TG-БОТ - пакет документации API

1. API Overview

1.1 Назначение

Документ фиксирует правила интеграции Telegram с нашим backend:

- Inbound: Telegram отправляет Update на наш webhook.
- Outbound: наш backend вызывает Telegram Bot API для отправки сообщений и обработки кликов по кнопкам.

1.2 Границы ответственности

- Telegram Platform: доставляет сообщения/клики и принимает наши вызовы Bot API.
- Telegram Bot API: HTTP API для исходящих команд бота.
- Наш backend: принимает webhook, выполняет бизнес-логику, пишет/читает нашу БД, вызывает Bot API.
- Наша БД: хранит доменные данные проекта. Telegram не пишет в БД напрямую.

1.3 Участники (логические сущности)

- E1 Пользователь (клиент Telegram)
- E2 Telegram (Platform + Bot API)
- E3 CMS (источник контента, пакетные обновления)
- E4 Карты (внешнее приложение, открывается по ссылке)

1.4 Хранилища (DFD L1, D1-D6)

Это предметные данные в единой БД, используемые логикой TG-бота:

- D1 Пользователи: профиль TG и статус подписки на канал.
- D2 Заведения и справочники: места, районы, типы, подкатегории и связи.
- D3 Мероприятия и справочники: события, категории, источники.
- D4 Избранное: связи пользователь-место и пользователь-событие.
- D5 Подборки: подборки и элементы подборок.
- D6 Акции: промокоды и акции (единые, без учета выдачи).

Служебные данные (логи, метрики, traceId, дедуп) не относятся к D1-D6 и живут в инфраструктурном контуре.

1.5 Процессы (P1-P10)

- P1 Онбординг и проверка подписки
- P2 Навигация и роутинг разделов
- P3 Поиск мест (заведения): списки, фильтры, пагинация
- P4 Афиша: список событий, фильтры, пагинация
- P5 Подборки: списки подборок и переход
- P6 Карточка объекта: место или событие, действия
- P7 Избранное: добавить, удалить, список, переход в карточку

- P8 Промокоды: список акций и переходы
- P9 Помощь и обратная связь: помощь в чат, обратная связь без хранения в БД
- P10 Загрузка и обновление контента из CMS: upsert в D2/D3/D5/D6 (отдельный контур, не часть webhook/Bot API)

1.6 Ключевые правила (обязательные)

1. Webhook ACK: backend подтверждает прием Update быстрым HTTP 200 с пустым телом.
2. Идемпотентность inbound: один Update может прийти повторно, дедупликация по `update_id`.
3. Inline-кнопки: клики приходят как `callback_query`, backend отвечает `answerCallbackQuery` для снятия ожидания на кнопке.
4. Избранное: хранится только для TG-ветки, без синхронизации с сайтом в MVP.
5. Промокоды: единые, без учета выдачи пользователям.
6. Обратная связь: текст обращений в БД не храним, пользователю выдаем контакт/ссылку.

1.7 Термины (кратко)

- Update: входящее событие от Telegram (сообщение или клик).
- Webhook: HTTPS endpoint нашего backend, куда Telegram отправляет Update.
- ACK: HTTP 200 на webhook как подтверждение приема.
- Bot API: методы Telegram для исходящих команд бота (sendMessage и др.).
- CallbackQuery: событие клика по inline-кнопке.
- Callback data: строка параметров действия, которая приходит в `callback_query.data`.

---

2. Inbound API Contract - Telegram Webhook

2.1 Назначение и границы

Документ фиксирует контракт входящей интеграции: доставка событий от Telegram в backend TG-бота через webhook.

- Вход: HTTP POST с JSON-объектом Update.
- Выход: HTTP `200` от backend как подтверждение приема (ACK). Payload не требуется.
- Исходящие вызовы Bot API (sendMessage, getChatMember, answerCallbackQuery и т.п.) описаны в секции 3 (Outbound API).

Ключевая граница: Telegram не пишет в нашу БД, все записи выполняет только backend.

2.2 Транспорт и точка входа

- Протокол: HTTP(S)
- Метод: POST
- Content-Type: application/json
- Endpoint (в нашей системе): `POST /api/v1/integrations/telegram/webhook`

Примечание: хост/домен и внешний URL для регистрации webhook задаются конфигурацией окружения и не являются частью контракта (на уровне пути фиксируем только маршрут).

2.3 Аутентификация и защита входа

2.3.1 Верификация источника запроса

Используем секретный токен webhook на стороне Telegram и валидируем входящий заголовок:

- Header: `X-Telegram-Bot-Api-Secret-Token: <secret_token>`

Правила:

- `secret_token` длина 1-256, допустимые символы: A-Z a-z 0-9 _ -
- При отсутствии/несовпадении токена backend возвращает `401` Unauthorized

Пример запроса (локальный тест):

```
curl -X POST "https://api.example.com/api/v1/integrations/telegram/webhook" \
  -H "Content-Type: application/json" \
  -H "X-Telegram-Bot-Api-Secret-Token: <secret_token>" \
  -d '{"update_id":10000001,"message":{"message_id":10,"from":
{"id":555000111,"is_bot":false,"first_name":"User"},"chat":
{"id":555000111,"type":"private"},"date":1700000000,"text":"/start"}}'
```

2.4 Ответ backend (ACK)

Webhook используется только как транспорт доставки Update.

- Успешный прием: HTTP 200 OK, body пустой
- Вызовы Bot API не выполняем "ответом на webhook", только отдельными исходящими запросами backend

Задача ACK:

- подтвердить Telegram, что событие принято
- не провоцировать повторные доставки из-за ожидания долгой обработки

2.5 Модель входящих событий (Update)

2.5.1 Общая схема

Тело webhook-запроса - объект Update. В одном Update присутствует не более одного типа события (message, callback_query и т.п.).

Ключевое поле:

- `update_id` (integer) - уникальный идентификатор события, используется для идемпотентности (повторы возможны)

2.5.2 Типы Update, используемые в MVP

Фиксируем 2 ключевых типа, покрывающих сценарии P1-P9:

1. `message`

- команды и текстовые сообщения (включая `/start`)

2. `callback_query`

- нажатия inline-кнопок в сообщениях бота (меню, списки, пагинация, избранное, подробнее и т.п.)

2.6 Извлечение идентификаторов из Update (для маршрутизации)

Используем единые правила извлечения (нужно для связки с D1 и для исходящих ответов):

- `telegram_user_id`:
  - `message.from.id` или `callback_query.from.id`
- `chat_id` (куда отвечаем):
  - `message.chat.id` или `callback_query.message.chat.id`

2.7 Контракт данных для callback_query.data

Ограничение Telegram: `InlineKeyboardButton.callback_data` до 64 байт.
Фиксируем компактный формат без персональных данных.

2.7.1 Формат

`action:<entity_type>:<entity_id>[:<extra>]`

Где:

- `action` - код операции (пример: `open`, `fav_add`, `fav_remove`, `page`, `back`)
- `entity_type` - тип сущности (`place`, `event`, `collection`, `promo`, `menu`)
- `entity_id` - id сущности из нашей БД (целое число) или служебный код для `menu`
- `extra` - доп параметры (страница/фильтр), если требуется

Примеры:

- `open:place:123`
- `open:event:456`
- `open:collection:77`
- `open:promo:55`
- `fav_add:place:123`
- `fav_remove:event:456`
- `open:menu:places`
- `page:place:0:2` (пример: page=2, реализацию кодирования extra фиксирует backend)

2.7.2 Правило обработки callback_query

После получения `callback_query` backend должен выполнить исходящий вызов Bot API `answerCallbackQuery` (чтобы клиент Telegram не "крутил" ожидание на кнопке). Детали вызова описываются в outbound-части.

2.8 Правила обработки webhook (нормы реализации)

2.8.1 Идемпотентность

Backend должен устойчиво обрабатывать повторы одного и того же Update.

Минимальные требования:

- не создавать дубликаты пользователя по `telegram_id`
- повторные нажатия кнопок не должны ломать состояние (избранное, пагинация, повторные открытия карточки)

Практическое решение (MVP):

- дедупликация по `update_id` в рамках короткого окна (in-memory cache или Redis)
- если событие уже обработано - вернуть 200 OK и не выполнять повторных действий

2.8.2 Проверка подписки (P1)

Факт подписки на канал не генерирует событие в webhook.
Проверка статуса выполняется при следующем входящем действии пользователя (message или callback_query).

2.8.3 Обратная связь (P9)

В MVP "обратная связь без хранения", поэтому входящие события не приводят к записи текстов обращений в БД.

2.9 Ошибки приема (уровень транспорта)

- `200 OK` - событие принято (в том числе если payload не обработан из-за валидации, тип не поддержан или `callback_data` некорректен).
- `401 Unauthorized` - неверный/отсутствующий `X-Telegram-Bot-Api-Secret-Token`.
- `500 Internal Server Error` - только при фатальной ошибке до возможности сформировать ответ (крайний случай).

Примечание по ретраям:

- Backend inbound не ретраит. Повторы доставки делает Telegram при неуспешном ACK.

2.10 Примеры входящих payload (для разработки и тестов)

2.10.1 Message: команда /start (P1)

```
{
  "update_id": 10000001,
  "message": {
    "message_id": 10,
    "from": { "id": 555000111, "is_bot": false, "first_name": "User" },
```

```
    "chat": { "id": 555000111, "type": "private" },
    "date": 1700000000,
    "text": "/start"
  }
}
```

Примечания:

- `date` - Unix timestamp в секундах
- `from.id` и `chat.id` используются для связывания с `Пользователь.telegram_id`

2.10.2 CallbackQuery: открыть карточку place_id=123 (P6)

```
{
  "update_id": 10000002,
  "callback_query": {
    "id": "abc123",
    "from": { "id": 555000111, "is_bot": false, "first_name": "User" },
    "message": {
      "message_id": 22,
      "chat": { "id": 555000111, "type": "private" }
    },
    "data": "open:place:123"
  }
}
```

---

3. Outbound API Usage - Telegram Bot API

3.1 Назначение и границы

Outbound-часть описывает, как backend отправляет команды в Telegram Bot API:

- чтобы показывать пользователю сообщения и меню (inline-кнопки)
- чтобы проверять подписку на канал (доступ)
- чтобы корректно обрабатывать клики по inline-кнопкам (снимать "ожидание" в клиенте)
- чтобы отправлять изображения (фото) для карточек/акций/избранного согласно требованиям

Webhook-ответ (ACK) не используется для отправки сообщений пользователю. Все пользовательские действия выполняются через Bot API отдельными HTTP-запросами.

3.2 Авторизация и конфигурация

3.2.1 Bot Token

- Bot Token хранится как секрет (env/secret storage).
- Запрещено писать token в логи и отдавать в клиент.
- Все запросы в Bot API выполняются с использованием token.

Рекомендуемый подход:

- В конфиге: `TELEGRAM_BOT_TOKEN`
- В runtime: формируем базовый URL `https://api.telegram.org/bot{TOKEN}/...`

3.2.2 Таймауты и ретраи

- Таймаут на исходящий HTTP-запрос к Bot API должен быть ограничен (например, 3-5 сек).
- Ретраи, backoff и обработка 429/5xx описаны в разделе 4 (Ошибки и ретраи). В этом разделе не дублируем правила.

3.3 Настройка webhook (операция окружения)

Webhook регистрируется один раз (при деплое или смене адреса), отдельным сервисным вызовом.

Рекомендованный вариант:

- `setWebhook` с URL нашего endpoint `POST /api/v1/integrations/telegram/webhook`
- задаем `secret_token`, который backend будет валидировать во входящих запросах

Пример (для DevOps/разработчика, вручную):

```
curl -X POST "https://api.telegram.org/bot<TOKEN>/setWebhook" \
  -H "Content-Type: application/json" \
  -d '{
    "url": "https://api.example.com/api/v1/integrations/telegram/webhook",
    "secret_token": "<secret_token>"
  }'
```

3.4 Используемые методы Bot API (MVP)

В MVP фиксируем минимальный набор методов, который покрывает сценарии P1-P9 и вывод изображений для карточек/акций/избранного.

3.4.1 sendMessage

Назначение:

- основной способ выдачи "экранов" бота: меню, списки, карточки (текстовая версия), помощь
- прикрепление inline-кнопок через `reply_markup.inline_keyboard`

Ключевые поля:

- `chat_id` (куда отправлять)
- `text` (контент)
- `reply_markup` (inline keyboard)

Пример: отправка меню с inline-кнопками

```
curl -X POST "https://api.telegram.org/bot<TOKEN>/sendMessage" \
  -H "Content-Type: application/json" \
  -d '{
```

```
    "chat_id": 555000111,
    "text": "Выберите раздел:",
    "reply_markup": {
      "inline_keyboard": [
        [
          { "text": "Места", "callback_data": "open:menu:places" },
          { "text": "Афиша", "callback_data": "open:menu:events" }
        ],
        [
          { "text": "Подборки", "callback_data": "open:menu:collections" },
          { "text": "Избранное", "callback_data": "open:menu:favorites" }
        ],
        [
          { "text": "Промокоды", "callback_data": "open:menu:promos" },
          { "text": "Помощь", "callback_data": "open:menu:help" }
        ]
      ]
    }
  }'
```

Пример: карточка места (текстовый вариант)

```
curl -X POST "https://api.telegram.org/bot<TOKEN>/sendMessage" \
  -H "Content-Type: application/json" \
  -d '{
    "chat_id": 555000111,
    "text": "Название: Кофейня X\nАдрес: ...\nРейтинг: ...",
    "reply_markup": {
      "inline_keyboard": [
        [
          { "text": "В избранное", "callback_data": "fav_add:place:123" },
          { "text": "Назад", "callback_data": "back:menu:places" }
        ],
        [
          { "text": "Найти на Яндекс-картах", "url": "https://yandex.ru/maps/?text=
<encoded_address>" }
        ]
      ]
    }
  }'
```

3.4.2 sendPhoto

Назначение:

- отправка фото (обложки/первого изображения) для карточек места/события (P6)
- отправка фото заведения для карточек акций/промокодов (P8)
- отправка фото для карточки элемента избранного при открытии (P7)

Правило применения в MVP:

- `sendPhoto` используется для карточек (place/event/promo) - 1 фото (обложка/первое).
- Списки (включая список избранного и список промокодов) в MVP отправляются через `sendMessage` (текст + кнопки открытия карточек), без рассылки фото по каждому элементу списка.

Важно:

- В MVP не загружаем бинарные файлы в Telegram от имени backend.
- Используем URL изображений (Telegram сам скачает изображение по ссылке).
- Если URL отсутствует или недоступен - fallback на `sendMessage` (текст без фото) по правилу раздела 4.10.

Ключевые поля:

- `chat_id`
- `photo` (URL изображения)
- `caption` (опционально, краткое описание)
- `reply_markup` (inline keyboard)

Пример: карточка места с фото

```
curl -X POST "https://api.telegram.org/bot<TOKEN>/sendPhoto" \
  -H "Content-Type: application/json" \
  -d '{
    "chat_id": 555000111,
    "photo": "https://cdn.example.com/photos/place/123/main.jpg",
    "caption": "Кофейня X\nАдрес: ...\nРейтинг: ...",
    "reply_markup": {
      "inline_keyboard": [
        [
          { "text": "Подробнее", "callback_data": "open:place:123" },
          { "text": "В избранное", "callback_data": "fav_add:place:123" }
        ],
        [
          { "text": "Найти на Яндекс-картах", "url": "https://yandex.ru/maps/?text=
<encoded_address>" }
        ],
        [
          { "text": "Назад", "callback_data": "back:menu:places" }
        ]
      ]
    }
  }'
```

Пример: акция/промокод с фото заведения

```
curl -X POST "https://api.telegram.org/bot<TOKEN>/sendPhoto" \
  -H "Content-Type: application/json" \
  -d '{
    "chat_id": 555000111,
    "photo": "https://cdn.example.com/photos/place/123/main.jpg",
    "caption": "Акция 1\nАдрес: ...\nОписание акции: ...\nСрок действия: ...",
    "reply_markup": {
      "inline_keyboard": [
        [
          { "text": "Подробнее о месте", "callback_data": "open:place:123" }
        ],
        [
          { "text": "Назад", "callback_data": "back:menu:promos" }
        ]
```

```
      ]
    }
  }'
```

3.4.3 getChatMember

Назначение:

- проверка подписки пользователя на канал (используется в P1 как проверка доступа)

Ключевые поля:

- `chat_id` (id канала/чата, где проверяем подписку)
- `user_id` (telegram_user_id)

Пример:

```
curl -X POST "https://api.telegram.org/bot<TOKEN>/getChatMember" \
  -H "Content-Type: application/json" \
  -d '{
    "chat_id": "@channel_username_or_id",
    "user_id": 555000111
  }'
```

Интерпретация результата (на уровне backend):

- status `member/administrator/creator` - подписка есть
- status `left/kicked` - подписки нет

3.4.4 answerCallbackQuery

Назначение:

- обязательный быстрый отклик на нажатие inline-кнопки
- снимает "ожидание" у кнопки в клиенте Telegram
- опционально показывает краткое уведомление

Ключевые поля:

- `callback_query_id` (из входящего Update: `callback_query.id`)
- `text` (опционально, короткий toast)
- `show_alert` (опционально)

Правило:

- `answerCallbackQuery` должен выполняться для каждого `callback_query` максимально быстро, независимо от длительности основной бизнес-логики.

Пример: снять ожидание без текста

```
curl -X POST "https://api.telegram.org/bot<TOKEN>/answerCallbackQuery" \
  -H "Content-Type: application/json" \
  -d '{
```

```
    "callback_query_id": "abc123"
  }'
```

3.5 Правила формирования inline-кнопок и callback_data

3.5.1 Формат callback_data (единый)

`action:<entity_type>:<entity_id>[:<extra>]`

Требования:

- `callback_data` не содержит персональные данные, токены, длинные строки
- `entity_id` должен однозначно ссылаться на сущность, доступную backend (место/событие/подборка/акция) или на служебный пункт меню
- длина должна укладываться в лимит Telegram (для callback_data)

3.5.2 Роутинг по callback_data

Рекомендуемая логика в backend:

- парсинг `callback_data` в структуру `{action, entity_type, entity_id, extra}`
- маршрутизация на процесс (P2-P8) по `action/entity_type`
- выполнение бизнес-логики и ответ пользователю через `sendMessage` или `sendPhoto` (в зависимости от наличия фото)
- для любого `callback_query` выполнять `answerCallbackQuery`

3.6 Матрица "процесс -> outbound методы" (MVP)

Процесс	Outbound метод(ы)
P1 Онбординг и проверка подписки	getChatMember, sendMessage
P2 Навигация	answerCallbackQuery, sendMessage
P3 Поиск мест	answerCallbackQuery (если кнопки), sendMessage
P4 Афиша	answerCallbackQuery, sendMessage / sendPhoto (для карточек событий)
P5 Подборки	answerCallbackQuery, sendMessage
P6 Карточка объекта	answerCallbackQuery, sendPhoto / sendMessage (fallback)
P7 Избранное	answerCallbackQuery, sendMessage (список); sendPhoto / sendMessage (карточка, fallback)
P8 Промокоды	answerCallbackQuery, sendMessage (список); sendPhoto / sendMessage (карточка, fallback)
P9 Помощь и обратная связь	sendMessage

3.7 Примечание про методы редактирования сообщений (не MVP)

Если будет принято решение обновлять один и тот же "экран" без спама новыми сообщениями, в этот раздел добавляются методы:

- `editMessageText`
- `editMessageReplyMarkup`

И отдельно фиксируются правила:

- как выбираем message_id для редактирования
- как избегаем конфликтов при повторных кликах и повторах Update (идемпотентность UI-обновлений)

---

4. Ошибки, ретраи и служебные правила (Telegram integration)

4.1 Назначение раздела

Раздел фиксирует нормы надежности интеграции:

- как подтверждаем прием событий (ACK)
- как обеспечиваем идемпотентность и дедупликацию
- как обрабатываем ошибки и повторные попытки (retry policy) для Bot API
- что и как логируем (включая служебные корреляционные атрибуты)
- правила fallback для медиа (sendPhoto)

4.2 Принципиальная схема надежности

Интеграция состоит из двух независимых каналов:

1. Inbound (Telegram -> webhook):

- Telegram доставляет Update на наш webhook
- backend отвечает ACK (HTTP 200) и дальше обрабатывает внутри

2. Outbound (backend -> Telegram Bot API):

- backend вызывает методы Bot API (sendMessage, sendPhoto, answerCallbackQuery, getChatMember)
- надежность доставки обеспечивается ретраями на стороне backend (и ограничениями Telegram)

Правило: ACK на webhook не является "успешным выполнением бизнес-операции", это только подтверждение приема события.

4.3 Контракт ACK и поведение Telegram при доставке

4.3.1 ACK (для webhook)

- Успешный прием Update: HTTP 200 OK
- Тело ответа: пустое

Норма:

- ACK отдаем максимально быстро
- тяжелую бизнес-логику не выполняем "до ACK", чтобы не держать соединение

4.3.2 Повторы доставки

Telegram может повторить доставку Update, если:

- webhook недоступен
- backend не ответил вовремя
- ответ неуспешен (не 2xx)

Вывод:

- backend обязан быть идемпотентным по входящим событиям

4.4 Идемпотентность, дедупликация и защита от повторов

4.4.1 Ключ идемпотентности

Ключ: `update_id` из входящего Update.

Норма: повторная обработка одного и того же `update_id` не должна приводить к повторному выполнению бизнес-действий:

- двойное добавление в избранное
- дубли сообщений/фото
- повторные переходы, меняющие состояние

4.4.2 Минимальная реализация дедупликации (MVP)

Допустимые варианты:

- in-memory cache (если один инстанс сервиса)
- Redis / внешний cache (если несколько инстансов)

Рекомендованная схема:

- при приеме Update проверяем `update_id` в cache
- если уже был - вернуть 200 OK и завершить без действий
- если новый - пометить как "in-progress/processed" с TTL и выполнить обработку

TTL:

- выбирается с запасом (минимум десятки минут) для перекрытия повторных доставок

4.4.3 Идемпотентность доменных операций

Помимо дедуп по update_id, доменная логика должна быть устойчивой:

- создание пользователя: уникальность по `telegram_id`
- избранное: добавление/удаление идемпотентны (повтор не создает дубль)
- навигация/открытие карточек: повтор не меняет состояние в БД (кроме счетчиков/логов, если они есть)

4.5 Единый подход к ошибкам (классификация)

Ошибки разделяем по каналу и источнику:

4.5.1 Inbound (webhook)

Типовые случаи:

- невалидный JSON
- структура Update не распознана
- неподдерживаемый тип события (не message и не callback_query)
- callback_data не соответствует формату

Поведение:

- ACK 200
- логируем причину
- не выполняем исходящие вызовы Bot API и операции с БД

4.5.2 Outbound (Bot API)

Типовые случаи:

- 429 Too Many Requests
- 5xx (ошибка Telegram)
- timeout/network errors
- 4xx (ошибка запроса или прав)
- ошибки медиа при sendPhoto (недоступный URL, неверный формат/слишком большой файл и т.п.)

Поведение:

- 429, 5xx, timeouts: ретраи по политике 4.6
- 4xx: как правило без ретраев, требуется исправление запроса/логики
- sendPhoto media errors (как правило 4xx): без ретраев, применяем fallback на sendMessage (см. 4.10)

4.5.3 Внутренние ошибки backend

Типовые случаи:

- ошибка БД (read/write)
- ошибка бизнес-логики/маппинга
- ошибка конфигурации (token/secret отсутствует)
- непредвиденное исключение

Поведение:

- логируем
- для webhook сохраняем приоритет ACK 200
- исходящие вызовы Bot API - по ситуации (в MVP не отправляем пользователю "ошибочные" сообщения)

4.6 Retry policy для исходящих вызовов Bot API

4.6.1 Общие нормы

Ретраим только временные ошибки:

- 429
- 5xx
- network timeout

Не ретраим:

- 400 (bad request)
- 401 (unauthorized)
- 403 (forbidden)
- 404 (not found)
- иные "логические" ошибки запроса
- ошибки медиа, связанные с URL/валидностью контента (см. 4.10)

4.6.2 Параметры ретраев (MVP)

- max_attempts: 3 (1 основная + 2 ретрая)
- backoff: экспоненциальный (например 1s, 2s, 4s)
- jitter: небольшой

Для 429:

- если в ответе есть параметр "retry_after", используем его как минимальную задержку перед повтором

4.6.3 Приоритеты отправки

Рекомендуемый порядок по критичности UX:

1. `answerCallbackQuery` (снять ожидание на кнопке)
2. `sendPhoto` / `sendMessage` (показать экран/результат)
3. `getChatMember` (проверка подписки - не критична по таймингу UI)

Правило:

- `answerCallbackQuery` выполняем всегда максимально быстро
- если `sendPhoto` не удался по причине медиа (не временная ошибка) - выполняем fallback на `sendMessage`

4.6.4 Поведение при исчерпании ретраев

Если после max_attempts запрос не удался:

- фиксируем ошибку в логах/метриках
- не спамим пользователя повторными сообщениями
- допускается внутренняя повторная отправка отдельным механизмом (очередь/задача), если он существует в архитектуре

4.7 Каталог ошибок (внутренние коды)

Формат внутренней ошибки (для логов/диагностики):

- `error_code` (строка)
- `error_class` (строка)
- `message` (кратко)
- `details` (опционально)
- `[служебное] traceId` (корреляция)

Минимальный набор error_class:

- `INBOUND_VALIDATION`
- `OUTBOUND_RATE_LIMIT`
- `OUTBOUND_TRANSPORT`
- `OUTBOUND_BAD_REQUEST`
- `OUTBOUND_MEDIA`
- `INTERNAL_DB`
- `INTERNAL_CONFIG`
- `INTERNAL_UNEXPECTED`

Минимальный набор error_code:

- `TG_INBOUND_INVALID_JSON`
- `TG_INBOUND_UNSUPPORTED_UPDATE`
- `TG_INBOUND_INVALID_CALLBACK_DATA`
- `TG_OUTBOUND_429_RATE_LIMIT`
- `TG_OUTBOUND_5XX`
- `TG_OUTBOUND_TIMEOUT`
- `TG_OUTBOUND_4XX_BAD_REQUEST`
- `TG_OUTBOUND_MEDIA_INVALID_URL`
- `TG_OUTBOUND_MEDIA_FETCH_FAILED`
- `TG_OUTBOUND_MEDIA_TOO_LARGE_OR_UNSUPPORTED`
- `TG_OUTBOUND_MEDIA_FALLBACK_TO_TEXT`
- `INTERNAL_DB_ERROR`
- `INTERNAL_CONFIG_ERROR`
- `INTERNAL_UNEXPECTED_ERROR`

4.8 Логирование, метрики и служебные атрибуты

4.8.1 Что логируем (минимум)

- `update_id`
- `event_type` (message/callback_query)
- `telegram_user_id` (from.id)
- `chat_id`
- для callback: parsed `{action, entity_type, entity_id, extra}`
- выбранный outbound метод: `sendMessage/sendPhoto/...`
- для sendPhoto: `photo_url` (без токенов и приватных query-параметров), результат загрузки
- результат обработки (ok/error)
- `error_code` / `error_class` (если ошибка)
- попытки outbound (attempt, delay, status)

4.8.2 Что не логируем

- bot token
- secret_token webhook
- персональные данные и чувствительный контент пользователя (в частности: тексты обратной связи)

4.8.3 [служебное] traceId

Назначение:

- связать логи разных компонент/шагов в рамках одного Update

Правило:

- traceId генерируется backend на входе обработки Update
- прокидывается во все логи и внутренние вызовы
- не требует хранения в доменной БД (не поле модели)

4.9 Нормы для специфичных процессов

4.9.1 P1 Проверка подписки

- Подписка проверяется вызовом `getChatMember` при следующем действии пользователя.
- Ошибка Bot API при проверке подписки: применяем retry policy как для outbound.

4.9.2 P9 Обратная связь

- В MVP текст обращения в БД не сохраняем.
- Допускается логирование факта обращения (event_type/telegram_user_id/traceId) без сохранения текста.

4.10 Правило fallback для sendPhoto (MVP)

Если по бизнес-сценарию нужно показать фото (карточка, избранное, промокод), но `sendPhoto` не может быть выполнен:

1. Если ошибка временная (429, 5xx, timeout):

- ретраим по общей политике 4.6
- при исчерпании ретраев: отправляем `sendMessage` (текст без фото) и фиксируем `TG_OUTBOUND_MEDIA_FALLBACK_TO_TEXT`

2. Если ошибка не временная (типично 4xx, связанные с URL/контентом):

- без ретраев
- сразу отправляем `sendMessage` (текст без фото)
- фиксируем один из кодов:
  - `TG_OUTBOUND_MEDIA_INVALID_URL`
  - `TG_OUTBOUND_MEDIA_FETCH_FAILED`
  - `TG_OUTBOUND_MEDIA_TOO_LARGE_OR_UNSUPPORTED`

3. Если URL фото отсутствует в данных:

- `sendPhoto` не вызываем
- сразу `sendMessage` (текст без фото)
- фиксируем `TG_OUTBOUND_MEDIA_FALLBACK_TO_TEXT` (details: "photo_url_missing")