Busbud coding challenge : Documentation

(See TimmyCarbone on GitHub)

I – Development environment

Application has been developed with Ruby 2.0.0 and Sinatra. Tests had been proceeded over a local Thin server and the final application is now deployed to Heroku :

http://guarded-refuge-1622.herokuapp.com/

Since the application does not contain many functionalities, I prefered to centralize its code on a single class. I could separate it in different classes (one for parsing, one for scoring and one to organize for example) but I tought that it would be complicating the principle of this application. However, the code is commented and each functionnality is being clearly noticed so if this application is willing to expand its possibilities, we could easily implement new classes.

II – Data extraction

Cities extraction from the CSV table is a heavy operation that spend a lot of execution time. Auto-completion is a tool needing to be as fast as possible because it concerns user interaction with the machine. The analysis showed that searching in the CSV cost around 1.3 seconds on the local machine and we can't afford to have an auto-completion lasting more than 1 second.

The goal was to reduce this execution time by parsing the CSV file only once, when the server loads, and then store it in a global variable. It surely takes more resources but the gain on user experience is clearly worth it. Finally, responding to a request only last around 30ms allowing to answer to more than 30 requests (users) by second with a single Heroku dyno.

The cities which population is under 5000 are not considered during the parsing. It means that not all the cities are stored in a variable but only the interesting ones. This helps in saving some resources.

III – Parameters checking

There are several errors that can be made if parameters are not considered correctly. That's why it is necessary to check if parameters are correct :

- Latitude and longitude must be float numbers
- A query parameter should be pass (q= ...) even if it is empty

Also, the application needs to consider different ways of writing so it downcases, deletes accents and consider spaces from the query and cities names.

A new parameter has been implemeted. This parameter is called 'limit' and describes the maximum number of cities to be returned. For example, if the query should return 25 cities but 'limit' is set to 15, only the best 15 cities will be returned. It could be useful for a UI implementation that doesn't want a large list of possibilities to auto-complete, but only a precise and efficient one.

III – Scoring algorithm

The scoring algorithm parses the array containing all the cities and considers those which matches with the prefix in parameters (q=....). For each of these cities, 3 scores are given :

- A matching score : depending on the length of the query regarding the length of the city name.
    - Score = length of query/city name
- A distance score : depending on the distance between the city and the localisation parameters (latitude and longitude)
    - Score = 1 - distance/maximum distance
- A population score : depending on the city population
    - Score = 1 - 5000/population

The distance score considers earth like a sphere (not exactly the case but precise enough) and the maximum distance being the distance between two points on exact opposites of the sphere.

The population score changes between 0 (5000 pop) to nearly 1 (very large population). This score is made because we expect people to search for big cities rather than searching for small ones.

The score is rounded to two decimals.

Matching cities are sorted by score (greater before) and alphabetical order if the score is equal.

IV – Support and maintenance

You can check the application on GitHub at :

https://github.com/TimmyCarbone/coding-challenge-backend-b

If you have any question or feedback, please refer to my GitHub page :

https://github.com/TimmyCarbone