

Wialon Message Hub — техническое описание архитектуры

Документ основан на анализе исходного кода репозитория `WialonMessageExzotron` (состояние на 22.08.2026, ветка `main`, коммит `df5ad53` и позднее) и живой эксплуатируемой версии на `http://93.188.206.185`. Все утверждения ниже подтверждены конкретными файлами и строками кода; предположения не делались.

1. Название продукта

В интерфейсе (шапка панели, `templates/base.html`) и в исходном коде (`app/main.py: FastAPI(title="Wialon Message Gateway", ...)`) используются два незначительно различающихся названия: витринное «**Wialon Message Hub**» (логотип «EXZOTRON — Wialon Message Hub» на каждой странице) и внутреннее техническое «**Wialon Message Gateway**» (**WMG**) — сокращение **WMG** используется как префикс Docker-образов (`wmg-web`, `wmg-collector`, `wmg-dispatcher`), имя каталога на сервере и в комментариях кода. В документации используется публичное название **Wialon Message Hub**, техническое сокращение **WMG** указывается как синоним при первом упоминании в каждом документе.

2. Технологический стек

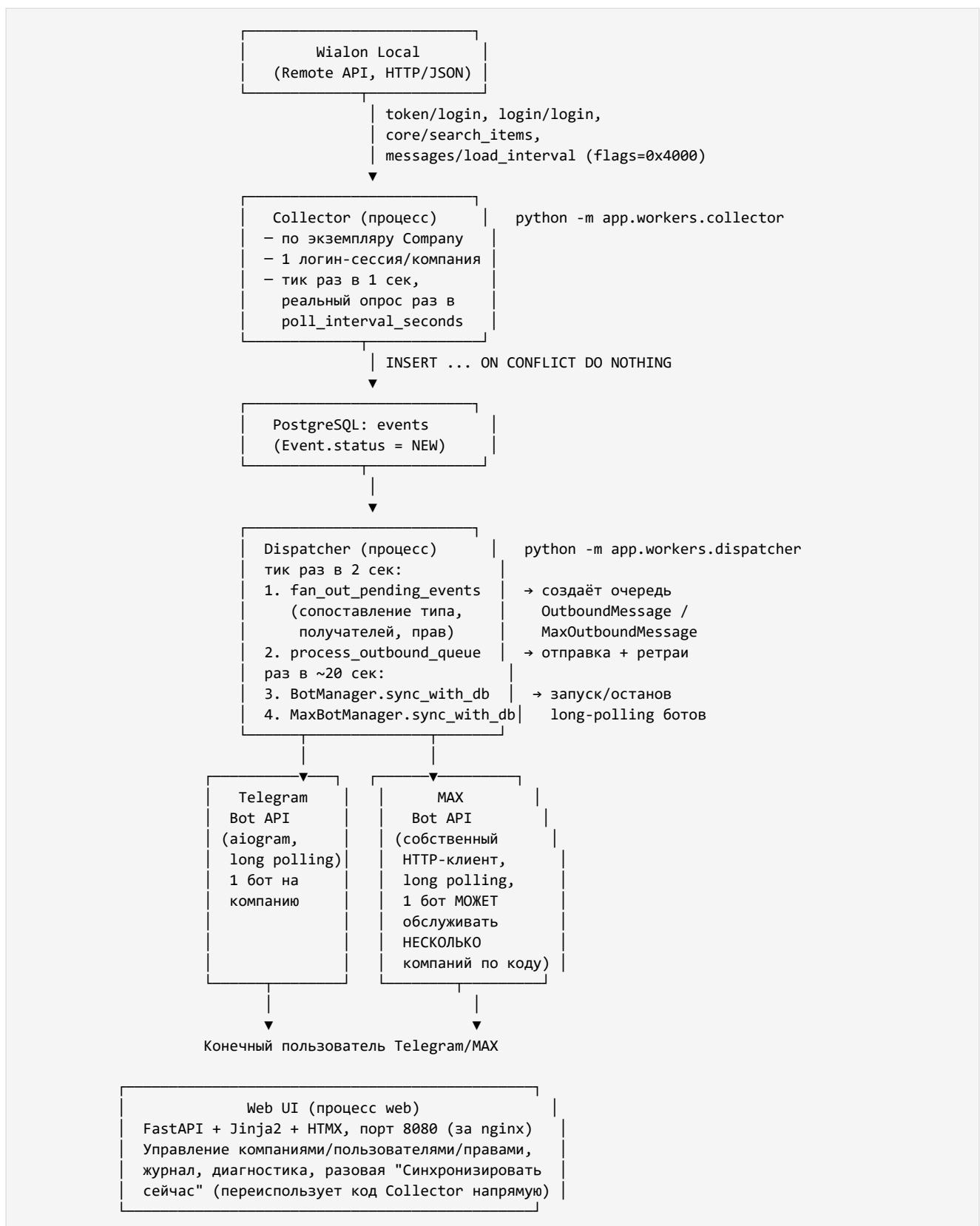
Слой	Технология	Версия	Источник
Язык	Python	3.12	Dockerfile (python:3.12-slim)
Web-framework	FastAPI	0.115.0	requirements.txt
ASGI-сервер	Uvicorn	0.30.6	requirements.txt
ORM	SQLAlchemy (async)	2.0.35	requirements.txt
Драйвер БД	asyncpg	0.29.0	requirements.txt
Миграции	Alembic	1.13.2	requirements.txt, migrations/
СУБД	PostgreSQL	16 (alpine-образ)	docker-compose.yml
Шаблонизатор	Jinja2	3.1.4	requirements.txt, templates/
Динамика на фронтенде	HTMX	1.9.12 (CDN)	templates/base.html
UI-фреймворк	Bootstrap	5.3.3 (CDN) + Bootstrap Icons 1.11.3	templates/base.html
Telegram-бот	aiogram	3.13.1	requirements.txt
HTTP-клиент	httpx	0.27.2	requirements.txt
Шифрование секретов	cryptography (Fernet)	43.0.1	app/security/crypto.py
Хэширование пароля	argon2-cffi (Argon2id)	23.1.0	app/security/passwords.py
Подпись сессий	itsdangerous	2.2.0	app/security/sessions.py

Слой	Технология	Версия	Источник
Автоматизация браузера (получение Wialon-токена)	Playwright	1.47.0 (пакет установлен, браузерный бинарник — нет)	requirements.txt, Dockerfile

Собственная тёмная CSS-тема — `static/css/app.css`, без стороннего дизайн-фреймворка. Отдельного frontend-приложения (SPA, сборки на JS) нет: вся динамика — `server-rendered` HTML-фрагменты, подгружаемые через HTMX (`hx-get/hx-post`, ответы с заголовком `HX-Redirect`).

3. Общая структура компонентов

Фактическая архитектура (подтверждена `app/main.py`, `docker-compose.yml`, `app/workers/*.py`) отличается от предварительной схемы, приложенной к постановке задачи: в системе нет единого «Message Processing»-модуля между Collector и Dispatcher — сбор, дедупликация и запись событий целиком выполняются в Collector, а вся классификация/маршрутизация/форматирование сообщений — в Dispatcher. Также в системе с 20.08.2026 два независимых, параллельных транспортных канала (Telegram и MAX), а не единственный, и у каждого канала собственный подпроцесс регистрации ботов, встроенный в тот же процесс Dispatcher.



Все четыре процесса (web, collector, dispatcher, migrate — одноразовый) собираются из одного и того же Dockerfile и работают в отдельных Docker-контейнерах (docker-compose.yml), обращаясь к общей базе PostgreSQL. Контейнер nginx — единственная публично открытая точка входа (порт 80), проксирует все запросы на web:8080.

4. Многокомпанейность (multi-tenancy)

Система с самого начала спроектирована как многокомпанейная: одна инсталляция WMG обслуживает произвольное число независимых компаний-клиентов (`app/models/company.py`). У каждой компании:

- собственные учётные данные Wialon (`CompanyWialonSettings: server_url, token_encrypted, login, password_encrypted`);
- собственные токены Telegram и MAX (`CompanyTransportSettings`);
- независимые флаги `collector_enabled/dispatcher_enabled` (раздельный RUN/STOP сбора и рассылки);
- собственный курсор синхронизации (`last_sync_at, sync_from`);
- собственный справочник типов уведомлений (`NotificationType.company_id`);
- собственные пользователи Telegram и MAX (`TelegramUser.company_id, MaxUser.company_id`) и их права (`UserNotificationPermission, MaxUserNotificationPermission` — транзитивно через `user_id→компания` и `notification_type_id→компания`).

Изоляция данных между компаниями обеспечивается фильтрацией по `company_id` практически во всех запросах (подтверждено по всем веб-роутерам и сервисам `Dispatcher/Collector`); межкомпанийных утечек данных в проверенном коде не обнаружено.

Важное архитектурное исключение — MAX. В отличие от Telegram, где Telegram Bot API физически не допускает двух параллельных long-polling клиентов на одном токене (второй клиент получает `TelegramConflictError`), MAX Bot API это допускает молча. Из-за этого один физический MAX-бот технически может быть подключён сразу к нескольким компаниям одновременно (см. раздел 8) — это единственный случай в системе, где строгая изоляция «одна компания — один канал» опциональна и требует явной настройки кода регистрации.

5. Хранение и защита секретов

Все учётные секреты (Wialon-пароль и Wialon-токен, Telegram Bot Token, MAX Bot Token) хранятся в PostgreSQL зашифрованными симметричным алгоритмом Fernet (`app/security/crypto.py`, ключ `APP_MASTER_KEY` из `.env` сервера, не хранится в БД и не публикуется в репозитории). В интерфейсе секреты никогда не показываются полностью — только маскированное значение вида `c7a242*****31E` (первые 6 и последние 3 символа, `mask_secret`). Пароль администратора хранится хэшированным по Argon2id, обратимо не восстанавливается.

Подробный разбор безопасности (включая обнаруженные ограничения — отсутствие HTTPS и нерабочий CSRF) вынесен в отдельный документ «Выявленные проблемы и рекомендации по развитию».

6. Классификация событий — фактический алгоритм

```

Wialon (messages/load_interval, flags=0x4000)
|
▼
Event.wialon_notification_id = p.notification_id | p.nid (может отсутствовать)
Event.notification_name      = p.name | p.n | raw.tp
|
▼
Dispatcher.fan_out_pending_events:
|
├─ 1) ищем NotificationType по company_id + wialon_notification_id
│   (справочник синхронизирован ОТДЕЛЬНО через core/search_items)
├─ 2) если не нашли – fallback: ищем NotificationType
│   по company_id + name == notification_name
|
▼
если тип не найден вообще – событие ждёт следующего цикла Collector
(тип ещё не синхронизирован)
|
▼
находим получателей: TelegramUser/MaxUser с enabled=true,
status=APPROVED, messages_paused=false, у которых есть
UserNotificationPermission/MaxUserNotificationPermission(enabled=true)
на найденный NotificationType
|
▼
создаём OutboundMessage/MaxOutboundMessage(status=PENDING)
на каждого получателя, в обоих каналах независимо

```

Ключевая деталь, задокументированная прямо в коде (`app/services/dispatcher/logic.py`, комментарий к `fallback`-поиску): числовой `wialon_notification_id`, который приходит в теле сработавшего события, и `wialon_notification_id`, которым заполнен справочник `NotificationType` (получен отдельным вызовом `core/search_items`), на части `Wialon`-серверов не совпадают или отсутствуют в событии вовсе — поэтому основной путь сопоставления по ID часто не срабатывает, и решающим оказывается сопоставление по точному совпадению названия уведомления.

7. Защита от дублирования — два независимых уровня

- Уровень событий (Wialon → БД).** Уникальный индекс `UniqueConstraint(company_id, external_event_key)` на таблице `events`, вставка через `INSERT ... ON CONFLICT DO NOTHING` (`app/workers/collector.py`). Ключ `external_event_key` формируется в `app/services/wialon/notifications.py::build_external_event_key`: если в сыром сообщении `Wialon` есть натуральный идентификатор (`p.id`, `p.event_id` или `i`) — ключ `wid:<id>`; если нет — SHA-256-хэш (первые 32 hex-символа) от строки `"{company_id}|{unit_id}|{notification_name}|{trigger_time}|{text}"` с префиксом `h:`. Это защищает от повторной вставки одного и того же события при перехлёсте временного окна опроса (см. раздел 8 документа `Collector`) — окно намеренно перехлёстывается на 30 секунд (`OVERLAP_SECONDS`) назад при каждом опросе, чтобы не потерять события на границе окна, а уникальный индекс гарантирует, что повторно прочитанные события не задвоятся в базе.
- Уровень раскладки по получателям (fan-out).** `fan_out_pending_events` обрабатывает только события, для которых ещё нет ни одной записи в очереди отправки (`OutboundMessage.id IS NULL` через `LEFT JOIN`) — событие раскладывается по получателям ровно один раз.

8. MAX: маршрутизация одного бота на несколько компаний

Инцидент 20.08.2026 обнажил архитектурное отличие MAX от Telegram (см. раздел 4): два независимых цикла опроса с одним и тем же токеном MAX получали одно и то же входящее сообщение /start от пользователя дважды и регистрировали его сразу в двух компаниях. Архитектура была переработана:

- MaxBotManager (app/services/max/manager.py) группирует активные компании не по company_id, а по **значению самого токена** — на один уникальный токен запускается ровно один long-polling цикл, независимо от того, сколько компаний его используют.
- Внутри цикла, обслуживающего несколько компаний (bot_runtime.py::_handle_message), маршрутизация конкретного пользователя к конкретной компании выполняется по коду регистрации (Company.registration_code, то же поле, что используется и для Telegram): пользователь обязан прислать /start код_компании; бот ищет среди обслуживаемых им компаний ту, чей registration_code совпал, и регистрирует пользователя только там.
- Если бот обслуживает ровно одну компанию и код для неё не задан — сохранено прежнее упрощённое поведение («написал /start — сразу зарегистрирован»), чтобы не ломать уже настроенные одиночные интеграции.
- Пользователь, зарегистрированный через один и тот же MAX-аккаунт сразу в нескольких компаниях (по нескольким кодам, в разное время), корректно обрабатывается для команд /status (сводка по всем регистрациям), /stop//start_messages (применяются ко всем регистрациям сразу).

9. Обработка сбоев Wialon-сессии — три исправленных дефекта Collector

Инцидент 20.08.2026 (обе тестовые компании стабильно показывали wialon: ERROR) выявил и позволил исправить три отдельных дефекта в Collector, зафиксированных прямо в коде комментариями с датой:

1. **Отсутствие повторного логина после порчи сессии не при входе.** _ensure_login пропускал повторный вход, если в памяти уже был объект wialonSession, никак не проверяя, жива ли сессия на стороне Wialon. Если сессию выбивало (например, параллельным логином той же учётной записи через кнопку «Проверить Wialon» в web-интерфейсе — Wialon Local, по наблюдениям, ограничивает число одновременных сессий на один токен), Collector бесконечно повторял wialon API error 1 (Invalid session), никогда не пытаясь войти заново. Исправлено: любая wialonError, возникшая после успешного входа (не только при самом логине), теперь сбрасывает объект клиента в памяти, и следующий тик форсирует новый вход.
2. **Гонка при параллельном запуске синхронизации одной компании.** Планировщик тиков не дожидался завершения предыдущей синхронизации компании, если та выполнялась дольше poll_interval_seconds — два параллельных запуска делили один и тот же изменяемый объект состояния без синхронизации, что после исправления пункта 1 стало проявляться как AttributeError (обнуление объекта клиента одним запуском посреди работы другого). Исправлено добавлением флага in_progress на объект состояния компании — вторая синхронизация не запускается, пока не завершилась первая.
3. **Ошибка одного объекта роняла сбор по всей компании.** Если Wialon отклонял запрос по конкретному объекту (например, error 7 – доступ запрещён, объект убран из ресурса пользователя), исключение прерывало цикл по всем объектам компании, включая доступные — компания целиком помечалась ERROR, хотя события по остальным объектам собрать было можно. Исправлено оборачиванием запроса по одному объекту в отдельный try/except: недоступный объект логируется и пропускается, остальные обрабатываются штатно.

10. Расхождения с исходной постановкой задачи

№	В постановке задачи предполагалось	Фактически реализовано
1	Единая цепочка «Wialon → Collector → определение компании → определение типа → определение объекта → определение получателей → фильтрация → формирование сообщения → Dispatcher → Telegram/MAX» как последовательность внутри одного модуля	Функционально то же самое, но разнесено по двум независимым процессам: весь сбор и дедупликация — в Collector, вся классификация/маршрутизация/форматирование и отправка — в Dispatcher; «определение компании» не отдельный шаг — Collector изначально работает в разрезе одной компании (по одной сессии Wialon на компанию)
2	«Wialon Message Hub» как рабочее название решения	Подтверждено как публичное название интерфейса; внутри кодовой базы и инфраструктуры (имена Docker-образов, заголовков FastAPI-приложения) используется техническое сокращение «Wialon Message Gateway» / WMG — оба названия относятся к одному и тому же продукту, расхождения по сути нет, оставлено как есть
3	Единая, общая для всех каналов доставка и общая логика регистрации	Telegram и MAX реализованы как полностью независимые модули с отдельными таблицами данных (telegram_users/max_users, outbound_messages/max_outbound_messages) — сознательное архитектурное решение ради безопасности на живой продакшн-системе (см. app/models/max_outbound_message.py, докстринг), а не общая абстрактная «доставка»
4	Права пользователя на «объекты Wialon» (раздел 19 постановки задачи, схема с деревом объектов)	Права реализованы на уровне типа уведомления , а не конкретного объекта/группы объектов — в моделях UserNotificationPermission/MaxUserNotificationPermission нет ни одного поля, ссылающегося на конкретный объект Wialon (см. документ «Техническое описание архитектуры», раздел 7 отчёта многокомпанейности)
5	«Один MAX-бот — одна компания» как подразумеваемая по умолчанию модель (аналогично Telegram)	Обнаружено и осознанно поддержано архитектурой: один физический MAX-бот может обслуживать несколько компаний одновременно, с маршрутизацией по коду регистрации (раздел 8 этого документа) — это не ошибка, а следствие того, что MAX Bot API, в отличие от Telegram, технически допускает несколько параллельных клиентов на одном токене

№	В постановке задачи предполагалось	Фактически реализовано
6	Кнопка «Повторить отправку» как часть штатного функционала (раздел 25 постановки задачи)	Серверная логика реализована, элемент управления в веб-интерфейсе — нет (см. документ «Выявленные проблемы и рекомендации», п. 2.2)
7	Резервное копирование как часть штатной эксплуатации (раздел 43 постановки задачи)	Не реализовано ни в коде, ни в инфраструктуре (см. документ «Выявленные проблемы и рекомендации», п. 2.1)

11. Схема базы данных (сводно)

15 таблиц, 3 линейные Alembic-миграции (0001_initial → 0002_max_messenger → 0003_max_company_status):

admins, companies, company_wialon_settings, company_transport_settings, notification_types, events, telegram_users, user_notification_permissions, outbound_messages, max_users, max_user_notification_permissions, max_outbound_messages, service_state, system_log, audit_log.

Полный список полей каждой таблицы, значимых для эксплуатации, приведён в разделах соответствующих руководств (компании — в руководстве администратора, права — в руководстве пользователя).