# Geolocator Wrapper API

The Canadian Geospatial Platform (CGP) needs the ability to do Geocoding

*Geocoding is the process of converting addresses (like "1600 Amphitheatre Parkway, Mountain View, CA") into geographic coordinates (like latitude 37.423021 and longitude -122.083739), which you can use to place markers on a map, or position the map. Google Maps Geocoding API*

There is many different APIs on the market to achieve this task. Each one of them has pros and cons and the use of only one API introduce limitations. The idea is the create a wrapper around many APIs and custom sources of geolocated features to be able to build a standard response for CGP tools to interact with.

## Overview

Graphical user interface, website

Description automatically generated

## Implementation

The API development in AWS:

* Implemented in AWS:
  + <https://nrcan-rncan.awsapps.com/start>
  + Fgp-webpresence-dev
  + nrn-awscloud.internal
  + Region: central-1
* Lambda functions:
  + geolocator-lambda
  + geolocator\_concurrency\_test
* [AWS API Gateway](https://docs.aws.amazon.com/apigateway/index.html)s:
  + geolocator-API
  + geolocator-concurrency-test
* JSON schemas stored in AWS S3 bucket:
  + Dev-app-geolocator
    - Api
      * in-api-schema.json
      * out-api-schema.json
    - services
      * geonames-schema.json
      * nominatim-schema.json
      * locate-schema.json
      * nts-schema.json
      * findplace-schema.json

### Parameters

Geocoding

|  |  |
| --- | --- |
| Q | The query: spaces ‘ ‘; plus sign ‘+’; scape\_code ‘%20’ are accepted  – mandatory if no table parameter |
| Lang | The return language. valid values [en/fr] – Optional, default: en |
| Keys | The ids of APIs to query separated by , – Optional, default: all of them |
| Dev | Show null and undefined values for development [true,false]– Optional, default: false |
| Table | Return contents of table created by csv file [generic, province, component, category, tableurl]- Mandatory if no Q parameter, default: none |

Ex.

https://fr59c5usw4.execute-api.ca-central-1.amazonaws.com/dev?q=Meech%20lake&lang=en&keys=geonames,nominatim

## Supported API’s

Geonames:

url: https://geogratis.gc.ca/services/geoname/\_PARAM1\_/geonames.json

where \_\_PARAM1\_\_ is the language for the query (see ‘parameters’).

Parameters:

* q: string text including spaces ‘ ‘ or plus signs ‘+’ with the data requested.
* lang: language for the query. Only accepts ‘en’ or ‘fr’

Tables: It requires two series of tables originally found in specific urls (see schema), then read and saved in memory.

Returns: Json dictionary with a key named ‘items’ containing the features.

Nominatim:

url: https://nominatim.openstreetmap.org/search

Parameters:

* q: string text including spaces ‘ ‘ or plus signs ‘+’ with the data requested.
* lang: replaced by “accept-language”.

staticParams: Required to get data in the expected format.

* “countrycodes”: “CA”
* “format”: “jsonv2”

Returns: Json with the features list.

Additional. The coordinates inside the bbox must be rearranged to match the required order.

Locate:

url: https://geogratis.gc.ca/services/geolocation/\_PARAM1\_/locate

where \_\_PARAM1\_\_ is the language for the query (see ‘parameters’).

Parameters:

* q: string text including spaces ‘ ‘ or plus signs ‘+’ with the data requested.
* lang: language for the query. Only accepts ‘en’ or ‘fr’

staticParams: Required to get data in the expected format.

* “expand”: “component”.

Returns: Json with the features list.

\*\*NTS:

url: https://geogratis.gc.ca/services/delimitation/\_PARAM1\_/nts

where \_\_PARAM1\_\_ is the language for the query (see ‘parameters’).

Parameters:

* q: string text including spaces ‘ ‘ or plus signs ‘+’ with the data requested.
* \*lang: language for the query. Only accepts ‘en’ or ‘fr’

Returns: Json with the features list.

\* This parameter is irrelevant for the query. It must be removed.

\*\* This service is for developing purposes. Should be removed from future versions.

findplace:

url: https://maps.googleapis.com/maps/api/place/findplacefromtext/json

Parameters:

* q: string text including spaces ‘ ‘ or plus signs ‘+’ with the data requested.

staticParams: Required to get data in the expected format.

* “inputtype”: “textquery”
* “fields”: “name,geometry”
* \*\*\*“key”: “?”

Returns: Json dictionary with a key named ‘candidates’ containing the features.

geocode:

url: https://maps.googleapis.com/maps/api/geocode/json

Parameters:

* q: string text including spaces ‘ ‘ or plus signs ‘+’ with the data requested.

staticParams: Required to get data in the expected format.

* \*\*\*“key”: “?”

Returns: Json dictionary with a key named ‘candidates’ containing the features.

\*\*\* The key is provided by the service provider (in this case Google) for a specific (???). In our case it should be for the API service (???).

## Adding a new API data source

1. Add into the list of services to make it valid for the parameter ‘keys’ and be present in the list on the UI.

S3.dev-app-geolocator/api/In-api-schema.json:

              "items": {

                "type": "string",

                "enum": ["geonames", "nominatim", "locate", "nts"]

              },

1. Add the service into the properties list to make it feasible into the services loop inside the lambda funcion.

S3.dev-app-geolocator/api/out-api-schema.json:

  "properties": {

    "geonames": {

      "type": "array",

      "uniqueItems": true,

      "items": { "$ref": "#/definitions/output" },

      "description": "The Geoname api result set."

    },

1. Add the service’s schema in the S3 container.

Amazon S3.Buckets/dev-app-geolocator/services/

Table

Description automatically generated with low confidence

1. Test and modify the schema to match the load data structure.

## Next steps and improvements

* Schemas
  + Redefine. Remove from the schema’s syntax all the unnecessary tokens (inherited from AWS API’s Schemas rules) to simplify the interpretation/application of them.
  + Addition. At least two more schemas from google can now be easily tested to be included.
  + Authorize. To get a key for access Google API from AWS cloud instead of this computer.
* Code
  + Class/Object refactoring. There is a couple of classes already implemented inside the application. The ultimate goal would be to have an all-object application.
  + Some “pairs review” to maintain the quality of the coding.
* Versatility
  + Refactoring the AWS API to include several resources. Each one adapted to a specific set of parameters. (ex. ‘latlng’ instead of ‘q’).

https://fr59c5usw4.execute-api.ca-central-1.amazonaws.com/dev/byQuery?q=Meech%20lake&lang=en&keys=geonames,nominatim

https://fr59c5usw4.execute-api.ca-central-1.amazonaws.com/dev/byCoords?latlng=46.77,-71.10&buff=20&lang=en&keys=geonames,nominatim

* Reusability
  + Either create several functions to satisfice all query formats using the same objects or create one flexible enough to interpret and validate correctly several set of parameters.
* UI. Complete and develop the front-end