

Промежуточные варианты (Ожерельев)

Ранние черновики архитектуры и T3 Sync Agent. Не каноничны — сохранены для истории согласования.

- [Техническое описание архитектуры продукта](#)
- [T3: Разработка агента синхронизации данных \(Sync Agent\)](#)

Техническое описание архитектуры продукта

Технической описание архитектуры продукта

Версия: 1.0 **Дата:** 3 мая 2026

Источник: Ожерельев В.А. файл Техническое_описание_архитектуры.[md](#)

1. Обзор продукта

1.1 Назначение системы

Мультиагентная AI-система **Grace CRM Assistant** предназначена для автоматизации процессов продаж и повышения эффективности работы отдела продаж компании.

Система реализует цифровых AI-ассистентов (агентов), которые анализируют данные CRM, контролируют качество данных, формируют рекомендации менеджерам и обеспечивают управленческий контроль.

Система интегрируется с Grace CRM (Laravel BAP) через API и webhooks и функционирует как надстройка (advisory layer) над CRM, не нарушая принцип *CRM = source of truth*

1.2 Цели внедрения

Основные бизнес-цели системы:

- Повышение конверсии продаж
- Снижение потерь лидов

- Повышение качества данных в CRM
 - Автоматизация контроля работы менеджеров
 - Ускорение реакции на клиентские события
 - Формирование единого корпоративного знания (knowledge layer)
-

1.3 Функциональные ВОЗМОЖНОСТИ

Система обеспечивает:

1.3.1 AI-ассистирование продаж

- Анализ текущего состояния сделки
- Генерация рекомендаций (следующее действие, срок, обоснование)
- Контекстный анализ на основе CRM + базы знаний

1.3.2 Контроль качества данных

- Проверка полноты карточек сделок
- Выявление пропущенных активностей
- Формирование задач менеджерам

1.3.3 Уведомления и эскалации

- Уведомления через Telegram / CRM
- Эскалация на РОП при отсутствии реакции
- SLA-контроль

1.3.4 Корпоративный поисковый контур

- Поиск по CRM-данным (OpenSearch)
- Поиск по базе знаний (RAGflow)
- Гибридный поиск (semantic + keyword)

1.3.5 Управление знаниями

- Индексация документов
- Формирование wiki-страниц
- Поиск с цитированием
- Анализ актуальности знаний

1.4 Состав системы

Система включает два функциональных контура: контур продаж и контур базы знаний.

Контур 1 -- Sales AI (Grace CRM Assistant)

Агент	Наименование	Назначение
Sync Agent	Агент синхронизации	Получает данные из Grace CRM через API/webhooks, обновляет PostgreSQL и инициирует индексацию в OpenSearch
Quality Agent	Агент контроля качества данных	Проверяет полноту и корректность данных CRM, выявляет пробелы, формирует задачи и сигналы
Recommendation Agent	Агент рекомендаций	Анализирует данные CRM и базы знаний, формирует конкретные рекомендации менеджерам (действие, срок, обоснование)
Notification Agent	Агент уведомлений	Доставляет сообщения пользователям (корпоративный мессенджер / CRM), отслеживает статус реакции
Orchestrator	Оркестратор (LLM-слой)	Управляет агентами, приоритизацией задач, SLA, маршрутизацией и эскалациями

Контур 2 -- Knowledge AI (база знаний)

Агент	Наименование	Назначение
Ingest Agent	Агент загрузки знаний	Принимает документы, структурирует их, создает/обновляет сущности базы знаний и инициирует индексацию
Query Agent	Агент поиска знаний	Выполняет поиск (семантический + полнотекстовый) и формирует ответы с ссылками на источники
Lint Agent	Агент аудита знаний	Анализирует базу знаний, выявляет противоречия, устаревшие данные и логические разрывы

1.5 Ключевые принципы

- Grace CRM является основным источником операционных данных (source of truth)
- AI-система выполняет advisory-функцию (рекомендации, контроль)
- Все изменения в CRM происходят только через API
- Архитектура построена на слабой связанности (loosely coupled services)
- Используется событийная модель (webhooks + polling)
- Система разворачивается on-premise

Дополнительно:

- Система предоставляет **собственные API-интерфейсы**, позволяющие в будущем:
 - интеграцию с внешними системами (ERP, 1C, BI и др.)
 - доступ к данным OpenSearch (статистике) и базе знаний
 - получение AI-сигналов и рекомендаций
- Архитектура допускает расширение через внешний API-layer (integration gateway)

Требования к LLM-слою

- Оркестратор (LLM) должен поддерживать:
 - подключение различных провайдеров (OpenAI, GigaChat, YandexGPT и др.) или использование собственной (on-premise) LLM
 - возможность замены модели без изменения бизнес-логики
- Конфигурация должна включать:
 - выбор модели
 - параметры генерации (temperature, max tokens и др.)

- routing запросов (multi-model strategy)
-

Коммуникационный слой

- Уведомления отправляются через:
 - **корпоративный мессенджер (например: Telegram / Mattermost)**
 - интерфейс CRM
 - Конкретная платформа мессенджера выбирается на этапе внедрения
 - Архитектура предусматривает абстракцию над каналом доставки (adapter pattern)
-

2. Архитектура решения

2.1 Общая архитектурная модель

Система построена по принципу **мультиагентной архитектуры с централизованной оркестрацией**:

- Оркестратор (LLM) управляет логикой
 - Агенты выполняют специализированные функции
 - Данные хранятся в нескольких специализированных слоях
 - Интеграция осуществляется через API и webhooks
-

2.2 Основные компоненты

Система использует многослойную модель хранения данных, где каждый слой оптимизирован под свой класс задач.

Источники данных

- Grace CRM (основная система)

- Внешние документы (для базы знаний)

Слой данных

- PostgreSQL -- операционные данные
- OpenSearch -- аналитика
- RAGflow -- база знаний и семантический поиск

AI-слой

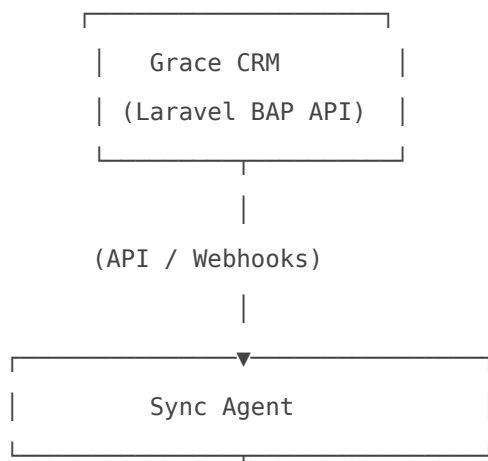
- Orchestrator (LLM)
- Набор агентов (Sales + Knowledge)

Интеграционный слой

- REST API (CRM ↔ AI)
 - Webhooks (AI ↔ CRM)
 - Telegram Bot API/Messenger API
-

2.3 Схема взаимодействия систем

Логическая схема взаимодействия:



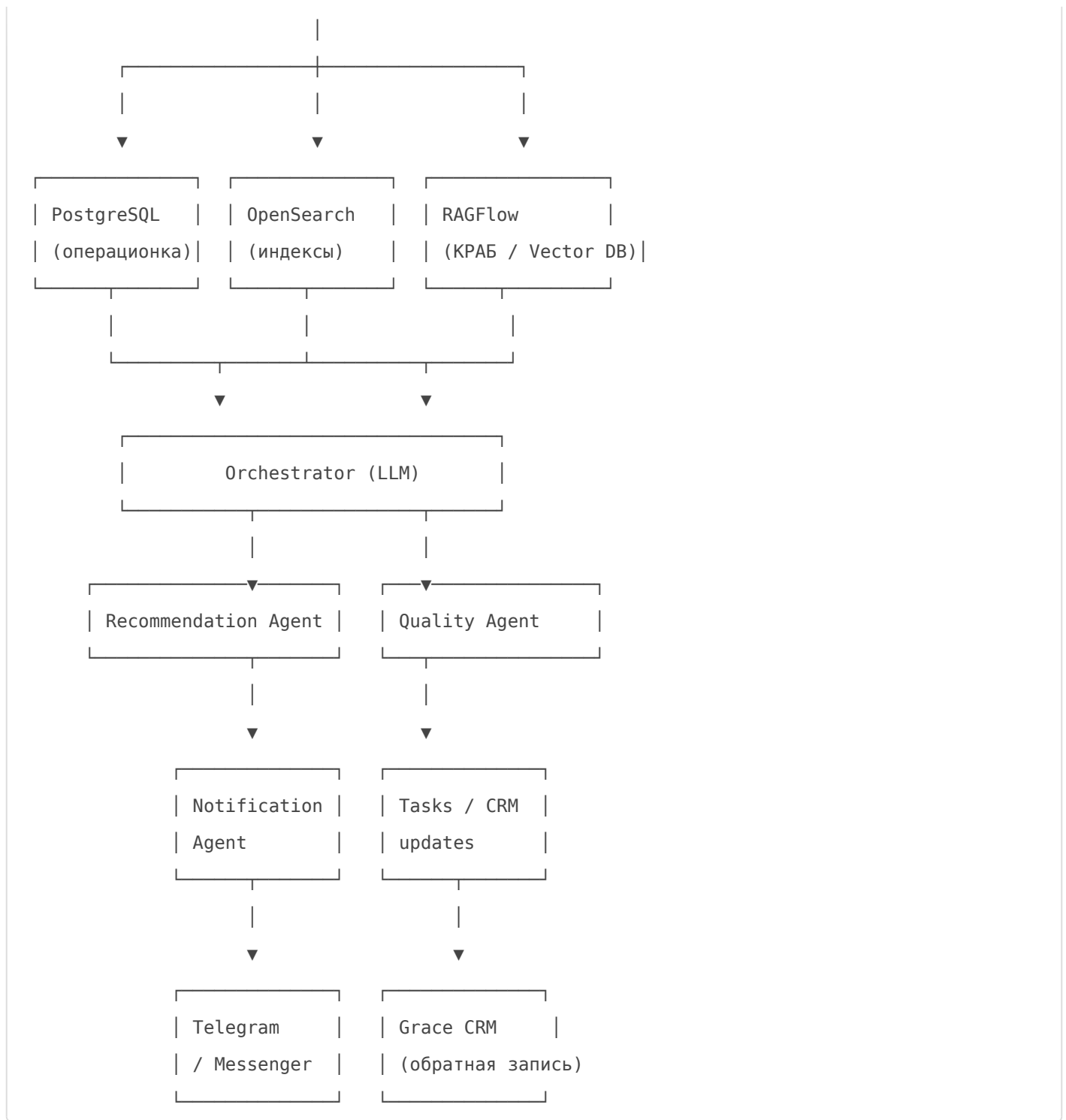


Схема взаимодействия систем (Mermaid)**

❗ (диаграмма `tekhnicheskoy-opisanie-arkhitektury-produkta.mermaid` отсутствовала в исходнике Gramax — восстановить)

2.4 Потоки данных (основной сценарий)

1. Синхронизация

- Sync Agent получает данные из CRM
- Обновляет PostgreSQL
- Обновляет индексы OpenSearch (webhook + polling)

2. Анализ

- Quality Agent проверяет данные
- Recommendation Agent формирует рекомендации
- Используются:
 - OpenSearch (история)
 - Vector DB (знания)
 - PostgreSQL (операционка)

3. Оркестрация

- Orchestrator:
 - управляет приоритетами
 - контролирует SLA
 - инициирует эскалации

4. Доставка

- Notification Agent:
 - отправляет сообщения (Telegram / CRM)
 - фиксирует статус

5. Обратная запись

- AI записывает:
 - рекомендации

- задачи
 - активности в CRM
-

2.5 Архитектурные особенности

- Событийная модель:
 - primary: webhooks
 - secondary: polling (5–10 мин)
 - fallback: reindex
 - Разделение хранения:
 - OLTP -> PostgreSQL
 - Search -> OpenSearch
 - Semantic -> Vector DB (RAGFlow)
 - Масштабируемость:
 - агенты независимы
 - можно добавлять новые роли
-

3. Описание компонентов решения

--

3.1 PostgreSQL -- операционный слой (OLTP)

PostgreSQL используется как **основное операционное хранилище AI-системы**, обеспечивающее консистентность и управление внутренним состоянием.

Назначение:

- хранение служебных данных системы
- фиксация состояния агентов

- хранение очередей задач и событий
- управление lifecycle рекомендаций

Типы данных:

- статус обработки сущностей (sync state, offsets)
- задачи и сигналы (quality issues, SLA events)
- рекомендации (черновики, статусы, feedback)
- логи работы агентов
- настройки системы и пользователей

Почему PostgreSQL:

- транзакционность (ACID) -- критично для согласованности
- удобство работы с реляционными структурами
- поддержка сложных запросов и join-логики
- надежность для хранения состояния системы

Роль в архитектуре:

→ System of record для AI-слоя (но не для бизнес-данных CRM)

3.2 OpenSearch -- поисковый и аналитический слой

OpenSearch используется как **унифицированный слой быстрого доступа к данным CRM и аналитики**.

Назначение:

- полнотекстовый поиск по CRM-данным
- агрегации и аналитика
- построение дашбордов
- быстрый доступ к денормализованным данным

Типы данных:

- сделки (projects)
- активности (activities)
- клиенты (clients)
- рекомендации (индекс для аналитики)

Особенности:

- данные поступают из CRM через Sync Agent
- хранятся в виде индексов (denormalized views)
- обновляются через:
 - webhooks (primary)
 - polling (fallback)
 - reindex (recovery)

Почему OpenSearch:

- высокая скорость поиска и агрегаций
- масштабируемость
- нативная поддержка аналитики (Dashboards)
- возможность построения BI без отдельного DWH

Роль в архитектуре:

→ Search & Analytics Layer (read-heavy слой)

3.3 RAGFlow -- база знаний (Knowledge Layer)

Назначение:

- хранение корпоративных знаний
- семантический поиск
- генерация контекстных ответов

Дополнительно:

- поддержка RAG
- расширение графовой моделью (knowledge graph)

→ Semantic Knowledge Layer

3.4 Итоговое разделение ответственности слоя данных

Слой	Система	Роль
Операционный	PostgreSQL	Состояние системы и агентов
Поисковый	OpenSearch	Быстрый доступ и аналитика CRM
Семантический	RAGFlow	Знания и контекст для AI

4. База знаний (компактное описание)

4.1 Какие документы входят в базу знаний

База знаний -- это:

структурированная модель опыта продаж компании + отраслевой контекст, а не просто хранилище документов.

База знаний должна содержать строго структурированные типы контента:

1. **Продуктовые материалы** Описания продуктов, УТП, ограничения, сценарии применения

2. **Кейсы продаж (ключевой блок)** Реальные сделки: контекст, действия менеджера, результат
 3. **Скрипты и best practices** Шаблоны коммуникаций, стратегии ведения сделки
 4. **Возражения и ответы** Типовые возражения клиентов и эффективные способы их обработки
 5. **Отраслевые материалы** Боли, особенности и контекст различных отраслей
 6. **Конкурентный анализ** Сравнение с конкурентами, аргументация
 7. **Регламенты и процессы** Правила работы, этапы воронки, SLA
 8. **Ошибки и анти-паттерны** Причины провалов сделок и нежелательные сценарии
 9. **FAQ / быстрые ответы** Короткие стандартизированные формулировки
-

4.2 Что ищут менеджеры

Поисковые сценарии делятся на 4 типа:

Ситуационные

- что делать на текущем этапе сделки
- следующий шаг / действие

Контекстные

- как работать с конкретной отраслью или типом клиента

Поведенческие

- как обработать возражение
- как ускорить или закрыть сделку

Тактические

- примеры писем / сообщений
 - формулировки и аргументы
-

4.3 Структура базы знаний

1. Иерархия

- Блок (например: Кейсы, Продукты)
 - Подблок (например: Отрасль / Тип клиента)
 - Документ (конкретная страница)
-

2. Метаданные (обязательные)

Каждый документ должен содержать:

- отрасль
 - тип клиента
 - стадия сделки
 - продукт
 - теги (возражения, сценарии и др.)
-

3. Граф связей (knowledge graph)

В рамках данной архитектуры база знаний дополнительно **структурируется с использованием графовой модели (граф БД / knowledge graph)** для этого используется логическая графовая модель внутри RAGFlow:

- связи между документами
- связи между сущностями (клиенты, отрасли, кейсы)
- контекстные зависимости

Пример связей между сущностями:

- кейсы ↔ продукты
 - кейсы ↔ отрасли
 - возражения ↔ ответы
 - клиенты ↔ поведенческие паттерны
-
-

4.4 Как это использует AI

Recommendation Agent:

- ищет похожие кейсы
- извлекает успешные паттерны
- формирует действие

Quality Agent:

- проверяет наличие обязательных шагов

Orchestrator:

- учитывает контекст + знания
-

ТЗ: Разработка агента синхронизации данных (Sync Agent)

ТЗ: Разработка агента синхронизации данных (Sync Agent)

Источник файл Ожерельев В.А. ТЗ: Разработка агента синхронизации.md

Дата: 03.05.2026

1. Назначение

Агент синхронизации обеспечивает загрузку и актуализацию данных из Grace CRM в:

- PostgreSQL (операционный слой)
- OpenSearch (поисковый слой)

Система должна поддерживать:

- near real-time обновление (webhooks)
 - инкрементальную синхронизацию
 - полную переиндексацию
-

2. Состав синхронизируемых сущностей

Обязательные сущности:

Сущность	Описание
Projects	Сделки
Clients	Клиенты
Objects	Объекты клиента (ключевая связь клиент -> актив)
Activities	Активности
Tasks	Задачи
Comments	Комментарии
Calculations	Расчёты / КП
Orders	Заказы
Users	Пользователи

3. API интеграция с CRM

Основные endpoints

```
GET /api/projects
GET /api/projects/{id}
GET /api/projects?updated_since=timestamp

GET /api/activities
GET /api/activities?project_id=
GET /api/activities?updated_since=timestamp

GET /api/clients
GET /api/clients/{id}

GET /api/tasks
GET /api/tasks?project_id=

GET /api/comments
GET /api/comments?entity_type=&entity_id=

GET /api/calculations
```

```
GET /api/calculations?project_id=
```

```
GET /api/orders
```

```
GET /api/orders?project_id=
```

```
GET /api/users
```

3.1 Объекты (обязательное расширение CRM)

⚠ В рамках проекта требуется наличие сущности:

Objects (объекты клиента)

Если отсутствует -- требуется добавить API:

```
GET /api/objects
```

```
GET /api/objects/{id}
```

```
GET /api/objects?client_id=
```

```
GET /api/objects?updated_since=timestamp
```

Назначение:

- связь **Client -> Object -> Project**
- понимание контекста сделки

Пример:

- клиент: застройщик
- объект: ЖК "Северный"
- проекты: сделки по этому ЖК

4. Архитектура Sync Agent

Компоненты:

- API Connector
 - Sync Scheduler
 - Webhook Listener
 - Data Transformer
 - Indexer (OpenSearch)
 - Storage Writer (PostgreSQL)
-

5. Mapping схемы OpenSearch (JSON)

5.1 crm-projects

```
{
  "mappings": {
    "properties": {
      "project_id": {"type": "keyword"},
      "client_id": {"type": "keyword"},
      "object_id": {"type": "keyword"},
      "name": {"type": "text"},
      "status": {"type": "keyword"},
      "stage": {"type": "keyword"},
      "amount": {"type": "double"},
      "manager_id": {"type": "keyword"},
      "created_at": {"type": "date"},
      "updated_at": {"type": "date"},
      "last_activity_at": {"type": "date"},
      "activities_count": {"type": "integer"}
    }
  }
}
```

5.2 crm-objects (новый индекс)

```
{
  "mappings": {
    "properties": {
      "object_id": {"type": "keyword"},
      "client_id": {"type": "keyword"},
      "name": {"type": "text"},
      "type": {"type": "keyword"},
      "status": {"type": "keyword"},
      "location": {"type": "text"},
      "created_at": {"type": "date"},
      "updated_at": {"type": "date"}
    }
  }
}
```

5.3 crm-clients

```
{
  "mappings": {
    "properties": {
      "client_id": {"type": "keyword"},
      "name": {"type": "text"},
      "industry": {"type": "keyword"},
      "segment": {"type": "keyword"},
      "created_at": {"type": "date"},
      "updated_at": {"type": "date"}
    }
  }
}
```

5.4 crm-activities

```
{
  "mappings": {
    "properties": {
      "activity_id": {"type": "keyword"},
      "project_id": {"type": "keyword"},
      "type": {"type": "keyword"},
      "description": {"type": "text"},
      "result": {"type": "text"},
      "created_at": {"type": "date"}
    }
  }
}
```

5.5 crm-tasks

```
{
  "mappings": {
    "properties": {
      "task_id": {"type": "keyword"},
      "project_id": {"type": "keyword"},
      "assignee_id": {"type": "keyword"},
      "status": {"type": "keyword"},
      "due_date": {"type": "date"}
    }
  }
}
```

5.6 crm-comments

```
{
  "mappings": {
```

```
"properties": {
  "comment_id": {"type": "keyword"},
  "entity_type": {"type": "keyword"},
  "entity_id": {"type": "keyword"},
  "text": {"type": "text"},
  "author_id": {"type": "keyword"},
  "created_at": {"type": "date"}
}
}
```

```
{
  "mappings": {
    "properties": {
      "calculation_id": {"type": "keyword"},
      "project_id": {"type": "keyword"},
      "amount": {"type": "double"},
      "version": {"type": "integer"},
      "created_at": {"type": "date"}
    }
  }
}
```

5.8 crm-orders

```
{
  "mappings": {
    "properties": {
      "order_id": {"type": "keyword"},
      "project_id": {"type": "keyword"},
      "amount": {"type": "double"},
      "status": {"type": "keyword"},
      "created_at": {"type": "date"}
    }
  }
}
```

6. Распределение данных между PostgreSQL и OpenSearch

Данный раздел определяет, **какие данные и в каком виде сохраняются в:**

- PostgreSQL (операционный слой)
- OpenSearch (витрины / search layer)

Ключевой принцип:

- PostgreSQL -> нормализованные данные и состояние системы
 - OpenSearch -> денормализованные витрины для поиска и аналитики
-

6.0 Диаграмма последовательности обмена данными

```
sequenceDiagram
    participant CRM
    participant Webhook
    participant SyncAgent
    participant PostgreSQL
    participant OpenSearch

    CRM->>Webhook: project-updated
    Webhook->>SyncAgent: событие

    SyncAgent->>CRM: GET /projects/{id}
    CRM-->>SyncAgent: project data

    SyncAgent->>SyncAgent: transform + normalize
```

SyncAgent->>PostgreSQL: upsert
SyncAgent->>OpenSearch: bulk index

Note over SyncAgent: update last_sync_timestamp

6.1 Таблица распределения данных

Сущность	PostgreSQL (что сохраняется)	OpenSearch (что индексируется)
Projects (сделки)	Полная структура проекта (raw JSON + нормализованные поля), статус синхронизации, технические поля (sync_state, timestamps)	Денормализованная витрина: project_id, client_id, object_id, стадия, статус, сумма, менеджер, last_activity_at, activities_count
Clients (клиенты)	Полная карточка клиента, связи, служебные поля	Упрощённая витрина: client_id, название, отрасль, сегмент
Objects (объекты)	Полная модель объекта, связь с клиентом	Витрина: object_id, client_id, тип, статус, локация
Activities (активности)	Полные записи активностей (raw), связь с проектом	Индекс коммуникаций: тип, текст, результат, дата
Tasks (задачи)	Полная структура задач, статусы, SLA	Витрина: task_id, project_id, исполнитель, статус, due_date
Comments (комментарии)	Полные тексты, связь с entity	Индекс текстов для поиска: entity_type, entity_id, текст
Calculations (расчёты)	Полные расчёты, версии, параметры	Витрина: сумма, версия, привязка к проекту
Orders (заказы)	Полная структура заказов	Витрина: статус, сумма, дата
Users (пользователи)	Полная модель пользователей и ролей	Витрина: user_id, роль, команда
Sync metadata	offset, updated_since, last_sync_time, retry state	❑ не индексируется
Ошибки / логи sync	Полный лог операций	❑ не индексируется

6.2 Принципы формирования витрин OpenSearch

1. Денормализация

В OpenSearch данные агрегируются:

Пример:

- project + client + object -> единый документ
 - activities -> агрегаты (count, last_activity_date)
-

2. Обогащение

При индексации добавляются:

- derived fields
- агрегаты
- вычисленные признаки

Пример:

- activities_count
 - days_without_activity
 - is_overdue
-

3. Оптимизация под сценарии поиска

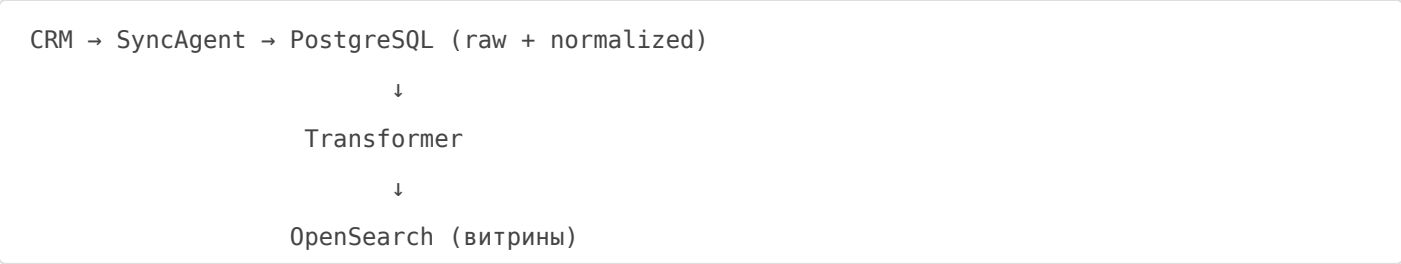
OpenSearch индекс строится под:

- фильтрацию
- агрегации
- полнотекстовый поиск

НЕ под:

- транзакции
- сложные связи

6.3 Поток записи данных



6.4 Ключевые различия слоёв

Критерий	PostgreSQL	OpenSearch
Тип данных	нормализованные	денормализованные
Назначение	состояние системы	поиск и аналитика
Обновление	строгое (upsert)	bulk indexing
Источник истины	да (для AI слоя)	нет
Использование	Sync, Orchestrator	Recommendation, Analytics

6.5 Важное архитектурное требование

- OpenSearch **не должен использоваться как источник истины**
- Все изменения идут через:
 - CRM -> SyncAgent -> PostgreSQL -> OpenSearch
- Разделение позволяет:
- избежать перегрузки PostgreSQL аналитикой
- обеспечить быстрый поиск и рекомендации
- гарантировать консистентность системы

7. Модель синхронизации: когда, что и как обновляется

Синхронизация реализуется через **3 параллельных механизма**, каждый из которых решает свою задачу:

- Webhooks -> реакция в реальном времени
 - Polling -> гарантированная консистентность
 - Full Sync -> восстановление и выравнивание
-

7.1 Webhooks (near real-time синхронизация)

Назначение

Мгновенная реакция на изменения в CRM.

Когда срабатывает

При событиях в CRM:

- создание проекта
- обновление проекта
- создание активности
- (опционально) изменение задачи

Поток

CRM → webhook → SyncAgent → точечный fetch → update

Какие данные синхронизируются

Сущность	Что делаем
Projects	загружаем 1 проект по ID
Activities	загружаем активность или проект целиком
Tasks	загружаем задачу
Comments	при наличии webhook
Calculations	при изменении проекта

Особенность

Webhook **не доверяем полностью** -> всегда делаем **дочитывание через API**

SLA

- задержка: **1-5 секунд**
 - режим: near real-time
-

7.2 Polling (инкрементальная синхронизация)

Назначение

Гарантия, что:

- ничего не потерялось
- данные актуальны

Механика

Используется:

GET /api/*?updated_since=timestamp

Частота по сущностям

Сущность	Частота	Причина
Projects	каждые 5 мин	ключевая сущность
Activities	каждые 5 мин	динамика общения
Tasks	5-10 мин	SLA
Comments	5-10 мин	контекст
Objects	10-15 мин	реже меняются
Clients	15-30 мин	редко меняются
Calculations	10-15 мин	средняя динамика
Orders	10-15 мин	финансовые события
Users	1 раз/час	почти статично

Поток

Scheduler → SyncAgent → API (bulk) → batch processing → upsert → index

Особенности реализации

- используется `updated_at`
- хранится `last_sync_timestamp`
- обрабатываются батчи (100-1000 записей)

SLA

- задержка актуализации: **до 5-10 минут**

7.3 Full Sync (полная синхронизация)

Назначение

- восстановление консистентности
- устранение ошибок sync
- перерасчёт витрин

Когда выполняется

- 1 раз в сутки (ночью)
- вручную (по триггеру)

Что синхронизируется

Сущность	Поведение
Все	полная выгрузка
Projects	пересборка агрегатов
Activities	полная история
Objects	восстановление связей
Clients	обновление справочника

Поток

Full load → overwrite / reindex → rebuild OpenSearch

SLA

- допускается длительность: **до нескольких часов**
- выполняется вне рабочего времени

7.4 Приоритет механизмов

Механизм	Приоритет	Роль
Webhooks	высокий	скорость
Polling	средний	надёжность
Full Sync	низкий	восстановление

7.5 Конфликты и консистентность

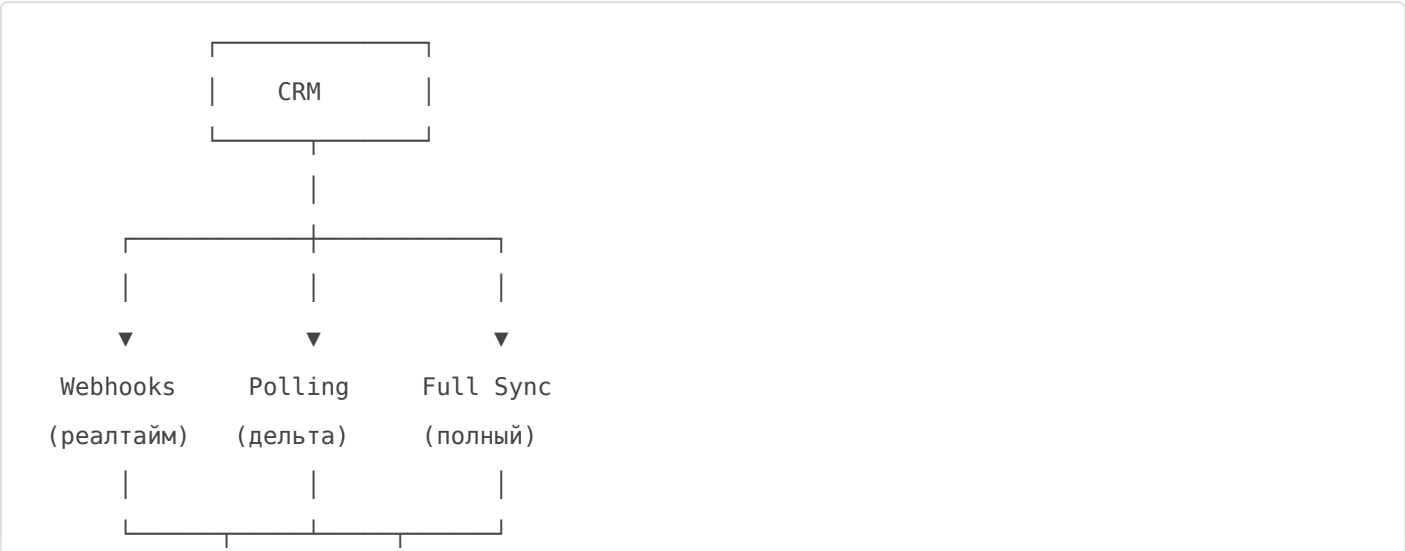
Правило:

последнее изменение по updated_at побеждает

Дополнительно:

- idempotent операции
- защита от дублей
- retry (3+ попытки)

7.6 Сводная схема





7.7 Ключевой принцип (важно для разработки)

Нельзя полагаться только на один механизм:

- Webhooks -> могут теряться
- Polling -> не realtime
- Full Sync -> тяжёлый

→ Только вместе они дают:

- **актуальность**
- **надёжность**
- **консистентность**

Итог

Тип данных	Как обновляется
Критичные (проекты, активности)	webhook + polling
Средние (задачи, расчёты)	polling + webhook (если есть)
Справочники (клиенты, users)	редкий polling
Связи (objects)	polling + full sync

8. Требования к реализации

Обязательные:

- idempotency (повторяемость без дубликатов)
- поддержка bulk операций
- retry (3 попытки минимум)
- логирование ошибок
- контроль `updated_at`

Производительность:

- batch загрузка (100–1000 записей)
 - bulk indexing OpenSearch
-

9. Ключевые особенности

- поддержка связи:
 - Client -> Object -> Project
 - денормализация данных в OpenSearch
 - готовность к аналитике и AI
-

10. Результат

После реализации:

- все CRM-данные доступны в OpenSearch
 - данные связаны через Object layer
 - AI-агенты получают полноценный контекст
-