

Разработка

Документация для разработчиков: сервисы, базы, интеграции. Сотрудникам отделов — в книге AmoCrm.

- [Ядро данных](#)
 - [Реплика amoCRM \(amo_db\)](#)
 - [Как ходить в amoCRM из своего сервиса](#)
 - [Вход в виджеты amoCRM](#)
- [Оценка звонков](#)
 - [Как устроена оценка звонков](#)
- [Боты и уведомления](#)
 - [Приём заявок: bug_hunter](#)
 - [Релей GlitchTip в Telegram](#)
- [Сервисы вокруг amoCRM](#)
 - [Дедупликация сделок: anti_double_deals](#)
- [Виджеты amoCRM](#)
 - [Встречи Контур.Толка из сделок: contur_talk_widget](#)
- [Лендинги](#)
 - [Квиз-лендинги](#)
- [Инфраструктура](#)
 - [Мониторинг и автоматизация серверов: infra](#)
 - [Телеметрия использования: usage_telemetry](#)
- [Внутренние сервисы](#)

- [SMS-шлюз: beeline-service](#)
- [Воронка найма: hiring_project](#)
- [Интеграции](#)
 - [Отчёты по встречам Контур.Толка: contur_transcribe](#)
 - [YouGile и GitHub: yogile_connector](#)
- [Продукты](#)
 - [Личный кабинет Юмобайл: web-app-umoible](#)
 - [WebRTC-телефония: pbx](#)

Ядро данных

Реплика amoCRM и шлюзы, через которые сервисы ходят в CRM.

Реплика amoCRM (amo_db)

Для разработчиков. Подробности устройства — в README репозитория `Buharestskie/amo_db`.

Что это

amo_db — копия нашего аккаунта amoCRM в обычной PostgreSQL. Сделки, контакты, задачи, примечания, неразобранное, каталоги, справочники и журнал событий. Изменение из amoCRM попадает в базу за несколько секунд.

Если сервису нужно **читать** данные amoCRM, он читает их отсюда, а не из API. У API лимит в 7 запросов в секунду на интеграцию и 50 на весь аккаунт — его делят все наши сервисы, виджеты и телефония, и когда он кончается, amoCRM сначала отвечает ошибками, а потом блокирует аккаунт. Из базы можно выбирать что угодно и сколько угодно.

Писать в amoCRM через базу нельзя: у всех ролей только чтение. Запись — как и раньше, через API amoCRM.

Как получить доступ

1. Попросить администратора завести роль с доступом «только чтение» к базе `amo_db`. Роли выдаются через панель pg_provisioner, пароль показывается один раз.
2. Попросить внести в список доступа внешний IP сервера, с которого будет подключение. Для своей машины разработчику достаточно запустить `./scripts/dev-db-access.sh` из репозитория amo_db — скрипт сам определит адрес. Если адрес часто меняется, есть туннель поверх SSH: `./scripts/db-tunnel.sh`.
3. Подключаться к `db.donula26.ru`, порт **6432**, база `amo_db`.

Порт 6432 — это пулер соединений. Он возвращает соединение в общий пул после каждой транзакции, поэтому не работают `LISTEN/NOTIFY`, временные таблицы и сессионные `SET`. Если приложение на asyncpg, кэш prepared statements нужно выключить (`statement_cache_size=0`), иначе под нагрузкой начнут плавать ошибки `prepared statement ... does not exist`.

Четыре правила чтения

Правило	Почему
Фильтровать <code>is_deleted = false</code>	Удалённое в amoCRM здесь не стирается, а помечается. Без фильтра в выборку попадут удалённые сделки и контакты.
В справочниках брать <code>is_current = true</code>	Статусы, воронки, поля, пользователи хранятся с историей версий: переименованный статус остаётся расшифровываемым в старых сделках. Для текущего названия нужна текущая версия.
Кастомные поля — из <code>amo_custom_field_values</code>	Там значения развёрнуты построчно, по ним удобно фильтровать. Разбирать JSONB в <code>amo_leads.custom_fields</code> не нужно.
Поиск контакта по телефону — через <code>amo_phone_tails(phones)</code>	<code>phone_normalized</code> — только первый телефон контакта. Поиск по любому номеру: <code>WHERE amo_phone_tails(phones) && ARRAY['9001112233']</code> , номер — последние 10 цифр.

Главные таблицы

Таблица	Что в ней
<code>amo_leads</code>	Сделки
<code>amo_contacts</code>	Контакты
<code>amo_tasks</code>	Задачи
<code>amo_notes</code>	Примечания: звонки, SMS, превью писем. Переписки из мессенджеров здесь нет
<code>amo_links</code>	Связи между сущностями — например, какие контакты у сделки
<code>amo_events</code>	Журнал изменений аккаунта: смены этапов, ответственных, удаления
<code>amo_statuses</code> , <code>amo_pipelines</code> , <code>amo_users</code> , <code>amo_custom_fields</code>	Справочники, которыми расшифровываются id в сделках

Все таблицы поле за полем — в `MODELS.md` в репозитории.

Чего в базе нет

- **Переписки из мессенджеров.** Схема под неё готова, наполнение отложено.
- **Компаний** отдельной таблицей. `company_id` в сделках и контактах есть.
- **Покупателей и сегментов** — в аккаунте они выключены.

Как понять, что данные свежие

`https://replika.donula26.ru/sync/state` показывает каждый поток синхронизации. Смотреть на `last_success_at` и `last_error`: если успех был в последние минуты и ошибки нет, поток работает. На `lag_seconds` не ориентироваться — у удалений и каталогов он часами растёт просто потому, что меняться нечему.

Если поток встал, мониторинг сам пишет в Telegram.

Как ходить в amoCRM из своего сервиса

Для разработчиков. Подробности устройства — в README репозитория `Buharestskie/amo_queue`.

Коротко

Если ваш сервис **читает** данные amoCRM — читайте их из реплики, см. «Реплика amoCRM (amo_db)». Если сервис **пишет** в amoCRM или ему нужно то, чего в реплике нет, — ходите через шлюз `https://amo-queue.donula26.ru`, а не напрямую в amoCRM.

Зачем шлюз

amoCRM пускает не больше 7 запросов в секунду от одной интеграции и не больше 50 от всего аккаунта. Превышение — ошибка `429`, повторные превышения — блокировка всего аккаунта: встают все виджеты, дашборды и телефония разом.

Каждый сервис по отдельности может аккуратно держать свои 7 запросов, но про соседей он не знает — и вместе они пробивают общие 50. Шлюз ставит запросы всех сервисов в одну очередь и следит за обоими лимитами сразу.

Как подключиться

Поменять в конфиге сервиса один адрес — больше ничего:

Было	Стало
<code>https://donula.amocrm.ru/api/v4/leads?with=contacts</code>	<code>https://amo-queue.donula26.ru/api/v4/leads?with=contacts</code>

Пути, методы, тело запроса и заголовок `Authorization: Bearer <токен>` остаются прежними. Регистрировать сервис в шлюзе не нужно: он отличает интеграции по токену.

Обязательно передавайте имя сервиса в заголовке `X-Integration-Name`:

```
curl https://amo-queue.donula26.ru/api/v4/account \
  -H "Authorization: Bearer $AMO_TOKEN" \
  -H "X-Integration-Name: my-service"
```

Без имени в статистике и алертах виден только кусок хэша токена, и понять, чей это сервис, уже нельзя. Имя — латиница, цифры, точка, дефис, подчёркивание, до 40 символов.

Что может ответить шлюз

Почти всегда вы получаете ответ самой amoCRM без изменений. Свои ответы шлюз отдаёт в четырёх случаях:

Код	Что значит	Что делать
401	Нет заголовка <code>Authorization</code>	Передать токен
429 с телом <code>gateway queue is full</code>	Ваш сервис прислал больше, чем его очередь разгребает за 10 секунд	Повторить через <code>Retry-After</code> и растянуть отправку во времени. Это не amoCRM и не сбой шлюза — это ваш сервис шлёт пачкой
502	amoCRM недоступна	Повторить позже
504	amoCRM не ответила за 30 секунд	Повторить позже; если повторяется — упростить запрос

Сколько можно слать

На один токен — 6 запросов в секунду, на все сервисы вместе через шлюз — 30. Оставшиеся до 50 — запас для тех, кто в amoCRM ходит напрямую: реплика, телефония, сторонние виджеты.

Запросы сверх 6 в секунду не отбрасываются сразу, а ждут своей очереди до 10 секунд. Поэтому короткие всплески проходят незаметно, а массовые выгрузки — нет: их надо отправлять равномерно или разбивать на части с паузами.

Если фоновая выгрузка мешает быстрым запросам того же сервиса (кнопки в виджете начинают тормозить), заведите для фоновой задачи отдельную интеграцию в amoCRM со своим токеном — у неё будет своя очередь.

Лендинги ходят не сюда

Бэкенды квиз-лендингов (mfctest, bfl_fspp, do-nula-refinancing-2 и подобные) отправляют заявки не через шлюз, а через отдельный сервис `amocrm-service` — `https://donula26.ru/service/amocrm/`. Причины:

- **Токен amoCRM не лежит на лендинге.** Лендинг присылает готовую сделку, контакт и примечание, а токен знает только `amocrm-service`.
- **Защита от спама с форм.** Не больше 9 заявок с одного IP посетителя в час и 10 сущностей на один телефон в сутки. Поэтому в каждом запросе лендинг передаёт `ip` посетителя и его `phone`.
- **Заявка не теряется, если amoCRM тормозит.** `POST /leads/async` ставит цепочку «контакт → сделка → примечание» в очередь и сразу отвечает посетителю; дальше её доводит воркер.

Пускают туда только известные адреса. Новый лендинг — это новая строка в списке доступа сервиса: попросите администратора добавить его до запуска, иначе на каждую заявку будет ответ `403 Caller IP is not permitted`. Всё остальное — ручки, коды ответов, что делать при сбое — в README репозитория `Buharestskie/amocrm-service`.

Как посмотреть, что с вашим сервисом

- `https://amo-queue.donula26.ru/stats` — сейчас: темп, очередь и счётчики ответов по каждому сервису. Главное число — `queue_seconds`: сколько секунд ждёт запрос, пришедший прямо сейчас. Около нуля — всё хорошо.
- `https://amo-queue.donula26.ru/stats/history?hours=24` — то же по часам, до 7 суток назад.

Если у вашего сервиса растёт `queue_full` в счётчиках — он шлёт пачками, и часть запросов до amoCRM не доходит.

Вход в виджеты amoCRM

Для разработчиков виджетов. Подробности устройства — в README репозитория [Buharestskie/widget_auth](https://github.com/Buharestskie/widget_auth).

Зачем отдельный сервис

Виджет живёт внутри интерфейса amoCRM и своему бэкенду может предъявить только одно — токен `X-Auth-Token`, который amoCRM подписывает секретным ключом интеграции. Если проверять этот токен в каждом бэкенде, секрет интеграции окажется во всех виджетах сразу, и утечка из любого из них откроет все.

Поэтому секрет хранится в одном месте — в сервисе `widget_auth` на <https://widget-auth.donula26.ru>. Он проверяет токен amoCRM и выдаёт взамен свой, короткоживущий, подписанный отдельным секретом конкретного бэкенда. Бэкенд виджета секрета интеграции не видит никогда.

Сейчас так входят «Файлы сделки на Яндекс.Диске», чек-листы и шаблоны сообщений.

Как это выглядит со стороны виджета

1. Виджет вызывает обмен через `$authorizedAjax` — amoCRM сама подставит `X-Auth-Token`:

```
self.$authorizedAjax({
  url: 'https://widget-auth.donula26.ru/api/auth/exchange',
  method: 'POST'
}).done(function (response) {
  // response.access_token, response.expires_in (600 секунд)
});
```

2. Полученный `access_token` виджет передаёт своему бэкенду в заголовке `Authorization: Bearer <access_token>`.

3. Через 10 минут токен истекает — бэкенд ответит `401`, и виджет должен заново получить токен.

Как бэкенд проверяет токен

Токен — JWT, алгоритм HS256. Проверять обязательно все четыре вещи:

Что	Чем
Подпись	<code>ACCESS_TOKEN_SECRET</code> — свой у каждого бэкенда
Аудитория <code>aud</code>	<code>ACCESS_TOKEN_AUDIENCE</code> — своя у каждого бэкенда, например <code>yandex-widget-backend</code>
Издатель <code>iss</code>	<code>ACCESS_TOKEN_ISSUER=amocrm-widget-auth-service</code> — общий
Срок <code>exp</code> , <code>nbf</code>	Библиотека JWT проверяет сама; допуск на расхождение часов — несколько секунд

```
payload = jwt.decode(token, ACCESS_TOKEN_SECRET, algorithms=["HS256"],
                      audience=ACCESS_TOKEN_AUDIENCE, issuer=ACCESS_TOKEN_ISSUER, leeway=5)
```

Кто пришёл — в полях `account_id` и `user_id` (то же в `sub` как `"account_id:user_id"`). Это id пользователя amoCRM — по нему и решайте, что ему можно.

Не пропускайте проверку `aud`: именно она не даёт токену одного виджета пройти в бэкенд другого.

Подключить новый виджет

Делает администратор, у которого есть доступ к серверу:

1. В настройках интеграции amoCRM взять её ID (`client_uuid`) и секретный ключ.
2. Добавить виджет в список `widget_auth`: ID и секрет интеграции, новый случайный секрет бэкенда (от 32 символов) и уникальную аудиторию для него. Перезапустить сервис — список читается только при старте.
3. В настройки бэкенда виджета вписать тот же секрет бэкенда, ту же аудиторию и общий издатель.

Порядок команд и формат записи — в README репозитория, раздел «Подключить новый виджет».

Если вход не работает

Ответ	Что проверить
<code>401 Missing X-Auth-Token</code> от <code>widget_auth</code>	Запрос идёт не через <code>\$authorizedAjax</code> , а обычным <code>\$.ajax</code> или <code>fetch</code> — amoCRM подставляет токен только в первом случае
<code>401 Invalid X-Auth-Token</code> от <code>widget_auth</code>	Виджета нет в списке, неверный секрет интеграции или не совпала аудитория входящего токена. Обычно ошибка в только что добавленной записи
<code>401</code> от своего бэкенда	Токен истёк (прошло больше 10 минут) или секрет / аудитория в бэкенде не совпадают с записью в <code>widget_auth</code>
Ошибка CORS в консоли браузера	Виджет открыт не в нашем аккаунте amoCRM — обмен разрешён только со страниц <code>donu1a.amocrm.ru</code>

Оценка звонков

Как устроена оценка ЗВОНКОВ

Для разработчиков. Состояние на 24.09.2026. Подробности — в README репозитория `Buharestskie/amocrm_transcribe_service` и `Buharestskie/yandex_transcribe_service`.

Коротко

Все дашборды — МПП2, МОС и МПП1 — расшифровывают и оценивают звонки через один сервис: **amocrm_transcribe_service** (в конфигах дашбордов он называется «S2T apiscrm»). Он отдаёт запись в распознавание речи A5X, раскладывает реплики на менеджера и клиента и оценивает разговор языковой моделью A5X по промπτу, который прислал дашборд. Критерии оценки живут не в сервисе, а в «Настройках» каждого дашборда.

Путь звонка

1. **Звонок попадает в дашборд** из телефонии — Sipuni, МегаФон или своя PBX: вебхуком или синхронизацией истории.
2. **Дашборд решает, что оценивать**, по своим правилам: МПП2 — серия звонков после входа сделки в «Принимают решение», МОС — звонки по касаниям, рискам и рекомендациям, МПП1 — звонки охотников. Звонки одного клиента за день склеиваются в серию.
3. **Запрос в сервис** — `POST /internal/transcribe-analyze` на внутренний порт сервиса на `global` (адрес — в README): ссылки на записи, тип телефонии у каждой, промπτ и адрес для ответа. Сервис сразу отвечает `202`.
4. **Распознавание** — запись целиком уходит в `s2t.a5x.pro`. Левый канал становится `SPEAKER_00`, правый — `SPEAKER_01`.
5. **Роли** — кто из них менеджер, сервис решает по типу телефонии, не по тексту (таблица ниже).
6. **Оценка** — сегменты уходят в `llm.a5x.pro` (модель `qwen36-35b-a3b`): промπτ дашборда — системным сообщением, разговор — в виде `[секунда роль] текст`.
7. **Ответ** — сервис присылает дашборду обратный вызов: расшифровку, ответ модели (сырой и разобранный JSON) и расход — секунды аудио и токены. Дашборд сохраняет разбор и показывает его: МПП2 — на вкладке «Качество» и в виджете

«Оценка разговоров» в карточке сделки, МОС — в касаниях, рисках и рекомендациях, МПП1 — в карточке звонка.

Дашборд	Куда приходит ответ	Где промпты
МПП2	<code>/api/transcriptions/callback</code>	Настройки: две консультации, касания, тёплый пул
МОС	<code>/api/telephony/webhooks/transcribe</code>	Настройки: касания, риски, рекомендации
МПП1	<code>/api/v1/internal/transcribe/callback</code>	Настройки: гейт и QA-аудитор

Каналы по телефониям

Телефония	Менеджер	Клиент
Sipuni	правый канал	левый
МегаФон	левый	правый
Своя PBX	правый	левый

Запись должна быть стерео: моно-запись получает ошибку, роли не угадываются. Подключаете новую телефонию — сначала строка в этой таблице (`src/speaker_mapping.py`), иначе менеджер и клиент перепутаются во всех разборах.

Резерв: Yandex

yandex_transcribe_service (тоже на `global`) — распознавание SpeechKit и оценка YandexGPT в два этапа (гейт, затем полный разбор). Раньше был основным, с 17.09.2026 рабочих запросов нет: остался кнопкой ручной перерасшифровки «через Yandex» в МПП2 и МПП1. Каталог Yandex Cloud при этом живой и платный.

Ещё два репозитория — `assembly_transcription` (AssemblyAI) и `gemini_service` (Gemini) — нигде не развёрнуты.

Если оценки не приходят

- 1. **Жив ли сервис:** с `global` — `curl -s localhost:8083/health`.
- 2. **Доходят ли запросы:** `docker logs --since 1h amocrm_transcribe_service-apiscrm-transcribe-1 | grep transcribe-analyze`.

3. **Ошибки** — в GlitchTip, проект сервиса, и в обратных вызовах с полем ошибки: их дашборд показывает у звонка.
4. **Сервис перезапускали?** Задачи в работе живут в памяти и при перезапуске теряются. МОС повторяет звонок без ответа через час (потом через 30 минут и 2 часа, всего четыре попытки), МПП1 — задачей раз в 20 минут.
5. **Ответ пришёл, а разбора нет** — модель ответила не JSON: смотреть сырой `analysis_text` в данных звонка и промпт в «Настройках».

Что стоит денег

Каждый разбор — минуты распознавания A5X и токены модели. Ключ у распознавания и модели сейчас один: отзовут — встанут оба этапа. Правка промпта в «Настройках» действует только на новые звонки; старые заново не оцениваются.

Боты и уведомления

Приём заявок: bug_hunter

Для разработчиков. Состояние на 24.09.2026. Как пользоваться ботом сотрудникам — статья «Как сообщить об ошибке». Подробности и запуск — README `Buharestskie/bug_hunter_bot`.

Коротко

bug_hunter — приём заявок в разработку: сотрудник заводит ошибку или задачу в мини-приложении Telegram, заявка уходит ответственному за сервис, встаёт в общую очередь и связывается с задачей в YouGile. Разработчик работает в админке, в мини-приложении, в YouGile или через MCP-сервер из Claude Code — статус и переписка синхронизируются.

Где живёт

Сервер	<code>ulrica</code> (Алматы — оттуда доступен Telegram); каталог и контейнеры — в README
Адрес	<code>https://bug-hunter.donula26.ru</code> (nginx на сервере)
Бот	<code>@donula_bug_hunter_bot</code> , вебхук Telegram
Админка	<code>/admin</code> — вход через Telegram, только ответственные; запасной вход — <code>/start login</code> в боте
Мини-приложение	<code>/app</code> — всем, кроме забаненных

Один процесс FastAPI: вебхук Telegram, API, `/mcp`, `/yougile/webhook` и статика. Данные — Postgres, черновики ввода в чате — Redis, вложения — диск `/data/media`. Фронт собирается на машине разработчика и заливается готовым: Node на сервере нет. Миграции накатываются на старте контейнера.

Заявка: путь и статусы

Статус	Смысл
Новая	ждёт разбора ответственным
На согласовании	задача ждёт согласующего; в очереди не стоит

Статус	Смысл
В очереди	принята, ждёт исполнителя
В работе	кто-то взял
Нужны детали	задан вопрос автору; ответ автора возвращает прежний статус
Готово	сделано; автор задачи может вернуть в работу один раз за 14 дней
Отклонена / Не баг / Дубль / Отклонена при согласовании / Отозвана	закрыта без выполнения

Куда встаёт новая задача: автор с правом согласования — сразу в очередь; «незнакомый» автор (ни одной разобранной заявки) — в «Новая», решает ответственный; остальные — на согласование, если есть согласующий уровнем выше, иначе в «Новая».

Очередь: срочность (Вне очереди → Срочно → В ближайшее время → Не срочно), затем уровень автора (только у задач), затем дата. У ошибки срочности нет, пока её не назначит ответственный. Объединённые заявки занимают одно место.

Кому уходит новая заявка: ответственным за сервис; если их нет — всем ответственным с пометкой «Ответственный за сервис не назначен». Задачи на согласовании не уведомляют никого — у согласующих только счётчик в приложении.

Переписка двигает статус: не внутреннее сообщение разработчика переводит заявку в «Нужны детали» (даже из «В работе»), ответ автора возвращает её обратно. Внутренние заметки статус не трогают и автору не видны.

Объединение: присоединённые заявки получают статус и задачу YouGile главной, смена статуса и срочности идёт на всю группу, автору — одно уведомление на все его заявки группы.

Каталог сервисов

Каталог — репозитории GitHub организации `Buharestskie` (сверка раз в час) плюс сервисы, заведённые руками. Новый репозиторий появляется **скрытым**: админам приходит «В GitHub появились новые репозитории... Задайте им названия, иначе они не будут видны в боте». Пока в админке «Сервисы» ему не дали человеческое имя и не включили видимость, сотрудники его не выберут. Архивные репозитории из бота пропадают сами. Там же назначаются ответственные за сервис — от этого зависит, кому уходят заявки.

YouGile

- **Когда заводится задача:** у ошибки — кнопкой «Создать задачу» в карточке, через MCP или переносом карточки на нашей доске в связанную колонку; у задачи — автоматически после согласования или «Принять в очередь».
- **Содержимое:** заголовок «[Сервис] кратко», описание — поля заявки, ссылки на вложения и на карточку; стикеры «Репозиторий» и «Срочность». Исполнитель — кто ведёт заявку, иначе ответственные за сервис, иначе нажавший кнопку. «Создал» в YouGile — всегда владелец API-ключа.
- **От нас в YouGile:** «Готово» — в завершающую колонку с отметкой выполнения; отказы — туда же без отметки и с комментарием причины; «В работе» — в рабочую колонку.
- **Из YouGile к нам:** вебхук (`task-moved` , `task-updated` , `task-deleted` , `chat_message-created`) и свёрка каждые 10 минут. Статус берётся из колонки: у каждой колонки доски в админке задано, какой статус она ставит. Удалили задачу в YouGile — открытые заявки становятся «Отклонена».
- **Комментарии из YouGile** попадают в заявку, только если у автора комментария в «Людах» указан сотрудник YouGile, и только через вебхук — свёрка их не догоняет.
- Доска на проде — «Новая доска» проекта «Разработка-2»: Заблокировано · Бэклог · Ожидает информации · В работе · на проверке · Готово.

MCP-сервер

POST `https://bug-hunter.donula26.ru/mcp` , JSON-RPC, заголовок `Authorization: Bearer <MCP_TOKEN>` . Действует от имени первичного админа с наименьшим Telegram ID из `ADMIN_TELEGRAM_IDS` .

Инструмент	Что делает
<code>list_bugs</code> , <code>get_bug</code> , <code>get_queue</code>	список, заявка целиком с перепиской и историей, очередь с местами
<code>set_status</code>	статус группы: уведомления автору, перенос в YouGile
<code>set_urgency</code>	срочность группы, стикер в YouGile
<code>route_task</code>	задача «незнакомца»: принять или отправить на согласование
<code>merge_bugs</code> , <code>unmerge_bug</code>	объединить дубли / отсоединить
<code>add_note</code>	сообщение в заявку, по умолчанию внутреннее
<code>create_yougile_task</code>	завести задачу в YouGile
<code>list_services</code>	каталог сервисов

MCP ведёт себя не во всём как API: `add_note` с `internal=false` не ставит «Нужны детали» и не уходит в YouGile; `set_status` «В работе» для уже взятой заявки молча ничего не делает (API ответил бы 409).

На что наступают

- **Прод отстаёт от `main`**: на сервере `7f7966a`, не выкачен `837f6c3` (сверка с YouGile перестала откатывать статусы). До выкатки сверка может перезаписать «Нужны детали» и отказы статусом колонки.
- **Ошибки самого бота видны только в логах контейнера приложения.** GlitchTip подключён в коде, но `SENTRY_DSN` на проде не задан. Заявка, которую некому отправить, тоже только в логе.
- **«В работе», пришедшее из YouGile, автору не сообщается** — только «готово», «отклонено» и сообщения.
- **Автор дубля не узнаёт номер исходной заявки** из уведомления — только в карточке.
- **Сверка каждые 10 минут читает чат каждой карточки доски** — сотни запросов к YouGile при общем на компанию лимите частоты.
- **Анализатора нет.** Блок «Гипотеза (ИИ)» в карточке и переменные `ANALYZER_*` — заготовки. Мёртвые также `YUGILE_BOARD_NAME`, `YUGILE_COLUMN_NAME`, `YUGILE_REPO_STICKER_NAME`: доска и стикеры берутся из настроек в базе.
- **Enum в базе хранится именами** (`DONE`, а не `done`) — помнить в миграциях.
- **Локальный стек и прод делят один домен.** Оба ставят вебхук Telegram на `PUBLIC_BASE_URL` при старте — кто стартовал последним, тот и получает обновления. Перед работой с прод-ботом локальный `docker compose` остановить.
- **Локальная копия боевой базы с включённой связью заводит в YouGile подписки на прод-домен** с чужим секретом. Перед локальным запуском выключить связь в настройках или убрать `YUGILE_API_KEY`.
- **Бэкапов нет.**
- Выкатка — с окном обслуживания: сервис на статус-странице как «Сбор багов и задач» (проба `/healthz`).

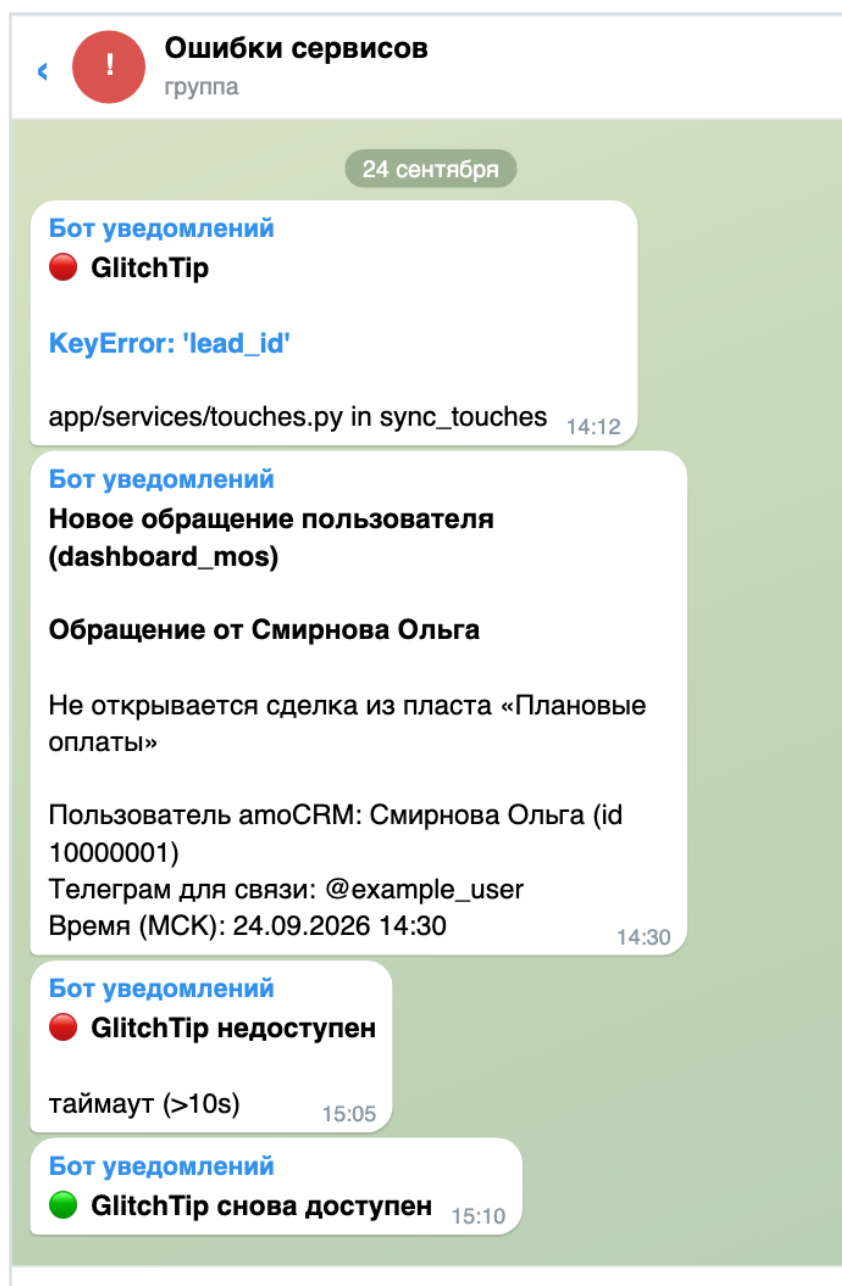
Релей GlitchTip в Telegram

Для разработчиков. Состояние на 24.09.2026. Подробности — [README](#)

[Buharestskie/gitctip_tg_bot](#). Отчёт по оплатам для группы ОС описан отдельно — статья «Отчёт по оплатам группы ОС».

Коротко

gitctip_tg_bot — релей в Telegram. Принимает HTTP-вебхуки и пересылает их в чаты: алерты GlitchTip, обращения «Сообщить о проблеме» из дашбордов и виджетов, отчёт смены менеджера МОС. Сам следит, жив ли GlitchTip, и раз в день собирает отчёт по оплатам группы ОС.



Где живёт

Сервер	<code>ulrica</code> — с российских площадок <code>api.telegram.org</code> недоступен; каталог, контейнер и адрес — в README
Адрес	HTTP без TLS и без reverse проху, на отдельном порту <code>ulrica</code>
Код	доставляется git-бандлом: у <code>ulrica</code> нет доступа к GitHub

Что куда уходит

Все пути защищены одним токеном в адресе: `/hook/<WEBHOOK_TOKEN>`. Чужой токен — `401`.

Кто шлёт	Путь	Как бот понимает, что это	Чат
GlitchTip (все проекты)	<code>/hook/...</code>	всё, что не подошло ниже	<code>TELEGRAM_CHAT_ID</code>
«Сообщить о проблеме»: МОС, МПП1, МПП2, виджет Яндекс.Диска, <code>return_amo_hook_service</code>	<code>/hook/...</code> (у отправителя — <code>FEEDBACK_WEBHOOK_URL</code>)	в теле есть непустой <code>alias</code>	<code>TELEGRAM_CHAT_ID</code>
Отчёт смены из виджета МОС	<code>/mos-workday-report/...</code> (у отправителя — <code>WORKDAY_REPORT_WEBHOOK_URL</code>)	в теле есть объект <code>report</code>	<code>WORKDAY_REPORT_CHAT_ID</code>
Ручной запуск отчёта по оплатам	<code>POST /payments-report/...</code>	—	<code>PAYMENTS_REPORT_CHAT_ID</code>

Тип определяет **тело**, а не путь: отчёт смены, присланный на `/hook/`, всё равно уйдёт в чат отчётов.

Из пейлоада GlitchTip читаются только `text` и `attachments[].title / title_link / text`. Уровень события не читается — у любого алерта шапка «GlitchTip».

Подключить новый проект GlitchTip

1. В GlitchTip, в настройках проекта, — правило алерта с интеграцией **generic webhook**.
2. Адрес — `.../hook/<WEBHOOK_TOKEN>` релея: проще всего скопировать из правила любого уже подключённого проекта; адрес — в README, токен — в `.env` релея на сервере.
3. Как часто алертить, решает правило в GlitchTip: сам релей ничего не фильтрует и не склеивает — один POST, одно сообщение.

Монитор GlitchTip

Раз в 5 минут релей запрашивает `GLITCHTIP_HEALTH_URL` (таймаут 10 секунд). Пишет только при смене состояния: «`GlitchTip` недоступен» с причиной и «`GlitchTip` снова доступен». Одного неудачного опроса хватает для алерта; падение и подъём между двумя опросами не заметит.

На что наступают

- **Нет очереди и повторов.** Если Telegram отказал, отправитель получает `502` и должен повторить сам — иначе сообщение потеряно.
- **Сообщение длиннее 4096 символов** Telegram отклонит — тоже `502`. Режется на части только отчёт по оплатам.
- **Сменили `WEBHOOK_TOKEN`** — поправить адрес в правилах GlitchTip и `FEEDBACK_WEBHOOK_URL` / `WORKDAY_REPORT_WEBHOOK_URL` у всех отправителей.
- **Токен в пути** попадает в access-лог.
- **Поля `...Seconds` в отчёте смены МОС на самом деле минуты** — дашборд уже переводит их, а имена полей не меняет.
- **`aiohhttp` не указан в `requirements.txt`** — работает транзитной зависимостью.
- **Отчёт по оплатам падает с `KeyError` молча** — если зарплатный дашборд поменял формат ответа, в чат не придёт ни отчёт, ни сообщение о сбое.
- Внешней пробы у релея нет — окно обслуживания при выкатке открыть не на что.

Сервисы вокруг атоCRM

Дедупликация сделок: anti_double_deals

Для разработчиков. Состояние на 24.09.2026. Подробности, переменные и выкатка — [README Buharestskie/anti_double_deals](#).

Коротко

anti_double_deals не даёт заводить в amoCRM ([donula.amocrm.ru](#)) повторные сделки на одного клиента и раскладывает новые заявки по этапам. На каждую новую сделку он сливает контакты с тем же телефоном, решает, какая сделка лишняя, убирает её в этап для дублей и переносит оставшуюся в рабочий этап. Всё, что сделал, пишет в журнал.

Как обрабатывается сделка

1. **Вебхук о новой сделке** (штатный amoCRM или Sensei) → [api](#) ставит задачу в Celery на каждую сделку. [worker](#) берёт блокировку сделки в Redis.
2. **Все телефоны иностранные** (казахстанские [+7 7xx](#) тоже) → этап для дублей, тег «Иностранный номер». Конец.
3. **Слияние контактов**. Ищутся контакты с теми же последними 10 цифрами телефона. Выживает самый старый, остальные сливаются в него. В публичном API слияния нет — используется внутренний [/ajax](#) amoCRM, для которого worker входит в веб-интерфейс браузером Playwright под служебным пользователем; сессия хранится в Redis. После слияния все сделки группы заново привязываются к выжившему контакту.
4. **Кто дубль**. Если у контакта есть более старая *активная* сделка в одной из 11 «рабочих» воронок (не в статусе 143) — дублем считается новая: этап для дублей, тег «Дубль новая». Иначе в этап для дублей уходят старые неактивные сделки с тегом «Дубль старая», а новая остаётся.
5. **Маршрутизация**. Оставшаяся сделка переносится в этап группы А или В по полю-списку 664291 (пять значений — группа А, остальное и пустое — В).
6. **Журнал**. Исход пишется в [lead_routing_log](#), одна строка на сделку: «Маршрутизация», «Дубль новая», «Иностранный номер», «В обработке». Страница «Журнал сделок» показывает последние 100 записей со ссылками на сделки.

Страховка: `beat` раз в 30 минут прогоняет сделки старше 5 минут, застрявшие на входном этапе, и раз в 5 минут лечит сделки, больше 10 минут висящие «В обработке».

Откуда данные: историю (кто ещё с этим телефоном, сделки контакта, справочники) сервис читает из реплики `amo_db`, с откатом на API при ошибке. Всё, от чего зависит решение, и всё, что читается сразу после записи, — из amoCRM напрямую: реплика отстаёт на секунды, в худшем случае на минуты.

Где живёт

Сервер `global`, compose из шести контейнеров: `api`, `worker`, `beat`, `frontend`, `postgres`, `redis`. Наружу — через шлюз `donu1a26.ru`. Каталог на сервере **без git**: выкатка копированием, порядок — в README. GitHub Actions `deploy.yml` продю не соответствует — коммиты в `main` пока только с `[skip ci]`. База бэкапится ежедневно, хранение 14 дней. В учёте обращений сервис называется «Журнал маршрутизации лидов».

На что наступают

- **Бизнес-правила защиты в код** (`backend/app/constants.py`): рабочие воронки, поле 664291, значения группы А, этапы групп, входной этап страховки. Поменяли воронку или этап в amoCRM — нужна выкатка. Id этапов печатает `backend/scripts/dump_pipelines.py`.
- **Слияние держится на входе через браузер.** Включат служебному пользователю 2FA, сменяют пароль или amoCRM поменяет форму входа — слияние встанет, сделки с дублями будут повторяться до 5 раз и зависать «В обработке». Скриншот и HTML неудачного входа — в `/tmp` внутри контейнера `worker`; сброс сессии — удалить ключ `amocrm:ajax_session_v2` в Redis.
- **Локальный запуск с боевыми ключами обрабатывает боевые сделки** параллельно с продом и не видит его блокировок. Локально — только с тестовыми ключами.
- **Одна сделка может идти минутами:** после слияния `worker` ждёт до 120 секунд, пока amoCRM удалит слитые контакты; блокировка живёт 600 секунд.
- **Лимит 7 запросов в секунду считается на процесс**, а у `worker` несколько процессов — вместе они могут упереться в лимит amoCRM; спасают повторы на 429.
- **Реплика пускает не всех:** IP сервера должен быть в списке доступа `amo_db`, иначе сервис молча работает только через API и тратит квоту. Видно по `amo_db read ... failed` в логах.
- **Имя каталога = имена контейнеров и тома.** Бэкап ищет контейнер `anti_double_deals-postgres-1`; выкатка в каталог с другим именем сломает бэкап и поднимет пустую базу.

- **Redis без тома** — очередь Celery пропадает при пересоздании. Сделки со входного этапа подберёт страховка, остальные — вручную.
- **Страница журнала — dev-сервер Vite**, без сборки; в интерфейсе только последние 100 записей.

Виджеты amoCRM

Встречи Контур.Толка из сделок: `contur_talk_widget`

Для разработчиков и тех, кто решает, нужна ли эта связка. Состояние на 25.09.2026.
Подробности — [README](#) `Buharestskie/contur_talk_widget`.

Что делает

`contur_talk_widget` — связка amoCRM и Контур.Толка. Когда сделка переходит на выбранный этап воронки, сервис сам создаёт видеовстречу в Толке «Разбор с финансовым экспертом» на час, начиная со времени из поля сделки с датой встречи. Организатор — ответственный по сделке (если он найден среди пользователей Толка по имени и фамилии), иначе — организатор по умолчанию. Ссылка на встречу записывается в поле сделки, и менеджер видит её прямо в карточке.

Работает в одну сторону: из Толка в amoCRM ничего не возвращается. Разбор записей встреч — отдельный проект, `kontur-talk-reporter`.

Состояние

Сейчас сервис не работает нигде. Последние правки — июнь 2026. На старом сервере маршрут к нему был, но вёл в пустоту, и при переезде в сентябре 2026 его не перенесли. Встречи в Толке из сделок сейчас автоматически не создаются.

Если решат запустить

- Нужны: поле сделки с датой и временем встречи, поле для ссылки, ключ API Толка, токен amoCRM, вебхук amoCRM на нужный этап нужной воронки. Порядок — в README, раздел «Деплой».
- **Этап выбирается только настройкой вебхука в amoCRM.** Сам сервис этапы не фильтрует: встреча создаётся на каждую пришедшую смену статуса.
- **Повторы не отсекаются.** Вернули сделку на этап — будет вторая встреча.

- **Ответственный ищется по «Имя Фамилия».** Если в атоCRM записано «Фамилия Имя», совпадения не будет, и встречу получит организатор по умолчанию.
- Ошибки видны только в логах сервиса — уведомлений о сбоях нет. Перед запуском стоит завести пробу в мониторинг.

Лендинги

Квиз-лендинги

Для разработчиков. Состояние на 25.09.2026. Подробности, переменные и выкатка — README репозитория `Buharestskie/mfctest`, `do-nula-refinancing-2`, `bfl_fspp`, `mfcpisan`. Что видит отдел продаж — статья «Заявки с лендингов».

Коротко

Четыре лендинга-квиза — лидогенераторы отдела продаж. Посетитель отвечает на несколько вопросов и оставляет телефон, лендинг заводит в amoCRM контакт и сделку, а три из четырёх — ещё и примечание с ответами (у `do-nula-refinancing-2` примечания нет: ответы только в полях сделки). Своей бизнес-логики у них почти нет: сборка заявки, лимиты от спама и, у трёх из четырёх, код из SMS.

Репозиторий	Сайт	Что обещает	Воронка → этап	После телефона	Где живёт
<code>do-nula-refinancing-2</code>	реструктуризация.online	реструктуризация долгов	«Продажи МПП» → «Квалификация»	код из SMS	<code>global</code>
<code>bfl_fspp</code>	fssp.site	проверка долгов по базе ФССП (обращения к ФССП в коде нет)	«Продажи МПП» → «Квалификация»	код из SMS	<code>global</code>
<code>mfcpisan</code>	mfcpisan.ru	списание долгов через МФЦ	«Продажи МПП» → «Квалификация»	код из SMS	сторонний сервер, доступа нет
<code>mfctest</code>	mfctest.ru	списание долгов через МФЦ	«МПП ИИ-агент» → «Новый лид Тест»	ссылка в мессенджер, бот монолита donula	<code>global</code>

Родословная

Первым был `do-nula-refinancing-2` (апрель 2026). `bfl_fspp` — его форк, `mfcspisan` — форк `bfl_fspp`: файлы бэкенда местами совпадают байт в байт, `requirements.txt` одинаковый у всех. `mfctest` стоит особняком: это лендинг `pages/dolgimfc`, вырезанный из монолита `donula`, с переписанным бэкендом. Аудит 11.09.2026 рекомендовал слить четыре лендинга в один параметризуемый — различаются набор вопросов, домен, воронка и тексты, ровно то, что выносится в конфиг. Не сделано.

Как устроен каждый

- **Один процесс FastAPI** (Python 3.12, httpx) раздаёт статику лендинга и несколько JSON-ручек. Redis — коды из SMS и лимиты по IP. Базы нет. У `mfctest` статику отдаёт nginx прямо с диска, бэкенд видит только `/quiz/`.
- **В amoCRM лендинги сами не ходят** — только через шлюз [amocrm-service](#), который держит токен, лимит и квоты. Шлюз пускает только адреса из своего allowlist: сети контейнеров на `global` и адрес стороннего сервера `mfcspisan`. Переехал лендинг — поправить allowlist, иначе анкета отвечает `502`.
- **SMS** — через `beeline-service`. У `do-nula` и `bfl_fspp` первый код запрашивает фронт после отрисовки страницы ввода кода, у `mfcspisan` — сервер внутри `/quiz/submit`.
- **Сделка создаётся до подтверждения кода**. Проверка кода в amoCRM ничего не пишет, подтверждённые и неподтверждённые заявки в воронке не различить.
- **ID воронок, этапов, полей и тегов — константы в `app/constants.py`**. Пересоздали поле в amoCRM — правка кода и пересборка образа.
- **Юридические документы** (`legal/` у трёх лендингов, с 22.09.2026): каждая редакция — отдельный файл с датой в имени, старые остаются доступны по адресу с датой, дата действующей редакции согласия пишется в сделку. Существующий файл не правят — добавляют новый. Документы запечены в образ: новая редакция появится после пересборки. У `bfl_fspp` этого обновления нет.

mfctest: две сделки

После телефона сделка «Лид с сайта» создаётся сразу, SMS нет. Потом посетитель выбирает мессенджер, ссылка ведёт в бота монолита `donula` с параметром `ai-<id сделки>`. Бот заводит **свою** сделку «ИИ Лид» в той же воронке и записывает строку `ai-<id>` в её поле; сделку лендинга он не трогает. Связь двух сделок — только эта строка.

Выкатка

- `do-nula-refinancing-2`, `bfl_fspp`, `mfctest` — на `global` git-клоны `main`: `git pull --ff-only` и `docker compose up -d --build`. У всех трёх есть внешняя проба («Реструктуризация долгов», «Проверка по ФССП», «Анкета МФЦ», на статус-странице скрыты) — выкатка в окне обслуживания.
- `mfctest`: статику `nginx` берёт прямо с диска, после `git pull` она уже на сайте; пересборка нужна для бэкенда и документов.
- `mfcpisan` — на стороннем сервере, как он там выкатывается, неизвестно; пробы у нас нет.

На что наступают

- Домен `mfctest.ru` ещё указывает на старый сервер `app`. Пока `app` жив, трафик идёт через него, и проба смотрит туда же. Перед выключением `app` А-запись нужно переключить.
- У `mfctest` корень сайта совпадает с корнем проекта. Всё, что не закрыто в `nginx`, видно снаружи — включая `README` и `compose`. Секреты в репозиторий не класть.
- Кнопка «Изменить номер» у `do-nula` создаёт новые контакт и сделку, у `bfl_fspp` и `mfcpisan` меняет телефон у того же контакта.
- Сбой после создания сделки даёт дубль. Если упал запрос примечания или отправка SMS, посетитель видит ошибку, а сделка уже есть — повторная отправка создаст вторую.
- Сборка на `global` ходит в зеркало `PyPI` (прописано в `Dockerfile` через `ENV`): прямой `PyPI` из сети сборки недоступен. У `mfcpisan` зеркала нет — при переносе на `global` сборка упадёт.
- Без файла согласия в `legal/soglasie/` каждая анкета падает с `500`: дата редакции обязательна для сделки.
- Кэш браузера. Скрипты подключены с меткой версии в адресе — после правки JS/CSS её нужно поменять в HTML.

Инфраструктура

Мониторинг и автоматизация серверов: infra

Для разработчиков. Состояние на 28.09.2026. Подробности, переменные ролей и команды — README репозитория `infra` и `files/heartbeat/README.md`. Как считается, кто и чем пользуется, — статья «Телеметрия использования: `usage_telemetry`».

Коротко

Репозиторий `infra` — единственный источник правды о серверах: инвентарь Ansible, роли базовой настройки, стек мониторинга, правила алертов, дашборды Grafana, статус-страница и служба окон обслуживания. Главное правило проекта: **изменение, сделанное мимо репозитория, не существует**. Накликанное на сервере или в веб-интерфейсе затрётся следующим прогоном роли.

Что	Где в репозитории	Снаружи
Prometheus, Alertmanager, blackbox_exporter, Pushgateway, Grafana	<code>roles/monitoring_server</code> , правила — <code>prometheus/rules/</code>	Grafana — <code>grafana.donula26.ru</code> (вход по логину); остальное наружу не смотрит
Агент метрик на каждом сервере	<code>roles/monitoring_agent</code>	—
Метрики PostgreSQL	<code>roles/postgres_exporter</code>	—
Статус-страница	<code>roles/status_page</code>	<code>status.donula26.ru</code>
Окна обслуживания (админка и MCP)	<code>roles/status_admin</code>	<code>status.donula26.ru/admin/</code> , <code>status.donula26.ru/mcp</code>
Heartbeat периодических задач	<code>files/heartbeat/heartbeat.py</code>	<code>push.donula26.ru</code>
GlitchTip (ошибки приложений)	роли нет; проба и реестр проектов — <code>docs/projects.md</code>	<code>monitoring.donula26.ru</code>
Бэкапы баз, фаервол, базовая настройка, аудит	<code>roles/db_backup</code> , <code>roles/firewall</code> , <code>roles/base</code> , <code>playbooks/audit.yml</code>	—

Что	Где в репозитории	Снаружи
Коллектор телеметрии использования	roles/telemetry_collector	—

Метрики и алерты

- **Prometheus** опрашивает цели раз в 30 секунд и хранит данные 90 дней. Список серверов для опроса собирается из инвентаря (`inventory/hosts.yml`), поэтому новый сервер появляется в мониторинге только после прогона `monitoring_server`.
- **Сервер изнутри** — `node_exporter` с TLS и basic auth, плюс скрипты для `textfile collector`: состояние контейнеров (статус, число перезапусков, `healthcheck`, время старта), метрики хоста, возраст свежих бэкапов. `sAdvisor` не используется, поэтому CPU и памяти по отдельным контейнерам в метриках нет.
- **Снаружи** — `blackbox_exporter` проверяет публичные адреса из `monitoring_probe_targets` (`inventory/group_vars/all/main.yml`) и срок действия TLS-сертификатов. По умолчанию модуль `http_2xx`, для остальных случаев модуль задаётся явно. Пробы бьют в `/health`, где он есть.
- **Базы** — `postgres_exporter` у `amo_db` и у баз мобильного оператора: соединения, архивация WAL, отставание синхронизации.
- **Правила** — `prometheus/rules/*.yml`: `hosts`, `containers`, `blackbox`, `backups`, `amo_db`, `umobile`, `contur_transcribe`, `meta` (сам мониторинг и `heartbeat`). В Grafana алерты не заводятся.

Alertmanager шлёт уведомления в Telegram через прокси (напрямую `api.telegram.org` недоступен), в два чата:

Алерт	Куда и когда
<code>severity: critical</code>	В критический чат сразу, с 08:00 до 23:00 по Москве. Если сработал ночью, придёт в 08:00, при условии что проблема ещё держится. Повтор раз в час.
<code>severity: warning</code> + <code>delivery: prompt</code>	В тот же чат, днём сразу, ночью придерживается до утра. Повтор раз в 6 часов. Метка для того, что не срочно, но нужно узнать в тот же день.
Остальные <code>warning</code> и <code>info</code>	Утренняя сводка: одно сообщение в окне 08:00–08:20.

Подавления: при `HostDown` молчит всё, что есть на этом хосте. При `DockerDown` и `DockerMetricsStale` молчат контейнерные алерты. При `AmoDbPostgresDown` молчат производные проверки `amo_db`. При `DiskUsageCritical` молчат предупреждения о том же диске. Истёкшие тишины Alertmanager хранит 30 дней: это журнал объявленных работ.

Дашборды Grafana лежат в папке «Инфраструктура»: «Инфраструктура: серверы», «Инфраструктура: контейнеры», «Доступность снаружи», «Периодические задачи», «amo_db», «Использование приложений». Их JSON генерируется скриптами `generate.py` и

`generate_usage.py` в `roles/monitoring_server/files/dashboards/`.

Правка дашборда в браузере не сохранится: включено `allowUiUpdates: false`. Дашборды меняют в генераторе, потом прогоняют роль.

GlitchTip

GlitchTip на `monitoring.donula26.ru` собирает исключения приложений через `SENTRY_DSN`. Своя запись в нём есть у каждого нашего проекта. Одной переменной `SENTRY_DSN` мало: в коде нужны зависимость `sentry-sdk` и вызов инициализации. Проще всего взять `init_sentry()` из пакета `usage_telemetry`: `install_telemetry()` вызывает её сам.

- Об ошибке в чат приходит уведомление, когда одна и та же ошибка случилась пять раз за десять минут. Разовые ошибки видны только в интерфейсе GlitchTip.
- Uptime-мониторы в GlitchTip есть, но уведомления о доступности от них выключены с 18.09.2026. О том, что сервис недоступен, сообщает только Alertmanager, чтобы одно событие не приходило двумя сообщениями.
- Роли Ansible для GlitchTip нет: проекты и правила уведомлений заводятся в его интерфейсе. Сам GlitchTip проверяется внешней пробой. Эндпоинта здоровья у него нет, поэтому проба смотрит корень сайта.

Статус-страница

`status.donula26.ru` открыта без авторизации и сделана для сотрудников. Это не Uptime Kuma и не отдельный монитор: страницу генерирует скрипт `roles/status_page/files/generate.py`. Он запускается по таймеру раз в минуту, берёт из Prometheus ту же метрику `probe_success`, на которой построены алерты, и пишет статический HTML. Поэтому страница не может показывать не то, что пишут в чат алерты.

Список сервисов задаётся в `status_page_services`. Поля: `probe` — адрес пробы, как в `monitoring_probe_targets`; `title` — название, понятное сотруднику; `visible` — показывать ли на странице (инфраструктурные сервисы проверяются, но скрыты); `project` и `job` на странице не видны, по ним окно обслуживания глушит алерты. Внутренние имена, адреса и версии на страницу не попадают.

Состояние	Когда
Работает	проба успешна, и за сутки вне окон обслуживания сбоев не было
С перебоями	сейчас отвечает, но за сутки были сбои; или падает, но меньше двух минут подряд

Состояние	Когда
Не работает	все пробы за последние 2 минуты неуспешны (<code>status_page_down_window</code>)
На обслуживании	сейчас идёт объявленное окно; перекрывает все остальные состояния
Нет данных	по пробе нет метрик, например недоступен Prometheus

У каждого сервиса есть полоса за сутки из 48 получасовых сегментов и процент доступности за сутки. Минуты внутри окон обслуживания в процент не входят.

Окна обслуживания

Любой деплой, обновление или фикс сервиса, у которого есть внешняя проба, идёт в окне обслуживания. Одна запись об окне делает три вещи:

- на статус-странице сервис становится синим, с бейджем «На обслуживании», а причина работ видна в подсказке к сегменту;
- его сбои во время окна не портят процент за сутки;
- в Alertmanager заводятся тишины на все имена сервиса: `instance` (проба), `project` (контейнеры) и `job` (отметки heartbeat). Алерты по сервису в чат не приходят.

Порядок работ: открыть окно, выкатить и проверить, **сразу** закрыть окно. Пока окно открыто, мониторинг сервиса слепой, поэтому лучше открыть короткое окно и продлить его, чем забыть длинное. Причину пишут человеческим языком, её видит любой сотрудник: «Обновление дашборда», а не «деплой v2.3.1».

Через админку

1. Открыть `status.donula26.ru/admin/`. Вход по именному логину: имя вошедшего записывается автором окна.
2. Выбрать «Сервис», заполнить «Причина», проверить «Начало» и «Конец». По умолчанию окно идёт от текущего момента на 30 минут.
3. Нажать «Объявить работы».
4. После работ нажать «Закончить сейчас» в строке окна. Конец окна переносится на текущий момент, тишины снимаются, запись остаётся в истории.

Кнопка «Убрать» удаляет запись целиком, вместе с синим сегментом. Она нужна только для окон, заведённых по ошибке. В таблице видны окна за последние трое суток и все предстоящие.

Через MCP

Та же служба отвечает по MCP на `https://status.donula26.ru/mcp`. Имя сервера — `status-maintenance`, авторизация — заголовок `Authorization: Bearer <токен>`. Токен у каждого разработчика свой. Пример подключения в Claude Code:

```
claude mcp add --transport http status-maintenance https://status.donula26.ru/mcp \
  --header "Authorization: Bearer $STATUS_MCP_TOKEN"
```

Инструмент	Параметры	Что делает
<code>list_maintenance</code>	<code>hours</code> (по умолчанию 72)	идушие, предстоящие и недавние окна с их <code>id</code>
<code>open_maintenance</code>	<code>service</code> , <code>minutes</code> (не больше 120), <code>reason</code> , необязательный <code>start</code> в ISO по Москве	открывает окно; по умолчанию оно начинается сейчас
<code>close_maintenance</code>	<code>id</code>	закрывает окно сейчас и снимает тишины

- В `service` передают название со статус-страницы или адрес пробы. Если название подходит нескольким сервисам (например, «МПП2» — и дашборду, и виджету), служба возвращает ошибку со списком подходящих, а не выбирает первый.
- Автором окна записывается «помощник <владелец токена>».
- Окно длиннее двух часов через MCP не открыть: такие работы объявляются вручную в админке.

Доступы задаются в Ansible Vault: `status_admin_web_users` (логины админки) и `status_admin_mcp_tokens` (токены MCP), затем прогоняется `playbooks/status_admin.yml`. Чтобы отозвать доступ у одного человека, его строку удаляют и снова прогоняют роль. Остальные доступы продолжают работать.

Heartbeat

Проблема: Celery Beat или APScheduler умирают, не роняя контейнер. Процесс жив, healthcheck зелёный, проба отвечает 200, а задача не запускалась трети сутки. Такое видит только сама задача, поэтому после каждого успешного прогона она отправляет отметку в Pushgateway.

- **Подключение.** Скопировать `files/heartbeat/heartbeat.py` в проект (зависимостей нет) и вызвать одну функцию: `install_celery(celery_app, job="...")`, `install_apscheduler(scheduler, job="...")` или `ping(job=..., task=..., max_age_seconds=...)` для разовой отметки. Сейчас так подключены шесть проектов.

- **Переменные:** `HEARTBEAT_URL` (`https://push.donula26.ru`; пустое значение выключает отправку — так работает локальная разработка), `HEARTBEAT_USER`, `HEARTBEAT_PASSWORD` (лежит в Vault), `HEARTBEAT_INSTANCE` (имя источника; для контейнеров его задают явно).
- **Порог** задача объявляет сама метрикой `heartbeat_max_age_seconds`. По умолчанию это полтора интервала расписания плюс пять минут, отдельные задачи переопределяют его через `max_age_overrides`. Правило `HeartbeatMissing` (critical) срабатывает, если отметки нет дольше порога.
- Отметку отправляет воркер после успешного выполнения. Упавшая задача отметку не обновляет, и молчание становится сигналом. Ошибки отправки пишутся в лог и не роняют задачу.

Ansible: как выкатывать

- Запуск — с ноутбука разработчика по ssh, Ansible стоит в `.venv` внутри проекта (`pip install -r requirements.txt`). Секреты лежат в `inventory/group_vars/all/vault.yml` под Ansible Vault. Пароль от хранилища — в файле `.vault_pass`, в git он не попадает, копия есть в VaultWarden.
- Порядок любого прогона: `--check --diff`, затем `--limit` на одном хосте, затем на всех.
- Новый сервер: завести в `inventory/hosts.yml`, затем прогнать `base` → `firewall` → `monitoring_agent` → `monitoring_server`. Если фаервол настроить после агента, агент не создаст правило для своего порта.
- Службу окон обслуживания ставят до nginx: сначала `playbooks/status_admin.yml`, потом `playbooks/web_proxy.yml`.
- `playbooks/audit.yml` только читает: что запущено на серверах и что требует внимания. Результат попадает в `reports/`, этот каталог в git не коммитится.
- Работы на сервисе с пробой проводятся в окне обслуживания, см. выше.

На что наступают

- **У сервиса три имени, тишина нужна на каждое.** Если у сервиса в `status_page_services` не заполнены `project` и `job`, окно заглушит только пробу, а контейнеры и heartbeat продолжат писать в чат. Лишнее имя тоже вредно: чужой `project` заглушит соседний сервис. Заполняют только то, что сверено с метками в Prometheus.
- **Сервиса нет в пробах — окно открыть не на что.** Сначала сервис заводят в `monitoring_probe_targets` и `status_page_services`, только потом объявляют для него работы.
- **Окно, заведённое задним числом,** красит историю, но тишины не создаёт: Alertmanager не принимает тишины в прошлом.

- **Обслуживание не прячет соседнюю аварию.** Сегмент на полосе синееет, только если все неудачные минуты получаса попали внутрь окна.
- **Heartbeat после деплоя молчит до первого прогона.** Правило отсчитывает возраст от более позднего из двух событий: последней отметки или последнего старта контейнеров проекта. Это работает, только если `job` отметок совпадает с именем compose-проекта. Проект с другим именем заалертит сразу после деплоя.
- **Переехал проект — поправьте `HEARTBEAT_INSTANCE`** и удалите старую группу из Pushgateway. То же с задачей, убранной из расписания: Pushgateway помнит отметки бесконечно, и старая группа будет держать алерт.
- **Порог heartbeat считается при старте процесса.** Новое расписание подхватится только после перезапуска контейнера.
- **Лимит docker-логов не действует сразу.** Роль `base` пишет `daemon.json`, но нужен полный перезапуск docker (в окне обслуживания) и пересоздание контейнеров.
- **Docker публикует порты в обход ufw.** Роль `firewall` закрывает этот обход. Порт контейнера, который должен смотреть наружу, открывают маршрутным правилом через `firewall_route_rules`. В файлах правил ufw пишите комментарии только латиницей, иначе ufw падает.
- **Ansible ставят только в `.venv`.** Фильтр `password_hash` не работает (`passlib` несовместим с `bcrypt 5`), поэтому bcrypt-хеши хранятся в Vault готовыми.
- **`--check` на чистой машине падает** на первой же задаче `apt`: нет `python3-apt`. Первый прогон на пустом сервере делается сразу боевым.
- **С ноутбука ssh может прийти не на тот сервер:** локальный прокси подменяет адрес. Признаки — «Permission denied» с рабочим ключом и предупреждение о смене ключа хоста. Сервер проверяют по `hostname`, а не по тому, что подключение прошло.
- **Хост вне инвентаря не мониторится.** Инвентарь пополняется по мере того, как доступы переносятся в VaultWarden.

Телеметрия

использования:

usage_telemetry

Для разработчиков. Состояние на 28.09.2026. Подробности и переменные — [README](#) репозитория `usage_telemetry`. Выкатка коллектора и дашборд лежат в репозитории `infra`, про него — статья «Мониторинг и автоматизация серверов: `infra`».

Коротко

`usage_telemetry` отвечает на вопрос, какими приложениями и какими их разделами пользуются живые люди. Цель — отличить приложение, которое висит мёртвым грузом, от приложения, которым пользуются редко, но по делу, и выключать только первое. Проект состоит из двух частей:

- **пакет** `usage-telemetry`: встраивается в приложение одной строкой и считает обращения;
- **коллектор**: принимает события, складывает их в PostgreSQL и раз в час пересчитывает суточные своды.

Результат смотрят в Grafana, на дашборде «Использование приложений».

приложение (middleware) —пачками по HTTP—> коллектор —> PostgreSQL —> Grafana

Как подключить к проекту

1. Добавить пакет в `requirements.txt` (или в `pyproject.toml`), закрепив тег:

```
usage-telemetry @  
https://github.com/Buharestskie/usage_telemetry/archive/refs/tags/v0.2.0.tar.gz
```

Для отправки ошибок в GlitchTip нужен ещё `sentry-sdk` (extra `sentry`).

2. Повесить сбор на приложение FastAPI или Starlette:

```
from usage_telemetry import install_telemetry

app = FastAPI()

install_telemetry(app)
```

Если задан `SENTRY_DSN`, `install_telemetry` заодно инициализирует GlitchTip. Если в проекте SDK уже поднят своим кодом, вызывайте `install_telemetry(app, sentry=False)`.

3. Сообщить, кто пришёл. В зависимости авторизации, где сессия или токен уже разобраны:

```
from usage_telemetry import mark_user

mark_user(request, user_id=session.amocrm_user_id, auth="session",
          session_id=session_id)
```

Значения `auth`: `session` — человек в браузере; `jwt` — виджет amoCRM с токеном от `widget_auth`; `telegram` — бот или Mini App; `basic` — админки; `api_key` — вызов между сервисами; `none` — без авторизации. Без `mark_user` вид входа угадывается по заголовкам, но идентификатора пользователя не будет.

4. Для бота на aiogram 3:

```
from usage_telemetry import install_bot_telemetry

install_bot_telemetry(dispatcher)
```

5. Добавить переменные в compose проекта и выкатить в окне обслуживания.

Переменная	Обязательна	Что означает
<code>TELEMETRY_SERVICE</code>	да	машинный код приложения. Должен совпадать с <code>project</code> этого сервиса в <code>status_page_services</code> (<code>infra</code>)
<code>TELEMETRY_URL</code>	да	адрес коллектора. Приложения на одном сервере с коллектором используют <code>http://host.docker.internal:8130</code> , с других серверов — <code>https://donula26.ru/telemetry</code>
<code>TELEMETRY_ENABLED</code>	нет	<code>0</code> — выключает сбор целиком
<code>TELEMETRY_QUEUE_SIZE</code> , <code>TELEMETRY_BATCH_SIZE</code> , <code>TELEMETRY_FLUSH_SECONDS</code> , <code>TELEMETRY_TIMEOUT</code>	нет	очередь (5000), размер пачки (200), интервал отправки неполной пачки (5 с), таймаут запроса (3 с)

Переменная	Обязательна	Что означает
<code>SENTRY_DSN</code>	нет	проект в GlitchTip; без него ошибки не отправляются

Если не задан `TELEMETRY_SERVICE` или `TELEMETRY_URL`, пакет пишет предупреждение в лог и ничего не собирает. Приложение при этом работает как обычно.

Что собирается

Одно событие на одно обращение к приложению:

Поле	Что в нём
<code>at</code> , <code>service</code> , <code>source</code>	время, код приложения, источник: <code>http</code> или <code>bot</code>
<code>method</code> , <code>path</code>	метод и шаблон пути из маршрута (<code>/api/deals/{id}</code>), а не <code>/api/deals/451</code>). Если маршрута нет, путь нормализуется: числа заменяются на <code>{id}</code> , UUID — на <code>{uuid}</code> , длинные непрозрачные сегменты — на <code>{token}</code>
<code>status</code> , <code>duration_ms</code>	код ответа и длительность
<code>auth</code> , <code>user_id</code> , <code>session_id</code>	чем запрос аутентифицирован и чей идентификатор из уже существующей авторизации: <code>amocrm_user_id</code> , id из JWT, <code>telegram_id</code>
<code>user_agent</code>	первые 200 символов
<code>client_ip</code>	доходит только до коллектора. Там из него, user-agent и общей соли считается отпечаток посетителя, а сам адрес не сохраняется

- **Не пишется никогда:** строка запроса, тело, referer, имена и почта. В строке запроса ходят разовые токены, а хранилище не должно превращаться в базу утёкших секретов.
- **Отсекается на входе:** `OPTIONS`, `/health`, `/healthz`, `/metrics`, `/ping`, `/favicon.ico`, `/robots.txt`, пути под `/static/`, `/assets/`, `/_next/`, `/_app/` и файлы статики по расширению. Свои исключения добавляются параметрами `skip_paths` и `skip_prefixes`.
- **В ботах** считаются только входящие от людей: сообщения (команда с короткой меткой вроде `/start beeline` или тип вложения) и нажатия кнопок. Данные кнопки сводятся к шаблону: `service:1842` превращается в `service:{id}`. Рассылки не считаются, прочтения Telegram ботам не отдаёт.

Как устроено внутри

- **Приложение ничего не ждёт.** Middleware кладёт событие в очередь в памяти, фоновая задача отправляет пакки в `POST /events`. Если коллектор недоступен, пачка выбрасывается без повторов. Если очередь переполнилась, она сбрасывается целиком. При остановке приложения остаток отправляется.
- **Коллектор** — FastAPI с PostgreSQL. Сырые события хранятся в таблице `events` 90 дней. Раз в час коллектор пересчитывает своды за последние трое суток и чистит устаревшее сырьё. Сводьы хранятся бессрочно:
 - `daily_users` — человеко-дни: пара «день + человек». В этот свод попадают только обращения с `auth`, `session`, `jwt`, `telegram` или `basic` и с известным `user_id`. Вызовы между сервисами сюда не входят;
 - `daily_paths` — разделы по дням: число обращений, людей, посетителей по отпечатку, ошибок 5xx и 95-й перцентиль длительности.
- **Справочник приложений** (таблица `services`) собирается из `status_page_services` и `telemetry_extra_services` в инвентаре `infra`. Коллектор читает его только при старте. По отметке первого события видно, раскатан ли сбор в приложение.
- **Ручки коллектора:** `POST /events` (ответ `{"accepted": N}`), `GET /health`, `GET /stats` — сколько накоплено и когда был последний свод. Коллектор слушает только внутренний интерфейс сервера. С других серверов события идут через путь `/telemetry/` на donula26.ru, куда пускают только адреса из списка в роли `web_proxy`.

Где посмотреть

Grafana, grafana.donula26.ru → папка «Инфраструктура» → дашборд «Использование приложений». Период по умолчанию — 30 дней.

Панель	Что показывает
Приложений под наблюдением	сколько приложений в справочнике
Сбор не раскатан	приложения, от которых не пришло ни одного события. Пока здесь не ноль, выводы о неиспользуемых делать рано
Без единого обращения за период	сбор работает, а обращений не было — кандидаты на выключение
Приложения по живости	главная таблица: обращения, люди, посетители (оценка), человеко-дни, последний заход и замечание: «сбор не подключён», «никто не заходил», «люди не опознаются» или «только оценка по отпечатку»
Человеко-дни по приложениям	график по дням
Разделы и данные	разделы выбранного приложения (переменная «Приложение»)
Кто пользуется	идентификаторы пользователей выбранного приложения, без имён: нужны, чтобы предупредить людей перед выключением

Панель	Что показывает
Медленное и ошибающееся	разделы с ошибками или с 95-м перцентилем больше 3 секунд

Дашборд генерируется скриптом `roles/monitoring_server/files/dashboards/generate_usage.py` в `infra`. В браузере он не редактируется.

Выкатка

- **Коллектор** выкатывается из репозитория `infra`: `ansible-playbook playbooks/telemetry_collector.yml`. Роль забирает код из ветки `main` этого репозитория, выпускает самоподписанный сертификат для базы, один раз создаёт соль отпечатка, раскладывает справочник приложений и поднимает compose. Пароль базы берётся из Vault (`telemetry_collector_db_password`), без него роль останавливается. База входит в ежедневные бэкапы.
- **Пакет** приезжает в приложения тегом в зависимостях. Новая версия пакета — это новый тег здесь и правка строки в каждом проекте.
- **Проверки:** `python tests/test_telemetry.py` проверяет пакет без базы и сети. `COLLECTOR_TEST_DSN=... python collector/tests/test_collector.py` проверяет коллектор и требует настоящий PostgreSQL: тест работает в отдельной схеме и удаляет её за собой.

На что наступают

- `TELEMETRY_SERVICE` не совпал с `project` в `status_page_services`. События пишутся, но приложение не попадает в список ожидаемых и на дашборде в таблице живости не видно. Приложениям без пробы на статус-странице код прописывают в `telemetry_extra_services`.
- **После правки справочника в инвентаре** коллектор нужно перезапустить. Роль `telemetry_collector` делает это сама.
- **События теряются осознанно.** Коллектор лежал — в данных дыра. Приложение, которое молчит, и приложение, которым не пользуются, на сводах выглядят одинаково. Различают их по колонке «Замечание».
- **Отпечаток — оценка, а не счёт людей.** Сотрудники из офиса сидят за одним адресом, и различает их только user-agent: одинаковые браузеры схлопнутся в одного посетителя.
- **Приложение на другом сервере шлёт в пустоту,** если адреса этого сервера нет в списке для `/telemetry/` в роли `web_proxy`: nginx отвечает 403, пакет молча выбрасывает события.
- **Без привязки к завершению приложения** последняя пачка теряется при каждом перезапуске. `install_telemetry` привязывается к `lifespan`, а если не получилось,

пишет предупреждение в лог. Его не игнорируйте.

- **Не у каждого бота есть** `Dispatcher`. У односторонних ботов, которые только рассылают, входящих от людей нет, и мерить там нечего.

Внутренние сервисы

SMS-шлюз: beeline-service

Для разработчиков. Состояние на 28.09.2026. Переменные, все ручки и выкатка по шагам — README репозитория `Buharestskie/beeline-service`. Как лендинги работают целиком — статья о квиз-лендингах в этой же книге.

Коротко

`beeline-service` — общий шлюз исходящих SMS. Сервис присылает номер и готовый текст, шлюз проверяет, что номер российский мобильный, применяет антиспам-лимиты и отдаёт сообщение Билайну (A2P) или хабу МегаФона. Шаблонов у шлюза нет: текст целиком приходит от вызывающего. Снаружи шлюз доступен по адресу `https://donula26.ru/service/beeline`, пускает только адреса из своего списка доступа.

Кто вызывает

Сейчас — три квиз-лендинга, которые подтверждают телефон посетителя кодом из SMS («Ваш код подтверждения: 1234»):

Лендинг	Что передаёт
<code>do-nula-refinancing-2</code>	<code>phone</code> , <code>message</code> , <code>ip</code> посетителя. <code>lead_id</code> не передаёт НИКТО
<code>bfl_fspp</code>	
<code>mfcsipisan</code> (на стороннем сервере)	

`mfctest` шлюз не вызывает: SMS-подтверждения у него нет, после телефона посетитель уходит в мессенджер. Все вызывающие ходят через `nginx` по публичному адресу, поэтому адрес каждого нового вызывающего нужно заранее внести в `ALLOWED_CALLER_CIDRS`, иначе он получит `403 Caller IP is not permitted.`

Как устроен

- **Один процесс FastAPI** (Python 3.12, httpx, APScheduler) и свой Redis в том же compose. Redis нужен только для лимитов. Базы нет, тестов нет.
- **Номер приводится к +7XXXXXXXXX**. Принимаются +7..., 8... и десять цифр с пробелами, скобками и дефисами. Всё, что не складывается в десять цифр с первой девяткой, получает 400 «Введите правильный российский номер» — до лимитов и до провайдера.
- **Четыре лимита** проверяются по порядку, каждый отвечает 429 с заголовком Retry-After:

Лимит	Порог	Когда действует
Повтор на тот же номер	1 SMS в 60 секунд	только если передан ip
Разных номеров с одного ip	3 за 24 часа	только если передан ip
SMS на один номер	3 за 24 часа	всегда
Всего SMS	1000 за сутки по UTC	всегда

ip — адрес конечного посетителя, который видел бэкэнд лендинга, а не адрес самого лендинга. Пороги — константы в app/constants.py, меняются только вместе с кодом.

- **Отправитель** в обоих каналах — donulya.ru. Вызывающий получает 202 и сырой ответ провайдера строкой в provider_response. Ошибка провайдера — 502.
- **Примечание в amoCRM**. Если в запросе есть lead_id, после ответа фоновой задачей в сделку добавляется примечание «исходящая SMS» с номером и текстом. Шлюз пишет его своим токеном напрямую, не через amocrm-service и не через amo_queue, со своим ограничением 6 запросов в секунду. Ошибка примечания только пишется в лог.

Билайн или МегаФон

SMS уходит в хаб МегаФона, только если выполнены оба условия:

1. на странице настроек шлюза включён переключатель «Использовать сервис Мегафон»;
2. номер по плану нумерации выделен МегаФону: таблица диапазонов лежит в app/services/megafon_ranges.json (25 DEF-кодов) и читается один раз при старте.

Всё остальное идёт через Билайн. По умолчанию переключатель выключен, то есть без ручного включения все SMS уходят Билайну.

- **Страница настроек** — <https://donula26.ru/service/beeline/megafon-sms-settings>: переключатель и кнопка «Сохранить». Нужны одновременно адрес из списка доступа и отдельный логин и пароль страницы (MEGAFON_SETTINGS_PAGE_LOGIN / MEGAFON_SETTINGS_PAGE_PASSWORD, к учётке МегаФона отношения не имеют).

- **Первого числа каждого месяца в 00:00 по Москве** переключатель выключается сам. Если в этот момент сервис лежал, выключение сработает при старте в тот же день.
- **Оператор определяется без учёта переноса номера.** Номер, ушедший от МегаФона к другому оператору, всё равно пойдёт через МегаФон, и наоборот. Скрипта для обновления диапазонов в репозитории нет.

Как вызвать

```
curl -X POST https://donula26.ru/service/beeline/sms/send \  
-H 'Content-Type: application/json' \  
-d '{"phone": "+7 000 000-00-01", "message": "Ваш код подтверждения: 4821", "ip": "<ip посетителя>"}'
```

Поле	Обязательно	Что это
<code>phone</code>	да	Российский мобильный в любом написании, 5–32 символа
<code>message</code>	да	Текст, 1–1000 символов, уходит провайдеру как есть
<code>ip</code>	нет	Адрес посетителя. Без него не действуют лимит повтора и лимит номеров на IP
<code>lead_id</code>	нет	Сделка amoCRM для примечания об SMS

Проверка живости — `GET /sms/health`, открыта без списка доступа, отвечает `{"status": "ok"}`.

Где код и как выкатывается

- Репозиторий `Buharestskie/beeline-service`, ветка `main`.
- На сервере — git-клон `main`: `git pull --ff-only` и `docker compose up -d --build`. Контейнер опубликован только на локальном интерфейсе сервера, снаружи — через nginx с префиксом `/service/beeline/`.
- Внешней пробы в мониторинге у шлюза нет, поэтому окно обслуживания на выкатку объявить не на что.
- Локально: `cp .env.example .env`, заполнить, поднять Redis и `uvicorn app.main:app --port 8001`. С боевыми ключами SMS уйдут на настоящие номера.

На что наступают

- **После выкатки переключатель МегаФона снова выключен.** Его состояние хранится в файле внутри контейнера, тома под него нет. Если маршрут был включён — перед выкаткой проверьте его (`GET /megafon-sms-settings/api`) и после неё включите заново на странице настроек.
- **202 от шлюза не значит, что МегаФон принял SMS.** Хаб отвечает HTTP 200 и при отказе, вердикт лежит в теле. Шлюз такой отказ только пишет в лог. Смотрите в `provider_response`: у МегаФона там `error.code`, `0` — принято.
- **Передавайте номер всегда в одном формате.** Лимиты считаются по строке `phone` как её прислали: `+790000000001` и `8 900 000-00-01` — это разные счётчики.
- **Счётчики растут и на отказах.** Если сработал другой лимит или провайдер вернул `502`, повторить отправку на тот же номер можно только через 60 секунд, а суточный счётчик номера уже увеличен.
- **Суточный лимит в 1000 SMS обнуляется в 03:00 по Москве** (сутки считаются по UTC), а ежемесячный сброс МегаФона срабатывает в 00:00 по Москве.
- **Адрес шлюза в `.env.example` лендингов неправильный.** У `bfl_fspp` и `mfcspsan` там `/services/beeline` с «s», у `do-nula-refinancing-2` — адрес внутри чужой docker-сети. Рабочий адрес один: `https://donula26.ru/service/beeline`.
- **Страница настроек локально не загрузит состояние:** её скрипт ходит на боевой путь `/service/beeline/megafon-sms-settings/api`. Локально переключайте маршрут через `curl` на `/megafon-sms-settings/api`.
- **Примечания с `lead_id` нагружают общий лимит amoCRM** мимо общего шлюза. Если начнёте передавать `lead_id` массово, учитывайте это вместе с остальными сервисами (см. «Как ходить в amoCRM из своего сервиса»).

Воронка найма: hiring_project

Для разработчиков. Состояние на 28.09.2026. Переменные, все ручки, порядок выкатки и отката — README и RUNBOOK репозитория `Buharestskie/hiring_project`.

Коротко

`hiring_project` — воронка найма менеджеров по продажам («охотников») для самой компании: от входа кандидата на сайт до выхода на обучение. Кандидат входит через Telegram, VK или MAX, заполняет анкету и записывает голосовой ответ. Голос оценивает Gemini, решение принимает РОП прямо в Google Sheet, дальше кандидату приходит сообщение в мессенджер, приглашение на групповое собеседование в Телемосте и ссылка на онлайн-обучение. В amoCRM кандидат попадает только после собеседования.

Сейчас сервис на наших серверах не развёрнут: проект передан новому владельцу и ждёт переезда. Домен `hiring.donula.online`, заложенный в целевую схему, пока не настроен. Внешней пробы в мониторинге нет.

Путь кандидата

1. **Вход.** Сайт запрашивает одноразовый токен (`/api/auth/token`, живёт 10 минут) и открывает бота в выбранном мессенджере. Кандидат отправляет боту телефон, сайт опрашивает `/api/auth/check` и получает имя и телефон из мессенджера, а не из формы.
2. **Анкета.** `/api/submit` принимает анкету и голос (`audio/webm`, `audio/mp4`, `audio/m4a`, от 1 KiB до 10 MiB). Анкета сначала пишется в журнал на диске, затем строкой в лист «Кандидаты» через Apps Script. Повтор с тем же `request_id` вторую строку не создаёт.
3. **Оценка.** Бэкенд отправляет голос в Gemini (через ProхуAPI): пять критериев от 1 до 5. 20 баллов и выше — PASS, кандидату сразу уходит приглашение на собеседование; 10–19 — GRAY, решает РОП; меньше 10 — FAIL, кандидату уходит отказ. Если модель недоступна, ставится GRAY с пометкой «AI недоступен»,

слушайте вручную».

4. **Решение РОПа** — в таблице: столбец T («Одобрён» / «Отклонён») и W — итог собеседования («Принят» / «Отклонён» / «Не пришел»). Apps Script складывает изменения в лист «Webhook Outbox» и доставляет их в бэкенд подписанным запросом на `/api/webhook/rop`; бэкенд пишет кандидату.
5. **Собеседование** — ближайший понедельник, среда или пятница не раньше чем через сутки, в 12:00 по Москве, в Телемосте. Напоминания — за 24 часа и за час.
6. **Обучение**. После «Принят» создаются контакт и сделка `Найм: <имя>` в воронке найма атоCRM, кандидат получает персональную ссылку на курс (живёт 90 дней). Сайт обучения отмечает визит и прохождение, сделка двигается только вперёд: принят → зашёл на обучение → готов к ролевке. После курса открывается ссылка в закрытое сообщество.

Как устроен

- **Бэкенд** — один процесс FastAPI на Python 3.12 (`runtime/backend/`). Состояние — в каталоге `DATA_DIR`: SQLite для переписки, JSON-файлы для соответствия чатов, записей на собеседование и токенов атоCRM, и журнал анкет. Журнал одновременно служит очередью эффектов: после рестарта недоделанное доигрывается.
- **Google Sheet и Apps Script** (`runtime/apps-script/google-apps-script-hiring.js`) — рабочее место РОПа: листы «Кандидаты», «Обучение», «Трафик», «Расписание МР», «Дашборд», «Системное», «Webhook Outbox». Скрипт выкатывается вручную в редакторе таблицы.
- **Мессенджеры** — своя реализация, не общая с `messenger_connector`: Telegram через long polling, VK и MAX через вебхуки.
- **Inbox** (`runtime/sites/inbox`) — общая лента переписки со всеми кандидатами из трёх мессенджеров и ответ из веб-интерфейса.
- **Фоновые циклы внутри процесса**: напоминания, вечерняя сводка РОПу в Telegram-группу в 20:00 и утренняя в 09:00 по Москве.
- **Статика** `runtime/sites/{auth,inbox,training}` отдаётся nginx и обращается к бэкенду по относительным `/api/...` — открывать её нужно с того же адреса, что и бэкенд.

Где код и как запустить

- Репозиторий `Buharestskie/hiring_project`. Документы о передаче проекта (`STATUS.md`, `TRANSFER.md`, `ROUTE_MAP.md`) описывают старое размещение и устарели.
- Офлайн-тесты (внешняя сеть заблокирована, нужен ещё Node.js для тестов Apps Script): `bash runtime/tests/run_all.sh` после установки `runtime/backend/requirements.lock` в `venv`.

- Бэкенд локально — `uvicorn main:app` из `runtime/backend` с явными `DATA_DIR`, `DISABLE_SCHEDULED=true` и `TG_POLLING_ENABLED=false`. Без явного `DATA_DIR` запуск упадёт на правах доступа. `/api/submit` без настоящего Apps Script тестовой таблицы отвечает `503`.
- Целевая выкатка — релизными архивами: `scripts/build-release.sh` собирает архив, `scripts/deploy-release.sh` раскладывает его на хост и переключает симлинк, при провале `/health` возвращает предыдущий; `scripts/rollback-release.sh` — откат. systemd-юниты и шаблон nginx — в `infra/`. Когда сервис будет развёрнут, сначала завести внешнюю пробу, потом выкатывать в окне обслуживания.

На что наступают

- **Только один Telegram-поллер и один планировщик.** Второй процесс на тех же токенах без `DISABLE_SCHEDULED=true` и `TG_POLLING_ENABLED=false` даёт дубли сообщений кандидатам.
- **`DISABLE_SCHEDULED` не выключает эффекты анкеты.** Оценка, сообщение кандидату и уведомление РОПу запускаются прямо из `/api/submit`. Стенд с боевыми токенами пишет живым людям.
- **Утренняя сводка включена по умолчанию.** Если `ENABLE_ROP_MORNING_NOTIFICATION` не задана, она уходит, хотя по решению проекта выключена.
- **Сессии входа живут в памяти.** Рестарт сбрасывает незавершённые входы кандидатов и начатую авторизацию amoCRM. Токен входа удаляется через 10 минут, поэтому напоминания «через 30/60 минут» почти не срабатывают.
- **Строки в «Кандидатах» не удалять и не сортировать.** Строки находятся по скрытому `request_id` в столбце X. Листы «Системное» и «Webhook Outbox» руками не редактируются.
- **Триггер Apps Script должен быть installable** и работает от имени создателя. Потерял аккаунт доступ — решения РОПа копятя в «Webhook Outbox» и не уходят.
- **Поток GRAY с пометкой «AI недоступен»** — почти всегда кончился ключ или баланс ProxyAPI.
- **Сделку в amoCRM бэкенд ищет по названию** `Найм: <имя>` (первый результат) — однофамильцы могут перепутаться. Ошибка создания сделки только логируется.
- **MAX:** `user_id` не равен `chat_id`, соответствие хранится в JSON-файле в `DATA_DIR`. Потеря файла ломает доставку в MAX.
- **Запись голоса во встроенном браузере мессенджера не работает** — сайт просит открыть ссылку в Safari или Chrome.
- **`build-release.sh` не проверяет чистоту дерева** (проверка унаследована из монорепозитория и смотрит не туда): незакоммиченные правки попадут в архив под чужим SHA. Перед сборкой — `git status`.
- **`runtime/backend/Dockerfile`, `backup_sheets.sh` и `legacy/`** — наследие старого размещения, целевая схема их не использует.

Интеграции

Отчёты по встречам

Контур.Толка:

contur_transcribe

Для разработчиков. Состояние на 28.09.2026. Переменные, песочница и выкатка по шагам — [README](#) репозитория `Buharestskie/contur_transcribe`. На сервере проект развёрнут под именем `donu1a-kontur-reporter`.

Коротко

`contur_transcribe` — демон без интерфейса и без входящих запросов. Раз в пять минут он забирает новые записи встреч из пространства Контур.Толка, получает расшифровку, делает управленческий разбор языковой моделью и раскладывает результат: короткий отчёт и файл расшифровки — в закрытую группу Telegram, структурированные поля — в рабочую таблицу Google Sheets отдела продаж. Задача — чтобы итоги тет-а-тетов и общих встреч РОПов доходили до таблицы без ручной работы.

Как устроен один проход

1. **Список записей.** Один ключ пространства (`KTALK_SPACE_API_KEY`) читает записи всех РОПов за последние `LOOKBACK_DAYS` дней (по умолчанию 7). Уже опубликованные и тестовые (по шаблонам названий) пропускаются.
2. **Расшифровка.** Сначала берётся готовая расшифровка Толка. Если текста в ней нет, запись скачивается в качестве 240p, из неё вытаскивается звук (ffmpeg) и отправляется в сервис распознавания а5х; задача опрашивается до готовности (до часа). Если у Толка запись ещё не готова, встреча откладывается до следующего прохода — это не ошибка.
3. **Разбор.** Диалог уходит в языковую модель а5х (`A5X_MODEL`). Она возвращает короткий управленческий радар: план и факт отдельно по каждому сотруднику, одна строка по каждому участнику, до трёх главных проблем, все поставленные задачи с ответственными и сроками (до 12), где нужен руководитель, оценка

полезности встречи 0–10. Отсутствующее помечается «Не зафиксировано». Длинные встречи разбираются через конспекты частей. Если модель недоступна, разбор собирается из протокола Толка.

4. **Маршрутизация.** Если участников два-три, однозначно опознан один сотрудник и на встрече был его РОП — это тет-а-тет: заполняется строка сотрудника на листе этого РОПа. Всё остальное — на лист «Общие встречи». Полная расшифровка в любом случае пишется на скрытый лист «Расшифровки Контур.Толк».
5. **Telegram.** Отчёт одним сообщением (длинный — разбивается на части) и расшифровка отдельным `.txt`.
6. **Метрики.** В конце прохода счётчики `contur_transcribe_cycle_*` (найденно, опубликовано, отложено, упало, пропущено) уходят в Pushgateway мониторинга.

Каждый выполненный шаг сразу отмечается в `data/state.json`. Сбой посередине не теряет встречу и не публикует её дважды: следующий проход доделает только оставшееся.

Кто есть кто

Сотрудники и РОПы описаны в двух справочниках вне кода: `config/rops.json` (РОП и название его листа) и `config/employees.json` (сотрудник, его РОП и признаки опознания: `emails`, `konturUserKeys`, `aliases`, `roomNames`, `titlePatterns`). Образцы — `*.example.json`. Заполнять руками не нужно: `python -m tools.sync_identities` показывает, что изменится, с `--write` — дописывает каждому найденному человеку ключ, почту и форму имени из справочника пространства.

Мониторинг

Проба снаружи сервису бессмысленна: у него нет адреса. Вместо неё два правила в `prometheus/rules/contur_transcribe.yml` репозитория `infra`:

Алерт	Когда	Обычная причина
<code>ConturTranscribeStalled</code>	30 минут подряд: встречи находятся, часть падает, опубликовано ноль	отказ распознавания или модели а5х
<code>ConturTranscribeSilent</code>	проходов не было больше 30 минут	контейнер упал, завис или не поднялся

Метрики уходят, только если задан `METRICS_PUSH_URL` (с ним — `METRICS_PUSH_USER`, `METRICS_PUSH_PASSWORD`, `METRICS_INSTANCE`); без адреса отправка молча выключена. Здоровое состояние без новых встреч выглядит как «найденно 59, пропущено 59, опубликовано 0» — пропущенные в основном уже опубликованы раньше.

Где код и как запустить

- Репозиторий `Buharestskie/contur_transcribe`, Python 3.12, httpx, pydantic. Команды: `python -m app.cli check` — проверить все доступы, ничего не публикуя; `once` — один боевой проход (есть `--limit` и `--only`); `daemon` — бесконечный цикл, так работает контейнер.
- **Песочница:** `tools.create_sandbox` создаёт тестовую книгу, `tools.find_chat` находит чат тестового бота. Значения кладутся в `.env.sandbox` со своим `STATE_FILE`, команды запускаются с `--env-file .env.sandbox` — рабочая таблица и группа отдела не задеваются.
- Тесты — `pytest`, внешние сервисы замочаны.
- **Выкатка:** на сервер доставляются `app/`, `pyproject.toml`, `Dockerfile`, `docker-compose.yml`. Серверные `.env`, `secrets/`, `data/` и справочники не перезаписываются. Порядок: резервная копия `data/state.json` → `docker compose build reporter` → `check` и `once` разовым контейнером → `docker compose up -d reporter` → логи.

На что наступают

- `data/state.json` **нельзя удалять или обнулять**. Пустой файл — повторная рассылка всех встреч за `LOOKBACK_DAYS` дней в группу и в таблицу. Перед любыми работами — копия.
- **Не запускайте `once` поверх работающего демона**. Второй процесс упадёт с «Предыдущий запуск ещё работает»: состояние защищено блокировкой.
- **Имена сотрудников в Толке и в справочнике расходятся**. Большинство заходит гостями и вписывает имя как придётся, иногда латиницей. Одно слово имени опознаёт человека, только если оно в отделе у одного. При двух Аннах «анна» бесполезна, и встреча уходит в общие. После любого изменения состава отдела прогоняйте `tools.sync_identities`.
- `titlePatterns` **пишите по основе фамилии**, а не по полной форме: шаблон с «никитина» не совпадёт с «Никитиной».
- **Листы РОПов сервис не создаёт**. Нет листа из `rops.json` — `check` сообщит об этом. «Общие встречи» и лист расшифровок создаются сами. Лист «Контроль РОПа+» считается формулами книги: после новых строк и месяцев проверьте, что диапазоны формул их охватывают.
- **Файл для распознавания ограничен 110 МБ** (`A5X_MAX_UPLOAD_MB`, сервис a5x отказывает около 128 МБ). Без ffmpeg запись уходит видеофайлом и может не пройти.
- **Сервер общий с другими боевыми контейнерами**, лимиты памяти и CPU в `docker-compose.yml` занижены намеренно.
- **Три имени у одного проекта:** репозиторий `contur_transcribe`, на сервере `donula-kontur-reporter`, локальная папка часто `kontur-talk-reporter`. Ищите по всем трём.

YouGile и GitHub: yogile_connector

Для разработчиков. Состояние на 28.09.2026. Переменные и все ручки — README
репозитория `Buharestskie/yogile_connector`.

Коротко

`yogile_connector` связывает задачи YouGile с ветками GitHub и двигает задачу по доске вслед за кодом: ветки нет — задача ждёт ветку; ветка появилась — задача в работе; ветка слита в основную — задача закрыта; после слияния пришли новые коммиты — задача снова в работе. Внутри YouGile скрипт-витрина показывает состояние бейджем на карточке и блоком «GitHub» в открытой задаче. Сервис доступен по адресу `https://dev-tasks.donula26.ru`, настройки — в админ-панели `/admin`.

Как задача находит свои ветки

- **По ID задачи в имени ветки.** Номер задачи из YouGile плюс префикс проекта дают ID вида `PRI-18`. Любая ветка, в имени которой есть этот токен (`feat/PRI-18-cart`, `pri_18`), считается веткой задачи — в каком бы репозитории организации она ни появилась.
- **Веток может быть несколько**, по одной в разных репозиториях. Каждая ведётся отдельно, общее состояние собирается из них: пока не слита хоть одна — задача в работе, слиты все — закрыта.
- **«Слито»** — у ветки нет коммитов вне основной (`compare`, `ahead_by == 0`), и при этом собственные коммиты на ней хоть раз были. Основная ветка у каждого репозитория своя: у части проектов это `dev`, а не `main`.
- **Стикер «Репозиторий»** на задаче выбирает один репозиторий — тот, где ветка создаётся автоматически. Задачи без стикера и без привязанных репозитория сервис не двигает (состояние «репозиторий не выбран»). Стикер сервис заводит и поддерживает сам: вешает на включённые доски, дописывает новые репозитории организации, убирает исчезнувшие.

Что сервис делает сам

Событие	Что происходит
Задачу перенесли в рабочую колонку	Создаётся ветка в репозитории из стикера (если на доске включено автосоздание)
Для задачи появилась ветка	Задача переезжает в рабочую колонку
Все ветки слиты	Задача уезжает в «Готово» и отмечается выполненной
После слияния пришли новые коммиты	Задача возвращается в работу
Задача ушла из рабочей колонки	Пустая ветка (без своих коммитов) удаляется — кроме колонок, где ветки велено сохранять
Задача закрыта	Слитая ветка удаляется. Неслитая не удаляется никогда — в чат задачи один раз приходит сообщение с её именем и числом неслитых коммитов

Автосоздание ветки на доске по умолчанию выключено, удаление пустых и слитых веток — включено; всё это переключается по доскам в админ-панели. Каждый переход пишется в чат задачи, если задан `YOUGILE_BOT_USER_ID`. Цвет карточки меняется под состояние: серый — нет репозитория или ветки, зелёный — слито.

Откуда события

- **GitHub** → `POST /webhooks/github`: события `push`, `create`, `delete`, `pull_request`. Подпись проверяется по `GITHUB_WEBHOOK_SECRET`, повторные доставки отбрасываются. Push в основную ветку пересчитывает все задачи досок этого репозитория.
- **YouGile** → `POST /webhooks/yougile`: события `task-created` и `task-moved`.
- **Плановая сверка** всей доски раз в `RECONCILE_INTERVAL_MINUTES` (по умолчанию 15) — страховка на случай потерянного вебхука.
- **Витрина** (`yougile-script/github-widget.js`) раз в 30 секунд читает `GET /api/board/state` и ничего не решает сама. Ответ кешируется на 30 секунд.

Админ-панель

Настройки живут в базе (SQLite в `./data` по умолчанию) и правятся на `/admin`: репозиторий и ключи по умолчанию, параметры YouGile, таблица досок — какие синхронизировать, какой репозиторий у каждой, какие колонки рабочая и завершающая, создавать ли ветку автоматически. При первом запуске значения переносятся из окружения, дальше окружение нужно только для базы и токенов входа. Кнопки панели ходят в `/admin/api/*` с заголовком `X-Admin-Token`:

Ручка	Что делает
<code>POST /admin/api/reconcile</code>	Полная сверка досок
<code>POST /admin/api/webhooks/yougile</code>	Создать подписки YouGile, тело <code>{"public_url": "https://dev-tasks.donula26.ru"}</code>
<code>POST /admin/api/webhooks/github</code>	Завести вебхук сервиса во всех репозиториях (или в переданном списке <code>repos</code>)
<code>POST /admin/api/reload</code>	Перечитать настройки и структуру досок
<code>GET /admin/api/status</code>	Сводка состояния

Проверка живости — `GET /healthz`, отвечает `{"status": "ok"}`.

Где код и как выкатывается

- Репозиторий `Buharestskie/yogile_connector`: FastAPI, SQLAlchemy, httpx. Локально — `uvicorn app.main:app --reload --port 8000` с заполненным `.env`. Вебхуки на ноутбук не придут — пересчёт вызывайте через `POST /admin/api/reconcile`.
- На сервере — git-клон: `git pull`, затем `docker compose -f docker-compose.yml -f docker-compose.prod.yml up -d --build`. Прод-файл — надстройка: убирает публикацию порта и подключает контейнер к сети внешнего прокси. Собранный без неё контейнер опубликует порт наружу и потеряет прокси.
- У сервиса есть внешняя проба «Связь с YouGile» (на статус-странице скрыта) — выкатка в окне обслуживания.
- Витрина ставится внутри YouGile** отдельно: конфигуратор скриптов (`Ctrl + ~`, только у администратора компании) → «добавить скрипт». Проще всего взять готовый текст с `/widget/script.js?token=...` — адрес сервиса и токен в нём уже подставлены.

На что наступают

- Кириллица в скрипте YouGile ломается** при передаче в песочницу: регулярка с `[a-яё]` валит скрипт целиком, строки превращаются в мусор. Код витрины пишется только латиницей, все русские подписи отдаёт сервис вместе с состоянием.
- Витрина работает в песочнице без `window`** и не видит ID задачи вида `PRI-18` — поэтому сопоставлением занимается сервис.
- App.httpGet из API конфигуратора не годится для GitHub**: нет `User-Agent`, GitHub отвечает `403`. Работает обычный `fetch`. `UI.safeHtml` вырезает SVG — плашки рисуются через `UI.rawHtml` с ручным экранированием.
- Стикер к доске привязывается только своим**. Если отправить в `PUT /boards/{id}` весь набор стикеров, YouGile отвергнет запрос из-за текстовых стикеров.

- **Первое появление задачи ничего не двигает** — сервис только запоминает состояние. Переходы работают со второго наблюдения, иначе первый запуск перетасовал бы всю доску.
- **Токену GitHub нужна запись.** Для создания и удаления веток — `Contents: Read and write`; с токеном только на чтение GitHub отвечает `403`, сервис пишет об этом в чат задачи. Без токена лимит — 60 запросов в час, полная сверка съедает его быстро.
- **События на одну задачу приходят пачкой** (`create` и `push` почти одновременно), поэтому обработка задачи идёт под мьютексом. Не убирайте его — задача получала по три одинаковых сообщения.
- **Зеркальную колонку нельзя делать рабочей:** задача туда переезжает, но пропадает из виду своей доски.
- **Задача, унесённая на ненастроенную доску,** больше не двигается, но её состояние в базе остаётся.
- **YouGile периодически отвечает `502` / `503`.** Событие, упавшее на такой ошибке, доделает плановая сверка — не выключайте её (`RECONCILE_INTERVAL_MINUTES=0`).

Продукты

Личный кабинет Юмобайл: web-app-umoible

Для разработчиков. Состояние на 28.09.2026. Переменные, ручки и выкатка подробно — в README репозитория `Buharestskie/web-app-umoible`; карта «экран → запрос → поля» — `API-MAP.md` там же.

Коротко

Личный кабинет абонента Юмобайла (MVNO на сети билайн): баланс, тариф, остатки, статус номера, несколько номеров на одном аккаунте, пополнение через СБП и картой, новости, чат с поддержкой, уведомления и активация новой SIM — от скана серийника до подписания договора в Госключе или проверки паспорта. Данные абонента — не наши: их отдаёт API оператора, приложение показывает ровно то, что он вернул. Своё у нас — два бэкенда: чат и уведомления, и выдача номеров при активации.

Часть	Папка	Что это	Живое?
Веб-кабинет (PWA)	<code>web/</code>	React 19 + TypeScript + Vite, открывается на <code>app.umobile.ru/app/</code> , ставится на экран «Домой»	да, основная форма продукта
Чат и уведомления	<code>chat-backend/</code>	FastAPI + Postgres: чат с оператором через amoCRM, лента уведомлений, web-push и FCM, ведение сделки активации в amoCRM	да
Сервис активации	<code>activation-admin/</code>	FastAPI + Postgres: пул номеров и партии SIM, выдача номера при скане, привязка в ЛК оператора, сессии, админка	да
Нативная оболочка iOS	<code>native/</code>	SwiftUI + WKWebView; нативные чат, сканер SIM, пуши через Firebase	заморожена с 26.08.2026, см. ниже

Часть	Папка	Что это	Живое?
Конфиги nginx	server/	справочные копии боевых конфигов обоих доменов и зоны микрокэша	справочно
Выкатка	deploy/ , .github/workflows/ci.yml	скрипт выкатки и CI	да
Документация	корень	lk_umobile_api.yaml (спека API оператора), API-MAP.md, SPEC-*.md и TZ-*.md — задачи для разработчика API оператора, ПИСЬМО-АВТОРУ-API*.md	справочно

AGENT_CONTEXT.md местами описывает прежний фронт (оболочка app.html с экранами в iframe). Его убрали из дерева, он лежит под тегом legacy-frontend. Всё про экраны сверяйте с web/src.

Откуда данные и куда уходят

- **API оператора (ЛК)** — lk.umobile.ru/api/v1, не наш. Отдаёт вход по коду из SMS, регистрацию нового абонента, профиль, данные номера (баланс, статус, тариф, остатки), новости, регистрацию в Госключе и её статус, проверку паспорта. Из браузера приложение ходит туда через наш nginx по пути /lkapi/. Спецификация — lk_umobile_api.yaml в корне репозитория.
- **Оплата** — отдельный домен оператора umobile.ru, через наш путь /payapi/: создание заказа (СБП или карта), статус, чек. Список банков для СБП приложение берёт напрямую у НСПК (qr.nspk.ru) — это единственный запрос мимо нашего прокси.
- **Чат-бэкенд** (/api/v1, /ws/) — сообщения абонента уходят в чат amoCRM (канал amojo), ответы оператора приходят вебхуком и доставляются в открытое приложение по WebSocket, а в закрытое — пушем. Сюда же биллинг оператора шлёт события о пополнении и смене статуса SIM (POST /api/v1/notify, типы balance, status, info).
- **Сервис активации** (/actapp/) — выдаёт номер из своего пула, когда абонент сканирует пустышку, и привязывает его в ЛК оператора вместе с серийником SIM и id сделки amoCRM.

С какими нашими сервисами связан

- **amoCRM.** Чат-бэкенд ходит в amoCRM сам, напрямую, со своим ограничителем частоты — не через шлюз `amo-queue.donula26.ru`. Аккаунт задаётся в `AMO_SUBDOMAIN`, это не основной аккаунт donula. Воронка «Активация 2026». ID воронки, этапов и полей («Выбрал номер», «Серийный номер SIM», «Ссылка на ЛК») — константы в `chat-backend/funnel.py`.
- **Мониторинг.** У чата и активации есть внешние пробы, но на публичной статус-странице они скрыты.
- Других сервисов на `*.donula26.ru` код не вызывает.

Как устроено

Веб-кабинет

- Маршруты — в адресной строке (`web/src/app/App.tsx`). Вкладки: «Главная», «Новости», «Поддержка» (чат), «Профиль». Отдельно: вход и восстановление пароля, заказ SIM (ссылки на магазины Ozon и Wildberries), итог оплаты `/pay/:result`, установка на «Домой» `/install`, вход по ссылке `/magic`, флоу активации `/activation/*`.
- Все адреса API относительные (`web/src/shared/config.ts`). Всё проксирует nginx того же домена, прямые адреса оператора из браузера не используются.
- Главная одна на все состояния номера. Подписи статусов — в `web/src/shared/lib/numberStatus.ts`: «Активен», «Блокирован», «Ожидает оплаты», «Ожидает регистрацию», «Активируется». Незнакомый статус подписывается нейтрально.
- Service worker (`web/src/sw.ts`) отвечает только за пуши и клик по уведомлению, офлайн-кеша нет. HTML отдаётся без кэша, ассеты — с хешем в имени.
- Иконочный шрифт хранится у нас (`public/fonts/ms.woff2`): Google Fonts из РФ недоступен.

Активация SIM

1. Контактный номер → код из SMS. Код шлёт и проверяет ЛК оператора (регистрация по коду), в amoCRM заводится сделка.
2. Скан серийника пустышки. Сервис активации бронирует номер из пула на 30 минут, берёт у чат-бэкенда id сделки, привязывает номер в ЛК. Пул разделён на B2C и B2B, пропорция выдачи настраивается в админке.
3. Оплата, дальше развилка по сегменту номера: B2C подписывает договор через Госуслуги и Госключ, B2B снимает паспорт своей камерой с рамкой (`/activation/kyc`).
4. На каждом шаге фронт двигает сделку по этапам воронки (`POST /api/v1/activation/stage`).

Статусы в базе активации. Номер: `pool` (свободен), `res` (бронь), `activ` (активируется), `act` (активен), `taken` (уже занят в ЛК), `blk`, `quar` (карантин). SIM: `ship` → `pay` → `prov` → `bound`, отдельно `blk`. Каждое движение номера пишется в журнал `number_log`, в админке он открывается щелчком по номеру или ICCID. Сама таблица `numbers` хранит только текущее состояние.

Нативная оболочка

Приложение `native/` грузит не React-сборку, а свою офлайн-копию прежнего фронта (`native/Yumobile/app.html` и `screens/`). Чат, регистрацию пуш-токена и часть API берёт с `app.umobile26.ru`. Собирается скриптом `native/build-install.sh` с бесплатной подписью разработчика. Правки в `web/` в нативку не попадают.

Авторизация в общих словах

- **Абонент ↔ ЛК оператора:** вход по номеру и коду из SMS, в ответ Bearer-токен ЛК. Он хранится в браузере и идёт в `/lkapi/`.
- **Абонент ↔ наши бэкенды:** токен ЛК обменивается на токен нашей сессии (`POST /actapp/session`). Сервис активации проверяет токен запросом профиля в ЛК и берёт номера абонента оттуда. Сессия живёт 30 дней и подписана `SESSION_SECRET` — он общий у обоих бэкендов, поэтому сессию, выданную активацией, принимает и чат. Ручки чата, уведомлений, подписки на пуши, скана и состояния активации сверяют номер из запроса с номерами в сессии.
- **Сервис ↔ сервис:** биллинг — заголовок со своим секретом (`NOTIFY_SECRET`), к ЛК — внутренний ключ (`LK_INTERNAL_KEY`), активация к чату — `APP_SECRET` чата.
- **Админка активации** — `app.umobile26.ru/activation/`, за basic-auth и ключом `ADMIN_KEY`.

Переменные окружения

Значения — в `.env` рядом с `docker-compose.yml` каждого бэкенда, в git их нет. Полный список чата — `chat-backend/config.py`.

Сервис	Переменные
--------	------------

chat-backend	DATABASE_URL, AMO_SUBDOMAIN, AMO_CLIENT_ID, AMO_CLIENT_SECRET, AMO_REDIRECT_URI, AMOJO_CHANNEL_ID, AMOJO_CHANNEL_CODE, AMOJO_CHANNEL_SECRET, APP_SECRET, SESSION_SECRET, NOTIFY_SECRET, LK_INTERNAL_KEY, MAGIC_SECRET, MAGIC_WORKER, VAPID_PRIVATE_PATH, VAPID_SUBJECT, FCM_PROJECT_ID, FCM_CREDENTIALS_PATH, PUBLIC_BASE_URL, WORKERS
activation-admin	DATABASE_URL, ADMIN_KEY, APP_KEY, SESSION_SECRET, LK_BASE, LK_INTERNAL_KEY, CHAT_BASE, APP_SECRET, OTP_TEST_MODE
web (при сборке)	VITE_APP_SECRET, VITE_APP_KEY, VITE_VAPID_PUBLIC_KEY — необязательные, без них берутся значения из web/src/shared/config.ts

Ссылка с автоходом

В чат-бэкенде есть догон тех, кто бросил активацию на середине. На вебхук amoCRM о новой сделке бэкенд выпускает короткую ссылку `app.umobile.ru/i/<код>` и пишет её в поле сделки «Ссылка на ЛК». По ссылке абонент попадает в кабинет уже вошедшим, без ввода номера и кода, а главная сама показывает следующий шаг. Код гасит не переход по ссылке, а запрос из приложения — так превью-боты мессенджеров не сжигают ссылку раньше человека. Фоновая рассылка напоминаний включается переменной `MAGIC_WORKER`, по умолчанию выключена. По `SPEC-magic-link.md` 03.09.2026 рассылку приостановили до ответа автора API оператора. Что включено сейчас, смотреть в `.env` на сервере.

Локальный запуск

- **Фронт:** `cd web && npm install && npm run dev`, открыть `http://localhost:5173/app/`. Дев-сервер проксирует API на боевой `app.umobile.ru`: тестового контура у оператора нет, данные настоящие.
- **Мок-режим** — чтобы открыть экраны за авторизацией без живого абонента. В одном терминале `npm run mock-api`, в другом `npm run dev:mock`. Нужный экран открывается адресом `/app/__dev?to=/app/news`. Состояние номера в моке переключается адресами мок-сервера `/__state/...`, `/__seg/...`, `/__kyc/...` (см. `web/mock-server.mjs`).
- **Бэкенды:** `docker compose up -d --build` в `chat-backend/` и `activation-admin/`. Нужны Postgres и заполненный `.env`. Чат при первом старте связывается с amoCRM через `/amo/auth`, проверка — `GET /health` (`amo_authorized`, `amojo_connected`). Админка активации — `http://localhost:8020/?k=<ADMIN_KEY>`.
- **iOS:** `cd native && ./build-install.sh sim` (нужны macOS и Xcode).

Выкатка

- **Выкатывает CI.** Пуш в `main` запускает `.github/workflows/ci.yml`: `lint → sast → build → deploy`. Ветка `dev` и pull request проходят первые три стадии без выкатки. Deploy идёт на своём раннере на боевом сервере, ключей от сервера в GitHub нет.
- Раннер запускает только `deploy/umobile-deploy.sh` с целью `web`, `activation` или `chat`. Порядок: фронт, активация, чат. Для бэкендов скрипт снимает копию текущей версии, пересобирает образ и возвращает старую версию, если новая не ответила или в контейнере активации пустые ключи.
- У чата и активации есть внешние пробы, поэтому выкатка — в окне обслуживания. Окно открывать до мержа в `main`: выкатка начинается сама.
- Конфиги nginx в `server/` — только справочные копии, CI их не выкатывает. Правка nginx делается на сервере руками.

На что наступают

- **Разработка идёт на боевых данных.** `npm run dev` ходит в боевой ЛК и боевые бэкенды. WebSocket дев-сервер проксирует на боевой домен даже в мок-режиме.
- **Мерж в `main` — это выкатка на прод**, отдельной кнопки нет.
- **Выкатка фронта заменяет каталог сайта целиком** (`rsync --delete`). Всё, чего нет в сборке `web/dist`, с сайта пропадает. Класть файлы руками рядом со сборкой бесполезно — их сотрёт следующая выкатка.
- **Бэкенды — только с пересборкой образа.** Код копируется в образ, поэтому `docker compose restart` правку не подхватит, нужен `up -d --build`.
- **Секрет со знаком `$` в `environment:` compose приезжает обрезанным.** Сервис поднимается и молча отвечает 401. Секреты передавать только через `env_file`: в `activation-admin/docker-compose.yml` `APP_SECRET` нарочно не выписан.
- **`SESSION_SECRET` должен совпадать у обоих бэкендов, а `APP_SECRET` активации — с `APP_SECRET` чата.** Разошёлся первый — все персональные ручки чата отвечают 403. Нет второго — активация не получает сделку, и каждый скан SIM отвечает `amo_lead_missing`.
- **Активация зависит от amoCRM и чат-бэкенда.** Без сделки номер в ЛК не привязывается: скан возвращает `amo_lead_missing`, бронь откатывается, номер возвращается в пул.
- **Ошибки этапов воронки не видны абоненту.** Фронт отправляет этап и игнорирует ответ, так что сбой видно только в логах чат-бэкенда.
- **Статус «Активируется» ЛК сам не отдаёт.** Приложение ставит его себе, когда видит подпись в Госключе или принятые документы.
- **ЛК виснет на частых запросах.** Поэтому `/lkapi/profile/` и `/lkapi/number/data/` идут через микрокэш nginx на 5 секунд с токеном в ключе. Состояние кэша видно в заголовке `X-LK-Cache` (`HIT`, `MISS`, `STALE`). Повтор кода входа раньше чем через минуту ЛК отклоняет с `425`.

- **Таймауты фронта:** обычный запрос — 15 секунд (`REQUEST_TIMEOUT_MS`), отправка снимков паспорта — 90 секунд (`UPLOAD_TIMEOUT_MS`). Новый запрос с тяжёлым телом на общем таймауте будет сдаваться на мобильной сети.
- **Не писать пары «свойство + `-webkit-` -двойник» руками.** Префиксы проставляет сборка по `build.cssTarget` в `web/vite.config.ts`. Иначе минификатор выкинет одну из форм, и размытие пропадёт в Chrome или Safari. В деве CSS не минифицируется, поэтому такие вещи проверять через `npm run preview`.
- **Номер, снятый с пустышки после привязки в ЛК, уходит в карантин (`quar`), а не в пул.** Метода отвязки у ЛК нет: привязку снимают в ЛК руками, потом номер переводят в «Свободен» в админке. Карантин B2C и B2B не смешивается.
- **ОТР-ручки сервиса активации (`/actapp/send-code`, `/actapp/verify-code`) SMS не отправляют,** SMS-шлюза нет. Веб подтверждает номер через ЛК оператора и эти ручки не вызывает.
- **Нативка живёт своей копией фронта.** Изменения веба туда не доезжают. Бесплатная подпись разработчика действует 7 дней, потом приложение на телефоне перестает запускаться — пересобрать `./build-install.sh device`.
- **Postgres стоит на хосте, не в контейнере.** Контейнеры ходят к нему через `host.docker.internal`. Если оба сервиса падают на старте с ошибкой подключения к базе, сначала смотреть, на каких адресах слушает Postgres.

WebRTC-телефония: pbx

Для разработчиков. Состояние на 28.09.2026. Подробности, полный список ручек и переменных — README репозитория `Buharestskie/pbx`. Кода самого виджета amoCRM в репозитории нет — только его серверная часть.

Коротко

pbx — собственная WebRTC-телефония отдела продаж на Asterisk. Главное в ней — автодозвонщик: менеджер ставит статус «Готов», система берёт из очереди следующий лид, звонит сначала менеджеру, после его ответа — клиенту, и соединяет их. Менеджер разговаривает из браузера: виджет в amoCRM или приложение-софтфон. После звонка в сделке появляется примечание со ссылкой на запись.

Лиды попадают в очередь из amoCRM вебхуками этапов: сделка перешла на этап — контакт встал в кампанию. Руководители настраивают кампании, операторов и смотрят дашборды в админке.

Из чего состоит

Каталог	Что это
<code>backend/</code>	Один процесс на Node.js 20 (ES-модули): HTTP API на Express (порт <code>API_PORT</code> , 3000), WebSocket для виджетов (<code>WS_PORT</code> , 3001), клиент Asterisk ARI (<code>ari-client</code>), свой клиент AMI (только для стерео-записи), PostgreSQL (<code>pg</code>). Там же фоновые задачи: ready-тик раз в 10 с, воркер вебхуков, сторожа зависших контактов и каналов, перевод из затянувшейся постобработки в «Паузу», отправка статусов менеджеров наружу.
<code>admin/</code>	Админка — статические HTML-страницы: кампании, операторы, группы, транки, график работы, история звонков, дашборды. Ходят в <code>/api/admin/...</code> по относительному пути. Вкладка «Трансфер» — пока статичный макет без запросов к API.
<code>leadboard/</code>	Одна страница: таймлайн работы с лидами МПП-2 (<code>/api/admin/dashboards/lead-timeline</code>).

Каталог	Что это
<code>guide/</code>	Инструкции для операторов и заметки о релизах виджета. Чистая статика, к API не обращается.
<code>scripts/</code>	Серверные скрипты по cron: бэкап (<code>pbx-backup.sh</code>), разбор громкости свежих записей (<code>analyze-recordings.sh</code>), генератор отчёта о постобработках (<code>regen-wrapup-report.py</code>), его же запускает кнопка в админке).

Статику backend не раздаёт: в Express её нет, страницы отдаёт веб-сервер перед ним. Скрипты в корне `backend/` (`checkleads*.mjs`, `chat-*.js`, `reconcile.mjs` и подобные) — разовые сверки, в работе сервиса не участвуют.

Чего в репозитории нет: исходников виджета amoCRM, софтбокса для Mac/Windows и расширения браузера, конфигов Asterisk (pjsip, диалплан, звуки опроса) и coturn. Бэкенд только генерирует из базы SIP-аккаунты операторов и клиентов приложения и перезагружает pjsip.

Путь звонка

- Лид в очереди.** amoCRM шлёт вебхук этапа на `/api/admin/webhook/<токен кампании>`. Ручка только кладёт id сделки в таблицу `webhook_queue`. Воркер раз в 2 с берёт до 5 строк, тянет сделку и телефон контакта из amoCRM и добавляет контакт в кампанию.
- Менеджер «Готов».** Виджет держит WebSocket с бэкендом (`auth`, `agent_status`, `sip_registered`...) и SIP-регистрацию в Asterisk через WebRTC. Звонить начинают только после `sip_registered`.
- Выбор контакта.** PL/pgSQL-функция `pick_next_contact` атомарно выдаёт контакт (`FOR UPDATE SKIP LOCKED`) с учётом приоритета кампаний, закрепления за менеджером, окна звонков по часовому поясу клиента и циклов повторов.
- Звонок менеджеру.** Виджету уходит `screen_pop` с карточкой лида, через 300 мс — ARI originate на `PJSIP/<extension>` в Stasis-приложение (`STASIS_APP`, по умолчанию `pbx-dialer`) с аргументами `agent,<callId>`.
- Менеджер ответил** (событие `StasisStart` с ролью `agent`) — виджету `dialing_client`, статус менеджера становится `oncall`, менеджер назначается ответственным за сделку в amoCRM (кроме нескольких воронок-исключений в коде). Затем originate клиенту через транк: по умолчанию шаблон `OUTBOUND_ENDPOINT_TEMPLATE`, у оператора может быть свой транк.
- Клиент ответил** (`StasisStart` с ролью `client`) — оба канала в mixing-bridge, виджету `connected`, стартует запись: стерео через AMI MixMonitor, при сбое — моно через ARI.
- Конец разговора** (`StasisEnd`) — строка в `call_log`, примечание-звонок в сделку amoCRM со ссылкой на запись, виджету `call_ended`; если клиент ответил, менеджер уходит в постобработку. Если трубку первым положил менеджер, а разговор длился

не меньше `RATING_SURVEY_MIN_TALK_SEC` (40 с), клиента оставляют на линии и просят оценить разговор цифрой 1-5 (включается `RATING_SURVEY_ENABLED=1`, оценки — в `call_ratings`).

Если SIP менеджера отвалился посреди звонка, есть восстановление: 15 с на переподключение и повторный originate. Состояние активных звонков живёт только в памяти процесса.

Другие пути через тот же диалер:

- **Ручной звонок** — `manual_call` по WebSocket или click-to-call из amoCRM (`/api/app/click-call`). Личный транк менеджера (линия «Личная») используется только здесь, автообзвон всегда идёт с общей линии.
- **Перевод МПП-1 → МПП-2** (`/api/transfer/initiate`): клоузер выбирается среди «Готовых» по rr-credit из mpp2-touches, ему уходит `screen_pop` и originate с ролью `closer`; если заданы `MPP2_PIPELINE_ID` и `MPP2_STATUS_TRANSFERRED`, сделка переставляется в воронку МПП-2 на клоузера.
- **Звонок в мобильное приложение ДОНУЛЯ** (`/api/app/provision`, `/call`, `/connect`): клиенту заводится SIP-аккаунт, бэкенд приложения получает запрос на push, после ответа в приложении Asterisk соединяет его через диалплан, а не через Stasis.

Входящих звонков в диалере нет: `StasisStart` без известной роли игнорируется.

Откуда данные и куда уходят

- **amoCRM** — **напрямую**, не через шлюз [amo-queue](#). Своя OAuth-интеграция: токены в таблице `settings`, обновляются по 401 и пишутся туда же. Все запросы идут через общий лимитер процесса (не чаще ~4 в секунду, пауза на 429/403). Читает сделки и контакты, пишет примечания о звонках, ответственного, этапы при переводе и в дедупликации.
- **PostgreSQL** — вся логика очередей, кампании, операторы, журнал звонков `call_log`, оценки, лента событий, настройки. База своя, отдельная.
- **Записи разговоров** лежат на диске сервера телефонии. Наружу они уходят только ссылкой `https://pbx.donula26.ru/api/recordings/<файл>` в примечании amoCRM: подпись HMAC (`RECORDINGS_HMAC_SECRET`) и срок жизни (`RECORDINGS_TTL_SEC`, 30 суток). Если wav уже сжат в opus, по той же ссылке отдаётся opus.
- **AI-разбор звонков МПП-2** — по вебхуку `/api/admin/webhook-analysis/<токен>` при переходе сделки в «Думает» или «Обещал оплатить»: склейка записей, распознавание (Yandex SpeechKit через Object Storage, запасной вариант AssemblyAI), анализ YandexGPT. Результат пишется в таблицу `call_analysis`. Включается настройкой `mpp_analysis_enabled` и флагом у менеджера.
- **TURN** — `turn.donula26.ru`: виджет и приложение берут временные кредиты в `/api/widget/turn-config` и `/api/app/turn-config`.

Связи с нашими сервисами:

Сервис	Как связан
mpp2-touches	pbx берёт у него rr-credits для выбора клоузера, сообщает о назначении менеджера и о смене статусов (<code>MPP2_TOUCHES_BASE</code>)
mpp2-tasks-widget	получает смену статусов менеджеров (<code>MPP2_TASKS_STATUS_URL</code>); сам ставит статус «Готов» / «Офф» через <code>POST /api/widget/agent-status</code> и читает график работы
dashboard, dashboard_mpp2, SpeechApi	забирают историю звонков из <code>/api/history</code> — API в формате команды <code>history</code> Мегафон OATC (<code>EXTERNAL_API_TOKEN</code>)
webhook_for_call_record	скачивает записи по ссылкам из примечаний amoCRM
бэкенд мобильного приложения, <code>app.donula26.ru</code>	заводит клиентам SIP-аккаунты в pbx и получает от pbx запрос на push о входящем (<code>APP_BACKEND_URL</code> , <code>APP_PROVISION_SECRET</code>)

Статусы всех менеджеров отдаются в `GET /api/manager-statuses` (`MANAGER_STATUS_TOKEN`): `ready` «Готов», `oncall` «Занят», `pause` «Пауза», `wrapup` «Постобработка», `learning` «Обучение», `offline` «Офф».

Где код

- Репозиторий `Buharestskie/pbx`, ветка `main`. Точка входа — `backend/src/app.js`, на проде процесс стартует через `backend/src/bootstrap.js`.
- Диалер — `backend/src/dialer/` (`agent-first.js` — основной сценарий, `transfer.js` — перевод, `originate.js`), события ARI разводятся в `app.js`.
- WebSocket — `backend/src/ws/gateway.js`, ручки — `backend/src/http/routes/`, amoCRM — `backend/src/integrations/amo/`, очередь и воркеры — `backend/src/queue/`, миграции — `backend/src/db/migrations/`.
- Переменные окружения читаются из `backend/.env`; полный список с описанием — в README. Часть настроек (токены amoCRM, `mpp_analysis_*`) живёт в таблице `settings`.

Локальный запуск

- Нужны Node.js 20+ и PostgreSQL. Asterisk не обязателен: без ARI backend пишет `Running in offline mode (no Asterisk)`, поднимает HTTP и WebSocket, но звонить не может.

2. Схема базы — только дампом схемы с прода (как снять — в README): миграции её с нуля не создают. Из `settings` удалить токены amoCRM, если локальный процесс не должен ходить в боевой amoCRM.
3. `cd backend && cp .env.example .env`, задать `PG_*`, любое значение `ASTERISK_AMI_PASSWORD` и локальные пути `PJSIP_AGENTS_PATH` / `PJSIP_APP_CLIENTS_PATH`.
4. `npm install`, `npm run migrate`, `npm start`, проверить `curl -s localhost:3000/health` — ответ `{"status":"ok","asterisk":"offline",...}`.
5. WebSocket — любым клиентом на `ws://localhost:3001`, первым сообщением `{"type":"auth","agent_id":...}`. Страницы `admin/` и `leadboard/` нужно открывать с того же адреса, что и API.

Выкатка

- **Только после 21:00 МСК.** Перезапуск бэкенда обрывает текущие разговоры, а днём простой телефонии видит вся компания. Читать логи можно в любое время.
- На сервере нет git: код копируется из `main` (rsync, без `--delete`). Перед этим — бэкап `scripts/pbx-backup.sh`. Дальше по обстоятельствам: `npm ci --omit=dev`, если менялись зависимости; `npm run migrate`, если появились миграции; перезапуск systemd-юнита `pbx-backend`, если менялся `backend/`. Правки `admin/`, `guide/`, `leadboard/` — статика, перезапуск не нужен.
- Проверка: `https://pbx.donula26.ru/health` отвечает `"asterisk":"connected"`, в журнале юнита — `Stasis app "pbx-dialer" started`. Виджеты сами переподнимают SIP примерно через 5 с после старта (сообщение `force_sip_reconnect`).
- Внешней пробы в мониторинге у pbx сейчас нет, поэтому окно обслуживания на статус-странице открыть не на что.

На что наступают

- **Виджета amoCRM в репозитории нет.** В amoCRM он загружен архивом, исходники не найдены. Меняя протокол WebSocket или `/api/widget/`, помните, что клиент поправить нечем.
- **Миграции не поднимают базу с нуля.** Части таблиц (`agent_groups`, `agent_campaigns`, `settings`, `trunks`, `transfers` и др.) нет ни в одной миграции, в нумерации дыры и дубли (два `028_*`, два `029_*`). На пустой базе `npm run migrate` падает.
- **Без `ASTERISK_AMI_PASSWORD` процесс падает на старте**, даже если Asterisk не нужен.
- **ARI подключается один раз, при старте.** Если Asterisk поднялся позже бэкенда, процесс жив, но не звонит; в `/health` — `"asterisk":"offline"`. Лечится перезапуском бэкенда после Asterisk.
- **Прод стартует `bootstrap.js`, а `npm start` — `app.js`.** `bootstrap.js` ограничивает параллельные HTTP-соединения к Asterisk шестью; локально этого ограничения нет. `src/config/app.js` — устаревшая копия `app.js`, её никто не импортирует.

- **Бэкенд должен жить на одном хосте с Asterisk.** Он сам пишет rjsip-конфиги, вызывает `asterisk -rx`, `ffmpeg/ffprobe`, а серверный путь к генератору отчёта постобработок зашит в код.
- `rsync --delete` **сотрёт то, чего нет в git:** установщики софтбоа в `guide/app` и файлы, которые генерирует сервер. `admin/wrapup-report.html` и `admin/.wrapup-status.json` в git лежат старыми копиями — при выкатке их исключают, иначе затрётся свежий отчёт.
- **Кнопка «Перезапустить Asterisk» в админке** рвёт все звонки разом.
- **Поддомен amoCRM по умолчанию в коде** — `donulya2`, а админка и скрипты ссылаются на `donula.amocrm.ru`. `AMO_SUBDOMAIN` задавать явно.
- **Блокировка «Готов» работает в режиме fail-open:** если от amoCRM 5 минут нет вебхуков, пропускает всех. Аварийный откат — `READY_CHECK_ENABLED=0`.
- **Дедупликация лидов «Охотников» по умолчанию в dry-run** — решения только в логге. Реальные действия в amoCRM — с `DEDUP_DRY_RUN=0`.
- **Запись второго плеча перевода** по умолчанию пишется в другой каталог (`RECORDING_DIR`), чем тот, из которого раздаёт `/api/recordings`, — по ссылке её не получить.
- `.env.example` **не совпадает с кодом:** 8 переменных код не читает, около 20 читаемых там нет. Сверяйтесь с таблицей в README.
- **Ошибки в GlitchTip не уходят** — SDK в коде нет. Смотреть журнал юнита и `logs/error.log`.