CORE.TELEGRAM.ORG · MAPA DE APIS

APIs do Telegram: o que uma URL da plataforma devolve

Existem duas plataformas de acesso (Bot API HTTPS e Client API MTProto) e várias superfícies derivadas. A resposta ao "consultar uma URL" muda por completo conforme *como* se consulta — HTTP público sem login versus sessão autenticada.

◇ 9 superfícies mapeadas ◇ 4 vias de consulta direta ◇ formatos: JSON · TL binário · HTML

01 A pergunta em duas respostas

Toda URL `t.me/…` ou `tg://…` é um envelope em torno de um token (`domain`, `phone`, `hash`, `path`). O que você recebe de volta depende do regime de consulta.

SEM LOGIN · HTTP PURO

Buscar a URL pela web

`GET https://t.me/… → HTML`

Um `GET` simples devolve **markup HTML**, nunca JSON, e revela só o que é público. Duas rotas:

- **Cartão** `t.me/<user>`: meta tags Open Graph (`og:title/description/image`) + bloco `tgme_page_*` — nome, selo verificado, bio.
- **Contagem classifica o tipo**: canal mostra "N subscribers"; grupo "N members, N online"; usuário comum mostra só "Send Message", sem número.
- **Feed** `t.me/s/<canal>`: posts reais de canais públicos — texto, `datetime`, views, mídia (CDN), forwards. Pagina com `?before=/?after=`.
- **Privado/inexistente** → página genérica, sem dados.

COM SESSÃO · CLIENT API

Resolver a URL via MTProto

`contacts.resolveUsername → objetos TL`

Com `api_id/api_hash` + login, o cliente faz o parse local da URL e invoca o método TL. Retorno **muito mais rico**:

- **Natureza do alvo** pelo tipo do `Peer`: `peerUser` (usuário/bot), `peerChannel` (canal/supergrupo), `peerChat` (grupo).
- `id` + `access_hash` + flags `verified/scam/fake`, título, username.
- **Convite privado** `checkChatInvite(hash)` expõe `title`, `about`, `participants_count` e amostra de participantes **sem entrar no grupo**.
- 2º estágio: `getFullChannel`/`getFullUser` → bio, contagens, chat vinculado.

Meio-termo — Bot API. Resolve só entidades **públicas por @username**: `getChat("@x")` → `ChatFullInfo` (id, type, title, description, invite_link). Não resolve usuário comum por username, nem convites `joinchat/+hash`, nem links de mensagem. Não expõe `resolveUsername` nem `checkChatInvite`.

02 As quatro vias de consulta, lado a lado

Como se chama, o que autentica e em que formato responde.

JSON envelope ok/result · TL BINÁRIO objeto serializado MTProto · JSON ASSÍNC. @type via td_receive · HTML scraping de DOM

SUPERFÍCIE	COMO SE CONSULTA	AUTENTICAÇÃO	RESPOSTA
Bot API `api.telegram.org`	HTTP GET/POST `/bot<token>/METHOD`. Updates via `getUpdates` (polling) **ou** `setWebhook`.	**Token de bot** do @BotFather. Sem api_id/api_hash.	JSON `{"ok":true,"result":…}`; erro traz `error_code` + `retry_after`.
Client API `MTProto`	RPC **binária**, não REST. Transporte (TCP/WS/HTTP) até um DC; método TL cifrado em AES-256.	**api_id+api_hash** (my.telegram.org) **+** login de conta (telefone/2FA) ou bot.	TL BINÁRIO objeto boxed, constructor id de 32 bits. JSON só na doc.
TDLib	Funções C (`td_send` / `td_receive`) com **strings JSON**; fala MTProto por baixo.	Igual à MTProto: api_id/api_hash + login guiado por `updateAuthorizationState`.	JSON ASSÍNC. objetos com `@type` , `@extra` , `@client_id` .
Preview web `t.me`	GET HTTP simples, sem cabeçalho. `t.me/<user>` , `t.me/s/<canal>` , `?embed=1` .	**Nenhuma** — sem token, sem sessão, sem número.	HTML Open Graph + classes `tgme_*` . Sem JSON oficial.

03 Catálogo: link → método TL → objeto retornado

Da página `core.telegram.org/api/links`. O cliente faz o parse do link e chama o método correspondente. `user-only` = só conta de usuário; `público` = também via HTML.

PADRÃO DE URL	MÉTODO	OBJETO RETORNADO	ACESSO
`t.me/<username>` `tg://resolve?domain=`	`contacts.resolveUsername`	`contacts.resolvedPeer{peer, chats[], users[]}` — tipo do peer classifica usuário/canal/grupo; id+access_hash.	público
`t.me/+<phone>`	`contacts.resolvePhone`	`contacts.ResolvedPeer`	user-only
`t.me/+<hash>` `t.me/joinchat/<hash>`	`messages.checkChatInvite`	`chatInvite{title, about, photo, participants_count, participants[]}` — metadados de grupo **privado sem entrar**. Ou `chatInviteAlready/Peek` .	user-only
`t.me/<canal>/<id>` `t.me/c/<int>/<id>`	resolve domain/channel + `messages.getHistory`	`messages.messagesSlice{count, messages[], chats[], users[]}`	público*
`tg://…` desconhecido	`help.getDeepLinkInfo(path)`	`help.deepLinkInfo{update_app, message, entities}` ou `…Empty` .	sessão
`t.me/addstickers/<slug>`	`messages.getStickerSet`	`messages.StickerSet`	user-only
`t.me/<bot>?start=`	`messages.startBot`	`Updates`	user-only
enriquecimento pós-resolução	`channels.getFullChannel` · `users.getFullUser`	`channelFull` (about, participants/admins_count, linked_chat_id, slowmode) · `userFull` (bio, common_chats, blocked).	público

04 Superfícies adjacentes

Identidade e recursos de app — não resolvem URLs, mas fazem parte do mapa.

Login Widget

AUTENTICAÇÃO · OAUTH-LIKE

Site recebe **id, first_name, username, photo_url, auth_date, hash**. Valida com `HMAC-SHA256` sobre `SHA256(bot_token)`. Doc atual migrou para OpenID Connect (JWT).

Telegram Passport

IDENTIDADE · E2E

Chega como `Update.passport_data`: docs cifrados (AES256-CBC) + `secret` cifrado com a chave pública RSA do serviço. Só o serviço decifra.

Mini Apps / Web Apps

WEB DENTRO DO APP

`window.Telegram.WebApp.initData` — query-string assinada (**user, query_id, auth_date, hash**). Chave = `HMAC_SHA256(bot_token,"WebAppData")`.

Payments

COBRANÇA · PROVIDER

Objetos JSON do Bot API (**Invoice, PreCheckoutQuery, SuccessfulPayment**). Nem Telegram nem o bot veem dados de cartão — o provedor (ex.: Stripe) processa.

Instant View

PÁGINA → NATIVA

Templates por domínio transformam uma URL externa em `Article/RichText`. Link `t.me/iv?url=…&rhash=…`. Sem API JSON pública documentada.

Deep links tg://

ESQUEMA INTERNO

`tg://resolve`, `tg://join`, `tg://privatepost`… **Não** resolvíveis por HTTP — só o app os abre. Proxy/share não têm método de API.

05 O regime que condiciona toda consulta

Credenciais, limites de flood e Termos — valem para qualquer resolução em escala.

- **Bot API:** ~1 msg/s por chat, 20 msg/min por grupo, ~30 msg/s de broadcast. Excesso → HTTP `429` com `parameters.retry_after`.
- **Client API:** erro `420 FLOOD_WAIT_X` / `FLOOD_PREMIUM_WAIT_X` / `SLOWMODE_WAIT_X` — aguardar X segundos. Redirecionamento de DC via `303`.
- **Privacidade:** `messages.checkChatInvite` vaza metadados de grupos **privados** a qualquer possuidor do link — é `user-only` (bots não podem).
- **Termos (api/terms):** proíbem agir "on behalf of the user without consent" e agregar dados "to train, fine-tune… artificial intelligence". Flooding/inflar contadores → banimento permanente.
- **Preview web:** só conteúdo público; classes `tgme_*` podem mudar sem aviso; IPs que raspam em volume podem ser bloqueados.

Fonte: documentação oficial `core.telegram.org` · `t.me` (verificado ao vivo) schema TL evolui por "layers" — TL-lines são retrato de um layer