АСТРА ИИ Цифровой офис (ASTRA AI Digital Office)

Руководство администратора

Москва, 2026

Общие положения

Настоящее руководство администратора описывает установку, настройку, администрирование и сопровождение программного комплекса **АСТРА ИИ Цифровой офис** (**Astra Digital Office)** (далее — комплекс, продукт). Документ предназначен для специалистов, отвечающих за эксплуатацию комплекса в инфраструктуре организации: системных администраторов, DevOps-инженеров и администраторов информационной безопасности.

Руководство охватывает все компоненты продукта: бэкенд-микросервисы, Data-Science (DS) часть, фронтенд-приложение, а также инфраструктурные зависимости (PostgreSQL, Redis, объектное хранилище S3/MinIO, Kafka, векторная база Qdrant, инференс-серверы моделей машинного обучения). Подробное описание архитектурных принципов и цикла обработки данных приведено в отдельном документе «Техническое описание»; настоящее руководство ориентировано на практические задачи администрирования.

Назначение документа

Документ решает следующие задачи:

- ознакомление администратора с составом и архитектурой комплекса;
- подготовка инфраструктуры к развёртыванию (требования к аппаратному и программному обеспечению, сети);
- пошаговое выполнение локального и кластерного развёртывания;
- настройка переменных окружения и шлюза nginx;
- управление пользователями, ролями, правами, проектами, файлами, чатами и специализированными модулями;
- управление подключаемыми моделями искусственного интеллекта и внешними интеграциями;
- организация мониторинга, резервного копирования и устранения типовых неполадок.

Аудитория

Основная аудитория документа — **системный администратор** комплекса. Предполагается владение следующими технологиями:

- контейнеризация Docker и оркестрация Kubernetes (Helm);
- администрирование Linux (Ubuntu/Debian или Astra Linux);
- реляционные СУБД PostgreSQL;
- объектные хранилища, совместимые с S3 (MinIO);
- брокеры сообщений Apache Kafka;
- обратные прокси-серверы nginx;
- основы HTTP/REST и JSON Web Token (JWT).

Для администрирования DS-части дополнительно полезно знакомство с векторными базами данных (Qdrant) и принципами обслуживания инференс-серверов, совместимых с OpenAI API (vLLM).

Термины и определения

Термин	Определение
Build Project (Проект сборки)	Логическая рабочая область (workspace), изолирующая данные пользователей, файлы, чаты и настройки ролей. Пользователь может входить в несколько проектов; активный проект определяется полем active_build_project_id в JWT
Source (Источник)	Файл или папка, загруженные в базу знаний и используемые ИИ для поиска (RAG). Источники с флагом cross доступны во всех чатах проекта; источники с флагом tmp привязаны к конкретному чату через ChatSourcesUsed
Chat (Чат)	Диалог пользователя с ассистентом; содержит сообщения, привязанные источники и активный шаблон фичи. Ответы могут включать текст, сгенерированный документ, HTML-дашборд или HTML-презентацию
Template (Шаблон)	Многошаговый пайплайн промптов; каждый шаг (step-template) соответствует одному промпту в цепочке
Feature (Функциональность)	Тип AI-функции, доступной пользователю: summarization (суммаризация), question_answering (вопрос-ответ), dashboard (дашборд), presentation (презентация)
УПД	Универсальный передаточный документ; специализированный модуль проводки УПД в 1С
КС-3	Акт выполненных строительных работ; устаревший (deprecated) модуль распределения
RAG	Retrieval-Augmented Generation — подход, при котором ответ языковой модели дополняется фрагментами, найденными по векторному индексу документов
MCP	Model Context Protocol — протокол публикации инструментов (tools), вызываемых агентом; реализован сервисом mcp-server
JWT	JSON Web Token — компактный токен аутентификации (HS256), выпускаемый auth-сервисом
RBAC	Role-Based Access Control — ролевая модель доступа: роли связываются с правами, права проверяются на endpoints
Session token	Долгоживущий токен сессии в httpOnly cookie, валидируемый nginx через auth_request
Access JWT	Краткосрочный JWT (по умолчанию 900 с), выпускаемый на каждый запрос и форвардимый в upstream как Authorization: Bearer
Owner-scoping	Фильтрация данных по владельцу: каждая бизнес-сущность несёт user_id и build_project_id; менеджер фильтрует по значениям из JWT
S3 / MinIO	Объектное хранилище, совместимое с Amazon S3; хранит зашифрованные блобы файлов
Qdrant	Векторная база данных; одна коллекция на user_id
vLLM	Сервер инференса LLM, совместимый с OpenAI API
Kafka	Брокер сообщений для request/reply обмена между сервисами бэкенда и api-external
api-external	Единая точка исходящего трафика (egress) во внешние системы: DS, 1C, DIT-модель
Healthcheck	Публичный endpoint /<prefix>/api/health/, возвращающий статус сервиса; используется для liveness/readiness проб

Соглашения документа

В тексте применяются следующие обозначения:

- команды оболочки приводятся в блоках кода с подсветкой bash;
- примеры значений переменных окружения — в блоках env;
- имена переменных окружения, endpoints, таблиц БД и файлов выделены моноширинным шрифтом;
- фрагменты, требующие заполнения конкретными значениями инфраструктуры заказчика, обозначены как [ЗАПОЛНИТЬ: описание];
- URL формата https://<gateway>/<service_prefix>/api/<handler_path> описывают глобальный маршрут через шлюз.

О продукте

Назначение

АСТРА ИИ Цифровой офис (**Astra Digital Office)** — корпоративный AI-ассистент, предназначенный для интеллектуальной обработки документов и автоматизации рутинных задач сотрудников организации. Продукт обеспечивает взаимодействие пользователей с корпоративной базой знаний посредством чата на основе технологии RAG, а также предоставляет инструменты генерации структурированных артефактов: суммаризаций, презентаций и дашбордов. Отдельный класс задач охватывает специализированные модули, в первую очередь автоматизированную проводку универсальных передаточных документов (УПД) в системе 1С.

Продукт ориентирован на применение в средних и крупных организациях, использующих большие объёмы неструктурированной документации: договорной, технической, бухгалтерской, отчётной. Целевая аудитория включает сотрудников, работающих с документами, а также администраторов, управляющих доступом и конфигурацией системы.

Основные возможности

Возможность	**Описание**
Чат с документами (RAG)	Пользователь загружает документы в источники знаний и задаёт вопросы; ассистент выполняет поиск релевантных фрагментов и формирует ответ с учётом контекста
Суммаризация	Генерация краткого документа (Markdown) на основе загруженного файла или набора файлов
Генерация презентаций	Создание HTML-презентаций со слайдами, навигацией и графиками (Chart.js)
Генерация дашбордов	Создание HTML-дашбордов с визуализацией данных
Индексация мультимедиа	Извлечение текста из PDF (включая сканированные через OCR), DOCX, TXT, а также расшифровка аудио и видео (ASR)
Управление источниками	Загрузка, структурирование, версионирование и мягкое удаление файлов; источники cross (все чаты) и tmp (привязанные к чату)
Шаблоны промптов	Многошаговые пайплайны промптов, привязанные к функциональностям; назначение шаблона по умолчанию
Проводка УПД в 1С	Специализированный модуль: загрузка XML-документа, создание УПД, пошаговая сверка с 1С, разрешение расхождений, обновление статусов
Ролевая модель доступа	RBAC с ролями admin/user, гранулярные права, изоляция по проектам и владельцам
Многоязычный интерфейс	Фронтенд на русском и английском (по умолчанию русский)

Состав комплекса

Комплекс состоит из трёх взаимосвязанных компонентов, каждый из которых поставляется отдельным репозиторием и набором Docker-образов.

Бэкенд

Бэкенд реализован как монорепозиторий Python-микросервисов на базе FastAPI. Включает шесть бизнес-сервисов и сервис агрегации OpenAPI-спецификации.

Сервис	URL-префикс	Назначение
auth-service	/auth	Аутентификация, сессии, выпуск JWT, RBAC
general-service	/general	Справочный API: каталог LLM, промпты, проекты сборки
fs-manager-service	/fs_manager	Файловое хранилище: PostgreSQL + S3, шифрование
secretary-service	/secretary	Чаты, сообщения, источники, фичи, шаблоны
api-external-service	/api_external	Единая точка egress: вызовы DS, 1C, DIT-модели
one-c-dit-service	/one_c_dit	Интеграция УПД/XML с 1C и DIT-моделью
openapi-aggregator	—	Агрегация OpenAPI-спецификаций; порт 8099

Data-Science часть

DS-часть реализована как отдельный репозиторий Python-сервисов (Python 3.11–3.12, FastAPI) и предоставляет агентную MCP-инструментную платформу, вызываемую бэкендом.

Сервис	Порт	Назначение
agent-server	8100	Единая точка входа DS; LLM tool-loop, генерация заголовков чатов
mcp-server	8101	Прокси tool-calls через MCP; публикует 4 инструмента
tool-api	8102	Реализации модулей: summarization, question_answering, dashboard, presentation
file-processing	8103	Препроцессинг и индексация документов (OCR, ASR, эмбеддинги, Qdrant)
Qdrant	8104	Векторная база данных (контейнер)

Фронтенд

Веб-приложение на Next.js 15 (App Router, React 19, TypeScript, MUI 7, Redux Toolkit + RTK Query), построенное по методологии Feature-Sliced Design. Поставляется как standalone-сборка Docker-образа на базе node:20-alpine, обслуживается внутренним nginx на порту 3000.

Раздел	Назначение
Platform	Чат, управление источниками и шаблонами промптов
Modules	Специализированные процессы: проводка УПД (/modules/jobs) и устаревшие модули КС-3, протоколов
Admin	Управление участниками, ролями и правами проекта (/admin)
Profile	Профиль пользователя: смена email/пароля, язык, выход (/profile)

Архитектура

Общая схема

Комплекс построен по микросервисной архитектуре. Все компоненты поставляются и выполняются в виде Docker-контейнеров, что обеспечивает воспроизводимость окружения и изоляцию зависимостей. Взаимодействие между сервисами реализовано через синхронный REST и асинхронный обмен сообщениями Kafka (паттерн request/reply).

Архитектура разделяется на три логических компонента: **бэкенд**, **DS-часть** и **фронтенд**. Подробное описание принципов проектирования, цикла обработки данных и типов обрабатываемых данных приведено в документе «Техническое описание». Ниже приведена сводная схема взаимодействия.

Диаграмма взаимодействия

Взаимодействие компонентов при типовом сценарии (пользователь отправляет сообщение в чат) представлено следующей цепочкой:

```
Фронтенд (FE)
  │ httpOnly session_token cookie
  ▼
nginx-шлюз (порт 8090) ──auth_request──► auth-service: POST /auth/api/verify_token/
  │ (получает краткосрочный JWT в X-Auth-Jwt)
  │ форвардит Authorization: Bearer <JWT>
  ▼
secretary-service: POST /secretary/api/chats/{id}/send_message
  │ Kafka request/reply (topic api_external_DSOutRequests)
  ▼
api-external-service: POST {DS_MODEL_URL}/agent/run
  │ заголовок X-User-Data-Jwt + x-api-key
  ▼
agent-server (DS, порт 8100): POST /agent/run
  │ LLM tool-loop через MCP
  ▼
mcp-server (DS, порт 8101) ──► tool-api (DS, порт 8102) ──► LLM / S3 / Postgres / Qdrant
```

Ключевые принципы взаимодействия:

- **Фронтенд не хранит токен на клиенте.** Источником истины сессии служит httpOnly cookie session_token. nginx-шлюз прозрачно обменивает его на краткосрочный JWT при каждом запросе.
- **Бэкенд не вызывает DS напрямую из бизнес-сервисов.** Все вызовы DS, 1С и DIT-модели проходят через Kafka в api-external-service — единственную точку исходящего трафика.
- **agent-server не имеет прямого доступа к tool-api.** Весь доступ к модулям осуществляется через mcp-server по протоколу MCP.
- **Чат синхронный.** WebSocket-соединения в бэкенде отсутствуют; ответ ассистента возвращается как HTTP-ответ на POST /send_message с увеличенным таймаутом прокси (3600 с).

Изоляция данных

Изоляция данных пользователей реализована на двух уровнях:

1. **Реляционная база данных.** Каждая бизнес-сущность несёт поля user_id (владелец; NULL для системных сидов) и build_project_id (проект). Менеджер user_scope фильтрует выборки по значениям id и active_build_project_id из JWT; флаг include_system=True открывает системные записи (user_id IS NULL).
2. **Векторная база Qdrant.** На каждого пользователя создаётся отдельная коллекция (имя = user_id); retrieval выполняется с фильтрами по file_id и chat_id.

Подробное описание архитектурных принципов, цикла обработки данных и подсистем приведено в отдельном документе «Техническое описание».

Системные требования

Требования к аппаратному обеспечению

Требования к вычислительным ресурсам зависят от развёртываемого компонента и ожидаемой нагрузки. В приведённой таблице указаны минимальные и рекомендуемые значения для производственной конфигурации.

Компонент	CPU (мин.)	CPU (реком.)	RAM (мин.)	RAM (реком.)	Диск	Примечание
Бэкенд (6 сервисов + nginx)	4 vCPU	8 vCPU	8 ГБ	16 ГБ	50 ГБ SSD	Включая PostgreSQL, Redis
DS-часть (4 сервиса + Qdrant)	4 vCPU	8 vCPU	8 ГБ	16 ГБ	50 ГБ SSD	Qdrant-хранилище растёт с объёмом индекса
Инфраструктура (PostgreSQL, MinIO, Kafka)	4 vCPU	8 vCPU	8 ГБ	16 ГБ	200 ГБ SSD	Зависит от объёма файлов и сообщений
Инференс LLM (vLLM)	8 vCPU	16 vCPU	32 ГБ	64 ГБ	100 ГБ	Требует GPU
Инференс OCR/ASR/эмбеддинги	4 vCPU	8 vCPU	16 ГБ	32 ГБ	50 ГБ	Требует GPU
Фронтенд	2 vCPU	4 vCPU	4 ГБ	8 ГБ	10 ГБ	Standalone Next.js

> ***GPU опционален.*** *Модели машинного обучения (LLM, OCR, ASR, эмбеддинги) обслуживаются внешними инференс-серверами, совместимыми с OpenAI API. Для локального развёртывания инференса на базе vLLM рекомендуется использование GPU NVIDIA (например, A100 для основного LLM). При использовании внешних (внутри периметра организации) инференс-серверов GPU на узлах комплекса не требуется.*

Требования к программному обеспечению

Компонент	Версия	Назначение
ОС	Linux (Ubuntu 22.04+, Debian 12+, Astra Linux)	Хост-система
Docker	24.0+	Контейнеризация
Docker Compose	2.20+ (v2)	Локальное развёртывание
Kubernetes	1.27+ (опционально)	Кластерное развёртывание
Helm	3.12+ (опционально)	Управление K8s-релизами
Python	3.10 (бэкенд), 3.11–3.12 (DS)	Внутри контейнеров
Node.js	22+ (разработка фронтенда)	Сборка фронтенда; в runtime используется node:20-alpine

PostgreSQL	16	Основная СУБД
Redis	7	Хранилище сессий
MinIO / S3	совместимое с S3	Файловое хранилище
Apache Kafka	2.5.0+ (KRaft 4.1 в K8s)	Брокер сообщений
Qdrant	latest	Векторная база
nginx	1.27+	Шлюз бэкенда и фронтенда
ffmpeg / ffprobe	6.0+	Нормализация медиа (в контейнере file-processing)
poppler	recent	Рендеринг PDF в изображения (в контейнере file-processing)

Требования к сети

Порты бэкенда

При локальном развёртывании через Docker Compose используются следующие хост-порты:

Порт	**Сервис**	**Назначение**
8090	nginx-шлюз	Единая точка входа REST API бэкенда
8080	general-service	Прямой доступ (debug)
8081	fs-manager-service	Прямой доступ (debug)
8082	api-external-service	Прямой доступ (debug)
8083	one-c-dit-service	Прямой доступ (debug)
8084	secretary-service	Прямой доступ (debug)
8099	openapi-aggregator	Swagger/ReDoc/OpenAPI
5432	PostgreSQL	База данных
9000	MinIO	S3 API
9001	MinIO	Web-консоль
9092	Kafka	Брокер сообщений
6380	Redis	Сессии (отображение на 6379)
9999	kafka-ui	Web-консоль Kafka

Порты DS-части

Порт	**Сервис**	**Назначение**
8100	agent-server	Основной контракт POST /agent/run
8101	mcp-server	MCP streamable-http
8102	tool-api	Реализации модулей
8103	file-processing	Препроцессинг и индексация
8104	Qdrant	Векторная БД (отображение на 6333)

Порты фронтенда

Порт	Сервис	Назначение
3000	Next.js (nginx)	Веб-приложение; в K8s — NodePort (по умолчанию 30198)

Внешние зависимости

Зависимость	Назначение	Конфигурация
PostgreSQL 16	Единая логическая БД; схемы public, secretary, one_c_dit	POSTGRESQL_* env
Redis 7	Хранилище сессий auth	REDIS_* env
S3/MinIO	Файловое хранилище (зашифрованные блобы)	S3_* env
Kafka	request/reply между сервисами	KAFKA_* env
Qdrant	Векторная БД (на user_id)	QDRANT_URL env
vLLM (LLM)	Основная языковая модель	VLLM_BASE_URL env
vLLM (эмбеддинги)	Векторные представления текста	EMBEDDING_ENDPOINT env
vLLM (OCR)	Распознавание сканированных документов	OCR_ENDPOINT env
vLLM (ASR)	Расшифровка аудио/видео	ASR_ENDPOINT env
1С	Проводка УПД, каталог номенклатуры	ONE_C_URL env
DIT-модель	Обработка XML, поиск расхождений УПД	DIT_MODEL_URL env

Развёртывание

Локальное развёртывание (Docker Compose)

Локальное развёртывание предназначено для разработки и тестирования. Используются два основных compose-файла: docker-compose-db.yml (только инфраструктура) и docker-compose-startup.yml (полный стек сервисов).

Подготовка сети

Перед запуском создайте общую внешнюю сеть, используемую compose-файлами:

```
docker network create astra_shared_net
```

> *Сеть astra_shared_net объявлена как external: true в per-service compose-файлах (docker-compose-startup-*.yml) и должна существовать до запуска.*

Запуск инфраструктуры

Файл docker-compose-db.yml поднимает только инфраструктурные компоненты: PostgreSQL, MinIO, Zookeeper, Kafka, kafka-ui, Redis.

```
docker compose -f docker-compose-db.yml up -d
```

Запущенные сервисы:

Сервис	Контейнер	Порт(ы)
pg_db	postgres:16	5432 (vol ./pgdata)
minio	minio	9000, 9001 (vol minio-data)
zookeeper	wurstmeister/zookeeper	2181
kafka	wurstmeister/kafka:2.12-2.5.0	9092
kafka-ui	provectus/kafka-ui	9999
redis	redis:7	6380→6379

Запуск сервисов бэкенда

Файл docker-compose-startup.yml поднимает полный стек: все семь сервисов бэкенда плюс инфраструктуру и nginx-шлюз.

```
docker compose -f docker-compose-startup.yml up -d
```

Порядок зависимостей определён в compose-файле: сервис auth зависит от secretary, general, fs-manager, one-c-dit. Полный рекомендуемый порядок запуска при раздельном подъёме per-service файлами:

3. Инфраструктура (docker-compose-db.yml).
4. general → fs-manager → api-external.

5. one-c-dit.
6. secretary.
7. auth (после зависимостей).
8. nginx (шлюз, docker-compose-nginx.yml).

Пример запуска одного сервиса:

```
docker compose -f docker-compose-startup-secretary.yml up -d
```

Healthchecks

Каждый контейнер бэкенда экспонирует публичный endpoint /api/health/. Docker Compose использует python-пробу на основе urllib:

```
python3 -c "import urllib.request; urllib.request.urlopen('http://localhost:8080/api/health/')"
```

Проверка состояния всех сервисов после запуска:

```
# Проверка через шлюз
curl -s http://localhost:8090/health
# Прямые проверки (пример)
curl -s http://localhost:8080/general/api/health/
curl -s http://localhost:8084/secretary/api/health/
curl -s http://localhost:8083/one_c_dit/api/health/health_one_c_and_dit_model/
```

Развёртывание DS-части

DS-часть разворачивается собственным docker-compose.yml из репозитория ds-develop. Порядок запуска определяется зависимостями (healthchecks):

```
docker compose up -d
```

Сервис	Зависимость	Порт
qdrant	— (TCP healthcheck)	8104→6333
file-processing	qdrant healthy	8103→8000
tool-api	qdrant healthy	8102→8000
mcp-server	tool-api healthy	8101→8000
agent-server	mcp-server healthy	8100→8000

> *Для OCR/ASR/эмбеддинг требуется сетевая доступность инференс-серверов (OCR_ENDPOINT, ASR_ENDPOINT, EMBEDDING_ENDPOINT). В compose используется extra_hosts: host.docker.internal:host-gateway для доступа к хост-сети.*

Развёртывание в Kubernetes (Helm)

Кластерное развёртывание использует Helm chart baum-ai-platform. Для бэкенда chart находится в gitlab-ci-files/baum-ai-platform/ (appVersion: "2.4"); для DS — отдельная копия chart (appVersion: "1.0").

Установка релиза

```
helm upgrade --install <release-name> \
  ./gitlab-ci-files/baum-ai-platform \
  -f ./gitlab-ci-files/ai-platform-deploy-values/<env>.yaml \
  -n <namespace> --create-namespace
```

Окружения и значения

Per-env values-файлы хранятся в gitlab-ci-files/ai-platform-deploy-values/. Соответствие окружений, веток и значений приведено в таблице.

Окружение	Values-файл	Ветка	Namespace	Примечание
DEV_240	dev-240.yaml	develop	platform	Dev-стенд, GitLab registry
TESTING_201	testing.yaml	testing	do-testing	Тестирование
FIRSTVDS	firstvds.yaml	master	platform	Prod, Yandex Container Registry
TEST_92	test-92.yaml	master	—	Prod, Yandex Container Registry
SMART_TECH	smart-tech.yaml	master	digital-office	Prod, security-образы, WireGuard
ASTRA	astra-ai.yaml	master	digital-office	Prod, Yandex Container Registry

Структура chart

Chart формирует один Deployment + Service (ClusterIP :8080) на каждый сервис (service-*-service.yaml). Дополнительно:

- **backend-gateway** — Deployment + Service nginx (nginx:1.27-alpine), Service типа NodePort (gateway.nodePort), liveness/readiness пробы на /health.
- **kafka-united** (опционально) — Bitnami Kafka 4.1.0 в режиме KRaft (kafka-united.yaml).
- **_helpers.tpl** — общий блок переменных окружения (runtime/postgresql/s3/kafka/requests/frontend), подключаемый в каждый сервис.

Секреты

Секреты хранятся в K8s Secret и монтируются в контейнеры. Используемые секреты:

Имя секрета	Назначение
postgresql-...	Учётные данные БД
s3-access-key / s3-secret-key	Доступ к S3/MinIO
auth-secret	auth_default_admin_password, auth_default_base_user_password,

	internal_auth_verify_secret
sentry secretName	SENTRY_DSN и связанные

Журналирование

Логи собираются в NFS storage class. Включается параметром storage.enabled; размер тома задаётся logsPersistenceSize (по умолчанию 1Gi). Сервисы пишут логи в stdout, агрегируемые платформой.

NodePort

Доступ извне кластера к шлюзу бэкенда осуществляется через Service типа NodePort. Порт задаётся в values:

```
gateway:
  nodePort: 30710  # значение зависит от окружения (см. таблицу ниже)
```

Окружение	**NodePort**
dev-240, firstvds, test-92	30710
testing	30720
smart-tech	30730

Фронтенд в K8s экспонируется отдельным NodePort (по умолчанию 30198 в values-файлах).

Развёртывание фронтенда

Фронтенд (Next.js 15) поставляется как standalone Docker-образ и разворачивается отдельным Helm chart, независимым от бэкенда. Развёртывание состоит из сборки образа и K8s-ресурсов.

Dockerfile (two-stage)

Сборка образа выполняется двухстадийным Dockerfile на базе node:20-alpine:

Стадия	**Назначение**
build	Базовый образ node:20-alpine; выполняется npm ci, npm run build; артефакты .next/static и каталог public копируются в standalone-выход
runner	Базовый образ node:20-alpine; устанавливается tzdata, таймзона Europe/Moscow; standalone-сборка копируется в /app/app; EXPOSE 3000; запуск node server.js

Standalone output

В next.config.ts установлено output: "standalone" — Next.js собирает самодостаточный серверный бандл, не требующий установки node_modules в runtime.

Standalone-сервер Next.js

В production-образе работает только Node-процесс node server.js (без nginx): Next.js standalone-сервер слушает порт 3000 (ENV PORT=3000, HOSTNAME=<bind-address>). В Kubernetes контейнер экспонирует порт 3000 (используется как NodePort target).

Helm chart фронтенда

Отдельный Helm chart расположен в gitlab-ci-files/helm/ (chart name frontend, appVersion: "1.0"). Шаблон templates/common.yaml формирует Deployment + Service (NodePort 30198 или ClusterIP) на каждую запись в .Values.frontends.

Per-env values-файлы:

Values-файл	Окружение
dev.yaml	dev-240
testing.yaml	testing-201
firstvds.yaml	firstvds prod
test-92.yaml	test-92 prod
astra-ai.yaml	astra
smart-tech.yaml	smart-tech

Каждый values-файл задаёт containerPull.registry/pullSecret, replicaCount: 1, ingress и список frontends с одним digital-office entry на NodePort 30198.

Окружения фронтенда

Окружение	URL
dev-240	http://dev.digitaloffice.aib.pro
testing-201	http://testing.digitaloffice.aib.pro
firstvds / test-92 (prod)	https://gpt.aib.pro
astra	astra-ai
smart-tech	smart-tech

Переменные окружения

Переменные окружения сгруппированы по общим пакетам, используемым всеми сервисами. Ниже приведены таблицы по каждой группе. Значения по умолчанию указаны для dev-конфигурации; в production должны быть заданы безопасные значения через секреты.

Группа microservice

Переменная	Описание	Default

API_HOST	Адрес привязки сервиса (привязка ко всем интерфейсам — значение по умолчанию)	<bind-address>
API_PORT	Порт привязки сервиса	8080
API_URL_PREFIX	Префикс URL API	/api
API_DEBUG	Режим отладки	false
FRONTEND_URLS	Разрешённые CORS-источники (через запятую)	http://localhost:3000
SENTRY_DSN	DSN Sentry/Bugsink	—
SENTRY_ENVIRONMENT	Имя окружения для Sentry	—
SENTRY_RELEASE	Версия релиза	—
SENTRY_SERVER_NAME	Имя сервера	—
GIT_COMMIT	Хэш фиксации (для трассировки)	—
HOSTNAME	Имя хоста	—

Группа postgresql_adapter

Переменная	**Описание**	**Default (dev)**
POSTGRESQL_USERNAME	Пользователь БД	root
POSTGRESQL_PASSWORD	Пароль БД	root
POSTGRESQL_HOST	Хост БД	localhost
POSTGRESQL_PORT	Порт БД	5432
POSTGRESQL_DEFAULT_DB	Имя базы данных	test_db

> *В K8s имена баз: bai (prod), bai_testing (testing). Пароли и хост задаются через секреты.*

Группа kafka_adapter

Переменная	**Описание**	**Default**
KAFKA_SERVER	Адрес(а) брокера	localhost:9092
KAFKA_CONSUMER_GROUP_ID	Группа консьюмера	равно имени сервиса
KAFKA_SECURITY_PROTOCOL	Протокол безопасности	SASL_SSL
KAFKA_SASL_MECHANISM	SASL-механизм	SCRAM-SHA-512
KAFKA_SASL_USERNAME	SASL-пользователь	—
KAFKA_SASL_PASSWORD	SASL-пароль	—
KAFKA_API_VERSION	Версия API Kafka	2.5.0
TOPIC_FOR_ANSWER	Топик ответа (per-service)	<service>_for_answer

> *Поддерживаются сообщения большого размера (~2 ГБ).*

Группа redis_adapter

Переменная	Описание	Default
REDIS_HOST	Хост Redis	localhost
REDIS_PORT	Порт Redis	6379
REDIS_DB	Номер БД Redis	0
REDIS_USERNAME	Пользователь Redis	—
REDIS_PASSWORD	Пароль Redis	—
REDIS_SSL	Использовать TLS	false

Группа s3_adapter

Переменная	Описание	Default (dev)
S3_BUCKET_NAME	Имя бакета	bucket
S3_ENDPOINT_URL	Endpoint S3/MinIO	http://localhost:9000
S3_ACCESS_KEY	Ключ доступа	minioadmin
S3_SECRET_ACCESS_KEY	Секретный ключ	minioadmin
TEXT_EXT_FORMAT	Текстовые расширения (JSON-строка списка)	[".txt", ".pdf", ".docx"]
MEDIA_EXT_FORMAT	Медиа-расширения (JSON-строка списка)	[".mp4", ".mp3", ".wav"]
FS_ENCRYPTION_ENABLED	Включить шифрование файлов	true
FS_ENCRYPTION_MASTER_KEY	Мастер-ключ AES-256-GCM	— (обязательно в prod)
FS_CHUNK_SIZE	Размер чанка шифрования	65536
DEFAULT_BUILD_PROJECT_ID	Проект по умолчанию	d9ca634d-f42f-4dd5-b9fc-b93ec2ca0001

> *Значение DEFAULT_BUILD_PROJECT_ID по умолчанию — d9ca634d-f42f-4dd5-b9fc-b93ec2ca0001 (системный build-проект из build_projects_templates.json, сидируется general-service при запуске). Используется как active_build_project_id для системных пользователей.*

Группа auth_utils

Переменная	Описание	Default
JWT_SECRET_KEY	Секрет подписи JWT (HS256)	— (обязательно)
JWT_ALGORITHM	Алгоритм JWT	HS256
JWT_EXPIRES_IN_SECONDS	Время жизни access JWT	900

SESSION_TTL_SECONDS	TTL сессии в Redis	14400
VERIFY_CACHE_TTL_SECONDS	TTL кэша verify	10
FINGERPRINT_SECRET_SALT	Соль fingerprint сессии	— (обязательно)
INTERNAL_AUTH_VERIFY_SECRET	Секрет внутреннего auth_request	— (обязательно)
SECURE_MOD_FOR_TOKEN	Флаг Secure для cookie	зависит от окружения
AUTH_DEFAULT_ADMIN_PASSWORD	Пароль системного администратора (dev)	—
AUTH_DEFAULT_BASE_USER_PASSWORD	Пароль базового пользователя (dev)	—

> *JWT_SECRET_KEY, JWT_ALGORITHM должны совпадать в auth-service, api-external-service и Kafka-консьюмерах. FS_ENCRYPTION_MASTER_KEY должен совпадать в fs-manager, secretary и one-c-dit.*

Группа logger

Переменная	Описание	Default
LEVEL	Уровень логирования	DEBUG
DEBUG_LOG	Расширенное логирование	false

Группа api-external

Переменная	Описание	Default
DS_MODEL_URL	URL agent-server (DS)	http://agent-server:8000
DS_MODEL_API_KEY	API-ключ DS (заголовок x-api-key)	—
DS_RAG_URL	URL file-processing (DS RAG)	http://file-processing:8000
DS_OUTBOUND_USER_JWT_HEADER	Заголовок user-data JWT	X-User-Data-Jwt
ONE_C_URL	URL 1C	http://<IP-адрес-сервера-1C>
ONE_C_AUTHORIZATION	Заголовок Authorization для 1C (Basic)	—
DIT_MODEL_URL	URL DIT-модели	—
DIT_MODEL_HEALTH_URL	URL health DIT-модели	—
DIT_MODEL_DI_KEY	API-ключ DIT (заголовок x-api-key)	—
DEFAULT_REQUESTS_TIMEOUT	Таймаут исходящих запросов	3600

Группа DS (общие)

Переменная	Описание	Default

LOG_LEVEL	Уровень логирования DS	INFO
REQUEST_TIMEOUT_SECONDS	Таймаут запросов	3600
VLLM_BASE_URL	URL основного LLM	http://<IP-адрес-LLM-сервера>:8000/v1
VLLM_MODEL_NAME	Имя модели LLM	MiniMaxAI/MiniMax-M2.7
VLLM_API_KEY	API-ключ LLM (пустой = без Authorization)	—
POSTGRES_*	Параметры Postgres (shared с бэкендом)	см. postgresql_adapter
S3_*	Параметры S3 (shared с бэкендом)	см. s3_adapter
QDRANT_URL	URL Qdrant	http://qdrant:6333
QDRANT_API_KEY	API-ключ Qdrant	—

Группа DS (file-processing)

Переменная	**Описание**	**Default**
OCR_ENDPOINT	URL OCR-модели	http://<IP-адрес-сервера-OCR-ASR>:8041
OCR_MODEL	Имя OCR-модели	zai-org/GLM-OCR
OCR_BATCH_SIZE	Размер батча OCR	5
OCR_PAGE_TIMEOUT_S	Таймаут страницы OCR	60
OCR_MAX_PAGES	Максимум страниц	500
OCR_DPI	DPI рендеринга	300
EMBEDDING_ENDPOINT	URL эмбеддинг-модели	http://<IP-адрес-LLM-сервера>:8044
EMBEDDING_MODEL	Имя эмбеддинг-модели	intfloat/multilingual-e5-large
EMBEDDING_DIM	Размерность векторов	1024
EMBED_BATCH_SIZE	Размер батча эмбеддингов	64
EMBED_TIMEOUT_S	Таймаут эмбеддингов	30
ASR_ENDPOINT	URL ASR-модели	http://<IP-адрес-сервера-OCR-ASR>:8017
ASR_MODEL	Имя ASR-модели	Qwen/Qwen3-ASR-1.7B
ASR_CHUNK_DURATION_SEC	Длительность чанка ASR	60
ASR_MAX_CONCURRENT	Параллелизм ASR	16
ASR_TEMPERATURE	Температура ASR	0.0
ASR_TIMEOUT_S	Таймаут ASR	300
CHUNK_SIZE_TOKENS	Размер чанка (токены)	512

CHUNK_OVERLAP_TOKENS	Перекрытие чанка	64
PIPELINE_VERSION	Версия пайплайна	1.0

Группа DS (per-service)

Переменная	**Описание**	**Default**
AGENT_SERVER_*	Параметры agent-server	—
MCP_SERVER_*	Параметры mcp-server	—
TOOL_API_*	Параметры tool-api	—
MCP_SERVER_URL	URL mcp-server	http://mcp-server:8000
TOOL_API_URL	URL tool-api	http://tool-api:8000

Переменные окружения фронтенда

Фронтенд (Next.js) использует отдельный набор переменных окружения, отличный от бэкенда. Переменные с префиксом NEXT_PUBLIC_ встраиваются в клиентский бандл во время сборки; прочие доступны только на server-side.

Переменная	**Описание**
NEXT_PUBLIC_URL_API	Базовый URL REST API бэкенда (используется RTK Query)
BACKEND_API_URL	Реальный адрес бэкенда для dev-only прокси /backend-api/* (server-side)
NEXT_PUBLIC_URL_WS	URL WebSocket сервера (объявлена, клиентская реализация отсутствует)
NEXT_PUBLIC_PATH_WS	Путь WebSocket эндпоинта, по умолчанию /api/chat/socket.io (объявлена, не реализована)

> *Переменные NEXT_PUBLIC_URL_WS и NEXT_PUBLIC_PATH_WS объявлены в конфигурации фронтенда и CI, однако клиентская реализация WebSocket (socket.io/EventSource) в текущей версии отсутствует — чат работает через синхронный REST API (POST /secretary/api/chats/{id}/send_message).*

Пример .env для локального развёртывания

```
# --- microservice ---
API_HOST=<bind-address>  # адрес привязки; по умолчанию — все интерфейсы; при необходимости ограничьте конкретным адресом
API_URL_PREFIX=/api
FRONTEND_URLS=http://localhost:3000

# --- postgresql_adapter ---
POSTGRESQL_USERNAME=root
POSTGRESQL_PASSWORD=root
```

```
POSTGRESQL_HOST=pg_db
POSTGRESQL_PORT=5432
POSTGRESQL_DEFAULT_DB=test_db

# --- redis_adapter ---
REDIS_HOST=redis
REDIS_PORT=6379

# --- s3_adapter ---
S3_BUCKET_NAME=bucket
S3_ENDPOINT_URL=http://minio:9000
S3_ACCESS_KEY=minioadmin
S3_SECRET_ACCESS_KEY=minioadmin
FS_ENCRYPTION_ENABLED=true
FS_ENCRYPTION_MASTER_KEY=[ЗАПОЛНИТЬ: сгенерировать 32-байтный ключ]

# --- auth_utils ---
JWT_SECRET_KEY=[ЗАПОЛНИТЬ: сгенерировать надёжный секрет]
INTERNAL_AUTH_VERIFY_SECRET=[ЗАПОЛНИТЬ: сгенерировать надёжный секрет]
FINGERPRINT_SECRET_SALT=[ЗАПОЛНИТЬ: сгенерировать надёжную соль]
AUTH_DEFAULT_ADMIN_PASSWORD=[ЗАПОЛНИТЬ: пароль администратора]
AUTH_DEFAULT_BASE_USER_PASSWORD=[ЗАПОЛНИТЬ: пароль пользователя]

# --- api-external ---
DS_MODEL_URL=http://agent-server:8000
DS_RAG_URL=http://file-processing:8000
ONE_C_URL=http://<IP-адрес-сервера-1С>
```

Конфигурация nginx-шлюза

nginx-шлюз (config/nginx/) — единая точка входа REST API бэкенда. Конфигурация реализует прозрачный обмен сессионного cookie на краткосрочный JWT и маршрутизацию запросов к сервисам.

Основная конфигурация

Файл nginx.conf определяет сервер на порту :8080 (внешне отображается на 8090), client_max_body_size 2000m (для загрузки больших файлов) и подключает подконфиги:

- includes/upstreams.conf — upstream-ы сервисов;
- includes/auth_policy.map — карта публичных/защищённых endpoints;
- includes/auth_gateway.conf — логика auth_request;
- routes/*.conf — маршруты к сервисам.

Значение INTERNAL_AUTH_VERIFY_SECRET читается из env nginx.

Upstreams

Файл includes/upstreams.conf объявляет upstream-ы (все на внутренний порт :8080 контейнеров):

Upstream	**Сервис**
service_auth_upstream	auth-service
service_general_upstream	general-service
service_fs_manager_upstream	fs-manager-service

service_api_external_upstream	api-external-service
service_one_c_dit_upstream	one-c-dit-service
service_secretary_upstream	secretary-service
service_openapi_aggregator_upstream	openapi-aggregator (порт 8099)

Политика аутентификации

Файл includes/auth_policy.map определяет map "$request_method:$request_uri" → $auth_is_public. Публичные endpoints (значение 1) не требуют аутентификации:

Метод + путь	**Публичный**
GET/HEAD /health	да
GET/HEAD /docs, /redoc, /openapi.json	да
POST /auth/api/login	да
POST /auth/api/register	да
GET/HEAD /<service>/api/health/ (для всех префиксов)	да
GET/HEAD /one_c_dit/api/health/health_one_c_and_dit_model	да
GET/HEAD /<service>/docs, /<service>/openapi.json	да
Все прочие	нет

Шлюз аутентификации

Файл includes/auth_gateway.conf реализует цепочку:

9. /_auth_noop — возвращает 204 (no-op для публичных).
10. /_auth_dispatch — возвращает 204, если endpoint публичен ($auth_is_public = 1), иначе 418.
11. /_auth_verify — internal location; proxy_pass POST на service_auth_upstream/auth/api/verify_token/, форвардит cookie/UA/IP и заголовок X-Internal-Auth-Secret.
12. Цепочка error_page 418 = /_auth_verify связывает dispatch → verify.

nginx захватывает JWT из ответа через auth_request_set $auth_jwt и форвардит в upstream заголовок Authorization: Bearer $auth_jwt.

Маршруты

Route-файлы (routes/*.conf) определяют location /<prefix>/ с директивой auth_request /_auth_dispatch. Особенности:

- **`secretary.conf`** — добавляет long-timeout (proxy_read_timeout 3600s и др.) для паттерна ^/secretary/api/chats/[^/]+/send_message$, поскольку генерация ответа может занимать длительное время.
- **`auth.conf`** — скрывает заголовок X-Auth-Jwt от клиента (через proxy_hide_header) и возвращает 404 для внешнего доступа к /auth/api/verify_token/ (endpoint внутренний).

Тестирование DS-компонента

DS-часть проекта содержит набор автоматических тестов, организованных в едином централизованном каталоге tests/ на верхнем уровне репозитория ds-develop/. Тесты реализованы на базе фреймворка **pytest** (версия 8.4.1) с плагином **pytest-asyncio** (1.1.0) для асинхронных тест-функций и библиотекой **httpx** (0.28.1) в качестве транспорта для fastapi.testclient.TestClient. Внутри сервисов (agent_server, mcp_server, tool_api, file_processing) локальных каталогов tests/ нет — все тесты собраны централизованно.

Структура тестового каталога

```
ds-develop/tests/
├── conftest.py              — настройка путей импорта (sys.path)
├── test_chat_indexing.py    — индексация истории чата (tool_api)
├── test_chat_title.py       — генератор заголовков чата (agent_server)
├── test_fastmcp_server.py   — MCP-сервер: список и вызов инструментов
└── test_service_flow.py     — оркестрация агента: tool-loop, chaining
```

Файл conftest.py не определяет фикстур — он добавляет корни пакетов (services/agent_server, services/mcp_server, services/tool_api, services/) в sys.path, что позволяет использовать плоские импорты вида from agent_server.schemas.chat import .... Отдельного pytest.ini, pyproject.toml или setup.cfg нет — pytest работает с настройками по умолчанию.

Покрытие по сервисам

Сервис	Тестовый модуль	Кол-во тестов	Что проверяется
agent-server	test_service_flow.py	12	Оркестрация агента: цепочки инструментов (summarization→dashboard), форсирование features_target_name, инжекция user_id/chat_id в QA-инструмент, обработка ошибок (отсутствие шаблона, нет источников), сокрытие технических полей из схемы, видимой LLM
agent-server	test_chat_title.py	8	Генерация заголовков: нормализация (кавычки, пробелы), обрезка до 80 символов, fallback при пустом ответе LLM, обрезка user_msg до 2000 символов, валидация pydantic-схемы, интеграционный тест маршрута /chat/title через TestClient
mcp-server	test_fastmcp_server.py	2	Список инструментов через fastmcp.Client (ровно 4: summarization, dashboard, question_answering, presentation — в

			фиксированном порядке), валидация JSON-schema каждого инструмента, реальный вызов summarization через MCP-клиент
tool-api	test_chat_indexing.py	6	Чанкинг текста (chunk_text), детерминизм UUIDv5 (deterministic_uuid), значения по умолчанию pydantic-схемы ChatMessageInput, пропуск пустых сообщений, идемпотентность индексации по (chat_id, message_id, chunk_idx)
file-processing	—	0	Тестов нет (OCR, парсеры, ASR, медиа-препроцессинг не покрыты)

Тестовые двойники

Отдельных файлов фикстур, фабрик или общего слоя моков нет. Все тестовые двойники реализованы как inline-классы Fake*/Stub* внутри соответствующих модулей:

Класс	**Модуль**	**Назначение**
FakeLLMClient	test_chat_title.py, test_service_flow.py	Имитация ответов LLM (create_text_completion, create_chat_completion) с запрограммированной последовательностью
FakeMCPClient	test_service_flow.py	Отдаёт 4 MCP-инструмента с input_schema, логирует вызовы invoke_tool
FakeToolAPIClient	test_fastmcp_server.py, test_service_flow.py	Заглушка индексации истории и вызова tool-api
FakeChatHistoryService	test_service_flow.py	Заглушка истории чата
FakeTemplatePromptService	test_service_flow.py	Разрешение шаблонов промптов, умеет выбрасывать ошибку для template_missing
StubChatHistoryIndexingService	test_chat_indexing.py	Запоминает вызовы, считает непустые сообщения

Запуск тестов

Тесты запускаются локально из корня ds-develop/:

```
# Установка зависимостей (pytest уже в requirements.txt)
pip install -r requirements.txt

# Запуск всех тестов
pytest tests/ -v

# Запуск конкретного модуля
pytest tests/test_service_flow.py -v

# Запуск с фильтром по маркеру
pytest tests/ -v -k "asyncio"
```

> *Тесты не требуют запущенных сервисов или инфраструктуры (PostgreSQL, Redis, S3, Qdrant, vLLM) — все внешние зависимости заменены inline-моками. Исключение — `test_fastmcp_server.py`, который запускает MCP-сервер in-memory через `fastmcp.Client` без сетевого взаимодействия.*

Управление пользователями, ролями и правами

Управление доступом в комплексе реализовано в auth-service. Модель включает аутентификацию на основе сессий, двухтокенную схему (session cookie + краткосрочный JWT) и ролевое управление доступом (RBAC) с гранулярными правами, привязанными к проектам.

auth-service

auth-service (URL-префикс /auth) отвечает за аутентификацию, управление сессиями, выпуск JWT и RBAC. Инфраструктура сервиса: PostgreSQL + Redis (без S3, без Kafka). Сервис сидирует роли, права, системных пользователей и проекты из JSON-шаблонов при запуске.

REST endpoints

Endpoint	Метод	Назначение	Публичный
/auth/api/register	POST	Регистрация нового пользователя	да
/auth/api/login	POST	Вход; создание сессии, установка cookie	да
/auth/api/logout	POST	Выход; удаление сессии	нет
/auth/api/verify_token	POST	Внутренняя проверка сессии (вызывается nginx)	внутренний
/auth/api/users	GET	Список пользователей	нет
/auth/api/users/me	GET	Профиль текущего пользователя	нет
/auth/api/users	PATCH	Редактирование пользователя	нет
/auth/api/update_auth_param/email	POST	Смена email	нет
/auth/api/update_auth_param/password	POST	Смена пароля	нет
/auth/api/roles	GET/POST/PATCH/DELETE	Управление ролями	нет
/auth/api/rights	GET	Каталог прав	нет
/auth/api/rights/{roleId}/rights/{rightId}	POST/DELETE	Назначение/отзыв права роли	нет
/auth/api/health/	GET	Healthcheck	да

Аутентификация

Регистрация

Регистрация выполняется запросом POST /auth/api/register. Требования к паролю: не менее 8 символов, наличие строчных и прописных букв, цифр и спецсимволов. Хэширование пароля выполняется алгоритмом argon2.

```
curl -X POST https://<gateway>/auth/api/register \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Иван",
    "lastname": "Иванов",
    "email": "ivanov@example.com",
    "login": "ivanov",
    "password": "[ЗАПОЛНИТЬ: надёжный пароль]"
  }'
```

Вход

Вход выполняется запросом POST /auth/api/login. При успешной проверке пароля (argon2) создаётся сессия в Redis и устанавливается httpOnly cookie session_token.

```
curl -X POST https://<gateway>/auth/api/login \
  -H "Content-Type: application/json" \
  -c cookies.txt \
  -d '{
    "login": "ivanov",
    "password": "[ЗАПОЛНИТЬ: пароль]"
  }'
```

Cookie session_token содержит JWT без поля exp (claims sub, sid, typ=session). Полезная нагрузка сессии (user_id, session_id, fingerprint) сохраняется в Redis. TTL сессии — SESSION_TTL_SECONDS (по умолчанию 14400 с / 4 часа), продлевается при каждом обращении.

Сессии

Сессии хранятся в Redis. Структура ключей:

Ключ	Назначение	TTL
SESSION_REDIS_KEY_PREFIX:<session_token>	Полезная нагрузка сессии (user_id, session_id, fingerprint)	SESSION_TTL_SECONDS (продлевается)
USER_SESSION_REDIS_KEY_PREFIX:<user_id>	Токен активной сессии пользователя (для инвалидации при повторном входе)	SESSION_TTL_SECONDS
VERIFY_CACHE_REDIS_KEY_PREFIX:<token>	Кэш проверки verify (краткосрочный)	VERIFY_CACHE_TTL_SECONDS (10 с)
auth:healthcheck	Проба доступности Redis при запуске	—

Fingerprint сессии вычисляется как SHA-256 (бесключевой хэш) от строки user_agent|client_ip|FINGERPRINT_SECRET_SALT. Проверка fingerprint защищает от угона cookie: при несовпадении UA/IP сессия отклоняется. Сравнение выполняется константным по времени методом (hmac.compare_digest).

Параметры cookie:

Параметр	Значение
Имя	session_token
httpOnly	да
secure	определяется SECURE_MOD_FOR_TOKEN (включается на HTTPS)
samesite	lax

Проверка токена (VerifyToken)

Проверка токена выполняется внутренней процедурой VerifyToken.post по endpoint POST /auth/api/verify_token/. Процедура вызывается nginx-шлюзом через механизм auth_request при каждом непубличном запросе.

Логика:

13. nginx направляет внутренний подзапрос к /_auth_verify, который проксирует POST на service_auth_upstream/auth/api/verify_token/.
14. nginx передаёт cookie, User-Agent, IP клиента и заголовок X-Internal-Auth-Secret.
15. VerifyToken.post проверяет совпадение X-Internal-Auth-Secret == INTERNAL_AUTH_VERIFY_SECRET.
16. Валидируется сессия (с 10-секундным кэшем verify в Redis).
17. Загружаются данные пользователя, резолвятся роли и права для активного проекта (active_build_project_id).
18. Выпускается краткосрочный JWT (HS256, JWT_EXPIRES_IN_SECONDS = 900 с).
19. JWT возвращается в заголовке X-Auth-Jwt; nginx захватывает его (auth_request_set $auth_jwt) и форвардит в upstream как Authorization: Bearer <jwt>, удаляя X-Auth-Jwt из ответа клиенту.

Claims краткосрочного JWT:

Claim	Описание
id	Идентификатор пользователя
session_id	Идентификатор сессии
login	Логин
email	Email
name	Имя
lastname	Фамилия
roles	Список ролей
rights	Список прав для активного проекта
active_build_project_id	Активный проект

> *JWT_SECRET_KEY и JWT_ALGORITHM должны совпадать в auth-service, api-external-service и всех Kafka-консьюмерах. INTERNAL_AUTH_VERIFY_SECRET должен совпадать в auth-service и конфигурации nginx.*

Ролевая модель (RBAC)

Модель доступа построена на ролях и правах, привязанных к проектам сборки (Build Project). Структура хранения в схеме public:

Таблица	**Назначение**
users	Пользователи
build_project	Проекты сборки
rules	Роли (имя = rule)
rights	Права (имя = right_name)
rights_rel	Связь роль

general.build_projects.switch	Смена активного проекта
auth.logout.post	Выход из системы
auth.update_auth_param.password	Смена пароля
auth.update_auth_param.email	Смена email

> *Роль admin получает право system.admin, которое даёт полный административный доступ ко всем API и операциям, не требуя явного перечисления гранулярных прав.*

Системные пользователи

При запуске сидируются два системных пользователя из users_templates.json:

Логин	Имя	Email	Роль	Пароль (env)	Проект
platform_admin	System Administrator	admin@astra.local	admin	AUTH_DEFAULT_ADMIN_PASSWORD	d9ca634d-f42f-4dd5-b9fc-b93ec2ca0001
platform_user	System BaseUser	user@astra.local	user	AUTH_DEFAULT_BASE_USER_PASSWORD	d9ca634d-f42f-4dd5-b9fc-b93ec2ca0001

> *Системные пользователи имеют is_system: true. Пароли задаются через переменные окружения (только для dev-окружения). В production пароли должны быть изменены после первого запуска.*

Каталог прав

Права (right_name) синхронизируются автоматически: при запуске каждого сервиса его handlers декларируют права через декоратор set_responses (параметры right_name, right_description), а функция service_rights_func создаёт соответствующие строки в таблице rights. Полный каталог прав, извлечённый из кода сервисов, приведён ниже.

Права auth-сервиса

Право	Описание
Пользователи	Просмотр профиля пользователя или списка пользователей
Создание пользователя	Создание пользователя администратором с назначением в проект
Редактирование пользователя	Изменение имени, фамилии или email пользователя
Удаление пользователя	Мягкое удаление пользователя из системы
Добавление в проект	Включение пользователя в проект сборки
Исключение из проекта	Удаление пользователя из проекта сборки

Назначение роли	Назначение роли пользователю в проекте
Отзыв роли	Снятие роли с пользователя в проекте
Список ролей	Просмотр ролей проекта и связанных с ними прав
Создание роли	Создание новой роли в проекте сборки
Редактирование роли	Изменение названия и описания роли проекта
Удаление роли	Мягкое удаление роли проекта
Каталог прав	Просмотр списка прав доступа
Назначение права роли	Добавление права к роли проекта
Отзыв права у роли	Удаление права из роли проекта
Мой профиль	Просмотр собственного профиля с членством и ролями
Смена email	Изменение email текущего пользователя
Смена пароля	Изменение пароля текущего пользователя

Права general-сервиса

Право	**Описание**
Проекты сборки	Просмотр проекта или списка всех проектов
Мои проекты	Просмотр проектов сборки, доступных текущему пользователю
Создание проекта	Создание проекта сборки с выбранными функциональностями
Редактирование проекта	Изменение параметров проекта сборки
Удаление проекта	Мягкое удаление проекта сборки
Смена активного проекта	Переключение активного проекта у текущего пользователя
Список моделей	Просмотр моделей, доступных для использования в проекте
Базовые промпты	Просмотр базовых шаблонов промптов general-сервиса
Поиск промптов	Поиск базовых промптов по имени и тексту

Права fs-manager-сервиса

Право	**Описание**
Список файлов	Просмотр файлов и папок в хранилище проекта
Загрузка файла	Загрузка нового файла в хранилище проекта
Замена файла	Обновление содержимого существующего файла
Редактирование файла	Изменение метаданных файла или папки
Удаление файлов	Мягкое удаление одного или нескольких файлов и папок
Скачивание файла	Скачивание файла из хранилища проекта

Права secretary-сервиса

Право	Описание
Список чатов	Просмотр чата или списка чатов пользователя
Поиск чатов	Поиск чатов по названию и тексту сообщений
Создание чата	Создание нового чата с автоматической инициализацией
Редактирование чата	Изменение названия или модели чата
Удаление чата	Мягкое удаление чата и связанных временных источников
Сообщения чата	Просмотр сообщений выбранного чата
Отправка сообщения	Отправка сообщения в чат и получение ответа ассистента
Шаблоны фич чата	Просмотр активных шаблонов фич для чата
Шаблон фичи в чате	Смена активного шаблона фичи для конкретного чата
Список источников	Просмотр источников с фильтрацией по типу, родителю и чату
Поиск источников	Поиск источников по имени и атрибутам
Создание источника	Создание файла, папки или ссылки в дереве источников
Редактирование источников	Изменение параметров одного или нескольких источников
Удаление источников	Мягкое удаление одного или нескольких источников
Скачивание источника	Скачивание файла, связанного с источником
Замена файла источника	Обновление содержимого файла, привязанного к источнику
Источники чата	Просмотр источников, подключённых к чату, со статусами
Подключение источника	Подключение источника к чату или изменение его активности
Список фич	Просмотр функциональных возможностей секретаря
Шаблоны	Просмотр шаблонов с фильтрацией по фиче
Поиск шаблонов	Поиск шаблонов по имени и тексту промптов шагов
Создание шаблона	Создание шаблона с привязкой шагов и промптов
Редактирование шаблона	Изменение шаблона и активных промптов его шагов
Удаление шаблона	Мягкое удаление пользовательского шаблона
Шаблон по умолчанию	Назначение шаблона по умолчанию для фичи
Шаги шаблона	Просмотр шагов шаблона и выбранных для них промптов
Шаги шаблонов	Просмотр шагов шаблонов с фильтрацией по фиче
Промпты шага	Просмотр промптов, привязанных к шагу шаблона
Создание промпта шага	Создание промпта и привязка его к шагу шаблона
Промпты секретаря	Просмотр промптов с фильтрацией по фиче, шагу и шаблону
Поиск промптов	Поиск промптов секретаря по имени и тексту
Редактирование промпта	Обновление пользовательского промпта секретаря
Удаление промпта	Мягкое удаление промпта секретаря

Права one-c-dit-сервиса

Право	Описание
XML без UPD	Просмотр XML-документов, для которых ещё не создан UPD
Создание XML	Создание XML-документа для последующей обработки в UPD
Удаление XML	Удаление XML-документа по id или всех записей пользователя
Список UPD	Просмотр UPD или списка документов пользователя
Создание UPD	Создание универсального передаточного документа на основе XML
Удаление UPD	Удаление UPD по id или всех записей пользователя
Шаги UPD	Просмотр шагов обработки UPD и их статусов
Обновление шага UPD	Изменение статуса шага обработки UPD
Расхождения UPD	Поиск расхождений UPD с данными 1С и варианты их решения
Решение расхождений	Применение выбранных вариантов устранения расхождений UPD

Управление доступом через UI

Управление пользователями, ролями и правами доступно через веб-интерфейс на странице /admin (компонент AdminPageView). Интерфейс предоставляет:

Вкладка	Функции
Участники (Members)	Просмотр списка участников проекта, добавление/исключение пользователей, назначение/отзыв ролей
Роли (Roles)	Просмотр ролей проекта, создание/редактирование/удаление ролей
Права (Permissions)	Просмотр прав роли, назначение/отзыв отдельных прав

Управление доступом ограничено рамками активного проекта (Build Project). Все изменения прав и ролей применяются в контексте проекта, выбранного пользователем.

Авторизация в сервисах

Авторизация в бизнес-сервисах построена на следующих принципах:

20. **Декларация прав.** Каждый handler декларирует right_name/right_description через декоратор set_responses. При запуске сервиса функция service_rights_func синхронизирует их как строки в таблице rights.
21. **Публичные endpoints.** Endpoints с auth=False не требуют аутентификации (healthchecks, docs, openapi).
22. **Проверка прав.** При обращении к защищённому endpoint проверяется наличие соответствующего права в claims JWT (rights).

23. **Owner-scoping.** Фильтрация данных по владельцу выполняется в PostgreSQL по user_id + active_build_project_id из JWT. Системные записи (user_id IS NULL) доступны через include_system=True.

Авторизация в Kafka

Для межсервисного обмена через Kafka данные пользователя (JwtUserDataSchema) упаковываются в JWT без exp и передаются в заголовке user_data Kafka-сообщения. Handlers, требующие user_data, выбрасывают UnauthorizedError при отсутствии или невалидности заголовка. Для исходящих HTTP-вызовов в DS api-external строит тот же JWT и отправляет его в заголовке DS_OUTBOUND_USER_JWT_HEADER (X-User-Data-Jwt).

Управление проектами

Проект сборки (Build Project) — основная единица изоляции данных в комплексе. Каждый проект объединяет пользователей, файлы, чаты, источники, шаблоны и настройки ролей. Пользователь может входить в несколько проектов; в каждый момент времени активен один проект, идентификатор которого хранится в JWT (active_build_project_id).

general-service

Управление проектами реализовано в general-service (URL-префикс /general). Сервис отвечает за справочный API: каталог LLM, промпты, проекты сборки. Инфраструктура: PostgreSQL + Kafka. Сиды проектов и моделей загружаются из JSON-шаблонов при запуске.

REST endpoints

Endpoint	Метод	Назначение	Право
/general/api/build_projects/	GET	Список всех проектов	Проекты сборки
/general/api/build_projects/	POST	Создание проекта	Создание проекта
/general/api/build_projects/{id}	GET	Просмотр проекта	Проекты сборки
/general/api/build_projects/{id}	PATCH	Редактирование проекта	Редактирование проекта
/general/api/build_projects/{id}	DELETE	Удаление проекта	Удаление проекта
/general/api/build_projects/my	GET	Проекты текущего пользователя	Мои проекты
/general/api/build_projects/switch	POST	Смена активного проекта	Смена активного проекта
/general/api/models	GET	Каталог LLM	Список моделей
/general/api/prompts	GET	Базовые промпты	Базовые промпты
/general/api/prompts/search	GET	Поиск промптов	Поиск промптов
/general/api/health/	GET	Healthcheck	—

Системный проект

При запуске general-service сидирует системный проект из build_projects_templates.json. Идентификатор системного проекта (d9ca634d-f42f-4dd5-b9fc-b93ec2ca0001) используется как active_build_project_id для системных пользователей и как DEFAULT_BUILD_PROJECT_ID в конфигурации S3.

Смена активного проекта

Смена активного проекта выполняется запросом POST /general/api/build_projects/switch. После смены последующий verify_token выпустит JWT с новым active_build_project_id, и все owner-scoping выборки будут фильтроваться по новому проекту.

```
curl -X POST https://<gateway>/general/api/build_projects/switch \
  -H "Content-Type: application/json" \
  -b cookies.txt \
  -d '{"build_project_id": "d9ca634d-f42f-4dd5-b9fc-b93ec2ca0001"}'
```

Изоляция проектов

Изоляция по проектам реализована на уровне базы данных: каждая бизнес-сущность несёт поле build_project_id. Менеджер user_scope фильтрует выборки по active_build_project_id из JWT совместно с user_id. Роли и права также привязаны к проекту через таблицы user_build_project_settings и rules_for_user.

Управление источниками и файлами

Управление файлами реализовано в fs-manager-service (URL-префикс /fs_manager). Сервис отвечает за файловое хранилище: PostgreSQL хранит логическое дерево файлов, S3/MinIO хранит плоские зашифрованные блобы. Инфраструктура: PostgreSQL + S3 + Kafka.

fs-manager-service

REST endpoints

Endpoint	Метод	Назначение	Право
/fs_manager/api/files/	GET	Список файлов и папок (roots, list)	Список файлов
/fs_manager/api/files/upload	POST	Загрузка файла (multipart + progress)	Загрузка файла
/fs_manager/api/files/{id}	GET	Просмотр файла	Список файлов
/fs_manager/api/files/{id}	PATCH	Редактирование метаданных	Редактирование файла
/fs_manager/api/files/{id}	DELETE	Мягкое удаление	Удаление файлов
/fs_manager/api/files/update_file	POST	Замена содержимого файла	Замена файла
/fs_manager/api/files/download	POST	Скачивание (blob, с конвертацией)	Скачивание файла
/fs_manager/api/health/	GET	Healthcheck	—

Корневые директории

При запуске (on_startup) fs-manager открывает S3-клиент, проверяет существование бакета (ensure_bucket_exists) и создаёт корневые директории (ensure_root_directories):

Корень	Назначение
main_fs	Основное файловое дерево проекта
source_fs	Источники знаний
chat_result_fs	Результаты генерации (артефакты чатов)

> *Корневые директории являются системными и иммутабельными. Попытка изменения системного корня вызывает raise_if_system_root_immutable. Удаление системных корней запрещено.*

Логическое дерево файлов

PostgreSQL хранит логическое дерево в таблицах схемы public:

Таблица	Назначение
files	Логическое дерево: dir (флаг папки), dir_parent_id, root_dir_id, current_version_id, storage_key, size_bytes, content_type, owner user_id + build_project_id, deleted
file_versions	Цепочка версий: parent_id, first_version_id

S3 хранит плоские объекты по пути files/{file_id}/{version_id}/{filename}. Менеджер S3FilesManager оркестрирует согласованность БД + S3 + шифрование.

Версионирование

Файлы поддерживают версионирование: при замене содержимого создаётся новая версия в file_versions, связанная с предыдущей через parent_id и с первой версией через first_version_id. Текущая версия файла указывается в current_version_id.

Мягкое удаление

Удаление файлов выполняется мягко (soft-delete): устанавливается флаг deleted. Физическое удаление блобов из S3 не выполняется автоматически, что обеспечивает возможность восстановления.

Шифрование файлов

Файлы шифруются потоковым алгоритмом AES-256-GCM (envelope-шифрование) в покое (at rest) в S3. Параметры шифрования:

Параметр	Описание
FS_ENCRYPTION_ENABLED	Включение/выключение шифрования
FS_ENCRYPTION_MASTER_KEY	Мастер-ключ (32 байта)
FS_CHUNK_SIZE	Размер чанка шифрования (по умолчанию 65536)

Метаданные шифрования (magic-префикс, версия формата, wrapped DEK, размер чанка) упакованы в бинарный заголовок inline в начале тела зашифрованного объекта (ciphertext), а не в метаданных S3-объектов. Ключ шифрования должен совпадать в fs-manager, secretary и one-c-dit (все три сервиса имеют доступ к S3).

> ***Внимание:*** *утеря FS_ENCRYPTION_MASTER_KEY делает все зашифрованные файлы невозмещаемо недоступными. Ключ подлежит резервному копированию и хранению в защищённом хранилище секретов.*

Конвертация форматов

fs-manager поддерживает конвертацию форматов при скачивании (file_converter): единственное поддерживаемое преобразование — Markdown → DOCX (allowed_conversions = {('md', 'docx')}). Список

поддерживаемых текстовых и медиа-расширений задаётся переменными TEXT_EXT_FORMAT и MEDIA_EXT_FORMAT.

Источники знаний

Источники знаний (Sources) управляются в secretary-service (URL-префикс /secretary) и связаны с файлами fs-manager через таблицу sources_connector (схема secretary).

Модель Sources использует булевы поля (не enum-тип) для разграничения области видимости:

Флаг	**Описание**
cross=True	Источники, доступные во всех чатах проекта; управляются на /platform
tmp=True	Источники, подключённые к конкретному чату (связь через join-таблицу ChatSourcesUsed); управляются в правой боковой панели чата

> *Вычисляемое поле source_type принимает значения 'folder' или 'file' и указывает на природу источника (папка или файл), а не на область видимости.*

REST endpoints источников

Endpoint	**Метод**	**Назначение**	**Право**
/secretary/api/sources/	GET	Список источников (фильтры по типу, родителю, чату)	Список источников
/secretary/api/sources/	POST	Создание источника	Создание источника
/secretary/api/sources/{id}	GET	Просмотр источника	Список источников
/secretary/api/sources/{id}	PATCH	Редактирование	Редактирование источников
/secretary/api/sources/{id}	DELETE	Мягкое удаление	Удаление источников
/secretary/api/sources/search	GET	Поиск по имени и атрибутам	Поиск источников
/secretary/api/sources/{id}/download	POST	Скачивание файла источника (blob)	Скачивание источника
/secretary/api/sources/update_file	POST	Замена файла источника	Замена файла источника
/secretary/api/chats/{id}/sources	GET	Источники чата	Источники чата
/secretary/api/chats/{id}/upsert_sources	POST	Подключение/изменение активности	Подключение источника

UI источников

Управление источниками через веб-интерфейс:

- источники cross (доступны во всех чатах) — на странице /platform (правая боковая панель);
- источники tmp (привязанные к чату) — в правой боковой панели активного чата (/platform/chat/[id]).

Поддерживается drag-and-drop источников (react-dnd) и dropzone-загрузка файлов (react-dropzone).

Управление чатами, сообщениями и шаблонами

Управление чатами, сообщениями, функциональностями и шаблонами промптов реализовано в secretary-service (URL-префикс /secretary). Сервис содержит основную бизнес-логику «секретаря». Инфраструктура: PostgreSQL + S3 + Kafka. При запуске (on_startup) сидирует таблицы prompts, features, sources, templates из JSON-шаблонов.

secretary-service

Чаты

Endpoint	Метод	Назначение	Право
/secretary/api/chats/	GET	Список чатов	Список чатов
/secretary/api/chats/	POST	Создание чата (с инициализацией)	Создание чата
/secretary/api/chats/{id}	GET	Просмотр чата	Список чатов
/secretary/api/chats/{id}	PATCH	Редактирование (название, модель)	Редактирование чата
/secretary/api/chats/{id}	DELETE	Мягкое удаление	Удаление чата
/secretary/api/chats/search	GET	Поиск по названию и тексту сообщений	Поиск чатов
/secretary/api/chats/{id}/messages	GET	Сообщения чата	Сообщения чата
/secretary/api/chats/{id}/send_message	POST	Отправка сообщения + ответ ассистента	Отправка сообщения
/secretary/api/chats/{id}/features/templates	GET	Активные шаблоны фич	Шаблоны фич чата
/secretary/api/features/set_chat_template	POST	Смена шаблона фичи	Шаблон фичи в чате

> *Поле name в таблице chats имеет GIN trgm-индекс для быстрого поиска; таблица messages имеет GIN trgm-индекс на текст и композитный индекс (chat_id, created_time). Вложения хранятся в JSONB-поле attachments.*

Процесс отправки сообщения

Отправка сообщения инициирует асинхронную обработку через Kafka. Цепочка:

24. Фронтенд отправляет POST /secretary/api/chats/{id}/send_message (query-параметры: text, model_id, template_id, feature_id, attachments).
25. secretary-service сохраняет сообщение в messages, эмитит Kafka-событие в api-external (topic api_external_DSOutRequests, обработчик send_message), передавая user_data в заголовках.
26. api-external-service вызывает POST {DS_MODEL_URL}/agent/run (таймаут 3600 с), передавая заголовки x-api-key: DS_MODEL_API_KEY и X-User-Data-Jwt.

27. agent-server (DS) выполняет LLM tool-loop через MCP, возвращает ответ + использованные инструменты + S3-ключи + артефакты.
28. api-external возвращает ответ в secretary-service через Kafka (topic <service>_for_answer).
29. secretary-service сохраняет ответ ассистента в messages и возвращает HTTP-ответ клиенту.

> *Таймаут прокси nginx для send_message увеличен до 3600 с (secretary.conf). Чат синхронный: WebSocket-соединения в бэкенде отсутствуют.*

Фичи-модули

Функциональности (Features) — типы AI-функций, доступных пользователю. Список функциональностей возвращается endpoint GET /secretary/api/features/ (право «Список функциональностей»). Функциональности сидируются из JSON при запуске.

Функциональность	**Описание**
summarization	Суммаризация (генерация Markdown-документа)
question_answering	Вопрос-ответ с RAG
dashboard	Генерация HTML-дашборда
presentation	Генерация HTML-презентации

Шаблоны промптов

Шаблоны (Templates) — многошаговые пайплайны промптов. Структура хранения в схеме secretary:

Таблица	**Назначение**
templates	Шаблоны
steps_template	Шаги шаблона
steps_template_settings	Настройки шагов (выбранные промпты)
steps_template_prompt_rel	Связь шаг

/secretary/api/templates/set_default_flag	POST	Шаблон по умолчанию	Шаблон по умолчанию
/secretary/api/steps_template/	GET	Шаги шаблонов (фильтр по фиче)	Шаги шаблонов
/secretary/api/steps_template/{id}/prompts	GET	Промпты шага	Промпты шага
/secretary/api/steps_template/prompts	POST	Создание промпта шага	Создание промпта шага
/secretary/api/prompts/	GET	Промпты секретаря	Промпты секретаря
/secretary/api/prompts/{id}	PATCH	Редактирование промпта	Редактирование промпта
/secretary/api/prompts/{id}	DELETE	Удаление промпта	Удаление промпта
/secretary/api/prompts/search	GET	Поиск промптов	Поиск промптов

> *Двухстадийные модули (dashboard, presentation) имеют шаги main_prompt и planning_prompt. Промпт-шаблоны резолвятся через TemplatePromptService.resolve_for_tool(tool_name, template_id): запрос соединяет steps_template_settings, steps_template, features, public.prompts.*

UI чатов и шаблонов

Страница	**Назначение**
/platform	Стартовый экран чата: preset-карточки + feature templates
/platform/chat/[id]	Активный тред чата с историей сообщений
/platform/templates	Управление промпт-шаблонами

Ответы ассистента в чате могут содержать текст, сгенерированный документ (компонент Document), презентацию в формате PDF (Presentation) или PPTX (PptxPresentation).

Модули

Модули — специализированные процессы, не решаемые обычным чатом. Доступны через раздел Modules веб-интерфейса. Активным модулем является проводка УПД; модули КС-3 и протоколов помечены как устаревшие (deprecated).

Модуль УПД → 1С

Модуль проводки УПД реализован в one-c-dit-service (URL-префикс /one_c_dit). Сервис обрабатывает XML-документы и УПД с пошаговым контролем статусов, поиском расхождений с данными 1С и вариантами их разрешения. Инфраструктура: PostgreSQL + S3 + Kafka. При запуске открывает S3-клиент и проверяет бакет.

Схема БД

Схема one_c_dit:

Таблица	Назначение
xml	XML-документы; JSONB-поле document_fields
upd	УПД; FK на xml, enum upd_status
upd_problems_and_options	Проблемы и варианты (JSONB, 1:1 с upd)
steps_and_status	Шаги обработки; enum step_status, упорядоченные шаги per upd

REST endpoints

Endpoint	Метод	Назначение	Право
/one_c_dit/api/xml/	POST	Создание XML	Создание XML
/one_c_dit/api/xml/{id}	DELETE	Удаление XML	Удаление XML
/one_c_dit/api/xml/get_without_upd	GET	XML без созданного UPD	XML без UPD
/one_c_dit/api/upd/	GET	Список UPD	Список UPD
/one_c_dit/api/upd/	POST	Создание UPD на основе XML	Создание UPD
/one_c_dit/api/upd/{id}	GET	Просмотр UPD	Список UPD
/one_c_dit/api/upd/{id}	DELETE	Удаление UPD	Удаление UPD
/one_c_dit/api/upd/reconciliation_data	GET	Данные сверки с 1С	Расхождения UPD
/one_c_dit/api/upd/get_step_and_status	GET	Шаги и статусы	Шаги UPD
/one_c_dit/api/upd/update_step_status	PATCH	Обновление статуса шага	Обновление шага UPD

/one_c_dit/api/upd/choosing_divergence_options	POST	Применение вариантов решения	Решение расхождений
/one_c_dit/api/health/	GET	Healthcheck	—
/one_c_dit/api/health/health_one_c_and_dit_model	GET	Health 1C + DIT (публичный)	—

Процесс проводки УПД

30. Пользователь загружает XML-документ (POST /one_c_dit/api/xml/).
31. Создаёт УПД на основе XML (POST /one_c_dit/api/upd/); УПД связан с XML через RELATIONSHIPS.
32. Запрашивает данные сверки с 1С (GET /one_c_dit/api/upd/reconciliation_data): api-external вызывает 1С (POST /bgu/hs/data/goods, check_nomenclatures) и DIT-модель (POST /api/v2/workflows, get_options_problems) через Kafka.
33. Просматривает шаги и статусы (GET /one_c_dit/api/upd/get_step_and_status).
34. При наличии расхождений выбирает варианты решения (POST /one_c_dit/api/upd/choosing_divergence_options).
35. Обновляет статусы шагов (PATCH /one_c_dit/api/upd/update_step_status).

Интеграция с 1С

Вызовы 1С выполняются через api-external (Kafka topic api_external_OneCOutRequests):

Обработчик	**Метод 1С**	**Назначение**
health	GET /bgu/hs/data/health	Проверка доступности 1С
check_nomenclatures	POST /bgu/hs/data/goods	Проверка номенклатуры
get_catalog_nomenclature	GET /bgu/hs/data/catalog	Каталог номенклатуры
update_xml_document	—	Обновление XML-документа

Аутентификация в 1С: заголовок Authorization: ONE_C_AUTHORIZATION (Basic). URL 1С задаётся переменной ONE_C_URL (например, http://<IP-адрес-сервера-1С>).

Интеграция с DIT-моделью

Вызовы DIT-модели выполняются через api-external (Kafka topic api_external_DitOutRequests):

Обработчик	**Метод DIT**	**Назначение**
health	GET {DIT_MODEL_HEALTH_URL}	Проверка доступности DIT
get_options_problems	POST /api/v2/workflows	Поиск проблем и вариантов решения

Аутентификация в DIT: заголовок x-api-key: DIT_MODEL_DI_KEY. URL задаётся переменной DIT_MODEL_URL.

UI модуля

Страница	Назначение
/modules/jobs	Список работ УПД
/modules/jobs/new-form	Создание новой работы УПД

Компоненты UI: CreationProgress (ход выполнения), DataReconciliationCard (карточка сверки данных), SelectYpdCard (выбор УПД).

Модуль КС-3 (устаревший)

Модуль распределения КС-3 (акт выполненных строительных работ) помечен как **устаревший (deprecated)**. Реализован через сущность task (entity task, API-сервис ks3Api). Endpoints: /tasks, /tasks/{id}, /tasks/{id}/review, /tasks/{id}/files/ks3, /tasks/{id}/files/invoice, PUT /tasks/{id}/data.

UI:

Страница	Назначение
/modules/orchestrator	Список задач КС-3
/modules/orchestrator/tasks/[id]	Детали задачи
/modules/orchestrator/manual_processing/[id]	Ручная обработка КС-3

> *Модуль сохранён в кодовой базе, но не развивается. Использование в новых инсталляциях не рекомендуется.*

Модуль протокола/видео (устаревший)

Модуль суммаризации видео и протоколов совещаний помечен как **устаревший (deprecated)**. Реализован через сущность protocol (API-сервис protocolApi). Функциональность переносится в обычный чат.

Endpoints: /video_analysis_process (создание, multipart upload с прогрессом, транскрипция, суммаризация, дашборд), /prompts, /files/download, models.

UI:

Страница	Назначение
/modules/protocol	Список протоколов
/modules/protocol/generation/[id]	Генерация протокола

> *Модуль сохранён в кодовой базе, но не развивается. Каталог модулей (/modules) также помечен как устаревший.*

Модели искусственного интеллекта

Комплекс использует несколько моделей машинного обучения, обслуживаемых внешними инференс-серверами, совместимыми с OpenAI API. Все модели находятся вне репозиториев комплекса (отдельные vLLM/inference-серверы); ни весов, ни кода обучения, ни fine-tuning в поставке комплекса нет. Управление моделями осуществляется через переменные окружения.

Каталог моделей

general-service предоставляет справочник доступных LLM через endpoint GET /general/api/models/ (право «Список моделей»). Каталог сидируется из config/models_templates.json при запуске.

Модели DS-части

DS-часть использует четыре типа моделей. Все взаимодействуют через OpenAI-совместимый API.

Компонент	Модель	Endpoint env	Default host	API
Основной LLM	MiniMaxAI/MiniMax-M2.7	VLLM_BASE_URL	http://<IP-адрес-LLM-сервера>:8000/v1	/v1/chat/completions
Эмбеддинги	intfloat/multilingual-e5-large (1024-d)	EMBEDDING_ENDPOINT	http://<IP-адрес-LLM-сервера>:8044	/v1/embeddings
OCR	zai-org/GLM-OCR	OCR_ENDPOINT	http://<IP-адрес-сервера-OCR-ASR>:8041	/v1/chat/completions (vision)
ASR	Qwen/Qwen3-ASR-1.7B	ASR_ENDPOINT	http://<IP-адрес-сервера-OCR-ASR>:8017	/v1/audio/transcriptions

> *Имя модели передаётся per-request через поле model_id (alias model_name). В тестах используется альтернатива ai-forever/FRIDA для основного LLM.*

Основной LLM

Основная языковая модель (MiniMaxAI/MiniMax-M2.7) обслуживается vLLM и используется для чата, tool-loop и генерации артефактов. Поддерживаются:

- chat completions с OpenAI function-calling (tools, tool_choice);
- структурированный JSON-output (response_format json_schema / json_object с fallback repair);
- передача reasoning_content.

Параметры подключения:

Переменная	Описание	Default

VLLM_BASE_URL	URL vLLM	http://<IP-адрес-LLM-сервера>:8000/v1
VLLM_MODEL_NAME	Имя модели по умолчанию	MiniMaxAI/MiniMax-M2.7
VLLM_API_KEY	API-ключ (пустой = без Authorization header; непустой = Bearer token)	—

> *Пустой VLLM_API_KEY означает локальный vLLM без аутентификации; непустой — использование Bearer-токена.*

Эмбеддинги

Модель эмбеддингов intfloat/multilingual-e5-large (размерность 1024) используется для векторизации фрагментов документов и поисковых запросов. Применяется конвенция E5 prefix: passage: для индексации, query: для поиска. Размер батча — 64.

Переменная	Описание	Default
EMBEDDING_ENDPOINT	URL эмбеддинг-сервера	http://<IP-адрес-LLM-сервера>:8044
EMBEDDING_MODEL	Имя модели	intfloat/multilingual-e5-large
EMBEDDING_DIM	Размерность векторов	1024
EMBED_BATCH_SIZE	Размер батча	64
EMBED_TIMEOUT_S	Таймаут запроса	30

OCR

Модель OCR zai-org/GLM-OCR (vision LLM) распознаёт текст на сканированных страницах. Изображения передаются как base64 data URI в content-блоках image_url. Страницы рендерятся в PNG через pdf2image (poppler) с разрешением 300 DPI. Параллельное пакетное распознавание: размер батча 5, 3 повторные попытки с экспоненциальной задержкой (2/4/8 с). Максимум 500 страниц на документ.

Переменная	Описание	Default
OCR_ENDPOINT	URL OCR-сервера	http://<IP-адрес-сервера-OCR-ASR>:8041
OCR_MODEL	Имя модели	zai-org/GLM-OCR
OCR_BATCH_SIZE	Размер батча	5
OCR_PAGE_TIMEOUT_S	Таймаут страницы	60
OCR_MAX_PAGES	Максимум страниц	500
OCR_DPI	DPI рендеринга	300

Маршрутизация PDF (preprocess/router.py) классифицирует документ как pdf_text, pdf_scan или pdf_hybrid по тексту первых 3 страниц. Гибридные документы обрабатываются OCR только для страниц без текстового слоя.

ASR

Модель ASR Qwen/Qwen3-ASR-1.7B расшифровывает аудио и видео. Нормализация медиапотока выполняется ffmpeg: конвертация в mono 16 кГц PCM s16le WAV с фильтром highpass (70 Гц). WAV режется на 60-секундные чанки; параллельная транскрипция (семафор, до 16 одновременных фрагментов).

Переменная	**Описание**	**Default**
ASR_ENDPOINT	URL ASR-сервера	http://<IP-адрес-сервера-OCR-ASR>:8017
ASR_MODEL	Имя модели	Qwen/Qwen3-ASR-1.7B
ASR_CHUNK_DURATION_SEC	Длительность чанка	60
ASR_MAX_CONCURRENT	Параллелизм	16
ASR_TEMPERATURE	Температура	0.0
ASR_TIMEOUT_S	Таймаут	300

Управление endpoints

Управление подключениями к моделям осуществляется исключительно через переменные окружения. Для смены инференс-сервера (например, при переносе на другой GPU-узел) достаточно обновить соответствующую переменную (VLLM_BASE_URL, EMBEDDING_ENDPOINT, OCR_ENDPOINT, ASR_ENDPOINT) и перезапустить DS-сервисы.

Проверка доступности моделей:

```
# Основной LLM
curl -s http://<IP-адрес-LLM-сервера>:8000/v1/models

# Эмбеддинги (проверка через health)
curl -s http://<IP-адрес-LLM-сервера>:8044/health

# Через api-external (Kafka → DS health_model)
curl -s https://<gateway>/api_external/api/health/

# Комплексная проверка 1C + DIT
curl -s https://<gateway>/one_c_dit/api/health/health_one_c_and_dit_model
```

Интеграции

Интеграция с внешними системами реализована через api-external-service — единую точку исходящего трафика (egress). Бэкенд не вызывает внешние системы напрямую из бизнес-сервисов; все вызовы проходят через Kafka request/reply в api-external, который выполняет HTTP-запросы через requests_adapter.

api-external-service

api-external-service (URL-префикс /api_external) — оркестратор исходящих вызовов во внешние системы: DS model, DS RAG, 1C, DIT-модель. Инфраструктура: PostgreSQL + Kafka (без S3). Вся работа выполняется через Kafka-консьюмеры; REST ограничен публичным healthcheck.

Kafka-консьюмеры

Topic	Обработчики	Назначение
api_external_DSOutRequests	health_model, health_rag, send_message, get_chat_name, rag_init	Вызовы DS (требуют user_data)
api_external_OneCOutRequests	health, check_nomenclatures, get_catalog_nomenclature, update_xml_document	Вызовы 1C (без user_data)
api_external_DitOutRequests	health, get_options_problems	Вызовы DIT-модели (без user_data)

Каждый сервис слушает свой topic и отвечает в TOPIC_FOR_ANSWER (per-service <service>_for_answer). Паттерн request/reply реализован через KEvent.emit_and_wait_for_answer с временным ответным топиком answer_topic/<uuid>.

Интеграция с DS

api-external вызывает DS через HTTP с использованием клиентов ds_model.py и ds_rag.py.

DS model (agent-server)

Обработчик	Метод	Назначение	Таймаут
send_message	POST {DS_MODEL_URL}/agent/run	Основной контракт: запрос к агенту	3600 с
get_chat_name	POST {DS_MODEL_URL}/chat/title	Генерация заголовка чата	10 с
health_model	GET {DS_MODEL_URL}/health	Проверка доступности	—

Аутентификация: заголовок x-api-key: DS_MODEL_API_KEY + заголовок X-User-Data-Jwt (JWT JwtUserDataSchema без exp).

DS RAG (file-processing)

Обработчик	Метод	Назначение
rag_init	POST {DS_RAG_URL}/api/preprocess-and-index	Индексация загруженного файла
health_rag	GET {DS_RAG_URL}/health	Проверка доступности

Аутентификация: та же (x-api-key + X-User-Data-Jwt).

Интеграция с 1С

api-external вызывает 1С через HTTP-клиент one_c.py.

Обработчик	Метод 1С	Назначение
health	GET {ONE_C_URL}/bgu/hs/data/health	Проверка доступности 1С
check_nomenclatures	POST {ONE_C_URL}/bgu/hs/data/goods	Проверка номенклатуры
get_catalog_nomenclature	GET {ONE_C_URL}/bgu/hs/data/catalog	Каталог номенклатуры
update_xml_document	—	Обновление XML-документа

Аутентификация: заголовок Authorization: ONE_C_AUTHORIZATION (Basic). Вызовы 1С не передают user_data.

Параметры подключения:

Переменная	Описание	Default
ONE_C_URL	URL 1С	http://<IP-адрес-сервера-1С>
ONE_C_AUTHORIZATION	Basic-учётные данные	—

Интеграция с DIT-моделью

api-external вызывает DIT-модель через HTTP-клиент dit_model.py.

Обработчик	Метод DIT	Назначение
health	GET {DIT_MODEL_HEALTH_URL}	Проверка доступности DIT
get_options_problems	POST {DIT_MODEL_URL}/api/v2/workflows	Поиск проблем и вариантов решения УПД

Аутентификация: заголовок x-api-key: DIT_MODEL_DI_KEY. Вызовы DIT не передают user_data.

Параметры подключения:

Переменная	Описание	Default
DIT_MODEL_URL	URL DIT-модели	—
DIT_MODEL_HEALTH_URL	URL health DIT-модели	—
DIT_MODEL_DI_KEY	API-ключ DIT	—

Паттерн Kafka request/reply

Общий паттерн взаимодействия сервисов бэкенда с api-external:

36. Бизнес-сервис (secretary, one-c-dit) вызывает KEvent.emit_and_wait_for_answer с указанием topic и обработчика.
37. Создаётся временный ответный топик answer_topic/<uuid>.
38. api-external получает сообщение, выполняет HTTP-вызов во внешнюю систему.
39. api-external публикует ответ во временный топик.
40. Бизнес-сервис получает ответ и продолжает обработку.

Данные пользователя (JwtUserDataSchema) упаковываются в JWT без exp и передаются в заголовке user_data Kafka-сообщения. Для DS-вызовов api-external переупаковывает их в заголовок X-User-Data-Jwt.

Мониторинг и наблюдаемость

Мониторинг комплекса включает отслеживание ошибок (Sentry/Bugsink), проверки работоспособности (healthchecks), инспекцию Kafka, агрегированную OpenAPI-документацию и журналирование.

Отслеживание ошибок (Sentry / Bugsink)

Все сервисы бэкенда и DS интегрированы с Sentry SDK (версия 2.58.0). В качестве self-hosted Sentry-совместимого бэкенда используется Bugsink.

Переменная	Описание
SENTRY_DSN	DSN Sentry/Bugsink
SENTRY_ENVIRONMENT	Имя окружения (например, dev-240, testing)
SENTRY_RELEASE	Версия релиза (тег сборки)
SENTRY_SERVER_NAME	Имя сервера

Инициализация трассировки ошибок выполняется в start_up() каждого сервиса через init_error_tracking() (до запуска миграций). В openapi-aggregator включён PII capture.

Развёртывание Bugsink в K8s выполняется отдельным Helm chart bugsink-chart с использованием скрипта scripts/bugsink_provision_dsn.py для инициализации DSN-секретов Sentry по сервисам.

Healthchecks

Каждый сервис экспонирует публичный endpoint /api/health/ (auth=False), возвращающий статус сервиса. Формат глобального маршрута: https://<gateway>/<service_prefix>/api/health/.

Endpoint	Сервис	Назначение
/auth/api/health/	auth-service	Статус auth (+ проверка Redis auth:healthcheck)
/general/api/health/	general-service	Статус general
/fs_manager/api/health/	fs-manager-service	Статус fs-manager
/secretary/api/health/	secretary-service	Статус secretary
/api_external/api/health/	api-external-service	Статус api-external
/one_c_dit/api/health/	one-c-dit-service	Статус one-c-dit
/one_c_dit/api/health/health_one_c_and_dit_model	one-c-dit-service	Комплексная проверка 1С + DIT (публичный)
/health	nginx-шлюз	Статус шлюза (gated через auth_request, возвращает 200 ok)
/health	openapi-aggregator (порт 8099)	Статус агрегатора

GET /health	agent-server (порт 8100)	{"status":"ok","service":"agent-server"}
GET /health	tool-api (порт 8102)	Статус tool-api
GET /health	file-processing (порт 8103)	{status, version}

В Docker Compose healthcheck использует python-пробу urllib.request.urlopen('http://localhost:8080/api/health/'). В Kubernetes используются httpGet readiness/liveness пробы с initial delay 20–30 с.

Проверка зависимостей

При диагностике недоступности сервиса проверяйте зависимости:

```
# PostgreSQL
pg_isready -h <pg_host> -p 5432

# Redis
redis-cli -h <redis_host> -p 6379 ping

# Kafka
curl -s http://localhost:9999/api/clusters  # через kafka-ui

# S3/MinIO
curl -s http://localhost:9000/minio/health/live

# Qdrant
curl -s http://localhost:8104/healthz
```

kafka-ui

Web-консоль Kafka доступна на порту 9999 (контейнер provectus/kafka-ui в Docker Compose; в K8s — отдельный ресурс). Консоль позволяет:

- просматривать топики и consumer groups;
- инспектировать сообщения;
- мониторить лаги консьюмеров.

Основные топики для мониторинга: api_external_DSOutRequests, api_external_OneCOutRequests, api_external_DitOutRequests, ответные топики <service>_for_answer.

openapi-aggregator

openapi-aggregator (порт 8099) собирает /<prefix>/openapi.json каждого сервиса в единую OpenAPI 3.1 спецификацию. Источники задаются через переменную OPENAPI_SOURCES:

```
OPENAPI_SOURCES="auth=http://service-auth:8080/auth/openapi.json \
service_general=http://service-general:8080/general/openapi.json \
fs_manager=http://service-fs-manager:8080/fs_manager/openapi.json \
..."
```

Доступные endpoints:

Endpoint	Назначение
/openapi.json	Единая OpenAPI 3.1 спецификация
/docs	Swagger UI
/redoc	ReDoc
/health	Статус агрегатора

Журналирование

Сервисы пишут логи в stdout. Уровень логирования задаётся переменными LEVEL (бэкенд, по умолчанию DEBUG) и LOG_LEVEL (DS, по умолчанию INFO). Расширенное логирование включается переменной DEBUG_LOG.

В Kubernetes логи агрегируются платформой и сохраняются в NFS storage class. Включение NFS-хранилища: параметр storage.enabled; размер тома — logsPersistenceSize (по умолчанию 1Gi).

```
# Просмотр логов сервиса в K8s
kubectl -n platform logs -f deployment/<service>-service

# Просмотр логов в Docker Compose
docker compose -f docker-compose-startup.yml logs -f service-secretary
```

Резервное копирование и восстановление

Резервное копирование комплекса должно охватывать все хранилища данных: PostgreSQL, S3/MinIO, Qdrant и секреты конфигурации. Redis (сессии) не является критичным хранилищем: при потере данных сессии пересоздаются при следующем входе пользователя.

PostgreSQL

PostgreSQL — основная СУБД комплекса (схемы public, secretary, one_c_dit). Рекомендуется регулярное резервное копирование стандартными средствами.

Логическое резервное копирование

```
# Полный дамп базы
pg_dump -h <pg_host> -p 5432 -U <pg_user> -F c -d <db_name> -f backup_$(date +%Y%m%d).dump

# Восстановление
pg_restore -h <pg_host> -p 5432 -U <pg_user> -d <db_name> -C backup_YYYYMMDD.dump
```

Рекомендации

Аспект	Рекомендация
Частота	Ежедневно (полный дамп) + WAL-архивирование для PITR
Хранение	Не менее 7 ежедневных + 4 недельных + 12 ежемесячных копий
Хранилище резервных копий	Отдельное от БД (другой узел или объектное хранилище)
Тестирование	Периодическое восстановление в тестовое окружение для проверки

> *Миграции Alembic применяются при запуске сервиса (run_migrations). При восстановлении из дампа убедитесь, что версия дампа совместима с версиями миграций сервисов.*

S3 / MinIO

S3/MinIO хранит зашифрованные блобы файлов, кэш извлечения и артефакты. Рекомендуемые стратегии:

Стратегия	Описание
Версионирование бакета	Включить версионирование S3-бакета для защиты от случайного удаления/перезаписи
Репликация	Настроить репликацию в резервный бакет (другой узел или регион)

Снапшоты тома	Для self-hosted MinIO — снапшоты тома с данными (minio-data)
mc-клиент	Использовать mc mirror для периодического копирования

```
# Копирование бакета через mc
mc alias set src http://<minio_host>:9000 <access_key> <secret_key>
mc alias set dst http://<backup_minio>:9000 <access_key> <secret_key>
mc mirror --overwrite src/bucket dst/bucket-backup
```

> *Файлы зашифрованы AES-256-GCM. Для восстановления доступа к файлам требуется сохранённый FS_ENCRYPTION_MASTER_KEY. Резервное копирование мастер-ключа обязательно.*

Qdrant

Qdrant хранит векторные индексы (одна коллекция на user_id). Рекомендуется резервное копирование снапшотов.

```
# Создание снапшота коллекции
curl -X POST http://<qdrant_host>:8104/collections/{collection_name}/snapshots

# Восстановление: загрузить снапшот в директорию storage Qdrant
```

> *При потере индексов Qdrant данные могут быть восстановлены повторной индексацией исходных файлов через file-processing (POST /api/preprocess-and-index). Это требует доступности оригиналов в S3 и работоспособности моделей эмбеддингов/OCR/ASR.*

Redis

Redis хранит сессии auth и кэш verify. Данные не являются критичными: при потере Redis все активные сессии завершаются, и пользователи должны войти повторно. Специальное резервное копирование не требуется. При желании можно включить RDB-снапшоты стандартными средствами Redis.

Секреты конфигурации

Следующие секреты подлежат обязательному резервному копированию в защищённом хранилище (K8s Secret с резервированием, HashiCorp Vault и т. п.):

Секрет	**Назначение**	**Последствия утери**
JWT_SECRET_KEY	Подпись JWT	Все сессии недействительны; требуется перевыпуск
FS_ENCRYPTION_MASTER_KEY	Шифрование файлов	Все зашифрованные файлы недоступны
INTERNAL_AUTH_VERIFY_SECRET	Внутренний auth_request	Шлюз не сможет верифицировать сессии
FINGERPRINT_SECRET_SALT	Fingerprint сессий	Все сессии недействительны

S3_ACCESS_KEY / S3_SECRET_ACCESS_KEY	Доступ к S3	Нет доступа к файлам
POSTGRESQL_PASSWORD	Доступ к БД	Нет доступа к данным
ONE_C_AUTHORIZATION	Доступ к 1С	Модуль УПД неработоспособен
DIT_MODEL_DI_KEY	Доступ к DIT	Модуль УПД частично неработоспособен
DS_MODEL_API_KEY	Доступ к DS	Чат неработоспособен

> ***Особо критично:*** *утеря FS_ENCRYPTION_MASTER_KEY необратима. Зашифрованные файлы не могут быть расшифрованы без ключа. Храните ключ в нескольких защищённых местах.*

Устранение неполадок

В настоящем разделе приведены типовые проблемы и методы их диагностики.

Сервис не отвечает

Симптомы: сервис возвращает ошибки 502/504 или не отвечает на запросы.

Диагностика:

41. Проверьте healthcheck сервиса:

```
curl -s http://<service_host>:<port>/<prefix>/api/health/
```

42. Проверьте статус контейнера/пода:

```
# Docker Compose
docker compose -f docker-compose-startup.yml ps
docker inspect --format='{{.State.Health.Status}}' <container>

# Kubernetes
kubectl -n <namespace> get pods
kubectl -n <namespace> describe pod <pod-name>
```

43. Проверьте зависимости (PostgreSQL, Redis, Kafka, S3):

```
pg_isready -h <pg_host> -p 5432
redis-cli -h <redis_host> -p 6379 ping
curl -s http://<kafka_ui>:9999/api/clusters
curl -s http://<minio_host>:9000/minio/health/live
```

44. Проверьте логи сервиса:

```
docker compose -f docker-compose-startup.yml logs --tail=100 <service>
kubectl -n <namespace> logs <pod-name> --tail=100
```

Типичные причины: недоступность PostgreSQL (миграции не применены), недоступность Redis (auth), недоступность S3 (fs-manager, secretary, one-c-dit), недоступность Kafka (api-external, secretary).

Ошибки 401 / 403

Симптомы: клиент получает 401 Unauthorized или 403 Forbidden.

Диагностика:

45. Проверьте наличие cookie session_token (httpOnly).
46. Проверьте срок действия сессии (SESSION_TTL_SECONDS = 14400 с).
47. Проверьте совпадение JWT_SECRET_KEY в auth-service, api-external и Kafka-консьюмерах.
48. Проверьте совпадение INTERNAL_AUTH_VERIFY_SECRET в auth-service и конфигурации nginx.
49. Проверьте fingerprint сессии (совпадение User-Agent и IP клиента; FINGERPRINT_SECRET_SALT).
50. Проверьте права пользователя в активном проекте (JWT claim rights).

Типичные причины: истекла сессия (требуется повторный вход), несовпадение секретов между сервисами, смена IP/UA клиента (fingerprint), отсутствие права на запрашиваемый endpoint.

Ошибка отправки сообщения (send_message)

Симптомы: POST /secretary/api/chats/{id}/send_message возвращает ошибку или превышает таймаут.

Диагностика:

51. Проверьте доступность DS (agent-server):

```
curl -s http://<agent_server>:8100/health
```

52. Проверьте доступность api-external и его способность вызывать DS:

```
curl -s https://<gateway>/api_external/api/health/
```

53. Проверьте доступность основного LLM:

```
curl -s http://<vllm_host>:8000/v1/models
```

54. Проверьте переменные DS_MODEL_URL, DS_MODEL_API_KEY, VLLM_BASE_URL.
55. Проверьте Kafka: наличие сообщений в topic api_external_DSOutRequests, лаг консьюмеров, ответные топики.
56. Проверьте таймаут прокси nginx для send_message (должен быть 3600 с в secretary.conf).

Типичные причины: DS недоступен (agent-server не запущен), LLM недоступен (vLLM не отвечает), DS_MODEL_API_KEY неверен, Kafka-консьюмер api-external не обрабатывает сообщения, превышен таймаут генерации.

Ошибка загрузки файла

Симптомы: POST /fs_manager/api/files/upload возвращает ошибку.

Диагностика:

57. Проверьте доступность S3/MinIO:

```
curl -s http://<minio_host>:9000/minio/health/live
```

58. Проверьте существование бакета и корневых директорий (main_fs, source_fs, chat_result_fs).
59. Проверьте переменные S3_BUCKET_NAME, S3_ENDPOINT_URL, S3_ACCESS_KEY, S3_SECRET_ACCESS_KEY.
60. Проверьте переменные шифрования FS_ENCRYPTION_ENABLED, FS_ENCRYPTION_MASTER_KEY (должны совпадать в fs-manager, secretary, one-c-dit).
61. Проверьте client_max_body_size в nginx (должен быть 2000m).

Типичные причины: S3 недоступен, бакет не существует, неверные учётные данные S3, несовпадение FS_ENCRYPTION_MASTER_KEY между сервисами, превышен лимит размера загрузки.

Ошибка модуля УПД

Симптомы: операции модуля УПД возвращают ошибку.

Диагностика:

62. Проверьте доступность 1С:

```
curl -s -H "Authorization: <ONE_C_AUTHORIZATION>" http://<one_c_url>/bgu/hs/data/health
# Или через комплексную проверку:
curl -s https://<gateway>/one_c_dit/api/health/health_one_c_and_dit_model
```

63. Проверьте доступность DIT-модели:

```
curl -s -H "x-api-key: <DIT_MODEL_DI_KEY>" http://<dit_health_url>
```

64. Проверьте переменные ONE_C_URL, ONE_C_AUTHORIZATION, DIT_MODEL_URL, DIT_MODEL_DI_KEY.
65. Проверьте Kafka: топики api_external_OneCOutRequests, api_external_DitOutRequests.

Типичные причины: 1С недоступна, неверные учётные данные 1С (ONE_C_AUTHORIZATION), DIT-модель недоступна, неверный DIT_MODEL_DI_KEY, Kafka-консьюмер не обрабатывает сообщения.

Проблемы Kafka

Симптомы: межсервисные вызовы не выполняются, сообщения накапливаются, растёт лаг консьюмеров.

Диагностика:

66. Откройте kafka-ui (порт 9999) и проверьте: - наличие топиков (api_external_DSOutRequests, api_external_OneCOutRequests, api_external_DitOutRequests, <service>_for_answer); - consumer groups и их лаги; - наличие сообщений в топиках.
67. Проверьте переменные KAFKA_SERVER, KAFKA_CONSUMER_GROUP_ID, KAFKA_SASL_*.
68. Проверьте логи консьюмеров (api-external).

Типичные причины: отсутствует топик (требуется создание), консьюмер не зарегистрирован в группе, несовпадение KAFKA_CONSUMER_GROUP_ID, ошибки аутентификации SASL, превышение размера сообщения.

Проблемы миграций БД

Симптомы: сервис не запускается, ошибки в логах при run_migrations.

Диагностика:

69. Проверьте доступность PostgreSQL и учётные данные (POSTGRESQL_*).
70. Проверьте логи сервиса на этапе run_migrations(POSTGRESQL_URI_SYNC, schema_name=...).
71. Убедитесь, что миграции схемы public применяются первыми (порядок: public, затем схема сервиса).
72. Проверьте версии миграций Alembic в postgresql_adapter/alembic_config/<schema>/versions/.

Типичные причины: недоступность БД, неверные учётные данные, рассинхронизация версий миграций, нарушение порядка применения схем.

Краткая справочная таблица

Проблема	Первичная проверка	Переменные / ресурсы
Сервис не отвечает	healthcheck, зависимости	/api/health/, pg/redis/kafka/s3
401 / 403	cookie, fingerprint, права	JWT_SECRET_KEY, INTERNAL_AUTH_VERIFY_SECRET, FINGERPRINT_SECRET_SALT
send_message ошибка	DS health, LLM, Kafka	DS_MODEL_URL, DS_MODEL_API_KEY, VLLM_BASE_URL
Загрузка файла ошибка	S3 health, бакет, шифрование	S3_*, FS_ENCRYPTION_MASTER_KEY
УПД ошибка	1C health, DIT health	ONE_C_URL, ONE_C_AUTHORIZATION, DIT_MODEL_*
Kafka проблема	kafka-ui, топики, лаги	KAFKA_*, consumer groups
Миграции БД	pg, версии Alembic	POSTGRESQL_*, alembic_config/<schema>/versions/