CitySuggestionsAPI

Version	Version information	Author
1.0	Initial Documentation	Jean-François Leblanc

1. API Calls

1.1 api/cities/get

Parameters : int id
Returns : JsonResult representing a CityViewModel
Function : returns a CityDropDownViewModel populated with the city with corresponding id. If no corresponding city is found, returns a CityViewModel with default values.

1.2 api/suggestions/get

Parameters : string q, double? longitude = null, double? latitude = null
Returns : JsonResult of an array named suggestions, each element being a CityDropDownViewModel
Function : returns a list of cities in Canada or the United States which have q as their prefix. If q is empty, returns all cities. If longitude and latitude are valid coordinates, returns the array sorted according to a score representing how far each city is from the specified longitude and latitude.

2. CityUtilities

2.1 GetCities

Parameters : string q, double? longitude = null, double? latitude = null
Returns : IEnumerable<CityDropDownViewModel>
Function : calls GetCitiesFromTsv to get an IEnumerable<City> and then uses the prefix to get only the cities with a matching prefix. Processes the city data and then maps each city to a CityDropDownViewModel to be returned as an IEnumerable<CityDropDownViewModel>.

2.2 GetCity

Parameters : int id
Returns : CityViewModel
Function : calls GetCitiesFromTsv to get an IEnumerable<City> and then tries to match a city to id. If a city is found, maps it to a CityViewModel and returns it, if not returns an empty CityViewModel.

2.3 GetCitiesFromTsv

Parameters : none
Returns : IEnumerable<City>
Function : reads the cities found in the static cities.tsv file (read “Further Considerations” section for discussion on a better way to store data) and uses the open-source CsvReader to read through the Tsv and each entry to a City Object. Returns the enumerable of all those City objects.

3 City Extensions

3.1 Calculate Score

Parameters : this City city, double latitude, double longitude
Returns : void
Function : this is an extension method that can be called on a city object, which will take as parameters certain coordinates and then use CalculateDistance to get the distance between the city and the passed coordinates. Using that distance, a score will be returned. The score is calculated on a linear scale, 1 being an exact match and 0 being when the compared coordinates are further away than 12.756 KM which is the diameter of the earth. A difference of 1 KM is reflected as around 0.00008, so 5 decimal points are used.

3.2 MapToDropDownViewModel

Parameters : this City city
Returns CityDropDownViewModel
Function : this is an extension method that can be called on a City object and returns a populated CityDropDownViewModel representing that City.

3.3 MapToViewModel

Parameters : this City city
Returns CityViewModel
Function : this is an extension method that can be called on a City object and returns a populated CityViewModel representing that City.

3.4 UpdateAdministrativeRegionAndCountry

Parameters : this City city
Returns : void
Function : this is an extension method that updates the country name (for instance US becomes United States) and the administrative region (for instance in Canada, 10 becomes QC).

3.5 CalculateDistance

Parameters : double lat1, double lon1, double lat2, double lon1
Returns : double
Function : this function takes in two sets of coordinates representing the longitude and latitude of locations on earth and returns the distance between two points using the haversine formula as described in this article :
https://www.movable-type.co.uk/scripts/latlong.html

3.6 GetCountryName

Parameters : string code
Returns : string
Function : simple switch case that maps the country code to the country name. See “Further considerations” section for suggested improvements.

3.7 GetProvinceName

Parameters : string code
Returns : string
Function : simple switch case that maps the province code to the province name. For US states, simply returns the code itself. See “Further considerations” section for suggested improvements.

3.8 Radians

Parameters : double x
Returns : double
Function : mathematical function that converts angles into radians.

4. Further Considerations

4.1 Storing Data

For the purpose of this coding challenge, the CsvReader library was used. This library is quite easy to set up and fast at reading however it has many limitations. The main limitation is that it is forward only, which among other things limits the way in which we can access the data. It does not have any indexes like SQL would and therefore it is necessary to return all the cities to simply select a single city with an index.

If this API was to be used further, we would first need to migrate all the TSV data into SQL data, which would then be both faster and more flexible.

4.2 Country and province references

Similarly to point 4.1, the country and province references used to translate from CA to Canada for instance or from an admin number to a province code is a simple switch case. This is a simple function used to showcase the API but ideally this type of data would be stored in an SQL table and a simple select could return the matching province name to province code and country code to country name.

This would also easily allow the API to be used for cities outside Canada and the US, all we would have to do would be add all the country names and their country codes. We would then do the same for administrative regions and further administrative subdivisions if need be.