Guia da API i-go

A API do i-go busca endereços e lugares por nome, calcula rotas e matrizes de tempo e distância de carro, desenha o mapa com a identidade do i-go e planeja turnos com corridas agrupadas, no Brasil inteiro. Os dados vêm do OpenStreetMap, e os motores rodam em servidores próprios.

Este guia explica o que a especificação não cobre: cobertura, limites, erros, receitas prontas e como comparar os resultados com o Google. A lista completa de campos de cada rota está na referência da API.

Endereço da API	`https://igo-api.dev.brigtec.com.br`
Referência interativa	/docs/referencia
Especificação OpenAPI 3.1	/docs/openapi.yaml · /docs/openapi.json
Coleção do Postman	/docs/i-go.postman_collection.json
Este guia em PDF	/docs/i-go-api.pdf
Página de exemplo (mapa, busca, rota e modo viagem)	/docs/exemplo

> **Ambiente de testes.** Este endereço é o ambiente de desenvolvimento: não há acordo de disponibilidade, e o dado é atualizado sem aviso (a versão em uso vem em toda resposta, em `meta.data_version`). As rotas que ainda estão sob `/v1/experimental/` (estações por nome e agrupamento) podem mudar de formato ou sair.

Primeiros passos

Você precisa de uma credencial (`igo_…`), entregue por quem administra o i-go. Ela vai no cabeçalho `Authorization` de toda chamada.

1. Confira a credencial e a área atendida:

```
curl -s https://igo-api.dev.brigtec.com.br/v1/coverage \
  -H "Authorization: Bearer $IGO_CHAVE"
```

1. Busque um lugar por nome:

```
curl -s "https://igo-api.dev.brigtec.com.br/v1/search?
q=mercado%20central&lat=-7.12&lon=-34.88&limit=3" \
  -H "Authorization: Bearer $IGO_CHAVE"
```

1. Calcule uma rota entre dois pontos:

```
curl -s https://igo-api.dev.brigtec.com.br/v1/route \
  -H "Authorization: Bearer $IGO_CHAVE" \
  -H "Content-Type: application/json" \
  -d '{"points":[{"lat":-7.1195,"lon":-34.8450},
{"lat":-7.1469,"lon":-34.7975}],"geometry":"none"}'
```

Resposta:

```
{
  "data": { "criterion": "fastest", "duration_s": 1391.4,
"distance_m": 9335.3 },
  "meta": {
    "coverage": "br",
    "data_version": "2026-11-04.3",
    "overlay_cycle": "none",
    "degraded": false,
    "attribution": "© OpenStreetMap contributors",
    "request_id": "392ab16a603f282034a604b4"
  }
}
```

Na referência, o botão **Authorize** recebe a credencial e cada rota ganha um **Try it out** que chama a API de verdade. A credencial fica só na aba aberta; ao fechar, é esquecida.

Autenticação

- Envie `Authorization: Bearer igo_…` em toda requisição. A credencial nunca vai na URL: URLs ficam em histórico de navegador e em log de servidor.
- Cada credencial tem **escopos**, e cada rota exige um deles:

Escopo	Rotas
coverage	GET /v1/coverage
search	GET /v1/search, GET /v1/reverse
route	POST /v1/route
matrix	POST /v1/matrix
pooling	POST /v1/experimental/pooling/plan
nearest	POST /v1/nearest
stations	GET /v1/experimental/stations
tiles	GET /v1/tiles/{z}/{x}/{y}, GET /v1/styles/{nome}, GET /v1/fonts/{fonte}/{faixa}, GET /v1/sprites/{arquivo}, GET /v1/highlights

Pedir uma rota fora dos escopos da credencial dá `403 forbidden_scope`, com o escopo que faltou no `detail`. Credencial perdida ou exposta: avise quem a entregou, para que seja revogada e outra emitida.

No navegador. A API responde a páginas de qualquer origem (CORS), com a credencial no cabeçalho, nunca em cookie. Mas o que roda no navegador fica visível para quem abre a página: credencial usada ali deve ter só os escopos de que a tela precisa (em geral `tiles` e `search`) e uma quota baixa. Rota, matriz e agrupamento ficam melhor no seu servidor, com outra credencial.

Convenções

- **Coordenadas** entram e saem como objeto `{"lat": -7.1195, "lon": -34.8450}`. Só as geometrias (a linha da rota) seguem o GeoJSON, com cada ponto em `[lon, lat]`. Trocar a ordem é o erro mais comum: o ponto cai no oceano e a resposta é `422 out_of_coverage`.
- **Unidades:** distância em metros (`_m`), tempo em segundos (`_s`), horários em RFC 3339 com fuso (`2026-10-28T07:00:00-03:00`).
- **Envelope:** toda resposta de sucesso é `{"data": …, "meta": …}`. Em `meta`:
  - `data_version`: a versão do dado, com a data do retrato do OpenStreetMap e o número da geração;

    - `request_id`: identifica a chamada;
    - `degraded: true`: a resposta é uma aproximação, e o motivo vem em `degraded_reason`;
    - `attribution`: o texto que deve aparecer junto do mapa ou da rota.
- `X-Request-Id`**:** se você enviar esse cabeçalho (até 64 caracteres `A-Z a-z 0-9 . _ : -`), o i-go usa o seu valor; senão, gera um. Ao relatar um problema, informe esse identificador.
- **Campos novos:** dentro da v1 só entram mudanças aditivas. Ignore campos e valores de enum que você não conhece, em vez de falhar.

Cobertura e dados

- **Área atendida:** o Brasil inteiro (`br`). O polígono exato vem em `GET /v1/coverage` (`data.area`). Ponto fora dele dá `422 out_of_coverage`, e `field` diz qual ponto.
- **Origem dos dados:** OpenStreetMap, num retrato datado (`data_version`), regenerado periodicamente. Rua aberta depois do retrato não existe para o i-go até a geração seguinte.
- **Perfil:** só carro (`"profile": "car"`, o padrão).
- **Trânsito:** não há. O tempo é de fluxo livre, calculado pela velocidade típica de cada tipo de via, sem trânsito ao vivo nem histórico por horário.
- **Atribuição (ODbL):** quem mostrar mapa, rota ou endereço vindo do i-go deve exibir "© OpenStreetMap contributors".

Limites e quota

Limite	Valor
Requisições por credencial	20 por segundo, com rajada de 40
Pontos em `/v1/route`	2 a 25
Matriz (`/v1/matrix`)	até 25 origens × 25 destinos
Busca (`q`)	3 a 120 caracteres; até 10 resultados (`limit`, padrão 5)
Agrupamento	até 40 passageiros, 20 veículos, capacidade 8 por veículo
Corpo da requisição	256 KiB
Tempo máximo de uma chamada	15 segundos

Os limites em vigor também vêm em `GET /v1/coverage` (`data.limits`). Ao passar da quota, a resposta é `429 rate_limited` com o cabeçalho `Retry-After` (em segundos): espere e repita. Para comparações em lote, mantenha um ritmo de até 10 chamadas por segundo por credencial. Assim sobra folga para repetições.

Erros

Todo erro segue a RFC 9457 (`Content-Type: application/problem+json`):

```
{
  "type": "https://igo-api.dev.brigtec.com.br/erros/
out_of_coverage",
  "title": "Ponto fora da área atendida",
  "status": 422,
  "code": "out_of_coverage",
  "detail": "Fora de br.",
  "field": "points[1]",
  "request_id": "1f0c2e9a7b3d4c5e6f708192"
}
```

Decida pelo campo `code`: `title` e `detail` são texto para pessoas e podem mudar. Quando o problema é um campo, `field` diz qual (`points[1]`, `q`...).

code	HTTP	O que significa	O que fazer
`invalid_input`	400	Corpo ou parâmetro inválido: JSON malformado, campo obrigatório ausente, valor fora da faixa	Corrija o campo indicado em `field`; não repita igual
`unauthenticated`	401	Credencial ausente, inválida ou revogada	Confira o cabeçalho `Authorization: Bearer igo_…`
`forbidden_scope`	403	A credencial não tem o escopo da rota	Peça o escopo a quem administra o i-go
`not_found`	404	Rota inexistente, ou nada encontrado perto do ponto (reverso)	Confira o caminho; no reverso, o ponto não tem endereço por perto
`limit_exceeded`	413	Pontos, matriz ou corpo acima do limite	Divida a chamada em partes menores
`out_of_coverage`	422	Ponto fora da área atendida	Confira a ordem lat/lon e a área em `/v1/coverage`
`no_route`	422	Não há caminho de carro entre os pontos (ilha, área sem rua)	Aproxime o ponto de uma rua
`rate_limited`	429	Quota da credencial excedida	Espere o `Retry-After` e repita
`unavailable`	503	Motor de rotas ou de busca temporariamente fora	Repita depois de alguns segundos, com espera crescente
`internal`	500	Erro inesperado do i-go	Repita uma vez; persistindo, relate com o `request_id`

Só vale repetir a mesma chamada em `rate_limited`, `unavailable` e `internal`. Nos outros casos, a resposta seria a mesma.

Receitas

Busca de lugares e endereços por nome

`GET /v1/search?q=…` procura ruas, bairros, cidades e lugares (comércio, praças, órgãos públicos) dentro da área atendida.

- Completa a partir da 3ª letra e tolera acento e erro de digitação: "interm" acha "Intermares". Serve para autocompletar a cada tecla.
- Com `lat` e `lon`, o que está perto pesa mais na ordem, e cada resultado traz `straight_line_m`, a distância em linha reta até o ponto de referência.
- Quando o texto é exatamente o nome de uma cidade, a cidade vem antes dos homônimos menores (povoado, bairro, lugar): "Fortaleza" traz a capital. Só um homônimo exato a até 30 km do ponto de referência continua na frente de uma cidade mais distante: o bairro de mesmo nome na cidade de quem busca.
- `kind` é a categoria em português (Bairro, Rua, Restaurante, Shopping…).

- Pode mandar o endereço como a pessoa digitou: CEP, complemento ("Bloco 1, Apto 302") e ponto de referência ("Próximo ao mercado") são ignorados. Com número da casa, onde o mapa tem esse número a casa vem antes da rua; onde não tem (a maior parte do mapa, por enquanto), vem só a rua, com o ponto no meio dela.
- `match` diz o que o resultado é e, com isso, quão preciso é o ponto: `house_number` (o ponto é o da casa), `street` (fica na rua, não na casa), `place` (comércio, praça, órgão público...), `district` (bairro), `city`, `region` (estado) ou `postcode` (área do CEP). Quem pediu "Rua Grande, 53" e recebeu `street` sabe que o número não está no mapa.
- `address` é o endereço numa linha, para exibir. `address_parts` traz o mesmo endereço em campos: `street`, `house_number`, `neighborhood`, `city`, `state`, `state_code` (UF), `postcode` (CEP) e `country_code`. Campo ausente quer dizer que o mapa não tem o dado para aquele lugar; nada é adivinhado. Quando o resultado é a própria rua, bairro ou cidade (`match`), o nome dele também vem no campo correspondente. O CEP é o que está no mapa: em rua longa pode ser o de um trecho, não o da casa.
- Nada encontrado não é erro: a resposta é `200` com `results` vazio.

```
curl -s "https://igo-api.dev.brigtec.com.br/v1/search?
q=mercado%20central&lat=-7.12&lon=-34.88&limit=2" \
  -H "Authorization: Bearer $IGO_CHAVE"
```

```
{"data":{"results":[
  {"name":"Mercado Central","address":"Rua Francisca Moura,
Centro, João Pessoa, Paraíba",
   "address_parts":{"street":"Rua Francisca
Moura","neighborhood":"Centro","city":"João Pessoa",

"state":"Paraíba","state_code":"PB","country_code":"BR"},
   "kind":"Lugar","match":"place","location":
{"lat":-7.1237648,"lon":-34.879850087839245},"straight_line_m":
419},
  {"name":"Mercado Central","address":"Rua Doutor Francisco
Montenegro, Alagoa Grande, Alagoa Grande, Paraíba",
   "address_parts":{"street":"Rua Doutor Francisco
Montenegro","neighborhood":"Alagoa Grande","city":"Alagoa Grande",

"state":"Paraíba","state_code":"PB","country_code":"BR"},
   "kind":"Lugar","match":"place","location":
{"lat":-7.0352951,"lon":-35.6336183},"straight_line_m":83692}]},
 "meta":{…}}
```


Endereço de um ponto (reverso)

`GET /v1/reverse?lat=…&lon=…` devolve o lugar ou o endereço mais próximo do ponto, com os mesmos campos da busca (`address_parts` inclusive). Sem nada por perto, a resposta é `404 not_found`.

```
curl -s "https://igo-api.dev.brigtec.com.br/v1/reverse?
lat=-7.1153&lon=-34.8641" \
  -H "Authorization: Bearer $IGO_CHAVE"
```


```
{"data":{"place":{"name":"La Casserolle","address":"Praça Assis
Chateaubriand, 48, Jardim 13 de Maio, João Pessoa, Paraíba",
  "address_parts":{"street":"Praça Assis
Chateaubriand","house_number":"48","neighborhood":"Jardim 13 de
Maio",
                   "city":"João
Pessoa","state":"Paraíba","state_code":"PB","country_code":"BR"},
  "kind":"Restaurante","match":"place","location":
{"lat":-7.11549495,"lon":-34.863966513730176}}},"meta":{…}}
```

Rota

`POST /v1/route` calcula a rota de carro pelos pontos `points`, na ordem dada (origem, paradas, destino; até 25).

- **Critério:** `criterion` diz como o caminho foi escolhido. Hoje é sempre `fastest`: o menor tempo previsto pelo modelo de velocidades do i-go. Esse tempo não considera trânsito em tempo real (`meta.overlay_cycle` diz se houve ajuste ao vivo) nem o horário do dia. Não há rota alternativa nem "mais curta" por enquanto. Quando houver, entram como opção, sem mudar o que já existe. Aceite valores novos em `criterion`.
- `"geometry": "none"` devolve só `duration_s` e `distance_m`. É o modo mais leve para comparar números.
- Sem `geometry` (padrão `geojson`), vem a linha da rota em `data.geometry` (GeoJSON LineString, pontos em `[lon, lat]`).
- `"segment_durations": true` acrescenta `segment_durations_s`, o tempo de cada trecho da linha. A soma é `duration_s`. Com esses tempos, dá para calcular o tempo que falta a partir de qualquer ponto da rota.

```
curl -s https://igo-api.dev.brigtec.com.br/v1/route \
  -H "Authorization: Bearer $IGO_CHAVE" -H "Content-Type:
application/json" \
  -d '{"points":[{"lat":-7.1195,"lon":-34.8450},
{"lat":-7.1469,"lon":-34.7975}],"segment_durations":true}'
```

Matriz de tempos e distâncias

`POST /v1/matrix` dá o tempo e a distância de cada origem (`sources`) para cada destino (`destinations`), até 25 × 25. A célula `[i][j]` vai da origem `i` ao destino `j`, e `null` quer dizer sem rota. Vem com o mesmo `criterion` da rota (`fastest`).

```
curl -s https://igo-api.dev.brigtec.com.br/v1/matrix \
  -H "Authorization: Bearer $IGO_CHAVE" -H "Content-Type:
application/json" \
  -d '{"sources":[{"lat":-7.1195,"lon":-34.8450}],
       "destinations":[{"lat":-7.1469,"lon":-34.7975},
{"lat":-7.0920,"lon":-34.8330}]}'
```

```
{"data":{"durations_s":[[1391.4,617.4]],"distances_m":
[[9335.3,4758.8]]},"meta":{…}}
```


Para mais de 25 pontos, divida em blocos de até 25 × 25 e junte as partes. Cada bloco é uma chamada e conta na quota.

Agrupamento de corridas (planejamento de turno)

`POST /v1/experimental/pooling/plan` distribui os passageiros de um turno entre os veículos, com a ordem e o horário de cada parada.

- **Sentido:**
  - `inbound`: todos vão para o ponto comum (`common`, ex.: a empresa) e chegam até `arrive_by` menos `margin_min` (padrão 10 minutos);
  - `outbound`: todos saem do ponto comum em `depart_at`.
- **Frota:** `fleet.vehicles` veículos de `fleet.capacity` lugares. Com `fleet.base`, o táxi sai da base, busca e leva (ida) ou leva e volta à base (volta). O deslocamento vazio entra no horário e, com `fare.charge_from: "base"`, no custo.
- **Regras para quem está a bordo:**
  - o tempo a bordo vai até (1 + `max_detour_pct`/100) × o tempo da viagem direta (padrão 30%);
  - na ida, todos chegam no horário, e ninguém é buscado antes do seu `ready_at`.
- **Custo:** `fare.base` + `fare.per_km` × km + `fare.per_min` × minutos, por viagem. `baseline` é o custo de cada um no seu veículo, e `savings_pct` é a economia do plano sobre ele.
- **Quem não cabe** vem em `unserved`, com o motivo (`no_vehicle`, `no_time`, `no_route`). O plano nunca atrasa ninguém em silêncio.
- **Nenhum dado de pessoa:** `requests[].id` é um identificador seu (não use nome, CPF nem telefone), e `location`, a coordenada de embarque.

```
curl -s https://igo-api.dev.brigtec.com.br/v1/experimental/
pooling/plan \
  -H "Authorization: Bearer $IGO_CHAVE" -H "Content-Type:
application/json" \
  -d '{"direction":"inbound","common":
{"lat":-7.1760,"lon":-34.9040},
       "arrive_by":"2026-10-28T07:00:00-03:00",
       "fleet":{"vehicles":2,"capacity":4,"base":
{"lat":-7.1450,"lon":-34.8800}},
       "fare":{"base":5,"per_km":2.5,"per_min":
0.4,"charge_from":"base"},
       "requests":[{"id":"p1","location":
{"lat":-7.1195,"lon":-34.8450}},
                   {"id":"p2","location":
{"lat":-7.1300,"lon":-34.8600}},
                   {"id":"p3","location":
{"lat":-7.0920,"lon":-34.8330}}]}'
```

Resultado resumido: duas viagens (p1 e p2 no veículo 1, p3 no veículo 2), custo total de 181,63 contra 236,38 com cada um no seu carro (`savings_pct` 23,2). A resposta completa está na referência. Um turno de 40 passageiros leva cerca de 1,3 segundo.

Mapa com a identidade do i-go

O estilo `igo` (`GET /v1/styles/igo`) é o mapa do i-go pronto para o MapLibre GL JS (ou qualquer leitor de estilo Mapbox GL versão 8): tiles, rótulos, ícones e os destaques da cidade. As URLs dentro do estilo apontam para esta API e exigem a mesma credencial, então cada pedido do mapa leva o cabeçalho `Authorization` (`transformRequest`):

```
const API = "https://igo-api.dev.brigtec.com.br";
const mapa = new maplibregl.Map({
  container: "mapa",
  style: API + "/v1/styles/igo",
  center: [-34.88, -7.12], zoom: 12,
  transformRequest: (url) => url.startsWith(API + "/v1/")
    ? { url, headers: { Authorization: "Bearer " + CHAVE } } :
{ url },
});
```

Para desenhar a rota por cima, peça `POST /v1/route` (a geometria vem em GeoJSON) e acrescente uma camada de linha:

```
mapa.on("load", async () => {
  const r = await fetch(API + "/v1/route", { method: "POST",
    headers: { Authorization: "Bearer " + CHAVE, "Content-Type":
"application/json" },
    body: JSON.stringify({ points: [origem,
destino] }) }).then((x) => x.json());
  mapa.addSource("rota", { type: "geojson", data:
r.data.geometry });
  mapa.addLayer({ id: "rota", type: "line", source: "rota",
    layout: { "line-join": "round", "line-cap": "round" },
    paint: { "line-color": "#1c7a9c", "line-width": 6 } });
});
```

- A página de exemplo junta tudo: mapa, busca com autocompletar, toque no mapa para o reverso, rota e modo viagem. Ela usa só a credencial e as rotas públicas da v1; o código está em /docs/exemplo.js e pode servir de ponto de partida.
- **Atribuição:** o MapLibre mostra "© OpenMapTiles © OpenStreetMap contributors" no canto do mapa, a partir do estilo. Não esconda.
- **Destaques:** o mapa só traz pontos de interesse a partir do zoom 14. Para a pessoa se situar antes disso, `GET /v1/highlights` devolve as referências da cidade (universidade, hospital, shopping, estádio, parque grande…) em GeoJSON, cada uma com o zoom a partir do qual aparece (10 a 13). O estilo `igo` já as desenha; quem usa outro estilo pode acrescentá-las como fonte GeoJSON.
- `basic` é o estilo anterior, mantido enquanto houver cliente usando.

Modo viagem

O modo viagem (acompanhar a pessoa ao longo da rota, com o tempo que falta) é feito no aplicativo, com as rotas que já existem; não há rota própria para ele. A receita, que é a do aplicativo do i-go:

1. **Rota com tempo por trecho.** `POST /v1/route` com `"segment_durations": true`. Vêm a linha (`geometry`) e o tempo de cada trecho entre dois pontos dela (`segment_durations_s`).

2. **Onde a pessoa está na rota.** A cada posição do GPS, projete o ponto no segmento mais próximo da linha (não no vértice mais próximo: num trecho reto e longo, o vértice fica longe). Procure só numa janela à frente da última posição, porque a rota que vai e volta passa perto de si mesma.
3. **Quanto falta.** O tempo é o resto do trecho atual mais a soma dos trechos seguintes; a distância é o resto da linha.
4. **Saiu da rota.** Mais longe da linha que 60 m (ou 1,5 × a precisão do GPS) em três posições seguidas: peça uma rota nova a partir de onde a pessoa está.
5. **Chegou.** A 40 m do destino, encerre.
6. **Tela acesa:** peça a trava de tela (`navigator.wakeLock`) enquanto durar a viagem.

As funções dos passos 2 e 3, sem dependência nenhuma, estão em /docs/rota.js (`sobreRota` e `faltaS`), as mesmas que o aplicativo do i-go usa e testa. A página de exemplo mostra o modo viagem funcionando.

Não há instrução de manobra ("vire à direita em 200 m") nem voz: a rota traz a linha e os tempos. O tempo é o do modelo de velocidades, sem trânsito ao vivo. Cada recálculo é uma chamada de rota e conta na quota.

Estações de recarga

`POST /v1/nearest` (estações mais próximas por tempo de rota) e `GET /v1/experimental/stations` (estações por nome) estão descritas na referência. Elas exigem os escopos `nearest` e `stations`, que não vêm nas credenciais de teste de busca e roteamento.

Comparando com o Google

Cada rota do i-go tem uma equivalente no Google Maps Platform. Para que a comparação seja justa, as duas chamadas devem receber a mesma entrada e medir a mesma coisa.

O que se compara	i-go	Google
Busca por nome, com autocompletar	`GET /v1/search`	Places API (New): `places:autocomplete` ou `places:searchText`
Endereço por texto	`GET /v1/search`	Geocoding API (`geocode/json?address=`)
Endereço de um ponto	`GET /v1/reverse`	Geocoding API (`geocode/json?latlng=`)
Rota	`POST /v1/route`	Routes API `directions/v2:computeRoutes`
Matriz	`POST /v1/matrix`	Routes API `distanceMatrix/v2:computeRouteMatrix`
Agrupamento de turno	`POST /v1/experimental/pooling/plan`	Route Optimization API (`optimizeTours`), com outro modelo de entrada

Rota e matriz: mesma entrada, sem trânsito

- **Pontos:** compare por coordenada, não por endereço. Assim o teste mede só o roteamento, sem misturar a busca. Use os mesmos pontos, na mesma ordem.
- **Trânsito:** no Google, peça `"routingPreference": "TRAFFIC_UNAWARE"`. O i-go não usa trânsito, e com trânsito o Google mede outra coisa. Se quiser ver o efeito do trânsito, faça uma segunda chamada `TRAFFIC_AWARE` com `departureTime` e registre-a à parte.
- **Formato:** o Google devolve a duração como texto (`"1391s"`), a distância em `distanceMeters` e a linha em polyline codificada. A matriz do Google vem como uma lista de elementos com `originIndex` e `destinationIndex`, não como tabela.

Exemplo de rota no Google, equivalente à receita acima:

```
curl -s -X POST https://routes.googleapis.com/directions/
v2:computeRoutes \
  -H "Content-Type: application/json" \
  -H "X-Goog-Api-Key: $GOOGLE_CHAVE" \
  -H "X-Goog-FieldMask: routes.duration,routes.distanceMeters" \
  -d '{"origin":{"location":{"latLng":
{"latitude":-7.1195,"longitude":-34.8450}}},
       "destination":{"location":{"latLng":
{"latitude":-7.1469,"longitude":-34.7975}}},

"travelMode":"DRIVE","routingPreference":"TRAFFIC_UNAWARE"}'
```

O que registrar por par de pontos:

- distância e tempo do i-go e do Google;
- diferença percentual de cada um;

- `data_version` e `request_id` do i-go;
- data e hora da chamada.

Se houver o tempo real de uma viagem feita no trajeto, ele é a melhor referência para os dois.

Busca: uma lista de casos com a resposta certa

Monte uma lista de buscas reais, cada uma com o lugar que deveria aparecer, e rode a lista nos dois. Para cada busca, registre:

- se o lugar certo veio em 1º lugar e se veio entre os 5 primeiros;
- a distância entre a coordenada devolvida e a do lugar certo;
- as buscas em que um dos dois não achou nada.

Mande o mesmo ponto de referência nos dois: `lat`/`lon` no i-go, `locationBias` no Google. Inclua buscas digitadas pela metade ("interm", "av epit") e com erro de digitação, porque o autocompletar é onde mais se nota diferença.

O que já se sabe antes de começar

- **Comércio e pontos de interesse:** o i-go conhece o que está no OpenStreetMap. Em comércio recente e em números de porta, o Google tende a ter mais registros. A diferença existe e deve aparecer na comparação.
- **Tempo de rota em vias rápidas:** nos testes de campo de outubro de 2026, o tempo previsto pelo i-go ficou acima do tempo real em avenidas e rodovias. A calibração por tipo de via está em andamento.
- **Rotas idênticas, tempos diferentes:** é esperado. Os dois escolhem o mesmo caminho com frequência, e a diferença fica na velocidade atribuída a cada via.

Mudanças

- **05/11/2026.** O Brasil inteiro passou a ser atendido (`coverage: br`). Busca, reverso e mapa saíram de `/v1/experimental/` e entraram na v1, com a garantia de contrato da v1: `/v1/search`, `/v1/reverse`, `/v1/tiles/…`, `/v1/styles/…`, `/v1/fonts/…`, `/v1/sprites/…` e `/v1/highlights`. Os endereços antigos continuam respondendo igual, com os cabeçalhos `Deprecation` e `Link: <novo>; rel="successor-version"`; a data de remoção será avisada com antecedência. Busca e reverso

ganharam `address_parts`, o endereço em campos. A API passou a responder a páginas de qualquer origem (CORS).

Privacidade

- Não envie dados pessoais: nem nome, CPF, telefone ou e-mail no `id` do agrupamento, nem no texto da busca.
- **O que fica registrado:**
  - a API registra, por chamada, a rota, o status, a duração e a credencial; não grava as coordenadas nem os corpos das requisições;
  - o servidor de borda (proxy) mantém um log de acesso com a URL de cada chamada. Nas buscas e no reverso, a URL inclui o texto buscado e as coordenadas.

Suporte

Ao relatar um problema ou um resultado estranho, envie:

- o `request_id` (ou o `X-Request-Id` que você mandou);
- a hora da chamada;
- a entrada usada e o que esperava receber.

Com isso, a chamada é localizada no registro de uso sem que você precise mandar mais nada.