МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

АНДРОИД - ПРИЛОЖЕНИЕ «ИНТЕЛЛЕКТУАЛЬНЫЕ ЗАМЕТКИ»

Студент 08.06.26 Гавин Владимир Николаевич
(подпись, дата) (ФИО полностью)

Руководитель 08.06.26 Митрохина Н.Ю.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр 08.06.26 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.26 г. № 14)

Заведующий кафедрой М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 19 июня 2026г. № 39)

Секретарь ГЭК Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«07» 04 2026 г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Гавин Владимир Николаевич гр. 22ВВП2 факультета вычислительной техники

2.Тема работы: Андроид - приложение «Интеллектуальные заметки»

Тема утверждена приказом ПГУ №_347/о_____от "07"___апреля_____2026__г.

3. Руководитель бакалаврской работы к.т.н., доцент кафедры ВТ ПГУ Митрохина Н.Ю.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать мобильное приложение «Интеллектуальные заметки», позволяющее автоматизировать структурирование персональной информации. Язык программирования Dart, фреймворк Flutter, среда разработки Visual Stidio Code. Программа должна обладать интуитивно понятным интерфейсом и использовать следующие технологии:

1. Firebase authentication
2. Cloud Firebase
3. Асинхронное программирование
4. Material design 3
5. Gemini AI API.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области: изучить существующие аналоги, изучить способы проектирования и разработки мобильного приложения. Выбрать язык программирования, среду разработки и базу данных.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	48	07.04.2026 – 13.04.2026	
2	Анализ требований	36	14.04.2026 – 18.04.2026	
3	Выбор технологий, применяемых при проектировании и разработке	40	19.04.2026 – 24.04.2026	
4	Создание графического интерфейса	12	25.04.2026 – 26.04.2026	
5	Разработка мобильного приложения	168	27.04.2026 – 18.05.2026	
6	Оформление пояснительной записки	20	19.05.2026 – 21.05.2026	

Дата выдачи задания "07" апреля 2026 г.

Руководитель бакалаврской работы _____ Митрохина Н.Ю.
(подпись, дата)

Задание к исполнению принял студент _____ Гавин В.Н.
(подпись, дата)

Работу к защите допустить
Декан факультета _____ 10.06.2026 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 96 листов, 19 рисунков, 2 таблицы, 8 источников и 3 приложения.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, FLUTTER, DART, ГЕНЕРАТИВНЫЙ ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, GEMINI API, ОБЛАЧНАЯ СИНХРОНИЗАЦИЯ, FIREBASE, SUPABASE, АРХИТЕКТУРА ПРОГРАММНЫХ СИСТЕМ

Цель работы – разработать интеллектуальную мобильную систему для управления персональной информацией, позволяющую автоматизировать процессы категоризации и планирования с использованием технологий искусственного интеллекта.

Объектом разработки является мобильное приложение на базе интеллектуальной модели, обеспечивающее пользовательский интерфейс для создания, хранения и организации заметок.

В результате проделанной работы разработано приложение, полностью соответствующее поставленной задаче. Реализованы функции интеллектуальной обработки текста, а также механизмы интеграции с системными службами устройства. Внедрена гибридная система хранения данных, использующая локальную базу данных Hive и облачные сервисы Firebase и Supabase.

Содержание

Перечень сокращений и обозначений 4
Введение 5
1 Андроид – приложение «Интеллектуальные заметки» 6
1.1 Анализ предметной области 6
1.1.1 Обзор аналогов 7
1.1.2 Анализ требований к разработке 11
1.1.3 Выбор архитектуры приложения 12
1.1.4 Выбор языка программирования 13
1.1.5 Выбор серверной инфраструктуры 14
1.1.6 Выбор среды разработки 15
1.2 Проектирование и разработка 15
1.2.1 Реализация архитектуры приложения 16
1.2.2 Проектирование модели данных и структуры хранения 18
1.2.3 Интеграция с облачной инфраструктурой 20
1.2.4 Разработка интеллектуального модуля 24
1.3 Тестирование 28
1.3.1 Тестирование базового функционала системы 28
2 Безопасность жизнедеятельности 34
2.1 Правила и техника безопасности при работе с ПК 34
2.2 Правила и техника безопасности при работе со смартфоном 35
Заключение 36
Список использованных источников 37
Приложение А – UML-диаграммы 38
Приложение Б – Листинг программы 43
Приложение В – Презентация 86

Перечень сокращений и обозначений

В настоящей работе применяются следующие сокращения и обозначения

БД – база данных

AI (Artificial Intelligence) – искусственный интеллект

API (Application Programming Interface) – программный интерфейс приложения

HTTPS (HyperText Transfer Protocol Secure) – протокол безопасной передачи гипертекста с поддержкой шифрования

OSS (Open Source Software) – программное обеспечение с открытым исходным кодом

S3 (Simple Storage Service) – облачный сервис объектного хранения данных

UI (User Interface) – пользовательский интерфейс

UML (Unified Modeling Language) – унифицированный язык моделирования

UUID (Universally Unique Identifier) – универсальный уникальный идентификатор

VS (Visual Studio Code) – интегрированная среда разработки

Введение

Мобильные устройства служат основным средством фиксации персональной информации. Увеличение объемов данных делает традиционные форматы цифровых блокнотов малоэффективными.

Тексты, изображения и списки задач сохраняются единым массивом. Поиск нужных сведений усложняется с ростом количества создаваемых записей. Планы и данные о предстоящих встречах не переходят в системные календари автоматически, оставаясь обычным текстом.

Внедрение API LLM трансформирует работу с текстовыми данными. Нейросети способны извлекать метаданные, классифицировать записи, распознавать события и создавать напоминания.

Цель работы заключается в разработке мобильного приложения для управления заметками с интеллектуальными инструментами для повышения продуктивности и снижения когнитивной нагрузки.

Практическая часть включает проектирование архитектуры приложения с приоритетом локального хранения данных, создание модуля взаимодействия с нейросетью, настройку фоновой синхронизации с распределенными облачными базами и реализацию интеграции с нативными инструментами операционной системы.

1 Андроид - приложение «Интеллектуальные заметки»

1.1 Анализ предметной области

Предметная область исследования охватывает создание, хранение и извлечение записей на мобильных устройствах. Рост повседневного информационного потока снижает эффективность стандартных способов работы с текстовыми файлами.

Пользователь фиксирует мысли на бегу. Поток повседневных задач перегружает память, поэтому пользователь открывает приложение просто для быстрого сохранения идеи. Текст оседает в базе данных. Проблема забытых идей возникает из-за полного отсутствия контекстной связи между отдельными записями. Пользователь возвращается к заметке через неделю и теряет изначальный смысл написанного. Тут возникает потребность сконцентрироваться на главном. Огромный массив неструктурированных файлов сильно мешает найти нужную информацию. Пользователь боится забыть о важном деле. Хранилище быстро превращается в хаотичную свалку из-за нежелания тратить время на ручную сортировку.

Превращение заметки в задачу внутри классических программ требует от владельца устройства дополнительных действий: создания нового события, настройка даты и времени. Рядовой пользователь не будет тратить время на выполнение этих рутинных шагов при быстрой фиксации мыслей. В результате важная информация остается в базовом текстовом формате без привязки к системным напоминаниям.

Взаимодействие с людьми часто становится главной целью создания заметки. Пользователь пишет имя коллеги для будущего обсуждения рабочего вопроса. Наступление срока выполнения требует отдельного поиска нужного человека в системной адресной книге или мессенджере. Рутина ручного поиска нужного абонента сильно снижает общую скорость работы.

Форматирование текста на смартфонах часто сопряжено с лишними действиями и поиском нужных кнопок в меню. Из-за этого многие фиксируют данные сплошным текстом и не заботятся об их структуре.

Автоматизировать работу с записями помогает внедрение API LLM. Современные алгоритмы умеют не только исправлять опечатки или выделять главные тезисы, но и выстраивать общую логику документа. Анализируя контекст, нейросеть может самостоятельно находить даты для генерации напоминаний и распознавать контакты упомянутых людей. В связи с этим, цель данного исследования заключается в интеграции интеллектуальных инструментов в мобильную среду, чтобы ускорить обработку личной информации.

1.1.1 Обзор аналогов

Рынок приложений для записей предлагает решения с различными архитектурными подходами.

Базовые приложения для работы с заметками сфокусированы на быстром сохранении информации.

Evernote предоставляет инструменты сохранения веб-страниц и распознавания печатных символов на изображениях. Бесплатная версия приложения ограничивает ежемесячный объем загружаемых данных. Интерфейс включает большое количество элементов управления и настроек. На рисунке 1 представлен пользовательский интерфейс приложения Evernote.

Google Keep функционирует по принципу виртуальной доски с текстовыми карточками. Структуризация записей осуществляется посредством ярлыков и цветового кодирования, что исключает создание многоуровневой иерархии папок. Внешний вид рабочего пространства продемонстрирован на рисунке 2.

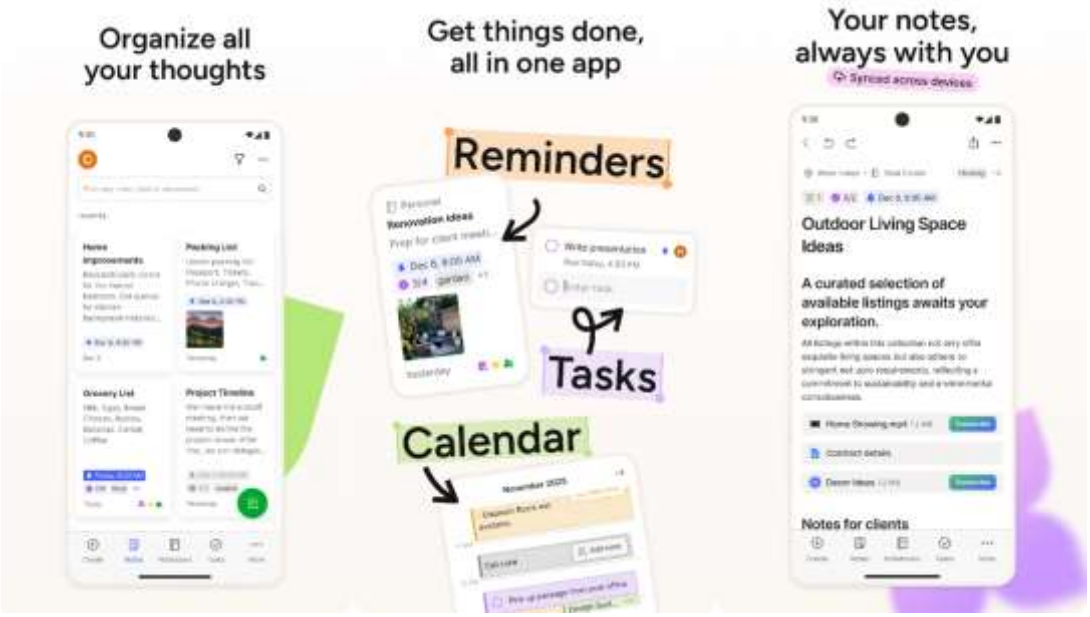


Рисунок 1 – Evernote

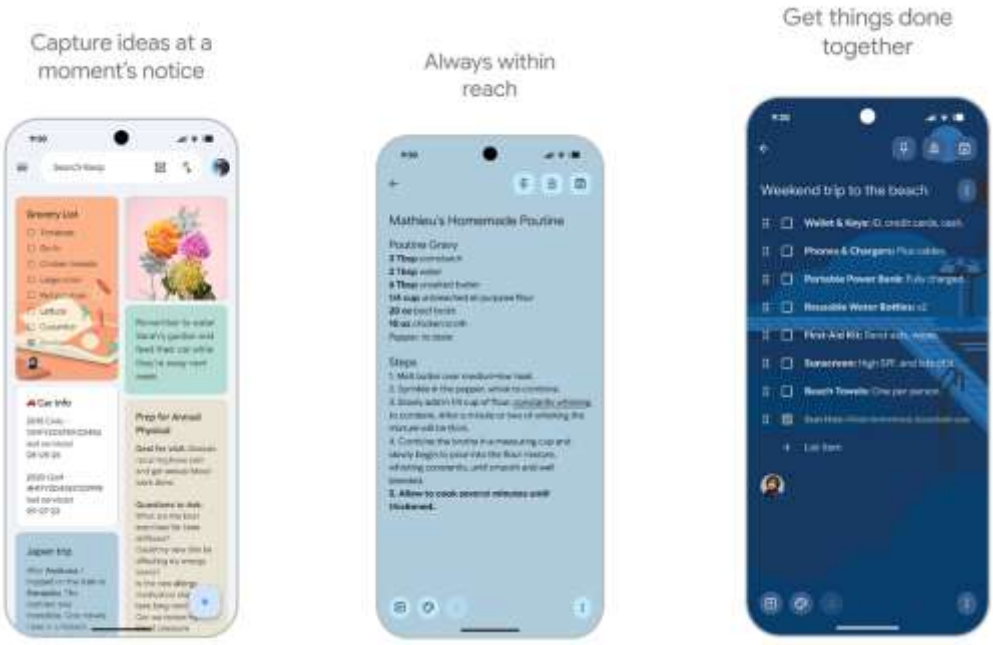


Рисунок 2 – Google Keep

Комплексные системы управления знаниями предлагают расширенный набор архитектурных решений.

Notion использует модульную систему организации контента. Рабочее пространство формируется из текстовых блоков, реляционных баз данных, досок

управления задачами и календарей. Встроенные алгоритмы искусственного интеллекта выполняют генерацию текста, создание выжимок и машинный перевод. Функционирование сервиса зависит от стабильного подключения к интернету, в автономном режиме часть функций недоступна. На рисунке 3 представлен интерфейс Notion.

Obsidian сохраняет данные в локальной памяти устройства в формате текстовых файлов Markdown. Особенностью системы является поддержка двусторонних ссылок. При упоминании одного документа в другом программа автоматически фиксирует связь. На основе этих упоминаний система генерирует визуальный граф для отображения пересечений между файлами. Функции среды расширяются установкой сторонних плагинов, включая модули для запуска локальных нейросетей. Синхронизация файлов осуществляется через платный сервис разработчика или ручную настройку интеграции со сторонними облачными хранилищами. На рисунке 4 отображен интерфейс Obsidian.

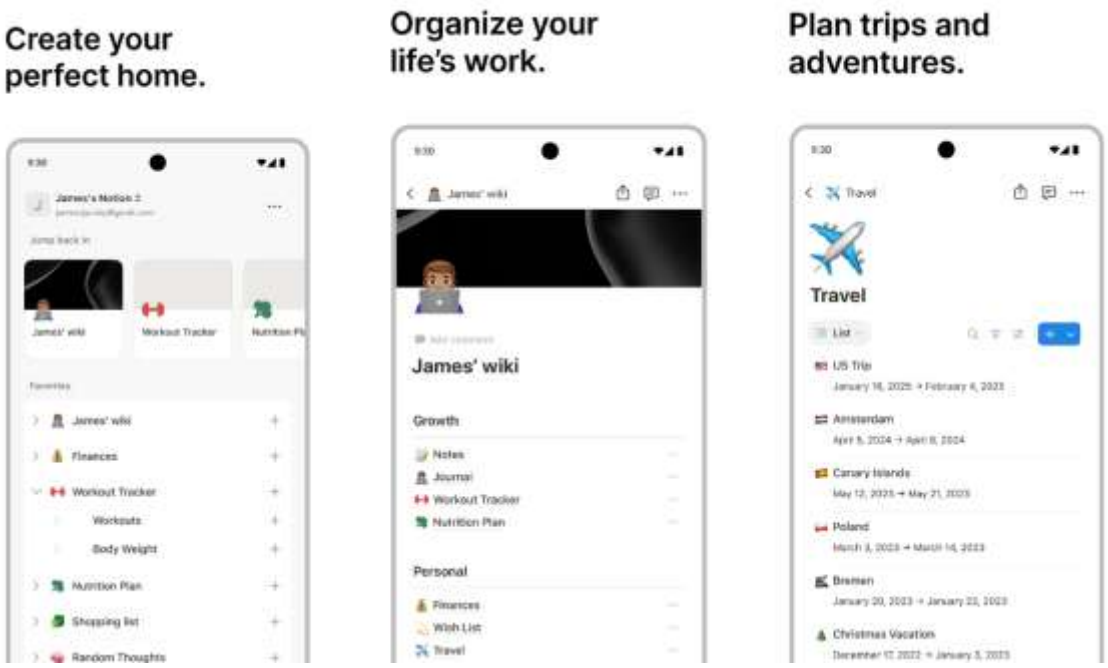


Рисунок 3 – Notion

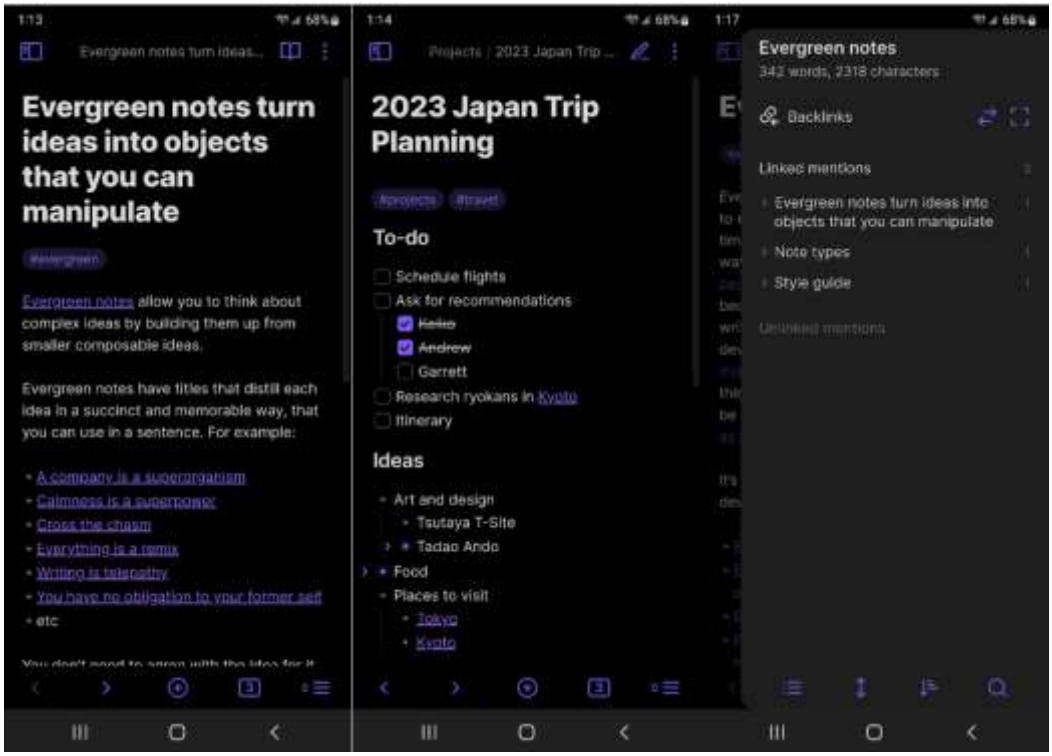


Рисунок 4 – Obsidian

Сопоставление ключевых параметров рассмотренных приложений приведено в таблице 1.

Таблица 1 — Сравнительная характеристика приложений для заметок

Сервис	Особенности	Ограничения
Evernote	Веб-клиппер, OCR на фото	Лимиты загрузки, перегруженный интерфейс
Google Keep	Простое создание заметок	Нет папок, ИИ инструментов
Notion	БД, ИИ-ассистент, модули расширений	Зависимость от сети, сложность обучения
Obsidian	Граф связей, Markdown разметка	Платная синхронизация, ручная настройка

1.1.2 Анализ требований к разработке

Проектная спецификация описывает параметры программного обеспечения. Диаграмма вариантов использования представлена на рисунке 5.

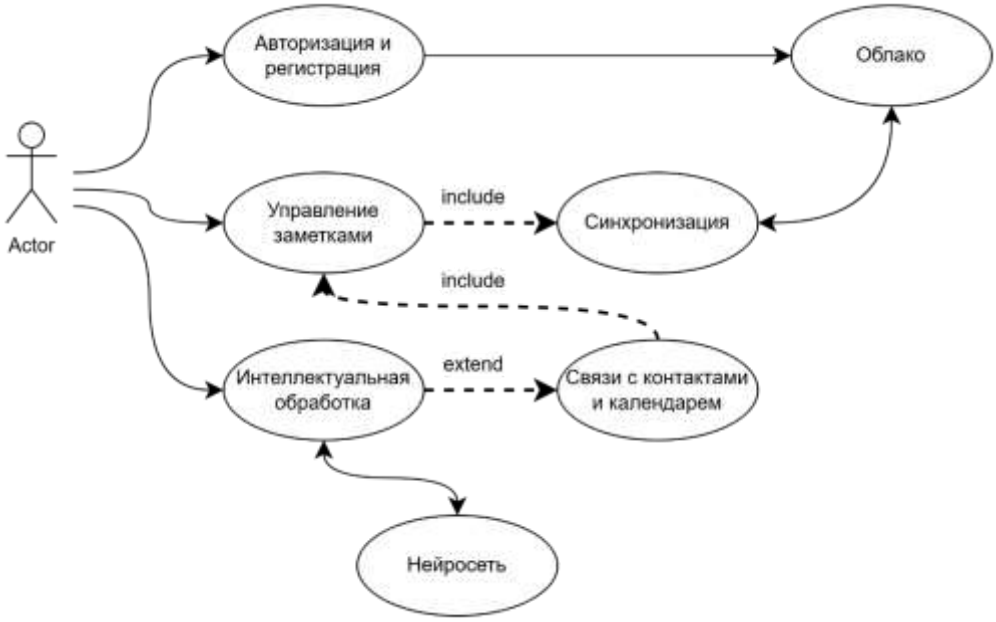


Рисунок 5 – Диаграмма вариантов использования

Функциональные требования задают точные действия системы. Приложение создает, читает, редактирует, удаляет заметки. Синтаксис текстовой разметки определяет визуальное форматирование записей. Файлы сохраняются в локальную базу данных. Данные при синхронизации с облачным хранилищем отправляются в фоновом режиме.

Встроенная нейросеть анализирует массив пользовательских заметок. Она выявляет смысловые пересечения между отдельными файлами базы. Связи для структурной организации идей визуализируются в интерфейсе программы в виде системы папок.

Нейросеть работает внутри конкретной записи. Алгоритм вычисляет контекст предложений. Приложение самостоятельно прикрепляет контакты из адресной книги смартфона и формирует события системного календаря.

Блок нефункциональных требований фиксирует параметры архитектуры. Приложение разрабатывается под операционную систему Android.

Нефункциональные требования:

– холодный запуск приложения занимает до 3 секунд;

– графика рендерится с адаптивной частотой вплоть до 120 гц;

– интерфейс откликается за 100 миллисекунд;

– ответ базы данных занимает не более одной секунды;

– поддержка Android API уровня 29 и выше;

– корректное отображение на экранах разного разрешения;

– портретная и альбомная ориентация;

– взаимодействие с сервером идет по протоколу HTTP;

– безопасная аутентификация для входа с новых устройств;

– элементы интерфейса адаптированы для управления пальцем;

– темная и светлая темы оформления.

1.1.3 Выбор архитектуры приложения

Архитектура строится по клиент-серверной модели. Разработка собственного сервера требует огромного количества времени. Инфраструктура делегирована специализированной облачной платформе класса BaaS. Она полностью берет на себя аутентификацию пользователей и синхронизацию данных между устройствами. Разработчик фокусируется на графическом интерфейсе.

Структура клиентского кода организована по принципу feature-first. Код жестко группируется по смысловым функциям. Мы уходим от монолитного связывания логики, разработчик открывает директорию и видит полностью изолированную рабочую среду. Папка заметок объединяет всю работу функции. Логика системной навигации лежит в соседней директории. Это упрощает сборку.

Разработчик получает быстрый доступ к компонентам. Изменение работы текстового редактора никак не затрагивает независимый модуль авторизации. Изолированный код гораздо легче покрывать тестами, а поиск возникающих ошибок занимает минимум времени из-за строгой локализации файлов. Добавление новых функций происходит быстрее благодаря прозрачной структуре проекта.

Эти архитектурные решения оптимальны для текущего проекта. Связка облачной платформы и модульного принципа feature-first минимизирует затраты времени на создание инфраструктуры. Облако берет на себя рутинную работу с серверной средой, модульная компоновка файлов при этом исключает техническую путаницу при масштабировании кодовой базы.

1.1.4 Выбор языка программирования

Для реализации пользовательского интерфейса и внутренней бизнес-логики мобильного клиента выбран кроссплатформенный фреймворк Flutter, использующий язык программирования Dart [1]. Выбор данной технологии обусловлен преимуществами:

– независимая отрисовка интерфейса;

– единая кодовая база;

– зрелая экосистема.

При анализе альтернатив рассматривалась разработка на языке Kotlin в среде Android Studio. Но такой подход жестко привязывает продукт к одной операционной системе. При масштабировании проекта на другие платформы весь код пришлось бы переписывать с нуля.

Если сравнивать Flutter с другими кроссплатформенными решениями, например, с React Native, то последний имеет существенный недостаток. Для взаимодействия с системой он использует программный мост, постоянная передача данных через который часто приводит к зависаниям при обработке сложных анимаций. Flutter решает эту проблему прямой компиляцией в

машинный код, благодаря чему приложение выглядит и работает одинаково на устройствах от любых производителей.

1.1.5 Выбор серверной инфраструктуры

Современное приложение требует надежной синхронизации и хранения пользовательских данных. Чтобы не тратить время и ресурсы на разработку собственного сервера, было принято решение использовать модель BaaS.

Проект использует платформу Firebase [2]. Она дает готовые инструменты:

– базы данных Realtime Database и Cloud Firestore для автоматической синхронизации данных;

– модуль Firebase Auth для авторизации пользователей;

– среда Cloud Functions для запуска серверного кода без прямого администрирования;

– сервисы Crashlytics и Performance Monitoring для сбора аналитики сбоев.

Хранение заметок требует структурной гибкости базы данных. Документоориентированная база Firebase сохраняет информацию в формате коллекций, разработчик просто меняет структуру данных при выпуске обновлений без миграции таблиц. Это решает проблему масштабирования.

Платформа специализируется на мобильной разработке. Инструментарий ориентирован на платформы iOS, Android и фреймворк Flutter.

Базовый тарифный план Spark предоставляется бесплатно. Стартапы используют сервис без стартовых вложений.

Платформа Firebase содержит нативную поддержку генеративных моделей Gemini. Облако дает интерфейсы для вызова нейросетей прямо из клиентского кода.

Аутентификация поддерживает аккаунт Google. Этот профиль есть у любого пользователя операционной системы Android. Вход происходит мгновенно, пользователь не тратит время на регистрацию.

1.1.6 Выбор среды разработки

Проект использует Visual Studio Code. Это легковесный редактор является одним из самых популярных инструментов для разработки на Flutter. Снижение аппаратной нагрузки при использовании VS Code ускоряет написание программного кода.

Инструмент функционирует на различных операционных системах, гарантируя разработчику аппаратную независимость. Универсальность среды позволяет переносить настроенное рабочее окружение между компьютерами.

Экосистема расширений решает задачи интеграции инструментов разработки. Установка официальных плагинов Flutter и Dart обеспечивает базовые инструменты для разработки программ.

Редактор VS Code применяется в разных сферах программирования. Унификация среды полностью исключает потребность постоянно привыкать к новому интерфейсу при смене стека технологий.

Крупные IT-компании доверяют Visual Code OSS. Например, Google создает продукты на базе форка редактора. Облачная платформа Firebase Studio и локальная агентская среда Antigravity используют эту архитектуру.

1.2 Проектирование и разработка

Данный раздел посвящен описанию архитектурных решений, обоснованию выбора технологического стека и детальному рассмотрению процесса реализации мобильного приложения для управления персональной информацией. В современных условиях избытка цифровых данных классические приложения для ведения заметок трансформируются в интеллектуальные системы, что требует применения гибких архитектурных паттернов и интеграции современных методов обработки естественного языка.

Проектирование системы велось с упором на модульность, расширяемость и отказоустойчивость. Основной целью разработки являлось создание кроссплатформенного решения, обеспечивающего не только хранение текстовых

данных, но и их интеллектуальную классификацию с помощью нейросетевых моделей, а также бесшовную синхронизацию с облачными сервисами и системными компонентами мобильного устройства.

1.2.1 Реализация архитектуры приложения

Программный каркас приложения построен на принципах Clean Architecture и Feature-First. Код строго разделен на три независимых уровня: Core, Data и Features.

Диаграмма классов на рисунке 6 показывает внутренние связи между этими компонентами.

Уровень Core содержит базовые механизмы приложения. Класс AuthService управляет авторизацией пользователя. SettingsService хранит параметры оформления и настройки нейросети. Интеграцию с операционной системой обеспечивают NotificationService для планирования локальных уведомлений и CalendarService для выгрузки событий в системный календарь. Вся работа с искусственным интеллектом вынесена в отдельный GeminiService, а SyncService отвечает за отправку медиафайлов на сервер.

Слой Data работает исключительно с хранением информации. Локальная запись файлов идет через NotesRepository на базе Hive. Синхронизацию с сервером выполняет FirestoreRepository. В этом же модуле описаны структуры данных: Note, NoteContact и StructuredNoteData.

Пользовательский интерфейс и бизнес-логика сгруппированы на уровне Features. Корневой виджет App задает тему дизайна и перехватывает внешние ссылки. Класс MainScreen управляет главной навигацией. Экран NoteEditorScreen объединяет интерфейс текстового редактора и панель инструментов искусственного интеллекта. Провайдер NotesNotifier связывает экраны с репозиториями, управляет отображением списков и контролирует фоновую загрузку данных.

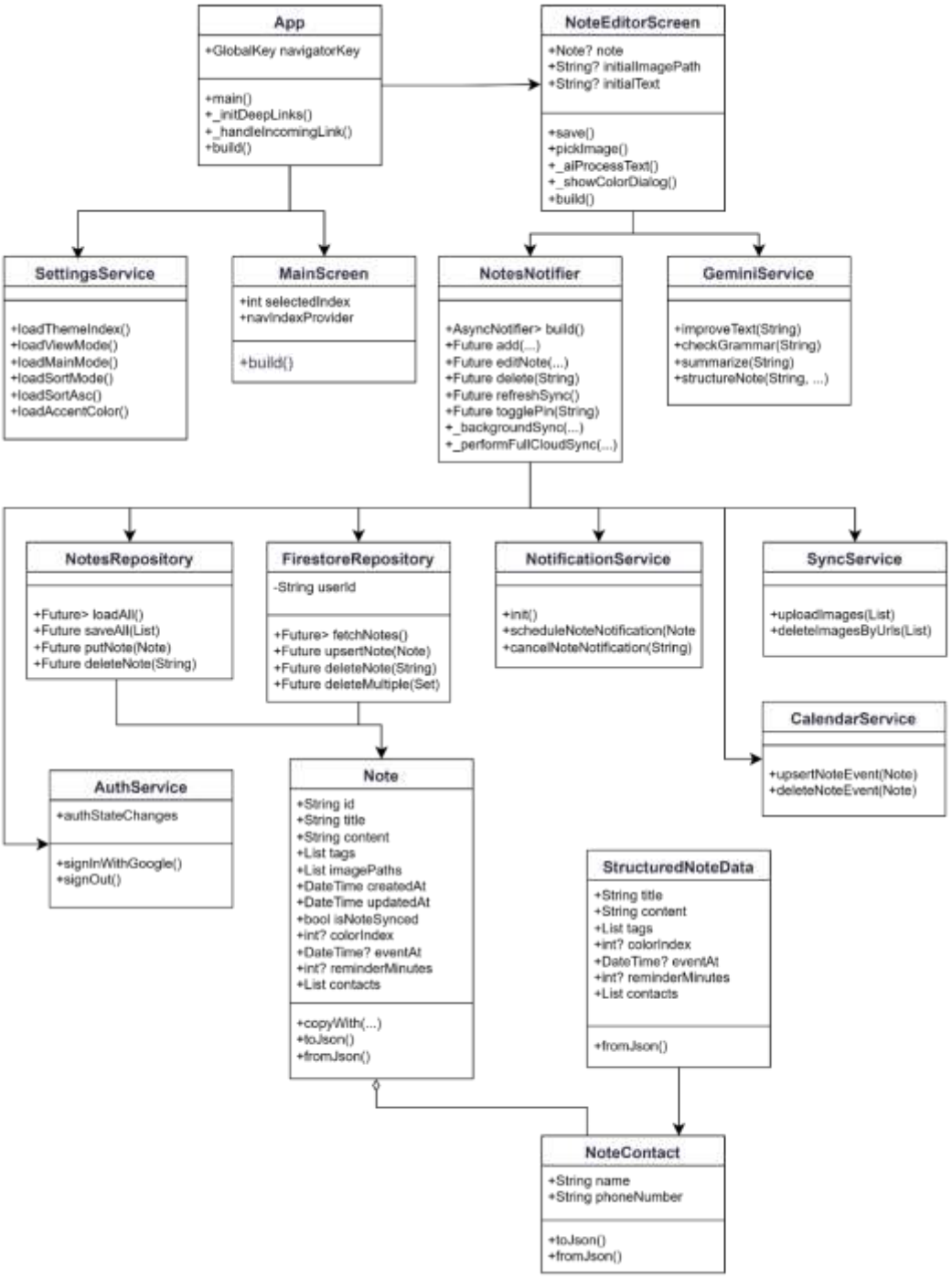


Рисунок 6 - Диаграмма классов

Библиотека Riverpod обеспечивает строгую типизацию и связывает компоненты между собой [3]. Элементы дизайна получают данные только через провайдеры. Разработчик может полностью переписать алгоритмы нейросети или облачной синхронизации без изменения визуальной части приложения.

Логика взаимодействия указанных компонентов и последовательность вызовов при выполнении ключевых операций приложения отражена на рисунке 7.

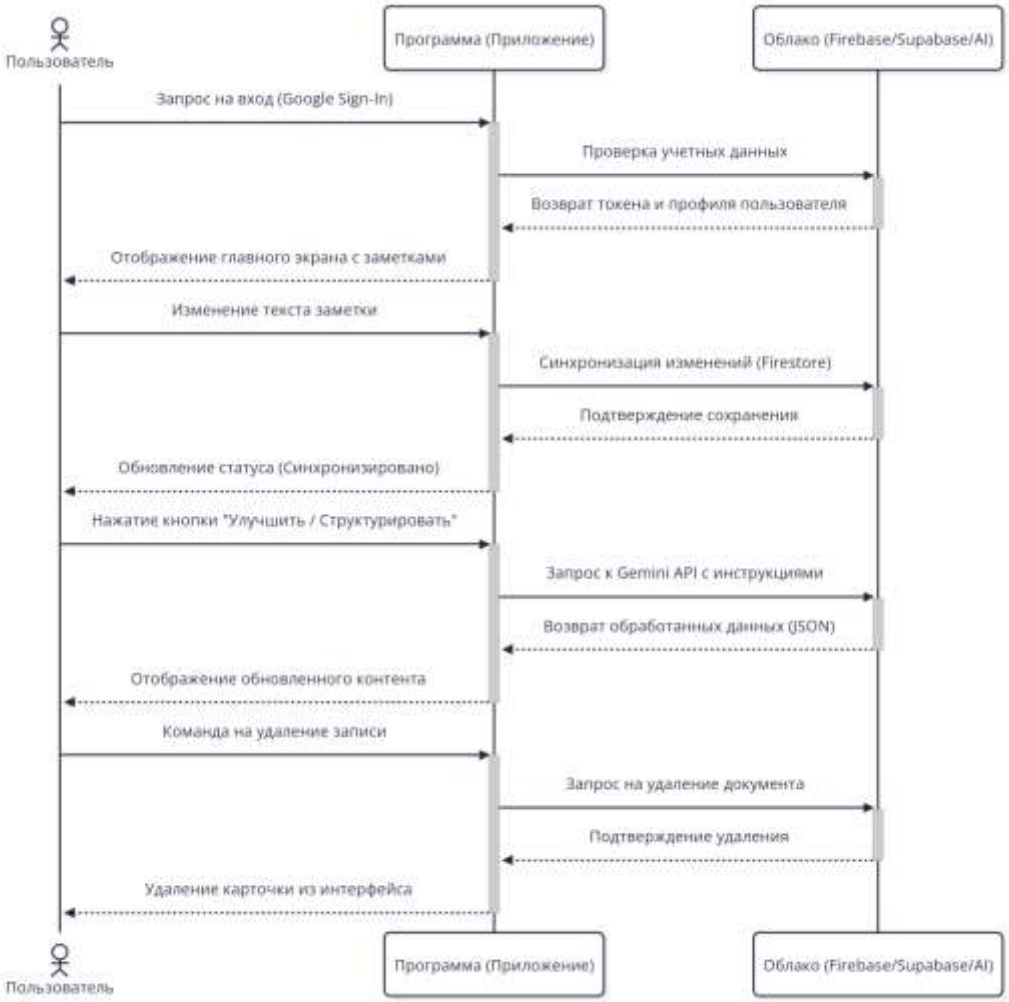


Рисунок 7 - Диаграмма последовательностей

1.2.2 Проектирование модели данных и структуры хранения

Основным элементом данных приложения выступает класс Note. В нём инкапсулированы текст, медиафайлы, параметры планирования (интеграция с календарем) и метаданные:

```
class Note {
  final String id;
  final String title;
  final String content;
```

```
  final List<String> tags;
  final List<String> imagePaths;
  final int? colorIndex;

  // Интеграция с системным календарем
  final DateTime? eventAt;
  final String? calendarEventId;
  final String? calendarId;

  // Флаг успешной загрузки в облако
  final bool isNoteSynced;
  bool get isSynced => isNoteSynced;

  // Автоматическая классификация (AI-маркировка)
  bool get isAiProcessed => tags.contains('AI');

  // ... (конструктор и остальные поля)
}
```

Для локального хранения выбрана NoSQL-база данных Hive [4]. Объекты хранятся в виде JSON-строк. Чтобы при чтении большого количества записей не блокировать основной поток интерфейса (UI-поток), ресурсоемкая операция десериализации вынесена в отдельный изолят с помощью функции compute:

```
// Парсинг больших данных вынесен из основного потока
List<Note> _parseNotes(List<String> raw) {
  return raw
      .map((e) => Note.fromJson(jsonDecode(e) as
Map<String, dynamic>))
      .toList();
}

class NotesRepository {
  static const _boxName = 'notesBox';
  Box<String> get _box => Hive.box<String>(_boxName);

  Future<List<Note>> loadAll() async {
    final raw = _box.values.toList();
    if (raw.isEmpty) return [];

    // Выполнение парсинга в отдельном изоляте
    return compute(_parseNotes, raw);
  }

  Future<void> putNote(Note note) async {
    await _box.put(note.id, jsonEncode(note.toJson()));
  }
}
```

Распределенное хранение и синхронизация (гибридная архитектура) обеспечиваются связкой Google Cloud Firestore (для текстовых данных и статусов) и Supabase Storage (для хранения медиафайлов).

Алгоритм синхронизации при наличии сети автоматически загружает локальные изображения в Supabase, заменяя пути в imagePaths на публичные URL-адреса, и устанавливает флаг isNoteSynced = true.

Пример итоговой структуры объекта Note после сериализации (toJson):

```
{
      "id": "7f2a1b9c-4d5e-4f6a-8b9c-0d1e2f3a4b5c",
      "title": "Встреча по проекту",
      "content": "Обсудить интеграцию с Gemini API и
Supabase",
      "tags": ["Работа/Разработка", "AI"],
      "imagePaths": [
        "https://usgquikdsefdgmuumiyu.supabase.co/storage/v1/ob
ject/public/notes/image_01.jpg"
      ],
      "createdAt": "2026-05-04T10:00:00.000Z",
      "updatedAt": "2026-05-04T12:30:00.000Z",
      "isPinned": true,
      "colorIndex": 2,
      "isNoteSynced": true,
      "eventAt": "2026-05-05T09:00:00.000Z",
      "reminderMinutes": 15,
      "calendarEventId": "cal_event_998877",
      "repeatMode": "none",
      "contacts": [
        {
          "name": "Иван Иванов",
          "phoneNumber": "+79991234567"
        }
      ]
    }
```

1.2.3 Интеграция с облачной инфраструктурой

Функционирование программного комплекса обеспечивается мультиоблачной архитектурой, объединяющей возможности платформ Google Firebase и Supabase для распределения нагрузки.

Диаграмма на рисунке 7 иллюстрирует взаимодействие мобильного клиента с сервисами Firebase и Supabase.

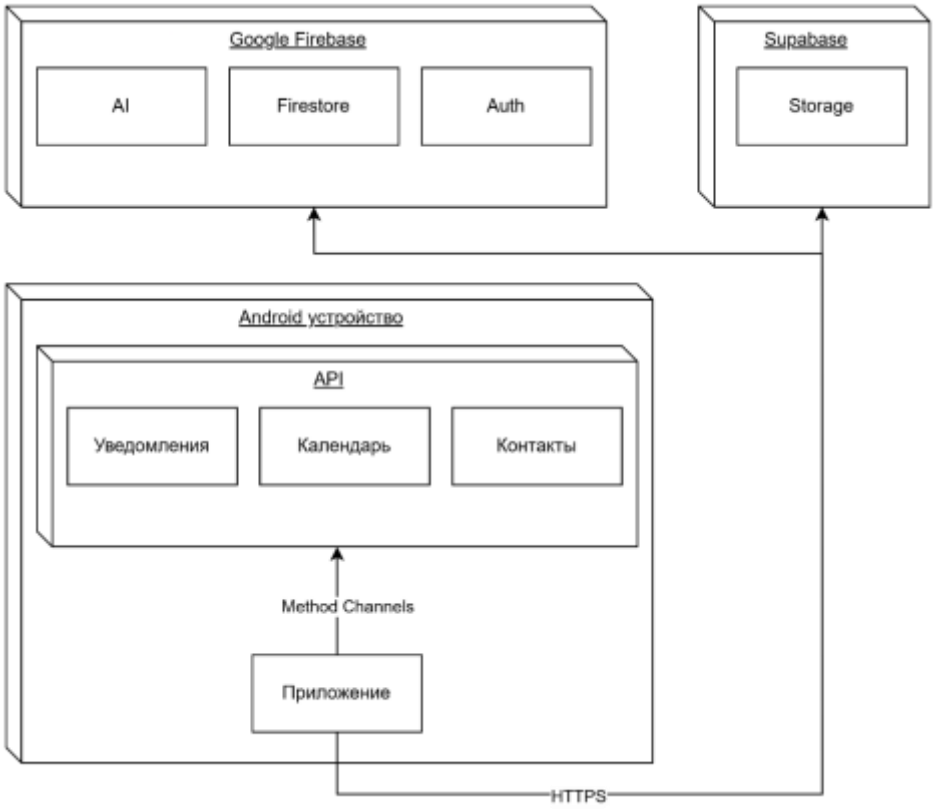


Рисунок 7 - Диаграмма развертывания

Инициализация обоих сервисов выполняется в main.dart при старте приложения:

```
// main.dart
await Firebase.initializeApp(options:
DefaultFirebaseOptions.currentPlatform);
await FirebaseAppCheck.instance.activate(
  androidProvider: AndroidProvider.playIntegrity,
);

await Supabase.initialize(
  url: 'https://usgquikdsefdgmuumiyu.supabase.co',
  anonKey: '...',
);
```

Firebase Auth и Cloud Firestore обеспечивают доступ к данным конкретного пользователя. Путь к коллекции заметок строится по шаблону users/{userId}/notes/{noteId}, что позволяет изолировать данные на уровне Security Rules:

```
// firestore_repository.dart
class FirestoreRepository {
```

```
            final String userId;

            CollectionReference<Map<String, dynamic>> get _notesRef
=>
                FirebaseFirestore.instance
                    .collection('users')
                    .doc(userId)
                    .collection('notes');

            Future<void> upsertNote(Note note) async {
              await _notesRef.doc(note.id).set(note.toJson());
            }
            Future<void> deleteMultiple(Set<String> ids) async {
              final batch = FirebaseFirestore.instance.batch();
              for (final id in ids) {
                batch.delete(_notesRef.doc(id));
              }
              await batch.commit(); // Все удаления — одна атомарная
операция
            }
          }
```

Рисунок 7 демонстрирует иерархическую структуру данных и работу правил безопасности.

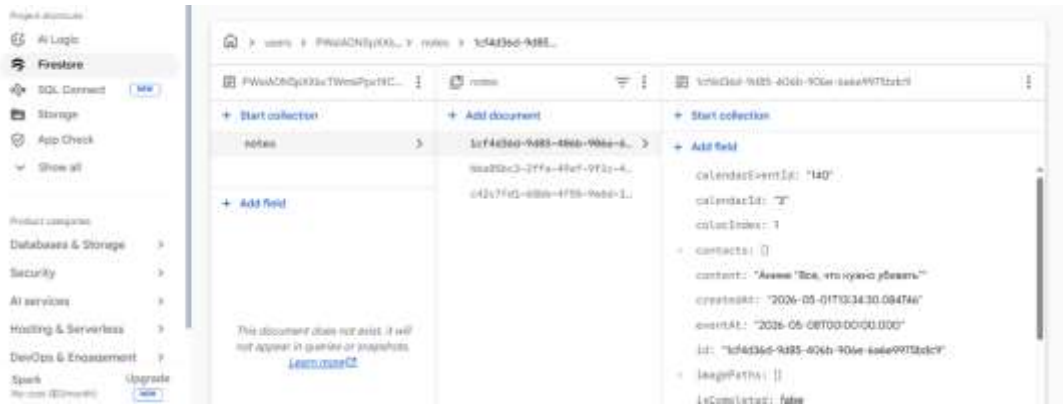


Рисунок 7 – Интерфейс консоли Firebase Firestore с отображением структуры документов пользователей.

Write Batch гарантирует атомарность: при массовом удалении группа операций либо выполняется целиком, либо не выполняется вовсе.

Supabase Storage используется для хранения изображений [5]. SyncService выполняется в три этапа — сжатие, загрузка, получение публичной ссылки:

```
// sync_service.dart
```

```
        Future<List<String>> uploadImages(List<String> paths) async
{
          for (String path in paths) {
     if (path.startsWith('http')) { resultUrls.add(path); continue;
}
             final XFile? compressedFile =
                 await FlutterImageCompress.compressAndGetFile(
                   file.absolute.path, targetPath, quality: 70,
minWidth: 1024,
                 );

             {userId}/{timestamp}.jpg
             final fileName =
'$userId/${DateTime.now().millisecondsSinceEpoch}.jpg';
             await _supabase.storage.from('notes').upload(fileName,
File(compressedFile!.path));
             final String publicUrl =
_supabase.storage.from('notes').getPublicUrl(fileName);
             resultUrls.add(publicUrl);
           }
           return resultUrls;
         }
```

При успешном завершении система обновляет состояние объекта Note. Локальный путь к изображению в массиве imagePaths заменяется на полученный URL-адрес, а параметру isNoteSynced присваивается значение true. Процедура удаления файлов из облака реализована через обратное преобразование: алгоритм анализирует строку URL, выделяет из неё относительный путь к объекту и вызывает метод storage.remove:

```
     Future<void> deleteImageByUrl(String url) async {
       final uri = Uri.parse(url);
       final idx = uri.path.indexOf('/notes/');
       final path = uri.path.substring(idx +
  '/notes/'.length);
       await _supabase.storage.from('notes').remove([path]);
     }
```

На рисунке 8 представлен интерфейс хранилища Supabase storage.

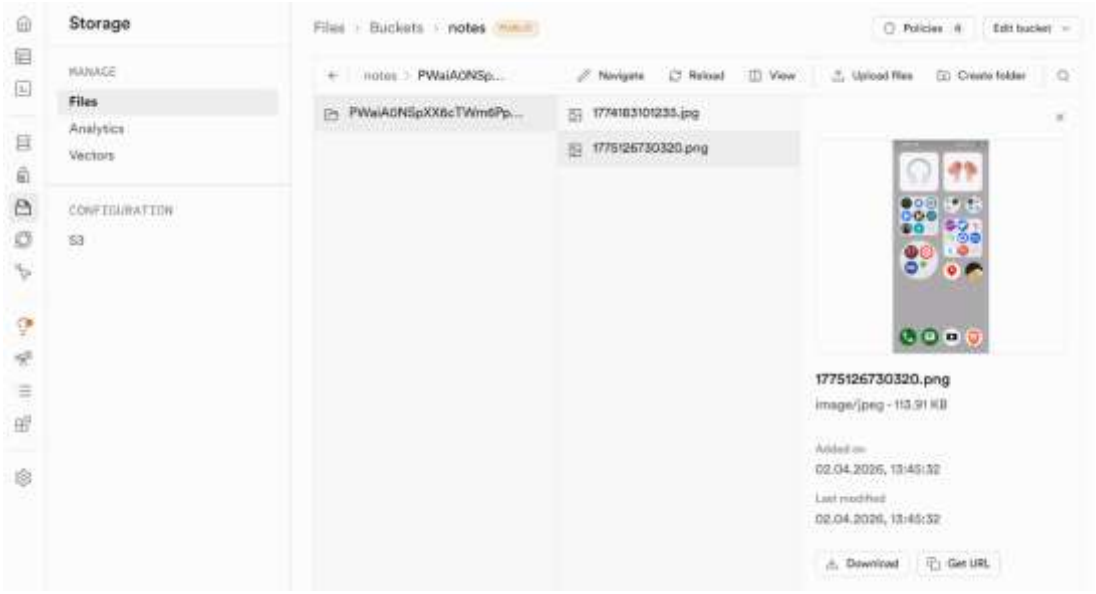


Рисунок 8 - Организация объектного хранилища в панели управления Supabase.

1.2.4 Разработка интеллектуального модуля

Интеллектуальный функционал реализован в классе GeminiService, который использует модель gemini-2.5-flash-lite и поддерживает два независимых канала запросов [6].

Основной канал работает через Firebase AI Logic. Резервный канал переключается на Google AI Studio API при наличии пользовательского ключа. Выбор канала выполняется автоматически в методе _generateWithRetry:

```
      // gemini_service.dart
         class GeminiService {
           final String _modelName = 'gemini-2.5-flash-lite';

           GeminiService(this._ref) {
             _primaryModel =
FirebaseAI.googleAI().generativeModel(
                 model: _modelName,
                 generationConfig: GenerationConfig(responseMimeType:
'application/json'),
               );
           }

           Future<String?> _generateWithRetry(String prompt) async
{
```

```
            final usePersonalKey =
_ref.read(useFallbackApiKeyProvider);
            final personalKey =
_ref.read(fallbackApiKeyProvider);

            if (usePersonalKey && personalKey != null &&
personalKey.isNotEmpty) {
              return _generateViaAIStudio(prompt, personalKey);
            } else {
              return _generateViaFirebase(prompt);
            }
          }
        }
```

Ключевой метод модуля — structureNote. Он принимает необработанный текст заметки и возвращает структурированный объект StructuredNoteData.

Промпт передаёт дату создания заметки как точку отсчёта для вычисления относительных дат, справочник контактов пользователя для сопоставления имён и список существующих папок/тегов для поддержания иерархии:

```
    Future<StructuredNoteData?> structureNote(
      String rawContent, {
      DateTime? noteDate,
      List<NoteContact> userContacts = const [],
      List<String> existingFolders = const [],
    }) async {
      if (rawContent.trim().isEmpty) return null;

      final referenceDate = noteDate ?? DateTime.now();

      String contactsContext = '';
      if (userContacts.isNotEmpty) {
        final simplified = userContacts
            .map((c) => {'name': c.name, 'phone': c.phoneNumber})
            .toList();
        contactsContext = '\nСПРАВОЧНИК КОНТАКТОВ
ПОЛЬЗОВАТЕЛЯ...\n${jsonEncode(simplified)}\n';
      }

      final prompt = '''
    Дата создания заметки (точка отсчета):
${referenceDate.toIso8601String()}.
    Строго используй ЭТУ дату как "сегодня" для высчитывания всех
относительных дат.
    $contactsContext
    ...
    $rawContent
    ''';
```

```
      final text = await _generateWithRetry(prompt);
      final jsonStr = text!.replaceAll('```json',
'').replaceAll('```', '').trim();
      return StructuredNoteData.fromJson(jsonDecode(jsonStr));
    }
```

Для пакетной обработки реализован метод structureNotesBatch, который отправляет массив заметок одним запросом к API. Ответ возвращается в виде словаря {noteId: StructuredNoteData}:

```
    Future<Map<String, StructuredNoteData>>
structureNotesBatch(
      List<Note> notes, {
      List<NoteContact> userContacts = const [],
      List<String> existingFolders = const [],
    } ) async {
      if (notes.isEmpty) return {};

      final notesList = notes.map((n) => {
        'id': n.id,
        'content': n.content,
        'createdAt': n.createdAt.toIso8601String(),
      }).toList();

      final prompt = '''
    ...
    Для КАЖДОЙ заметки используй её поле "createdAt" как точку
отсчета (сегодня) для высчитывания всех относительных дат в тексте
этой заметки.
    ВХОДНЫЕ ЗАМЕТКИ ДЛЯ ОБРАБОТКИ (JSON):
    ${jsonEncode(notesList)}
    ...
    ''';

      final text = await _generateWithRetry(prompt);
      final Map<String, dynamic> resultsMap = jsonDecode(
        text!.replaceAll('```json', '').replaceAll('```',
'').trim()
      );
     final Map<String, StructuredNoteData> finalResults = {};
      resultsMap.forEach((id, data) {
        finalResults[id] = StructuredNoteData.fromJson(data as
Map<String, dynamic>);
      });

      return finalResults;
    }
```

Ответы модели валидируются через jsonDecode. При нарушении выбрасывается исключение, которое перехватывается на уровне провайдера. Признак обработки записи хранится в теге 'AI'.

Внедрение Firebase AI Logic позволило организовать контроль за производительностью нейросетевых запросов и качеством генерируемого контента. Встроенные средства мониторинга собирают телеметрию по каждому обращению к модели Gemini. Платформа регистрирует время отклика сервера, объем израсходованных токенов и итоговый статус выполнения операции.

Полученные данные визуализируются в виде интерактивных графиков в консоли администратора. Функционал панели позволяет строго отслеживать потребление ресурсов, предотвращая выход за рамки квот и перерасход средств на облачную инфраструктуру. Интерфейс проиллюстрирован на рисунке 9.

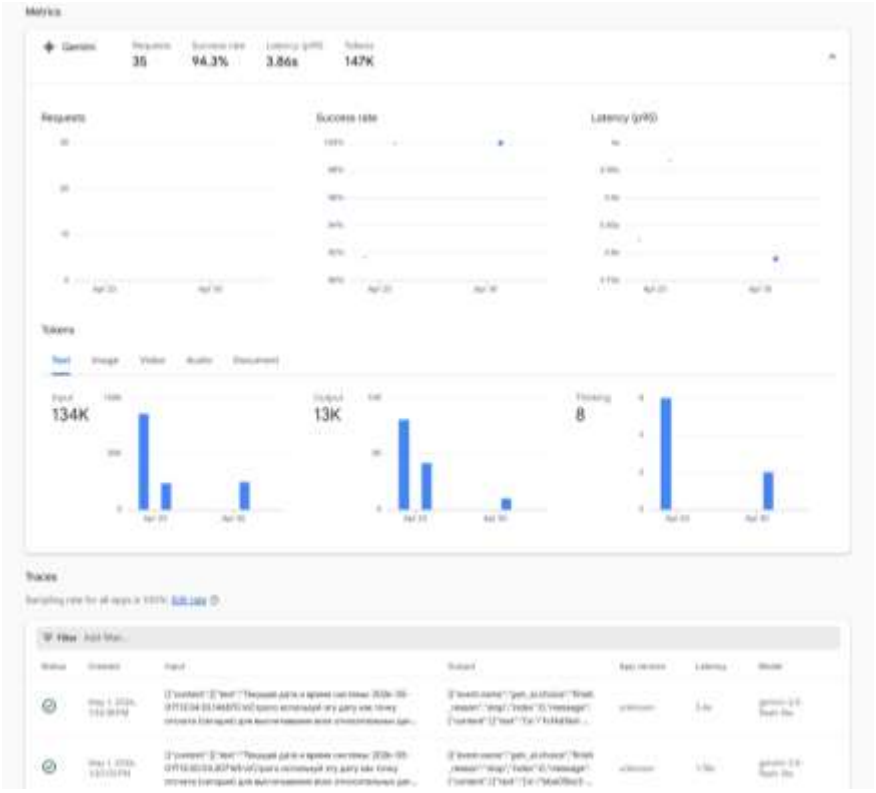


Рисунок 9 - Панель мониторинга Firebase AI Logic.

1.3 Тестирование

На опытной эксплуатации программа продемонстрировала стабильную работу всех заявленных компонентов.

Успешно протестирован следующий функционал:

– создание, редактирование и управление заметками;

– двухэтапная процедура удаления данных с использованием корзины;

– фоновая синхронизация данных с облачной базой данных Cloud Firestore;

– прикрепление медиа файлов с размещением в объектном хранилище Supabase Storage;

– интеллектуальный анализ текста и извлечение сущностей через модуль Google AI Logic;

– аутентификация пользователей на основе Google OAuth.

Разработанный программный продукт готов к эксплуатации, упрощает процесс накопления знаний и снижает нагрузку на пользователя при обработке данных.

1.3.1 Тестирование базового функционала системы

Тестирование модуля авторизации выполнялось через Google OAuth. Ожидается, что при нажатии на кнопку входа приложение вызовет нативное системное окно выбора аккаунта, передаст токен безопасности в Firebase Auth, создаст профиль пользователя. На рисунке 10 показан интерфейс выбора аккаунта Google в мобильном приложении, а рисунок 11 демонстрирует запись пользователя с типом провайдера Google в веб-консоли Firebase Auth.

Рисунок 11 — Панель управления Firebase Auth.

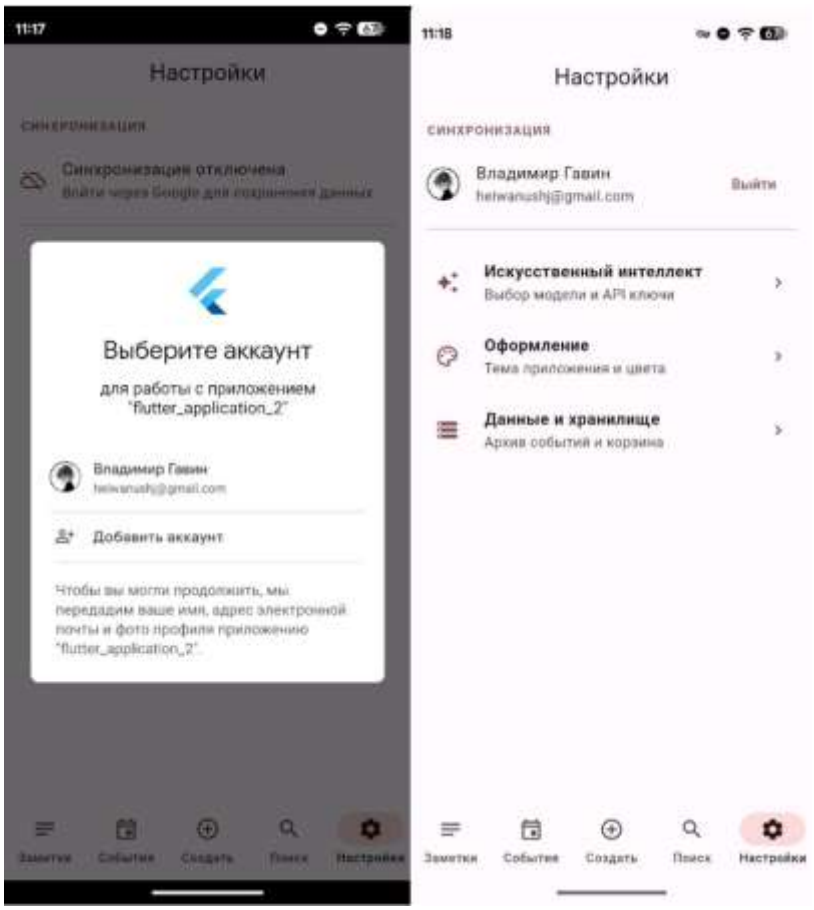


Рисунок 10 — Окно авторизации через Google-аккаунт в приложении.

Проверка создания заметки включает в себя ввод текста, прикрепление изображений, выбор контактов из телефонной книги и установку даты напоминания. Ожидаемое поведение системы заключалось в выполнении нескольких процессов: сжатии фотографии, её загрузке в облачное хранилище Supabase Storage, формировании задачи в системном календаре смартфона и сохранении единого JSON-документа в Cloud Firestore.

На рисунке 12 представлен внешний вид готовой заметки в редакторе программы с привязанным событием и контактами. Рисунок 13 демонстрирует загруженный файл в хранилище Supabase Storage, а рисунок 14 отображает структуру документа в консоли Cloud Firestore.

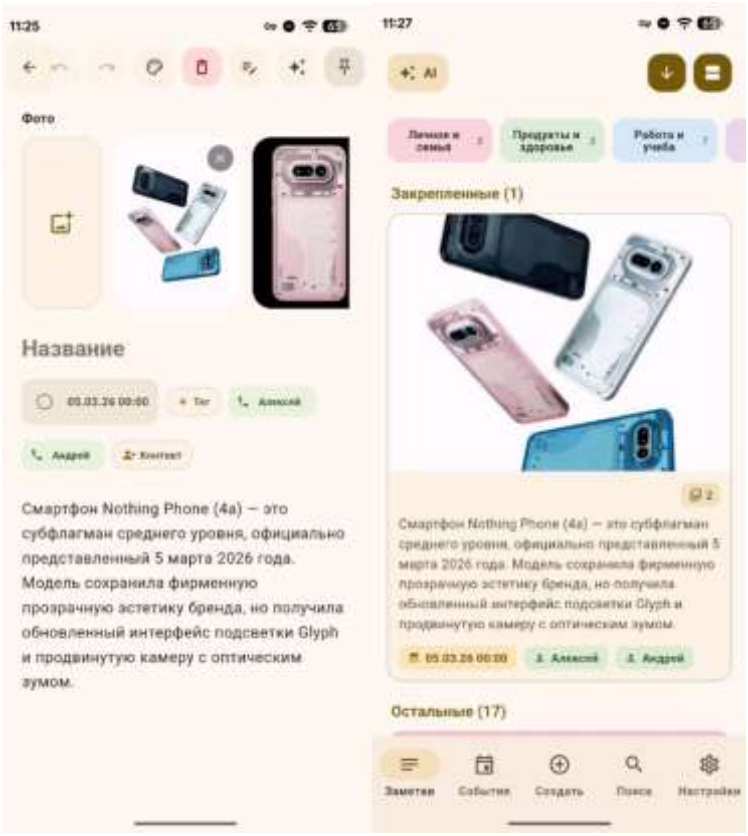


Рисунок 12 — Редактор заметки с заполненными данными, контактами и датой.

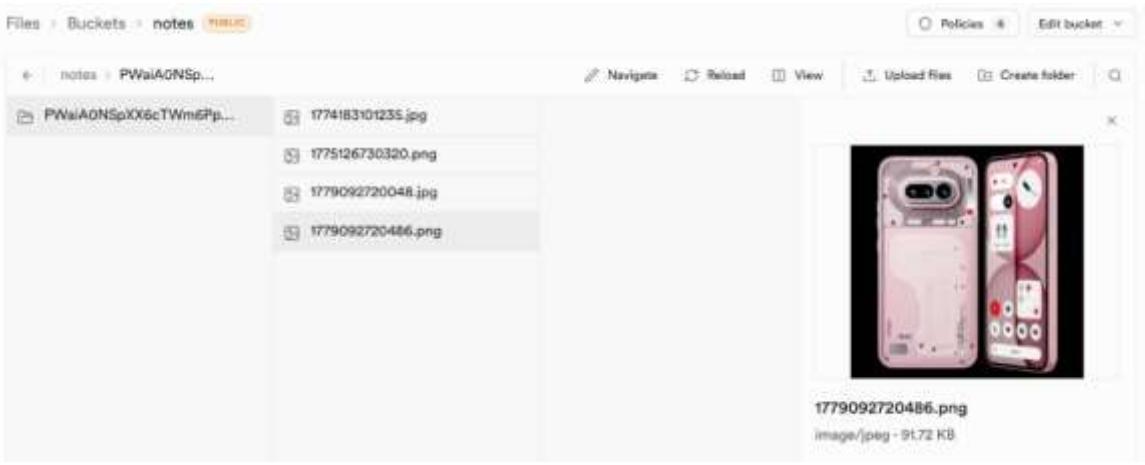


Рисунок 13 — Хранилище Supabase Storage.

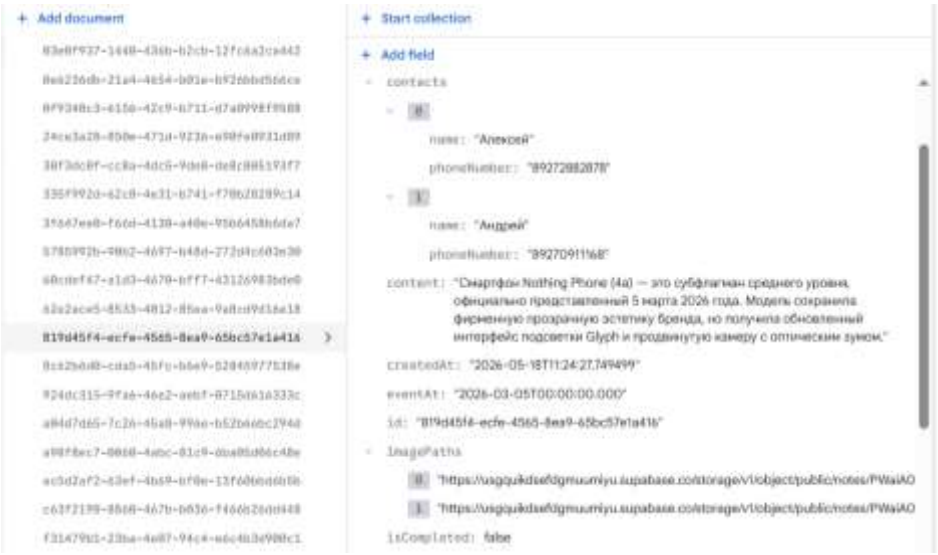


Рисунок 14 — Документ заметки в веб-консоли Cloud Firestore.

Верификация удаления. Ожидается, что карточка исчезнет из общего списка и перейдет в корзину, а документ на сервере Firestore сменит флаг isDeleted с false на true. На рисунке 15 представлен интерфейс корзины в приложении, а рисунок 16 иллюстрирует состояние консоли Firestore.

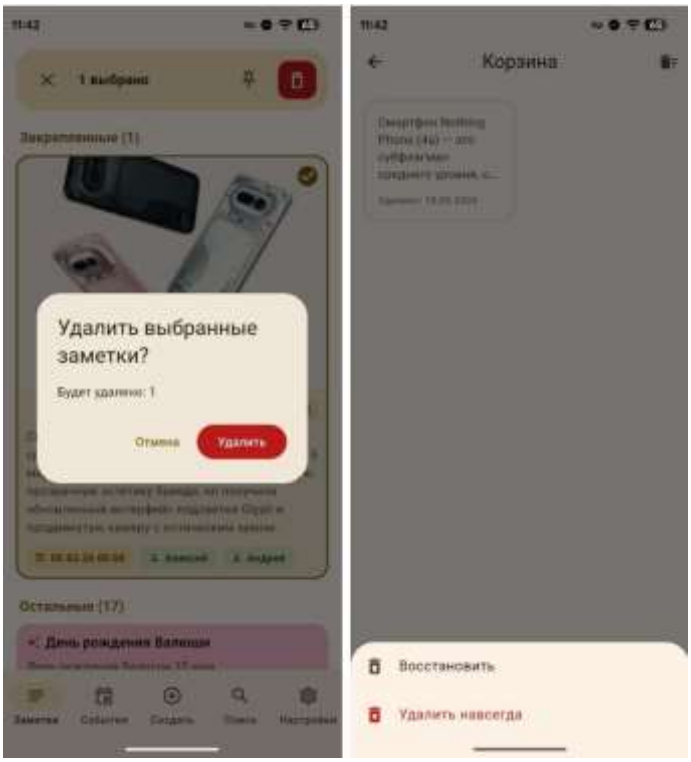


Рисунок 15 — Экран корзины в мобильном приложении.

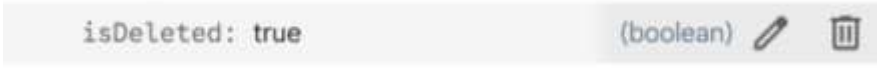


Рисунок 16 — Смена флага isDeleted в Firestore.

1.3.2. Тестирование интеллектуальной обработки

Для проверки точности извлечения сущностей использовалась тестовая заметка с содержанием: “В пятницу в два часа дня нужно обязательно созвониться с Денисом и Алексеем, чтобы обсудить интеграцию бэкенда и фронтенда нашего приложения. Надо проверить, как работает авторизация через Google, и протестировать загрузку фоток в Supabase. Алексей обещал скинуть новую структуру базы данных Cloud Firestore, а Денис посмотрит логи работы ИИ на сервере. Скинуть отчет руководителю до конца недели”.

Результаты сопоставления ожидаемого и фактического поведения системы зафиксированы в таблице 1.2.

Таблица 2 — Результаты тестирования семантического разбора тестовой заметки

Элемент анализа	Ожидание	Результат	Статус
Заголовок	Создание заголовка	“Обсуждение интеграции бекенда и фронтенда”	+
Дата и время	Пятница, время 14:00	2026-05-22T14:00	+
Контакты	Алексей, Денис	Алексей, Денис	+
Иерархия папок	Определение категории «Работа»	«Работа и учеба/Проекты»	+

На рисунке 17 представлено визуальное сравнение исходной неструктурированной записи и итоговой карточки с заполненными метаданными

папок. На рисунке 18 представлены логи Firebase AI logic, подтверждающие успешную генерацию ответа нейросети Gemini-2.5-flash-lite.

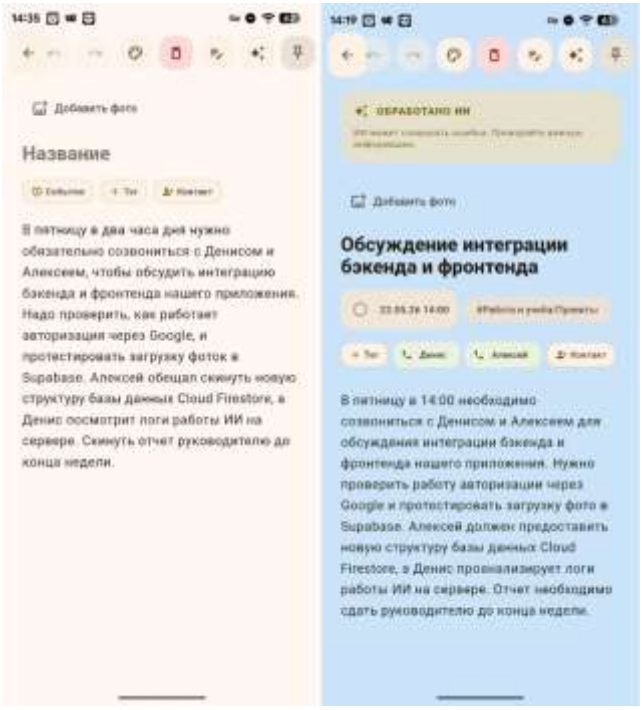


Рисунок 17 — Сравнение исходного материала и результата работы интеллектуального модуля.

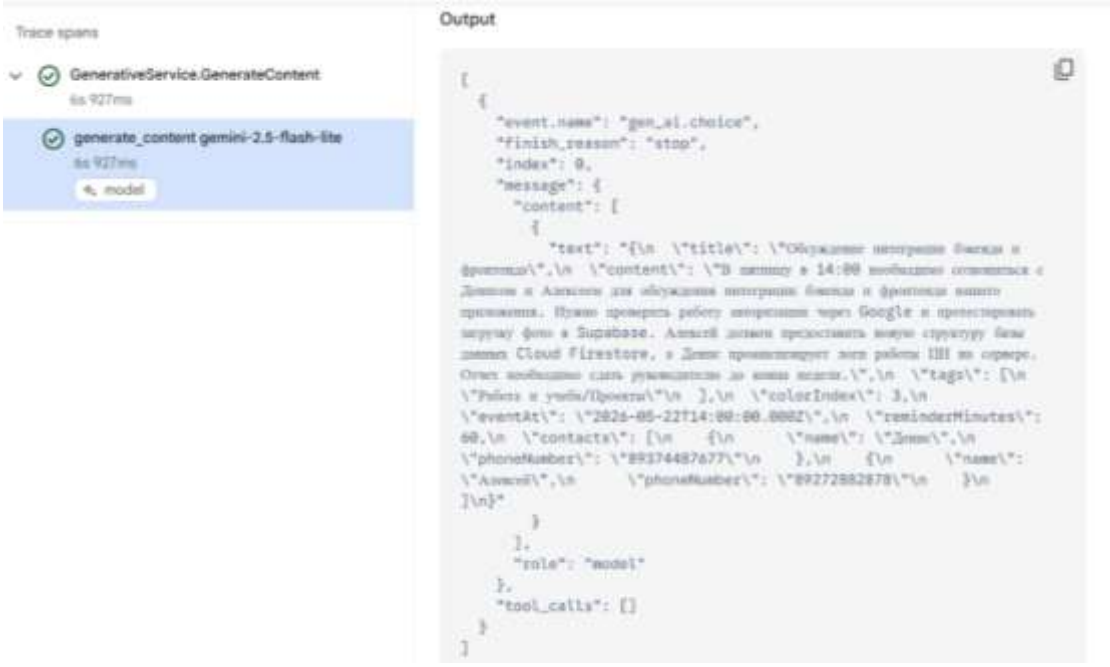


Рисунок 19 —Консоль Firebase AI Logic.

2 Безопасность жизнедеятельности

Ежедневное взаимодействие с цифровыми устройствами регламентируется нормами охраны здоровья и технической эксплуатации. Соблюдение установленных эргономических правил и стандартов безопасности позволяет защитить организм от негативного физического воздействия, сохранить зрение и осанку, а также предотвратить преждевременные поломки электронного оборудования.

В данном пункте будут рассмотрены основные правила и инструкции по работе человека с компьютером и мобильным телефоном.

2.1 Правила и техника безопасности при работе с ПК.

Монитор необходимо располагать на уровне глаз или немного ниже на расстоянии 50–70 сантиметров от лица. Для обеспечения правильного кровообращения поверхность стола требуется устанавливать на уровне локтей, а клавиатуру и мышь размещать на одной высоте, сохраняя нейтральное положение запястий. Во время работы спину держать ровно, опираясь на спинку кресла с поддержкой поясничного отдела. Каждые 40–50 минут обязательно делать регламентированные перерывы для выполнения физической разминки и снятия мышечного напряжения [7].

Системный блок и периферийные устройства запрещается подвергать воздействию влаги. Напитки и жидкости не допускается ставить рядом с техникой во избежание короткого замыкания при случайном проливании. Все соединительные кабели требуется надежно изолировать и аккуратно проложить, исключая риск их механического повреждения. Вентиляционные отверстия оборудования строго запрещено перекрывать посторонними предметами, так как нарушение циркуляции воздуха приводит к перегреву комплектующих и потенциальному возгоранию.

По завершении работы компьютер необходимо корректно выключать через интерфейс операционной системы. Очистку экрана, клавиатуры и корпуса от пыли проводить только специализированными салфетками или спреями при полностью обесточенном оборудовании.

2.2 Правила и техника безопасности при работе со смартфоном

Длительное взаимодействие с экраном создает нагрузку на зрение и опорно-двигательный аппарат. Для снижения напряжения глаз необходимо применять правило: каждые 20 минут отводить взгляд от дисплея на расстояние около 6 метров в течение 20 секунд. Устройство требуется держать на уровне глаз, чтобы не наклонять голову вперед и не перегружать шейный отдел позвоночника. Синий свет экрана подавляет выработку гормона сна, поэтому гаджеты рекомендуется откладывать как минимум за час до сна.

При переходе проезжей части телефон всегда убирать в карман и снимать наушники для полноценного контроля окружающей обстановки. Категорически запрещается использовать смартфон за рулем автомобиля. Взгляд на экран даже на пару секунд ведет к потере контроля над дорогой и аварийным ситуациям.

Заряжать телефон допускается только оригинальными или сертифицированными адаптерами и кабелями. Строго запрещено оставлять подключенный к сети аппарат под прямыми солнечными лучами, на приборной панели автомобиля или под подушкой. Нарушение температурного режима приводит к вздутию или воспламенению аккумулятора. Если устройство сильно нагрелось, его следует немедленно отключить от розетки и оставить на твердой прохладной поверхности до полного остывания [8].

Заключение

В ходе выполнения выпускной квалификационной работы было спроектировано и реализовано мобильное приложение для интеллектуального управления персональной информацией. Итоговый программный продукт представляет собой систему, объединяющую традиционные методы ведения заметок с современными возможностями генеративного искусственного интеллекта.

В процессе реализации проекта были получены навыки разработки на языке Dart с использованием фреймворка Flutter. Получен практический опыт построения архитектуры на базе Riverpod, а также интеграции API нейросетевых моделей Gemini в мобильный клиент. Работа над проектом позволила освоить принципы мультиоблачного взаимодействия с сервисами Firebase и Supabase.

Разработанное приложение полностью соответствует поставленным требованиям и готово к эксплуатации. В качестве перспектив развития проекта выделяются внедрение голосового ввода с автоматической расшифровкой и использование нейросетей для анализа долгосрочных паттернов активности пользователя. Выполненная работа демонстрирует, что интеграция ИИ и глубокая связь с функциями операционной системы являются ключевыми факторами создания современных инструментов продуктивности.

Список использованных источников

1. Документация к фреймворку Flutter. [Электронный ресурс] / Режим доступа: https://docs.flutter.dev/ свободный. (дата обращения 06.05.2026).
2. Официальная документация к облачной платформе Firebase. [Электронный ресурс] / Режим доступа: https://firebase.google.com/docs свободный. (дата обращения 06.05.2026).
3. Документация по библиотеке управления состоянием Riverpod. [Электронный ресурс] / Режим доступа: https://riverpod.dev/docs/introduction/getting_started свободный. (дата обращения 06.05.2026).
4. Документация к NoSQL базе данных Hive. [Электронный ресурс] / Режим доступа: https://docs.hivedb.dev/ свободный. (дата обращения 06.05.2026).
5. Справочное руководство разработчика Supabase. [Электронный ресурс] / Режим доступа: https://supabase.com/docs свободный. (дата обращения 06.05.2026).
6. Руководство по интеграции нейросетей Google Gemini API. [Электронный ресурс] / Режим доступа: https://ai.google.dev/docs свободный. (дата обращения 06.05.2026).
7. Эргономические особенности оптимизации длительной работы за компьютером [Электронный ресурс] / Режим доступа: https://cyberleninka.ru/article/n/ergonomicheskie-osobennosti-optimizatsii-dlitelnoy-raboty-za-kompyuterom свободный. (дата обращения 06.05.2026).
8. Влияние сотовых телефонов на здоровье человека. [Электронный ресурс] / Режим доступа: https://cyberleninka.ru/article/n/vliyanie-sotovyh-telefonov-na-zdorovie-cheloveka свободный. (дата обращения 06.05.2026).

Приложение А

(обязательное)

Андроид - приложение «Интеллектуальные заметки»

UML– диаграммы

Диаграмма прецендентов

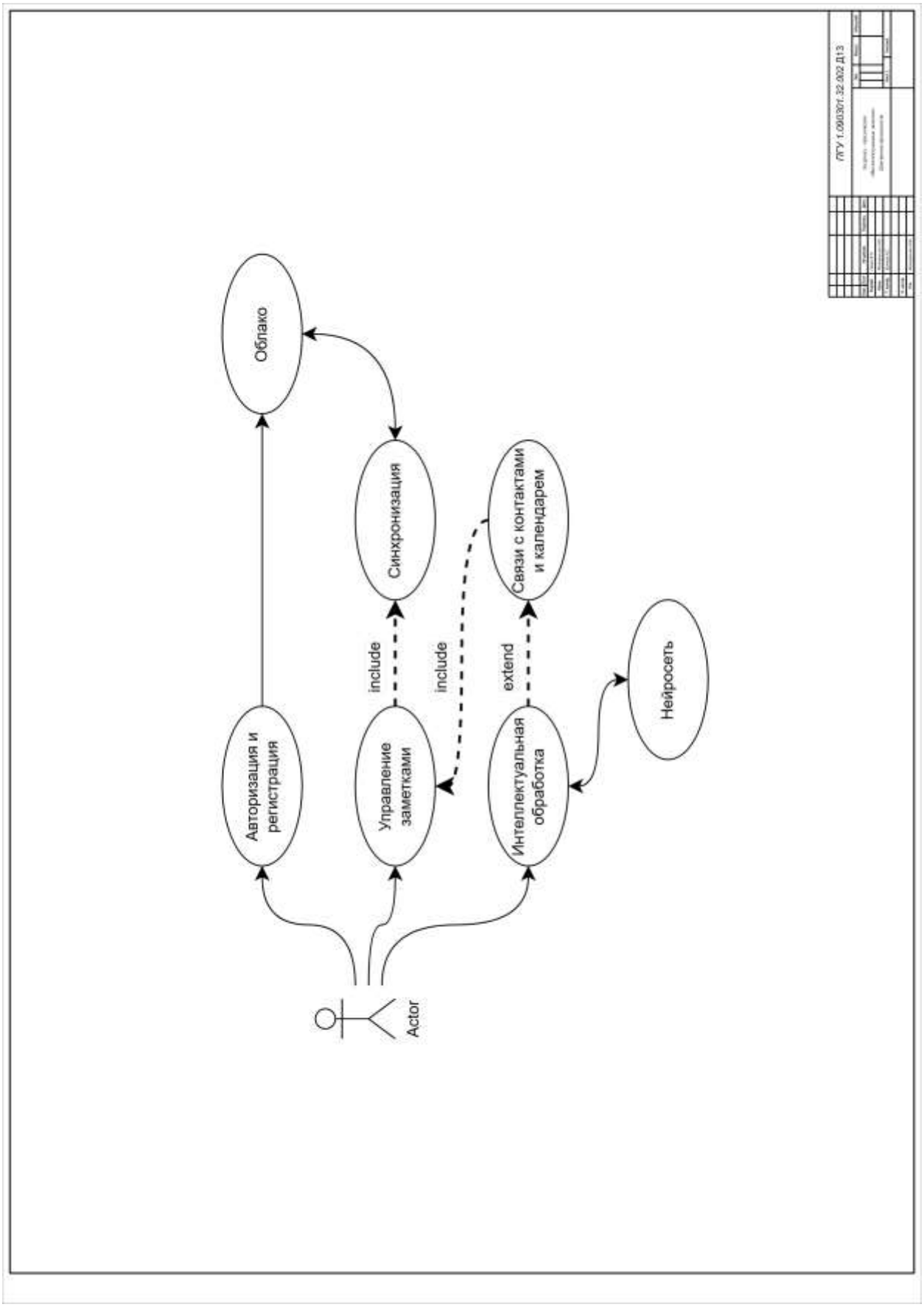


Диаграмма классов

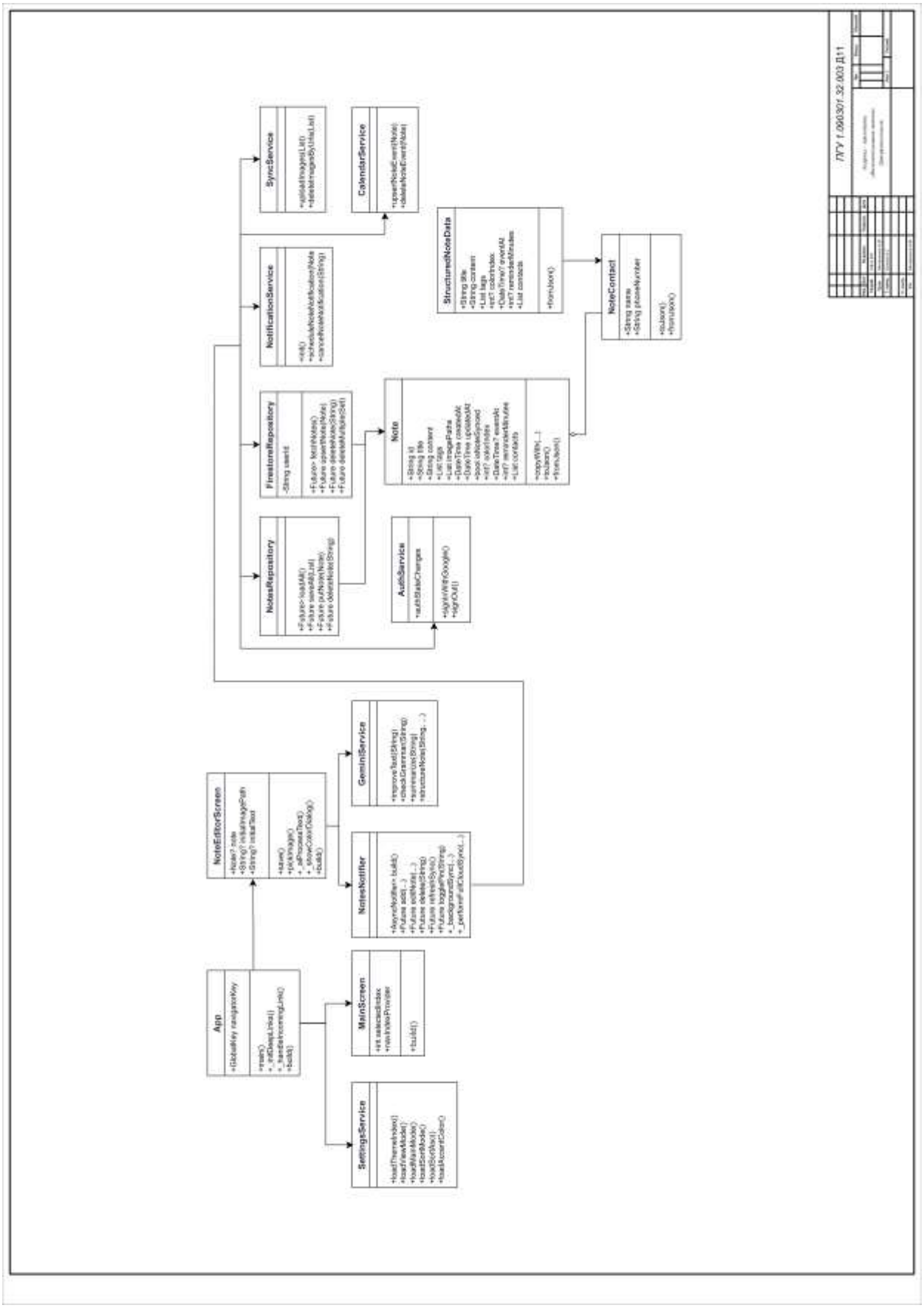


Диаграмма последовательностей

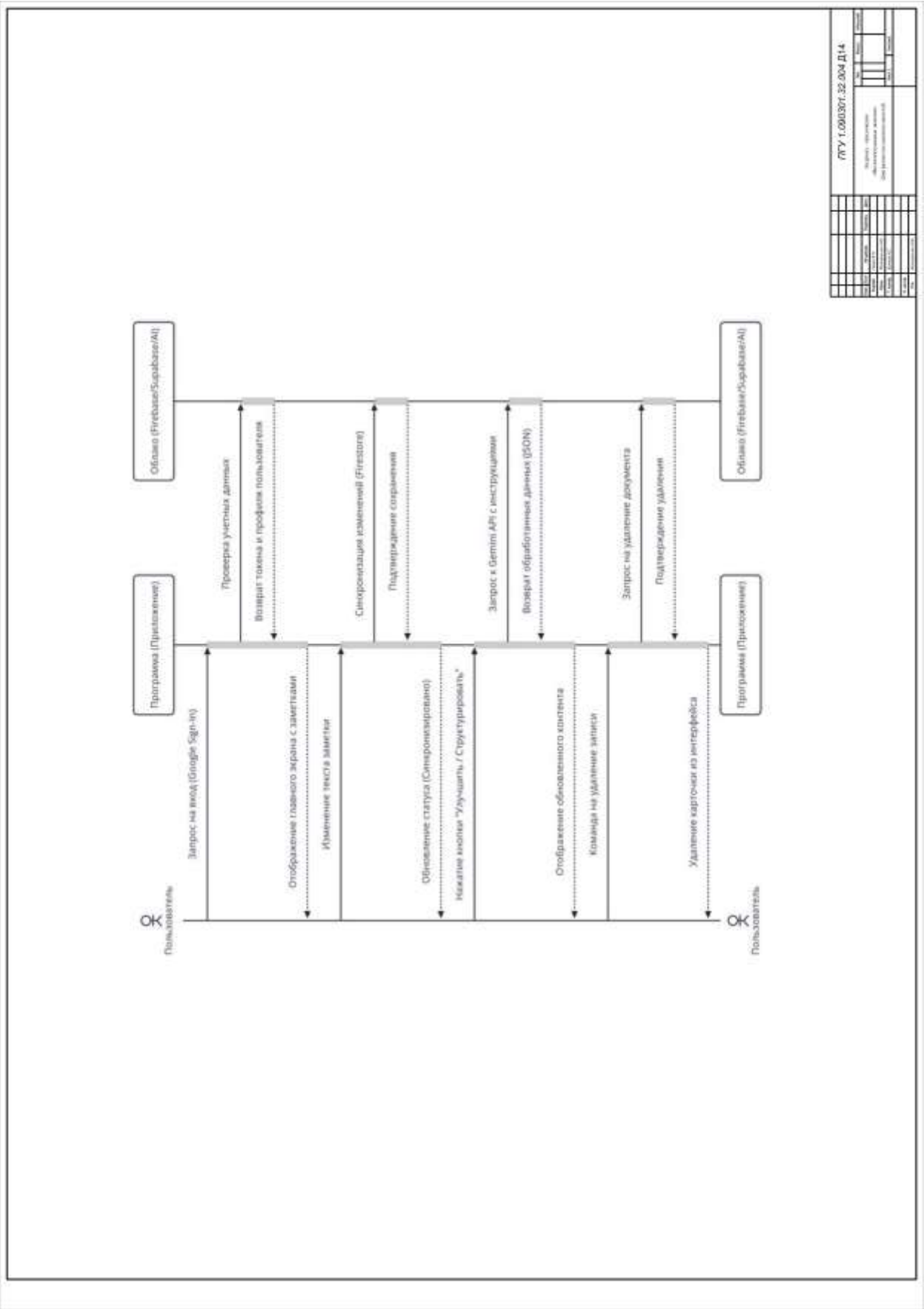
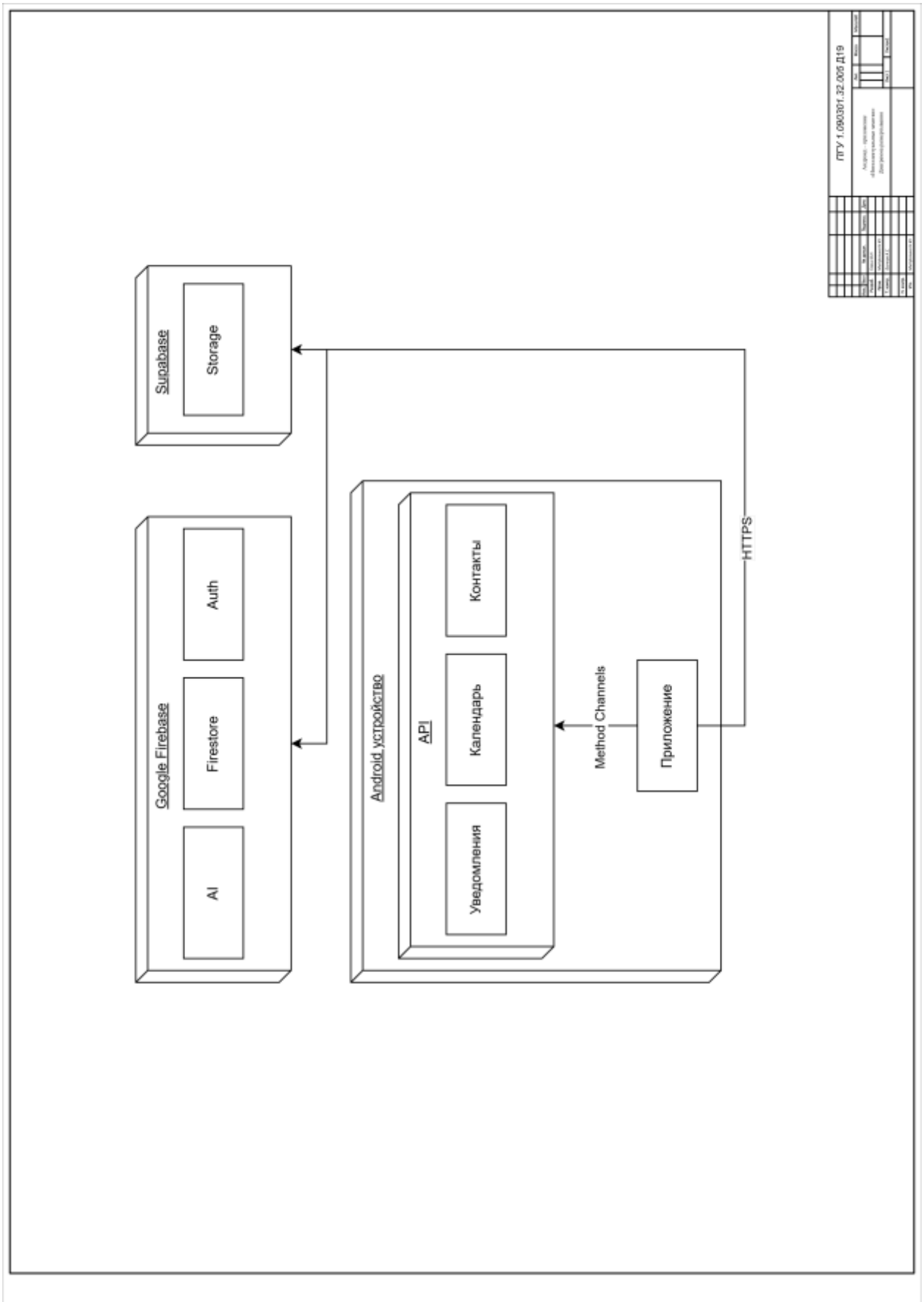


Диаграмма развертывания



Приложение Б

(обязательное)

Андроид - приложение «Интеллектуальные заметки»

Листинг программы

Файл «note.dart»

```
enum NoteRepeatMode {
  none,
  daily,
  weekly,
  monthly,
  yearly;

  String get label => switch (this) {
    none => 'Без повтора',
    daily => 'Ежедневно',
    weekly => 'Еженедельно',
    monthly => 'Ежемесячно',
    yearly => 'Ежегодно',
  };
}

class NoteContact {
  final String name;
  final String phoneNumber;

  const NoteContact({
    required this.name,
    required this.phoneNumber,
  });

  Map<String, dynamic> toJson() => {
    'name': name,
    'phoneNumber': phoneNumber,
  };

  factory NoteContact.fromJson(Map<String, dynamic> json) {
    // Handling potential variations from AI responses
    final name = (json['name'] ?? json['displayName'] ?? 'Без
имени').toString();
    final phone = (json['phoneNumber'] ?? json['phone_number']
?? json['phone'] ?? '').toString();

    return NoteContact(
      name: name,
      phoneNumber: phone,
    );
  }

  @override
  bool operator ==(Object other) =>
      identical(this, other) ||
      other is NoteContact &&
          runtimeType == other.runtimeType &&
          name == other.name &&
```

```
          phoneNumber == other.phoneNumber;

  @override
  int get hashCode => name.hashCode ^ phoneNumber.hashCode;
}

class Note {
  /// Считается синхронизированной, если isNoteSynced == true
  final bool isNoteSynced;
  bool get isSynced => isNoteSynced;
  final String id;
  final String title;
  final String content;
  final List<String> tags;
  final List<String> imagePaths;
  final DateTime createdAt;
  final DateTime updatedAt;
  final bool isPinned;
  final int? colorIndex;
  final DateTime? eventAt;
  final int? reminderMinutes;
  final String? calendarEventId;
  final String? calendarId;
  final bool isCompleted;
  final bool isDeleted;
  final NoteRepeatMode repeatMode;
  final String? originalContent;
  final List<NoteContact> contacts;

  int get notificationId => id.hashCode.abs();
  bool get isAiProcessed => tags.contains('AI');

  const Note({
    required this.id,
    required this.title,
    required this.content,
    required this.tags,
    this.imagePaths = const [],
    required this.createdAt,
    required this.updatedAt,
    this.isPinned = false,
    this.colorIndex,
    this.isNoteSynced = false,
    this.eventAt,
    this.reminderMinutes,
    this.calendarEventId,
    this.calendarId,
    this.isCompleted = false,
    this.isDeleted = false,
    this.repeatMode = NoteRepeatMode.none,
    this.originalContent,
```

```
    this.contacts = const [],
  });

  Note copyWith({
    String? title,
    String? content,
    List<String>? tags,
    List<String>? imagePaths,
    DateTime? updatedAt,
    bool? isPinned,
    int? colorIndex,
    bool? isNoteSynced,
    DateTime? eventAt,
    int? reminderMinutes,
    String? calendarEventId,
    String? calendarId,
    bool? isCompleted,
    bool? isDeleted,
    NoteRepeatMode? repeatMode,
    String? originalContent,
    List<NoteContact>? contacts,
    bool clearColor = false,
    bool clearEvent = false,
  }) => Note(
    id: id,
    title: title ?? this.title,
    content: content ?? this.content,
    tags: tags ?? this.tags,
    imagePaths: imagePaths ?? this.imagePaths,
    createdAt: createdAt,
    updatedAt: updatedAt ?? this.updatedAt,
    isPinned: isPinned ?? this.isPinned,
    colorIndex: clearColor ? null : (colorIndex ??
this.colorIndex),
    isNoteSynced: isNoteSynced ?? this.isNoteSynced,
    eventAt: clearEvent ? null : (eventAt ?? this.eventAt),
    reminderMinutes: clearEvent
        ? null
        : (reminderMinutes ?? this.reminderMinutes),
    calendarEventId: clearEvent
        ? null
        : (calendarEventId ?? this.calendarEventId),
    calendarId: clearEvent ? null : (calendarId ??
this.calendarId),
    isCompleted: isCompleted ?? this.isCompleted,
    isDeleted: isDeleted ?? this.isDeleted,
    repeatMode: repeatMode ?? this.repeatMode,
    originalContent: originalContent ?? this.originalContent,
    contacts: contacts ?? this.contacts,
  );
```

```
  Map<String, dynamic> toJson() => {
    'id': id,
    'title': title,
    'content': content,
    'tags': tags,
    'imagePaths': imagePaths,
    'createdAt': createdAt.toIso8601String(),
    'updatedAt': updatedAt.toIso8601String(),
    'isPinned': isPinned,
    'colorIndex': colorIndex,
    'isNoteSynced': isNoteSynced,
    'eventAt': eventAt?.toIso8601String(),
    'reminderMinutes': reminderMinutes,
    'calendarEventId': calendarEventId,
    'calendarId': calendarId,
    'isCompleted': isCompleted,
    'isDeleted': isDeleted,
    'repeatMode': repeatMode.name,
    'originalContent': originalContent,
    'contacts': contacts.map((e) => e.toJson()).toList(),
  };

  factory Note.fromJson(Map<String, dynamic> json) => Note(
    id: json['id'] as String,
    title: json['title'] as String? ?? '',
    content: json['content'] as String,
    tags:
        (json['tags'] as List<dynamic>?)?.map((e) => e as
String).toList() ??
        [],
    imagePaths:
        (json['imagePaths'] as List<dynamic>?)
            ?.map((e) => e as String)
            .toList() ??
        [],
    createdAt: DateTime.parse(json['createdAt'] as String),
    updatedAt: DateTime.parse(json['updatedAt'] as String),
    isPinned: json['isPinned'] as bool? ?? false,
    colorIndex: json['colorIndex'] as int?,
    isNoteSynced: (json['isNoteSynced'] is bool)
        ? json['isNoteSynced'] as bool
        : false,
    eventAt: json['eventAt'] != null
        ? DateTime.tryParse(json['eventAt'] as String)
        : null,
    reminderMinutes: json['reminderMinutes'] as int?,
    calendarEventId: json['calendarEventId'] as String?,
    calendarId: json['calendarId'] as String?,
    isCompleted: json['isCompleted'] as bool? ?? false,
    isDeleted: json['isDeleted'] as bool? ?? false,
    repeatMode: NoteRepeatMode.values.firstWhere(
```

```
      (e) => e.name == (json['repeatMode'] as String?),
      orElse: () => NoteRepeatMode.none,
    ),
    originalContent: json['originalContent'] as String?,
    contacts:
        (json['contacts'] as List<dynamic>?)
            ?.map((e) => NoteContact.fromJson(e as Map<String,
dynamic>))
            .toList() ??
        [],
  );
}
```

Файл «structured_note_data.dart»

```
import 'note.dart';

class StructuredNoteData {
  final String title;
  final String content;
  final List<String> tags;
  final int? colorIndex;
  final DateTime? eventAt;
  final int? reminderMinutes;
  final List<NoteContact> contacts;

  StructuredNoteData({
    required this.title,
    required this.content,
    required this.tags,
    this.colorIndex,
    this.eventAt,
    this.reminderMinutes,
    this.contacts = const [],
  });

  factory StructuredNoteData.fromJson(Map<String, dynamic> json)
{
    return StructuredNoteData(
      title: json['title'] as String? ?? 'Новая заметка',
      content: json['content'] as String? ?? '',
      tags: (json['tags'] as List<dynamic>?)?.map((e) =>
e.toString()).toList() ?? [],
      colorIndex: json['colorIndex'] as int?,
      eventAt: json['eventAt'] != null ?
DateTime.tryParse(json['eventAt'] as String) : null,
      reminderMinutes: json['reminderMinutes'] as int?,
      contacts: (json['contacts'] as List<dynamic>?)
              ?.map((e) => NoteContact.fromJson(e as Map<String,
dynamic>))
              .toList() ??
          [],
```

```
    );
  }
}
```

Файл «gemini_service.dart»

```
import 'dart:convert';
import 'dart:developer' as dev;

import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:firebase_ai/firebase_ai.dart';
import 'package:google_generative_ai/google_generative_ai.dart'
as google_ai;

import '../../data/models/note.dart';
import '../../data/models/structured_note_data.dart';
import '../services/settings_service.dart';

final geminiServiceProvider = Provider<GeminiService>((ref) {
  return GeminiService(ref);
});

class GeminiService {
  final Ref _ref;
  late final GenerativeModel _primaryModel;
  final String _modelName = 'gemini-2.5-flash-lite';

  GeminiService(this._ref) {
    _primaryModel = FirebaseAI.googleAI().generativeModel(
      model: _modelName,
      generationConfig: GenerationConfig(responseMimeType:
'application/json'),
    );
  }

  Future<String?> _generateWithRetry(String prompt) async {
    final usePersonalKey = _ref.read(useFallbackApiKeyProvider);
    final personalKey = _ref.read(fallbackApiKeyProvider);

    if (usePersonalKey && personalKey != null &&
personalKey.isNotEmpty) {
      dev.log('GeminiService: Использование персонального ключа
(AI Studio)');
      return _generateViaAIStudio(prompt, personalKey);
    } else {
      dev.log('GeminiService: Использование стандартного сервиса
(Firebase)');
      return _generateViaFirebase(prompt);
    }
  }

  Future<String?> _generateViaFirebase(String prompt) async {
```

```
    try {
      final response = await _primaryModel.generateContent([
        Content.text(prompt),
      ]);
      return response.text;
    } catch (e) {
      dev.log('GeminiService: Ошибка Firebase: $e');
      rethrow;
    }
  }

  Future<String?> _generateViaAIStudio(String prompt, String
apiKey) async {
    try {
      final fallbackModel = google_ai.GenerativeModel(
        model: _modelName,
        apiKey: apiKey,
        generationConfig: google_ai.GenerationConfig(
          responseMimeType: 'application/json',
        ),
      );

      final response = await fallbackModel.generateContent([
        google_ai.Content.text(prompt),
      ]);
      return response.text;
    } catch (e) {
      dev.log('GeminiService: Ошибка AI Studio: $e');
      rethrow;
    }
  }

  Future<String?> _processText(String content, String
instruction) async {
    try {
      if (content.trim().isEmpty) return null;

      final prompt =
          '''
$instruction

Верни ответ СТРОГО в формате JSON:
{
  "result": "текст обработанной заметки"
}

Текст для обработки:
$content
''';

      final text = await _generateWithRetry(prompt);
```

```
      if (text == null || text.isEmpty) return null;

      final jsonStr = text
          .replaceAll('```json', '')
          .replaceAll('```', '')
          .trim();
      final Map<String, dynamic> jsonMap = jsonDecode(jsonStr);
      return jsonMap['result'] as String?;
    } catch (e) {
      throw Exception('Ошибка Gemini API: $e');
    }
  }

  Future<String?> improveText(String content) async {
    return _processText(
      content,
      'Ты — профессиональный редактор. Улучши стиль текста,
сделай его более ясным, профессиональным и приятным для чтения.
Сохрани оригинальный смысл и детали. Исправь опечатки.',
    );
  }

  Future<String?> checkGrammar(String content) async {
    return _processText(
      content,
      'Проверь текст на наличие грамматических, пунктуационных и
орфографических ошибок. Исправь их, максимально сохраняя
авторский стиль и структуру текста.',
    );
  }

  Future<String?> summarize(String content) async {
    return _processText(
      content,
      'Сократи текст до самых главных мыслей. Создай краткую
выжимку (саммари), которая передает суть, но в разы короче
оригинала. Используй буллиты, если это уместно.',
    );
  }

  Future<StructuredNoteData?> structureNote(
    String rawContent, {
    DateTime? noteDate,
    List<NoteContact> userContacts = const [],
    List<String> existingFolders = const [],
  }) async {
    try {
      if (rawContent.trim().isEmpty) return null;

      final referenceDate = noteDate ?? DateTime.now();
```

```
      String contactsContext = '';
      if (userContacts.isNotEmpty) {
        final simplified = userContacts
            .map((c) => {'name': c.name, 'phone':
c.phoneNumber})
            .toList();
        contactsContext =
            '\nСПРАВОЧНИК КОНТАКТОВ ПОЛЬЗОВАТЕЛЯ (используй для
сопоставления):\n${jsonEncode(simplified)}\n';
      }

      final prompt =
          '''
Дата создания заметки (точка отсчета):
${referenceDate.toIso8601String()}.
Строго используй ЭТУ дату как "сегодня" для высчитывания всех
относительных дат
(например: "завтра", "послезавтра", "в следующую пятницу",
"через неделю", "утром").

Ты — помощник по автоматическому структурированию заметок. Твоя
задача — принимать
разрозненный или неструктурированный текст заметки и превращать
его в аккуратную
структуру. Пожалуйста, придумай короткий подходящий title.
Улучши и очисти content
(исправь опечатки и грамматику, но сохрани суть и подробности).
$contactsContext
ПОДБОР ЦВЕТА И КАТЕГОРИИ:
ОБЯЗАТЕЛЬНО выбери осмысленный colorIndex и соответствующую ему
КОРНЕВУЮ ПАПКУ для тега:
0 (желтый) -> "Идеи и заметки"
1 (красный) -> "Срочное и важное"
2 (зеленый) -> "Продукты и здоровье"
3 (синий) -> "Работа и учеба"
4 (фиолетовый) -> "Хобби и отдых"
5 (оранжевый) -> "Встречи и события"
6 (голубой) -> "Финансы и бизнес"
7 (розовый) -> "Личное и семья"
null (белый) -> "Разное"

ИНСТРУКЦИЯ ПО ТЕГАМ:
1. Каждая заметка должна иметь СТРОГО ОДИН тег.
2. Тег должен ОБЯЗАТЕЛЬНО начинаться с названия КОРНЕВОЙ ПАПКИ,
соответствующей выбранному colorIndex.
3. Если нужно уточнить категорию, используй вложенность через
"/": например, "Работа и учеба/Проект А/Дизайн".
4. Используй существующие подпапки из списка ниже, если они
подходят.

СУЩЕСТВУЮЩИЕ ПАПКИ/ТЕГИ ПОЛЬЗОВАТЕЛЯ:
```

```
${existingFolders.isNotEmpty ? existingFolders.join(', ') :
'Список пуст'}

Строго возвращай ТОЛЬКО JSON объект (без markdown) следующей
схемы:
{
  "title": "String",
  "content": "String",
  "tags": ["String"],
  "colorIndex": 4,
  "eventAt": "2024-05-20T18:00:00.000Z", // null if no event
  "reminderMinutes": 30, // null if no event
  "contacts": [
    {
      "name": "Имя",
      "phoneNumber": "+79991234567"
    }
  ]
}

Извлеки контакты (имена и телефоны). Твоя КРИТИЧЕСКАЯ задача —
сопоставить упомянутых людей с справочником.
1. СНАЧАЛА проверь, есть ли упомянутое имя в СПРАВОЧНИКЕ
КОНТАКТОВ ПОЛЬЗОВАТЕЛЯ. Если есть, ОБЯЗАТЕЛЬНО используй данные
оттуда.
2. НЕ выдумывай новые контакты для обычных имен, если в тексте
нет явного указания на создание контакта (например, "запиши
номер", "новый контакт", "вот его телефон") или если рядом нет
номера телефона.
3. Если номер телефона есть в тексте, привяжи его к контакту.
4. Если номера телефона НЕТ и человека нет в справочнике, НЕ
добавляй его в список contacts, даже если это имя. Мы добавляем
только реальные контакты для связи.

Проанализируй следующий текст заметки:
$rawContent
''';

      final text = await _generateWithRetry(prompt);
      if (text == null || text.isEmpty) {
        throw Exception('Gemini вернул пустой ответ.');
      }

      final jsonStr = text
          .replaceAll('```json', '')
          .replaceAll('```', '')
          .trim();
      dev.log('GeminiService: Raw AI JSON response: $jsonStr');
      final Map<String, dynamic> jsonMap = jsonDecode(jsonStr);
      return StructuredNoteData.fromJson(jsonMap);
    } catch (e) {
```

```
      throw Exception('Ошибка Gemini API: $e');
    }
  }

  Future<Map<String, StructuredNoteData>> structureNotesBatch(
    List<Note> notes, {
    List<NoteContact> userContacts = const [],
    List<String> existingFolders = const [],
  }) async {
    try {
      if (notes.isEmpty) return {};

      final now = DateTime.now();
      String contactsContext = '';
      if (userContacts.isNotEmpty) {
        final simplified = userContacts
            .map((c) => {'name': c.name, 'phone':
c.phoneNumber})
            .toList();
        contactsContext =
            '\nСПРАВОЧНИК КОНТАКТОВ
ПОЛЬЗОВАТЕЛЯ:\n${jsonEncode(simplified)}\n';
      }

      final notesList = notes.map((n) => {
        'id': n.id,
        'content': n.content,
        'createdAt': n.createdAt.toIso8601String(),
      }).toList();

      final prompt = '''
Ты — помощник по автоматическому структурированию заметок. Твоя
задача — обработать МАССИВ заметок.
Для КАЖДОЙ заметки используй её поле "createdAt" как точку
отсчета (сегодня) для высчитывания всех относительных дат в
тексте этой заметки.
(например: если createdAt = 2024-05-10, а в тексте "завтра", то
дата события должна быть 2024-05-11).

Для каждой заметки из списка:
1. Придумай короткий подходящий title.
2. Улучши и очисти content (исправь опечатки и грамматику, но
сохрани суть и подробности).
$contactsContext
ПОДБОР ЦВЕТА И КАТЕГОРИИ (для каждой заметки):
ОБЯЗАТЕЛЬНО выбери осмысленный colorIndex и соответствующую ему
КОРНЕВУЮ ПАПКУ для тега:
0 (желтый) -> "Идеи и заметки"
1 (красный) -> "Срочное и важное"
2 (зеленый) -> "Продукты и здоровье"
3 (синий) -> "Работа и учеба"
```

```
4 (фиолетовый) -> "Хобби и отдых"
5 (оранжевый) -> "Встречи и события"
6 (голубой) -> "Финансы и бизнес"
7 (розовый) -> "Личное и семья"
null (белый) -> "Разное"

ИНСТРУКЦИЯ ПО ТЕГАМ:
1. Каждая заметка должна иметь СТРОГО ОДИН тег.
2. Тег должен ОБЯЗАТЕЛЬНО начинаться с названия КОРНЕВОЙ ПАПКИ,
соответствующей выбранному colorIndex.
3. Если нужно уточнить категорию, используй вложенность через
"/": например, "Работа и учеба/Проект А/Дизайн".
4. Используй существующие подпапки из списка ниже, если они
подходят.

СУЩЕСТВУЮЩИЕ ПАПКИ/ТЕГИ ПОЛЬЗОВАТЕЛЯ:
${existingFolders.isNotEmpty ? existingFolders.join(', ') :
'Список пуст'}

ВХОДНЫЕ ЗАМЕТКИ ДЛЯ ОБРАБОТКИ (JSON):
${jsonEncode(notesList)}

Строго возвращай ТОЛЬКО JSON объект (без markdown), где ключи —
это "id" заметки:
{
  "note_id_1": {
    "title": "String",
    "content": "String",
    "tags": ["String"],
    "colorIndex": 4,
    "eventAt": "ISO datetime or null",
    "reminderMinutes": 30 or null,
    "contacts": [{"name": "Имя", "phoneNumber": "Телефон"}]
  },
  ...
}
''';

      final text = await _generateWithRetry(prompt);
      if (text == null || text.isEmpty) throw Exception('Gemini
вернул пустой ответ.');

      final jsonStr = text.replaceAll('```json',
'').replaceAll('```', '').trim();
      final Map<String, dynamic> resultsMap =
jsonDecode(jsonStr);

      final Map<String, StructuredNoteData> finalResults = {};
      resultsMap.forEach((id, data) {
        finalResults[id] = StructuredNoteData.fromJson(data as
Map<String, dynamic>);
```

```
      });

      return finalResults;
    } catch (e) {
      dev.log('GeminiService: Batch error: $e');
      throw Exception('Ошибка пакетной обработки Gemini: $e');
    }
  }
}
```

Файл «ai_batch_provider.dart»

```
import 'dart:async';
import 'dart:developer' as dev;
import 'package:flutter_riverpod/flutter_riverpod.dart';
import '../../../core/services/gemini_service.dart';
import '../../notes/providers/notes_provider.dart';
import '../../notes/providers/contacts_provider.dart';
import '../../notes/providers/notes_filters_provider.dart';

enum AiBatchStatus { idle, processing, completed, error }

class AiBatchState {
  final AiBatchStatus status;
  final int total;
  final int processed;
  final int overallTotal;
  final int overallProcessed;
  final String? error;

  AiBatchState({
    this.status = AiBatchStatus.idle,
    this.total = 0,
    this.processed = 0,
    this.overallTotal = 0,
    this.overallProcessed = 0,
    this.error,
  });

  AiBatchState copyWith({
    AiBatchStatus? status,
    int? total,
    int? processed,
    int? overallTotal,
    int? overallProcessed,
    String? error,
  }) {
    return AiBatchState(
      status: status ?? this.status,
      total: total ?? this.total,
      processed: processed ?? this.processed,
      overallTotal: overallTotal ?? this.overallTotal,
```

```
      overallProcessed: overallProcessed ??
this.overallProcessed,
      error: error,
    );
  }
}

final aiBatchProvider = StateNotifierProvider<AiBatchNotifier,
AiBatchState>((ref) {
  final notifier = AiBatchNotifier(ref);
  // Обновляем статистику при изменении списка заметок
  ref.listen(notesProvider, (prev, next) {
    notifier.refreshOverallStats();
  }, fireImmediately: true);
  return notifier;
});

class AiBatchNotifier extends StateNotifier<AiBatchState> {
  final Ref _ref;
  bool _isStopped = false;

  AiBatchNotifier(this._ref) : super(AiBatchState()) {
    // Начальный расчет статистики
    Future.microtask(() => refreshOverallStats());
  }

  void refreshOverallStats() {
    final allNotes = _ref.read(notesProvider).value ?? [];
    final activeNotes = allNotes.where((n) => !n.isDeleted &&
n.content.trim().isNotEmpty).toList();
    final processedNotes = activeNotes.where((n) =>
n.tags.contains('AI')).length;

    state = state.copyWith(
      overallTotal: activeNotes.length,
      overallProcessed: processedNotes,
    );
  }

  void stop() {
    _isStopped = true;
    state = state.copyWith(status: AiBatchStatus.idle);
  }

  void reset() {
    _isStopped = false;
    state = AiBatchState();
  }

  Future<void> startProcessing() async {
    if (state.status == AiBatchStatus.processing) return;
```

```
    _isStopped = false;
    final allNotes = _ref.read(notesProvider).value ?? [];

    // Выбираем заметки без тега "AI" и не удаленные
    final notesToProcess = allNotes
        .where((n) => !n.isDeleted && !n.tags.contains('AI') &&
n.content.trim().isNotEmpty)
        .toList();

    if (notesToProcess.isEmpty) {
      state = state.copyWith(status: AiBatchStatus.completed,
total: 0, processed: 0);
      return;
    }

    state = state.copyWith(
      status: AiBatchStatus.processing,
      total: notesToProcess.length,
      processed: 0,
    );

    final gemini = _ref.read(geminiServiceProvider);

    // Гарантируем загрузку контактов (ждем завершения фьючерса)
    final userContacts = await
_ref.read(allSystemContactsProvider.future);

    const int batchSize = 5;
    for (int i = 0; i < notesToProcess.length; i += batchSize) {
      if (_isStopped) break;

      final chunk = notesToProcess.sublist(
        i,
        i + batchSize > notesToProcess.length ?
notesToProcess.length : i + batchSize,
      );

      try {
        final existingFolders = _ref.read(allTagsProvider);
        final results = await gemini.structureNotesBatch(
          chunk,
          userContacts: userContacts,
          existingFolders: existingFolders,
        );

        for (final note in chunk) {
          final structured = results[note.id];
          if (structured != null) {
            // Исправлено: берем теги, созданные нейросетью, и
добавляем наш системный тег 'AI'
```

```
            final updatedTags = {...structured.tags,
'AI'}.toList();

            await _ref.read(notesProvider.notifier).editNote(
              note.id,
              title: structured.title,
              content: structured.content,
              tags: updatedTags,
              imagePaths: note.imagePaths,
              eventAt: structured.eventAt,
              reminderMinutes: structured.reminderMinutes,
              contacts: structured.contacts,
              originalContent: note.content,
              colorIndex: structured.colorIndex,
            );
          }
        }

        state = state.copyWith(processed: state.processed +
chunk.length);
      } catch (e) {
        dev.log('AiBatchNotifier: Error processing chunk: $e');
        // Продолжаем дальше или останавливаемся? Для пакетной
обработки лучше попробовать пропустить ошибку
        // но если ошибка фатальная (например, API лимит), то
лучше прервать.
        if (e.toString().contains('429')) {
           state = state.copyWith(status: AiBatchStatus.error,
error: 'Превышен лимит запросов (Rate Limit)');
           return;
        }
      }
    }

    if (!_isStopped) {
      state = state.copyWith(status: AiBatchStatus.completed);
    }
  }
}
```

Файл «notes_provider.dart»

```
import 'package:flutter/foundation.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:uuid/uuid.dart';
import '../../../data/models/note.dart';
import '../../../data/repositories/notes_repository.dart';
import '../../../data/repositories/firestore_repository.dart';
import '../../../core/services/auth_service.dart';
import '../../../core/services/calendar_service.dart';
import '../../../core/services/sync_service.dart';
import '../../../core/services/notification_service.dart';
```

```
final notesRepositoryProvider = Provider((_) =>
NotesRepository());
final selectedIdsProvider = StateProvider<Set<String>>((ref) =>
{});
final notesProvider = AsyncNotifierProvider<NotesNotifier,
List<Note>>(
  NotesNotifier.new,
);

class NotesNotifier extends AsyncNotifier<List<Note>> {
  static const _uuid = Uuid();
  NotesRepository get _localRepo =>
ref.read(notesRepositoryProvider);

  FirestoreRepository? get _remoteRepo {
    final user = ref.watch(authStateProvider).value;
    return user != null ? FirestoreRepository(user.uid) : null;
  }

  @override
  Future<List<Note>> build() async {
    final user = ref.watch(authStateProvider).value;
    final localNotes = await _localRepo.loadAll();

    if (user != null) {
      // Запускаем синхронизацию в фоне, не блокируя UI
      _backgroundSync(user.uid, localNotes);
    }

    return localNotes;
  }

  // Публичный метод для ручного обновления (pull-to-refresh)
  Future<void> refreshSync() async {
    final user = ref.read(authStateProvider).value;
    if (user == null) return;
    await _backgroundSync(user.uid, state.value ?? []);
  }

  Future<void> _backgroundSync(String uid, List<Note>
initialNotes) async {
    try {
      final remoteRepo = FirestoreRepository(uid);
      final cloudNotes = await remoteRepo.fetchNotes();
      final cloudIds = cloudNotes.map((n) => n.id).toSet();

      // Используем актуальное состояние, если оно уже
изменилось пока мы шли в облако
      final currentLocal = state.value ?? initialNotes;
```

```
      // 1. Заметки, помеченные как синхронизированные, но
отсутствующие в облаке.
      // Это означает, что они были удалены на другом
устройстве.

      // 2. Новые локальные заметки, которых еще нет в облаке
      final notesToUpload =
          currentLocal.where((n) => !n.isNoteSynced &&
!cloudIds.contains(n.id));

      // 3. Облачные заметки помечаем как синхронизированные
      final syncedCloudNotes =
          cloudNotes.map((n) => n.copyWith(isNoteSynced:
true)).toList();

      // 4. Формируем финальный список: все из облака + новые
локальные
      final finalNotes = [...syncedCloudNotes,
...notesToUpload];

      // Обновляем состояние и локальное хранилище
      state = AsyncData(finalNotes);
      await _localRepo.saveAll(finalNotes);

      // 5. Запускаем загрузку новых локальных заметок в облако
      for (final note in notesToUpload) {
        _performFullCloudSync(uid, note);
      }
    } catch (e) {
      debugPrint('Ошибка фоновой синхронизации: $e');
    }
  }

  // Общий метод для сохранения и запуска фоновой синхронизации
  Future<void> _persist(
    List<Note> notes, {
    Note? noteToSync,
    String? deletedId,
    Set<String>? deletedIds,
  }) async {
    state = AsyncData(notes);

    // Оптимизация: используем Hive точечно вместо полной
перезаписи всего списка
    if (noteToSync != null) {
      await _localRepo.putNote(noteToSync);
    } else if (deletedId != null) {
      await _localRepo.deleteNote(deletedId);
    } else if (deletedIds != null) {
      await _localRepo.deleteNotes(deletedIds);
    } else {
```

```
      await _localRepo.saveAll(notes);
    }

    final user = ref.read(authStateProvider).value;
    if (user != null) {
      final remoteFirestore = FirestoreRepository(user.uid);
      if (deletedId != null) await
remoteFirestore.deleteNote(deletedId);
      if (deletedIds != null) await
remoteFirestore.deleteMultiple(deletedIds);

      if (noteToSync != null) {
        // Запускаем фоновую синхронизацию (Supabase +
Firestore)
        _performFullCloudSync(user.uid, noteToSync);
      }
    }
  }

  // Фоновая магия: загружаем фото в Supabase, потом обновляем
ссылки в Firestore
  Future<void> _performFullCloudSync(String uid, Note note)
async {
    try {
      final syncService = SyncService(uid);
      final remoteFirestore = FirestoreRepository(uid);

      // 1. Загружаем только локальные картинки (не ссылки) в
Supabase и получаем ссылки
      final localImages = note.imagePaths
          .where((img) => !img.startsWith('http'))
          .toList();
      final remoteImages = note.imagePaths
          .where((img) => img.startsWith('http'))
          .toList();
      final uploadedUrls = await
syncService.uploadImages(localImages);
      // Собираем итоговый список: уже существующие ссылки +
новые загруженные
      // Если картинка не загрузилась (пустая строка), оставляем
локальный путь
      final filteredUrls = [
        ...remoteImages,
        for (int i = 0; i < uploadedUrls.length; i++)
          uploadedUrls[i].isNotEmpty ? uploadedUrls[i] :
localImages[i],
      ];
      final syncedNote = note.copyWith(
        imagePaths: filteredUrls,
        isNoteSynced: true,
      );
```

```
      // 2. Обновляем заметку в Firestore (теперь там ссылки
https://...)
      await remoteFirestore.upsertNote(syncedNote);

      // 3. Обновляем локальное состояние, чтобы UI использовал
облачные ссылки (для кеша)
      final currentNotes = state.value ?? <Note>[];
      final newList = <Note>[
        for (final n in currentNotes)
          if (n.id == syncedNote.id) syncedNote else n,
      ];
      state = AsyncData(newList);
      await _localRepo.putNote(syncedNote);
    } catch (e) {
      debugPrint('Ошибка фоновой синхронизации: $e');
    }
  }

  Future<Note> add({
    required String title,
    required String content,
    required List<String> tags,
    List<String> imagePaths = const [],
    int? colorIndex,
    bool isPinned = false,
    DateTime? eventAt,
    int? reminderMinutes,
    String? calendarEventId,
    String? calendarId,
    NoteRepeatMode repeatMode = NoteRepeatMode.none,
    String? originalContent,
    List<NoteContact> contacts = const [],
  }) async {
    final note = Note(
      id: _uuid.v4(),
      title: title,
      content: content,
      tags: tags,
      imagePaths: imagePaths,
      createdAt: DateTime.now(),
      updatedAt: DateTime.now(),
      colorIndex: colorIndex,
      isPinned: isPinned,
      isNoteSynced: false,
      eventAt: eventAt,
      reminderMinutes: reminderMinutes,
      calendarEventId: calendarEventId,
      calendarId: calendarId,
      repeatMode: repeatMode,
      originalContent: originalContent,
```

```
      contacts: contacts,
    );
    final notes = <Note>[...(state.value ?? <Note>[]), note];
    await _persist(notes, noteToSync: note);
    // Планируем уведомление
    await
ref.read(notificationServiceProvider).scheduleNoteNotification(n
ote);
    return note;
  }

  Future<Note?> editNote(
    String id, {
    required String title,
    required String content,
    required List<String> tags,
    required List<String> imagePaths,
    DateTime? eventAt,
    int? reminderMinutes,
    String? calendarEventId,
    String? calendarId,
    NoteRepeatMode? repeatMode,
    bool clearEvent = false,
    String? originalContent,
    List<NoteContact>? contacts,
    int? colorIndex,
  }) async {
    Note? updatedNote;
    final currentNotes = state.value ?? <Note>[];
    final oldNote = currentNotes.where((n) => n.id ==
id).isNotEmpty
        ? currentNotes.firstWhere((n) => n.id == id)
        : null;
    // Если были удалены картинки, удаляем их из Supabase
    if (oldNote != null) {
      final removedImages = oldNote.imagePaths
          .where((img) => !imagePaths.contains(img))
          .toList();
      final user = ref.read(authStateProvider).value;
      if (removedImages.isNotEmpty && user != null) {
        final syncService = SyncService(user.uid);
        await syncService.deleteImagesByUrls(removedImages);
      }
    }
    final notes = <Note>[
      for (final n in currentNotes)
        if (n.id == id)
          (() {
            updatedNote = n.copyWith(
              title: title,
              content: content,
```

```
              tags: tags,
              imagePaths: imagePaths,
              updatedAt: DateTime.now(),
              isNoteSynced: false,
              eventAt: eventAt,
              reminderMinutes: reminderMinutes,
              calendarEventId: calendarEventId,
              calendarId: calendarId,
              repeatMode: repeatMode,
              clearEvent: clearEvent,
              originalContent: originalContent,
              contacts: contacts,
              colorIndex: colorIndex ?? n.colorIndex,
            );
            return updatedNote!;
          })()
        else
          n,
    ];
    if (updatedNote != null) {
      await _persist(notes, noteToSync: updatedNote);
      // Обновляем уведомление
      await
ref.read(notificationServiceProvider).scheduleNoteNotification(u
pdatedNote!);
    }
    return updatedNote;
  }

  Future<void> delete(String id) async {
    final currentNotes = state.value ?? <Note>[];
    final noteToDelete = currentNotes.where((n) => n.id ==
id).isNotEmpty
        ? currentNotes.firstWhere((n) => n.id == id)
        : null;

    if (noteToDelete != null) {
      try {
        await
ref.read(calendarServiceProvider).deleteNoteEvent(noteToDelete);
      } catch (_) {}
    }

    Note? updatedNote;
    final notes = <Note>[
      for (final n in currentNotes)
        if (n.id == id)
          (() {
            updatedNote = n.copyWith(isDeleted: true,
isNoteSynced: false);
            return updatedNote!;
```

```
          })()
        else
          n,
    ];

    if (updatedNote != null) {
      await _persist(notes, noteToSync: updatedNote);
      // Отменяем уведомление
      await
ref.read(notificationServiceProvider).cancelNoteNotification(id)
;
    }
  }

  Future<void> deleteMultiple(Set<String> ids) async {
    final currentNotes = state.value ?? <Note>[];
    final updatedNotesList = <Note>[];

    for (final n in currentNotes) {
      if (ids.contains(n.id)) {
        // Запускаем удаление из календаря в фоне без блокировки

ref.read(calendarServiceProvider).deleteNoteEvent(n).catchError(
(_) {});

        final updated = n.copyWith(isDeleted: true,
isNoteSynced: false);
        updatedNotesList.add(updated);
        _remoteRepo?.upsertNote(updated).catchError((_) {});
      } else {
        updatedNotesList.add(n);
      }
    }
    state = AsyncData(updatedNotesList);
    await _localRepo.putNotes(updatedNotesList);

    // Отменяем уведомления для всех выбранных
    for (final id in ids) {
      await
ref.read(notificationServiceProvider).cancelNoteNotification(id)
;
    }
  }

  Future<void> permanentlyDelete(String id) async {
    final currentNotes = state.value ?? <Note>[];
    final noteToDelete = currentNotes.where((n) => n.id ==
id).isNotEmpty
        ? currentNotes.firstWhere((n) => n.id == id)
        : null;
    final user = ref.read(authStateProvider).value;
```

```
    if (noteToDelete != null && user != null) {
      final syncService = SyncService(user.uid);
      await
syncService.deleteImagesByUrls(noteToDelete.imagePaths);
    }
    if (noteToDelete != null) {
      try {
        await
ref.read(calendarServiceProvider).deleteNoteEvent(noteToDelete);
      } catch (_) {}
    }
    final notes = <Note>[
      for (final n in currentNotes)
        if (n.id != id) n,
    ];
    await _persist(notes, deletedId: id);
    // Отменяем уведомление окончательно
    await
ref.read(notificationServiceProvider).cancelNoteNotification(id)
;
  }

  Future<void> permanentlyDeleteMultiple(Set<String> ids) async
{
    final currentNotes = state.value ?? <Note>[];
    final user = ref.read(authStateProvider).value;
    if (user != null) {
      final syncService = SyncService(user.uid);
      final imagesToDelete = <String>[];
      for (final n in currentNotes) {
        if (ids.contains(n.id)) {
          imagesToDelete.addAll(n.imagePaths);
        }
      }
      await syncService.deleteImagesByUrls(imagesToDelete);
    }
    for (final note in currentNotes) {
      if (ids.contains(note.id)) {
        try {
          await
ref.read(calendarServiceProvider).deleteNoteEvent(note);
        } catch (_) {}
      }
    }
    final notes = <Note>[
      for (final n in currentNotes)
        if (!ids.contains(n.id)) n,
    ];
    await _persist(notes, deletedIds: ids);
    // Отменяем все
    for (final id in ids) {
```

```
      await
ref.read(notificationServiceProvider).cancelNoteNotification(id)
;
    }
  }

  Future<void> restore(String id) async {
    final currentNotes = state.value ?? <Note>[];
    Note? updatedNote;
    final notes = <Note>[
      for (final n in currentNotes)
        if (n.id == id)
          (() {
            updatedNote = n.copyWith(isDeleted: false,
isNoteSynced: false);
            return updatedNote!;
          })()
        else
          n,
    ];
    if (updatedNote != null) {
      await _persist(notes, noteToSync: updatedNote);
      // Восстанавливаем уведомление
      await
ref.read(notificationServiceProvider).scheduleNoteNotification(u
pdatedNote!);
    }
  }

  Future<void> emptyTrash() async {
    final currentNotes = state.value ?? <Note>[];
    final idsToDelete = currentNotes
        .where((n) => n.isDeleted)
        .map((n) => n.id)
        .toSet();
    if (idsToDelete.isNotEmpty) {
      await permanentlyDeleteMultiple(idsToDelete);
    }
  }

  Future<void> clearArchive() async {
    final currentNotes = state.value ?? <Note>[];
    final idsToDelete = currentNotes
        .where((n) => n.eventAt != null && n.isCompleted &&
!n.isDeleted)
        .map((n) => n.id)
        .toSet();
    if (idsToDelete.isNotEmpty) {
      await permanentlyDeleteMultiple(idsToDelete);
    }
  }
```

```
  Future<void> toggleCompleted(String id) async {
    Note? updatedNote;
    final notes = <Note>[
      for (final n in state.value ?? <Note>[])
        if (n.id == id)
          (() {
            updatedNote = n.copyWith(
              isCompleted: !n.isCompleted,
              isNoteSynced: false,
            );
            return updatedNote!;
          })()
        else
          n,
    ];
    if (updatedNote != null) {
      state = AsyncData(notes);
      await _localRepo.putNote(updatedNote!);
      final user = ref.read(authStateProvider).value;
      if (user != null) {
        await
FirestoreRepository(user.uid).upsertNote(updatedNote!);
        final syncedNote = updatedNote!.copyWith(isNoteSynced:
true);
        final currentNotes = state.value ?? <Note>[];
        final newList = <Note>[
          for (final n in currentNotes)
            if (n.id == id) syncedNote else n,
        ];
        state = AsyncData(newList);
        await _localRepo.putNote(syncedNote);
      }
      // Обновляем уведомление (отменяем если завершено, иначе
планируем)
      await
ref.read(notificationServiceProvider).scheduleNoteNotification(u
pdatedNote!);
    }
  }

  Future<void> togglePin(String id) async {
    Note? updatedNote;
    final notes = <Note>[
      for (final n in state.value ?? <Note>[])
        if (n.id == id)
          (() {
            updatedNote = n.copyWith(
              isPinned: !n.isPinned,
              isNoteSynced: false,
            );
```

```
            return updatedNote!;
          })()
        else
          n,
    ];
    if (updatedNote != null) {
      state = AsyncData(notes);
      await _localRepo.putNote(updatedNote!);
      final user = ref.read(authStateProvider).value;
      if (user != null) {
        await
FirestoreRepository(user.uid).upsertNote(updatedNote!);
        // После успешного обновления в Firestore выставляем
isNoteSynced=true
        final syncedNote = updatedNote!.copyWith(isNoteSynced:
true);
        final currentNotes = state.value ?? <Note>[];
        final newList = <Note>[
          for (final n in currentNotes)
            if (n.id == id) syncedNote else n,
        ];
        state = AsyncData(newList);
        await _localRepo.putNote(syncedNote);
      }
    }
  }

  Future<void> togglePinMultiple(Set<String> ids) async {
    final currentNotes = state.value ?? <Note>[];
    final anyUnpinned = currentNotes
        .where((n) => ids.contains(n.id))
        .any((n) => !n.isPinned);

    final updatedNotesList = <Note>[];
    for (final n in currentNotes) {
      if (ids.contains(n.id)) {
        final updated = n.copyWith(isPinned: anyUnpinned);
        updatedNotesList.add(updated);
        // Синхронизируем изменение в Firestore
        await _remoteRepo?.upsertNote(updated);
      } else {
        updatedNotesList.add(n);
      }
    }
    state = AsyncData(updatedNotesList);
    await _localRepo.putNotes(updatedNotesList);
  }

  Future<void> setColor(String id, int? colorIndex) async {
    Note? updatedNote;
    final notes = <Note>[
```

```
      for (final n in state.value ?? <Note>[])
        if (n.id == id)
          (() {
            updatedNote = n.copyWith(
              colorIndex: colorIndex,
              clearColor: colorIndex == null,
              isNoteSynced: false,
            );
            return updatedNote!;
          })()
        else
          n,
    ];
    if (updatedNote != null) {
      state = AsyncData(notes);
      await _localRepo.putNote(updatedNote!);
      final user = ref.read(authStateProvider).value;
      if (user != null) {
        await
FirestoreRepository(user.uid).upsertNote(updatedNote!);
        // После успешного обновления в Firestore выставляем
isNoteSynced=true
        final syncedNote = updatedNote!.copyWith(isNoteSynced:
true);
        final currentNotes = state.value ?? <Note>[];
        final newList = <Note>[
          for (final n in currentNotes)
            if (n.id == id) syncedNote else n,
        ];
        state = AsyncData(newList);
        await _localRepo.putNote(syncedNote);
      }
    }
  }

  Future<void> updateCalendarEventMeta(
    String id, {
    required String calendarId,
    required String calendarEventId,
  }) async {
    final notes = <Note>[
      for (final n in state.value ?? <Note>[])
        if (n.id == id)
          n.copyWith(calendarId: calendarId, calendarEventId:
calendarEventId)
        else
          n,
    ];

    state = AsyncData(notes);
    final noteToSave = notes.firstWhere((n) => n.id == id);
```

```
    await _localRepo.putNote(noteToSave);
  }
}
```

Файл «sync_service.dart»

```
import 'dart:io';
import 'package:supabase_flutter/supabase_flutter.dart';
import
'package:flutter_image_compress/flutter_image_compress.dart';
import 'package:path_provider/path_provider.dart';
import 'package:path/path.dart' as p;
import 'package:flutter/foundation.dart';

class SyncService {
    // Удалить файл из Supabase по публичной ссылке или пути
    Future<void> deleteImageByUrl(String url) async {
      try {
        // Извлекаем путь внутри bucket (после /object/notes/)
        final uri = Uri.parse(url);
        final idx = uri.path.indexOf('/notes/');
        if (idx == -1) return;
        final path = uri.path.substring(idx + '/notes/'.length);
        await _supabase.storage.from('notes').remove([path]);
      } catch (e) {
        debugPrint('Ошибка удаления из Supabase: $e');
      }
    }

    // Удалить несколько файлов
    Future<void> deleteImagesByUrls(List<String> urls) async {
      for (final url in urls) {
        if (url.startsWith('http')) {
          await deleteImageByUrl(url);
        }
      }
    }
  final String userId;
  SyncService(this.userId);

  final _supabase = Supabase.instance.client;

  Future<List<String>> uploadImages(List<String> paths) async {
    List<String> resultUrls = [];

    for (String path in paths) {
      // Если это уже ссылка — пропускаем
      if (path.startsWith('http')) {
        resultUrls.add(path);
        continue;
      }
```

```
        File file = File(path);
        if (!await file.exists()) {
          resultUrls.add("");
          continue;
        }

        try {
          // 1. СЖАТИЕ (Оптимизация места)
          final tempDir = await getTemporaryDirectory();
          final targetPath = p.join(
            tempDir.path,
            "${DateTime.now().millisecondsSinceEpoch}.jpg",
          );

          final XFile? compressedFile =
              await FlutterImageCompress.compressAndGetFile(
                file.absolute.path,
                targetPath,
                quality: 70,
                minWidth: 1024,
              );

          if (compressedFile == null) {
            resultUrls.add("");
            continue;
          }

          // 2. ЗАГРУЗКА В SUPABASE
          final fileName =

'$userId/${DateTime.now().millisecondsSinceEpoch}${p.extension(p
ath)}';

          await _supabase.storage
              .from('notes')
              .upload(
                fileName,
                File(compressedFile.path),
                fileOptions: const FileOptions(
                  cacheControl: '3600',
                  upsert: false,
                ),
              );

          // 3. ПОЛУЧЕНИЕ ССЫЛКИ
          final String publicUrl = _supabase.storage
              .from('notes')
              .getPublicUrl(fileName);
          resultUrls.add(publicUrl);

          // Удаляем временный сжатый файл
```

```
        await File(compressedFile.path).delete();
      } catch (e) {
        debugPrint('Ошибка Supabase: $e');
        resultUrls.add(""); // В случае ошибки не добавляем
локальный путь
      }
    }
    return resultUrls;
  }
}
```

Файл «notes_repository.dart»

```
import 'dart:convert';
import 'package:flutter/foundation.dart';
import 'package:hive_flutter/hive_flutter.dart';
import 'package:shared_preferences/shared_preferences.dart';
import '../models/note.dart';

List<Note> _parseNotes(List<String> raw) {
  return raw
      .map((e) => Note.fromJson(jsonDecode(e) as Map<String,
dynamic>))
      .toList();
}

class NotesRepository {
  static const _spKey = 'notes';
  static const _boxName = 'notesBox';

  Box<String> get _box => Hive.box<String>(_boxName);

  Future<List<Note>> loadAll() async {
    final prefs = await SharedPreferences.getInstance();

    // Бесшовная миграция из SharedPreferences в Hive
    if (prefs.containsKey(_spKey)) {
      final oldRaw = prefs.getStringList(_spKey) ?? [];
      final oldNotes = await compute(_parseNotes, oldRaw);

      final Map<String, String> mapToSave = {};
      for (final note in oldNotes) {
        mapToSave[note.id] = jsonEncode(note.toJson());
      }
      await _box.putAll(mapToSave);
      await prefs.remove(_spKey);

      return oldNotes;
    }

    final raw = _box.values.toList();
    if (raw.isEmpty) return [];
```

```
    // Парсинг больших данных в отдельном изоляте
    return compute(_parseNotes, raw);
  }

  Future<void> saveAll(List<Note> notes) async {
    final Map<String, String> mapToSave = {};
    for (final note in notes) {
      mapToSave[note.id] = jsonEncode(note.toJson());
    }
    await _box.clear();
    await _box.putAll(mapToSave);
  }

  Future<void> putNote(Note note) async {
    await _box.put(note.id, jsonEncode(note.toJson()));
  }

  Future<void> putNotes(List<Note> notes) async {
    final Map<String, String> mapToSave = {};
    for (final note in notes) {
      mapToSave[note.id] = jsonEncode(note.toJson());
    }
    await _box.putAll(mapToSave);
  }

  Future<void> deleteNote(String id) async {
    await _box.delete(id);
  }

  Future<void> deleteNotes(Set<String> ids) async {
    await _box.deleteAll(ids);
  }
}
```

Файл «firestore_repository.dart»

```
import 'package:cloud_firestore/cloud_firestore.dart';
import '../models/note.dart';

class FirestoreRepository {
  // Получить одну заметку по id
  Future<Note?> fetchNoteById(String id) async {
    final doc = await _notesRef.doc(id).get();
    if (!doc.exists) return null;
    return Note.fromJson(doc.data()!);
  }

  final String userId;
  FirestoreRepository(this.userId);

  // Путь к коллекции заметок конкретного пользователя
```

```
  CollectionReference<Map<String, dynamic>> get _notesRef =>
FirebaseFirestore
      .instance
      .collection('users')
      .doc(userId)
      .collection('notes');

  // Получить все заметки из облака один раз
  Future<List<Note>> fetchNotes() async {
    final snapshot = await _notesRef.get();
    return snapshot.docs.map((doc) =>
Note.fromJson(doc.data())).toList();
  }

  // Создать или обновить заметку в облаке
  Future<void> upsertNote(Note note) async {
    await _notesRef.doc(note.id).set(note.toJson());
  }

  // Удалить заметку из облака
  Future<void> deleteNote(String id) async {
    await _notesRef.doc(id).delete();
  }

  // Массовое удаление в облаке (Batch)
  Future<void> deleteMultiple(Set<String> ids) async {
    final batch = FirebaseFirestore.instance.batch();
    for (final id in ids) {
      batch.delete(_notesRef.doc(id));
    }
    await batch.commit();
  }
}
```

Файл «calendar_service.dart»

```
import 'package:device_calendar/device_calendar.dart';
import 'package:flutter_timezone/flutter_timezone.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:timezone/timezone.dart' as tz;

import '../../data/models/note.dart';
import '../utils/note_deep_links.dart';

final calendarServiceProvider = Provider((_) =>
CalendarService());

class CalendarSyncResult {
  final String calendarId;
  final String eventId;
```

```
  const CalendarSyncResult({required this.calendarId, required
this.eventId});
}

class CalendarPermissionDeniedException implements Exception {
  @override
  String toString() =>
      'Нет доступа к системному календарю. '
      'На Android откройте: Настройки -> Приложения -> Flutter
Application 2 -> '
      'Разрешения -> Календарь -> Разрешить. '
      'Если разрешение не появляется, переустановите приложение
и попробуйте снова.';
}

class CalendarUnavailableException implements Exception {
  final String message;

  const CalendarUnavailableException(this.message);

  @override
  String toString() => message;
}

class CalendarService {
  CalendarService() : _plugin = DeviceCalendarPlugin();

  final DeviceCalendarPlugin _plugin;

  Future<CalendarSyncResult> upsertNoteEvent(Note note) async {
    final eventAt = note.eventAt;
    if (eventAt == null) {
      throw const CalendarUnavailableException('У заметки не
выбрано время.');
    }

    await _ensurePermissions();
    final calendar = await _resolveCalendar(note.calendarId);
    final location = await _resolveTimeZone();
    final start = tz.TZDateTime.from(eventAt, location);
    final end = start.add(const Duration(minutes: 30));

    final event = Event(
      calendar.id,
      eventId: (note.calendarEventId?.isEmpty ?? true) ? null :
note.calendarEventId,
      title: note.title.trim().isNotEmpty ? note.title.trim() :
'Заметка',
      description: _buildDescription(note),
      start: start,
      end: end,
```

```
      allDay: false,
      status: EventStatus.Confirmed,
      availability: Availability.Busy,
      reminders: [Reminder(minutes: note.reminderMinutes ??
10)],
      url: buildNoteDeepLink(note.id),
    );

    final result = await _plugin.createOrUpdateEvent(event);
    if (result == null || !result.isSuccess || result.data ==
null) {
      final errorMessage = result != null &&
result.errors.isNotEmpty
          ? result.errors.first.errorMessage
          : 'Не удалось создать событие в календаре.';
      throw CalendarUnavailableException(errorMessage);
    }

    return CalendarSyncResult(calendarId: calendar.id!, eventId:
result.data!);
  }

  Future<void> deleteNoteEvent(Note note) async {
    if ((note.calendarId?.isEmpty ?? true) ||
        (note.calendarEventId?.isEmpty ?? true)) {
      return;
    }

    await _ensurePermissions();
    final result = await _plugin.deleteEvent(
      note.calendarId,
      note.calendarEventId,
    );
    if (!result.isSuccess) {
      final errorMessage = result.errors.isNotEmpty
          ? result.errors.first.errorMessage
          : 'Не удалось удалить событие из календаря.';
      throw CalendarUnavailableException(errorMessage);
    }
  }

  Future<void> _ensurePermissions() async {
    final current = await _plugin.hasPermissions();
    final hasAccess = current.data ?? false;
    if (hasAccess) return;

    final requested = await _plugin.requestPermissions();
    if (!(requested.data ?? false)) {
      throw CalendarPermissionDeniedException();
    }
  }
```

```
  Future<Calendar> _resolveCalendar(String? preferredCalendarId)
async {
    final calendarsResult = await _plugin.retrieveCalendars();
    final calendars =
        calendarsResult.data
            ?.where(
              (calendar) =>
                  calendar.isReadOnly != true &&
                  (calendar.id?.isNotEmpty ?? false),
            )
            .toList() ??
        const <Calendar>[];

    if (calendars.isEmpty) {
      throw const CalendarUnavailableException(
        'На устройстве не найден доступный календарь для
записи.',
      );
    }

    for (final calendar in calendars) {
      if (calendar.id == preferredCalendarId) return calendar;
    }

    final defaultCalendar =
calendars.cast<Calendar?>().firstWhere(
      (calendar) => calendar?.isDefault == true,
      orElse: () => null,
    );
    return defaultCalendar ?? calendars.first;
  }

  Future<tz.Location> _resolveTimeZone() async {
    try {
      final timezoneInfo = await
FlutterTimezone.getLocalTimezone();
      return tz.getLocation(timezoneInfo.identifier);
    } catch (_) {
      return tz.local;
    }
  }

  String _buildDescription(Note note) {
    final parts = <String>[];

    if (note.content.trim().isNotEmpty) {
      parts.add(note.content.trim());
    }

    if (note.tags.isNotEmpty) {
```

```
      parts.add('Теги: ${note.tags.join(', ')}');
    }

    return parts.join('\n\n');
  }
}
```

Файл «main.dart»

```
import 'dart:async';

import 'package:app_links/app_links.dart';
import 'package:dynamic_color/dynamic_color.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:firebase_app_check/firebase_app_check.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:hive_flutter/hive_flutter.dart';
import 'package:intl/date_symbol_data_local.dart';
import
'package:receive_sharing_intent/receive_sharing_intent.dart';
import 'package:supabase_flutter/supabase_flutter.dart';

import 'core/services/notification_service.dart';
import 'core/services/settings_service.dart';
import 'core/theme/app_theme.dart';
import 'core/utils/note_deep_links.dart';
import 'data/models/note.dart';
import 'features/notes/providers/notes_filters_provider.dart';
import 'features/notes/providers/notes_provider.dart';
import 'features/notes/screens/note_editor_screen.dart';
import 'firebase_options.dart';
import 'main_screen.dart';

final appNavigatorKey = GlobalKey<NavigatorState>();

void main() async {
  WidgetsFlutterBinding.ensureInitialized();

  SystemChrome.setEnabledSystemUIMode(SystemUiMode.edgeToEdge);
  SystemChrome.setSystemUIOverlayStyle(const
SystemUiOverlayStyle(
    statusBarColor: Colors.transparent,
    systemNavigationBarColor: Colors.transparent,
  ));

  await Hive.initFlutter();
  await Hive.openBox<String>('notesBox');
  await NotificationService().init();
  await initializeDateFormatting('ru');
```

```
  await Firebase.initializeApp(options:
DefaultFirebaseOptions.currentPlatform);
  await FirebaseAppCheck.instance.activate(
    // ignore: deprecated_member_use
    androidProvider: AndroidProvider.playIntegrity,
  );

  await Supabase.initialize(
    url: 'https://usgquikdsefdgmuumiyu.supabase.co',
    anonKey:

'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsIn
JlZiI6InVzZ3F1aWtkc2VmZGdtdXVtaXl1Iiwicm9sZSI6ImFub24iLCJpYXQiOj
E3NzQxNzI0MzksImV4cCI6MjA4OTc0ODQzOX0.asaTvbu5MB_ILPTfPPRJb_HFdR
33JhDBz-bgX63ohNg',
  );

  final container = ProviderContainer();
  final settings = container.read(settingsServiceProvider);

  final themeIndex = await settings.loadThemeIndex();
  final viewIndex = await settings.loadViewMode();
  final mainModeIndex = await settings.loadMainMode();
  final sortIndex = await settings.loadSortMode();
  final sortAsc = await settings.loadSortAsc();
  final accentColor = await settings.loadAccentColor();

  container.read(themeModeProvider.notifier).state =
      ThemeMode.values[themeIndex];
  container
      .read(viewModeProvider.notifier)
      .setInitial(ViewMode.values[viewIndex]);
  container
      .read(mainScreenModeProvider.notifier)
      .setInitial(MainScreenMode.values[mainModeIndex]);
  container
      .read(sortModeProvider.notifier)
      .setInitial(SortMode.values[sortIndex]);
  container.read(sortAscProvider.notifier).setInitial(sortAsc);
  container.read(accentColorProvider.notifier).state =
accentColor;

  runApp(UncontrolledProviderScope(container: container, child:
const App()));
}

class App extends ConsumerStatefulWidget {
  const App({super.key});

  @override
  ConsumerState<App> createState() => _AppState();
```

```
}

class _AppState extends ConsumerState<App> {
  final AppLinks _appLinks = AppLinks();
  StreamSubscription<Uri>? _linkSubscription;
  StreamSubscription<List<SharedMediaFile>>?
_intentDataStreamSubscription;

  @override
  void initState() {
    super.initState();
    _initDeepLinks();
    _initSharingIntent();
  }

  @override
  void dispose() {
    _linkSubscription?.cancel();
    _intentDataStreamSubscription?.cancel();
    super.dispose();
  }

  Future<void> _initDeepLinks() async {
    final initialLink = await _appLinks.getInitialLink();
    await _handleIncomingLink(initialLink);

    _linkSubscription = _appLinks.uriLinkStream.listen((uri) {
      _handleIncomingLink(uri);
    });
  }

  void _initSharingIntent() {
    // Handling media AND Text/URLs shared while app is in
memory
    _intentDataStreamSubscription =

ReceiveSharingIntent.instance.getMediaStream().listen((value) {
      if (value.isNotEmpty) {
        _handleSharedMedia(value);
      }
    }, onError: (err) {
      debugPrint("getMediaStream error: $err");
    });

    // Handling media AND Text/URLs shared while app is closed
    ReceiveSharingIntent.instance.getInitialMedia().then((value)
{
      if (value.isNotEmpty) {
        _handleSharedMedia(value);
      }
    });
```

```
  }

  void _handleSharedMedia(List<SharedMediaFile> media) async {
    if (media.isEmpty) return;

    final navigator = appNavigatorKey.currentState;
    if (navigator == null) return;

    // Reset to first tab (Notes)
    ref.read(navIndexProvider.notifier).state = 0;
    await Future<void>.delayed(Duration.zero);

    String? path;
    String? text;

    for (var file in media) {
      if (file.type == SharedMediaType.image) {
        path = file.path;
      } else if (file.type == SharedMediaType.text ||
          file.type == SharedMediaType.url) {
        text = file.path;
      }
    }

    if (path != null || text != null) {
      navigator.push(
        MaterialPageRoute(
          builder: (_) => NoteEditorScreen(
            initialImagePath: path,
            initialText: text,
          ),
        ),
      );
    }

    ReceiveSharingIntent.instance.reset();
  }

  Future<void> _handleIncomingLink(Uri? uri) async {
    if (uri == null) return;

    final navigator = appNavigatorKey.currentState;
    if (navigator == null || !mounted) return;

    // Handle "New Note" from Widget
    if (uri.scheme == 'notesapp' && uri.host == 'new') {
      ref.read(navIndexProvider.notifier).state = 0;
      await Future<void>.delayed(Duration.zero);
      navigator.push(
        MaterialPageRoute(builder: (_) => const
NoteEditorScreen()),
```

```
      );
      return;
    }

    // Handle "Screenshot & Note" from Tile
    if (uri.scheme == 'notesapp' && uri.host == 'screenshot') {
      final path = uri.queryParameters['path'];
      if (path != null) {
        ref.read(navIndexProvider.notifier).state = 0;
        await Future<void>.delayed(Duration.zero);
        navigator.push(
          MaterialPageRoute(
            builder: (_) => NoteEditorScreen(initialImagePath:
path),
          ),
        );
        return;
      }
    }

    // Handle existing note deep link
    final noteId = extractNoteIdFromDeepLink(uri);
    if (noteId == null) return;

    final notes = await ref.read(notesProvider.future);
    final note = _findNote(notes, noteId);
    if (note == null) return;

    ref.read(navIndexProvider.notifier).state = 0;

    await Future<void>.delayed(Duration.zero);
    navigator.push(
      MaterialPageRoute(builder: (_) => NoteEditorScreen(note:
note)),
    );
  }

  Note? _findNote(List<Note> notes, String id) {
    for (final note in notes) {
      if (note.id == id) return note;
    }
    return null;
  }

  @override
  Widget build(BuildContext context) {
    final themeMode = ref.watch(themeModeProvider);
    final accentColorValue = ref.watch(accentColorProvider);

    return DynamicColorBuilder(
      builder: (lightDynamic, darkDynamic) {
```

```
          final accentColor = accentColorValue == null
              ? null
              : Color(accentColorValue);
          final lightScheme = accentColor == null
              ? lightDynamic
              : ColorScheme.fromSeed(seedColor: accentColor);
          final darkScheme = accentColor == null
              ? darkDynamic
              : ColorScheme.fromSeed(
                  seedColor: accentColor,
                  brightness: Brightness.dark,
                );

          return MaterialApp(
            navigatorKey: appNavigatorKey,
            title: 'Заметки',
            theme: AppTheme.light(lightScheme),
            darkTheme: AppTheme.dark(darkScheme),
            themeMode: themeMode,
            home: const MainScreen(),
            debugShowCheckedModeBanner: false,
          );
        },
      );
    }
}
```

Приложение В

(обязательное)

Андроид - приложение «Интеллектуальные заметки»

Презентация

Интеллектуальные заметки

Автор работы:
Студент гр. 22ВВП2 Гавин Владимир Николаевич
Научный руководитель:
К.т.н., доцент кафедры ВТ Митрохина Наталья Юрьевна

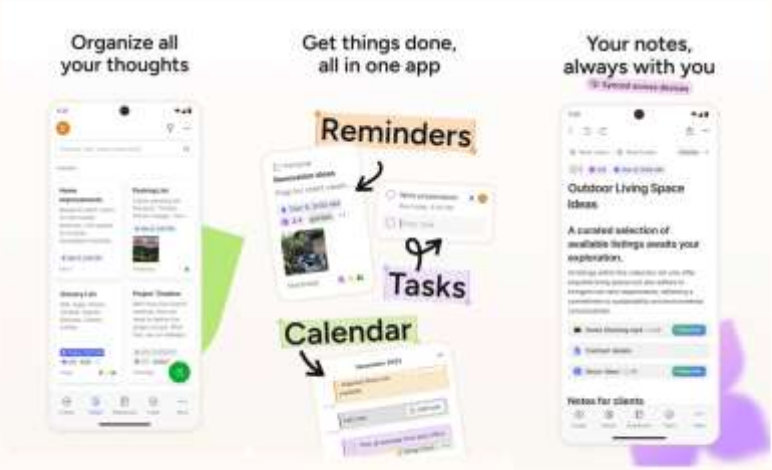
Пенза 2026

Актуальность проекта

Рост информационного потока перегружает мобильные заметки неструктурированным текстом. Из-за нехватки времени на ручную сортировку, форматирование и создание календарных напоминаний важные рабочие задачи, договоренности и контакты быстро теряются.

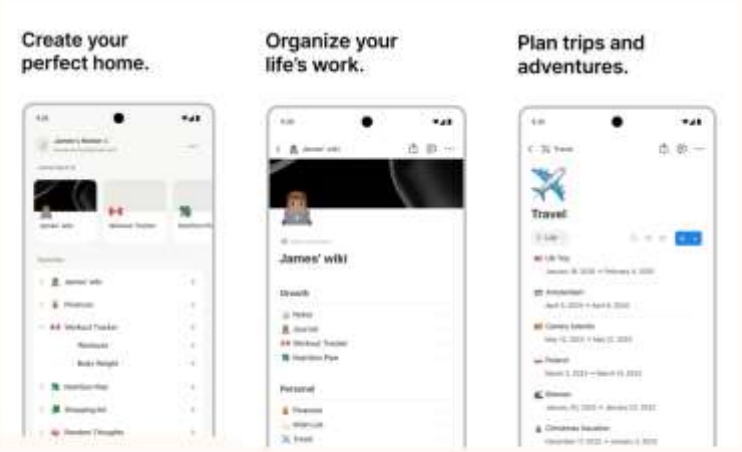
Интеграция LLM полностью автоматизирует этот процесс. Нейросеть может исправить текст, распределить его по папкам и связать данные с функциями смартфона.

Обзор аналогов: Evernote



Обзор аналогов: Google Keep

Обзор аналогов: Notion



Обзор аналогов: Obsidian

Обзор аналогов

Сервис	Особенности	Ограничения
Evernote	Веб-клиппер, OCR на фото	Лимиты загрузки, перегруженность интерфейса
Google Keep	Мгновенное создание заметок	Нет иерархии папок, ИИ инструментов
Notion	БД, ИИ-ассистент, модули расширений	Зависимость от сети, сложность обучения
Obsidian	Граф связей, Markdown разметка	Платная синхронизация, ручная настройка

Стек технологий

Диаграмма прецедентов

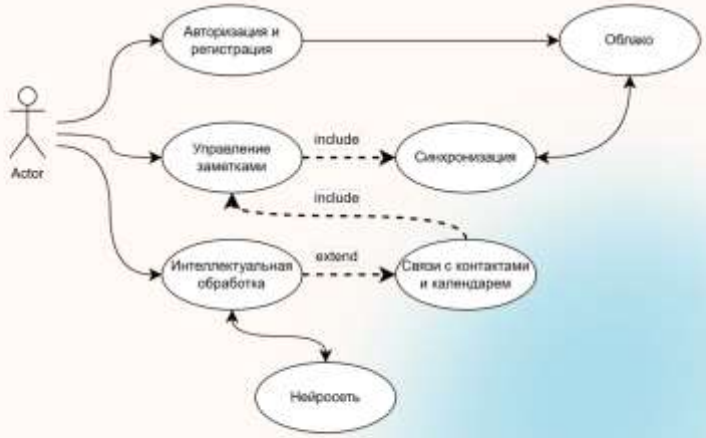


Диаграмма классов

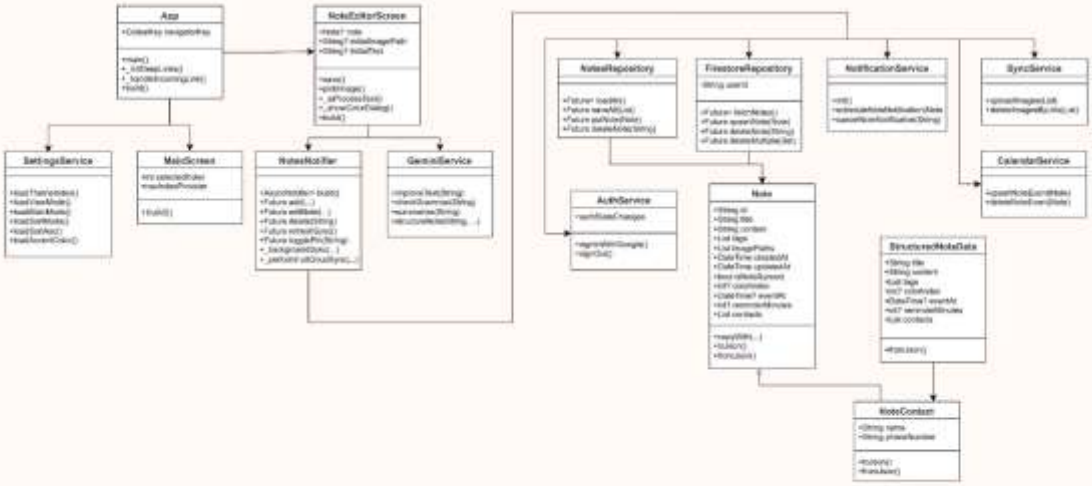


Диаграмма вариантов использования

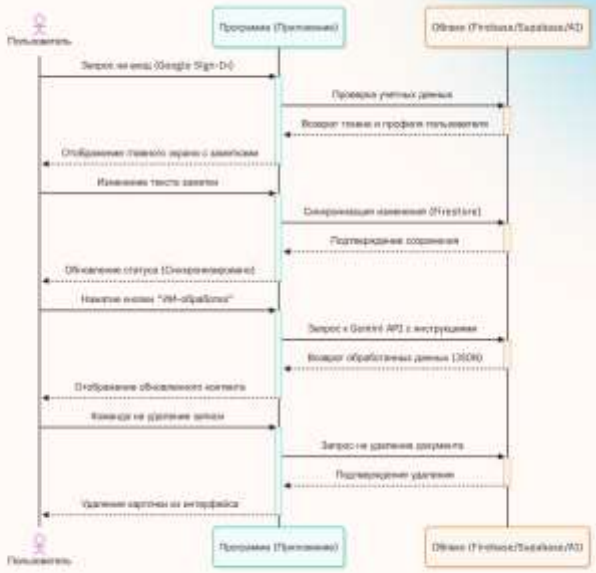
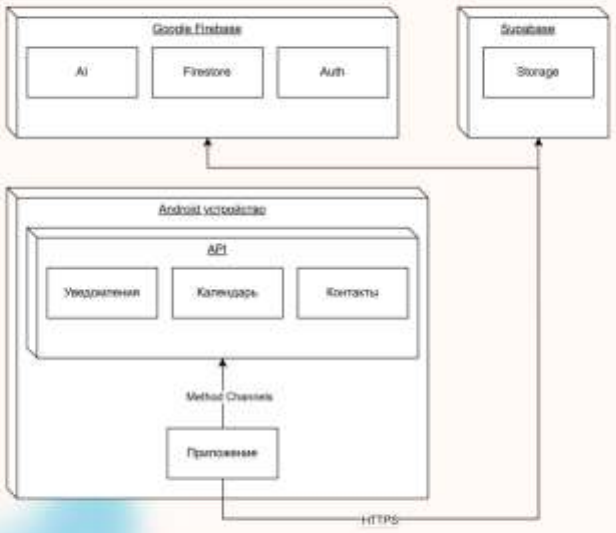
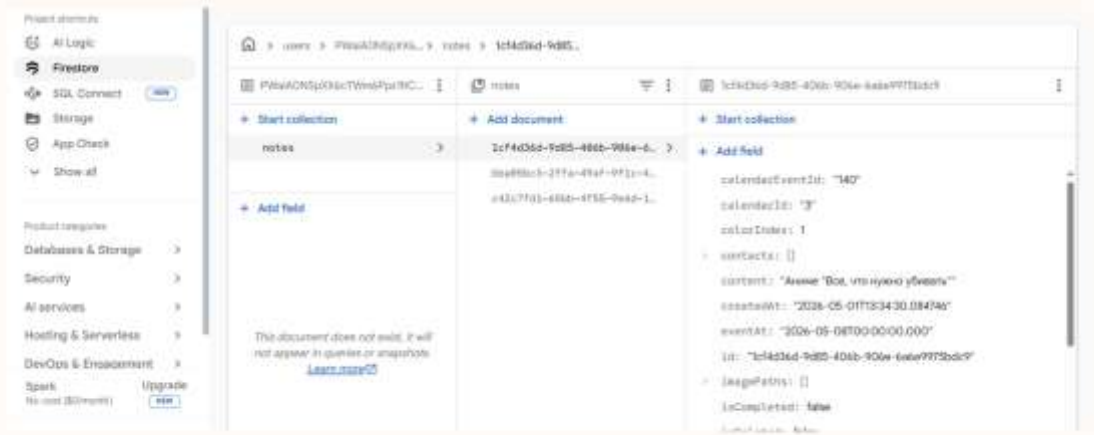


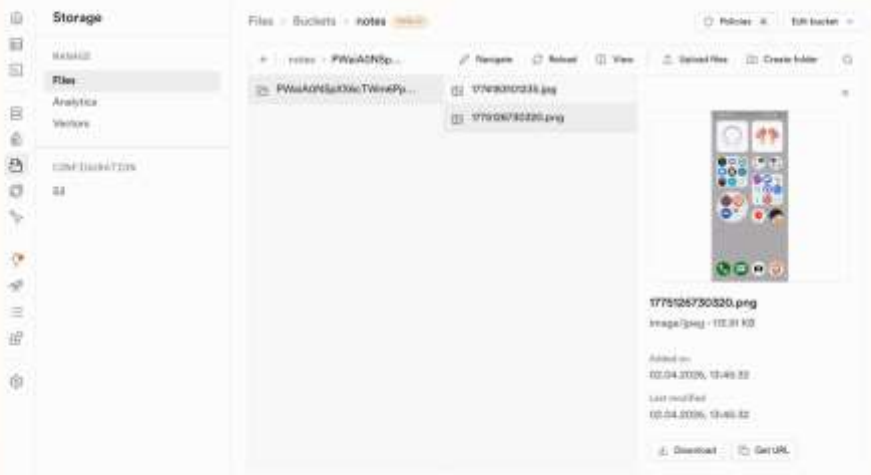
Диаграмма развертывания



Консоль Firebase Firestore

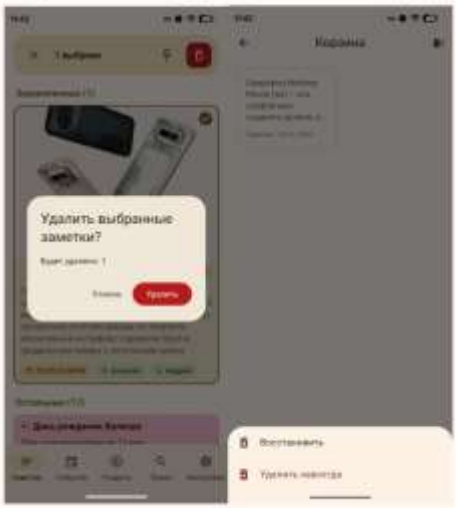


Консоль Supabase Storage



Клиентское приложение

Клиентское приложение

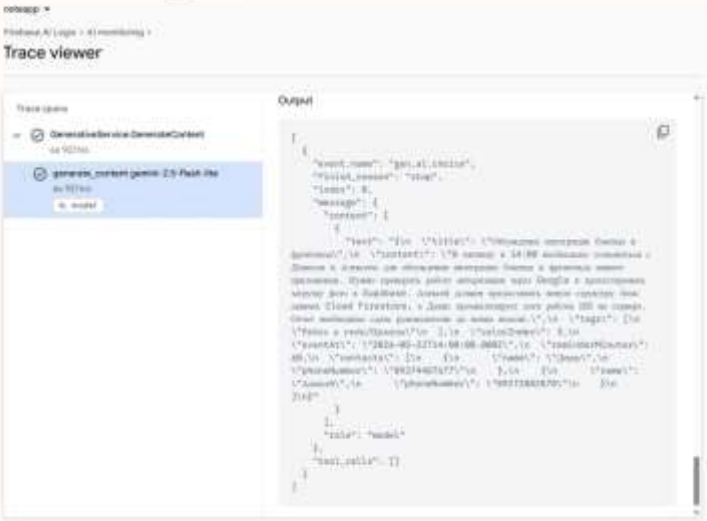


Работа ИИ модуля

до

после

Firebase AI Logic



Заключение

Интеграция ИИ и глубокая связь с функциями ОС — ключевые факторы создания современных приложений.

Результаты работы

Разработано и реализовано мобильное приложение для интеллектуального управления персональной информацией.

Объединены традиционные методы ведения заметок и современные возможности генеративного искусственного интеллекта.

Практический опыт

Освоена кроссплатформенная разработка на языке Dart с использованием фреймворка Flutter

Настроено взаимодействие с сервисами Firebase и Supabase

Реализована интеграция API нейросетевых моделей Gemini.

Спасибо за внимание!