HR2137 Bot — Технический FAQ
Руководство для пользователей

Проект: AI-ассистент для HR консалтинга
Анастасия Новосёлова

Версия документа: 1.0 Дата: 16 декабря 2025 г.

О чём этот документ

Назначение: FAQ для понимания архитектуры, компонентов и работы AI-бота HR2137.
Аудитория: технические специалисты, разработчики, администраторы системы.
Уровень: от базового до среднего уровня знаний Python и AI.

Содержание

1 Общие вопросы о проекте

1.1 Что такое HR2137 Bot?

Краткий ответ: HR2137 Bot — это интеллектуальный AI-ассистент для консалтинговой практики Анастасии Новосёловой, работающий в Telegram.

Основные функции бота:

- Автоматизация взаимодействия с клиентами через Telegram
- Ответы на вопросы о HR-консалтинге на основе базы знаний
- Поиск релевантной информации с помощью RAG-системы
- Генерация коммерческих предложений с учётом контекста
- Обработка и классификация заявок на услуги
- Интеграция с внешними системами (WEEEK, Google Sheets)

1.2 Какие источники лидов поддерживаются?

Система интегрирована с тремя основными источниками входящих заявок:

1. **HR Time API** — асинхронный polling для получения заявок из системы HR Time
2. **Yandex Email** — обработка входящих писем через протоколы IMAP/SMTP
3. **Сайт-визитка** — приём заявок с веб-формы через webhook

1.3 Какие задачи решает бот?

Задача	Описание
Обработка лидов	Автоматический приём заявок из разных каналов
Классификация запросов	Определение типа запроса (запись, вопрос, жалоба)
RAG-ответы	Поиск информации в базе знаний и генерация точных ответов
Коммерческие предложения	Генерация персонализированных КП на основе контекста
Управление проектами	Создание задач и проектов в WEEEK API
Аналитика	Сбор метрик и статистики по взаимодействию с клиентами

2 RAG-система и база знаний

2.1 Что такое RAG и зачем он нужен?

RAG (Retrieval-Augmented Generation) — это архитектура, которая объединяет поиск информации в базе знаний с генерацией ответов через языковую модель.

Преимущества RAG перед обычными чат-ботами:

- **Фактическая точность** — ответы основаны на реальных документах компании

- **Минимизация галлюцинаций** — модель не выдумывает информацию
- **Актуальность данных** — база знаний обновляется без переобучения модели
- **Прозрачность** — можно указать источники информации в ответах
- **Контекстуальность** — понимание вопросов с учётом семантики

2.2 Как работает поиск в базе знаний?

Процесс поиска состоит из четырёх этапов:

1. **Эмбеддинг запроса**
   Вопрос пользователя преобразуется в числовой вектор (эмбеддинг) через модель `Qwen2.5-1.5B-Instruct`.
2. **Векторный поиск**
   Qdrant ищет документы с похожими векторами по смысловому сходству (cosine similarity).
3. **Формирование контекста**
   Найденные фрагменты текста передаются в LLM как контекст для генерации ответа.
4. **Генерация ответа**
   Модель (DeepSeek или GigaChat) формирует ответ на основе контекста из базы знаний.

Важное отличие

Векторный поиск понимает **смысл** запроса, а не только ключевые слова. Например, запрос *«найм сотрудников»* найдёт документы про *«подбор персонала»*, хотя слова разные.

2.3 Что такое эмбеддинги (embeddings)?

Эмбеддинги — это числовые векторы, которые представляют текст в виде набора чисел в многомерном пространстве смыслов.

Пример преобразования:

`Текст:` “HR консалтинг”

↓ *(модель Qwen2.5)*

`Вектор:` [0.234, -0.123, 0.567, ..., 0.891]
(1536 чисел)

Свойства эмбеддингов:

- Похожие по смыслу фразы имеют похожие векторы
- Расстояние между векторами отражает семантическую близость
- Позволяют находить релевантную информацию даже при разной формулировке

2.4 Почему используется Qdrant?

Qdrant — это специализированная векторная база данных для работы с эмбеддингами.

Характеристика	**Описание**
Высокоразмерный поиск	Эффективный поиск среди миллионов векторов
Масштабируемость	Горизонтальное масштабирование для больших данных
Скорость	Быстрый поиск похожих векторов с использованием HNSW индексов
Open Source	Бесплатная версия для self-hosting
Фильтрация	Поддержка метаданных и сложных фильтров при поиске

2.5 Какие документы можно загружать?

Система поддерживает различные форматы документов:

- **PDF документы** — через скрипт `load_pdf.py`
- **Excel файлы** — прайс-листы через `load_pricelist.py`
- **Word документы** — инструкции, регламенты
- **Веб-страницы** — индексация через `scraper.py` из whitelist
- **Яндекс Диск** — массовая индексация через `index_yandex_disk.py`

3 LLM модели и генерация ответов

3.1 Какие языковые модели используются?

Система работает с двумя моделями в режиме **fallback** (резервирования):

1. **DeepSeek Chat** (основная модель)
   - Доступ через OpenRouter API
   - Высокое качество генерации текста
   - Хорошее понимание контекста
2. **GigaChat** (резервная модель)
   - Российское решение от Сбера
   - Автоматическое переключение при сбое основной модели
   - Независимый провайдер для надёжности

3.2 Как работает fallback между моделями?

Механизм резервирования

Если основная модель недоступна или даёт некачественный ответ, система автоматически переключается на резервную модель.

Алгоритм работы:

Попытка 1: DeepSeek → ошибка/низкая confidence

↓

Попытка 2: DeepSeek (повтор) → ошибка

↓

Попытка 3: GigaChat → успех

Преимущества fallback системы:

- Повышение надёжности — бот продолжает работать при сбоях
- Балансировка нагрузки между провайдерами
- Снижение зависимости от одного API
- Автоматическое восстановление без вмешательства администратора

3.3 Как бот запоминает контекст разговора?

Бот хранит в памяти **последние 6 сообщений** (12 реплик: 6 от пользователя + 6 от бота).

Пример диалога с контекстом:

Пользователь: "Сколько стоит подбор персонала?"

Бот: "От 50 000 руб. Включает поиск и проверку."

Пользователь: "А какие гарантии?"

Бот:
[понимает, что речь о подборе персонала из предыдущего вопроса]
"Гарантируем замену кандидата в течение..."

4 Архитектура и компоненты

4.1 Какие основные модули включает проект?

Модуль	Назначение
`app.py`	Главный файл Telegram бота с async handlers
`rag_chain.py`	RAG-система для поиска и генерации ответов
`qdrant_loader.py`	Загрузка документов в векторную базу данных
`llm_api.py`	Клиент для работы с LLM (DeepSeek + GigaChat fallback)
`google_sheets_helper.py`	Интеграция с Google Sheets для записей и услуг

Модуль	Назначение
`dashboard.py`	Веб-дашборд для управления RAG системой (FastAPI)
`web_interface.py`	Демо-интерфейс для презентации инвесторам
`config.yaml`	Централизованная конфигурация всей системы
`requirements.txt`	Список Python библиотек и зависимостей
`Dockerfile`	Инструкции для сборки Docker образа

4.2 Почему используется асинхронная архитектура?

Асинхронность (async/await) позволяет боту обрабатывать множество запросов одновременно без блокировки потока выполнения.

Критические преимущества для HR2137 Bot:

- **Параллельный polling** — одновременный опрос HR Time API, Email, webhook
- **Множественные пользователи** — обработка запросов от десятков пользователей параллельно
- **Долгие операции** — запросы к LLM и Qdrant не блокируют другие задачи
- **Эффективность** — меньше потребление ресурсов сервера

4.3 Как работает интеграция с WEEEK?

WEEEK API используется для создания проектов и задач при получении лидов.

Процесс обработки лида:

1. Новый лид поступает в бота (из любого источника)
2. Бот классифицирует запрос и извлекает детали (ФИО, услуга, бюджет)
3. Через WEEEK API создаётся:
   - **Проект** для клиента с заполненными полями
   - **Задачи** по обработке запроса (звонок, отправка КП и т.д.)
4. Команда получает уведомление в WEEEK для начала работы

4.4 Что такое веб-дашборд и зачем он нужен?

Dashboard (`dashboard.py`) — это FastAPI веб-интерфейс для администраторов.

Основные функции дашборда:

- **Загрузка документов** — добавление PDF, Excel, Word файлов в базу знаний
- **Статистика RAG** — количество документов, векторов, метрики поиска
- **Тестирование запросов** — проверка качества ответов RAG в реальном времени
- **Настройка параметров** — изменение min_score, top_k для поиска
- **Просмотр метрик** — Precision@K, MRR, Hallucination Rate
- **Управление документами** — удаление, обновление файлов в базе

Доступ: `http://localhost:8000` (локально) или через Railway deployment URL.

5 Развертывание и настройка

5.1 Какие переменные окружения необходимы?

Обязательные переменные:

```
TELEGRAM_TOKEN=           #            Telegram          (
   @BotFather)
OPENROUTER_API_KEY=       # API          OpenRouter        DeepSeek
QDRANT_URL=               # URL Qdrant (http://localhost:6333)
QDRANT_API_KEY=           # API                  Qdrant Cloud (prod)
```

Опциональные переменные:

```
GOOGLE_SHEETS_CREDENTIALS_PATH=  # credentials.json
GOOGLE_SHEETS_SPREADSHEET_ID=    # ID                Google Sheets
GIGACHAT_API_KEY=                # API          GigaChat (fallback)
USE_WEBHOOK=true                 # Webhook            (       Railway
   )
WEB_INTERFACE_PORT=8000          #               -
```

5.2 Как запустить проект локально?

Шаг 1: Установить зависимости

```
pip install -r requirements.txt
```

Шаг 2: Запустить Qdrant

```
docker run -p 6333:6333 qdrant/qdrant
```

Шаг 3: Создать файл .env

```
TELEGRAM_TOKEN=your_token_here
OPENROUTER_API_KEY=your_key_here
QDRANT_URL=http://localhost:6333
```

Шаг 4: Загрузить документы в базу

```
python load_pdf.py path/to/documents.pdf
```

Шаг 5: Запустить бота

```
python app.py
```

Шаг 6 (опционально): Запустить веб-интерфейс

```
python web_interface.py
```

5.3 Как развернуть на Railway?

1. **Подключить GitHub репозиторий** к Railway Dashboard
2. **Добавить переменные окружения** в Railway → Settings → Variables
3. **Установить** `USE_WEBHOOK=true` для webhook режима
4. Railway автоматически соберёт Docker образ и развернёт приложение
5. Получите публичный URL для webhook Telegram бота

6 Управление базой знаний

6.1 Как добавить новые документы в RAG?

Способ 1: Через Dashboard (рекомендуется)

1. Откройте `http://localhost:8000`
2. Перейдите в раздел **«Upload»**
3. Выберите файл (PDF/Excel/Word/TXT)
4. Система автоматически разобьёт на chunks и загрузит в Qdrant

Способ 2: Через CLI скрипты

```
#                       PDF
python load_pdf.py document.pdf

#
python index_yandex_disk.py --local-path /path/to/files

#                                       whitelist
python scraper.py
```

Способ 3: Добавить URL в whitelist

```
# config.yaml
whitelist:
  - https://your-company.com
  - file://media/documents/
```

Затем запустите: `python scraper.py`

6.2 Как удалить или обновить документ?

Через Dashboard → Manage Documents:

- Просмотр всех документов в базе с метаданными
- Удаление по ID документа или по фильтрам
- Обновление: удалите старый документ → загрузите новый

Через API:

```
from qdrant_client import QdrantClient

client = QdrantClient(url="http://localhost:6333")
client.delete(
    collection_name="hr_knowledge_base",
    points_selector={"document_id": "doc_123"}
)
```

6.3 Как настроить параметры поиска?

Основные параметры в `config.yaml`:

```
rag:
  min_score: 0.3           #
                                  (0-1)
  top_k: 5                 #

  chunk_size: 512          #
                   (              )
  chunk_overlap: 50        #
```

Рекомендации

- `min_score < 0.3` → больше результатов, но ниже точность
- `top_k > 5` → больше контекста, но дороже запросы к LLM
- `chunk_size < 512` → точнее поиск, но больше фрагментов в базе

7 Мониторинг и качество

7.1 Как проверить качество RAG-системы?

Использование RAG Evaluator:

```
python rag_evaluator.py
```

Основные метрики качества:

Метрика	Описание
Precision@K	Точность топ-K результатов поиска (0-1)
MRR (Mean Reciprocal Rank)	Средний ранг правильного ответа (0-1)
Groundedness	Насколько ответ основан на найденных документах (0-1)
Hallucination Rate	Процент «выдуманной» информации моделью (0-100%)
Latency	Время ответа системы (миллисекунды)

Результаты доступны в **Dashboard → Metrics**.

7.2 Как понять, что RAG работает плохо?

Признаки проблем с RAG:

- Бот часто отвечает «Не знаю» на вопросы из документов
- Ответы не соответствуют данным в базе знаний
- Высокий Hallucination Rate (>20%)
- Низкий Precision@3 (<0.6)
- Медленное время ответа (>5 секунд)

Решения проблем:

1. Снизить `min_score` с 0.7 до 0.3 в config
2. Увеличить `top_k` для большего контекста (до 10)

3. Проверить качество эмбеддингов — возможно нужна другая модель
4. Улучшить chunking — уменьшить `chunk_size` для точности
5. Пересмотреть промпты для LLM модели

7.3 Где смотреть логи и ошибки?

Источники логов:

- **Логи бота:** `tail -f bot.log`
- **Логи Dashboard:** вывод uvicorn в терминале
- **Qdrant логи:** `docker logs <container_id>`
- **Railway логи:** Railway Dashboard → Deployments → View Logs

8 Troubleshooting — Решение проблем

8.1 Бот не отвечает на сообщения

Чек-лист диагностики:

1. Проверьте корректность `TELEGRAM_TOKEN` в `.env`
2. Убедитесь, что процесс бота запущен: `ps aux | grep app.py`
3. Проверьте логи на ошибки: `tail -f bot.log`
4. Проверьте интернет-соединение для Telegram API
5. Убедитесь, что webhook настроен правильно (если используется)

8.2 RAG не находит документы в базе

Диагностика проблемы:

```
#                    ,        Qdrant
curl http://localhost:6333/collections

#
curl http://localhost:6333/collections/hr_knowledge_base
```

Решения:

1. Убедитесь, что документы загружены: **Dashboard → Stats**
2. Снизьте `min_score` с 0.7 до 0.3
3. Проверьте, что эмбеддинг модель Qwen запущена локально
4. Попробуйте переиндексировать документы

8.3 LLM модель не отвечает

Проверьте следующее:

1. `OPENROUTER_API_KEY` корректен и активен
2. Есть интернет-соединение для запросов к OpenRouter

3. Баланс на OpenRouter аккаунте положительный
4. Проверьте логи на наличие ошибок от API
5. Убедитесь, что fallback на GigaChat сработал (должно быть в логах)

8.4 Веб-интерфейс не открывается

Возможные решения:

```
# , 8000
lsof -i :8000

#
uvicorn dashboard:app --host 0.0.0.0 --port 8000

#
echo $WEB_INTERFACE_PORT
```

8.5 Ошибка при загрузке документов

Частые причины:

- **Формат не поддерживается** → используйте PDF, DOCX, XLSX, TXT
- **Файл повреждён** → проверьте, открывается ли вручную
- **Нет прав на чтение** → выполните `chmod 644 file.pdf`
- **Qdrant недоступен** → проверьте `QDRANT_URL`
- **Недостаточно памяти** → освободите ресурсы сервера

9 Полезные команды бота

Команда	Описание
`/start`	Запустить бота и показать приветствие
`/help`	Показать справку по командам
`/rag_search [запрос]`	Поиск в базе знаний и генерация ответа
`/rag_stats`	Показать статистику (кол-во документов, векторов)
`/rag_docs`	Список всех документов в базе знаний
`/demo_proposal [текст]`	Генерация коммерческого предложения
`/summary [проект]`	Создать краткое саммари проекта
`/upload`	Загрузить документ в базу знаний
`/settings`	Настройки пользователя
`/feedback`	Оставить отзыв о боте

10 Дополнительные ресурсы

10.1 Документация проекта

- `README_RU.md` — общее описание проекта на русском

- `QUICKSTART.md` — быстрый старт за 5 минут
- `LOCAL_SETUP.md` — детальная настройка для разработки
- `RAG_INTEGRATION.md` — глубокое погружение в RAG систему
- `ENV_VARIABLES.md` — полный список переменных окружения
- `DEPLOYMENT.md` — инструкции по deployment (Railway, Docker)

10.2 Внешние ресурсы

- Qdrant Documentation
- OpenRouter API Docs
- GigaChat API
- Telegram Bot API
- FastAPI Documentation

Заключение

Итоги

HR2137 Bot — это современная AI-система, которая объединяет:

- **Telegram бота** для общения с клиентами
- **RAG систему** для поиска информации в базе знаний
- **Google Sheets** для управления записями и услугами
- **Веб-интерфейсы** для администрирования и демонстрации
- **Интеграции** с внешними системами (WEEEK, Email, HR Time)

Принципы построения системы:

- **Надёжность** — fallback модели и обработка ошибок
- **Масштабируемость** — асинхронная архитектура, Singleton паттерны
- **Гибкость** — конфигурация через YAML, модульная структура
- **Прозрачность** — подробные логи, метрики, мониторинг