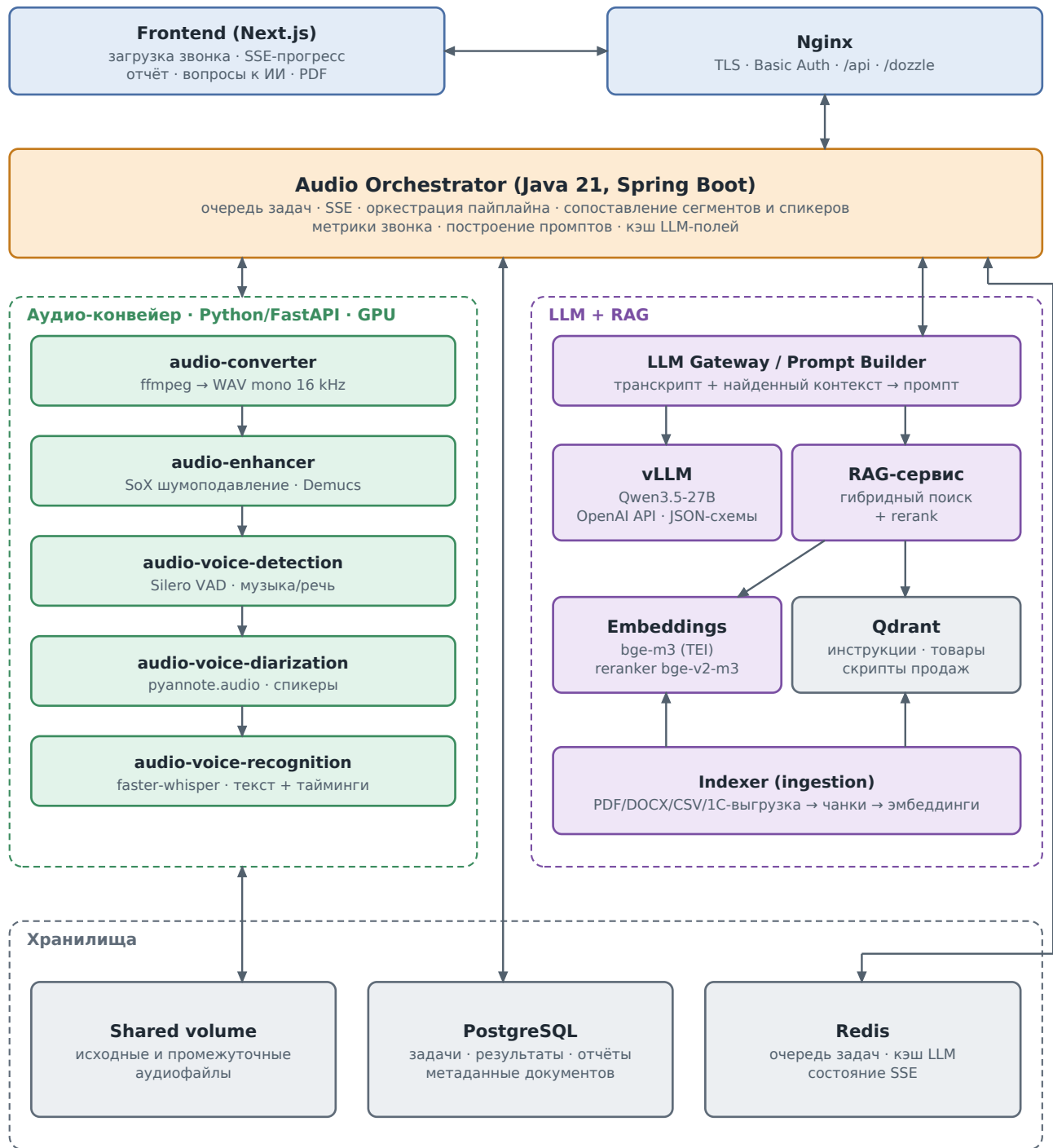


AI Calls — архитектура системы

Анализ телефонных звонков менеджеров: распознавание речи, разметка спикеров, LLM-оценка разговора, сверка со скриптом продаж и поиск упомянутых товаров. LLM работает локально на vLLM, контекст для неё даёт RAG по базе знаний: инструкции оператора, каталог товаров и скрипты продаж.



ядро аудио-ML LLM / RAG данные

1. Компоненты

Компонент	Технологии	Назначение
Frontend	Next.js, TypeScript	Загрузка аудио, прогресс по SSE, отчёт по звонку, вопросы к ИИ, экспорт в PDF.
Nginx	nginx	Единая точка входа: TLS, Basic Auth, проксирование /api и /dozzle.
Audio Orchestrator	Java 21, Spring Boot, WebClient	Очередь задач, вызов аудио-сервисов, сборка транскрипта по спикерам, метрики, генерация полей отчёта через LLM.
audio-converter	FastAPI, ffmpeg	Любой аудио/видеоформат → WAV mono 16 kHz.
audio-enhancer	FastAPI, SoX, Demucs	Шумоподавление, эквалайзер речи, отделение голоса от фона.
audio-voice-detection	FastAPI, Silero VAD	Поиск речевых участков, удаление тишины и музыки (IVR, ожидание).
audio-voice-diarization	FastAPI, pyannote.audio	Кто и когда говорит: сегменты по спикерам.
audio-voice-recognition	FastAPI, faster-whisper	Транскрипция с таймкодами, определение языка.
vLLM	vLLM, Qwen3.5-72B	LLM-сервер с OpenAI-совместимым API: генерация полей отчёта, ответы на вопросы, оценка по скрипту.
RAG-сервис	Python/FastAPI	Гибридный поиск (dense + sparse) по базе знаний, реранкинг, сборка контекста.
Embeddings	TEI: bge-m3, bge-reranker-v2-m3	Векторизация чанков и запросов, переранжирование кандидатов.
Qdrant	Qdrant	Векторная контекстная БД: инструкции, каталог товаров, скрипты.
Indexer	Python, воркер	Загрузка и обновление документов: парсинг, чанкинг, эмбединги, запись в Qdrant.
PostgreSQL / Redis	PostgreSQL 16, Redis 7	Задачи, результаты и отчёты; очередь задач и кэш LLM-ответов.
Dozzle	Dozzle	Просмотр логов контейнеров.

2. Обработка звонка

1. Пользователь загружает файл через Frontend → Nginx → Orchestrator. Задача ставится в очередь (Redis), прогресс передаётся по SSE.
2. Аудио-конвейер: конвертация → улучшение → VAD → диаризация → распознавание. Файлы передаются через shared volume, по HTTP идут только пути.
3. Orchestrator сопоставляет слова с сегментами спикеров, определяет роли (оператор/клиент) и имена, считает метрики: доли речи, паузы, перебивания.
4. LLM-этап: Prompt Builder берёт транскрипт и добавляет контекст из RAG (регламент, карточки товаров, эталонный скрипт), затем обращается к vLLM.
5. Результат (резюме, тип звонка, ошибки, рекомендации, соответствие скрипту, товары) пишется в PostgreSQL и отдаётся во Frontend.

3. LLM на vLLM

- Orchestrator обращается к vLLM через OpenAI-совместимый API (**/v1/chat/completions**), клиент — Spring AI.
- Почему vLLM: continuous batching и PagedAttention позволяют генерировать ~15 полей отчёта параллельно, а не по очереди, с кратно большей пропускной способностью.
- Structured outputs (guided JSON): поля отчёта и оценка по скрипту возвращаются строго по JSON-схеме, без разбора свободного текста.

- Модель: Qwen3.5-27B или старше. FP8 на отдельной GPU 48 ГБ, AWQ/INT4 укладывается в 24-32 ГБ. Контекст 32k+, prefix caching для общего системного промпта.
- Данные звонков не покидают контур: внешние LLM-API не используются.

4. RAG и контекстная база знаний

В Qdrant три коллекции. Каждая хранит гибридные векторы bge-m3 (dense + sparse) и payload для фильтрации:

Коллекция	Содержимое	Как используется
operator_instructions	Регламенты, работа с возражениями, FAQ, запрещённые формулировки	Ответы на вопросы к ИИ и рекомендации со ссылкой на пункт регламента; проверка нарушений.
products	Каталог: SKU, название, синонимы, характеристики, цена, наличие (payload)	Быстрый поиск упомянутых товаров: LLM извлекает упоминания, гибридный поиск + фильтры находит карточки, проверяются названные цены и характеристики.
sales_scripts	Скрипты продаж, разбитые на этапы: приветствие, выявление потребности, презентация, возражения, закрытие	Сверка разговора со скриптом: этапы транскрипта сопоставляются с эталоном, LLM оценивает каждый этап; итог — % соответствия и пропущенные шаги.

- Поиск: запрос → эмбединг → top-k из Qdrant (dense + sparse, RRF) → rerank → 3-8 чанков в промпт с указанием источника.
- Indexer: документы загружаются через API или админку, каталог синхронизируется по расписанию (CSV/1C). Версии документов и метаданные хранятся в PostgreSQL, при изменении файла переиндексируется только он.
- Скрипты и инструкции привязаны к отделу или продукту (payload **department, script_id**), поэтому звонок сверяется со «своим» скриптом.

5. Развёртывание

- Всё запускается через docker-compose в одной сети; наружу открыт только Nginx.
- GPU: vLLM получает отдельную карту, аудио-сервисы делят вторую. Embeddings/reranker работают на CPU или на той же карте, что и аудио.
- Модели (Whisper, ruannote, LLM, bge) лежат в общем кэше-томе и скачиваются один раз.
- Наблюдаемость: trace-id сквозь все сервисы, логи в Dozzle, health-check у каждого контейнера.