САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №1

Выполнил:

Хисаметдинова Д.Н.

Группа
K3441

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача
Необходимо спроектировать набор следующих диаграмм:

- общая архитектура решения (сервисы и их взаимосвязи, клиент-серверное взаимодействие);
- диаграмма компонентов;
- диаграммы БД по каждому сервису;
- диаграммы основных пользовательских сценариев (те сценарии, которые позволяют вашим приложением полноценно воспользоваться, пройти весь путь).

Ход работы

1. **Высокоуровневое описание архитектуры**

inBeTwin - это распределенная микросервисная платформа для автоматизации коммуникаций с помощью AI-агентов. Система построена на основе **событийно-ориентированной архитектуры** (Event-Driven Architecture) с использованием паттернов **API Gateway**, **Message Queue** и **Domain-Driven Design**.

Ключевые архитектурные принципы:

1. **Микросервисная архитектура** - каждый сервис отвечает за одну бизнес-доменную область
2. **Полиглот-персистенс** - разные базы данных для разных задач (PostgreSQL, Redis, Pinecone, MinIO)
3. **Асинхронная обработка** - тяжелые операции через очереди сообщений
4. **API Gateway Pattern** - единая точка входа с централизованной аутентификацией
5. **Горизонтальное масштабирование** - stateless сервисы

1. **Клиентский слой**

1. 1 Mobile App (Kotlin Multiplatform)

- Регистрация и аутентификация пользователей
- Управление подключениями к социальным сетям

- Настройка поведения AI-агента
- Мониторинг действий агента в реальном времени через ***Web Sockets***
- Управление базой знаний (документы, QA пары)
- Просмотр и квалификация лидов

1. 2 Telegram Bot

- Получение сообщений от конечных пользователей
- Отправка автоматических ответов через AI-агента
- Webhook-based архитектура для real-time обработки

1.3 Web

То же, что в мобилке, но с соответствующим интерфейсом

Все, кроме уведомлений – REST API (я хз нужно ли тут gRPC для блейзингли фаст перформанса)

2. **API Gateway Layer**

Authentication & Authorization

- Валидация JWT токенов в каждом запросе
- Извлечение userId из токена и передача в микросервисы
- Автоматический refresh истекших access токенов
- Проброс заголовка X-User-ID во все backend сервисы

Маршрутизация

/api/v1/auth/* → Auth Service (Node.js)

/api/v1/social/* → Social Service (Go)

/api/v1/agent/* → Agent Service (Node.js)

/api/v1/llm/* → LLM Service (Node.js)

/api/v1/rag/* → RAG Service (Node.js)

/api/v1/leads/* → Lead Scoring Service (Python)

- Централизованное логирование всех запросов (метод, путь, статус, latency)
- Автоматический fallback при недоступности микросервиса

3. **Микросервисы**

Каждый микросервис изолирован, имеет собственную базу данных и отвечает за конкретную бизнес-область:

Auth Service (Node.js + NestJS) – Управление пользователями и аутентификация

- User Management: Регистрация, профили, обновление данных
- Authentication: Login/logout, JWT генерация (access + refresh tokens)
- OAuth Integration: Вход через Google, Yandex
- Subscription Management: Управление тарифными планами (Free, Pro, Business)
- Password Reset: Безопасный механизм сброса пароля через email
- База данных (PostgreSQL):
    - users - основная таблица пользователей
    - oauth_providers - связи с OAuth провайдерами
    - refresh_tokens - активные refresh токены
    - subscriptions - информация о подписках

Social Service (Go + Fiber) – Интеграция с социальными сетями

Коммуникация:

1. **Platform Integration**
    - Поддержка Telegram, VK, Twitter/X, MAXXX
    - OAuth 2.0 flow для подключения аккаунтов
    - Хранение access/refresh токенов (encrypted)
2. **Webhook Management**
    - Прием входящих сообщений через webhooks
    - Валидация подписей (signature verification)
    - Быстрый ответ платформе + асинхронная обработка
    - Duplicate detection для предотвращения повторной обработки
3. **Message Operations**
    - **Inbound:** Прием → Сохранение → Публикация в RabbitMQ

    - **Outbound:** Получение команды → Отправка через Platform API → Логирование
4. **Contact Synchronization**
    - Периодическая синхронизация контактов (каждый час)
    - Real-time обновления через webhooks
    - Фильтрация контактов (пользователь выбирает, с кем агент может общаться)

База данных (PostgreSQL):

- integrations - подключенные социальные аккаунты
- contacts - синхронизированные контакты с метаданными
- messages - полная история сообщений (inbound/outbound)
- webhooks - конфигурация webhook endpoints

Event Publishing (RabbitMQ):

- message.received → для обработки LLM Service
- message.sent → для логирования и аналитики
- integration.connected → для уведомлений

Синхронная: REST API для прямых запросов

Асинхронная: Публикует события user.registered, subscription.updated

Agent Service (Node.js + NestJS) – Управление AI-агентами и их конфигурацией

1 Agent Configuration Management

1. Tools
    a. Built-in tools

        - `save_user_info` - сохранение информации о клиенте - `create_task` - создание задачи в Notion/Asana - `schedule_event` - добавление встречи в календарь - `call_operator` - эскалация к человеку

    b. HTTP Webhooks
    c. MCP Servers
2. Логирование действий агентов со статусами, деталями, типами тулов

3. БД
4. Кэширование, потому что системный промпт например, меняется раз в день максимум

LLM Service (Node.js + NestJS) – обработка сообщений и генерация ответов

1. **Tool Calling (Function Calling)**
- OpenAI возвращает tool_call с именем функции и параметрами
- LLM Service вызывает Agent Service для выполнения
- Результат добавляется в контекст
- Повторный вызов OpenAI для финального ответа
2. **Response Routing**
- Определение тональности ответа (sentiment analysis)
- Toxicity check (фильтрация неуместных ответов)
- Выбор агента (если у пользователя несколько агентов)

RAG Service (Node.js + NestJS) – управление базой знаний

Стек: Pinecone (vector database), Huggingface Embeddings API, Openrouter

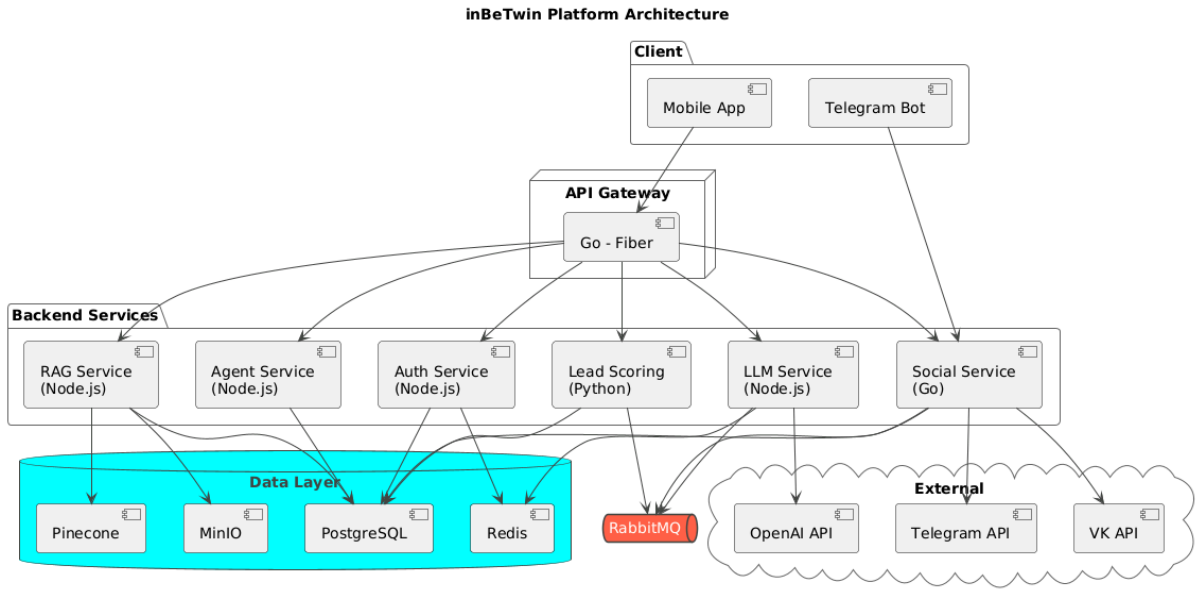

Рисунок 1 – Высокоуровневая архитектура сервиса

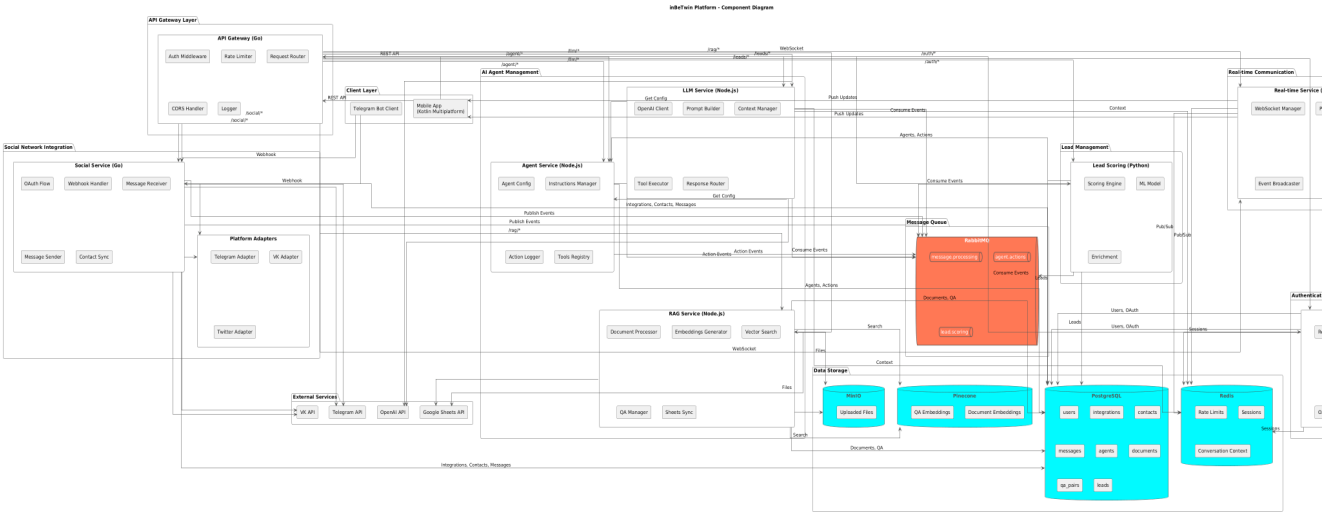

Рисунок 2 – Диаграмма компонентов

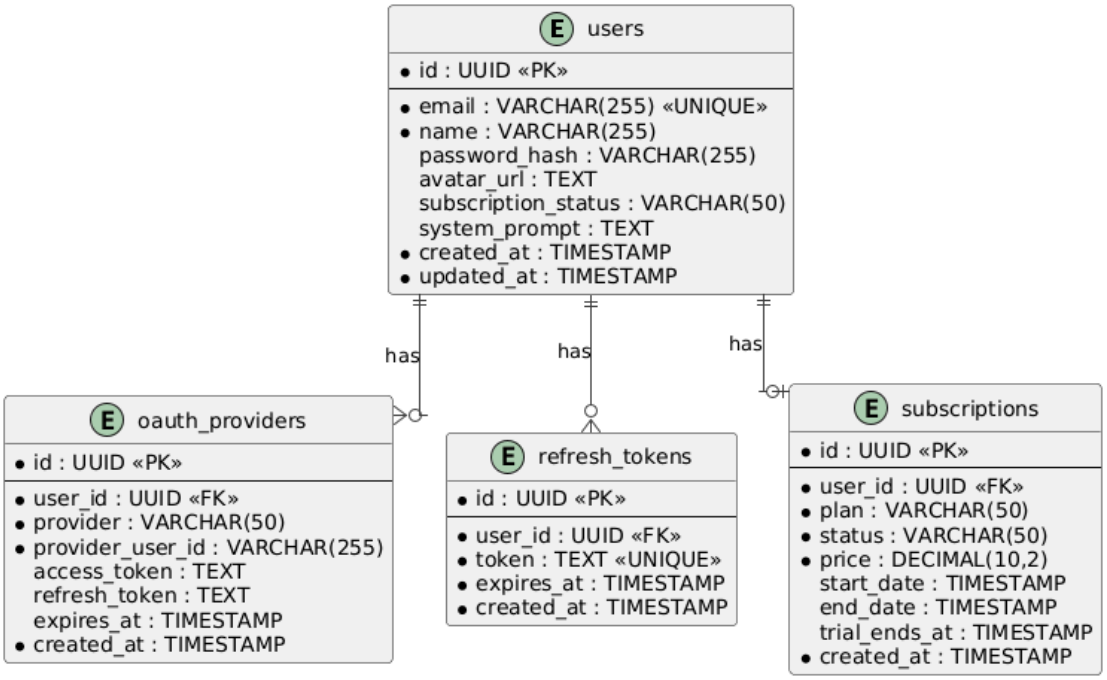

Рисунок 3 – ERD Auth Service

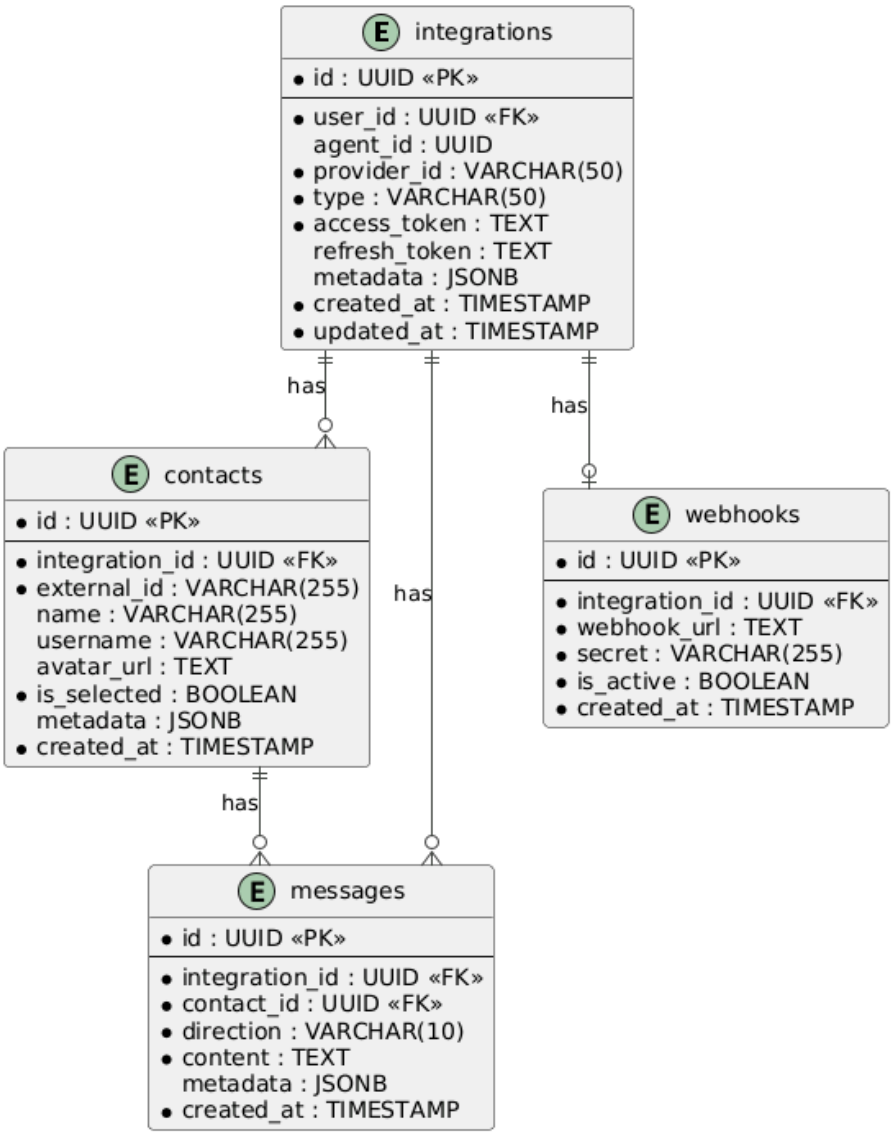

Рисунок 4 – ERD of Social Service

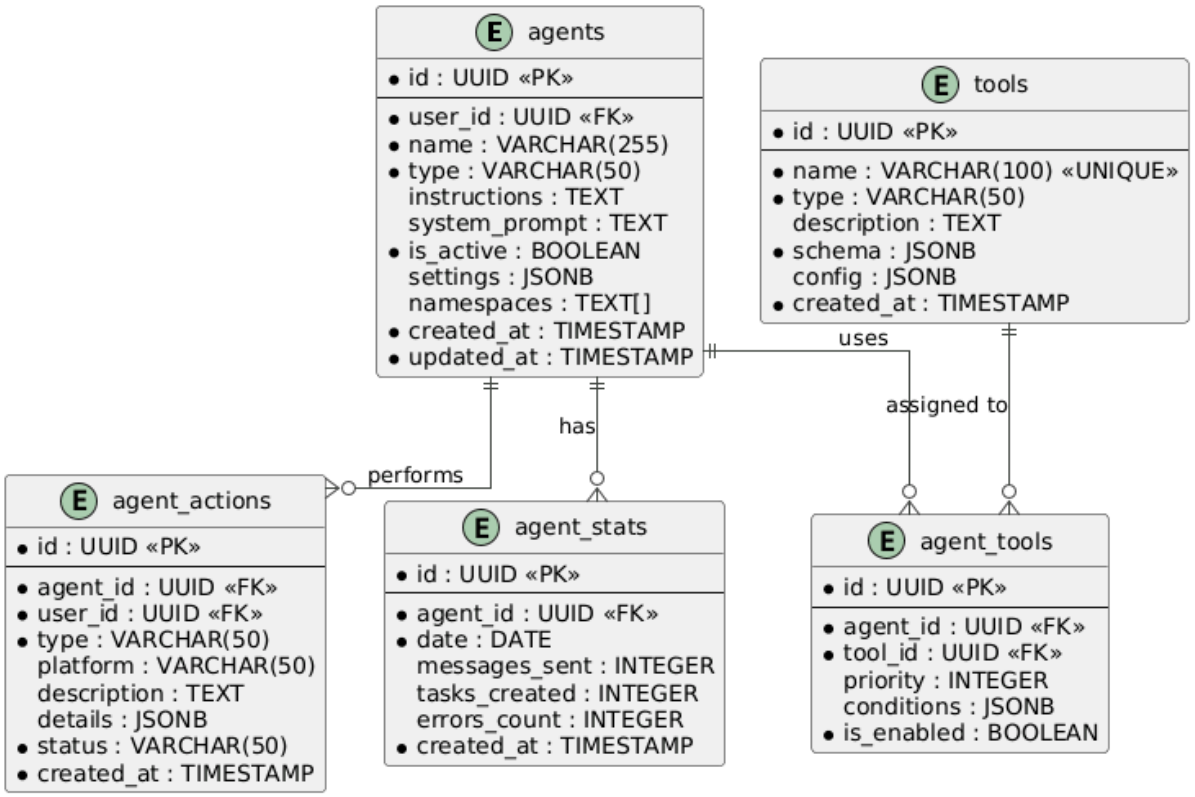

Рисунок 5 – ERD of Agent Service

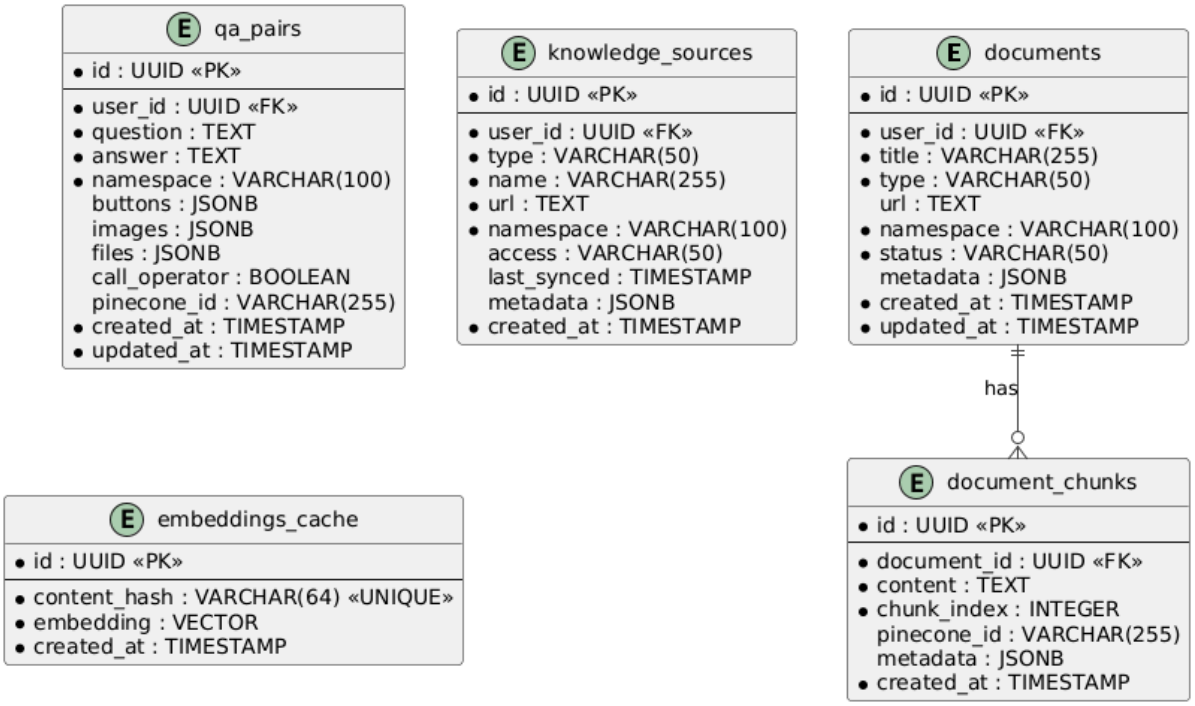

Рисунок 6 – ERD of RAG Service

Сценарий 1: Регистрация и настройка агента

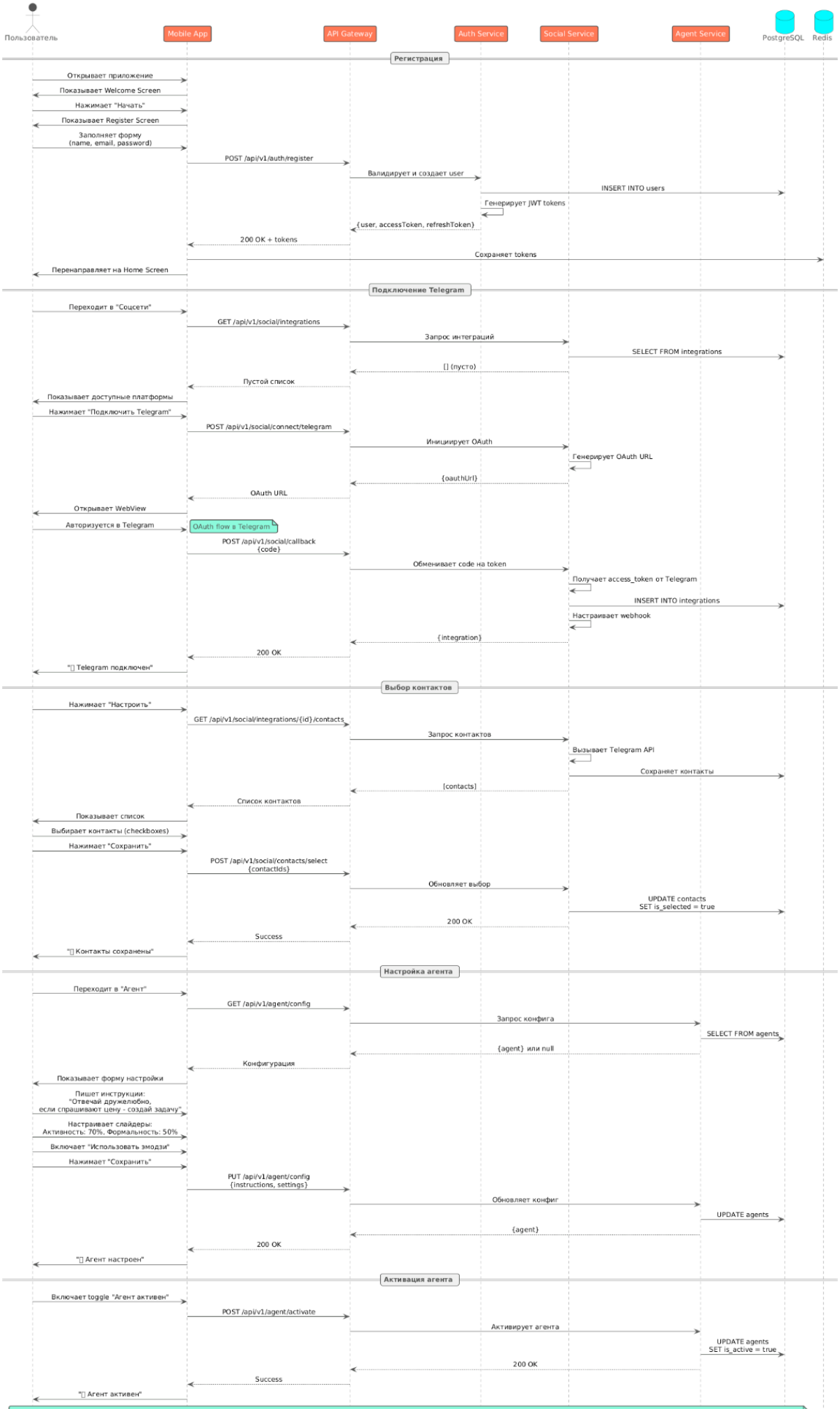

Рисунок 7 – Use Case Diagram регистрации

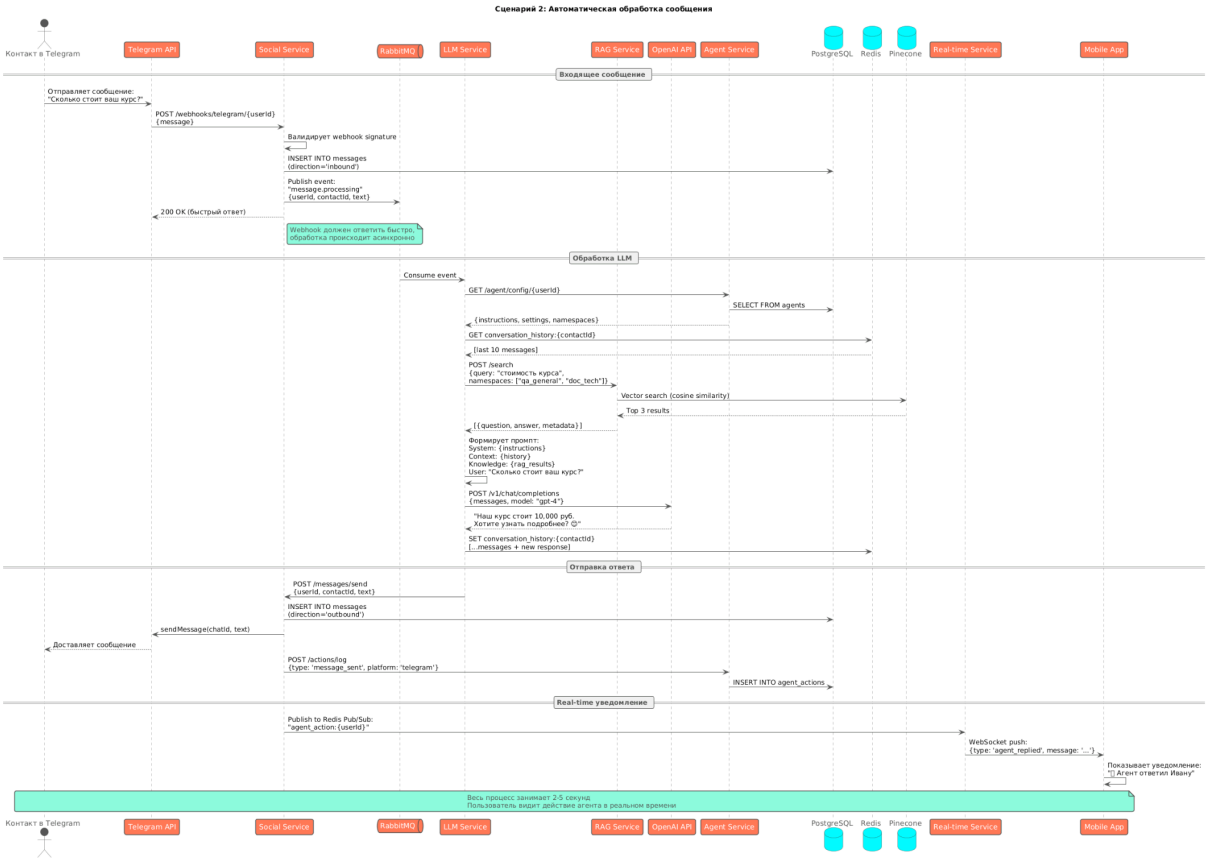

Рисунок 8 – Use Case Diagram взаимодействия с агентами

Вывод

Вывод по работе: была придумана большая идея сервиса, который декомпозирован на микровервисы. После анализа аудитории были написаны use-case-diagrams, набросано как должна глобально выглядеть оптимальная архитектура исходя из доменной области, best practices в этой сфере и 1 лекции по бэкенд-разработке.