МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

# РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ SIGMACARDS ДЛЯ ИНТЕРВАЛЬНОГО ПОВТОРЕНИЯ С СЕРВЕРНОЙ ЧАСТЬЮ НА ЯЗЫКЕ GO

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студентки 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Ореховой Алины Сергеевны

Научный руководитель
доцент, к. ф.-м. н. _____ С. В. Миронов

Заведующий кафедрой
доцент, к. ф.-м. н. _____ С. В. Миронов

Саратов 2026

# ВВЕДЕНИЕ

Большинство студентов сегодня работает с учебными материалами в цифровом виде: конспекты в PDF, фотографии с лекций, презентации прочно вошли в повседневный учебный обиход. При этом само по себе наличие материала ещё не означает, что он будет усвоен: из того, что действительно работает на долгосрочное запоминание, — распределённая практика и активное вспоминание. Системы интервального повторения (SRS) строятся именно на этих принципах, подстраивая расписание показов карточек под историю ответов конкретного пользователя. Узким местом остаётся подготовка самих карточек: для произвольного PDF-файла или фотографии конспекта это ручная, нередко утомительная работа. Языковые модели способны автоматизировать этот этап, хотя качество результата требует проверки.

Существующие SRS-приложения либо не предусматривают сквозного сценария создания карточек из произвольных учебных материалов, либо ориентированы на облачную модель без контроля над серверной частью. Для русскоязычного рынка дополнительным ограничением служит отсутствие штатного сценария OCR-распознавания кириллических конспектов с последующей ручной валидацией сгенерированных карточек в единой системе.

Цель работы — разработать систему SigmaCards: мобильное приложение на Flutter с серверной частью на Go, планировщиком повторений на основе алгоритма FSRS и интеграцией с ML-сервисом для автоматизированного создания учебных карточек.

Для реализации цели работы сформирован следующий набор задач:

1. Провести анализ психологических основ интервального повторения и сравнить алгоритмы SM-2 и FSRS;
2. Проанализировать существующие SRS-приложения и сформулировать требования к системе;
3. Спроектировать архитектуру сервиса повторений и сервиса генерации карточек, а также модель хранения данных;
4. Реализовать серверный контур: API, хранение состояния карточек и пересчёт расписания повторений;
5. Реализовать интеграцию с ML-сервисом для получения черновиков карточек из текстов и изображений;
6. Разработать мобильный клиент с поддержкой учебной сессии.

# 1 Описание разработки системы

## 1.1 Анализ предметной области и существующих решений

SigmaCards — система интервального повторения, в которой пользователь может не только учить карточки по расписанию, но и создавать их автоматически из своих учебных материалов. Мобильное приложение на Flutter отвечает за всё, что видит пользователь: колоды, карточки, учебные сессии, статистику. За хранение данных и расчёт расписания отвечает серверная часть на Go с PostgreSQL, а загруженные PDF-файлы и изображения уходят на обработку в отдельный ML-сервис, который возвращает черновые карточки через Kafka и gRPC. Слово «черновые» здесь принципиально: результат автоматической генерации пользователь просматривает и редактирует сам, прежде чем карточки попадут в расписание повторений.

Из рассмотренных аналогов наиболее близким является Anki: поддерживает классический сценарий интервального повторения и в актуальных версиях поддерживает FSRS. Однако сквозной сценарий — от загрузки учебного PDF через OCR-распознавание и генерацию черновых карточек до ручной проверки и включения в расписание повторений — не является для Anki единым штатным процессом и обычно требует внешних расширений. Quizlet и Duolingo ориентированы на облачную модель без самостоятельного развёртывания. Таким образом, SigmaCards рассматривается как прототип связанного процесса подготовки и повторения карточек для русскоязычных учебных материалов: импорт источника, OCR кириллицы, генерация черновика и дальнейшее повторение в едином контуре.

## 1.2 Алгоритмы интервального повторения

Алгоритм SM-2, предложенный Петром Возняком в 1987–1991 годах, стал первым массово применяемым решением. Пользователь выставляет оценку по шкале от 0 до 5, после чего система пересчитывает фактор лёгкости (EF, стартовое значение 2,5) и интервал. Ограничение SM-2 — упрощённая линейная модель: параметр EF не разделяет стабильность и сложность материала и не описывает текущее состояние памяти напрямую.

FSRS (Free Spaced Repetition Scheduler) опирается на DSR-модель. Для карточки хранятся три параметра: стабильность ($S$), сложность ($D$) и воспро-

изводимость ($R$). Текущая воспроизводимость вычисляется по формуле:

$$R(t, S) = \left(1 + \frac{FACTOR \cdot t}{S}\right)^{-0{,}5}, \tag{1}$$

где $t$ — число дней после последнего повторения, $FACTOR = 19/81 \approx 0{,}234$. При $t = S$ функция даёт $R = 0{,}9$ — целевой порог алгоритма. Следующий интервал при целевой воспроизводимости 0,9:

$$I = \left\lfloor \frac{S}{FACTOR} \cdot (R_{\text{target}}^{-2} - 1) \right\rfloor. \tag{2}$$

В опубликованных сравнениях на данных MaiMemo FSRS показал более низкие значения logloss и RMSE при прогнозировании успешного воспроизведения по сравнению с SM-2. В данной работе выбран FSRS как наиболее изученный подход с открытыми весами.

### 1.3 Выбранный вариант реализации и архитектура системы

По результатам анализа для каждого компонента определён конкретный инструмент. Серверная часть реализована на языке Go с фреймворком Gin: он обеспечивает нужную конкурентность и простое развёртывание в контейнере. В качестве СУБД выбран PostgreSQL 15: реляционная модель с поддержкой JSONB соответствует структуре данных проекта. Для асинхронного взаимодействия с ML-сервисом используется Apache Kafka, синхронный стриминг карточек реализован через gRPC. Мобильный клиент разработан на Flutter — единая кодовая база покрывает Android и iOS.

Диаграмма контейнеров системы (рисунок 1) показывает связи между компонентами. Мобильный клиент отправляет API-запросы к Go-серверу по HTTPS и получает потоковые ответы через gRPC. Go API хранит данные в PostgreSQL 15 и использует Redis для кэширования токенов и проверки идемпотентности POST-запросов. Через Kafka Go API публикует задачи на генерацию; ML-сервис выполняет OCR и генерирует черновики карточек.

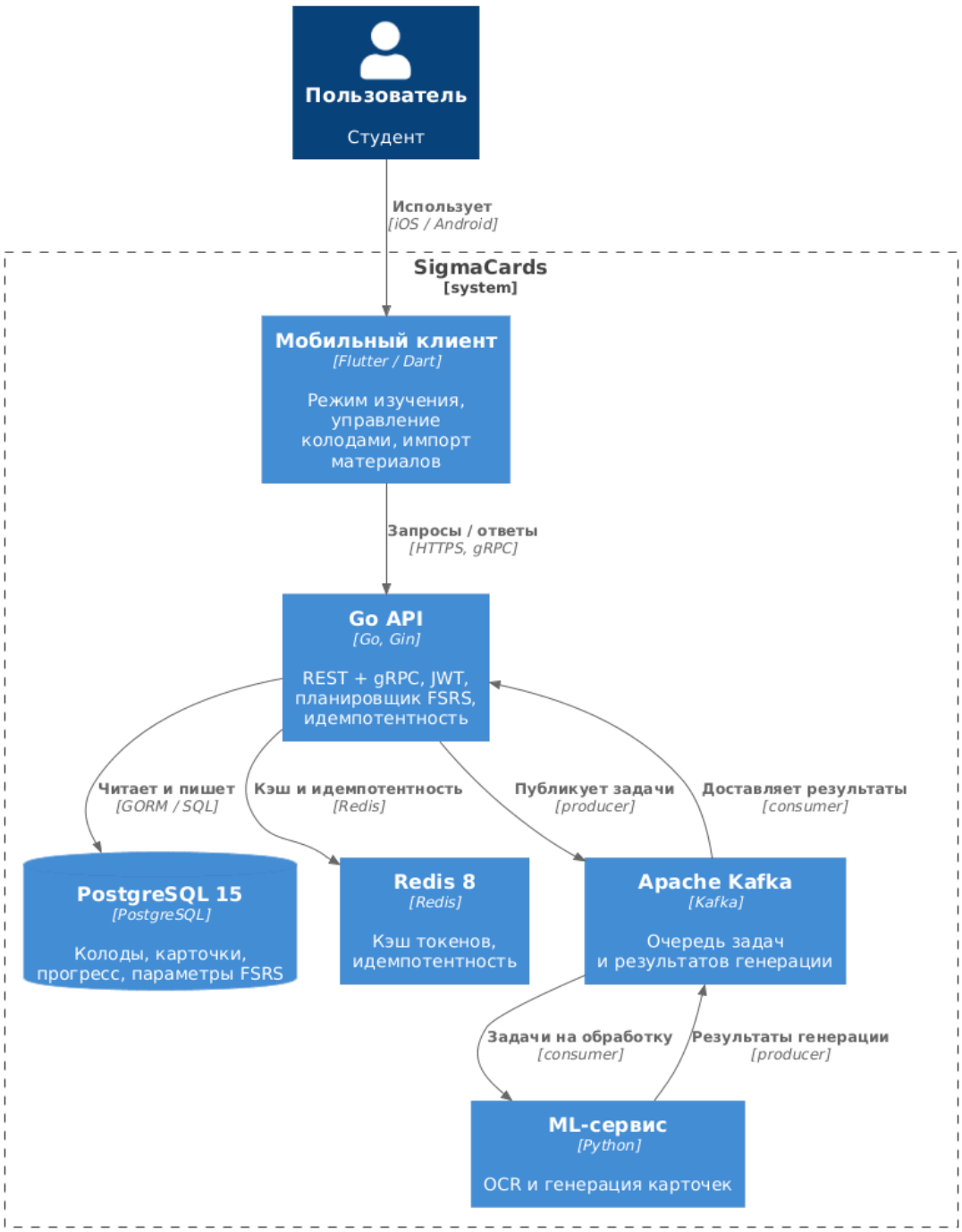


Рисунок 1 – Диаграмма контейнеров системы SigmaCards

Backend SigmaCards разделён на четыре слоя, чтобы HTTP/GORM-код не попадал в бизнес-логику повторений. Зависимости направлены строго внутрь: слой моделей (`internal/model/`) содержит структуры предметной области без зависимостей от инфраструктуры; слой сервисов (`internal/service/`) реализует бизнес-логику и зависит только от интерфейсов репозиториев; слой репозиториев (`internal/repository/`) инкапсулирует работу с PostgreSQL через GORM; слой обработчиков (`internal/handler/`) принимает HTTP-запросы и делегирует логику сервисам. Такое разделение имеет практическое значение для

тестируемости: сервисный слой зависит от интерфейсов, а не от конкретных реализаций, что позволяет подменять репозитории mock-объектами в юнит-тестах без запуска базы данных. Пакет `pkg/fsrs/` содержит вычисления алгоритма и не зависит ни от одного инфраструктурного компонента — его можно тестировать изолированно. Из DDD-подхода заимствованы выделение доменных сущностей, явное описание репозиториев как интерфейсов и размещение бизнес-правил в сервисном слое.

Схема базы данных разработана в соответствии с третьей нормальной формой и включает девять таблиц. Таблица `users` использует UUID как первичный ключ, bcrypt-хеш пароля и soft delete через `deleted_at`; partial-индекс `UNIQUE (email) WHERE deleted_at IS NULL` позволяет повторно зарегистрировать email после мягкого удаления, поскольку удалённые записи выпадают из условия уникальности. Таблица `refresh_sessions` хранит хеш токена, а не сам токен: если база окажется скомпрометированной, злоумышленник получит только хеши, по которым нельзя войти в чужой аккаунт. Таблица `decks` реализует оптимистичную блокировку через поле `version`: обновление применяется только при совпадении версии, иначе сервис отвечает HTTP 409, сигнализируя клиенту о необходимости повторить запрос с актуальными данными. Таблица `flashcards` хранит содержимое в JSONB-поле с типами блоков `text`, `image`, `latex`, `cloze`; валидация выполняется на трёх уровнях — HTTP DTO, сервисный слой и CHECK-constraint в самой БД. Таблица `user_cards` хранит FSRS-параметры для каждой пары (пользователь, карточка), а `review_logs` является append-only-журналом со снимком параметров памяти на момент каждого повторения. Загружаемые файлы хранятся в MinIO — S3-совместимом объектном хранилище; в таблице `images` сохраняется только путь к объекту (`object_name`), что исключает хранение бинарных данных в PostgreSQL и упрощает резервное копирование.

### 1.4 Go-проект и слой доступа к данным

Взаимодействие с PostgreSQL реализовано через GORM v2. Сервисный слой зависит от интерфейсов репозиториев, а не от конкретных реализаций — это позволяет подменять их mock-объектами в тестах без запуска базы данных. Интерфейс репозитория, например, объявляет методы `Create`, `GetByID`, `Update`, `Delete` и `ListByUser`, каждый из которых принимает контекст запроса для поддержки отмены и дедлайнов. Оптимистичная блокировка на уровне

репозитория: обновление выполняется только если версия записи совпадает с переданной клиентом, иначе `RowsAffected` равно нулю и сервис возвращает HTTP 409 Conflict. Управление схемой базы данных осуществляется через миграции библиотеки golang-migrate: при каждом старте приложения новые миграции применяются автоматически.

### 1.5 Планировщик повторений на основе FSRS

Планировщик вынесен в самодостаточный пакет `pkg/fsrs/` без зависимостей от инфраструктурного кода. Каждая карточка проходит через несколько состояний: новая карточка (`StateNew`) при первом показе переходит в состояние обучения (`StateLearning`). После нескольких успешных ответов подряд она переходит в режим повторения (`StateReview`) с постепенно увеличивающимися интервалами. Ошибка в режиме повторения переводит карточку в переобучение (`StateRelearning`).

При первом показе начальная стабильность $S_0$ задаётся предобученными весами: оценка «Again» — $S_0 = 0{,}4$ (следующий показ через несколько часов), «Good» — $S_0 = 3{,}44$ (около 3 дней), «Easy» — $S_0 = 9{,}03$ (около 9 дней). После каждого успешного повторения сложность корректируется: оценки выше среднего снижают её, ниже — повышают; второй член формулы работает как стабилизатор, притягивая значение к 4,072 — предобученному весу, полученному на данных сервиса MaiMemo. Обновление стабильности определяет прирост интервала: коэффициент $w$ зависит от оценки — «Hard» даёт минимальный прирост (1,1), «Easy» — максимальный (1,3). Минимальный интервал ограничен одним днём. Сервисный уровень оркестрирует операцию повторения в рамках одной транзакции: загружает запись `UserCard`, вызывает `fsrs.Schedule`, сохраняет обновлённые параметры в `user_cards` и добавляет строку в `review_logs`.

### 1.6 HTTP API

HTTP-сервер построен на фреймворке Gin. В системе используются два типа токенов: access-токен короткоживущий (30 минут), подписан HMAC-SHA256 и не хранится в базе; refresh-токен долгоживущий (30 суток), хранится в `refresh_sessions` в виде хеша и инвалидируется при выходе. Middleware аутентификации извлекает access-токен из заголовка `Authorization`, проверяет подпись и при успехе помещает идентификатор пользователя в контекст запроса.

Middleware идемпотентности предотвращает дублирование запросов при сетевых сбоях: клиент прикладывает уникальный заголовок `Idempotency-Key`, middleware проверяет его в Redis — если ключ уже есть, возвращает кэшированный ответ без повторного выполнения; если нет — выполняет запрос и кэширует результат с TTL 24 часа. POST-запросы на создание колод, карточек и повторений таким образом становятся идемпотентными.

Загрузка PDF для генерации карточек устроена в два шага: сначала клиент получает временный URL для загрузки файла напрямую в MinIO, затем отправляет запрос на старт генерации — сервер ставит задачу в очередь и немедленно отвечает идентификатором. Жизненный цикл задачи: `queued` → `extracting` → `generating` → `completed` (или `failed` / `cancelled`). Клиент периодически опрашивает статус; по завершении получает черновики карточек для проверки и редактирования.

Перед учебной сессией клиент запрашивает очередь карточек с наступившим сроком; после оценки карточки передаёт на сервер оценку от 1 до 4, время обдумывания и версию записи. Сервер пересчитывает параметры FSRS и возвращает дату следующего показа, стабильность и сложность. Ошибки сервисного слоя транслируются в HTTP-коды в одном месте: `ErrNotFound` → 404, `ErrConflict` → 409, `ErrForbidden` → 403.

### 1.7 Интеграция с ML-сервисом

Kafka-контур реализует асинхронный обмен задачами генерации. Для взаимодействия определены четыре топика: `ml.cards.extract.request` (Go→ML, запрос OCR), `ml.cards.extract.response` (ML→Go, результат распознавания), `ml.cards.generate.request` (Go→ML, запрос генерации) и `ml.cards.generate.response` (ML→Go, сгенерированные карточки). Ключом каждого сообщения служит `generation_id`, что гарантирует попадание всех сообщений одной задачи в один раздел и сохраняет их порядок. Консьюмеры Go-бэкенда объединены в consumer group: при горизонтальном масштабировании каждое сообщение обрабатывается ровно одним экземпляром.

Синхронная интеграция по gRPC используется для стриминга карточек из текста в реальном времени. Выбран gRPC server-streaming: работает поверх HTTP/2, обеспечивает мультиплексирование нескольких потоков. Flutter-клиент устанавливает gRPC-соединение к Go-серверу, передавая access-токен в metadata; Go-сервер проверяет JWT и проксирует вызов к ML-сервису, не

раскрывая его адрес клиенту напрямую.

### 1.8 Мобильный клиент Flutter

При старте приложение проверяет наличие сохранённого токена и направляет пользователя либо на главный экран, либо на онбординг. Маршрутизация построена через `MainShellScreen` с нижней навигационной панелью, переключающей разделы без перезагрузки состояния. Все сетевые запросы сосредоточены в классе `ApiService`: он добавляет заголовок авторизации к каждому запросу, обрабатывает ошибки и десериализует JSON-ответы в доменные модели.

Для хранения JWT-токенов используется `flutter_secure_storage`: на Android применяется Keystore, на iOS — Keychain. Такой подход соответствует требованиям OWASP MASVS-STORAGE-1: токены не хранятся в открытом виде в файловой системе. Режим изучения реализован в `StudySessionScreen`: анимация переворота карточки выполнена через `TweenAnimationBuilder` с трёхмерным вращением. Принципиальное архитектурное решение: клиент не пересчитывает параметры FSRS самостоятельно — он отправляет оценку на сервер и отображает полученную дату следующего показа, что обеспечивает согласованность состояния обучения при входе с разных устройств. Если сервер возвращает 409 Conflict (версия карточки устарела), клиент обновляет очередь и продолжает сессию без потери прогресса.

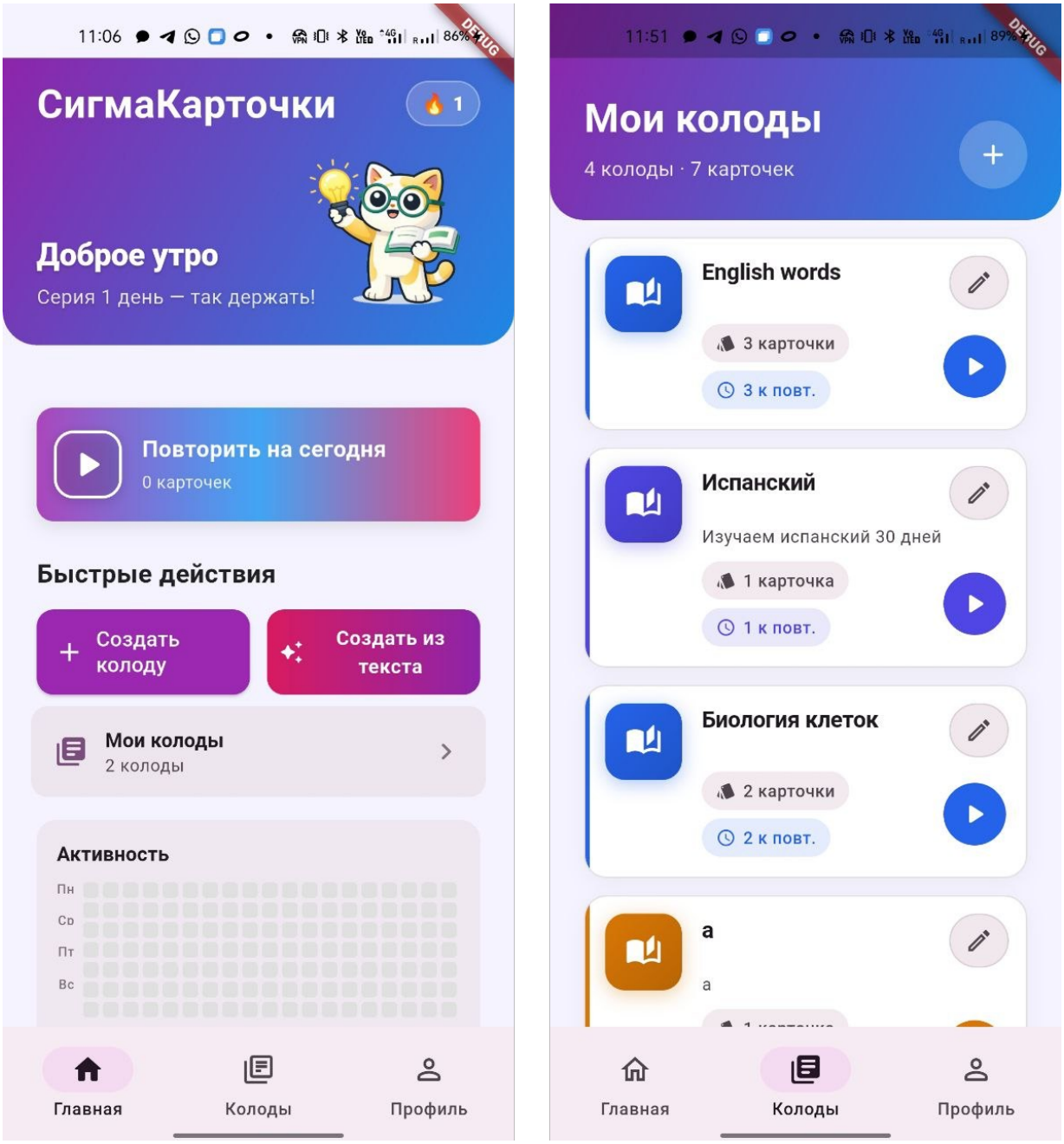


Рисунок 2 – Главный экран и экран списка колод мобильного приложения SigmaCards

### 1.9 Описание работы с приложением

При первом запуске приложение предлагает онбординг, после которого открывается экран аутентификации с переключением между входом и регистрацией. После успешного входа токены сохраняются в защищённом хранилище устройства, и пользователь попадает на главный экран. Там отображаются список колод, количество карточек к повторению на сегодня и тепловая карта учебной активности. С главного экрана можно создать новую колоду, открыть существующую или сразу запустить сессию повторения по всем просроченным карточкам.

При создании колоды пользователь задаёт название и описание; карточки добавляются вручную через редактор или генерируются с помощью ML-сервиса. Каждая карточка состоит из лицевой и оборотной сторон и может содержать текст, изображение, формулу или пропуск для заполнения. Для начала учебной сессии пользователь выбирает колоду — экран показывает лицевую сторону карточки; при нажатии карточка переворачивается и открывается ответ. После ознакомления с ответом пользователь выбирает оценку: «Again», «Hard», «Good» или «Easy». Оценка отправляется на сервер, который пересчитывает параметры FSRS и назначает дату следующего повторения; прогресс сессии отображается индикатором в верхней части экрана.

На экране импорта пользователь вводит текст или загружает PDF-файл. При текстовом вводе карточки появляются по мере генерации — пользователь видит результат в реальном времени через gRPC-стриминг и может остановить процесс досрочно. При загрузке PDF файл сначала сохраняется в MinIO через presigned URL, после чего запускается фоновая задача: Go-бэкенд публикует сообщение в Kafka, ML-сервис выполняет OCR и генерацию, статус обновляется по завершении. Полученные черновики можно проверить и отредактировать перед сохранением в колоду.

Вся инфраструктура запускается единой командой `docker compose up -build -d`. Сервисы PostgreSQL и Redis снабжены healthcheck — Docker Compose запускает API-контейнер только после их готовности; миграции применяются при старте автоматически. API спроектирован stateless, что позволяет масштабировать его горизонтально без дополнительной координации. Сборка релизного APK выполняется командой `flutter build apk -release`; REST API доступен на порту 8080, документация Swagger UI — по адресу `/swagger/index.html`.

# ЗАКЛЮЧЕНИЕ

В рамках данной выпускной квалификационной работы спроектирована и реализована система SigmaCards: мобильное приложение на Flutter, серверная часть на Go с PostgreSQL и интеграция с ML-сервисом, который по загруженным материалам генерирует черновики карточек. Все поставленные задачи выполнены: реализован полный цикл от регистрации пользователя до учебной сессии с пересчётом параметров FSRS; настроен асинхронный контур генерации через Kafka и синхронный стриминг через gRPC; разработан мобильный клиент с защищённым хранением токенов и анимированным режимом изучения.

Архитектурно система устроена в два контура. Синхронный — REST API и gRPC: клиент запрашивает карточки, отправляет оценки, получает статистику в реальном времени. Асинхронный — Kafka: длительные задачи OCR и генерации ставятся в очередь и обрабатываются ML-сервисом независимо от основного API. В прототипе реализована упрощённая форма FSRS: DSR-состояние пересчитывается на сервере с фиксированными предобученными весами.

Среди перспективных направлений развития наиболее интересны: персонализация планировщика через FSRS 6 (индивидуальная калибровка параметров под каждого пользователя); адаптивная генерация карточек с анализом истории ошибок; совместные колоды с разграничением прав между преподавателем и студентами; интеграция с университетскими LMS-системами для построения расписания повторений вокруг реального учебного плана и дат экзаменов.