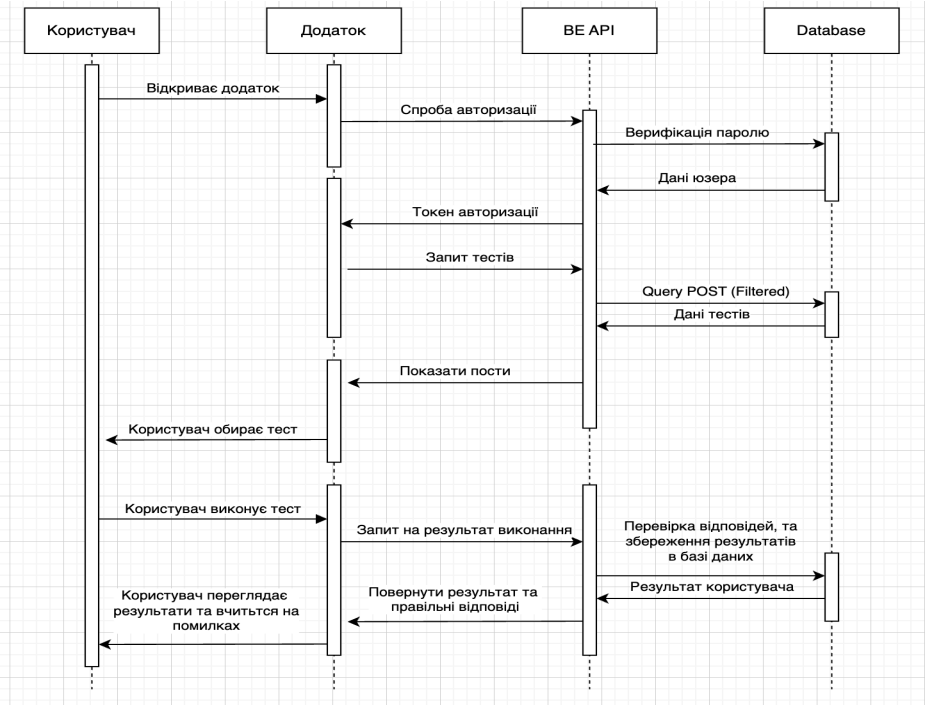


Рисунок 3.1 – Архітектура системи web sequence

Основним викликом було створення гнучкої архітектури, яка могла б підтримувати різні типи питань з програмування, забезпечувати справедливе оцінювання відповідей та інтегруватися з системою штучного інтелекту для персоналізованої навчальної підтримки.

3.1.1 Django framework та Django REST Framework

Django було обрано як основний фреймворк завдяки його потужним можливостям для швидкої розробки та вбудованим засобам безпеки. Фреймворк реалізує архітектурний паттерн Model-View-Template, який забезпечує чітке розділення між рівнем даних, обробкою запитів та відображенням результатів. У проекті «Код-детектив» Django використовується не просто як веб-фреймворк, а як основа для складної освітньої системи з унікальними вимогами до обробки різнотипних завдань.

Ключовим архітектурним рішенням стало створення єдиного Django додатку quiz, який інкапсулює всю бізнес-логіку системи. Це рішення забезпечило тісну інтеграцію між моделями даних, представленнями та бізнес-логікою, спростивши розробку та підтримку коду. Структура додатку включає models.py з визначеннями всіх моделей даних, views.py з логікою обробки HTTP запитів, serializers.py для трансформації даних між Python об'єктами та JSON, та urls.py для конфігурації маршрутизації.

Django REST Framework був адаптований для специфічних потреб мобільного додатку. Особливістю реалізації стало поєднання ViewSet класів для стандартних операцій з кастомними APIView для складної бізнес-логіки. ViewSet підхід використовується для LanguageViewSet та TestViewSet, які надають стандартні CRUD операції з автоматичною генерацією URL маршрутів через DefaultRouter.

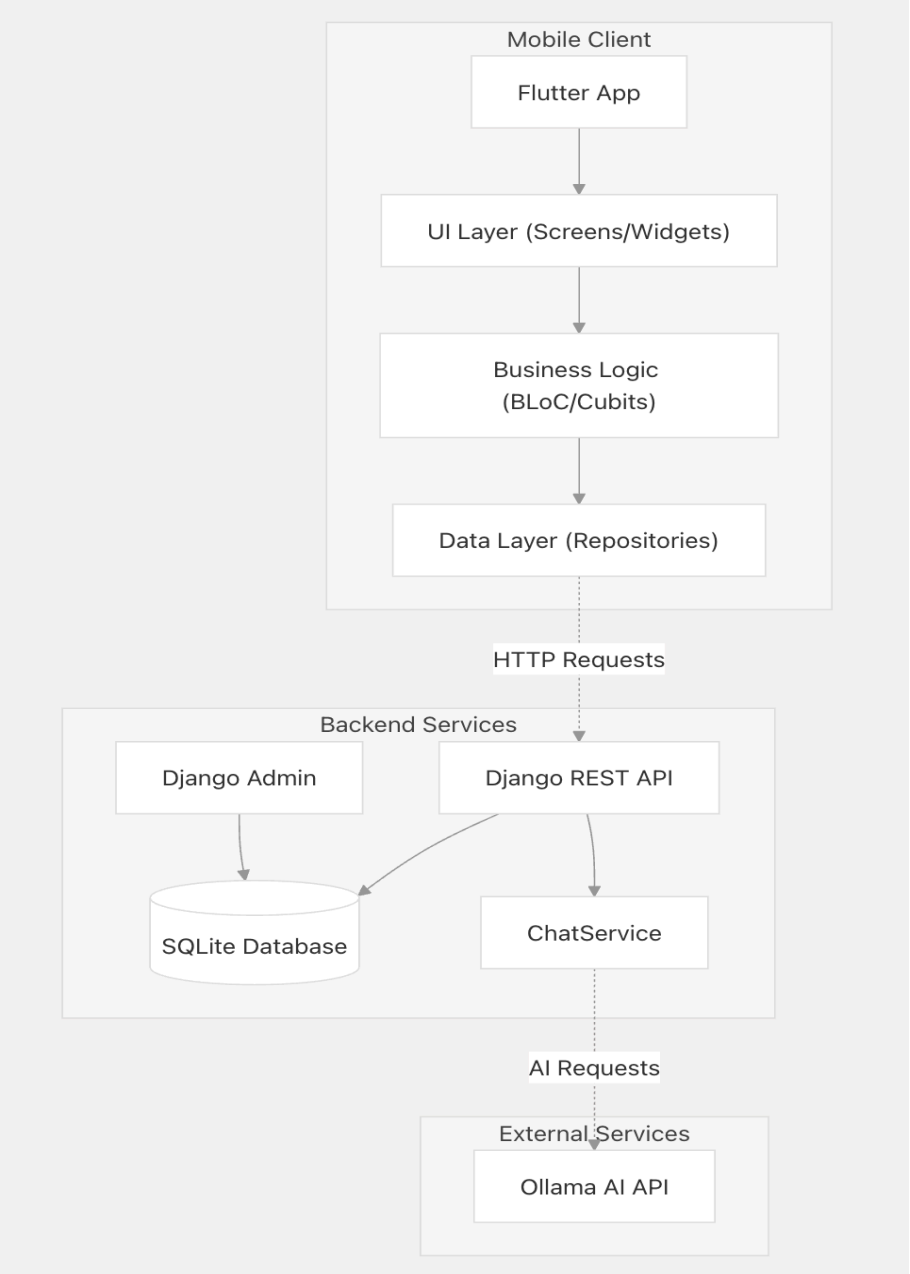


Рисунок 3.2 – Поєднання Flutter та Django

TestViewSet реалізує інтелектуальну фільтрацію через перевизначення методу get_queryset. Цей метод аналізує параметри запиту для фільтрації за мовою програмування (language_id або language_code) та для аутентифікованих користувачів автоматично виключає вже пройдені тести. Логіка фільтрації включає перевірку записів у таблиці TakenTest для поточного користувача та виключення відповідних тестів з результуючого набору даних.

Для складних операцій використовуються кастомні APIView класи. SubmitAnswersView обробляє подання відповідей користувачів з комплексною

логікою валідації, оцінювання та оновлення прогресу. QuestionAIAssistantView інтегрується з системою штучного інтелекту, підтримуючи як контекстну допомогу для конкретних питань, так і загальні консультації.

Серіалізація даних була спроектована з урахуванням складної структури питань з JSON метаданими. Створено спеціальні серіалізатори, які обробляють п'ять різних типів питань, кожен з унікальною структурою даних. QuestionSerializer використовує computed fields для додавання контекстної інформації, такої як назва мови програмування з пов'язаної моделі Language.

SubmitAnswerSerializer представляє найскладнішу частину серіалізації, валідуючи масиви відповідей з різнотипними даними. Серіалізатор включає вкладений AnswerItem клас, який обробляє окремі відповіді з полями question_id та answer. Валідація забезпечує перевірку існування питань, відповідність типів даних та цілісність структури відповідей.

Система аутентифікації реалізована через TokenAuthentication з кастомними представленнями. CustomAuthToken розширює стандартний ObtainAuthToken для повернення додаткової інформації про користувача разом з токеном. RegisterView обробляє реєстрацію нових користувачів з автоматичним створенням пов'язаних UserProfile записів.

Управління дозволами здійснюється через декоратори permission_classes на рівні представлень. Публічні endpoint'и, такі як отримання списку мов програмування та таблиці лідерів, доступні без аутентифікації. Захищені операції, включаючи подання відповідей та доступ до ШІ асистента, вимагають валідного токена через IsAuthenticated клас дозволів.

3.1.2 ШІ асистент та модель Llama 3 8B

Інтеграція штучного інтелекту стала одним з найскладніших та найцікавіших аспектів проекту. Для реалізації ШІ асистента було обрано модель Llama 3 8B, яка забезпечує високоякісні відповіді при оптимальному споживанні

ресурсів. Llama 3 8B представляє собою авторегресивну мовну модель з трансформерною архітектурою, що включає 8 мільярдів параметрів та контекстне вікно у 8000 токенів.

Модель використовує архітектуру Grouped-Query Attention (GQA), яка оптимізує процес генерації тексту через розподіл уваги між групами запитів. Це дозволяє досягти швидкості генерації 119,6 токенів на секунду при збереженні високої якості відповідей. Для проекту «Код-детектив» ці характеристики критично важливі, оскільки користувачі очікують швидких відповідей під час навчального процесу.

Унікальною особливістю реалізації стало створення персони "Детектив Код" - українськомовного асистента з детективною тематикою. ChatService було спроектовано як центральний компонент інтеграції, який виступає мостом між Django бекендом та зовнішнім Ollama API. Сервіс конфігурується з базовою URL адресою http://localhost:11434 та специфікацією моделі llama3:8b.

ChatService підтримує два режими роботи через методи get_response та get_general_response. Перший метод призначений для контекстної допомоги та приймає як параметри користувацький запит та структуровані дані питання. Другий метод обробляє загальні запити з програмування без специфічного контексту.

Для контекстної допомоги розроблено складний алгоритм збагачення промптів, який аналізує структуру питання та формує детальний контекст. Алгоритм витягує тип питання (single, blank, order, trace, debug), мову програмування, текст завдання та метадані. Для питань типу single система форматує список варіантів відповідей без зазначення правильної опції, що дозволяє ШІ надавати підказки без розкриття рішення.

Особливістю промптинг стратегії стало використання детективної термінології на всіх рівнях взаємодії. Базовий промпт встановлює персону: "Ти - Детектив Код, досвідчений детектив, який спеціалізується на розслідуванні програмних загадок та допомозі юним детективам у вивченні програмування."

Далі система інструктує модель використовувати специфічну термінологію: помилки у коді називати "підозрюваними", процес налагодження - "розслідуванням", а змінні в прикладах мають імена як "підозрюваний", "доказ", "свідок".

Контекстне збагачення промптів включає форматування інформації про питання у детективному стилі. Наприклад, для питання з JavaScript промпт може містити: "Справа стосується JavaScript розслідування. Тип справи: вибір підозрюваного. Деталі справи: [текст питання]. Підозрювані у справі: [список варіантів]." Такий підхід створює унікальну атмосферу, яка відрізняє додаток від традиційних освітніх платформ.

Система обробки помилок також витримана в детективному стилі з трьома рівнями fallback повідомлень. При успішному запиті до API, але відсутності відповіді система повертає: "Справу не розкрито. Мені потрібно більше доказів для вирішення цієї загадки." HTTP помилки генерують повідомлення: "Я натрапив на перешкоду в цьому розслідуванні. Код помилки: {status_code}." Технічні винятки обробляються повідомленням: "Моє розслідування було перервано. Технічні труднощі: {error}."

Інтеграція з Django представленнями реалізована через QuestionAIAssistantView, який обробляє два типи запитів. POST запити до /questions/{question_id}/ask/ включають контекст конкретного питання, тоді як запити до /ask/ обробляються як загальні консультації. View використовує get_object_or_404 для безпечного отримання питання та формує структуровані дані для передачі ChatService.

Асинхронна природа взаємодії з Ollama API вимагала особливої уваги до обробки таймаутів та помилок мережі. ChatService включає механізми retry та graceful degradation, що забезпечує стабільну роботу навіть при тимчасових проблемах з ШІ сервісом. Логування всіх взаємодій дозволяє відстежувати якість відповідей та оптимізувати промптинг стратегії.

3.1.3 Система керування базами даних SQLite

SQLite було обрано як оптимальне рішення для проекту завдяки простоті розгортання та достатній функціональності для освітнього додатку. SQLite являє собою самодостатню, безсерверну, нульової конфігурації, транзакційну SQL базу даних. На відміну від традиційних РСУБД, SQLite не є окремим серверним процесом, а вбудовується прямо в додаток, зберігаючи всю базу даних у одному файлі на диску.

Основним викликом стало проектування схеми бази даних, яка могла б ефективно зберігати різнотипні питання з програмування та детально відстежувати прогрес користувачів. SQLite підтримує повний стандарт SQL-92, включаючи складні запити з JOIN операціями, підзапити, індекси, тригери та представлення. Система забезпечує ACID властивості (Atomicity, Consistency, Isolation, Durability) через журналювання та блокування на рівні бази даних.

Ключовим архітектурним рішенням стало використання JSON полів для зберігання метаданих питань. SQLite підтримує JSON як нативний тип даних з JSON1 розширенням, що дозволяє виконувати запити безпосередньо до JSON структур. Це рішення забезпечило створення гнучкої системи, яка підтримує п'ять різних типів питань без дублювання таблиць або складних наслідувань.

Таблиця Language представляє базову структуру з полями id (INTEGER PRIMARY KEY), name (TEXT), code (TEXT UNIQUE) та icon (TEXT). Поле code використовується для програмного доступу та має унікальне обмеження для запобігання дублювання мов програмування. Django ORM автоматично генерує індекси для первинного ключа та унікальних полів.

Таблиця Test пов'язана з Language через зовнішній ключ language_id з CASCADE поведінкою при видаленні. Структура включає поля id, title, difficulty та language_id. Поле difficulty використовує Django choices для обмеження значень до 'easy', 'medium', 'hard', що забезпечує цілісність даних на рівні застосунку та бази даних.

Таблиця Question стала центральним елементом системи, поєднуючи універсальні поля з гнучкими JSON структурами. Основні поля включають id, test_id (FK), type, prompt (TEXT), hint (TEXT), clue (TEXT), metadata (JSON) та correct_answer (JSON). Поле type використовує enum значення: 'single', 'blank', 'order', 'trace', 'debug'.

Кожен тип питання має унікальну структуру метаданих, що демонструє гнучкість JSON підходу. Питання типу 'single' зберігають список варіантів у форматі {"options": [{"id": "a", "text": "варіант відповіді"}]}, а правильна відповідь має формат {"id": "b"}. Питання 'blank' не потребують складних метаданих - {} та зберігають правильну відповідь як {"text": "правильна відповідь"}.

Питання типу 'order' для впорядкування коду використовують структуру {"options": [{"id": 1, "text": "фрагмент коду"}]} з правильною відповіддю {"order": [1,2,3,4]}. Це дозволяє створювати завдання на логічне впорядкування рядків коду або алгоритмічних кроків.

Найскладнішими є питання типів 'trace' та 'debug'. Питання 'trace' включають код та змінні для відстеження: {"code": "програмний код", "variables": ["var1", "var2"]}, а правильна відповідь містить очікуваний результат: {"output": "результат виконання"}. Питання 'debug' зберігають код з помилками у метаданих {"broken_code": "код з помилкою"} та виправлену версію як відповідь {"fixed_code": "виправлений код"}.

Для відстеження навчального прогресу було спроектовано детальну систему аналітики через таблиці UserProfile, UserProgress та TakenTest. UserProfile розширює стандартну Django User модель через зовнішній ключ з CASCADE поведінкою, додаючи поле score для швидкого доступу до загального рахунку користувача без складних агрегацій.

Таблиця UserProgress представляє найдетальнішу аналітику взаємодії користувачів з питаннями. Структура включає user_id, question_id, is_correct (BOOLEAN), attempts (INTEGER), used_hint (BOOLEAN) та timestamp

(DATETIME). Композитний індекс на (user_id, question_id) забезпечує швидкий доступ до історії відповідей конкретного користувача.

TakenTest фіксує завершені тести з полями user_id, test_id, score_obtained та completed_at. Ця таблиця використовується для фільтрації вже пройдених тестів у TestViewSet та генерації таблиці лідерів. Індекс на completed_at дозволяє ефективно сортувати результати за часом.

Особливістю системи є динамічне нарахування балів з урахуванням складності питань. Розроблено алгоритм у SubmitAnswersView, який нараховує 1 бал за легкі питання, 3 бали за середні та 5 балів за складні. Логіка реалізована через Django ORM запити з агрегацією результатів та атомарним оновленням рахунку користувача.

Система також включає автоматизовані Django management команди для наповнення бази даних. Команда populate_more_questions демонструє програмне створення тестового контенту з використанням get_or_create методів Django ORM. Команда створює структуровані тести для JavaScript та Python з різними рівнями складності, включаючи всі п'ять типів питань.

Міграційна система Django забезпечує версіювання схеми бази даних через автоматично згенеровані файли міграцій. Кожна зміна в models.py генерує відповідну міграцію з SQL командами для оновлення структури. SQLite підтримує більшість DDL операцій, хоча деякі зміни (як видалення колонок) вимагають повного перестворення таблиці.

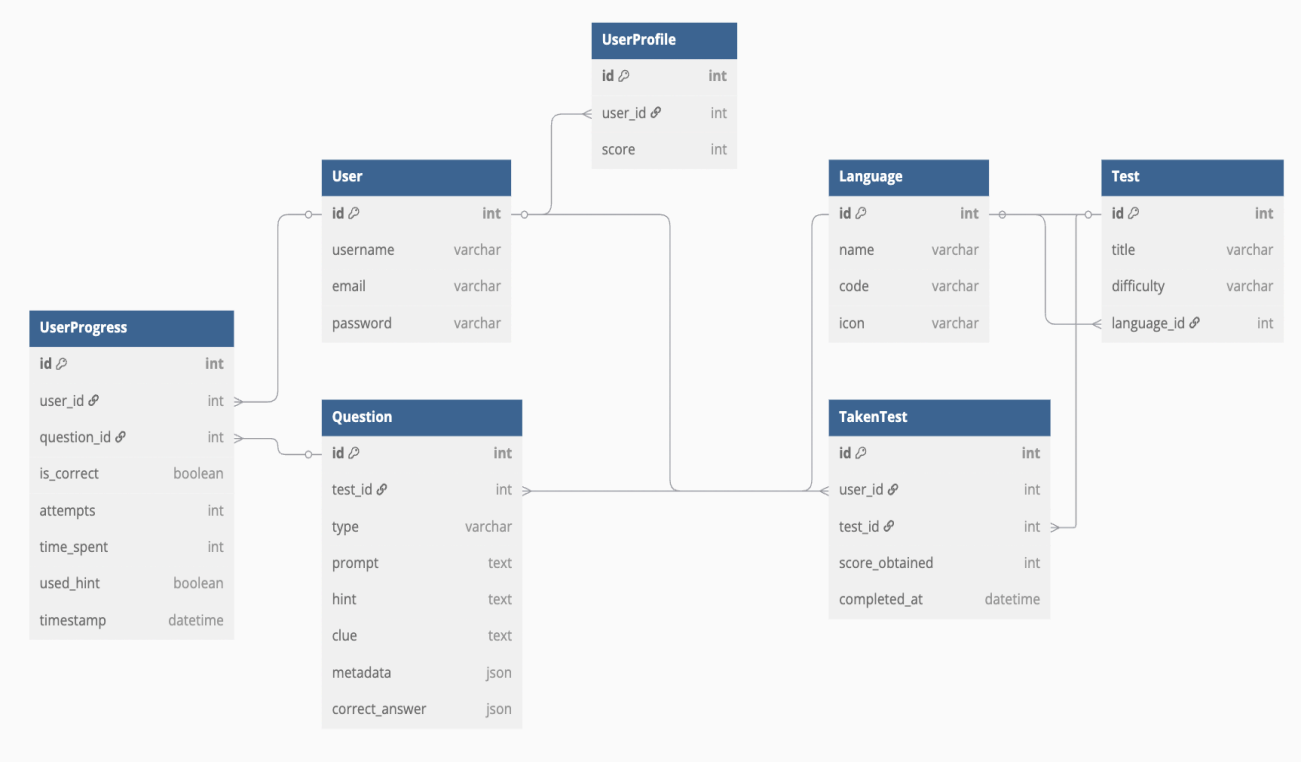


Рисунок 3.3 – Схема реляційної бази даних

Усі зв'язки між таблицями реалізовано через зовнішні ключі, що забезпечує цілісність даних та ефективність запитів. Django ORM автоматично обробляє ці зв'язки, дозволяючи розробникам працювати з даними на високому рівні абстракції. Система індексування SQLite забезпечує швидкий доступ до даних навіть при зростанні обсягу інформації в базі даних.

3.2 Розробка клієнтської частини

Клієнтська частина мобільного додатку являє собою інтерфейс, з яким користувачі безпосередньо взаємодіють. Сюди входить візуальний дизайн, навігація між екранами, анімації, обробка користувацького вводу та відображення даних. Клієнтська частина відповідає за створення зручного та інтуїтивного досвіду користування додатком, забезпечуючи швидке реагування на дії користувача та ефективне відображення інформації. Для розробки клієнтської частини проекту «Код-детектив» було обрано Flutter фреймворк з використанням

паттерна BLoC для управління станом та інтеграцією штучного інтелекту для навчальної підтримки.

3.2.1 Flutter фреймворк для кросплатформенної розробки

Flutter було обрано як основний фреймворк завдяки його унікальній архітектурі рендерингу та можливості створення нативно-продуктивних додатків для декількох платформ з єдиної кодової бази. У проекті застосовано принципи clean architecture з чітким розділенням на data, domain та presentation рівні через систему dependency injection на базі GetIt service locator.

У проекті «Код-детектив» архітектура побудована навколо системи віджетів з чітким розділенням на StatelessWidget для статичних компонентів та StatefulWidget для динамічних елементів інтерфейсу. Головний екран реалізовано через HomeScreen віджет, який поєднує AppBar з навігаційним меню, область фільтрації мов програмування, секцію доступних тестів та таблицю лідерів.

Особливістю реалізації стало використання анімацій входу через пакет flutter_animate з поетапними fade-in та slide ефектами. Кожен елемент головного екрану з'являється з затримкою, створюючи плавний та професійний досвід завантаження. Система pull-to-refresh реалізована через RefreshIndicator для оновлення даних з бекенду без повного перезавантаження екрану.

Для п'яти різних типів питань створено спеціалізовані віджети, які адаптуються до структури JSON метаданих. Віджети для single choice відображають список опцій з радіо-кнопками, blank type надають текстові поля для введення відповіді, order type дозволяють drag-and-drop впорядкування фрагментів коду. Trace та debug типи реалізують більш складні інтерфейси з кодовими блоками та інтерактивними елементами для трасування виконання та виправлення помилок.

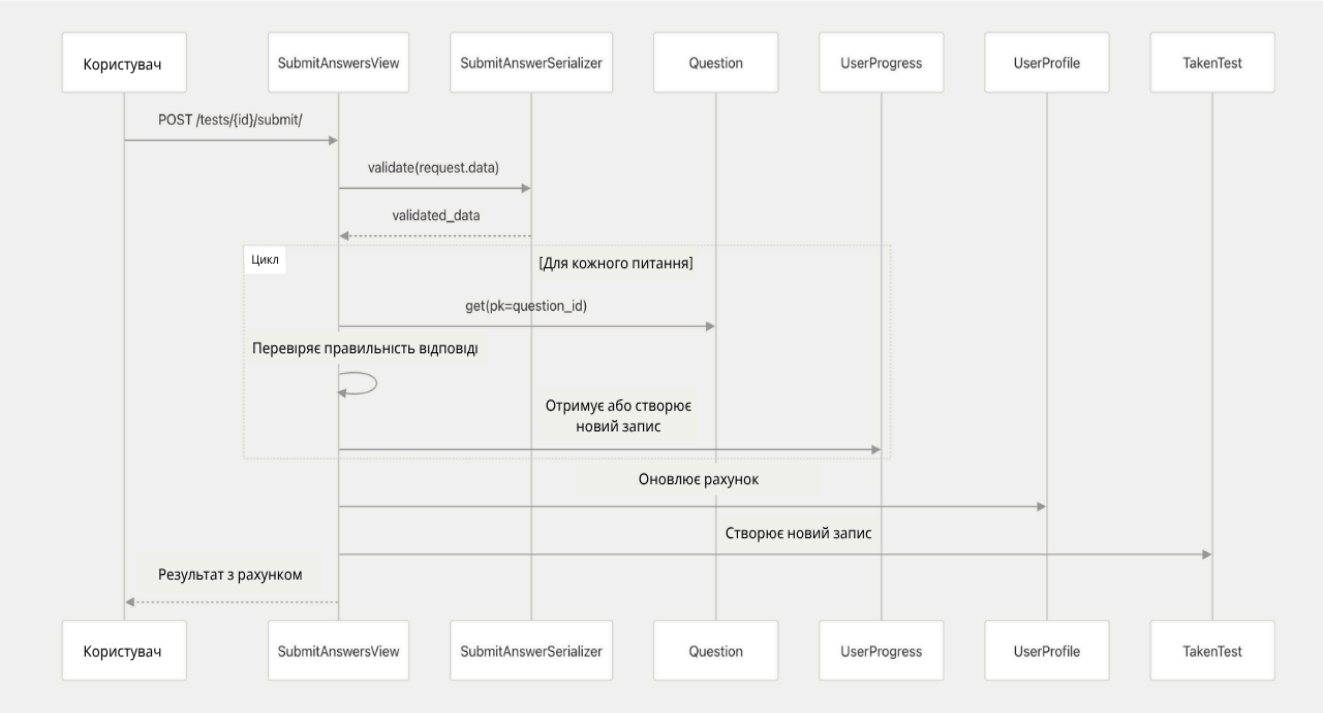


Рисунок 3.4 – Схема виконання тестів користувачем

Навігація між екранами організована через named routes з передачею параметрів через RouteSettings. Система включає екрани аутентифікації (LoginScreen), головний екран (HomeScreen), екран проходження тестів (TestScreen), профіль користувача (SelfProfileScreen) та модальні вікна для ШІ чату через showModalBottomSheet.

Dependency injection реалізовано через InjectionContainer, який поєднує AuthInjector та ChatsInjector mixin'и для організації залежностей за функціональними областями. HTTP клієнт Dio налаштовано з interceptor'ами для автоматичного додавання токенів аутентифікації, логування запитів та обробки помилок.

3.2.2 Управління станом через BLoC паттерн

Управління станом у проекті реалізовано через flutter_bloc бібліотеку з використанням Cubit класів для спрощення архітектури. Система побудована

навколо принципів односпрямованого потоку даних та immutable станів, що забезпечує передбачуваність та тестованість додатку.

HomeCubit керує станом головного екрану з методом getData() для завантаження тестів, мов програмування та статистики користувача. Cubit обробляє складну логіку фільтрації тестів за мовами програмування та автоматично виключає вже пройдені користувачем тести через інтеграцію з HomeRepository.

TestCubit представляє найскладнішу частину системи, реалізуючи повний життєвий цикл проходження тесту. Стани включають TestInitial, TestLoading при завантаженні питань, TestInProgress з поточним питанням та Map користувацьких відповідей, TestCompleted з результатами та TestFailure для помилок.

Унікальною особливістю TestCubit є інтегрований Timer для відстеження часу проходження тесту. Таймер запускається при завантаженні питань та оновлює UI кожну секунду через emit(state.copyWith()) з оновленим часом. Система навігації по питаннях включає методи nextQuestion(), previousQuestion() та jumpToQuestion() з валідацією меж масиву.

Збереження відповідей користувача реалізовано через методи answerQuestion() з різною логікою для кожного типу питання. Single choice зберігає ідентифікатор вибраної опції, blank type - введений текст, order - масив впорядкованих індексів, trace та debug - відповідні структури даних згідно з форматом API.

UserCubit управляє аутентифікацією користувача з інтеграцією SharedPreferences для збереження токенів та автоматичної реаутентифікації при запуску. Система включає методи login(), register(), logout() з відповідними переходами станів та очищенням локальних даних при виході.

Взаємодія між UI та Cubit класами відбувається через BlocBuilder для автоматичної перебудови інтерфейсу при зміні стану, BlocListener для side effects як навігація та показ повідомлень, та BlocConsumer для поєднання обох підходів у складних сценаріях.

3.2.3 Інтеграція штучного інтелекту

Система ШІ асистента реалізована через ієрархічну архітектуру Cubit класів з базовим абстрактним AiChatCubit та двома спеціалізованими реалізаціями для різних режимів використання. Архітектура забезпечує чітке розділення між контекстною допомогою для питань та загальними консультаціями з програмування.

Базовий AiChatCubit визначає чотири стани: AiChatInitial для початкового стану, AiChatLoading з опціональним повідомленням користувача для миттєвого відображення, AiChatSuccess з парою користувацьке повідомлення-відповідь ШІ, та AiChatError для обробки помилок з детальними повідомленнями.

QuestionAiChatCubit розширює базову функціональність властивістю questionId для контекстно-орієнтованих запитів. Метод sendMessage() викликає HomeRepository.askQuestionAi() з передачею ID питання та запиту користувача для отримання контекстної допомоги. Система автоматично очищає історію чату при зміні питання через updateQuestionId() для запобігання плутанині між різними завданнями.

GeneralAiChatCubit забезпечує загальну підтримку через HomeRepository.askGeneralAi() без прив'язки до конкретних завдань. Цей режим використовується для теоретичних питань, пояснень концепцій програмування та загальних рекомендацій поза контекстом тестування.

Користувацький інтерфейс ШІ реалізовано через AiChatBottomSheet віджет, який відображається як модальне вікно висотою 90% екрану. Компонент включає заголовок з іконкою SmartToy та кнопкою закриття, ScrollView область для історії повідомлень з автоматичним прокручуванням до останнього повідомлення, та секцію вводу з TextFormField та FloatingActionButton.

Система рендерингу повідомлень диференціює типи контенту: користувацькі повідомлення відображаються як фіолетові Align(alignment: Alignment.centerRight) контейнери, тоді як відповіді ШІ показуються в темних

лівосторонніх бульбашках з підтримкою MarkdownBody для форматування коду, заголовків та списків.

Стан-базоване відображення UI адаптується до поточного AiChatState: початковий стан показує привітальне повідомлення "Питай мене, що тебе цікавить, юний детективе!", стан завантаження відображає користувацьке повідомлення з CircularProgressIndicator, успішний стан показує повну історію чату, а стан помилки надає можливість повторної спроби.

Інтеграція з repository layer забезпечує уніфіковану обробку помилок через Either<Failure, String> функціональні типи. Repository повертає результати або об'єкти помилок, Cubit трансформує Failure в AiChatError стани з користувацькими повідомленнями, а UI відображає зрозумілі повідомлення про помилки з кнопками повтору запиту.

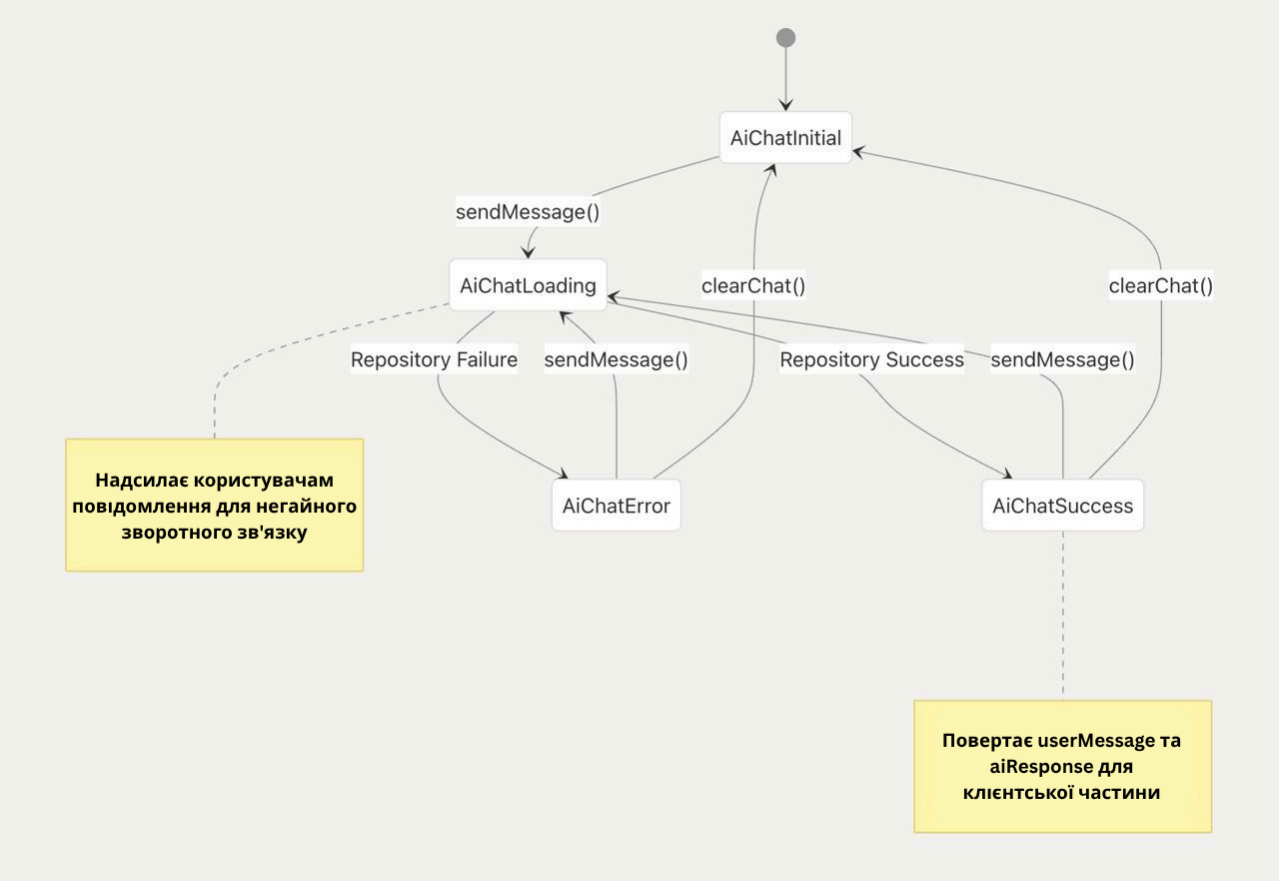


Рисунок 3.5 – Архітектура станів ШІ асистенту

Висновки до розділу 3

У третьому розділі було детально описано процес розробки програмної системи мобільного додатку «Код-детектив», зокрема її серверної та клієнтської частин. Для серверної частини було використано Django framework разом з Django REST Framework для створення ефективного та масштабованого API. Система керування базами даних SQLite забезпечує надійне зберігання навчального контенту, користувацьких даних та прогресу навчання з підтримкою різних типів питань та гнучкою JSON структурою метаданих. На клієнтській частині застосунку використано Flutter фреймворк для створення кросплатформенного мобільного додатку з єдиною кодовою базою. BLoC паттерн забезпечує структуроване управління станом та чітке розділення бізнес-логіки від презентаційного рівня. Інтеграція штучного інтелекту через спеціалізовані Cubit компоненти надає користувачам контекстну допомогу під час проходження тестів та загальну підтримку з питань програмування, створюючи інноваційний навчальний досвід.

4 РОБОТА КОРИСТУВАЧА З СИСТЕМОЮ

Цей розділ детально описує, як користувачі можуть взаємодіяти з системою через її інтерфейс.

4.1 Сторінки авторизації

Після того як користувач завантажить застосунок та відкриє його вперше, він потрапляє на сторінку реєстрації, звідки можна потрапити напряму до сторінки логіну, якщо акаунт вже існує.

4.1.1 Сторінка «Логін»

На цій сторінці користувачу потрібно ввести логін та пароль від свого акаунту.

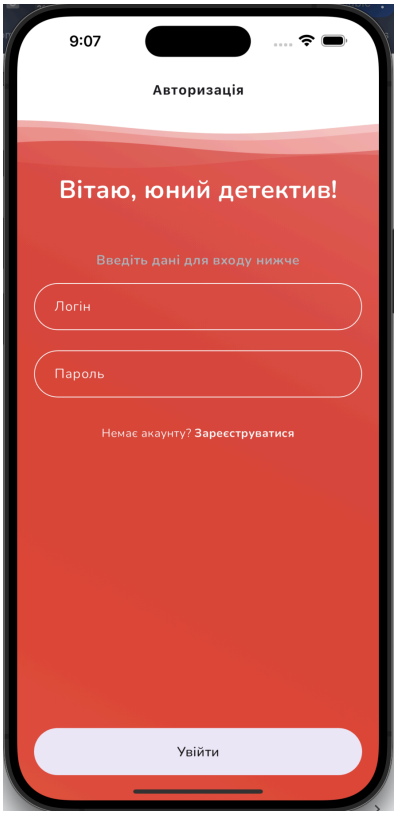


Рисунок 4.1 – Сторінка логіну

4.1.1 Сторінка «Авторизація»

На цій сторінці користувачу потрібно ввести логін та пароль від свого акаунту.

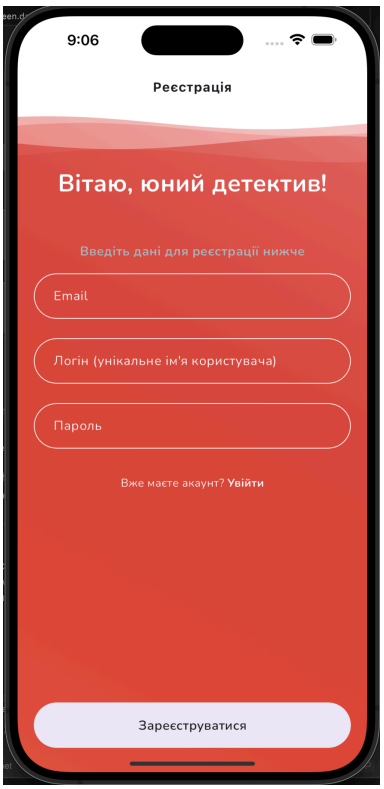


Рисунок 4.2 – Сторінка реєстрації

4.2 Головна сторінка

Після успішної авторизації користувач потрапляє на головну сторінку сторінку де відбуваються основні інтеракції користувача з додатком.

4.2.1 Секція «Тести»

Одразу його зустрічає приємна анімація після якої першою з’являється секція вибору тестів зі зручним сортуванням. Звідси користувач може переглянути усі тести відсортовані за складністю, мовою програмування та темами.

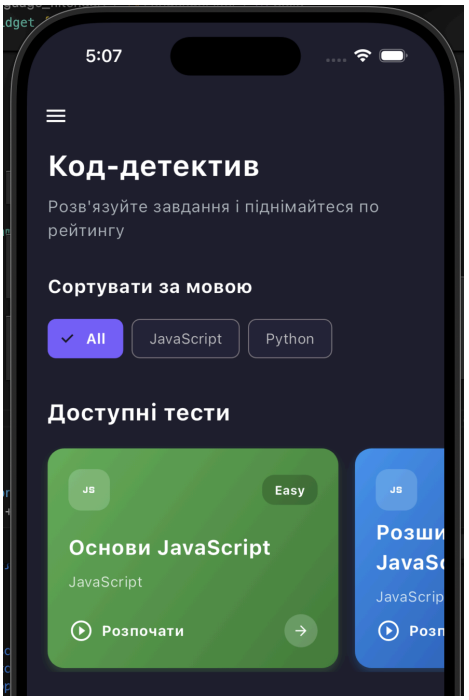


Рисунок 4.3 – Секція «Тести»

4.2.2 Секція «Список лідерів»

В секції списку лідерів користувач може переглянути поточний список лідерів та кількість балів кожного з них.

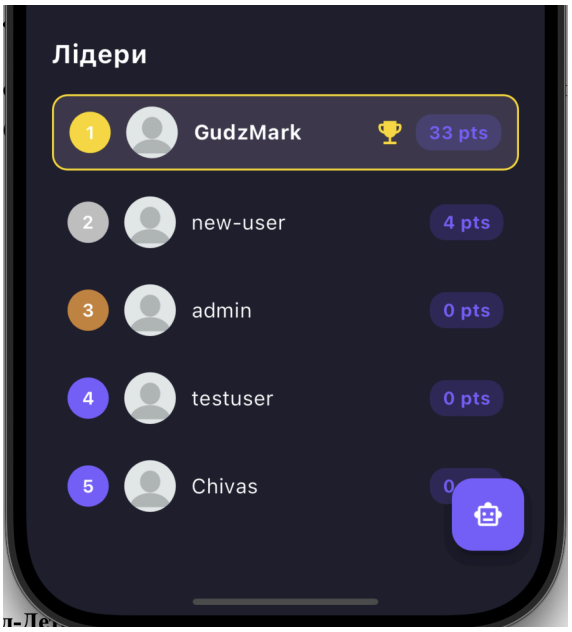


Рисунок 4.4 – Секція «Список лідерів»

4.2.3 Чат «Код-Детектив»

Натиснувши на кнопку в правому нижньому куті екрану (яка завжди знаходиться там, незалежно від позиції скролу користувача) юзер відкриває модальне вікно де може спілкуватися з інтегрованим штучним інтелектом на будь-яку тему. ШІ буде знати контекст, що включає рівень знань користувача (основуючись на пройдених їм тастах), його персональні дані та безпосередньо запитання, що хвилює.

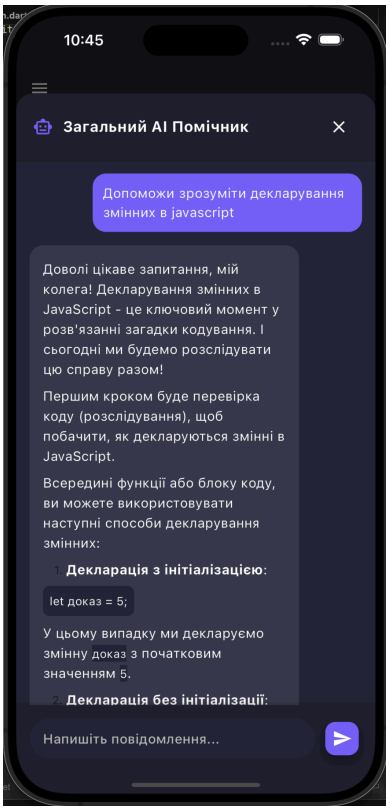


Рисунок 4.5 – Чат «Код-Детектив»

4.3 Сторінка «Профіль»

На сторінці профілю користувач може переглянути пройдені їм тести, а також свій загальний рахунок. Додатково надається можливість змінити його логін, але в цьому випадку потрібно буде використовувати новий логін для входу в акаунт в майбутньому.

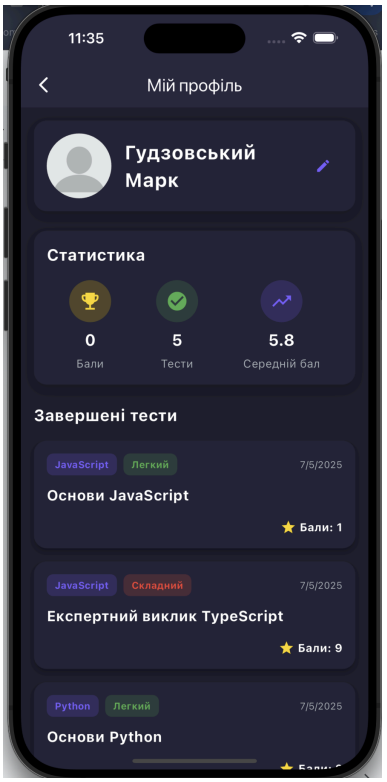


Рисунок 4.6 – Сторінка «Профіль»

Так само виглядатимуть сторінки інших користувачів, на які можна потрапити натиснувши на иконку юзера в таблиці лідерів. Проте, очевидно, що на сторінці іншого користувача буде відсутній функціонал редагування логіну.

4.4 Сторінка «Тестування»

Головним функціоналом застосунку є самі тести, та їх зручність. Після натискання на картку обраного тесту користувач потрапляє на екран тестування, де автоматично починається відлік, так як ми націлені на якість і швидкість вивчення програмування. Звідси він також може викликати ШІ-асистента та запитати його про поточне запитання. Штучний інтелект сам зрозуміє де ти наразі знаходишся, отримає доступ до правильної відповіді з бази даних та використає всі необхідні мета дані про питання, щоб допомогти користувачу знайти правильну відповідь і, найголовніше, вивчити щось нове. Кожен тест складається

з 5 типів завдань: з вибором відповіді, з пропущеним словом, розуміння правильного виходу коду, розставлення порядку для правильного виконання функції та виправлення помилок в коді.

4.4.1 Запитання з вибором відповіді

Першим питанням в кожному тесті завжди йде вибір однієї відповіді.

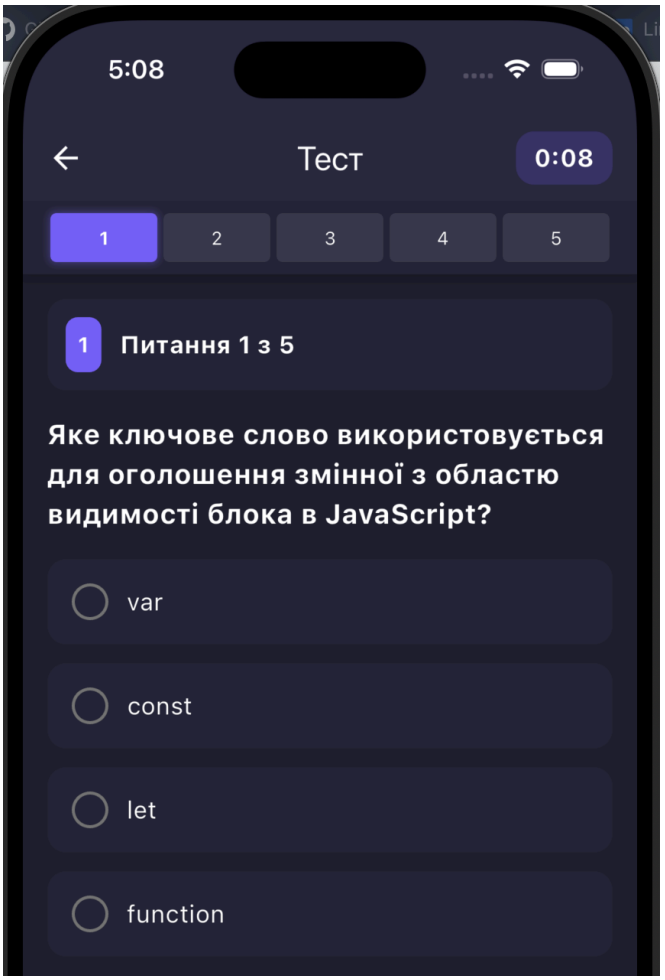


Рисунок 4.7 – Запитання з вибором відповіді

4.4.2 Запитання з пропущеним словом

Далі користувачу необхідно заповнити пропуск у визначенні якогось терміну, щоб відтворити його істиність.

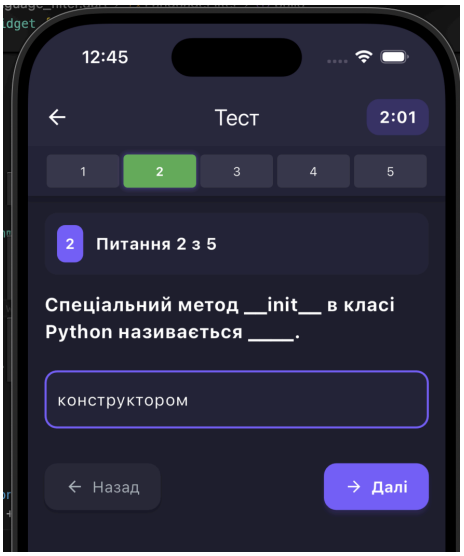


Рисунок 4.8 – Запитання з пропущеним словом

4.4.3 Запитання з переставленням порядку рядків коду

На цьому запитанні користувач має переставити місцями рядки коду так, щоб він працював коректно.

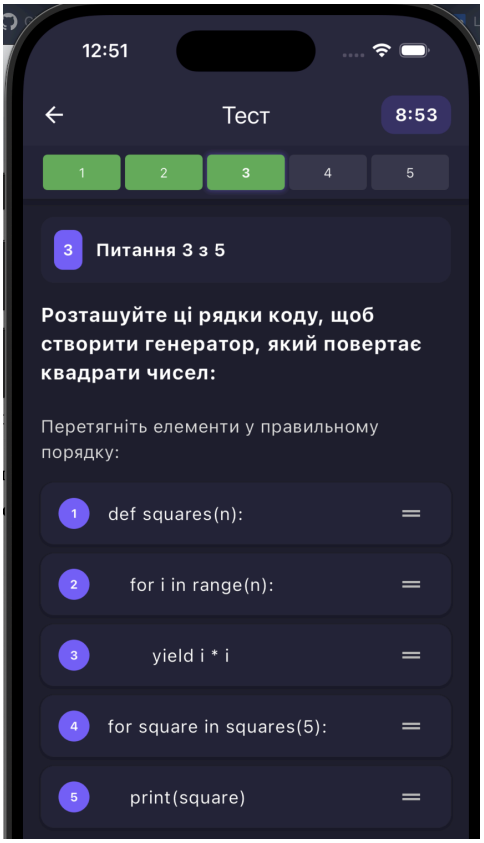


Рисунок 4.9 – Запитання з переставленням порядку рядків коду

4.4.4 Запитання «Trace»

Тут вже стає цікавіше — потрібно проаналізувати код і зрозуміти, яким буде його результат відпрацювання.

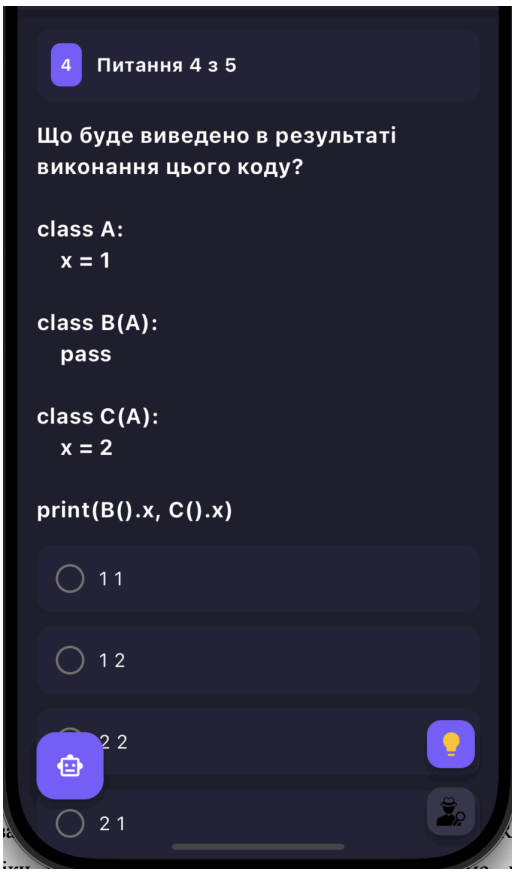


Рисунок 4.10 – Запитання «Trace»

4.4.5 Запитання «Знайди помилку»

Наприкінці користувача чекає запитання, де він має відредагувати код, щоб він міг відпрацювати коректно. Цей тип завдання був розроблений завдяки інтеграції бібліотеки flutter_code_editor, що реалізує дуже зручний редактор коду, з підтримкою більш ніж 40 мов програмування, автозаповненням а також дуже зручним інтерфейсом для мобільних пристроїв.

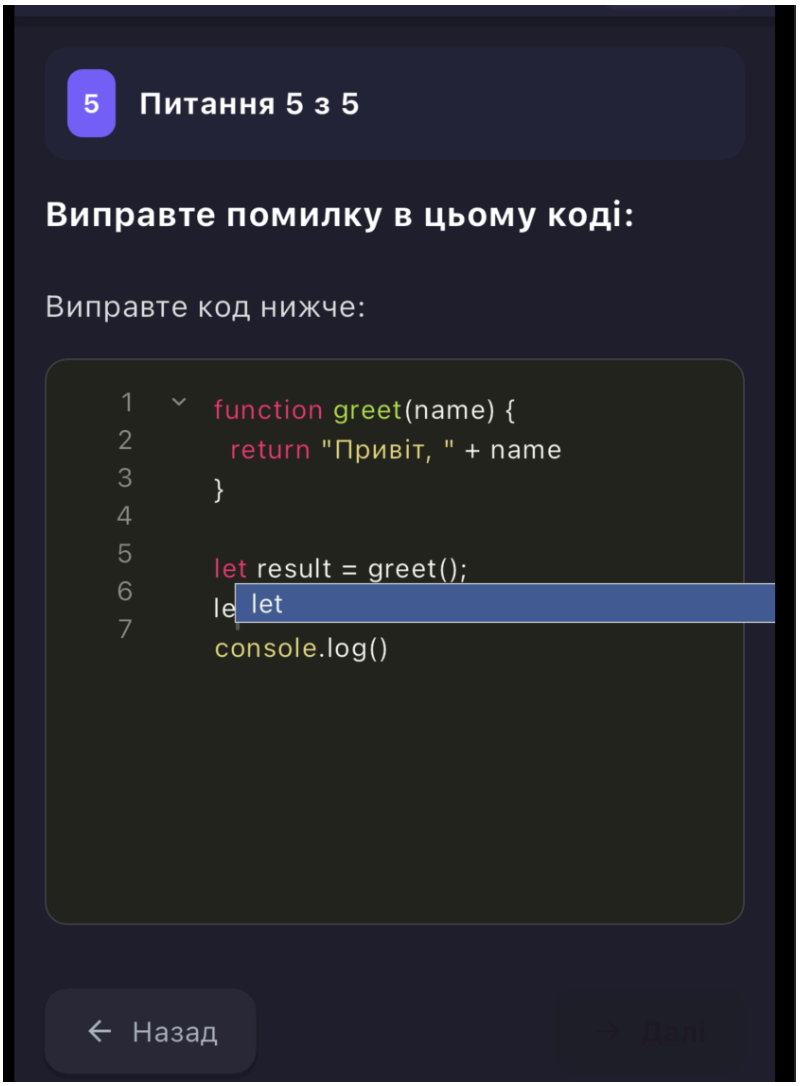


Рисунок 4.11 – Запитання «Знайди помилку»

4.4.6 Підказка від Код-Детектива

Якщо у користувача виникають складності з якимось запитанням, він має змогу використати підказку від Код-Детектива. Натиснувши на відповідну кнопку він побачить модальне вікно з підказкою, що натякає на правильний варіант відповіді або напрямок, в якому слід рухатись для розгадки логічного завдання. Використання підказки також впливає на його результат в кінці тесту. Без підказок буде більше балів.

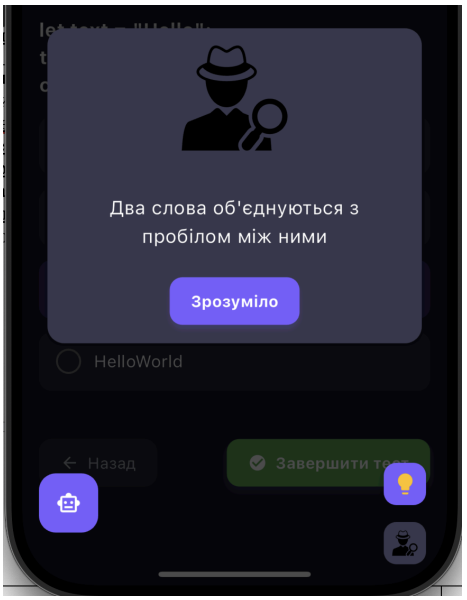


Рисунок 4.12 – Підказка від Код-Детектива

4.4.7 Результат тестування

Після проходження тесту користувач потрапляє на сторінку результату, де він може переглянути відсоток правильних відповідей, отримані бали а також правильні відповіді на запитання, де була допущена помилка.

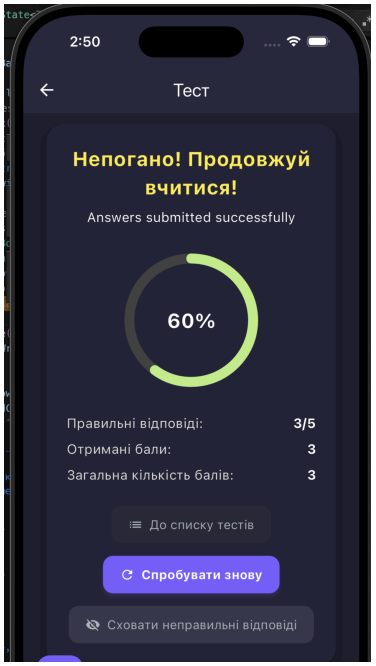


Рисунок 4.13 – Результат тестування

4.5 Панель адміністратора

Так як бекенд застосунку був розроблений за допомогою Django, було вирішено розробити CRM-систему використовуючи Django Web Framework. Користувач, в даному випадку адміністратор, за умови вже існуючого або прествореного для нього акаунту, може зайти в панель адміністратора через браузер. Звідти можна налаштовувати усі типи даних, додавати та видаляти. Також можна відновлювати пароль та інші персональні дані для користувачів, якщо вони загубили щось.

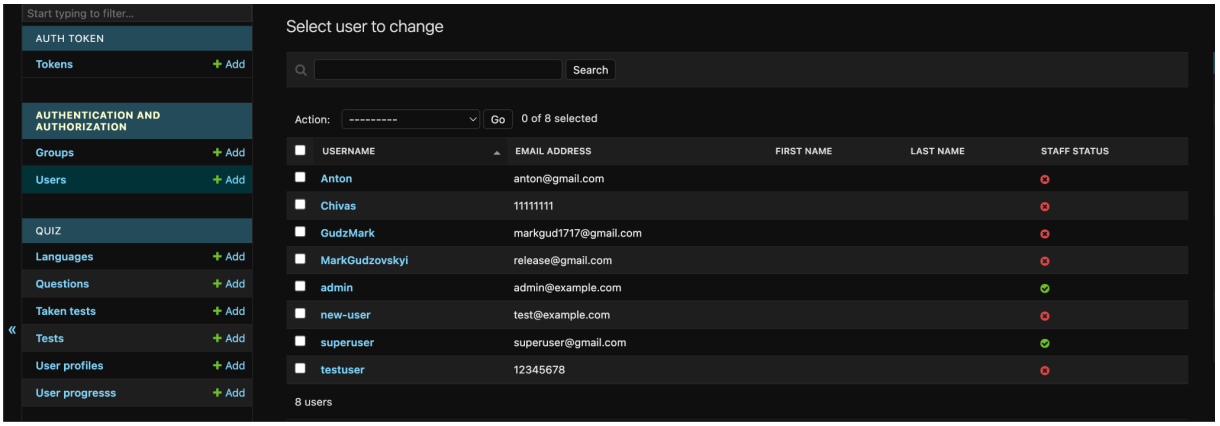


Рисунок 4.14 – Панель адміністратора

Висновки до розділу 4

У четвертому розділі розглянута взаємодія користувача з системою на різних етапах. Головна сторінка забезпечує загальний огляд усіх тестів, сортування, таблиці лідерів, а також доступ до ШІ помічника. Сторінка «Профіль» містить в собі дані про всі тести пройдені користувачем, його статистику а також . Сторінка «Тестування» надає доступ до всіх типів запитань тесту, а також правильні відповіді та статистику наприкінці його виконання. Все це в поєднанні робить систему зручною та закриває недоліки знайдені в аналогічних продуктах, що дуже сильно покращує UX для студентів.

ВИСНОВКИ

У даній роботі було вирішено завдання створення ефективного та інноваційного рішення для вивчення програмування через розробку мобільного додатку «Код-детектив». Проаналізувавши недоліки існуючих платформ для навчання програмуванню, була зосереджена робота на розробці системи, яка не тільки забезпечує зручне проходження тестів з мобільного пристрою, але й надає унікальний досвід навчання через детективну тематику та інтеграцію штучного інтелекту.

Завдяки використанню сучасних технологій мобільної розробки та серверних рішень, створений додаток вирізняється високою доступністю, кросплатформенністю та інтерактивністю. Використання таких технологій як Flutter для мобільного додатку, Django REST Framework для серверної частини, SQLite для зберігання даних та BLoC паттерна для управління станом, дозволяє забезпечити стабільну роботу системи та можливості для майбутнього розширення функціональності. Архітектурні рішення на основі принципів чистої архітектури забезпечують модульність коду, що робить процес підтримки та додавання нових функцій значно простішим і ефективнішим.

Особливою перевагою розробленого рішення є інтеграція штучного інтелекту, що надає студентам персоналізовану підтримку під час навчання. Система ШІ асистента забезпечує як контекстну допомогу для конкретних завдань, так і загальні консультації з питань програмування, що створює унікальний навчальний досвід. Гейміфікація процесу навчання через детективну тематику, систему очок та таблицю лідерів мотивує користувачів до регулярних занять та створює здорову конкуренцію серед студентів.

Кросплатформенність рішення дозволяє студентам навчатися в будь-якому місці та в будь-який час, використовуючи свої мобільні пристрої. Це особливо важливо в сучасному світі, де мобільні технології стають основним засобом доступу до освітніх ресурсів. Додаток підтримує різні типи питань - від вибору

варіантів до трасування коду, що забезпечує всебічну перевірку знань та навичок програмування.

Перспективи подальшого розвитку додатку включають розширення бази питань для додаткових мов програмування, впровадження більш складних алгоритмів оцінювання знань та персоналізації навчального процесу. Можливе також додавання функціональності для викладачів, що дозволить їм створювати власні тести та відстежувати прогрес студентів. Розвиток спільнотних функцій, таких як обговорення завдань та обмін досвідом між користувачами, може значно підвищити ефективність навчального процесу.

Впровадження більш досконалих можливостей штучного інтелекту, включаючи автоматичну генерацію питань на основі аналізу слабких місць користувача та адаптивне налаштування складності завдань, може зробити процес навчання ще більш ефективним та персоналізованим.

Таким чином, «Код-детектив» має значний потенціал стати важливим інструментом для всіх, хто прагне вивчати програмування в інтерактивній та захоплюючій формі. Додаток сприяє підвищенню мотивації до навчання, забезпечує доступність освітніх ресурсів та створює інноваційний підхід до вивчення програмування, що є критично важливим для підготовки кваліфікованих IT фахівців у сучасному світі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Flutter Documentation. [Електронний ресурс] — Режим доступу: https://docs.flutter.dev/ (дата звернення: 09.05.2025).

2. BloC Documentation. [Електронний ресурс] — Режим доступу: https://bloclibrary.dev/ (дата звернення: 12.05.2025).

3. Django Documentation. [Електронний ресурс] — Режим доступу: https://docs.djangoproject.com/en/5.2/ (дата звернення: 04.05.2025).

4. Llama 3b 8. [Електронний ресурс] — Режим доступу: https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct (дата звернення: 02.05.2025).

5. SQLite Documentation. [Електронний ресурс] — Режим доступу: https://www.sqlite.org/docs.html (дата звернення: 10.05.2025).

6. Dart Documentation. [Електронний ресурс] — Режим доступу: https://dart.dev/docs (дата звернення: 11.05.2025).

7. Pycharm. [Електронний ресурс] — Режим доступу: https://www.jetbrains.com/help/pycharm/getting-started.html (дата звернення: 14.05.2025).

8. How Flutter Renders Widgets. [Електронний ресурс] — Режим доступу: https://nimafarzin-pr.medium.com/how-flutter-renders-widgets-a-tale-of-widgets-elements-and-render-objects-2952084d6f09 (дата звернення: 07.05.2025).

9. Understanding Django MVT architecture and view functions. [Електронний ресурс] — Режим доступу: https://medium.com/@CodeMaple/understanding-django-mvt-architecture-and-view-functions-django-full-course-for-beginners-lesson-39c8da093b44 (дата звернення: 16.05.2025).

10. Peter Gostev LinkedIn post. [Електронний ресурс] — Режим доступу: https://www.linkedin.com/posts/peter-gostev_llama-3-8b-is-the-currently-best-in-class-activity-7187190058825752576-1Cvk/ (дата звернення: 08.05.2025).

ДОДАТОК А

Додаток "Код-детектив" для вирішення логічних завдань з програмування

Програмний код

Аркушів 4

2025

```
BlocBuilder<TestCubit, TestState>(
        bloc: testCubit,
        builder: (context, state) {
          if (state is TestLoading) {
            return const Center(
              child: CircularProgressIndicator(
                color: Color(0xFF7B61FF),
              ),
            );
          } else if (state is TestFailure) {
            return Center(
              child: Column(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  Text(
                    'Помилка: ${state.message}',
                    style: const TextStyle(color: Colors.red),
                    textAlign: TextAlign.center,
                  ),
                  const SizedBox(height: 16),
                  CustomTestButton(
                    text: 'Спробувати знову',
                    isPrimary: true,
                    onPressed: () => testCubit.loadTest(widget.testId),
                    icon: Icons.refresh,
                  ),
                ],
              ),
            );
          } else if (state is TestInProgress) {
            return Stack(
              children: [
                Column(
                  children: [
                    TestProgressBar(
                      currentIndex: state.currentQuestionIndex,
                      questions: state.questions,
                      answeredQuestionIds: state.userAnswers.keys.toList(),
                      onQuestionSelected: (index) {
                        testCubit.jumpToQuestion(index);
```

```
  },
),
Expanded(
  child: SingleChildScrollView(
    padding: const EdgeInsets.all(16.0),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Container(
          padding: const EdgeInsets.all(12),
          decoration: BoxDecoration(
            color: const Color(0xFF252538),
            borderRadius: BorderRadius.circular(12),
          ),
          child: Row(
            children: [
              Container(
                padding: const EdgeInsets.all(8),
                decoration: BoxDecoration(
                  color: const Color(0xFF7B61FF),
                  borderRadius: BorderRadius.circular(8),
                ),
                child: Text(
                  '${state.currentQuestionIndex + 1}',
                  style: const TextStyle(
                    color: Colors.white,
                    fontWeight: FontWeight.bold,
                  ),
                ),
              ),
              const SizedBox(width: 12),
              Text(
                'Питання ${state.currentQuestionIndex + 1} з ${state.questions.length}',
                style: const TextStyle(
                  fontSize: 16,
                  fontWeight: FontWeight.bold,
                  color: Colors.white,
                ),
              ),
            ],
```

```
            ),
          ),
          const SizedBox(height: 16),
          _buildQuestionWidget(state.currentQuestion, state.userAnswers),
          const SizedBox(height: 32),
          _buildNavigationButtons(state),
        ],
      ),
    ),
  ),
],
),
if (isClueVisible)
 _buildOverlay(
  state.currentQuestion.clue ?? 'Підказка недоступна',
  const Color(0xFF3A3A4E),
  () => setState(() => isClueVisible = false),
  false,
 ),
if (isHintVisible)
 _buildOverlay(
  state.currentQuestion.hint ?? 'Підказка недоступна',
  const Color(0xFF2A2A3C),
  () => setState(() => isHintVisible = false),
  true,
 ),
Positioned(
 bottom: 16,
 right: 16,
 child: Column(
  children: [
   if (state.currentQuestion.hint != null)
    FloatingActionButton(
     heroTag: 'hint',
     mini: true,
     backgroundColor: const Color(0xFF7B61FF),
     onPressed: () {
      setState(() {
       isHintVisible = true;
       isClueVisible = false;
```

```
                });
                testCubit.useHint();
              },
              child: const Icon(Icons.lightbulb, color: Colors.amber),
            ),
          const SizedBox(height: 8),
          if (state.currentQuestion.clue != null)
            FloatingActionButton(
              heroTag: 'clue',
              mini: true,
              backgroundColor: const Color(0xFF3A3A4E), // Changed to a dark background color
              onPressed: () {
                setState(() {
                  isClueVisible = true;
                  isHintVisible = false;
                });
                testCubit.useHint();
              },
              child: SvgPicture.asset(
                SvgIcons.detective,
                width: 24,
                height: 24,
              ),
            ),
        ],
      ),
    ),
  ],
 );
} else if (state is TestCompleted) {
 return TestCompletionCard(
  result: state.result,
  onRetry: () => testCubit.restartTest(),
  onBackToTests: () => Navigator.of(context).pop(),
  questions: state.questions,
  userAnswers: state.userAnswers,
  homeCubit: homeCubit,
 );
}
```

ДОДАТОК Б

Додаток "Код-детектив" для вирішення логічних завдань з програмування

Презентація

Аркушів 7

2025

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

Додаток "Код-детектив" для вирішення логічних завдань з програмування

виконав: студент групи ТВ-11 Гудзовський Марк Юрійович

керівник: ст. викл. Дацюк О.А.

2025

Актуальність теми

Проблематика: Абітурієнтам може бути важко сконцентруватися на навчанні. Завдяки ігровій формі, наш додаток дозволяє розглянути інші підходи до навчання.

Додаток "Код-детектив" має потенціал впливати та покращувати розвиток нового покоління IT-спеціалістів в Україні.

Постановка задачі

Мета: Розробка кросплатформеного додатку "Код-детектив" для вирішення логічних завдань з програмування

Задачі, які потрібно розв’язати для досягнення мети:

1. Розробити набір логічних завдань, пов'язаних з програмуванням та алгоритмами
2. Створити інтерфейс у стилі детективного розслідування для подачі завдань
3. Реалізувати систему підказок та допомоги для різних рівнів складності
4. Впровадити систему рейтингу та досягнень для мотивації користувачів.

Аналіз предметної області

Два головних рішення на ринку наразі:

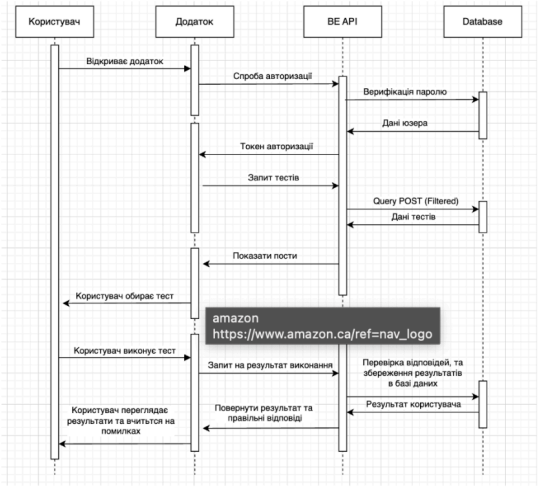
Основні проблеми, які наш додаток вирішує:

- Незручність мобільного використання
- Відсутність мотивації через нецікаву тематику

Архітектура системи

Архітектура системи

Проектування системи



Діаграма web sequence

База даних

Схема реляційної бази даних

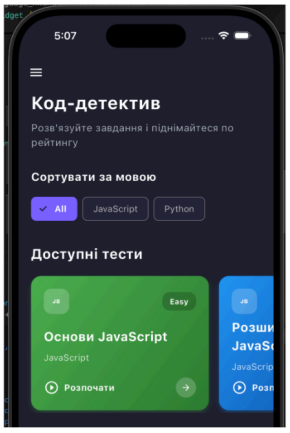
Стек технологій та сервісів розробки

Flutter

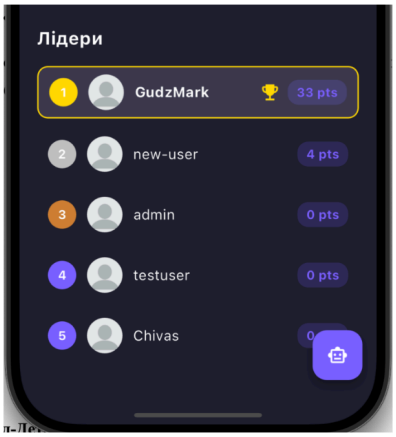
django

SQLite

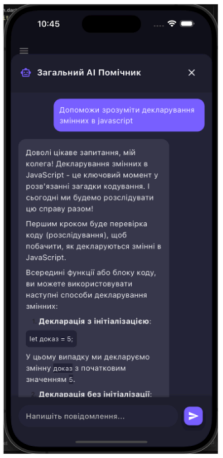
Головна сторінка та ШІ асистент



Секція тестів та стортування

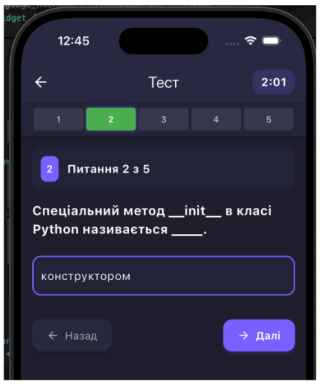


Секція таблиці лідерів

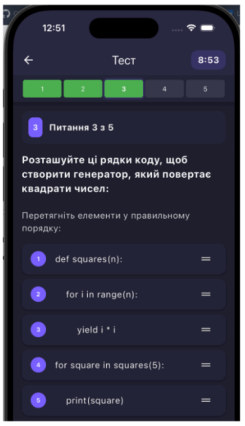


Чат асистент “Код-Детектив”

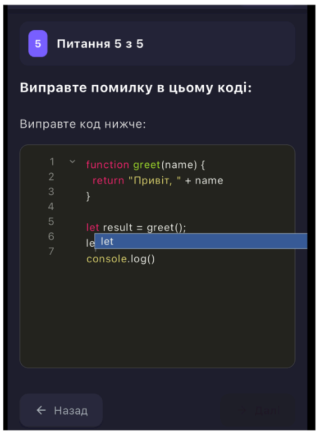
Сторінка тестування



Запитання з пропущеним словом

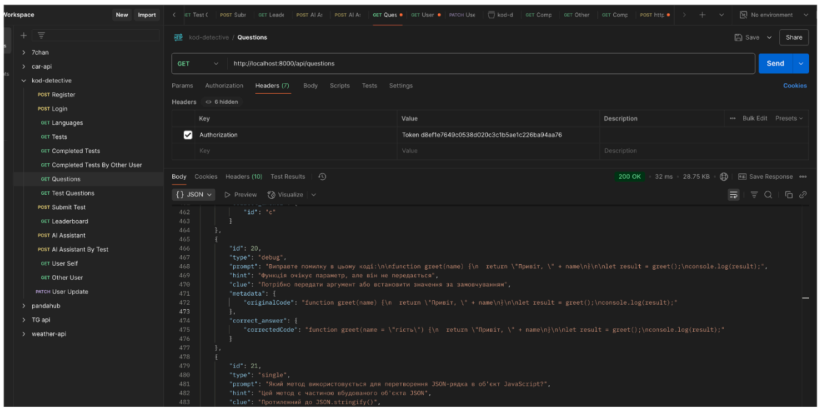


Запитання з переставленням порядку рядків коду

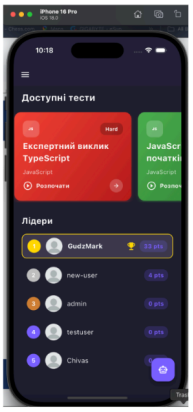


Запитання «Знайди помилку»

Тестування системи

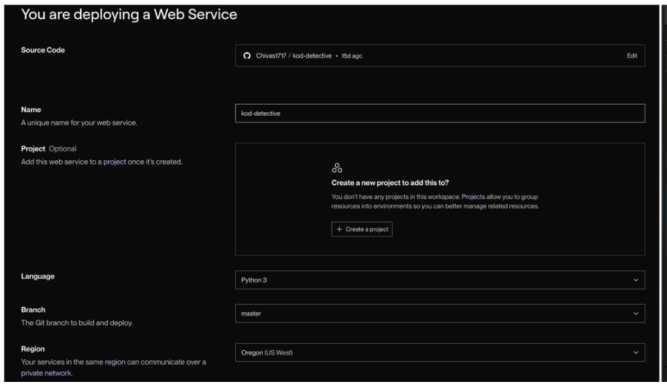


Postman для тестування бекенду

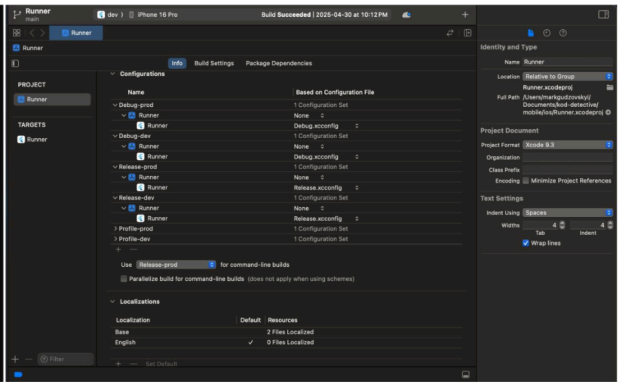


XCode iPhone симулятор для ручного тестування застосунку

Розгортання системи

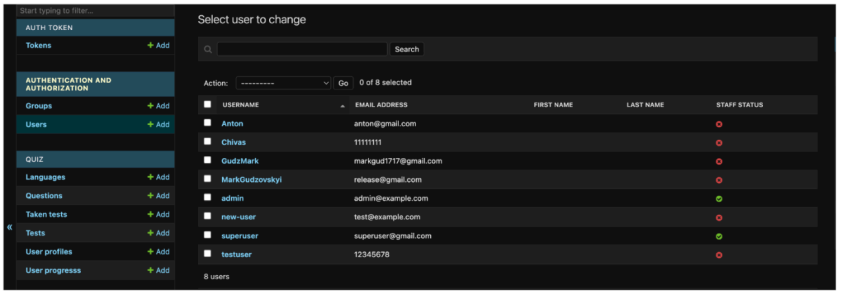


Налаштування Render веб сервісу



Налаштування застосунку в Xcode

Інтерфейс адміністратора



Інтерфейс адміністратора

Висновки

Основні результати роботи:

- Розроблено набір з 5 типів інтерактивних завдань з програмування
- Створено унікальний детективний інтерфейс з ШІ асистентом "Детектив Код"
- Реалізовано систему підказок та контекстної допомоги
- Впроваджено систему рейтингу та лідерборд
- Інтегровано Llama 3 8B для персоналізованої підтримки
- Створено кросплатформенний Flutter додаток (Android/iOS)
- Розроблено Django REST API з SQLite базою даних

Практична цінність:

Інноваційний підхід до навчання програмування через гейміфікацію та ШІ

Перспективи розвитку:

Розширення бази питань та мов програмування

Створення викладацького інтерфейсу

Впровадження адаптивної складності завдань