исходная архитектура потоков воркфлоу

Ниже — архитектура “минимально жизнеспособного” агент-бота в n8n для MAX → Voronka CRM → (OCR/документы) → Google Calendar → уведомления из CRM. Я опишу **какие workflow нужны**, **какие ноды в каждом**, **где хранить состояние**, и **как сделать подключение Google Calendar на пользователя** так, чтобы это масштабировалось (один бот, много пользователей/чатов).

---

0) Базовые принципы, чтобы система не развалилась

1. **LLM = только “интент + сущности + вопросы”**
   Все реальные действия (создать клиента, создать встречу, прикрепить паспорт) выполняются **детерминированными нодами** (HTTP Request/DB/CRM API). Это стандартный мировой паттерн “LLM as router / extractor”.
2. **Состояние диалога хранится вне n8n**
   n8n хорошо оркестрирует, но плохо хранит долгие “слоты” и контекст. Поэтому:
   - быстрые ожидания/TTL: **Redis**
   - долговременное: **Postgres** (у тебя уже есть)
3. **Один бот — много пользователей/чатов** ⇒ нужна таблица идентификации
   Минимально: `max_user_id`, `chat_id`, `tenant_id / vclient`, `node_vpn_ip`, роли/права, токены Google.
4. **Файлы и паспорта**: всегда “сырьё” в MinIO + ссылкой дальше
   Не гоняем большие байты через десяток нод. Сохранил → получил `s3://bucket/key` → передал дальше.

---

1) “Минимальный набор workflow” в n8n (на перспективу)

WF-1. MAX Inbound Router (единая точка входа)

Цель: принять любое сообщение (текст/фото/файл), определить пользователя/чат, понять намерение и запустить нужную ветку.

Минимальные ноды:

1. **Trigger**: MAX Trigger (если используешь community node) *или* Webhook node (MAX подписка на вебхуки: HTTPS, допустимые порты — у MAX это требование) (MAX для разработчиков)
2. **IF**: проверка `x-max-bot-api-secret` (как ты уже видишь в заголовках)
3. **Code/Set**: нормализация входа в единый формат:
   - `chat_id`, `max_user_id`, `message_id/mid`, `text`, `attachments[]`
4. **Postgres**: `tenant_resolve` (по max_user_id/chat_id найти vclient/node/права)
5. **Switch**: по типу входа:
   - `text` → в LLM-парсер
   - `attachments` → в файловый конвейер
6. **LLM (DeepSeek)**: твой системный промт → вернуть JSON-команду
7. **Parse JSON** (Code): распарсить строку → объект
8. **IF**: `missing_fields`? → ветка “уточнить”
9. **Switch** по `action` → дергаем “WF-x исполнители”
10. **Отправка ответа в MAX** (Send Message)

Шаблон/пример для MAX-бота в n8n уже есть: “AI chatbot for Max Messenger …” (там ещё и voice) — можно импортировать и выкинуть лишнее. (n8n)

WF-2. File Ingest (фото/паспорт/PDF/DOCX/XLSX/txt)

Цель: скачать вложение, положить в MinIO, получить текст/данные (OCR/парсинг), вернуть в WF-1 “готовую сущность”.

Ноды:

1. **HTTP Request**: скачать файл по `payload.url`
2. **MinIO/S3 node или HTTP**: положить в bucket (например `crm-inbox-raw`)
3. **Router по типу**:
   - image → OCR microservice

- pdf/docx/xlsx/txt → document parser microservice

4. **HTTP Request**: OCR/парсер → получить structured text
5. **Return**: упаковать результат как “input.text + input.file_ref” и передать обратно в LLM или сразу в действие `recognize_passport/create_client_from_passport/...`

WF-3. CRM Action Executor (создать/найти/обновить клиента, прикрепить документы)

Цель: единый исполнитель действий с Voronka.pro (через API/вебхук), чтобы WF-1 не разрастался.

Ноды:

1. **HTTP Request**: `api. voronka. pro` (resolver + маршрутизация на node1 по VPN)
2. **HTTP Request**: вызов CRM API на конкретной ноде (внутренний VPN адрес)
3. **Postgres**: запись связок (например `client_id`, `passport_doc_id`, `source_mid`)
4. **Return**: результат (например номер клиента)

WF-4. Google Calendar Connect (OAuth-привязка календаря к пользователю)

Цель: дать пользователю ссылку, он авторизуется Google, и ты получаешь refresh token.

Почему отдельный WF: multi-user OAuth в n8n **не “из коробки”** удобен — это частая тема, люди делают “dynamic credentials / broker” подход. (n8n Community)

Ноды:

1. **Trigger**: из WF-1 (команда “подключить календарь”)
2. **HTTP Request**: к твоему Laravel `api. voronka. pro` → “создать OAuth-линк” (state = max_user_id + nonce)
3. **Send Message (MAX)**: отправить ссылку пользователю
4. **Webhook (callback)**: (лучше пусть callback принимает Laravel, а не n8n) Laravel сохранит refresh token и дернет **внутренний webhook** в n8n “calendar_connected”
5. **Send Message (MAX)**: “календарь подключен ”

(Можно и напрямую через n8n-credentials, но тогда ты либо вручную плодишь креды на каждого пользователя, либо строишь сложный обход — есть даже шаблон, который автоматизирует создание Google OAuth кредов в n8n, но это всё равно про n8n-credential storage. (n8n))

WF-5. Create Meeting (Google Calendar + запись в CRM по клиенту)

Цель: команда “создай встречу …” → событие в календаре пользователя → запись/комментарий/активность в CRM в карточке клиента.

Ноды:

1. **Input**: из WF-1 (action = schedule_meeting)
2. **Postgres**: взять “Google refresh token” пользователя
3. **HTTP Request**: к твоему “OAuth broker” (Laravel) → получить access token (или прокси-вызов календаря)
4. **Google Calendar**: Create Event (или HTTP Request в Calendar API)
5. **HTTP Request**: в CRM (создать Activity/Comment/Task с ссылкой на event/meet link)
6. **Send Message (MAX)**: подтверждение + дата/время

WF-6. CRM → Bot Notifications (напоминания/чек-листы)

Цель: CRM по событию шлет webhook → n8n отправляет человеку/в группу задачи.

Ноды:

1. **Webhook**: входящий от CRM
2. **Postgres**: найти `chat_id`/пользователей, кому слать
3. **Send Message (MAX)**: текст + кнопки (если используешь callbacks, MAX это поддерживает) (MAX для разработчиков)

WF-7. Error & Audit (чтобы не забивать логи, но видеть ошибки)

Цель: ошибки в любом workflow → в один канал (лог-таблица + уведомление админам).

Ноды:

1. **Error Trigger** (глобальный)
2. **Postgres**: записать кратко (workflow, node, error, mid, tenant)
3. **Send Message (MAX) / Email**: только критичные (по фильтру)

---

2) Что лучше “не писать с нуля”: готовые основы/шаблоны

1. **MAX + AI-бот шаблон n8n**
   Есть готовый workflow именно под MAX (включая voice). Его можно взять как “скелет”, оставив только MAX Trigger/Send + твою бизнес-логику. (n8n)
2. **Шаблоны “бот + календарь/бронь”**
   У n8n много готовых booking-workflow под Google Calendar (проверка слотов, создание события, подтверждения). Их можно адаптировать под твою команду “создай встречу”. (n8n)
3. **Каталоги шаблонов (можно импортировать JSON и менять)**
   - Категория “AI Chatbot” в n8n (сотни шаблонов) (n8n)
   - `awesome-n8n-templates` (огромный набор JSON) (GitHub)
   - архив официальных n8n workflows (удобно версионировать/таскать оффлайн) (GitHub)
4. **Community node для MAX**
   Чтобы не собирать руками HTTP-вызовы: `n8n-nodes-max`. (GitHub)
   (И вообще n8n официально описывает установку community nodes и их режимы) (docs.n8n.io)

---

3) Первая “боёвая” настройка: подключение Google Calendar пользователем

Почему я предлагаю OAuth через твой Laravel (api.voronka.pro), а не “чисто n8n”

Потому что у тебя **один бот и много пользователей**, и каждому нужен **свой** Google доступ. В n8n это упирается в “dynamic credentials / multi-tenant OAuth” (у сообщества это отдельная боль, есть обсуждения и обходы). (n8n Community)
Laravel-broker решает это красиво: **хранит refresh token на пользователя**, а n8n всегда просто запрашивает “дай access token для max_user_id”.

Минимальный user-flow

1. Пользователь в MAX пишет: **“Подключить календарь”**
2. WF-1 распознаёт intent → вызывает WF-4
3. WF-4 запрашивает у `api.voronka.pro` ссылку Google OAuth (с state = max_user_id + одноразовый nonce)
4. Бот отправляет ссылку в чат
5. Пользователь проходит Google consent
6. Laravel сохраняет refresh token (привязка к max_user_id/tenant_id)
7. Laravel вызывает внутренний webhook n8n “calendar_connected”
8. n8n отправляет пользователю: **“Google Calendar подключен ”**

Что это даёт дальше

Когда пользователь попросит “создай встречу…”, WF-5:

- берет refresh token по max_user_id
- создаёт событие в календаре именно этого пользователя
- фиксирует связку в CRM/у клиента

---

4) Как это ложится на твою инфраструктуру (центральный сервер + node1)

- **n8n на центральном**: оркестрирует, хранит минимум, очереди/локи в Redis, аудиты в Postgres.
- **api.voronka.pro (Laravel/MySQL)**: идеальное место для:
  - tenant resolver (max_user_id → vclient/node)
  - OAuth broker (Google tokens per user)
  - политика доступа (кто что может)
- **node1 (CRM + базы клиентов)**: принимает только “чистые команды” от n8n (создать/найти/прикрепить документ/создать activity), по VPN адресу.
- **MinIO**: raw-файлы/паспортные сканы/промежуточные результаты (с TTL/корзиной).

---

5) Что делаем прямо сейчас, чтобы “встреча в календаре” стала реальной

Без кода, только шагами:

1. В WF-1 добавляешь команду **“подключить календарь”** (можно через LLM или просто if по тексту).
2. Делаешь WF-4 “Google connect”:
   - нода HTTP Request → `api.voronka.pro` получить auth_link
   - Send Message → отдать ссылку в MAX
3. В Laravel добавляешь страницу OAuth (ты и так можешь, у тебя billing на Laravel) — она хранит refresh token и дергает n8n webhook.
4. После “подключено” — запускаем WF-5: команда “создай встречу …” → Create Event → ответ в MAX.

Если ты скажешь, как ты хочешь решать **идентификацию пользователя с твоим tenant/vclient** (например: “бот доступен только авторизованным в billing, привязка по email/коду”), я предложу самый короткий onboarding-флоу (2–3 сообщения в MAX) и какие таблицы минимально нужны, чтобы дальше спокойно прикручивать CRM и паспорта.

Revision #1
Created 14 February 2026 03:04:37 by Admin
Updated 14 February 2026 03:04:56 by Admin