In Time AI Geolocation – API Documentation

Oct 27, 2026 ·

Introduction

The service returns geographic coordinates (lat/lng) for a Bulgarian address given as free text. It was developed by IN TIME Ltd. and IL WEB EOOD under the "Competitiveness and Innovation in Enterprises" 2021–2027 programme and runs in production in the IN TIME courier software.

This document is for developers of external systems who want to integrate the service. It covers the three API methods, the request and response formats, rate limits and integration examples.

The service combines several coordinate sources: its own ML model, Google Geocode, the Romin AI agent and cadastre data (KRS). Sources are called one after another in a configured order (fallback chain), and results are cached.

General conventions

All methods are called with HTTP POST and a JSON body, at `{baseURL}soapserver/json/{method}`.

Parameter	Value
Base URL (`baseURL`)	`https://<service domain>/` — provided by IN TIME when you contact us for access
HTTP method	`POST` for all methods
Request body	JSON, `Content-Type: application/json`
Encoding	UTF-8; addresses may be in Cyrillic or Latin script
Response format	JSON, `Content-Type: application/json`
HTTP caching	Disabled by the server (`Cache-Control: no-store, no-cache`)

Authentication

Every method that returns coordinates requires a username and password in the request body (`username` and `password` fields), on every request. Accounts are created by the service administrator, and each user has an assigned access level.

The server also sets `user` and `token` cookies in its responses. These cannot be used for authentication — clients should ignore them and always send `username` and `password`.

Endpoints

The API has three methods: an availability check, geocoding through the fallback chain, and geocoding through one chosen source.

Method	Path	Authentication	Purpose
`test`	`soapserver/json/test`	No	Check that the service is reachable
`coords`	`soapserver/json/coords`	Yes	Coordinates for an address via the fallback chain
`coords_by_api`	`soapserver/json/coords_by_api`	Yes	Coordinates for an address from one chosen source

test — availability check

Returns the string `"test"` if the service is running. Takes no parameters and no authentication.

Request:

```
POST {baseURL}soapserver/json/test
Content-Type: application/json

{}
```

Response (HTTP 200):

```
"test"
```

coords — coordinates via the fallback chain

The main method of the service. The address goes through the active sources in their configured order until one returns a result. The `api_used` field in the response shows which source answered.

Request fields:

Field	Type	Required	Description
`username`	string	Yes	Username
`password`	string	Yes	Password
`city`	string	Yes	City or village, free text
`postCode`	integer	No	Postal code (optional, improves accuracy), e.g. `1303`
`address`	string	Yes	Street, number, district, block etc., free text

Request:

```
{
    "username": "<username>",
    "password": "<password>",
    "city": "СПФОЯ",
    "postCode": 1303,
    "address": "Цар Симеон 97"
}
```

Response (HTTP 200):

```
{
    "success": true,
    "result": {
        "city": "СПФОЯ",
        "address": "Цар Симеон 97",
        "lat": 42.7019742,
        "lng": 23.3175689,
        "formatted_address": "София център, ул. „Цар Симеон“ 97, 1303 София,
България",
        "address_components": "
[{\"long_name\":\"97\",\"short_name\":\"97\",\"types\":[\"street_number\"]},
…]",
        "api_used": "google"
    }
}
```


In this example the city is misspelled ("СПФОЯ" instead of "София") and the service still resolves it to Sofia.

Response fields:

Field	Type	Description
`success`	boolean	`true` when coordinates were found
`result.city`	string	The city as submitted
`result.address`	string	The address as submitted
`result.lat`	number	Latitude (WGS 84)
`result.lng`	number	Longitude (WGS 84)
`result.formatted_address`	string	Normalized address, in Bulgarian
`result.address_components`	string	A JSON array serialized as a string, holding the address components (street number, street, neighbourhood, city, region, country, postal code); each element has `long_name`, `short_name` and `types`
`result.api_used`	string	Code of the source that returned the result (see Coordinate sources)

The `address_components` field must be decoded separately (e.g. `JSON.parse` / `json_decode`).

coords_by_api — coordinates from one chosen source

Returns coordinates from the requested source only, without the fallback chain. Each request queries exactly one source. Use it to compare sources or when a result from a specific source is needed.

Request fields: the same as `coords`, plus:

Field	Type	Required	Description
`api`	string	Yes	Source code: `own_ai`, `google` or `rominai`

Request:

```
{
    "username": "<username>",
    "password": "<password>",
    "city": "СПФОЯ",
    "postCode": 1303,
    "address": "Цар Симеон 97",
    "api": "own_ai"
}
```

Response (HTTP 200):

```
{
    "success": true,
    "results": [
        {
            "api": "own_ai",
            "success": true,
            "lat": 42.7019742,
            "lng": 23.3175689,
            "formatted_address": "София център, ул. „Цар Симеон“ 97, 1303
София, България"
        }
    ]
}
```

Response fields:

Field	Type	Description
`success`	boolean	`true` when the request was processed

Field	Type	Description
`results`	array	Always one element: the result of the requested source
`results[].api`	string	Source code
`results[].success`	boolean	Whether the source returned coordinates
`results[].lat`	number	Latitude (WGS 84)
`results[].lng`	number	Longitude (WGS 84)
`results[].formatted_address`	string	Normalized address, in Bulgarian

Coordinate sources and the fallback chain

Three sources can be requested through the API, each with a unique code. The code is returned in `api_used` (`coords`) and passed in `api` (`coords_by_api`).

Code	Source	Note
`own_ai`	Own AI model	ML model trained on IN TIME's historical delivery addresses
`google`	Google Geocode	External service
`rominai`	Romin AI agent	External paid service with a high cost per request

The `coords` method processes a request as follows:

1. Checks the cache. If the address was geocoded before, returns the stored result without calling any source.
2. Calls the first active source, in the order set in the "Address correction methods" admin screen.
3. If that source returns no result, the request moves on automatically to the next active source.
4. The first result found is cached and returned to the client, with the source code in `api_used`.

The administrator controls the order of the sources and which ones are active, without code changes. Clients should therefore not depend on a particular `api_used` value — it can change when the configuration changes.

The own model is tuned to return no result when it is unsure, rather than return inaccurate coordinates. Uncertain addresses then move on to the next source in the chain.

Rate limits

Requests are limited per minute, per hour and per day, both per account and per client IP address. The limit values are variable and may differ between accounts. When a limit is exceeded, the API returns an error naming the window, with `(ip_address)` added for IP limits (see Error handling).

Clients should spread requests evenly over time and handle a limit-exceeded response by waiting before retrying, not by retrying immediately.

Error handling

The API always returns HTTP status 200, including on errors. Clients must check the `success` field to tell success from failure. Use coordinates only when `success` is `true`; for `coords_by_api`, also check `results[].success`.

An error response has `success: false` and describes the error in the `errors` field. Example — wrong username or password:

```
{
    "success": false,
    "count_results": 0,
    "errors_count": 1,
    "errors": "login-failed"
}
```

Field	Type	Description
`success`	boolean	Always `false` on error
`count_results`	integer	Number of results returned; `0` on error
`errors_count`	integer	Number of errors
`errors`	string (may become array)	Error message (see the list below)

All error messages, as returned in `errors`:

Message	Meaning	Recommended action
`Empty username / password`	`username` or `password` is missing or empty	Send both fields in every request
`login-failed`	Wrong username or password	Check the credentials; do not retry automatically
`Access Denied: You do not have permission to access this method.`	The account's access level does not include this method	Ask the service administrator for access
`The fields 'city' and 'address' are required for geocoding.`	`city` or `address` is missing or empty	Fix the request
`The selected API ('<api>') does not exist or is disabled.`	`coords_by_api` was called with an unknown or disabled source; `<api>` is the value sent	Use `own_ai`, `google` or `rominai`, or another source if this one is disabled
`Too many requests per minute.`	Account limit per minute exceeded	Wait until the next minute, then retry
`Too many requests per hour.`	Account limit per hour exceeded	Wait until the next hour, then retry
`Too many requests per day.`	Account limit per day exceeded	Stop sending until the next day
`Too many requests per minute (ip_address).`	IP address limit per minute exceeded	Wait until the next minute, then retry
`Too many requests per hour (ip_address).`	IP address limit per hour exceeded	Wait until the next hour, then retry
`Too many requests per day (ip_address).`	IP address limit per day exceeded	Stop sending until the next day

Clients should match on these messages as fixed strings.

Currently every error response holds a single error, so `errors` is a string and `errors_count` is 1. A future version may return several errors as an array; clients should accept both a string and an array in `errors`.

Integration examples

The examples call `coords`. For `coords_by_api`, add the `api` field to the request body.

cURL

```
curl -X POST "https://<service domain>/soapserver/json/coords" \
  -H "Content-Type: application/json" \
  -d '{"username":"<username>","password":"
<password>","city":"София","postCode":1303,"address":"Цар Симеон 97"}'
```

PHP

```
<?php
$baseUrl = 'https://<service domain>/';

$payload = [
    'username' => getenv('INTIME_GEO_USER'),
    'password' => getenv('INTIME_GEO_PASS'),
    'city'     => 'София',
    'postCode' => 1303,
    'address'  => 'Цар Симеон 97',
];

$ch = curl_init($baseUrl . 'soapserver/json/coords');
curl_setopt_array($ch, [
    CURLOPT_POST           => true,
    CURLOPT_HTTPHEADER     => ['Content-Type: application/json'],
    CURLOPT_POSTFIELDS     => json_encode($payload, JSON_UNESCAPED_UNICODE),
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_TIMEOUT        => 30,
]);
$body = curl_exec($ch);
$httpCode = curl_getinfo($ch, CURLINFO_HTTP_CODE);
curl_close($ch);

$data = json_decode($body, true);
if ($httpCode === 200 && !empty($data['success'])) {
    $r = $data['result'];
    $components = json_decode($r['address_components'], true);
    printf("%F, %F (%s) via %s\n", $r['lat'], $r['lng'],
$r['formatted_address'], $r['api_used']);
} else {
    // address not found or request error
}
```

JavaScript (Node.js 18+)

```
const baseUrl = 'https://<service domain>/';

async function geocode(city, postCode, address) {
  const res = await fetch(baseUrl + 'soapserver/json/coords', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      username: process.env.INTIME_GEO_USER,
      password: process.env.INTIME_GEO_PASS,
      city, postCode, address,
    }),
  });
  const data = await res.json();
  if (!res.ok || !data.success) return null;
  const { lat, lng, formatted_address, api_used } = data.result;
  return { lat, lng, formatted_address, api_used };
}

geocode('София', 1303, 'Цар Симеон 97').then(console.log);
```

Best practices

- **Call the API from your server only.** Credentials travel in the request body and must never reach a browser or mobile app.
- **Store the coordinates you receive.** Geocode each new address once and save the result in your system, instead of calling the service every time the address is used. This also keeps you within the rate limits.
- **Send the fields separately.** City, postal code and address in their own fields give more accurate results than the whole address in one field.
- **Use `coords` by default.** `coords_by_api` is for comparing sources and special cases; it skips the fallback chain.
- **Limit requests to `rominai`.** The Romin AI agent is a paid service with a high cost per request, meant only for addresses the other sources cannot resolve.
- **Check critical addresses.** Coordinates come from automated processing; a wrong input address can produce a nearby but inaccurate point. For important addresses, compare `formatted_address` with the submitted address.
- **Coverage:** the service is designed for addresses in Bulgaria.