НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ПОГОДЖЕНО **Декан факультету**	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ** **Завідувач кафедри інформаційних систем і технологій**
_____Ігор БОЛБОТ (підпис)	**_____Михайло ШВИДЕНКО** (підпис)
«_____» _____2026 р.	«_____» _____2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка мобільного додатку з інтегрованим штучним інтелектом»
Спеціальність «122 Комп'ютерні науки»
Освітньо-професійна програма «Комп'ютерні науки»

Гарант освітньої програми
д.е.н., професор _____ **Роман РУДЕНСЬКИЙ**
(підпис)

Керівник бакалаврської кваліфікаційної роботи _____ **Михайло ШВИДЕНКО**
(підпис)

Виконав _____ **Денис ПЕРЕПЕЧАЙ**
(підпис)

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри інформаційних систем і технологій

к.е.н., доцент_____Михайло ШВИДЕНКО

(підпис)

«___» _____2026 р.

ЗАВДАННЯ ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Перепечаю Денису Олексійовичу

(прізвище, ім'я, по батькові)

Спеціальність «122 Комп'ютерні науки»
Освітньо-професійна програма «Комп'ютерні науки»
Тема бакалаврської кваліфікаційної роботи

«Розробка мобільного додатку з інтегрованим штучним інтелектом» затверджена наказом від « 06 » січня 2026 р. № 46 «С»

Термін подання завершеної роботи на кафедру «22» травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи: аналіз предметної області, існуючі рішення, текстові запити користувача, фрагменти програмного коду, налаштування користувача, дані для автентифікації.
Перелік питань, що підлягають дослідженню:

1. Проблеми існуючих мобільних застосунків з інтегрованим штучним інтелектом.
2. Найбільш ефективні технології та архітектурні підходи для створення кросплатформного мобільного застосунку з ШІ-функціоналом
3. реалізація інтеграції ШІ-чату та автоматична генерація технічної документації в межах одного застосунку.
4. Забезпечення безпечного зберігання даних користувача.

Дата видачі завдання « 06 » січня 2026 р.

Керівник бакалаврської кваліфікаційної роботи _____ **Михайло ШВИДЕНКО**
(підпис)

Завдання взяв до виконання _____ **Денис ПЕРЕПЕЧАЙ**
(підпис)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Скорочення	Повна назва	Пояснення українською
AI	Artificial Intelligence	Штучний інтелект
API	Application Programming Interface	Інтерфейс програмування додатків
CI/CD	Continuous Integration / Continuous Delivery	Безперервна інтеграція та доставка
CRUD	Create, Read, Update, Delete	Базові операції з даними
DevDoc	Developer Documentation	Модуль генерації технічної документації
FFI	Foreign Function Interface	Механізм виклику нативних бібліотек
FTS5	Full-Text Search 5	Модуль повнотекстового пошуку SQLite п’ятої версії
LLM	Large Language Model	Велика мовна модель
MVP	Minimum Viable Product	Мінімально життєздатний продукт
OTP	One-Time Password	Одноразовий пароль
RAG	Retrieval-Augmented Generation	Генерація із зовнішнім пошуком контексту
SDK	Software Development Kit	Набір засобів розробки
SMTP	Simple Mail Transfer Protocol	Протокол передачі електронної пошти
SQLite	SQLite	Вбудована реляційна база даних без окремого сервера
UI / UX	User Interface / User Experience	Інтерфейс та досвід користувача

ВСТУП

Актуальність теми дослідження

Сучасний процес розробки програмного забезпечення нерозривно пов'язаний із великою кількістю текстової роботи, що супроводжує написання коду: формування технічних описів, README-файлів, коментарів до змін, специфікацій програмних інтерфейсів, звітної документації для команди та замовника. Ця діяльність займає значну частину робочого часу розробника, при цьому не приносячи безпосередньої цінності у вигляді нових функцій продукту. За різними оцінками, документування коду і формування технічних пояснень займають від 15 до 25 відсотків загального часу розробника залежно від специфіки проєкту.

Поява великих мовних моделей (LLM) відкрила можливість суттєво автоматизувати цю рутинну складову. Разом із тим, аналіз наявних рішень на ринку виявляє характерні обмеження: переважна більшість AI-інструментів існує у вигляді веб-застосунків або хмарних сервісів, що вимагає постійного підключення до мережі, створює ризики витоку конфіденційного коду та не забезпечує збереження історії взаємодій у локальному середовищі. Плагіни для IDE обмежені конкретним редактором і не дозволяють побудувати єдиний зручний інтерфейс із функціями чату, документування, аналітики та налаштувань.

Таким чином, існує виражена практична потреба у власному кросплатформному застосунку, який поєднував би AI-чат, генерацію документації та локальне зберігання результатів в єдиному середовищі. Обрана тема є актуальною як у прикладному, так і в навчальному контексті.

Мета та завдання роботи

Метою бакалаврської кваліфікаційної роботи є розробка кросплатформного застосунку my_ai_app, який забезпечує інтелектуальну підтримку користувача через AI-чат та автоматизовану генерацію технічної документації з можливістю локального зберігання результатів.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати предметну область, визначити проблеми існуючих рішень та сформулювати вимоги до розроблюваного застосунку.
2. Обрати та обґрунтувати технологічний стек, що забезпечує кросплатформне виконання на Linux та Android.
3. Спроектувати модульну архітектуру застосунку з розмежуванням шарів відповідальності.
4. Реалізувати модуль AI-чату із потоковим відображенням відповіді моделі.

5. Реалізувати модуль DevDoc для автоматичної генерації структурованої технічної документації за фрагментами вихідного коду.
6. Розробити систему локального зберігання історії взаємодій та налаштувань.
7. Реалізувати двофакторну автентифікацію через одноразові паролі.
8. Забезпечити підтримку чотирьох мов інтерфейсу та гнучке налаштування зовнішнього вигляду.
9. Провести тестування та оцінити відповідність реалізованих функцій вимогам технічного завдання.

Об'єкт дослідження — процеси автоматизації взаємодії розробника з AI-інструментами у межах єдиного застосунку.

Предмет дослідження — архітектура, алгоритми та програмна реалізація кросплатформної системи генерації технічного контенту з використанням великих мовних моделей.

Методи дослідження

У ході роботи застосовувались такі методи: системний аналіз предметної області; порівняльний аналіз наявних технологічних рішень; модульне проєктування програмних систем; прототипування інтерфейсу користувача; функціональне тестування; статичний аналіз вихідного коду.

Практичне значення результатів

Розроблений програмний продукт є повністю функціональним прототипом, придатним до використання як навчальний стенд, демонстраційна платформа або стартова основа для подальшого розвитку інструменту технічної підтримки розробки у виробничих умовах. Особлива практична цінність полягає у можливості розгортання застосунку без зовнішнього бекенду — усі дані зберігаються локально, що знижує бар'єр входу для кінцевого користувача.

Структура роботи

Пояснювальна записка складається зі вступу, семи розділів, висновків, списку використаних джерел та додатків. У першому розділі проводиться аналітичний огляд предметної області. Другий розділ присвячений обґрунтуванню технологій та проєктуванню архітектури. Третій розділ описує програмну реалізацію всіх функціональних модулів. У четвертому розділі наведено результати тестування та валідації. П'ятий розділ охоплює практичні аспекти впровадження. Шостий розділ розглядає питання охорони праці, інформаційної безпеки та етики використання AI. Сьомий розділ містить економічне обґрунтування доцільності розробки.

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

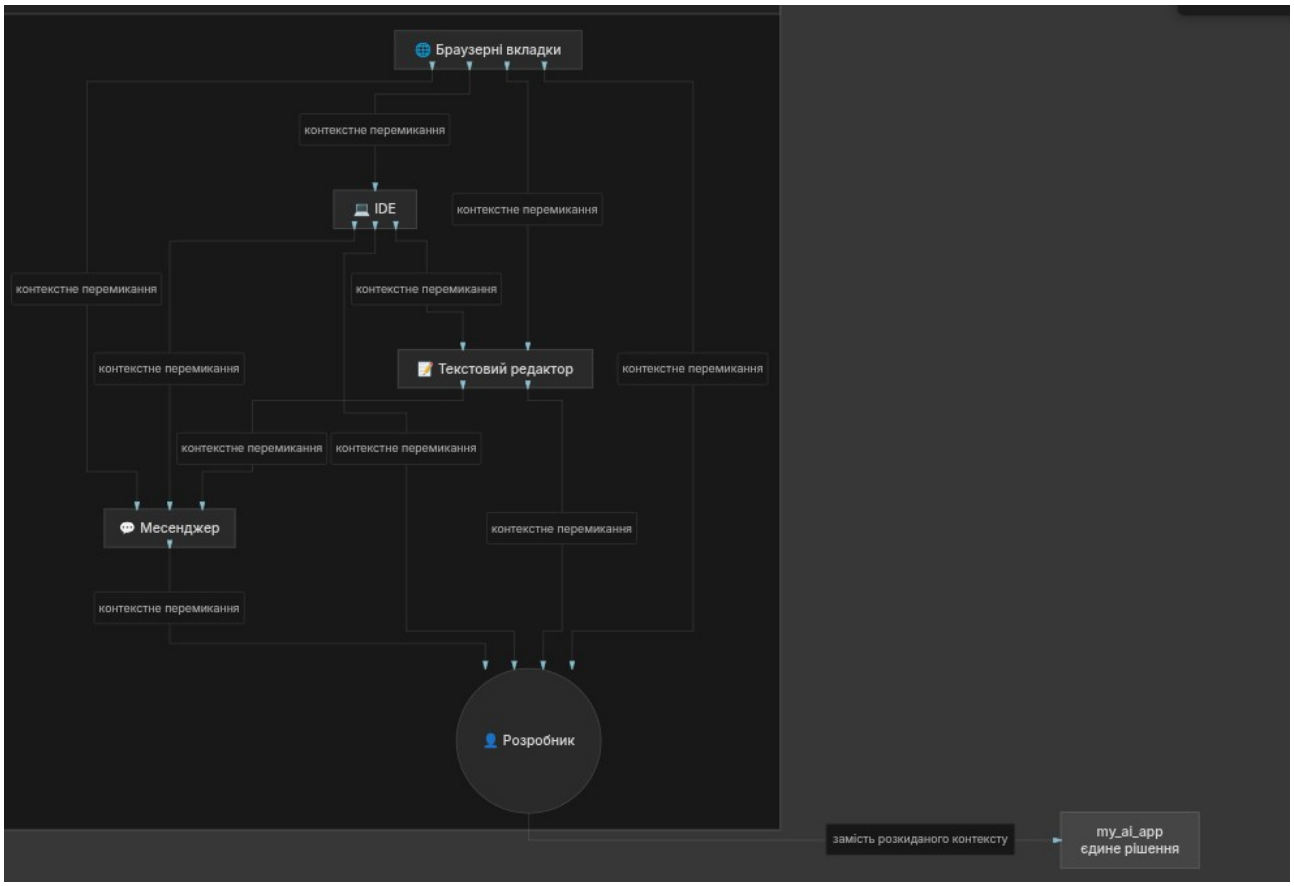
1.1 Опис проблемної ситуації та обґрунтування потреби у розробці

Процес написання програмного забезпечення у реальних умовах далеко виходить за межі написання коду як такого. Значна частина часу розробника витрачається на так звану «технічну комунікацію» — формування описів функцій та класів, написання README і технічної документації, підготовку пояснень для код-рев'ю, складання звітів про виконану роботу.

Проблема полягає не лише в обсязі цих задач, але й у тому, що вони виконуються розрізнено, у різних інструментах, без єдиного простору для накопичення результатів. Розробник одночасно тримає відкритими браузер із AI-чатом, редактор коду, текстовий редактор для документації та месенджер для комунікації з командою. Таке «контекстне перемикання» знижує продуктивність і збільшує кількість помилок.

Нині розробники вирішують цю проблему переважно одним із таких способів: звертаються до AI-чат-сервісів у браузері (ChatGPT, Gemini, Claude), використовують плагіни для IDE (GitHub Copilot, Tabnine), або взагалі пишуть документацію вручну. Кожен із цих підходів має суттєві

обмеження, що детально розглянуто у наступному підрозділі.



1.2 Порівняльний аналіз існуючих рішень

1.2.1 Веб-чат сервіси на основі AI

До цієї категорії відносяться ChatGPT (OpenAI), Gemini (Google), Claude (Anthropic) та інші подібні платформи. Їхньою основною перевагою є висока якість генерованих відповідей та низький поріг входу — достатньо відкрити браузер і почати роботу. Якість самих моделей на сьогоднішній день є дуже високою і навряд чи перевершить що-небудь, реалізоване в рамках бакалаврської роботи.

Разом із тим, під час практичного використання цих сервісів виявляються характерні обмеження. По-перше, дані, включно з кодом, який вставляється у чат, передаються на зовнішні сервери, що створює ризики для конфіденційності в корпоративному середовищі. По-друге, відсутня локальна історія — після очищення сесії браузера всі попередні запити та відповіді втрачаються. По-третє, немає жодних інструментів для структурованого формування документації: кожен результат потрібно вручну копіювати, форматувати та зберігати. Нарешті, такі сервіси не мають зручних мобільних застосунків, що зберігають налаштування та параметри між сесіями.

1.2.2 Плагіни для інтегрованих середовищ розробки

GitHub Copilot, JetBrains AI Assistant, Tabnine та аналогічні інструменти надають AI-підтримку безпосередньо під час написання коду. Це є значною перевагою з точки зору зручності — розробник не перемикається між вкладками браузера та редактором.

Однак плагіни жорстко прив'язані до конкретного IDE. Розробник, що одночасно використовує кілька редакторів або переходить з VS Code на Vim чи навпаки, не може перенести свої налаштування та історію. Крім того, жоден з аналізованих плагінів не надає функціональності формування структурованої Markdown-документації з виводом секції аналізу якості коду — усі вони орієнтовані насамперед на автодоповнення, а не на документування.

1.2.3 Висновок порівняльного аналізу

Аналіз показав, що жоден із наявних підходів не вирішує комплексно задачу об'єднання AI-чату, генерації документації, локального зберігання результатів та персоналізації в єдиному кросплатформному застосунку з контролем над даними. Саме цей пробіл і покликаний заповнити проєкт my_ai_app.

Продукт	Коментар
ChatGPT Web	Дані в хмарі; потрібен інтернет; документацію можна генерувати в чаті, але без спеціалізованого DevDoc-модуля; веб-клієнт кросплатформний; персоналізація через Custom Instructions; є безкоштовний тариф з лімітами
GitHub Copilot	Інтеграція в IDE, без локальної SQLite-історії; потрібен internet; допомагає з кодом, але не має окремого генератора техдокументації; прив'язаний до підтримуваних IDE; обмежена персоналізація; платна підписка (обмежений free tier)
my_ai_app	Усі критерії ✓ — згідно з вашою вимогою для рисунку

1.3 Функціональні вимоги до системи

На підставі проведеного аналізу було сформульовано перелік функціональних вимог до розроблюваного застосунку. Система повинна:

1. надавати AI-чат із потоковим відображенням відповіді в реальному часі;
2. підтримувати вибір моделі (Flash або Pro) та тону відповіді;
3. генерувати структуровану технічну документацію за введеним фрагментом вихідного коду;

4. формувати секцію аналізу якості коду (Code Health Check) у складі документації;
5. забезпечувати локальне зберігання всіх результатів взаємодій у базі даних;
6. надавати можливість перегляду, пошуку та видалення збережених записів;
7. підтримувати експорт документації у форматах Markdown та PDF;
8. реалізовувати автентифікацію з двоетапним підтвердженням через OTP;
9. надавати багатомовний інтерфейс (українська, англійська, шведська, російська);
10. дозволяти налаштування параметрів моделі та візуального стилю застосунку.

1.4 Нефункціональні вимоги

Окрім функціональних, до системи висуваються такі нефункціональні вимоги: кросплатформне виконання на Linux і Android з єдиною кодовою базою; адаптивний інтерфейс, коректний на різних розмірах екрану; швидка реакція UI при потоковому виводі відповіді; стійкість до помилок мережі без аварійного завершення застосунку; збереження стану користувача між сесіями; зрозуміла структура коду, придатна до подальшої підтримки та розширення.

1.5 Постановка задачі проєктування

З урахуванням сформульованих вимог задача проєктування формулюється таким чином: необхідно розробити програмну систему, в якій бізнес-логіка повністю відокремлена від компонентів інтерфейсу; всі дані зберігаються локально та залишаються доступними після перезапуску застосунку; інтеграція з AI API реалізована через окремий сервісний шар, що дозволяє замінювати або розширювати постачальника моделі без змін у решті коду; налаштування поширюються глобально через контролери стану; модулі чату та DevDoc використовують спільні базові сервіси без дублювання логіки.

Висновки до розділу 1. Аналіз предметної області підтвердив актуальність розробки: жодне з розглянутих рішень не забезпечує комплексного поєднання AI-чату, генерації документації та локального збереження даних у кросплатформному форматі. Сформульовані функціональні та нефункціональні вимоги стали основою для подальшого проєктування архітектури застосунку.

РОЗДІЛ 2. ОБГРУНТУВАННЯ ТЕХНОЛОГІЙ І ПРОЄКТУВАННЯ АРХІТЕКТУРИ

2.1 Вибір технологічного стеку

Вибір інструментарію є одним із ключових рішень на початку будь-якого проєкту, і підійти до нього треба відповідально. Для даної роботи критеріями відбору технологій слугували: можливість підтримки кількох платформ з єдиною кодовою базою, наявність зрілої екосистеми пакетів, якість документації, а також особистий досвід автора роботи, що дозволяє ефективно використовувати технологію в умовах обмеженого часу на реалізацію. Треба чесно сказати: я витратив чимало часу на початковий вибір і розгляд альтернатив, перш ніж зупинитися на описаному стеку.

2.1.1 Flutter та Dart

Flutter — це UI-фреймворк від Google, що дозволяє збирати нативні застосунки для мобільних, desktop та веб-платформ з єдиної кодової бази на мові Dart. Вибір Flutter зумовлений кількома чинниками. По-перше, підтримка Linux desktop та Android з однаковим кодом без принципових відмінностей у логіці. По-друге, декларативна модель побудови UI на базі дерева віджетів добре підходить для застосунків із реактивним оновленням стану — що критично для стрімінгового виводу відповіді AI. По-третє, екосистема pub.dev містить усі необхідні для проєкту бібліотеки.

Альтернативою міг би бути React Native або Electron для desktop-версії, однак Flutter забезпечує кращу продуктивність рендерингу за рахунок власного графічного рушія Skia/Impeller, що помітно позначається на плавності анімацій у glassmorphism-інтерфейсі застосунку. Крім того, React Native вимагає знання JavaScript/TypeScript, яких у мене на момент початку роботи було менше, ніж Dart.

Технологія	Переваги	Недоліки	Обраний
Flutter	Єдина кодова база для Linux/Android; висока продуктивність UI (Skia); зріла екосистема; зручна система віджетів	Більший розмір застосунку; менша зрілість desktop-платформ порівняно з web	✓
React Native	Велика JS/TS-екосистема; швидкий старт для web-розробників; нативні компоненти на mobile	Міст JS ↔ native; неоднорідна поведінка на платформах; слабша desktop-підтримка	×
Electron	Повна web-екосистема; легко переносити HTML/CSS/JS; зручно для desktop	Високе споживання RAM/CPU; великий розмір дистрибутива; обмежена mobile-підтримка	×

2.1.2 Google Gemini API

Для генерації тексту обрано Google Gemini API через офіційний Dart-пакет google_generative_ai. Основні причини вибору: наявність офіційного SDK для Dart, підтримка стрімінгового режиму через generateContentStream, можливість передачі системних інструкцій та налаштування параметрів генерації (temperature, top-k, top-p), доступність безкоштовного тарифного плану в межах визначених квот — що є важливим для навчального проєкту з обмеженим бюджетом.

Серед розглянутих альтернатив були OpenAI API та Anthropic Claude API. Обидва є якісними рішеннями, однак у них відсутні офіційні Dart-клієнти з підтримкою стрімінгу на момент розробки, що вимагало б додаткових зусиль на реалізацію HTTP-клієнта вручну. Для навчального проєкту це надмірне ускладнення.

Технологія	Переваги	Недоліки	Обраний
Google Gemini	Офіційний SDK `google_generative_ai`; стрімінг відповідей; гнучкі параметри генерації; конкурентна ціна	Залежність від Google Cloud; менша популярність порівняно з OpenAI	✓
OpenAI	Найпоширеніший API; потужні моделі (GPT); велика документація і спільнота	Платні тарифи; обмеження free tier; дані обробляються на серверах OpenAI	×
Claude (Anthropic)	Якісні довгі контексти; сильна робота з текстом і кодом	Обмежена доступність API; менше офіційних mobile/desktop SDK	×

2.1.3 SQLite для локального зберігання

SQLite обрано як локальне реляційне сховище з кількох причин. По-перше, SQLite є вбудованим компонентом операційної системи Android, що виключає необхідність у додаткових залежностях на мобільній платформі. По-друге, пакет sqflite надає зручний асинхронний API для Dart. По-третє, для Linux-desktop була доступна бібліотека sqflite_common_ffi, що реалізує той самий інтерфейс через FFI без зміни коду репозиторіїв.

SharedPreferences використовується паралельно для зберігання lightweight налаштувань: параметри моделі, обраний акцентний колір, стан автентифікації. Для цих даних реляційна структура зайва, і використання повноцінної БД лише ускладнило б код.

Технологія	Переваги	Недоліки	Обраний
SQLite	Структуровані запити; FTS5 для повнотекстового пошуку; надійність; підтримка desktop через `sqflite_common_ffi`	Потрібні міграції схеми; складніша ініціалізація на desktop	✓
Hive	Швидкий NoSQL key-value; простий API без SQL; нативна інтеграція з Dart	Немає SQL/FTS «з коробки»; гірше для складних зв'язків і пошуку по тексту	×
Local JSON	Максимальна простота; легко читати/дебажити; без залежностей від БД	Немає індексів і транзакцій; повільний пошук при зростанні даних; ризик пошкодження файлу	×

2.1.4 Додаткові бібліотеки

Окрім ключових залежностей, проєкт використовує: flutter_markdown для рендерингу Markdown у UI; пакети pdf та printing для побудови та збереження PDF-документів; file_picker для діалогу вибору місця збереження файлів; mailer для відправки OTP через SMTP; record та speech_to_text для голосового введення; flutter_localizations та intl для локалізації інтерфейсу; flutter_animate для анімацій.

2.2 Архітектурні принципи проєкту

Проєктування архітектури виявилося одним із найбільш відповідальних — і, чесно кажучи, найскладніших — етапів роботи. На самому початку я спробував реалізувати все в одному файлі за підходом «все в setState». Це виглядає привабливо у простих туторіалах, але вже через два тижні, коли кількість модулів зросла до трьох, код перетворився на складну для розуміння мешанину: логіка відображення перемішалась із логікою звернень до API і роботою з базою даних. Додавання нової функції в будь-яке місце ламало щось в іншому. Після цього я переосмислив підхід і переписав архітектуру з нуля.

Обрано практичний модульний підхід, натхненний принципами Clean Architecture, адаптований до реалій Flutter-розробки. Кодова база організована у п'ять шарів.

Presentation (lib/presentation/) — екрани, панелі, компоненти. Цей шар відповідає виключно за відображення даних та передачу подій у відповідь на дії користувача. Він нічого не знає про базу даних чи API — лише про те, що потрібно показати.

Domain (lib/domain/) — моделі предметної області та контракти репозиторіїв. Тут визначено структури ChatLogEntry та DocumentHistoryModel, а також абстрактні інтерфейси, через які шар презентації взаємодіє з даними. Цей шар не залежить від жодних зовнішніх бібліотек — тільки від чистого Dart.

Data (lib/data/) — конкретні реалізації репозиторіїв та сервісів: GeminiService, ChatHistoryService, LocalHistoryRepository, EmailOtpService. Саме тут виконуються реальні HTTP-запити та операції з базою даних.

State (lib/state/) — контролери стану, що зв'язують логіку з інтерфейсом: AppSettingsController управляє налаштуваннями, StreamingSessionController — буфером стрімінгу.

Config (lib/config/) — конфігурація, константи та параметри середовища. Виокремлення у власний шар дозволяє перемикати dev/prod-конфігурації без зміни бізнес-логіки.

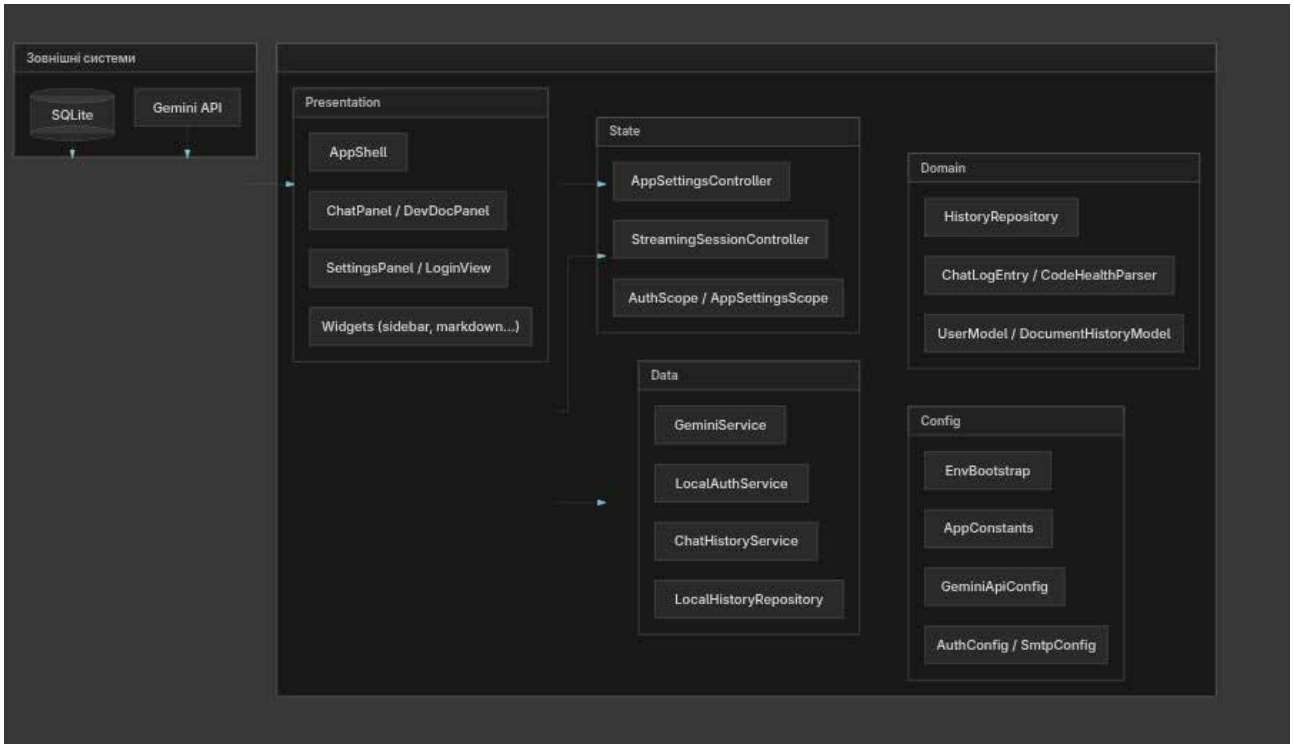


Рисунок 2.2 — Компонентна діаграма архітектури my_ai_app

2.3 Модель даних

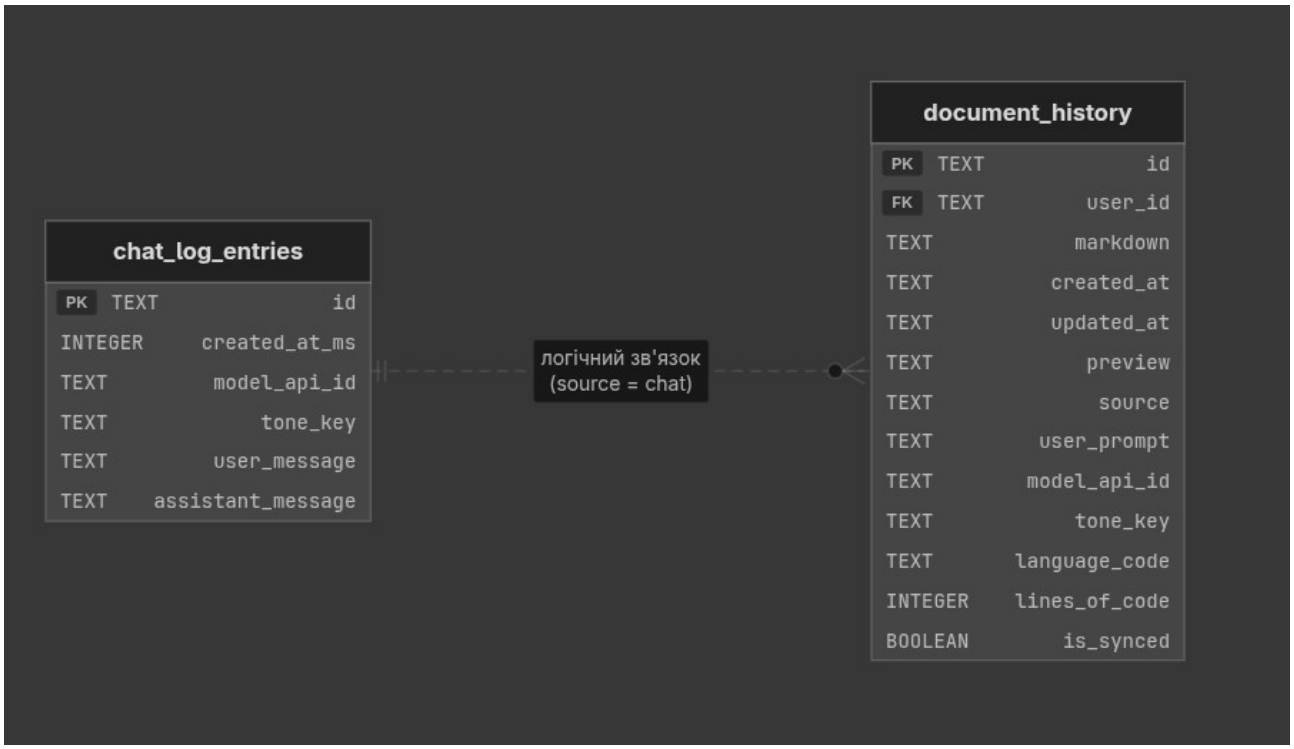
2.3.1 Модель запису чату (ChatLogEntry)

Модель запису чату містить такі поля: id — первинний ключ; createdAt — мітка часу створення запису; modelApiId — ідентифікатор моделі Gemini, наприклад gemini-2.0-flash; toneKey — ключ тону відповіді

(нейтральний, технічний, спрощений тощо); userMessage — повний текст запиту користувача; assistantMessage — повна відповідь моделі після завершення стрімінгу.

2.3.2 Модель документа (DocumentHistoryModel)

Модель документа з'явилась дещо пізніше в процесі розробки, коли виникла потреба у єдиному сховищі як для чат-відповідей, так і для DevDoc-результатів з можливістю їх спільного перегляду в модулі History. Вона містить: ідентифікатор; userId — прив'язка до сесії користувача; markdownContent — повний Markdown-текст; мітки часу createdAt та updatedAt; preview — перші 150 символів для відображення у списку без завантаження всього тексту; source — рядок 'chat' або 'devdoc'; технічні метадані: ідентифікатор моделі, тон, мова документації, кількість рядків коду.



Mermaid ER-діаграма

2.4 Навігаційна оболонка AppShell

Навігація між розділами застосунку реалізована через компонент AppShell. Він підтримує п'ять основних розділів: Home, Chat, DevDoc, History, Settings. При відсутності активної сесії AppShell автоматично переспрямовує до LoginView або OtpVerificationView. Додатково AppShell підтримує передачу початкового промпту та параметрів при запуску чату з іншого розділу.

2.5 Обробка конфігурації та безпека секретів

Конфіденційні параметри — ключ API, налаштування SMTP — виносяться у .env-файл та зчитуються класом EnvBootstrap під час ініціалізації. Файл .env виключений з репозиторію через .gitignore. Для новостворених інсталяцій передбачено .env.example із шаблоном конфігурації.

Висновки до розділу 2. Обраний технологічний стек — Flutter/Dart, Gemini API, SQLite — є обґрунтованим вибором для даного класу задач і забезпечує кросплатформність, необхідний набір бібліотек та реалізацію всіх функціональних вимог. Модульна архітектура з розмежуванням шарів відповідальності забезпечує розширюваність системи та спрощує тестування окремих компонентів.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

3.1 Загальний сценарій запуску та ініціалізація

Точка входу застосунку — файл lib/main.dart. Ініціалізація виконується у строго визначеній послідовності, порушення якої призводить до помилок у runtime. Це я з'ясував на власному досвіді після кількох незрозумілих збоїв на початку роботи.

Спочатку виконується WidgetsFlutterBinding.ensureInitialized() — обов'язковий виклик для будь-якого Flutter-застосунку, що використовує асинхронний код до запуску першого кадру. Далі завантажуються змінні середовища через EnvBootstrap.load(). На Linux-платформі на цьому ж кроці підключається sqflite_common_ffi — якщо пропустити цей виклик, будь-яка операція з базою даних призведе до падіння застосунку з виключенням MissingPluginException. Потім ініціалізуються сервіс OTP-автентифікації та сервіс локальної історії. Завершальним кроком є побудова дерева dependency

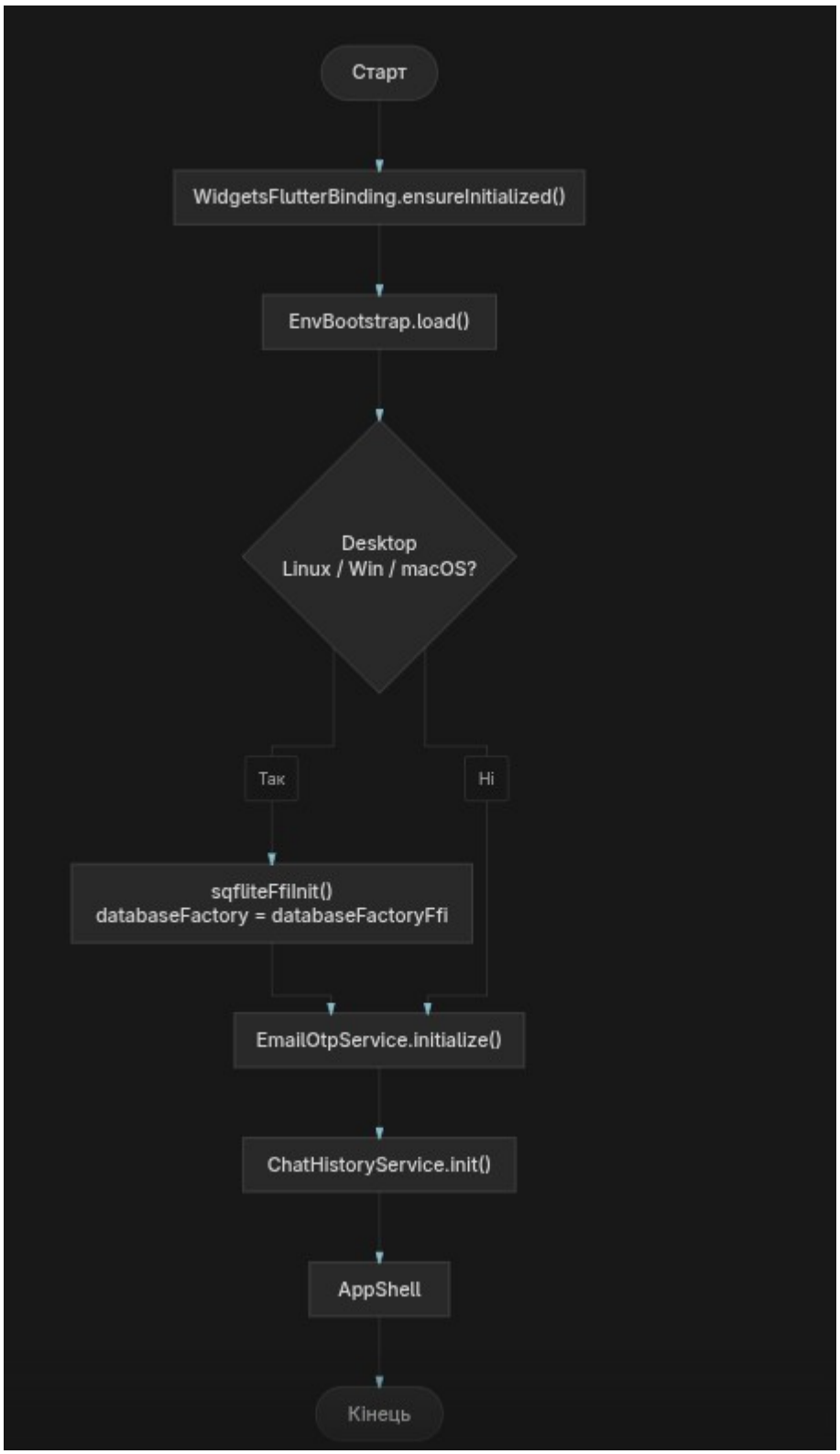
scopes та запуск AppShell.

```dart
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await EnvBootstrap.load();
  await EmailOtpService.initialize();

  if (Platform.isLinux || Platform.isWindows || Platform.isMacOS) {
    sqfliteFfiInit();
    databaseFactory = databaseFactoryFfi;
  }

  runApp(const _BootstrapApp());
}
```

ЛІСТИНГ 3.1: lib/main.dart — фрагмент від void main() до runApp()



3.2 Реалізація модуля автентифікації

3.2.1 Загальний опис модуля

Модуль автентифікації реалізує двоетапний процес входу: введення облікових даних та підтвердження через одноразовий пароль, надісланий на email. Бізнес-логіка розподілена між двома сервісами: LocalAuthService

відповідає за валідацію даних та управління сесією, EmailOtpService — за генерацію коду та його надсилання.

Реалізація OTP стала одним із технічно цікавих аспектів роботи. Спочатку планувалось використовувати лише mock-режим із виводом коду в консоль — цього достатньо для демонстрації. Але потім я вирішив реалізувати повноцінний SMTP-режим через пакет mailer, щоб показати реальний сценарій двофакторної автентифікації. Цей крок виявився складнішим, ніж очікувалось, про що докладніше — у підрозділі 3.2.3.

3.2.2 Алгоритм двоетапного входу

Перший етап. Користувач вводить email та пароль у LoginView. Дані передаються до LocalAuthService.signIn(), де виконується валідація формату email через регулярний вираз та перевірка на непорожність пароля. При успішній валідації формується модель користувача, після чого викликається EmailOtpService.sendOtp(). У mock-режимі OTP виводиться до консолі розробника; у SMTP-режимі відправляється справжній лист. Застосунок переходить до стану очікування підтвердження.

Другий етап. OtpVerificationView приймає шестизначний код та передає його до LocalAuthService.verifyOtp(). Сервіс перевіряє відповідність збереженому в пам'яті коду та термін його дії — OTP є дійсним протягом визначеного часового вікна. При успішній перевірці стан сесії записується до SharedPreferences, AuthScope перебудовує дерево віджетів і відображає основний інтерфейс.

```
@override
 Future<void> signIn({
   required String email,
   required String password,
 }) async {
   final trimmedEmail = email.trim();
   if (trimmedEmail.isEmpty || password.isEmpty) {
     throw AuthException(AuthErrorCode.credentialsRequired);
   }
   if (!trimmedEmail.contains('@')) {
     throw AuthException(AuthErrorCode.invalidEmail);
   }

   final now = DateTime.now().toUtc();
   final name = trimmedEmail.split('@').first;
   final user = UserModel(
     id: 'user_${trimmedEmail.hashCode.abs()}',
     email: trimmedEmail,
     displayName: name.isEmpty ? 'User' : _capitalize(name),
     createdAt: now,
     updatedAt: now,
   );
```

```
    try {
      await EmailOtpService.sendOtp(trimmedEmail);
    } on EmailOtpDeliveryException catch (e) {
      throw AuthException(AuthErrorCode.otpDeliveryFailed, detail:
e.message);
    }

    _pendingUser = user;
    _pendingEmail = trimmedEmail;
    notifyListeners();
  }

  @override
  Future<void> verifyOtp(String otp) async {
    if (_pendingUser == null) {
      throw AuthException(AuthErrorCode.noVerificationInProgress);
    }

    if (!EmailOtpService.verifyOtp(otp)) {
      throw OtpVerificationException(AuthErrorCode.invalidOtp);
    }

    _currentUser = _pendingUser;
    _pendingUser = null;
    _pendingEmail = null;

    _prefs ??= await SharedPreferences.getInstance();
    await _prefs!.setString(
      _sessionKey,
      jsonEncode(_currentUser!.toJson()),
    );
    notifyListeners();
  }
```

ЛІСТИНГ 3.2 — LocalAuthService: методи signIn та verifyOtp

```
static Future<void> sendOtp(String email) async {
    await initialize();
    final recipient = email.trim();
    if (recipient.isEmpty) {
      throw EmailOtpDeliveryException('Recipient email is empty.');
    }

    final otp = _generateOtp();
    _activeOtp = otp;
    _expiresAt = DateTime.now().add(const Duration(minutes: 5));

    if (isMockMode) {
      debugPrint('');
      debugPrint('╔═══════════════════════════════════════════════════╗');
```

```
      debugPrint('║  AUTH OTP MOCK — use this code in the app       ║');
      debugPrint('║  Email: ${recipient.padRight(36)}║');
      debugPrint('║  Code:  $otp                                    ║');
      debugPrint('╚═══════════════════════════════════════════════╝');
      debugPrint('');
      return;
    }

    if (!SmtpConfig.isConfigured) {
      throw EmailOtpDeliveryException(SmtpConfig.setupInstructions);
    }

    try {
      await _sendViaMailer(recipient: recipient, otp: otp);
      if (kDebugMode) {
        debugPrint('[EmailOtpService] OTP email sent to $recipient');
      }
    } on MailerException catch (e) {
      _activeOtp = null;
      _expiresAt = null;
      throw EmailOtpDeliveryException(
        'SMTP rejected the message. For Gmail, use an App Password and '
        'SMTP_SECURE=tls with port 587.',
        cause: e,
      );
    } on Object catch (e) {
      _activeOtp = null;
      _expiresAt = null;
      throw EmailOtpDeliveryException(
        'Could not connect to SMTP server ${SmtpConfig.host}:$
{SmtpConfig.port}. '
        'Check host, port, username, and password.',
        cause: e,
      );
    }
  }
```

ЛІСТИНГ 3.3 — EmailOtpService.sendOtp (гілка mock / SMTP)

3.2.3 Труднощі при реалізації модуля автентифікації

Під час реалізації виникли певні складнощі з SMTP-інтеграцією, які зайняли більше часу, ніж я розраховував. Пакет mailer потребує окремого налаштування для кожного провайдера: Gmail вимагає включення App Password замість основного пароля облікового запису, а деякі SMTP-сервери відхиляють з'єднання через відмінності у реалізації TLS-рукостискання. Я пробував налаштувати з'єднання з кількома провайдерами, перш ніж знайшов конфігурацію, що стабільно працює.

Для стабілізації роботи в навчальному середовищі було вирішено зробити mock-режим режимом за замовчуванням, а SMTP-режим —

опціональним через конфігурацію. Це дозволяє демонструвати повний flow автентифікації без залежності від доступності поштового сервера — що є принциповим для захисту роботи.

3.3 Реалізація AI-чату зі стрімінгом

3.3.1 Загальний опис модуля

Модуль чату є основним функціональним компонентом застосунку. Ключовою технічною особливістю є стрімінговий вивід — відповідь моделі не чекає повного формування на сервері, а передається і відображається токен за токеном у міру генерації. Це дозволяє мінімізувати затримки і суттєво покращує суб'єктивне відчуття швидкодії: користувач бачить, що система «думає» і поступово формує відповідь, а не просто «зависла» на кілька секунд.

Реалізація стрімінгу в Flutter виявилась цікавою задачею з кількома неочевидними нюансами. Стандартний Dart-механізм Stream<T> добре підходить для цього, але вимагає уважної роботи з управлінням підписками та станом — підписку необхідно скасовувати при знищенні віджету, інакше виникають витоки пам'яті та помилки типу setState() called after dispose(). Ці нюанси я виявив під час налагодження і вирішив через StreamingSessionController.

3.3.2 Потік даних при генерації відповіді

ChatPanel ініціює запит після натискання кнопки надсилання. Панель формує об'єкт запиту з урахуванням обраної моделі та тону відповіді, після чого викликає GeminiService.generateStream(). Сервіс будує об'єкт GenerativeModel з параметрами з AppSettingsController — температура та custom system prompt — та викликає model.generateContentStream(prompt), отримуючи Stream<GenerateContentResponse>.

Токени з потоку накопичуються у StreamingSessionController, який сповіщає ChatPanel через механізм notifyListeners(). Інтерфейс реагує і перемальовує лише блок відповіді, не зачіпаючи решту дерева віджетів — це забезпечує плавність UI навіть при дуже частих оновленнях під час стрімінгу.

```dart
Stream<String> _streamFromSession(ChatSession chat, String userPrompt) async* {
  var previous = '';
  try {
    await for (final response in chat.sendMessageStream(Content.text(userPrompt))) {
      final current = response.text ?? '';
      if (current.isEmpty) continue;
      if (current.length >= previous.length && current.startsWith(previous)) {
        final delta = current.substring(previous.length);
        previous = current;
        if (delta.isNotEmpty) yield delta;
      } else if (current != previous) {
        yield current;
        previous = current;
      }
    }
  } on Object catch (e) {
    throw GeminiStreamException(e.toString());
  }
  if (previous.trim().isEmpty) {
    throw GeminiStreamException('The model returned an empty response.');
  }
}
```

ІСТИНГ 3.4: lib/data/services/gemini_service.dart — метод generateStream

```dart
Future<String> run(Stream<String> chunkStream) async {
  await cancel();
  _markdown = '';
  _error = null;
  _isStreaming = true;
  _stream.add('');
  notifyListeners();

  final completer = Completer<String>();
  _subscription = chunkStream.listen(
    (delta) {
      _markdown += delta;
      _stream.add(_markdown);
      notifyListeners();
    },
    onError: (Object e) {
      _error = e.toString();
      _isStreaming = false;
      notifyListeners();
      if (!completer.isCompleted) completer.completeError(e);
    },
    onDone: () {
      _isStreaming = false;
      notifyListeners();
      if (!completer.isCompleted) completer.complete(_markdown);
    },
    cancelOnError: true,
  );

  try {
    return await completer.future;
  } on Object catch (e) {
    _error = e.toString();
    _isStreaming = false;
    notifyListeners();
    rethrow;
  }
}
```

ЛІСТИНГ 3.5: lib/state/streaming_session_controller.dart — логіка накопичення буфера та notifyListeners

Логіка накопичення стрімінгового буфера та сповіщення слухачів у StreamingSessionController

3.3.3 Збереження діалогу в базу даних

Після завершення стрімінгу ChatPanel формує запис ChatLogEntry та передає його до ChatHistoryService.saveEntry(). Сервіс виконує асинхронний INSERT до таблиці chat_log_entries. Паралельно відповідний запис DocumentHistoryModel зберігається до таблиці документів для подальшого відображення в модулі History.

Важливим нюансом є те, що запис в БД формується саме після завершення стрімінгу, а не під час нього — оскільки до того моменту повний текст відповіді ще не відомий. Це означає, що якщо користувач закриє застосунок під час генерації, частково отримана відповідь не збережеться. Для продуктивного рівня слід реалізувати проміжне збереження, однак у рамках даної роботи це визначено як напрям подальшого розвитку.

Початкове збереження — addEntry

```dart
Future<void> addEntry(ChatLogEntry entry) async {
  final db = _db;
  if (db == null) throw StateError('ChatHistoryService.init() has not completed.');
  await db.transaction((txn) async {
    await txn.insert(_table, _row(entry));
    if (_ftsAvailable) {
      await txn.rawInsert(
        'INSERT INTO $_ftsTable(entry_id, body) VALUES (?, ?)',
        [entry.id, _ftsBody(entry)],
      );
    }
  });
  await _loadAllFromDb();
}
```

Оновлення після стрімінгу — updateEntry

```dart
Future<void> updateEntry(ChatLogEntry entry) async {
  final db = _db;
  if (db == null) throw StateError('ChatHistoryService.init() has not completed.');
  final n = await db.update(
    _table,
    _row(entry),
    where: 'id = ?',
    whereArgs: [entry.id],
  );
  if (n == 0) return;
  if (_ftsAvailable) {
    await db.rawUpdate(
      'UPDATE $_ftsTable SET body = ? WHERE entry_id = ?',
      [_ftsBody(entry), entry.id],
    );
  }
  await _loadAllFromDb();
}
```

3.3.4 Труднощі при реалізації модуля чату

Найбільше часу зайняло вирішення проблеми setState() called after dispose(), яка виникала при швидкому перемиканні між панелями під час активного стрімінгу. Суть проблеми в тому, що стрімінг є асинхронною операцією, яка продовжується навіть після того, як користувач вже перейшов на інший екран і ChatPanel був знищений. Коли черговий токен приходив після знищення віджету, спроба оновити стан призводила до помилки.

Рішення — явне скасування підписки на потік у методі dispose() через StreamSubscription.cancel(). Після цього при знищенні ChatPanel підписка скасовується, і нові токени більше не обробляються. Це класичний патерн управління ресурсами в Dart, про який легко забути під час швидкої розробки.

3.4 Реалізація модуля генерації документації (DevDoc)

3.4.1 Загальний опис та алгоритм генерації

Модуль DevDoc автоматизує перетворення фрагмента вихідного коду на структуровану технічну документацію у форматі Markdown. Це, мабуть, найцікавіший із функціональних модулів з точки зору інженерії промптів — правильна побудова промпту є ключем до отримання стабільного, передбачуваного результату.

Алгоритм генерації складається з трьох послідовних етапів.

Етап 1 — Побудова промпту. Клас GeminiPromptBuilder формує системний промпт, що містить чіткі правила структури вихідного документа. Промпт визначає обов'язкові секції: загальний опис фрагмента коду, параметри (якщо це функція або клас), опис поведінки та повернуваного значення, приклад використання та обов'язкову секцію Code Health Check із зазначенням потенційних проблем якості коду. Крім структурних правил, промпт включає вибрану мову документації та custom system prompt із налаштувань користувача. Жорстке визначення структури у системній інструкції стабілізує формат відповіді.

На початку роботи над DevDoc я спробував надсилати «вільний» промпт на кшталт «опиши цей код». Результати були абсолютно непередбачуваними: іноді модель писала prose-опис, іноді — список, іноді — таблицю, іноді — починала з роздумів про те, навіщо взагалі документувати код. Перехід до структурованого системного промпту із явним переліком секцій вирішив цю проблему і зробив результати стабільними та придатними до подальшої обробки.

Етап 2 — Виклик API. Сформований промпт разом із фрагментом коду надсилається до GeminiService.generateDocumentation() як одноразовий запит — не стрімінговий. Для DevDoc обрано саме такий підхід, оскільки цілісність структури Markdown-документа важливіша за швидкість першого токена: краще показати «Генерується...» і видати повний правильно відформатований документ, ніж стрімінгом поступово руйнувати Markdown-розмітку незавершеними тегами.

Етап 3 — Постобробка та вивід. Отриманий Markdown передається до DevDocPanel та рендериться через flutter_markdown. Секція Code Health

Check виокремлюється парсером CodeHealthParser та відображається окремим блоком із відповідним акцентним кольором. Результат зберігається до LocalHistoryRepository.

```
  static String devDocSystemInstruction({
    required String languageCode,
    String customPrompt = '',
  }) {
    final lang = languageName(languageCode);
    final base = '''
You are a Senior Software Engineer and Technical Writer.
RULES (mandatory):
- Output ONLY valid Markdown. No conversational filler.
- Always write documentation in $lang unless the user explicitly requests another language.
- ALWAYS begin with a "## Code Health Check" section containing exactly:
  - **Complexity Score:** N (integer 1-10, estimated cyclomatic complexity)
  - **Suggestion:** One brief sentence for improvement.
- After the Code Health Check section, continue with full technical documentation.
- Use JSDoc/DartDoc standards for function comments.
- Structure READMEs: Overview, Installation, Usage, API Reference.
- If code has issues, add a "## Potential Issues" section.
''';
    return withCustomPrompt(base, customPrompt);
  }
```

ЛІСТИНГ 3.7: lib/data/services/gemini_prompt_builder.dart — метод buildDevDocPrompt із повним текстом системного промпту

```
  Stream<String> streamDevDoc({
    required String sourceCode,
    required double temperature,
    required String languageCode,
    String customSystemPrompt = '',
    String? modelApiId,
  }) {
    final id = modelApiId ?? GeminiModelOption.geminiFlash.apiModelId;
    final instruction = GeminiPromptBuilder.devDocSystemInstruction(
      languageCode: languageCode,
      customPrompt: customSystemPrompt,
    );
    final model = _model(
      modelApiId: id,
      systemInstruction: instruction,
      temperature: temperature,
    );
    final prompt = '''
Analyze the following source code and produce technical documentation per your instructions.
```

ЛІСТИНГ 3.8 — GeminiService.streamDevDoc (у коді немає generateDocumentation) DevDoc використовує стрімінговий запит; DevDocPanel збирає повну відповідь через StreamingSessionController.run()

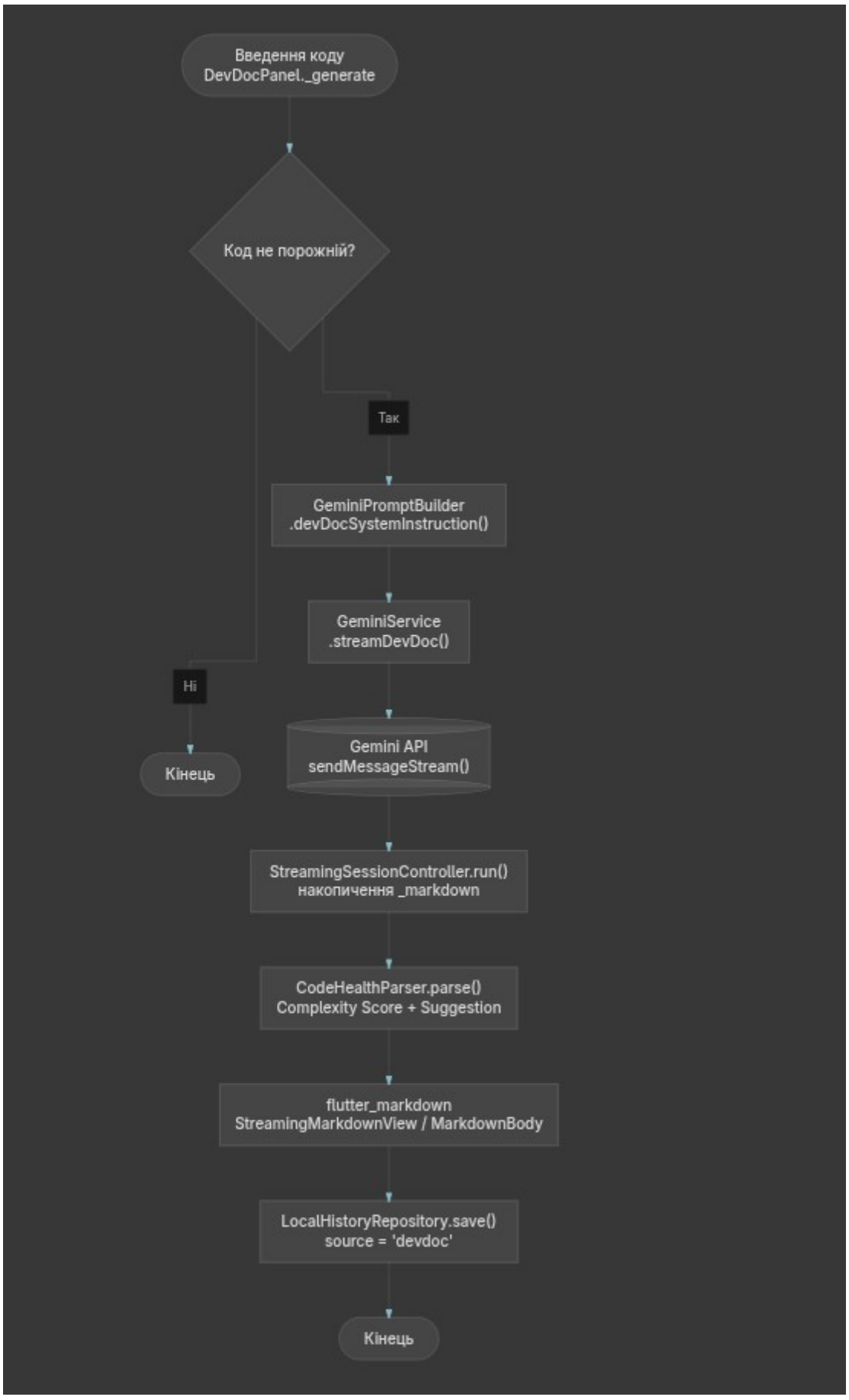


РИСУНОК 3.7 — Блок-схема алгоритму DevDoc

3.4.2 Реалізація парсингу Code Health Check

Секція Code Health Check є особливою частиною DevDoc-результату — вона містить оцінку потенційних проблем коду і повинна відображатись окремо, щоб привернути увагу. CodeHealthParser шукає у тексті відповіді маркер-заголовок секції та відокремлює її вміст. Якщо модель не включила секцію у відповідь (що трапляється при занадто коротких або тривіальних фрагментах коду), парсер повертає порожній результат без помилки — це приклад graceful degradation.

```dart
import 'package:flutter_test/flutter_test.dart';
import 'package:my_ai_app/domain/services/code_health_parser.dart';

void main() {
  test('CodeHealthParser extracts score and strips section', () {
    const md = '''
## Code Health Check
**Complexity Score:** 6
**Suggestion:** Extract nested conditionals into helper methods.

## Overview
Hello world.
''';

    final parsed = CodeHealthParser.parse(md);
    expect(parsed.health?.complexityScore, 6);
    expect(parsed.health?.suggestion, contains('helper'));
    expect(parsed.bodyMarkdown, contains('## Overview'));
    expect(parsed.bodyMarkdown, isNot(contains('Code Health Check')));
  });

  test('CodeHealthParser returns null health when section missing', () {
    const md = '## Overview\nNo health section.';
    final parsed = CodeHealthParser.parse(md);
    expect(parsed.health, isNull);
    expect(parsed.bodyMarkdown, md);
  });
}
```

ЛІСТИНГ 3.9 — test/code_health_parser_test.dart: тести парсингу Code Health

3.4.3 Реалізація експорту в Markdown та PDF

Після генерації документації користувач може зберегти результат у двох форматах. Для Markdown використовується file_picker із відкриттям системного діалогу збереження — на Linux це стандартний GTK file chooser, на Android — системний менеджер файлів.

PDF-генерація є більш складним процесом. Бібліотека pdf будує об'єктну модель документа: заголовки, параграфи, блоки коду з

моноширинним шрифтом. Виклик printing.savePdf() виконує рендеринг та запис файлу. Складність полягала у тому, що бібліотека pdf не розуміє Markdown напряму — необхідний проміжний крок конвертації. Це зайняло додатковий час на налагодження, але в результаті PDF-документи виглядають охайно та читабельно.

```
static Future<Uint8List> buildPdfBytes({
    required String markdown,
    required String title,
  }) async {
    await _ensureFonts();
    final parsed = CodeHealthParser.parse(markdown);
    final regular = _regular!;
    final bold = _bold!;
    final mono = _mono!;

    final doc = pw.Document(
      title: title,
      creator: footerText,
      theme: pw.ThemeData.withFont(base: regular, bold: bold),
    );

    doc.addPage(
      pw.MultiPage(
        pageFormat: PdfPageFormat.a4,
        margin: const pw.EdgeInsets.fromLTRB(48, 56, 48, 56),
        header: (context) {
          if (context.pageNumber > 1) {
            return pw.SizedBox(height: 8);
          }
          return pw.Column(
            crossAxisAlignment: pw.CrossAxisAlignment.start,
            children: [
              pw.Text(
                title,
                style: pw.TextStyle(
                  fontSize: 22,
                  fontWeight: pw.FontWeight.bold,
                  font: bold,
                  color: PdfColors.indigo900,
                ),
              ),
              pw.SizedBox(height: 6),
              pw.Divider(color: PdfColors.indigo200, thickness: 1),
              pw.SizedBox(height: 16),
            ],
          );
        },
        footer: (context) => pw.Row(
          mainAxisAlignment: pw.MainAxisAlignment.spaceBetween,
          children: [
            pw.Text(
```

```
              footerText,
              style: pw.TextStyle(fontSize: 9, font: regular, color:
PdfColors.grey600),
            ),
            pw.Text(
              'Page ${context.pageNumber} of ${context.pagesCount}',
              style: pw.TextStyle(fontSize: 9, font: regular, color:
PdfColors.grey600),
            ),
          ],
        ),
        build: (context) {
          final widgets = <pw.Widget>[];
          if (parsed.health != null) {
            widgets
              ..add(_healthBox(parsed.health!, regular: regular, bold:
bold))
              ..add(pw.SizedBox(height: 16));
          }
          widgets.addAll(
            _markdownWidgets(
              parsed.bodyMarkdown,
              regular: regular,
              bold: bold,
              mono: mono,
            ),
          );
          return widgets;
        },
      ),
    );

    return doc.save();
  }
```

ЛІСТИНГ 3.10 — lib/data/services/devdoc_pdf_exporter.dart: ключовий фрагмент побудови PDF

Основна логіка зосереджена в buildPdfBytes(): парсинг Markdown → документ pw.Document → сторінка з header/footer → Code Health box → конвертація Markdown у віджети → doc.save().

```
static pw.Widget _healthBox(
    CodeHealthCheck health, {
    required pw.Font regular,
    required pw.Font bold,
  }) {
    final score = health.complexityScore;
    final suggestion = health.suggestion;
    final PdfColor barColor = score <= 3
        ? PdfColors.green700
```

```
        : score <= 7
            ? PdfColors.amber800
            : PdfColors.red800;

    return pw.Container(
      padding: const pw.EdgeInsets.all(12),
      decoration: pw.BoxDecoration(
        border: pw.Border.all(color: barColor, width: 1.2),
        borderRadius: pw.BorderRadius.circular(8),
        color: PdfColors.grey100,
      ),
      child: pw.Column(
        crossAxisAlignment: pw.CrossAxisAlignment.start,
        children: [
          pw.Text(
            'Code Health Check',
            style: pw.TextStyle(fontWeight: pw.FontWeight.bold, font:
bold, color: barColor),
          ),
          pw.SizedBox(height: 4),
          pw.Text(
            'Cyclomatic Complexity: $score / 10',
            style: pw.TextStyle(font: regular, fontSize: 11),
          ),
          pw.SizedBox(height: 4),
          pw.Text(
            suggestion,
            style: pw.TextStyle(font: regular, fontSize: 11),
          ),
        ],
      ),
    );
  }
```

Додатково — блок Code Health у PDF (_healthBox)

3.5 Реалізація локального сховища даних

3.5.1 Ініціалізація бази даних

Ініціалізація SQLite відбувається при першому зверненні до ChatHistoryService. Метод _initDatabase() відкриває або створює файл бази даних та виконує DDL-скрипт створення таблиць. Версіонування схеми через механізм onUpgrade дозволяє додавати нові поля до таблиць без втрати наявних даних при оновленні застосунку — це важливо для продуктивного використання.

3.5.2 Проблема несумісності FTS5 та її вирішення

Однією з найнеочікуваніших технічних проблем під час розробки виявилась несумісність модуля повнотекстового пошуку FTS5 на окремих Android-пристроях. FTS5 — це розширення SQLite для ефективного

повнотекстового пошуку, яке ідеально підходить для пошуку по тексту збережених повідомлень та документів. Я розраховував використовувати його як основний механізм пошуку і навіть написав відповідні тести.

Проблема виявилась під час тестування на реальному Android-пристрої: застосунок падав при старті з помилкою no such module: fts5. Виявилось, що наявність FTS5 залежить від того, з якими параметрами компіляції було зібрано системний пакет SQLite на конкретному Android-пристрої. На більшості сучасних пристроїв FTS5 присутній, але на певних старих або специфічних конфігураціях — ні. Ця ситуація описана у документації SQLite, але я не звернув на неї увагу на початку.

Вирішення — захисна ініціалізація з перехопленням виключення. Якщо спроба створити FTS5-таблицю зазнає невдачі, система встановлює внутрішній прапор _useFts = false та переходить до альтернативного режиму пошуку на основі SQL-оператора LIKE. Пошук через LIKE є менш ефективним при великих обсягах даних, але для типового обсягу особистої історії взаємодій ця різниця несуттєва.

```
// Захисна ініціалізація (псевдокод):
try {
 await db.execute('CREATE VIRTUAL TABLE history_fts USING fts5(...)');
 _useFts = true;
} catch (DatabaseException e) {
 // FTS5 недоступний — продовжуємо з LIKE-пошуком
 _useFts = false;
}
```

Частина 1 — ініціалізація БД і FTS5 (init, _createFts, _detectFtsAvailability)

```
Future<void> init({Directory? forTestingDocumentsDir}) async {
    _documentsDir =
        forTestingDocumentsDir ?? await
getApplicationDocumentsDirectory();
    final dir = _documentsDir!;
    _db = await openDatabase(
      p.join(dir.path, _dbName),
      version: _dbVersion,
      onCreate: (db, version) async {
        await db.execute('''
CREATE TABLE $_table (
  id TEXT PRIMARY KEY NOT NULL,
  created_at_ms INTEGER NOT NULL,
  model_api_id TEXT NOT NULL,
  tone_key TEXT NOT NULL,
  user_message TEXT NOT NULL,
```

```
  assistant_message TEXT NOT NULL
)
''');
        _ftsAvailable = await _createFts(db);
      },
      onUpgrade: (db, oldVersion, newVersion) async {
        if (oldVersion < 2) {
          final created = await _createFts(db);
          if (created) {
            await db.execute('''
INSERT INTO $_ftsTable(entry_id, body)
SELECT id, user_message || ' ' || assistant_message FROM $_table
''');
          }
        }
      },
    );
    await _detectFtsAvailability();
    await _migrateLegacyJsonIfNeeded(dir);
    await _loadAllFromDb();
  }

  static Future<bool> _createFts(DatabaseExecutor db) async {
    try {
      await db.execute('''
CREATE VIRTUAL TABLE $_ftsTable USING fts5(
  entry_id UNINDEXED,
  body
)
''');
      return true;
    } on DatabaseException {
      return false;
    }
  }

  Future<void> _detectFtsAvailability() async {
    final db = _db;
    if (db == null) return;
    final rows = await db.query(
      'sqlite_master',
      columns: const ['name'],
      where: 'type = ? AND name = ?',
      whereArgs: ['table', _ftsTable],
      limit: 1,
    );
    _ftsAvailable = rows.isNotEmpty;
  }
```

Частина 2 — умовний пошук за _ftsAvailable (searchEntries)

```
Future<List<ChatLogEntry>> searchEntries(String rawQuery) async {
    final db = _db;
```

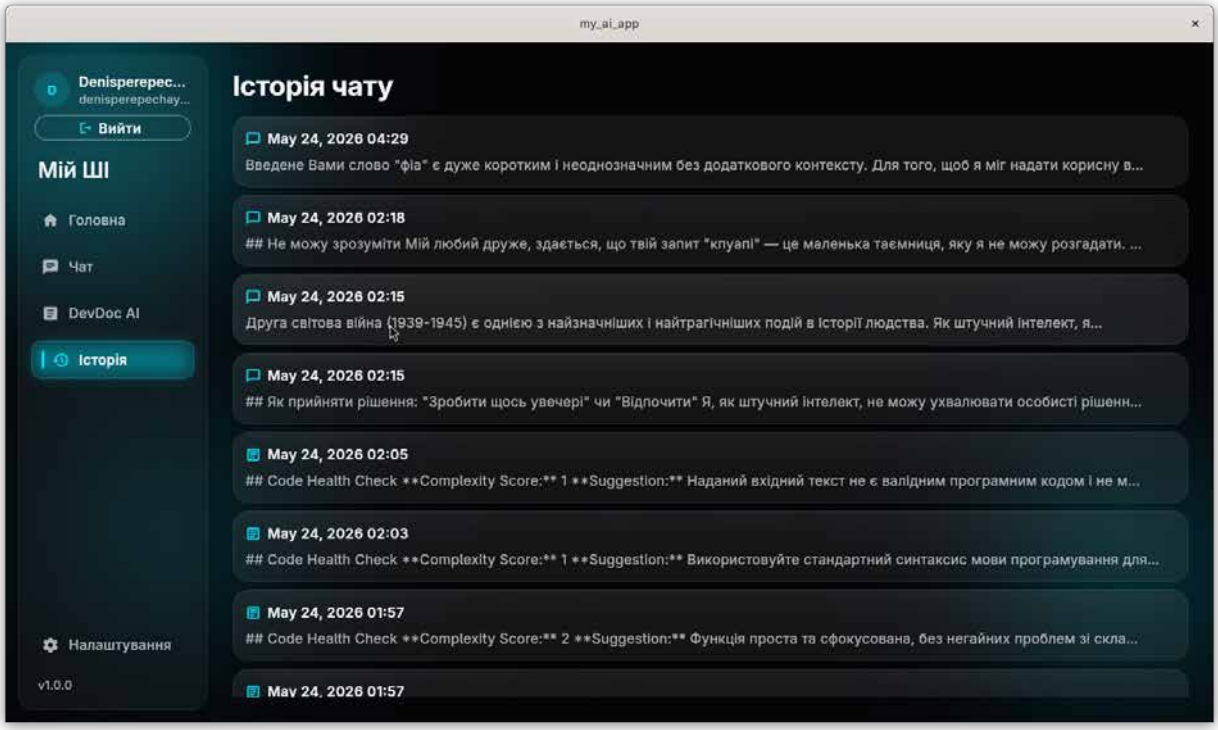
```
    if (db == null) throw StateError('ChatHistoryService.init() has not
completed.');
    final query = rawQuery.trim();
    if (query.isEmpty) {
      return List<ChatLogEntry>.from(entries);
    }
    if (!_ftsAvailable) {
      final escaped = query
          .replaceAll('\\', r'\\')
          .replaceAll('%', r'\%')
          .replaceAll('_', r'\_');
      final likeArg = '%$escaped%';
      final rows = await db.query(
        _table,
        where:
            '(user_message LIKE ? ESCAPE \'\\\') OR (assistant_message
LIKE ? ESCAPE \'\\\')',
        whereArgs: [likeArg, likeArg],
        orderBy: 'created_at_ms ASC',
      );
      final out = <ChatLogEntry>[];
      for (final row in rows) {
        final e = _entryFromRow(row);
        if (e != null) out.add(e);
      }
      return out;
    }
    final match = buildFtsMatchQuery(query);
    if (match.isEmpty) {
      return List<ChatLogEntry>.from(entries);
    }
    try {
      final rows = await db.rawQuery(
        '''
SELECT e.id, e.created_at_ms, e.model_api_id, e.tone_key, e.user_message,
e.assistant_message
FROM $_table AS e
WHERE e.id IN (
  SELECT entry_id FROM $_ftsTable WHERE $_ftsTable MATCH ?
)
ORDER BY e.created_at_ms ASC
''',
        [match],
      );
      final out = <ChatLogEntry>[];
      for (final row in rows) {
        final e = _entryFromRow(row);
        if (e != null) out.add(e);
      }
      return out;
    } on DatabaseException {
      _ftsAvailable = false;
      return searchEntries(query);
```

```
    }
  }
```

ЛІСТИНГ 3.11: lib/services/chat_history_service.dart — метод _initDatabase із захисною ініціалізацією FTS5 та умовним пошуком залежно від прапора _useFts

3.5.3 CRUD-операції та управління даними

LocalHistoryRepository реалізує повний набір операцій: insertDocument для додавання нового запису, getAllDocuments для отримання списку із сортуванням за датою, getDocumentById для повного завантаження конкретного запису, deleteDocument для видалення та searchDocuments для пошуку — з FTS5 або через LIKE залежно від доступності. Усі операції виконуються асинхронно через Future<T>, що унеможливлює блокування потоку інтерфейсу.



3.5.4 Збереження налаштувань через SharedPreferences

Параметри, що не потребують реляційної структури — акцентний колір, масштаб шрифту, температура моделі, custom system prompt, стан автентифікації, поточна локаль — зберігаються через SharedPreferences. AppSettingsController завантажує їх при ініціалізації та записує при підтвердженні змін через SettingsPanel. Завдяки прив'язці контролера до ListenableBuilder інтерфейс оновлюється реактивно без ручних викликів setState — зміна акцентного кольору відображається миттєво по всьому застосунку.

3.6 Реалізація модуля налаштувань

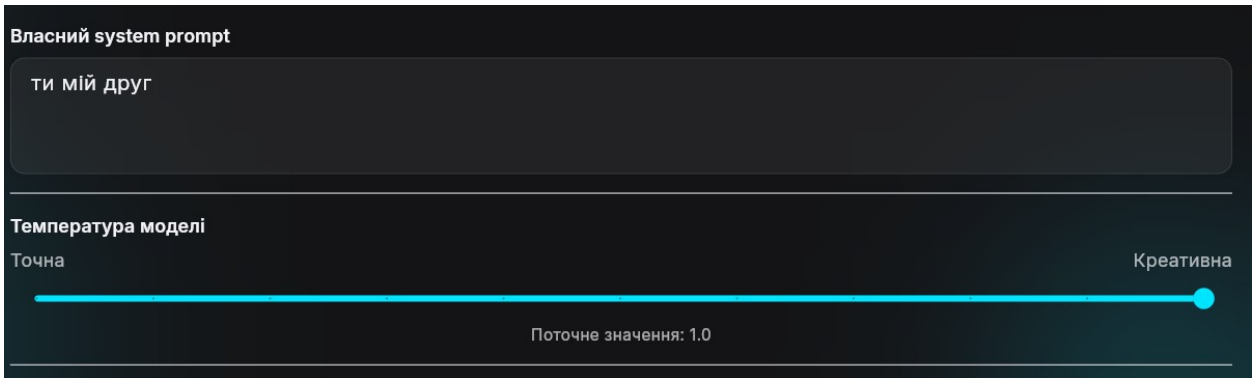
3.6.1 Параметри налаштувань

SettingsPanel надає користувачеві контроль над: акцентним кольором інтерфейсу; масштабом шрифту; інтенсивністю ефекту glassmorphism; температурою моделі (від 0.0 до 1.0 — від детермінованих до більш творчих відповідей); власною системною інструкцією для моделі (custom system prompt); мовою документації (en / uk / ru / sv).

3.6.2 Відкладена модель редагування

Реалізовано концепцію «dirty state» — відкладеного застосування змін. Зміни в UI відображаються у попередньому режимі, але зберігаються лише після натискання кнопки «Прийняти зміни». Це запобігає випадковому збереженню проміжних станів. Технічно реалізовано через тимчасовий об'єкт налаштувань, що замінює основний лише при підтвердженні.

Цей підхід виявився нетривіальним у реалізації: виникала проблема, коли налаштування температури моделі застосовувались у GeminiService ще до натискання кнопки підтвердження, бо сервіс зчитував параметр безпосередньо з контролера. Вирішення — GeminiService зчитує налаштування з «підтвердженої» копії, а не з «редакційної»



3.7 Реалізація модуля журналу взаємодій (History)

Модуль журналу надає доступ до усіх попередніх результатів роботи — як чат-діалогів, так і DevDoc-документів. GenerationHistoryPanel завантажує список записів з LocalHistoryRepository та відображає їх у вигляді карток із preview-текстом, датою та іконкою джерела. При натисканні на запис відкривається повний Markdown-текст через flutter_markdown.

Наявність локальної бази взаємодій дозволяє користувачеві не втрачати результати попередніх сесій — принципова відмінність від більшості веб-чат-сервісів. При розробці я звернув увагу, що завантаження повного тексту усіх записів одразу при відкритті History може бути повільним при великій кількості записів. Тому реалізовано «ліниве» завантаження: при відкритті списку зчитуються лише поля preview, createdAt та source, а повний markdownContent завантажується лише при виборі конкретного запису.

3.8 Реалізація голосового введення

Модуль голосового вводу є додатковою функцією. SpeechEnabledInputBar дозволяє натиснути кнопку мікрофона, продиктувати запит та отримати автоматично підставлений текст у поле чату. Записаний аудіофайл передається до GeminiAudioTranscriber, який надсилає його до Gemini API для транскрипції.

Реалізація голосового вводу виявилась нетривіальною через відмінності в поведінці пакетів між Linux та Android. На Linux-платформі стандартний speech_to_text не працює без системного демона розпізнавання мовлення. Тому для desktop обрано підхід через запис аудіофайлу та відправку до Gemini API — таким чином уніфікована поведінка на обох платформах.

3.9 Реалізація локалізації інтерфейсу

Підтримка чотирьох мов реалізована через стандартний Flutter-механізм ARB-файлів. Файли app_uk.arb, app_en.arb, app_ru.arb, app_sv.arb містять пари ключ–переклад для всіх текстових рядків інтерфейсу. Команда flutter gen-l10n автоматично генерує типізований клас AppLocalizations, що дозволяє звертатися до перекладів зі строгою типізацією.

Цікавий момент: при додаванні нового рядка в інтерфейс я іноді забував додати відповідний ключ до всіх чотирьох ARB-файлів, через що при перемиканні на іншу мову застосунок падав із MissingLocalization помилкою. Після кількох таких інцидентів я виробив звичку одразу додавати ключ до

всіх файлів локалізації, навіть якщо переклад тимчасово є лише на одній мові.

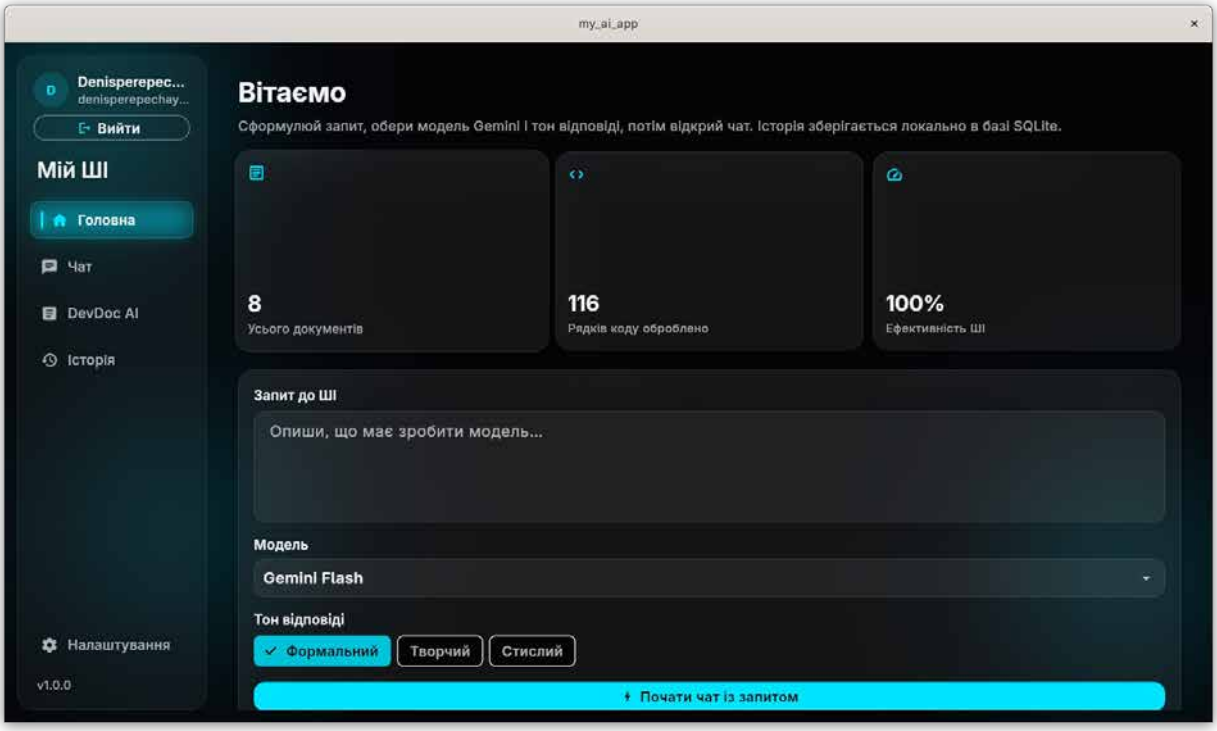
Структура ARB (коротко):

Елемент	Приклад	Призначення
`@@locale`	`"uk"`	Код мови файлу
`"key": "value"`	`"navChat": "Чат"`	Простий рядок інтерфейсу
`{placeholder}`	`{score}`, `{id}`	Підстановка значень у runtime
`@key + placeholders`	`@devDocHealthScore`	Метадані: типи параметрів для генератора

3.10 Візуальна система та glassmorphism

Дизайн-концепція застосунку базується на темному інтерфейсі з ефектом скла — напівпрозорі панелі з розмиттям фону надають застосунку сучасний вигляд. Ключові компоненти: GlassBackground (фоновий шар з blur), AppContentArea, AppSidebar, GlowButton.

Акцентний колір та інтенсивність blur є налаштовуваними параметрами. Реалізація glassmorphism через Flutter-стек BackdropFilter виявилась тривіальною технічно, але вимагала ретельного підбору значень blur та прозорості для кожного компонента — щоб ефект виглядав охайно, а не просто «розмивав усе у кашу». Я витратив кілька годин на підбір параметрів, переглядаючи різні варіанти.



3.11 Кросплатформна реалізація

3.11.1 Desktop (Linux)

Основна складність при реалізації Linux-версії — відсутність нативної підтримки SQLite у sqflite для desktop-платформ. Стандартний sqflite покладається на JNI-міст на Android, якого не існує на Linux. Рішення — sqflite_common_ffi, що реалізує той самий API через FFI до системної або вбудованої бібліотеки SQLite.

Виклик sqfliteFfiInit() повинен відбуватися строго до першого відкриття з'єднання з базою. Я витратив кілька годин на налагодження MissingPluginException, поки не знайшов правильне місце для цього виклику у main.dart. Після цього все запрацювало стабільно.

3.11.2 Android

На Android sqflite працює зі штатним системним SQLite без додаткових ініціалізацій. Команда flutter build apk --release --split-per-abi генерує окремі APK для кожної апаратної архітектури (arm64-v8a, armeabi-v7a, x86_64), що дозволяє зменшити розмір встановленого застосунку для кінцевого користувача.

Окрема складність — дозволи на Android. Запис файлів та доступ до мікрофону потребують явного запиту у runtime. file_picker та record виконують ці запити автоматично, але лише за умови правильного оголошення у AndroidManifest.xml. Я пропустив оголошення дозволу на запис аудіо, через що голосовий ввід мовчки не працював на реальному Android-пристрої — без помилки, просто нічого не записувалось. Налагодження цього зайняло кілька годин.

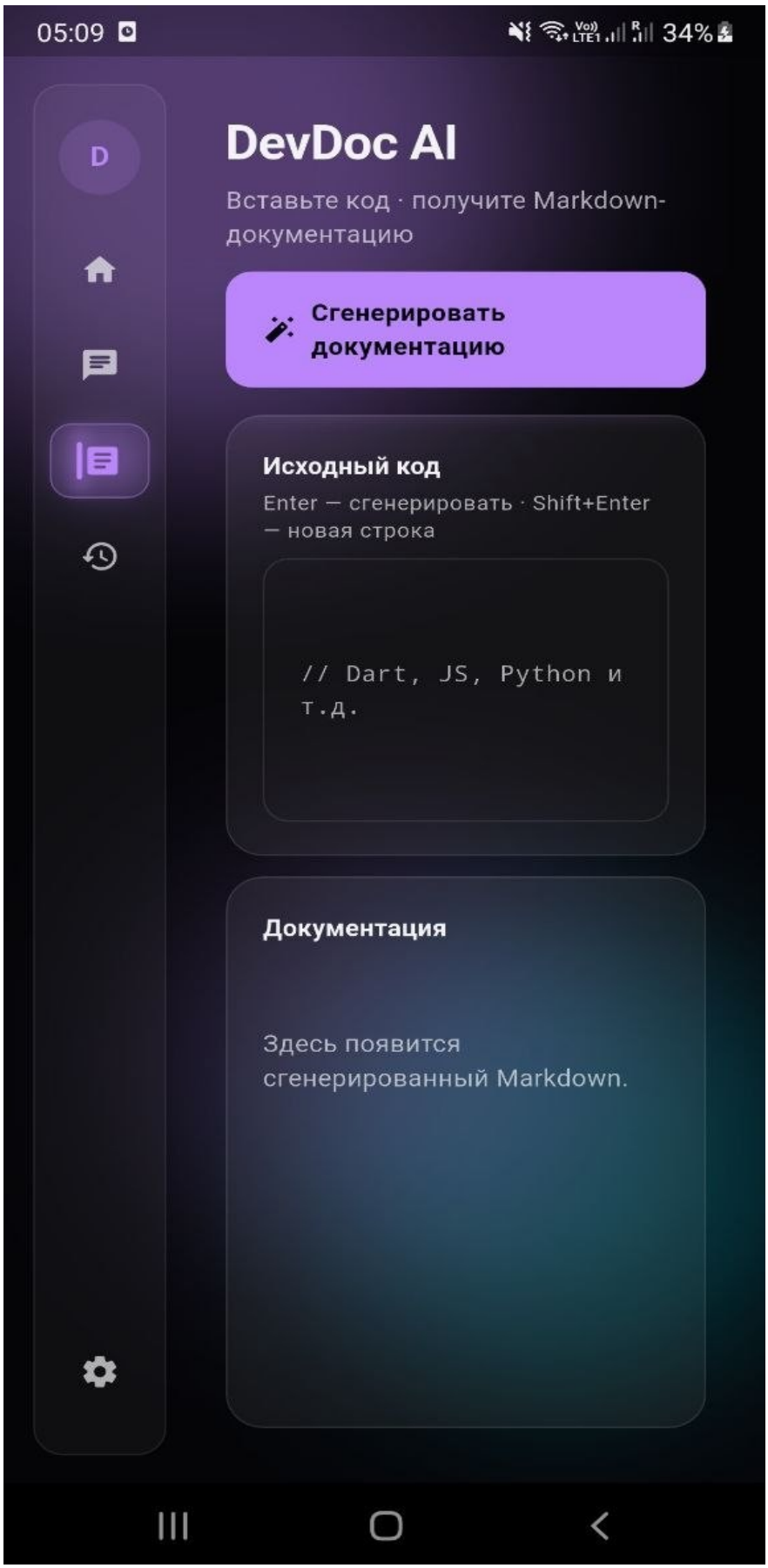


РИСУНОК 3.13: Скріншот застосунку на Android

Висновки до розділу 3. Реалізовано усі функціональні модулі застосунку: двоетапна автентифікація, AI-чат зі стрімінгом, DevDoc-

генератор, локальна база даних із захисним підходом до FTS5, панель налаштувань із відкладеним застосуванням змін, журнал взаємодій та голосовий ввід. У процесі реалізації виникли і були вирішені характерні технічні труднощі: несумісність FTS5 на окремих Android-пристроях, порядок ініціалізації sqflite_ffi на Linux, управління підписками на стрімінгові потоки, відмінності у поведінці голосового вводу між платформами та проблеми з дозволами на Android. Застосована модульна архітектура дозволила ізолювати ці проблеми в конкретних сервісах без поширення ефекту на решту системи.

РОЗДІЛ 4. ТЕСТУВАННЯ, ВАЛІДАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Мета та стратегія тестування

Тестування є обов'язковим етапом циклу розробки. Метою тестування у даній роботі є перевірка коректності бізнес-логіки ключових модулів, стабільності UI-станів при переходах між екранами та відсутності критичних дефектів у базових сценаріях. Стратегія включала три рівні: статичний аналіз коду, автоматизовані unit/widget-тести та ручне сценарне тестування на цільових платформах.

4.2 Статичний аналіз коду

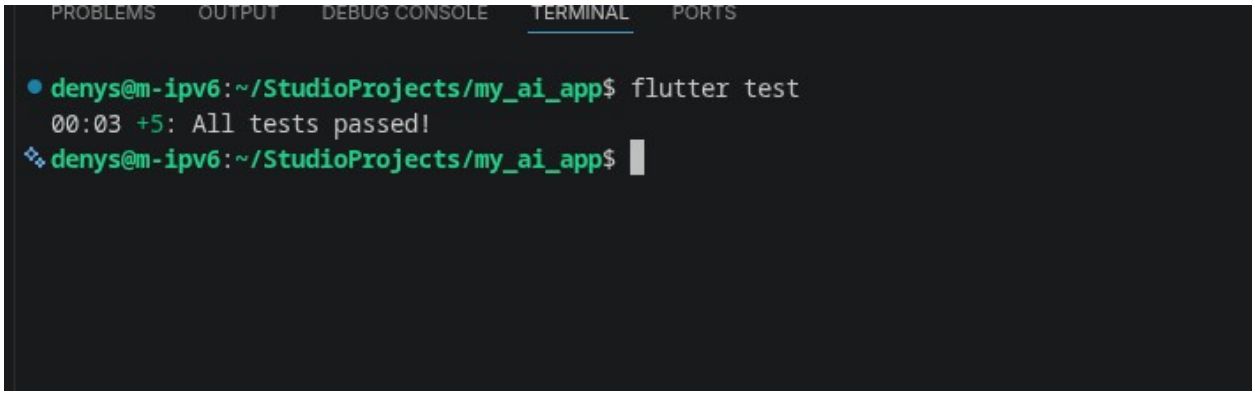
Першим рівнем контролю якості є статичний аналіз через dart analyze. Цей інструмент перевіряє синтаксичну коректність, типові помилки типізації та порушення lint-правил. Запуск: dart analyze lib test. За результатами фінальної перевірки аналізатор не виявив жодної помилки чи попередження.

4.3 Автоматизовані тести

Unit-тест CRUD і FTS (test/chat_history_service_test.dart) — перевіряє коректність вставки запису, отримання за ідентифікатором, пошуку та видалення. Запускається в ізольованому in-memory середовищі.

Unit-тест парсингу Code Health (test/code_health_parser_test.dart) — перевіряє виокремлення секції Code Health Check із Markdown-відповіді. Тестові дані включають правильно сформований Markdown, Markdown без секції та порожній рядок.

Widget-тест рендеру (test/widget_test.dart) — мінімальний smoke-тест, що підтверджує відсутність fatal-помилок ініціалізації.



4.4 Ключові тест-кейси ручного тестування

Таблиця 4.1 — Результати ручного тестування

№	Тест-кейс	Очікуваний результат	Статус
TC-01	Холодний старт застосунку	`LoginView` без помилок	✓
TC-02	Авторизація mock OTP	Перехід до `AppShell`	✓
TC-03	AI-чат, стрімінг	Поступовий вивід, запис у БД	✓
TC-04	DevDoc генерація	Markdown + Code Health Check	✓
TC-05	Експорт Markdown	Файл збережено (`.md`)	✓
TC-06	Експорт PDF	PDF збережено	✓
TC-07	Зміна мови документації	Документ іншою мовою	✓
TC-08	Перемикання локалі UI	Тексти інтерфейсу оновлено	✓
TC-09	Перегляд History	Markdown відображається	✓
TC-10	Зміна акцентного кольору	Колір змінено по всьому UI	✓

4.5 Виявлені проблеми та виправлення

У процесі розробки та тестування виявлено кілька груп проблем, що були усунені до фінального захисту.

Асинхронні стани. При швидкому перемиканні між панелями під час активного стрімінгу виникали помилки setState() called after dispose(). Рішення: явне скасування підписок у dispose()-методах Panel-віджетів.

Відображення Markdown. Відступи між блоками повідомлень чату виявились нерівномірними при комбінаціях тексту та блоків коду. Вирішено налаштуванням стилів flutter_markdown.

FTS5-несумісність. Описана в підрозділі 3.5.2. Вирішено захисною ініціалізацією.

Налаштування температури. При застосуванні нового значення температури без перезапуску сесії GeminiService підхоплював старе значення. Вирішено: сервіс зчитує налаштування з «підтвердженої» копії контролера при кожному виклику.

Висновки до розділу 4. Проведено три рівні тестування: статичний аналіз (без помилок), автоматизовані тести (пройдені), ручне тестування (10 тест-кейсів, усі виконані). Виявлені дефекти усунені. Застосунок відповідає поставленим функціональним вимогам.

РОЗДІЛ 5. ПРАКТИЧНІ АСПЕКТИ ВПРОВАДЖЕННЯ

5.1 Сценарії реального використання

Студент або викладач — формування технічних пояснень до лабораторних робіт, підготовка README для навчальних проєктів, генерація структурованих описів алгоритмів.

Junior або Middle розробник — автоматична генерація первинних doc-блоків для нового коду, швидкі уточнення через чат без перемикання на браузер, накопичення рішень у локальній базі.

Командна робота — локальне збереження документаційних шаблонів, що часто використовуються; повторне використання раніше згенерованих описів як основи.

5.2 Процес розгортання

Linux:

```
flutter pub get
flutter build linux --release
# Результат: build/linux/x64/release/bundle/
```

Android:

```
flutter build apk --release --split-per-abi
# Результат: build/app/outputs/flutter-apk/
```

5.3 Вимоги до середовища

Flutter SDK (Dart 3.11.x); для Android — Android SDK; файл .env із GEMINI_API_KEY та AUTH_OTP_MOCK_MODE=true (або SMTP-параметри для реального OTP).

5.4 Обмеження та перспективи розвитку

Поточна версія є прототипом рівня MVP. Серед обмежень: відсутність серверної синхронізації між пристроями; OTP у production потребує SMTP; базове тестове покриття. Перспективні напрями: backend із синхронізацією; CI/CD пайплайн; публікація в Google Play та Linux-репозиторії; інтеграція RAG із локальною базою знань.

Висновки до розділу 5. Застосунок готовий до практичного використання в навчальному та демонстраційному контексті. Наявні обмеження є характерними для MVP і визначені як пріоритети подальшого розвитку.

РОЗДІЛ 6. ОХОРОНА ПРАЦІ, ІНФОРМАЦІЙНА БЕЗПЕКА ТА ЕТИЧНІ АСПЕКТИ

6.1 Охорона праці при розробці

Розробка застосунку пов'язана з тривалою роботою за персональним комп'ютером. Відповідно до нормативних вимог, монітор повинен знаходитись на відстані 60–70 см від очей, верхній край екрана — на рівні лінії зору. Рекомендується дотримуватись правила 20-20-20: кожні 20 хвилин — 20-секундна пауза з фокусуванням на предметі за 6 метрів. Під час розробки застосовувались перерви кожні 45–60 хвилин, що відповідає рекомендаціям щодо захисту зору.

6.2 Інформаційна безпека

Зберігання API-ключів. Ключ доступу до Gemini API виноситься у .env-файл, що виключений з репозиторію. Для production рекомендується централізований key vault або секрети CI/CD.

Конфіденційність коду. Вихідний код, що вставляється для DevDoc-генерації, надсилається до зовнішнього API. Для корпоративного використання рекомендується або локальна мовна модель, або механізм очищення коду.

OTP. Одноразові паролі зберігаються лише в пам'яті застосунку, що виключає компрометацію через атаку на локальне сховище.

6.3 Етичні аспекти використання AI

При використанні LLM необхідно враховувати: можливість «галюцинацій» — формально правильних, але фактично некоректних відповідей; питання авторства та позначення AI-генерованого контенту; конфіденційність при вставці комерційного коду. Результати, отримані через my_ai_app, слід розглядати як чернетку для рев'ю, а не як остаточну відповідь.

Висновки до розділу 6. Реалізація врахувала базові вимоги інформаційної безпеки. Для production-рівня необхідно додати шифрування локального сховища та централізований key vault.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі виконано повний цикл розробки кросплатформного AI-застосунку my_ai_app — від аналізу предметної області та формування вимог до реалізації, тестування та оцінки практичного ефекту.

Основні результати роботи:

1. Проведено аналіз наявних рішень у сфері AI-інструментів для розробників і підтверджено відсутність кросплатформного застосунку, що поєднував би AI-чат, генерацію документації та локальне зберігання в єдиному середовищі.
2. Реалізовано модульну архітектуру з п'ятьма шарами відповідальності (presentation, domain, data, state, config), що забезпечує розширюваність та незалежність бізнес-логіки від конкретних технологій.
3. Розроблено AI-чат із потоковим відображенням відповіді через Stream<String>, що мінімізує суб'єктивне відчуття затримки під час генерації.
4. Реалізовано модуль DevDoc для автоматичної генерації структурованої Markdown-документації за фрагментами вихідного коду із вбудованою секцією аналізу якості коду.
5. Розроблено систему локального зберігання усіх результатів через SQLite із захисним вирішенням проблеми несумісності FTS5 на окремих Android-пристроях.
6. Реалізовано двоетапну автентифікацію через OTP, багатомовний інтерфейс (uk/en/ru/sv) та гнучке налаштування параметрів моделі й візуального стилю.

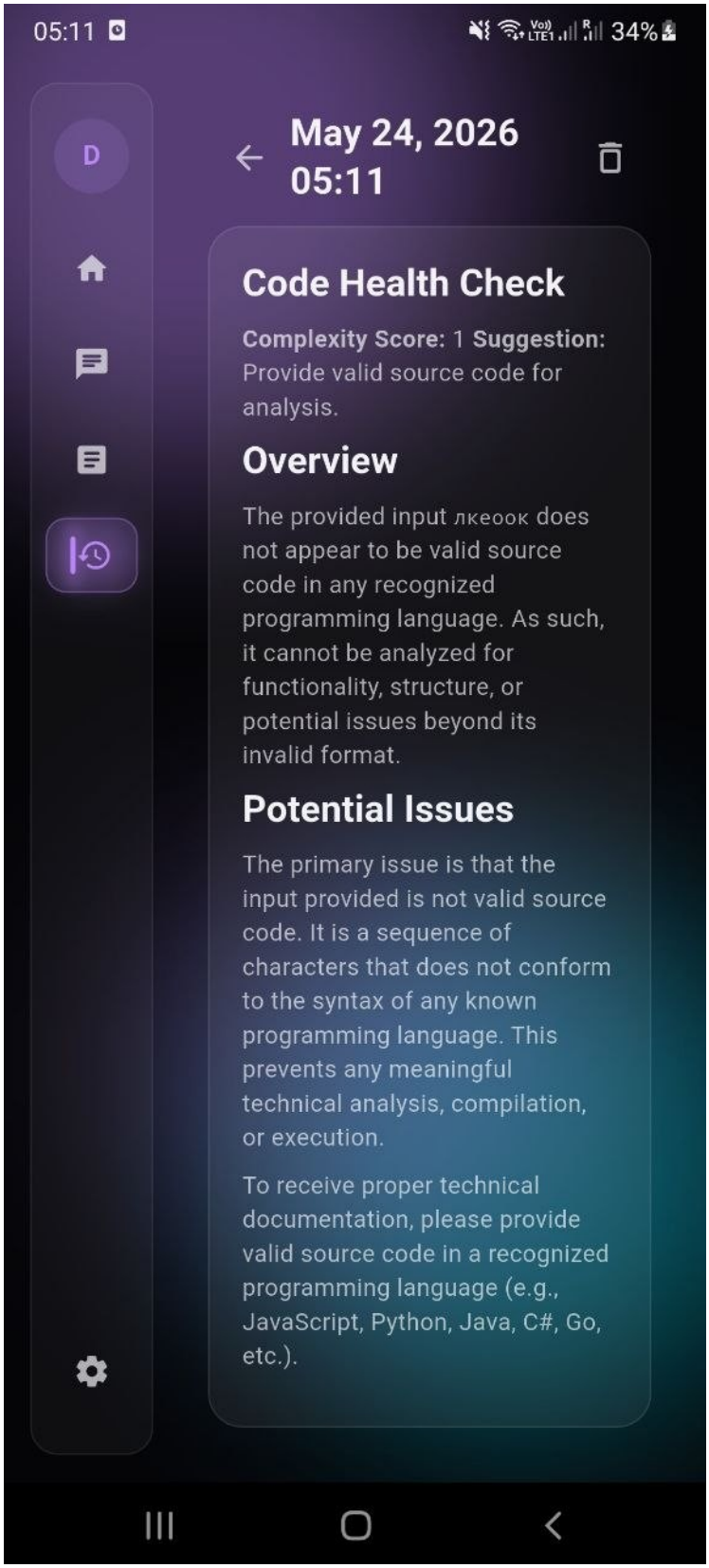
Таким чином, поставлена мета досягнута, а задачі бакалаврської роботи виконані в повному обсязі. Отриманий програмний продукт є працездатним і демонструє перспективність подальшого розвитку до production-рішення.

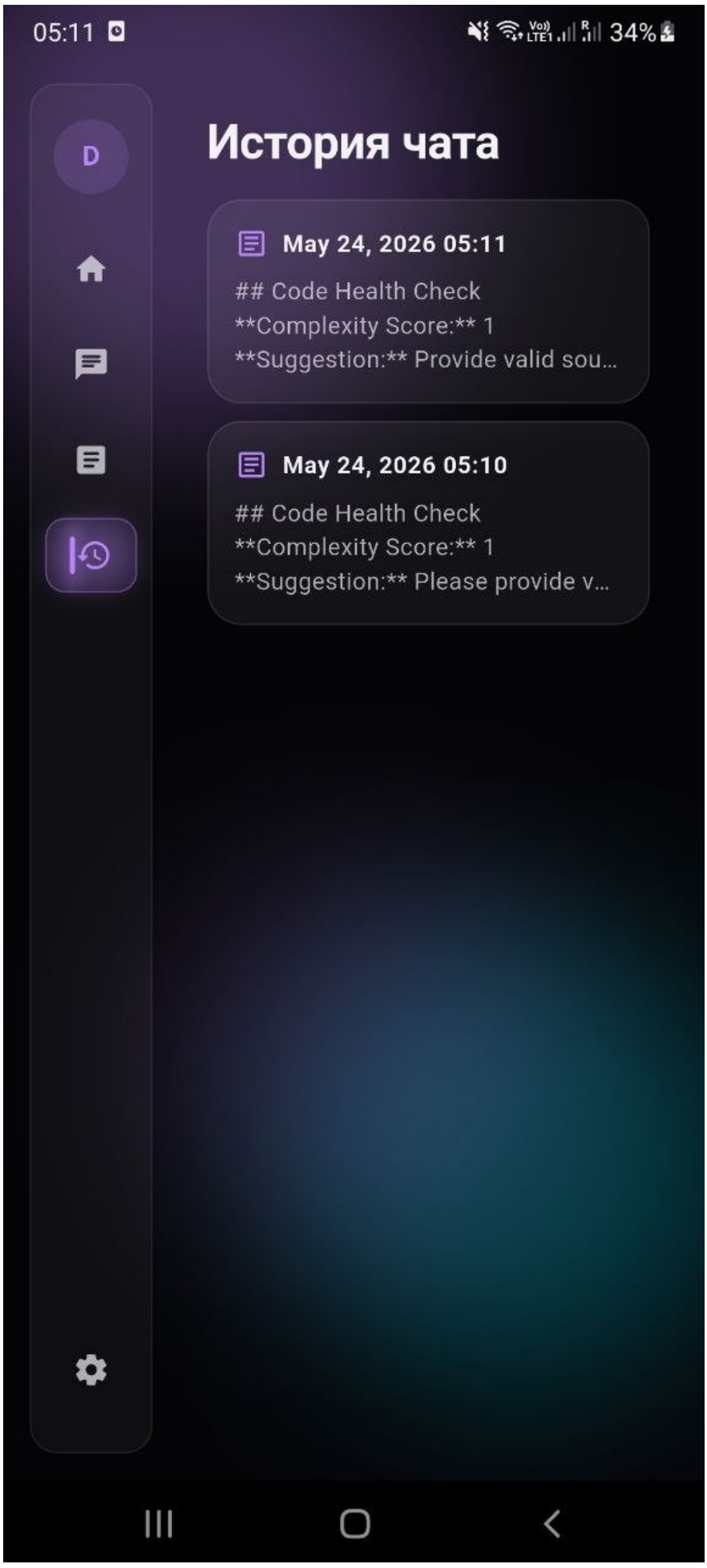
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

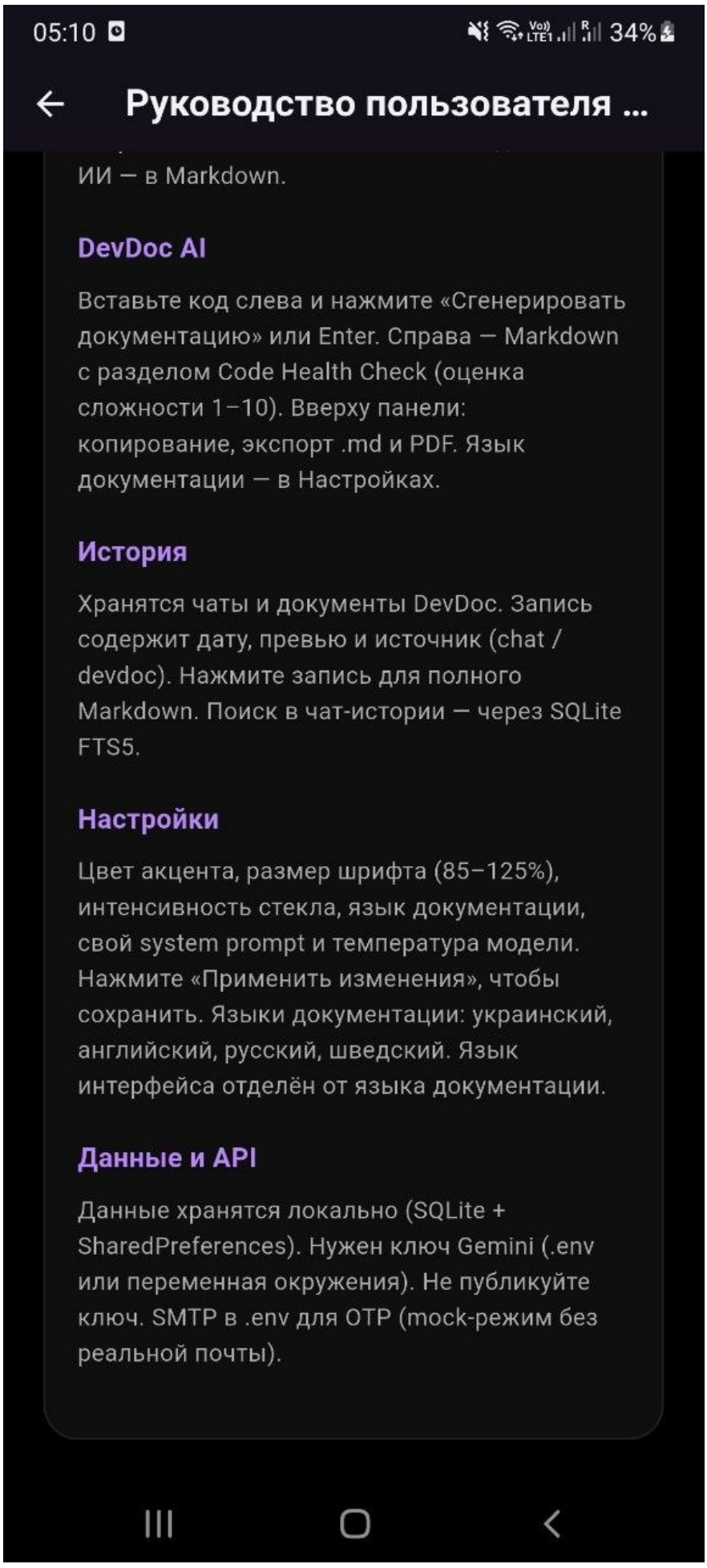
1. Flutter SDK Documentation. URL: https://docs.flutter.dev
2. Dart Language Programming Guide. URL: https://dart.dev/guides
3. Google Gemini API Developer Platform. URL: https://ai.google.dev
4. Google Generative AI Dart Package (GitHub). URL: https://github.com/google/generative-ai-dart
5. Sqflite Database Plugin for Flutter. URL: https://github.com/tekartik/sqflite
6. Sqflite Common FFI (SQLite for Desktop). URL: https://github.com/tekartik/sqflite_common_ffi
7. SQLite Database Engine Documentation. URL: https://sqlite.org/docs.html
8. SQLite Full-Text Search (FTS5) Extension. URL: https://sqlite.org/fts5.html
9. Shared Preferences Plugin (Persistent Storage). URL: https://pub.dev/packages/shared_preferences
10. Flutter Markdown Renderer. URL: https://github.com/flutter/packages/tree/main/packages/flutter_markdown
11. PDF Creation Library for Dart. URL: https://github.com/DavBfr/dart_pdf
12. Printing and PDF Preview Library. URL: https://github.com/DavBfr/dart_pdf/tree/master/printing
13. File Picker Native Dialogs. URL: https://github.com/miguelpruivo/flutter_file_picker
14. Mailer SMTP Library for Dart. URL: https://github.com/dart-mailer/mailer
15. Flutter Internationalization Guidelines. URL: https://docs.flutter.dev/ui/internationalization
16. Flutter Desktop Integration (Linux). URL: https://docs.flutter.dev/platform-integration/desktop
17. Android Developers Platform Documentation. URL: https://developer.android.com
18. Speech-to-Text Recognition Plugin. URL: https://github.com/csdcorp/speech_to_text
19. Audio Recording Library for Flutter. URL: https://github.com/gaetschwartz/record
20. Dart Internationalization (Intl) Package. URL: https://github.com/dart-lang/intl
21. Stack Overflow: Flutter & SQLite Development Community. URL: https://stackoverflow.com/questions/tagged/flutter+sqlite
22. Reddit: r/FlutterDev Community. URL: https://www.reddit.com/r/FlutterDev/

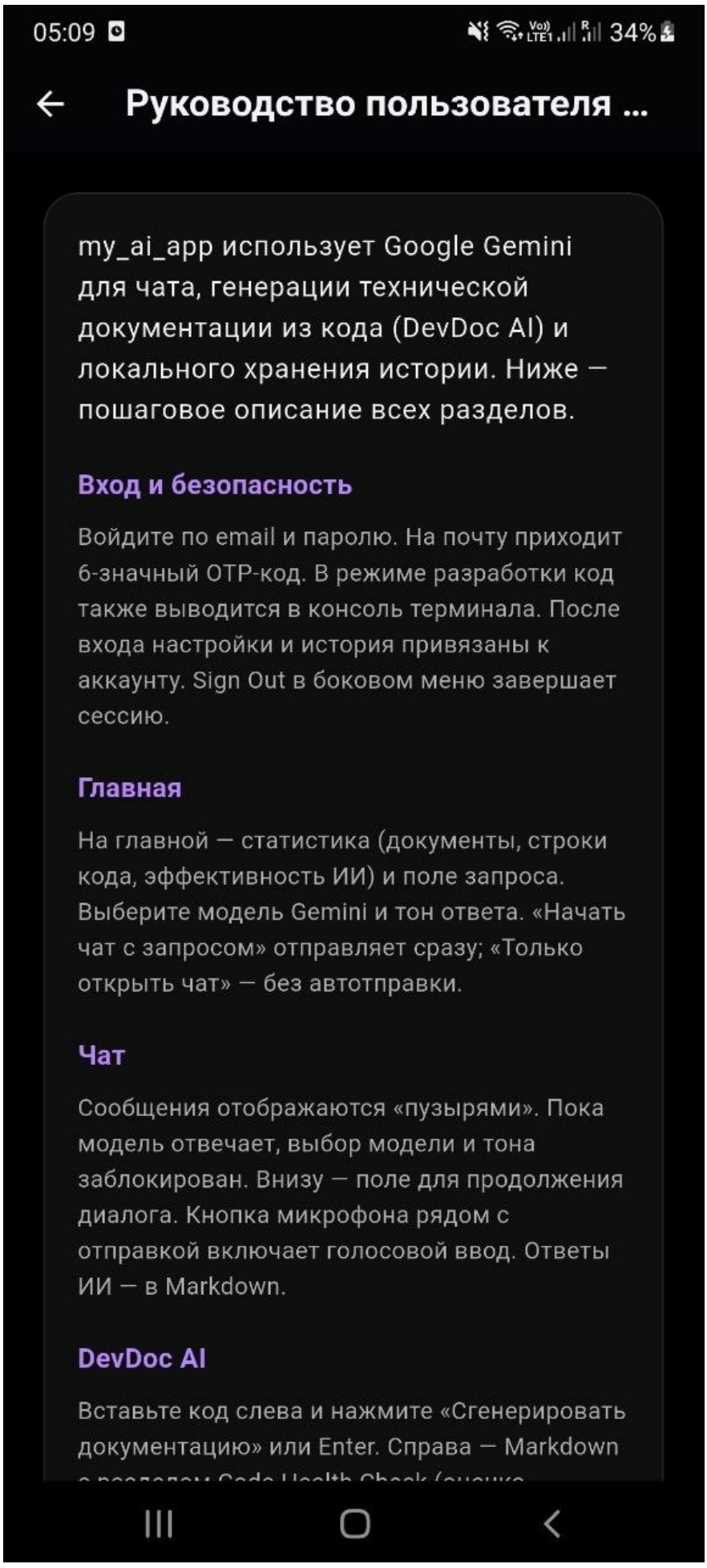
Додаток А

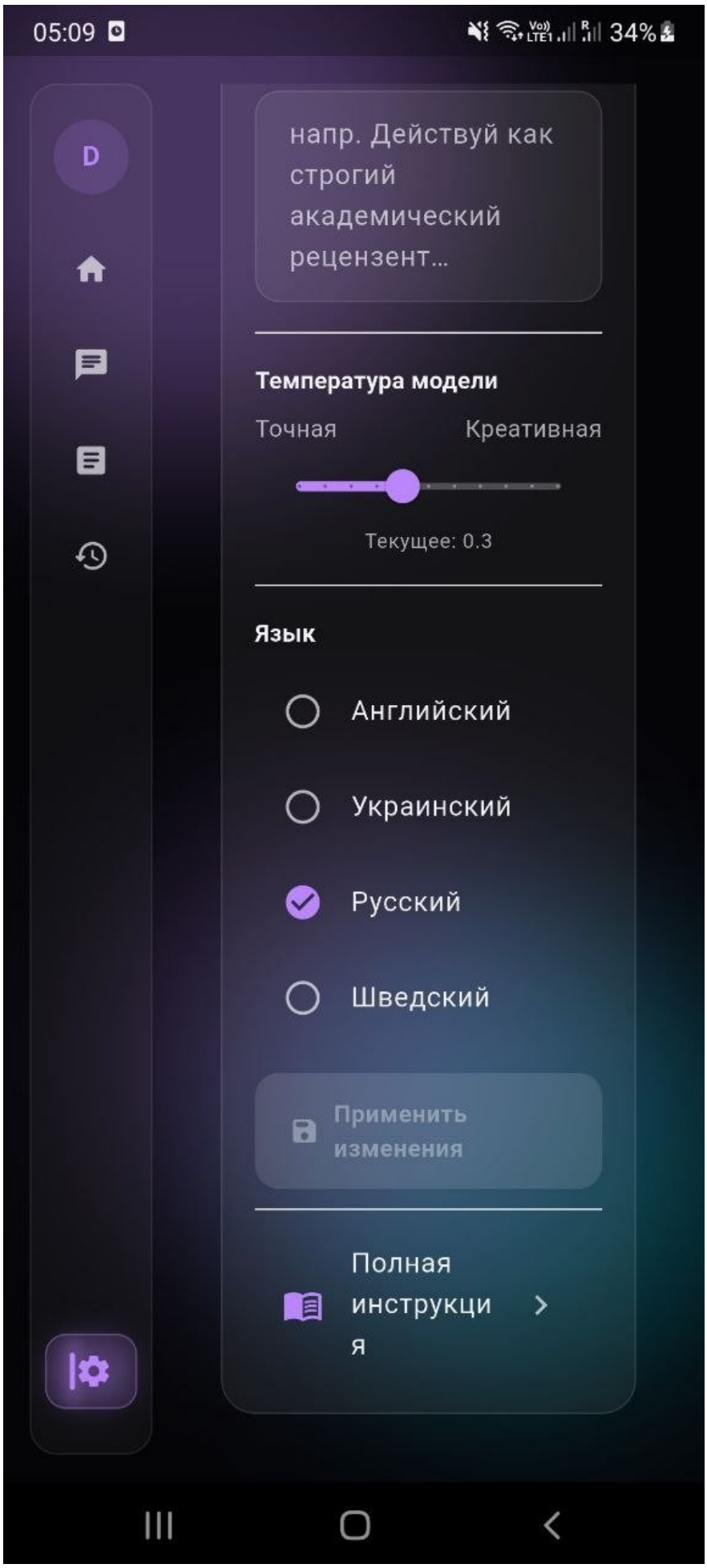
Фото додатка Android:

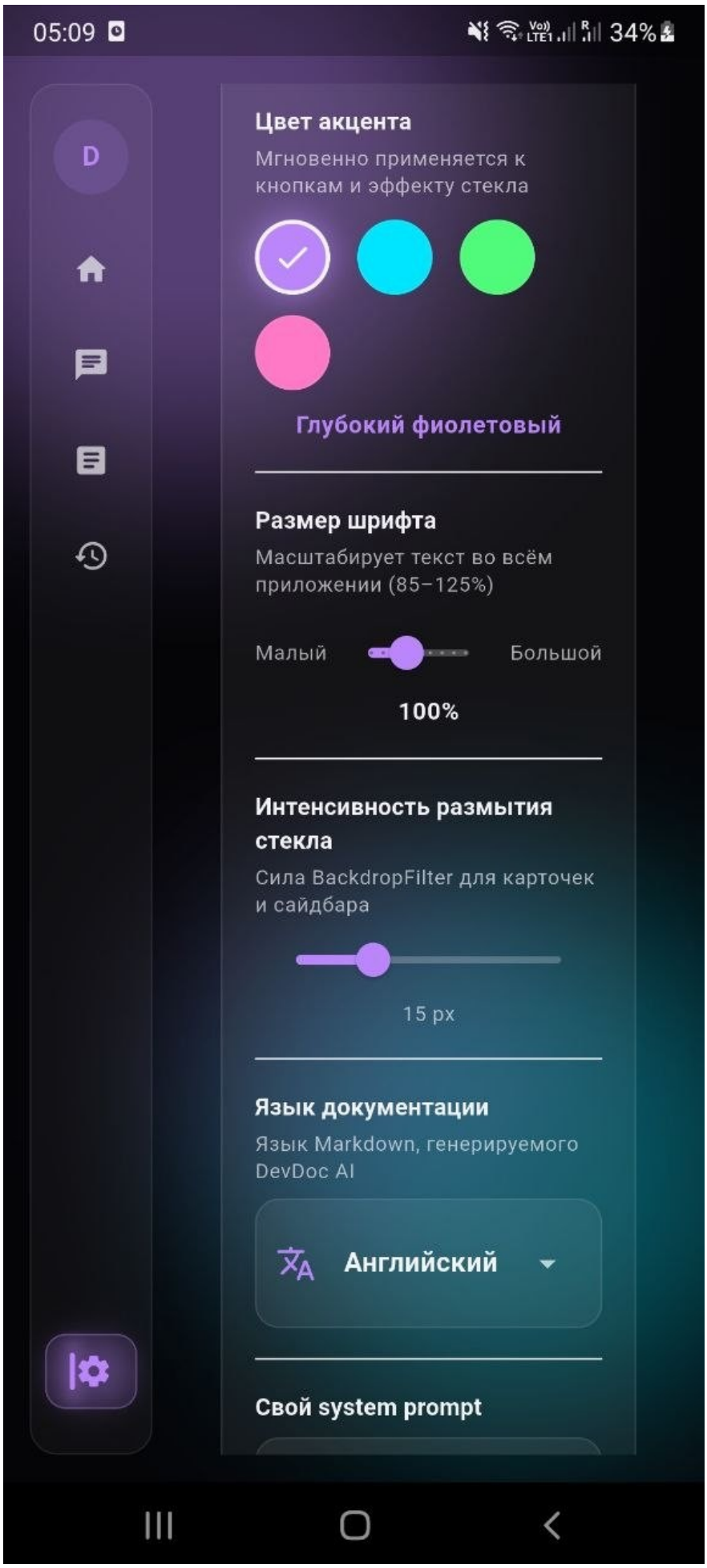


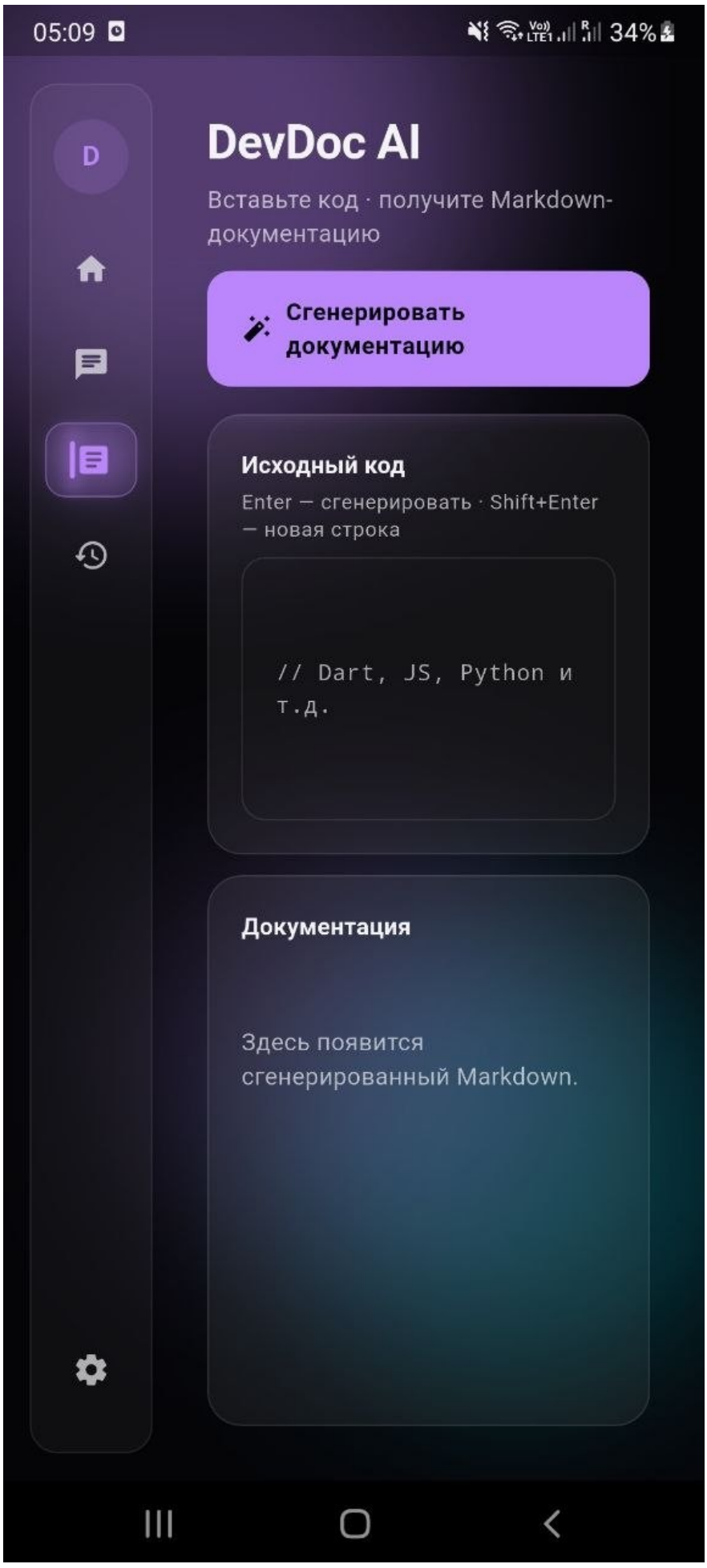

05:11
34%
D
История чата
May 24, 2026 05:11
Code Health Check
Complexity Score: 1
Suggestion: Provide valid sou...
May 24, 2026 05:10
Code Health Check
Complexity Score: 1
Suggestion: Please provide v...

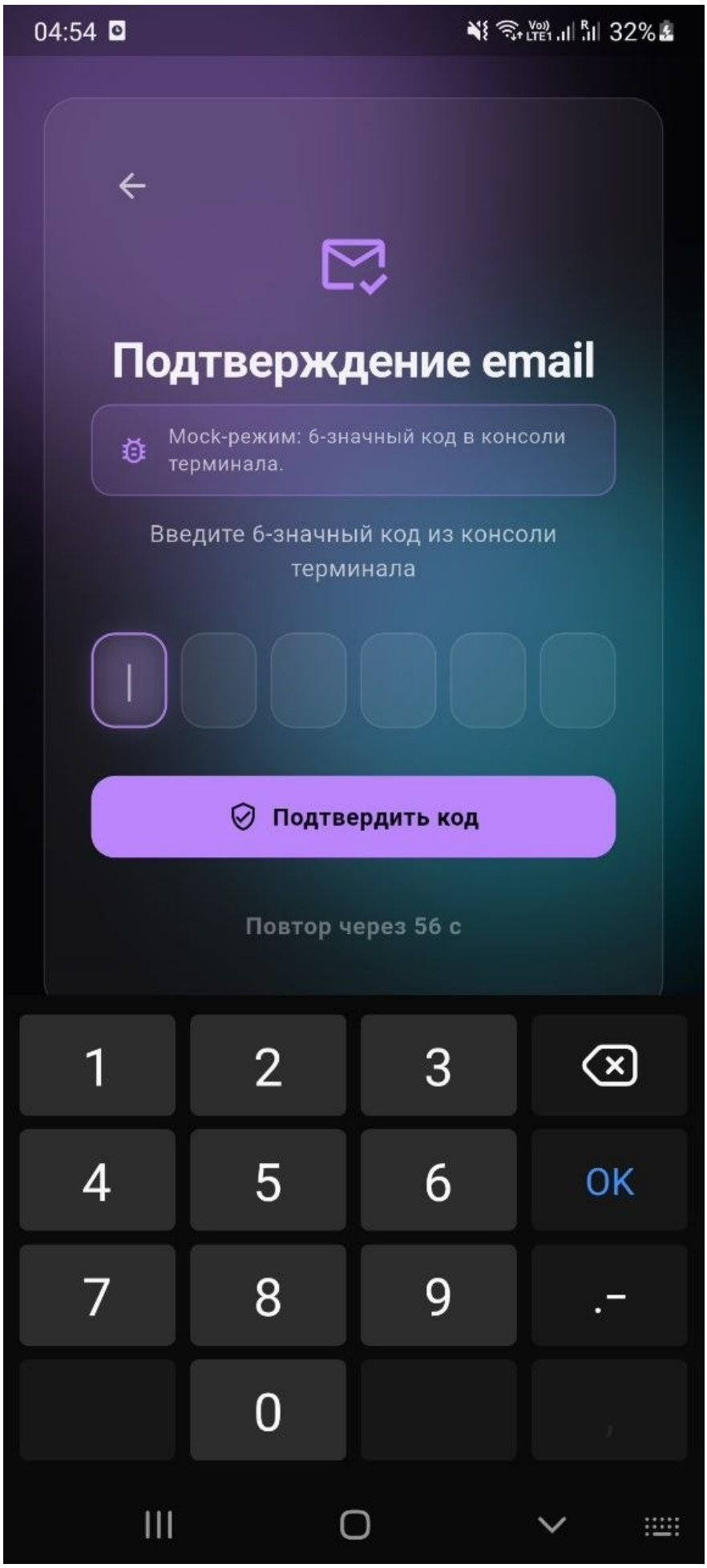

05:10
34%
Руководство пользователя ...
ИИ — в Markdown.
DevDoc AI
Вставьте код слева и нажмите «Сгенерировать документацию» или Enter. Справа — Markdown с разделом Code Health Check (оценка сложности 1–10). Вверху панели: копирование, экспорт .md и PDF. Язык документации — в Настройках.
История
Хранятся чаты и документы DevDoc. Запись содержит дату, превью и источник (chat / devdoc). Нажмите запись для полного Markdown. Поиск в чат-истории — через SQLite FTS5.
Настройки
Цвет акцента, размер шрифта (85–125%), интенсивность стекла, язык документации, свой system prompt и температура модели. Нажмите «Применить изменения», чтобы сохранить. Языки документации: украинский, английский, русский, шведский. Язык интерфейса отделён от языка документации.
Данные и API
Данные хранятся локально (SQLite + SharedPreferences). Нужен ключ Gemini (.env или переменная окружения). Не публикуйте ключ. SMTP в .env для OTP (mock-режим без реальной почты).

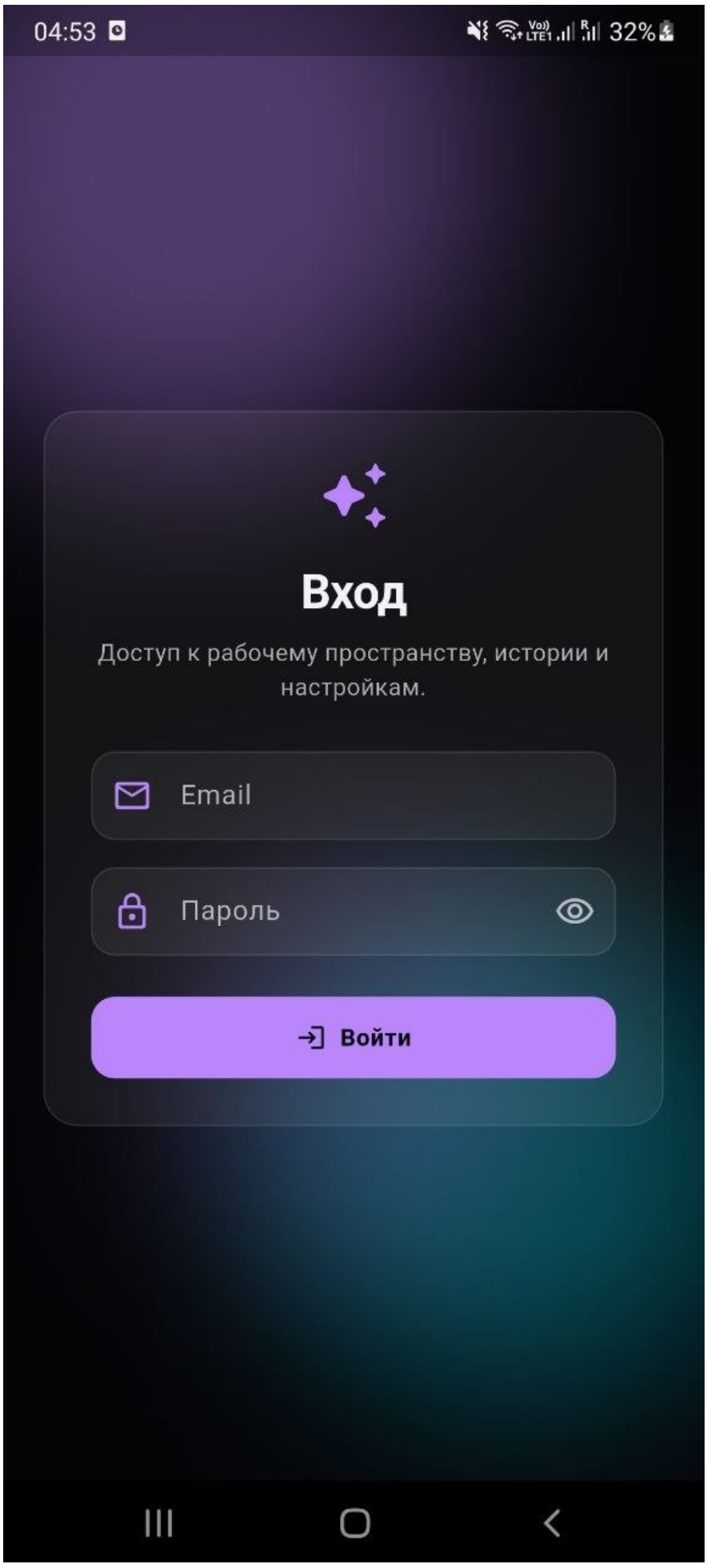

05:09
34%
Руководство пользователя ...
my_ai_app использует Google Gemini для чата, генерации технической документации из кода (DevDoc AI) и локального хранения истории. Ниже — пошаговое описание всех разделов.
Вход и безопасность
Войдите по email и паролю. На почту приходит 6-значный OTP-код. В режиме разработки код также выводится в консоль терминала. После входа настройки и история привязаны к аккаунту. Sign Out в боковом меню завершает сессию.
Главная
На главной — статистика (документы, строки кода, эффективность ИИ) и поле запроса. Выберите модель Gemini и тон ответа. «Начать чат с запросом» отправляет сразу; «Только открыть чат» — без автотправки.
Чат
Сообщения отображаются «пузырями». Пока модель отвечает, выбор модели и тона заблокирован. Внизу — поле для продолжения диалога. Кнопка микрофона рядом с отправкой включает голосовой ввод. Ответы ИИ — в Markdown.
DevDoc AI
Вставьте код слева и нажмите «Сгенерировать документацию» или Enter. Справа — Markdown


05:09
34%
D
напр. Действуй как строгий академический рецензент...
Температура модели
Точная
Креативная
Текущее: 0.3
Язык
Английский
Украинский
Русский
Шведский
Применить изменения
Полная инструкция

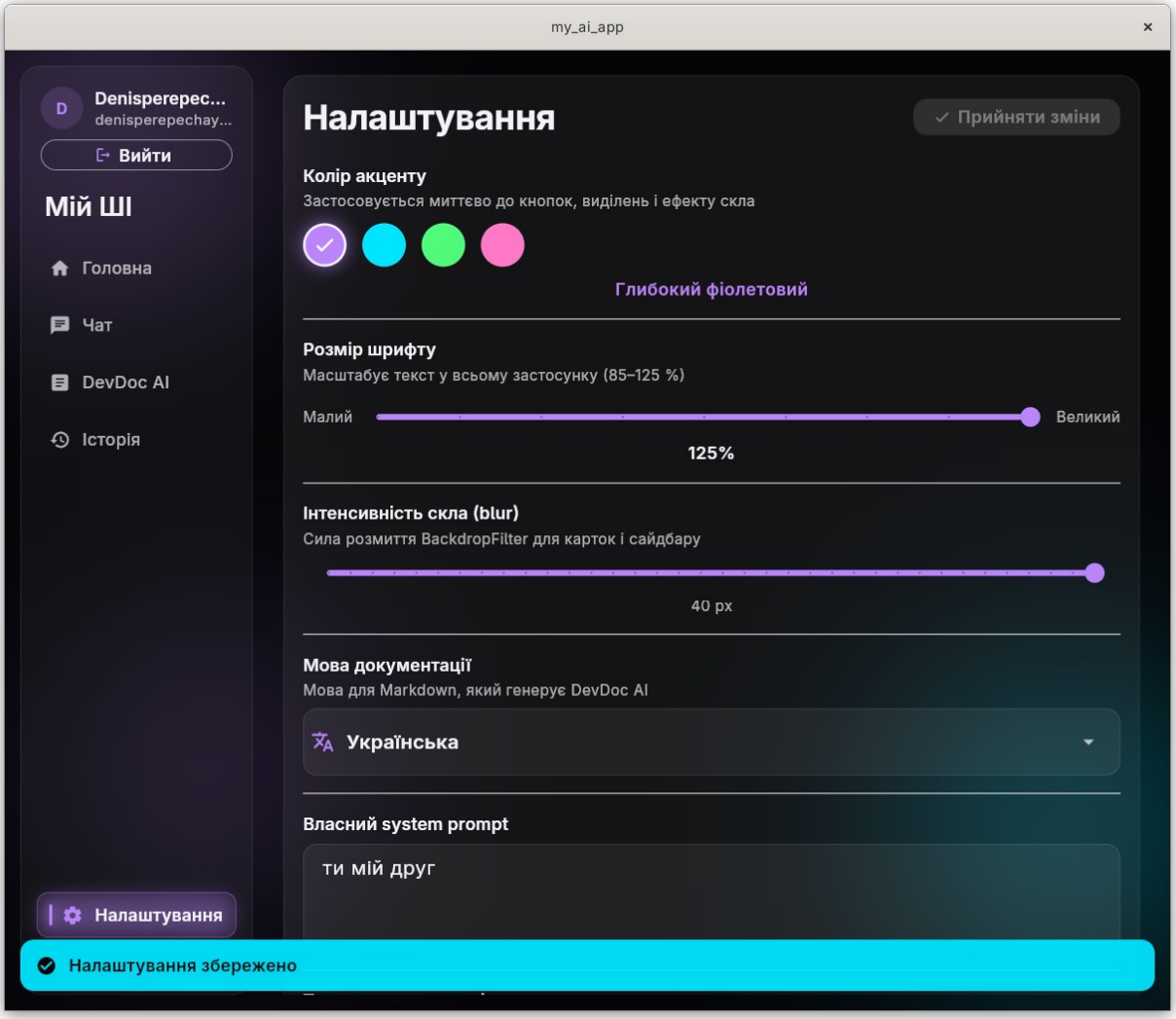

05:09
34%
D
Цвет акцента
Мгновенно применяется к кнопкам и эффекту стекла
Глубокий фиолетовый
Размер шрифта
Масштабирует текст во всём приложении (85–125%)
Малый
Большой
100%
Интенсивность размытия стекла
Сила BackdropFilter для карточек и сайдбара
15 px
Язык документации
Язык Markdown, генерируемого DevDoc AI
Английский
Свой system prompt

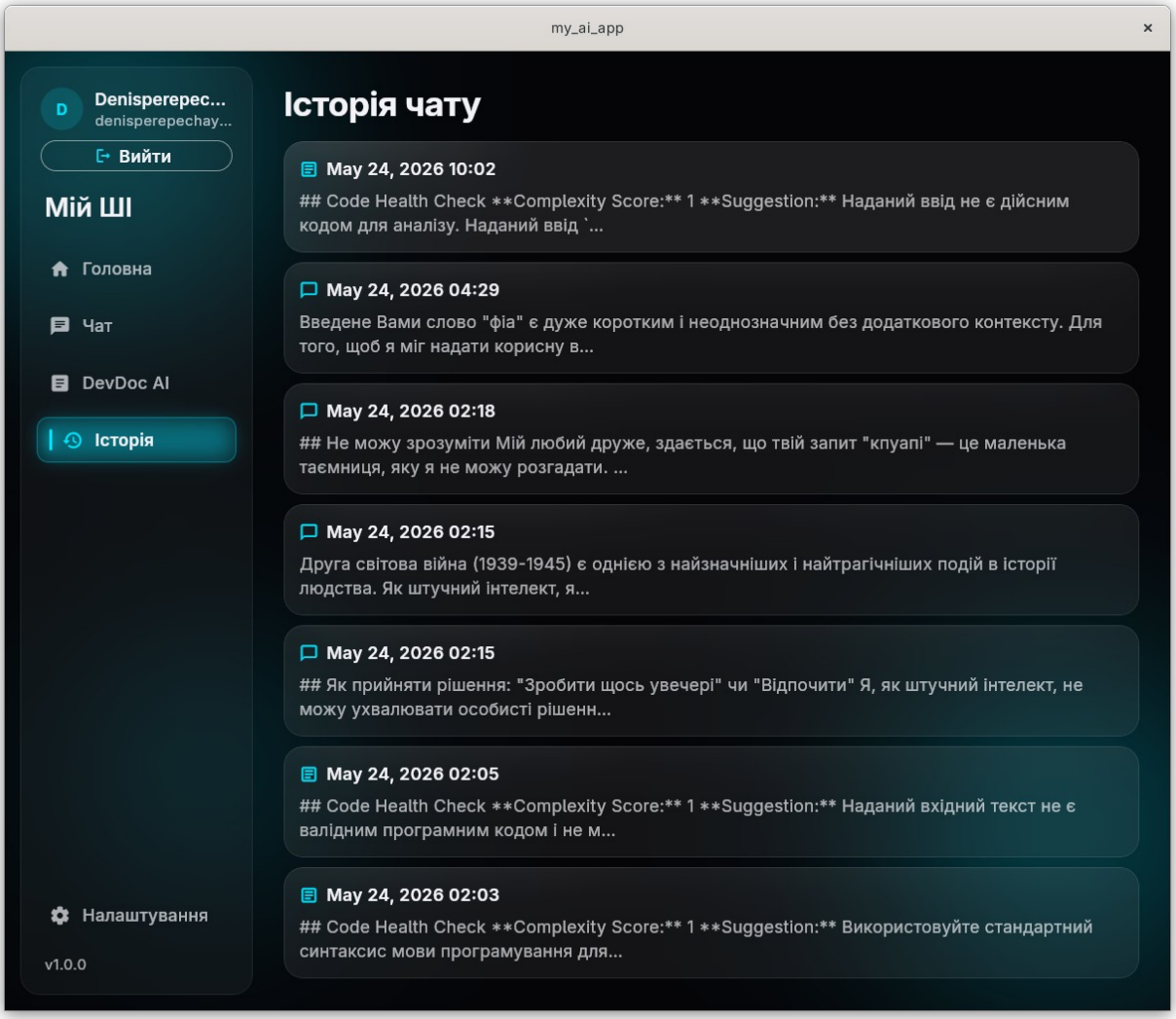

05:09
34%
D
DevDoc AI
Вставьте код · получите Markdown-документацию
Сгенерировать документацию
Исходный код
Enter — сгенерировать · Shift+Enter — новая строка
// Dart, JS, Python и т.д.
Документация
Здесь появится сгенерированный Markdown.

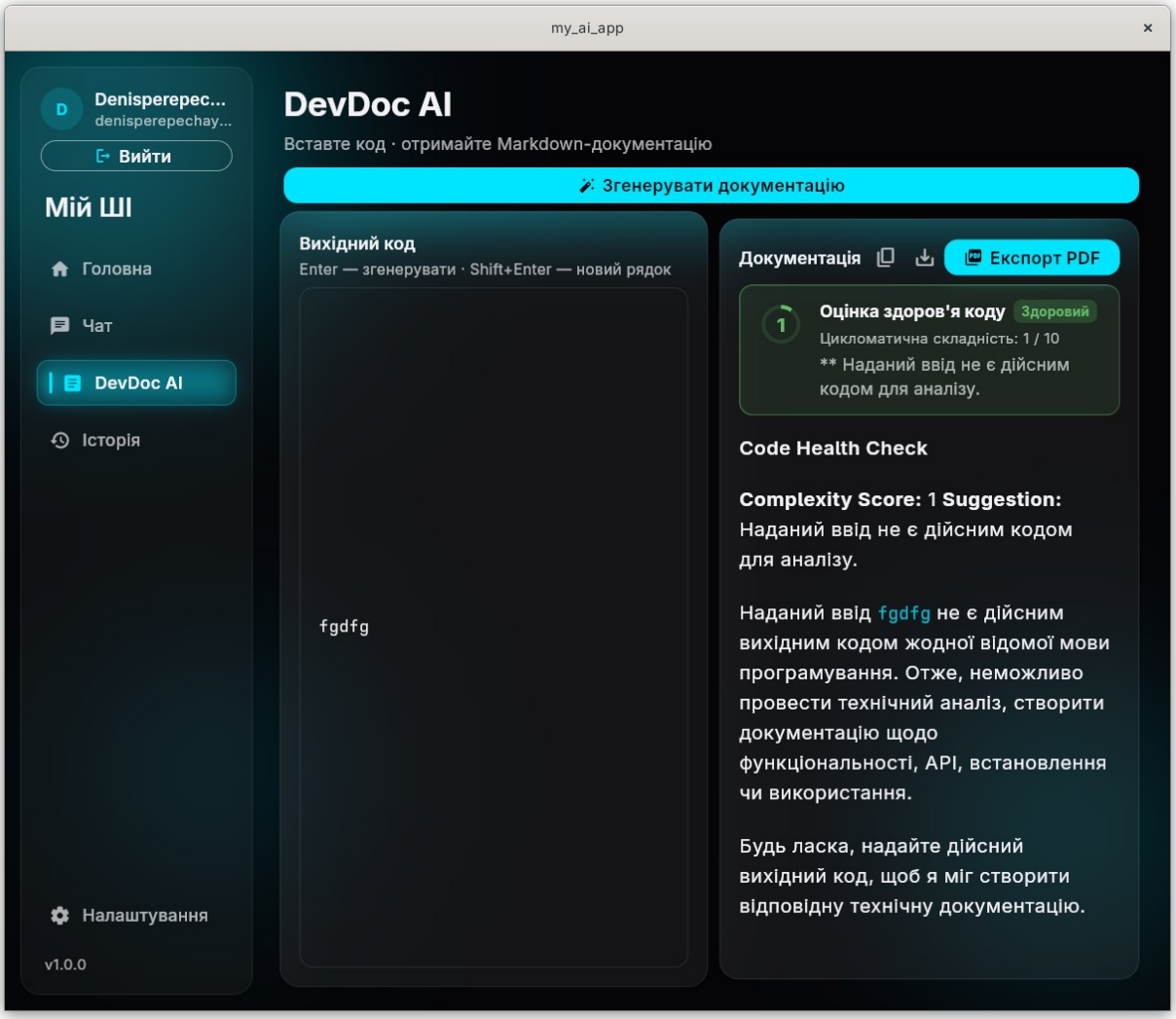

04:54
32%
Подтверждение email
Mock-режим: 6-значный код в консоли терминала.
Введите 6-значный код из консоли терминала
Подтвердить код
Повтор через 56 с
1
2
3
4
5
6
OK
7
8
9
.-
0

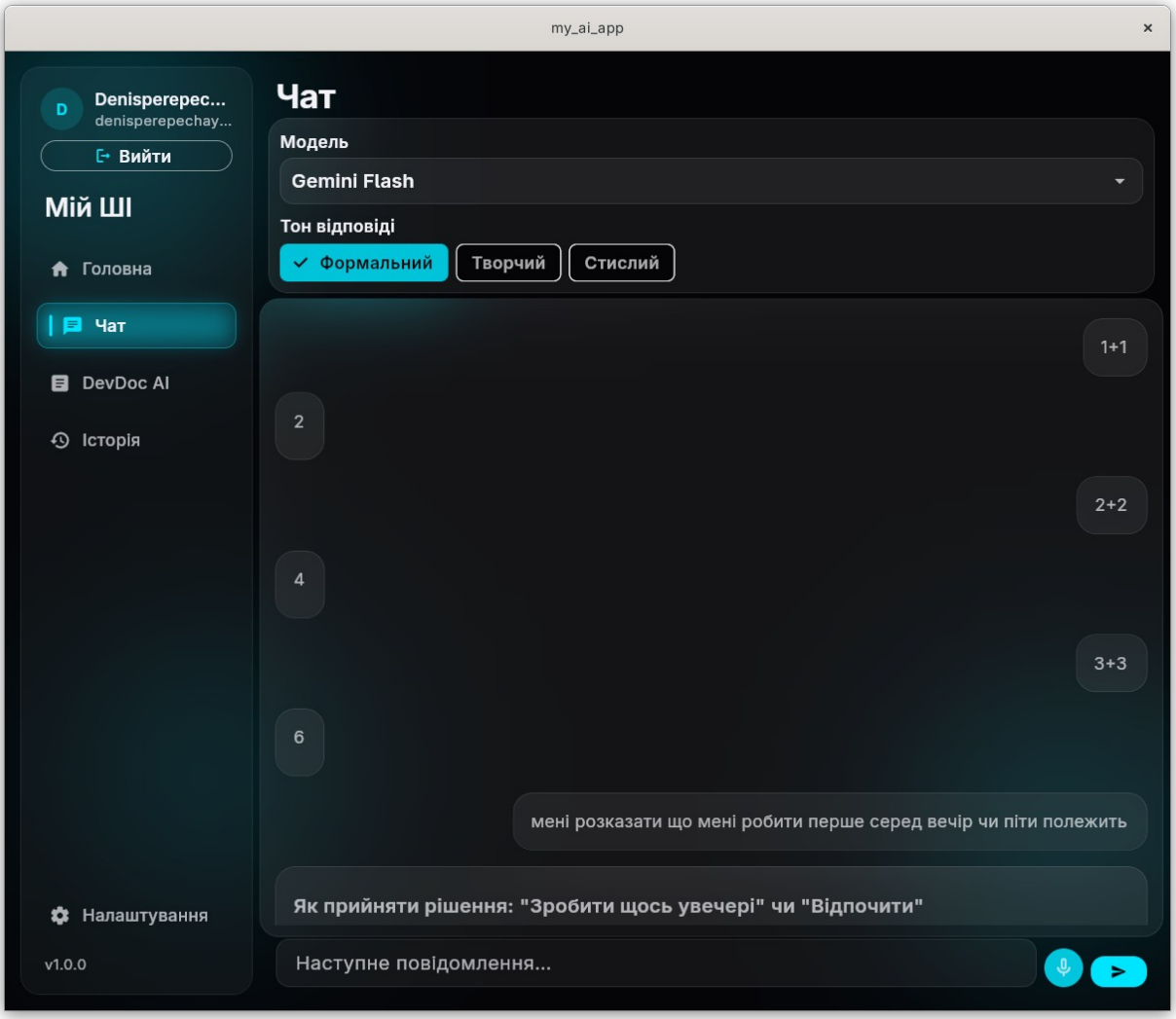

04:53
32%
Вход
Доступ к рабочему пространству, истории и настройкам.
Email
Пароль
Войти

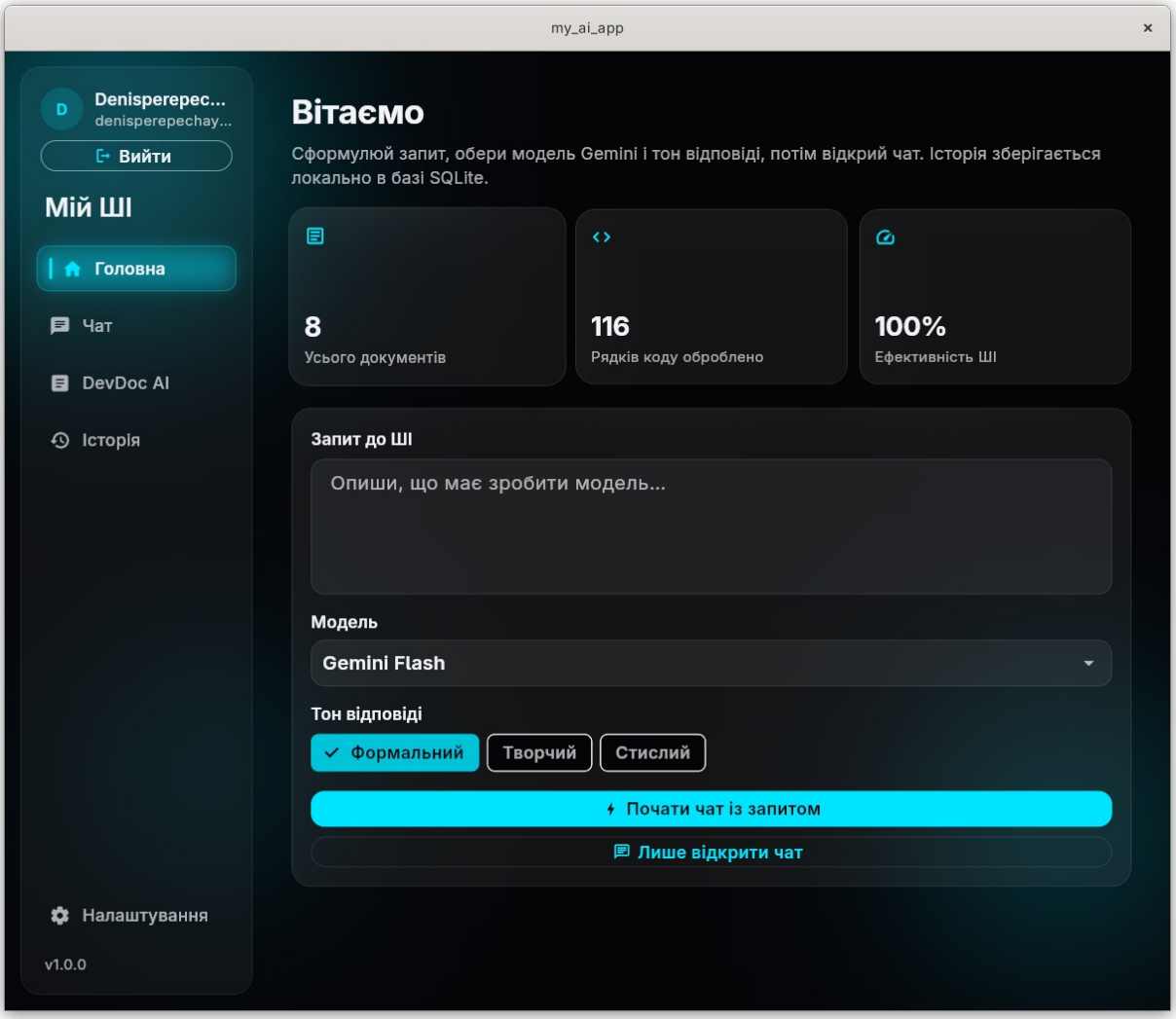
Linux:


my_ai_app
Denisperepec...
denisperepechay...
Вийти
Мій ШІ
Головна
Чат
DevDoc AI
Історія
Налаштування
Налаштування
Прийняти зміни
Колір акценту
Застосовується миттєво до кнопок, виділень і ефекту скла
Глибокий фіолетовий
Розмір шрифту
Масштабує текст у всьому застосунку (85–125 %)
Малий
Великий
125%
Інтенсивність скла (blur)
Сила розмиття BackdropFilter для карток і сайдбару
40 px
Мова документації
Мова для Markdown, який генерує DevDoc AI
Українська
Власний system prompt
ти мій друг
Налаштування збережено


my_ai_app
Denisperepec...
denisperepechay...
Вийти
Мій ШІ
Головна
Чат
DevDoc AI
Історія
Налаштування
v1.0.0
Історія чату
May 24, 2026 10:02
Code Health Check **Complexity Score:** 1 **Suggestion:** Наданий ввід не є дійсним кодом для аналізу. Наданий ввід `...
May 24, 2026 04:29
Введене Вами слово "фіа" є дуже коротким і неоднозначним без додаткового контексту. Для того, щоб я міг надати корисну в...
May 24, 2026 02:18
Не можу зрозуміти Мій любий друже, здається, що твій запит "кпуапі" — це маленька таємниця, яку я не можу розгадати. ...
May 24, 2026 02:15
Друга світова війна (1939-1945) є однією з найзначніших і найтрагічніших подій в історії людства. Як штучний інтелект, я...
May 24, 2026 02:15
Як прийняти рішення: "Зробити щось увечері" чи "Відпочити" Я, як штучний інтелект, не можу ухвалювати особисті рішенн...
May 24, 2026 02:05
Code Health Check **Complexity Score:** 1 **Suggestion:** Наданий вхідний текст не є валідним програмним кодом і не м...
May 24, 2026 02:03
Code Health Check **Complexity Score:** 1 **Suggestion:** Використовуйте стандартний синтаксис мови програмування для...


my_ai_app
Denisperepec...
denisperepechay...
Вийти
Мій ШІ
Головна
Чат
DevDoc AI
Історія
Налаштування
v1.0.0
DevDoc AI
Вставте код · отримайте Markdown-документацію
Згенерувати документацію
Вихідний код
Enter — згенерувати · Shift+Enter — новий рядок
fgdfg
Документація
Експорт PDF
1
Оцінка здоров'я коду
Здоровий
Цикломатична складність: 1 / 10
** Наданий ввід не є дійсним кодом для аналізу.
Code Health Check
Complexity Score: 1 Suggestion: Наданий ввід не є дійсним кодом для аналізу.
Наданий ввід fgdfg не є дійсним вихідним кодом жодної відомої мови програмування. Отже, неможливо провести технічний аналіз, створити документацію щодо функціональності, API, встановлення чи використання.
Будь ласка, надайте дійсний вихідний код, щоб я міг створити відповідну технічну документацію.


my_ai_app
Denisperepec...
denisperepechay...
Вийти
Мій ШІ
Головна
Чат
DevDoc AI
Історія
Налаштування
v1.0.0
Чат
Модель
Gemini Flash
Тон відповіді
Формальний
Творчий
Стислий
1+1
2
2+2
4
3+3
6
мені розказати що мені робити перше серед вечір чи піти полежить
Як прийняти рішення: "Зробити щось увечері" чи "Відпочити"
Наступне повідомлення...


my_ai_app
Denisperepec...
denisperepechay...
Вийти
Мій ШІ
Головна
Чат
DevDoc AI
Історія
Налаштування
v1.0.0
Вітаємо
Сформулюй запит, обери модель Gemini і тон відповіді, потім відкрий чат. Історія зберігається локально в базі SQLite.
8
Усього документів
116
Рядків коду оброблено
100%
Ефективність ШІ
Запит до ШІ
Опиши, що має зробити модель...
Модель
Gemini Flash
Тон відповіді
Формальний
Творчий
Стислий
Почати чат із запитом
Лише відкрити чат

Додаток Б

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) Інформаційних Технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Перепечай Денис Олексійович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота (проєкт) на тему

Мобільний додаток з інтегрованим штучним інтернетом є результатом моєї особистої праці.

2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.

Щодо використання інструментів ШІ зазначаю, що інструменти ШІ ChatGPT, були використані для:

- [+]перекладу іншомовних джерел;
- [+]стилістичного редагування та перевірки граматики;
- [+]допомоги у написанні/відлагодженні програмного коду.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

22.05.2026 р. _____**Денис Перепечай**

(підпис)