EMPATICA ASSIGNMENT DOCUMENTATION

1. Introduction

The purpose of this documentation is to briefly describe the implementation of a dashboard that can be used by the Empatica team to have a real time overview of their mobile apps downloads.

2. Implementation

The client and server implementation of the dashboard are described in this section.

2.1 Server

The server is implemented in python, using django and the django rest framework, used to exposes REST compliant APIs.
The data are stored in a sqlite database.

2.1.1 Database

The database contains a single table “Data”, with the following attributes:

- longitude
- latitude
- app_id
- downloaded_at
- country

Its structure is represented in the file “model.py”

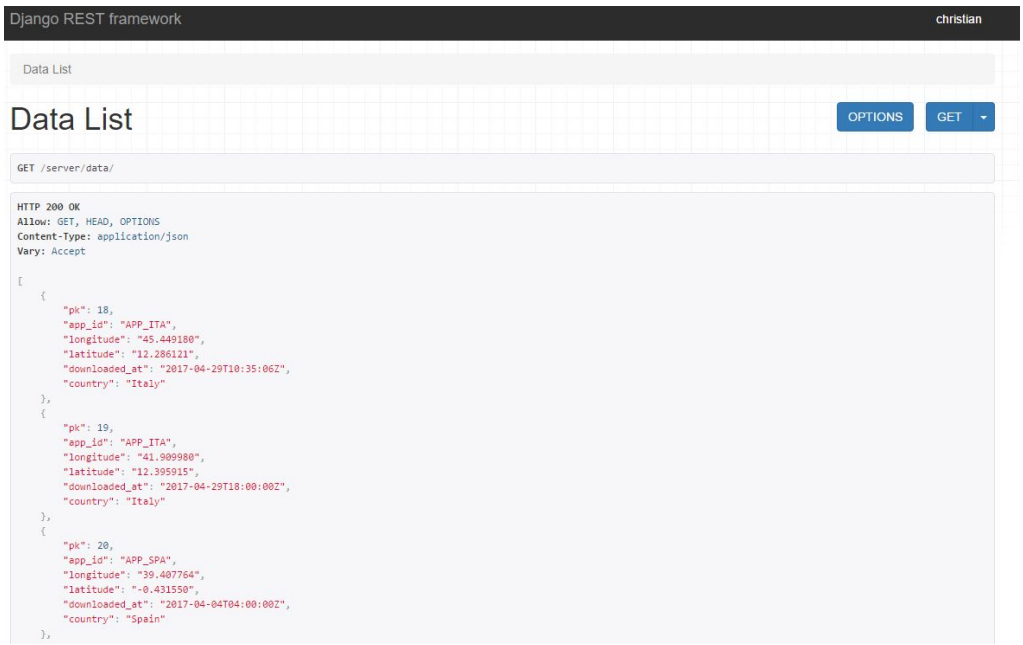
2.1.2 APIs

The server exposes the following APIs

- List of data:
  The only action allowed is **GET**, considering that it should not be possible for the Empatica team to add fake download data.
  Contains the full list of json serialized data, with all the information regarding the downloads. It is accessible via “*http://127.0.0.1:8000/server/data/*” url.

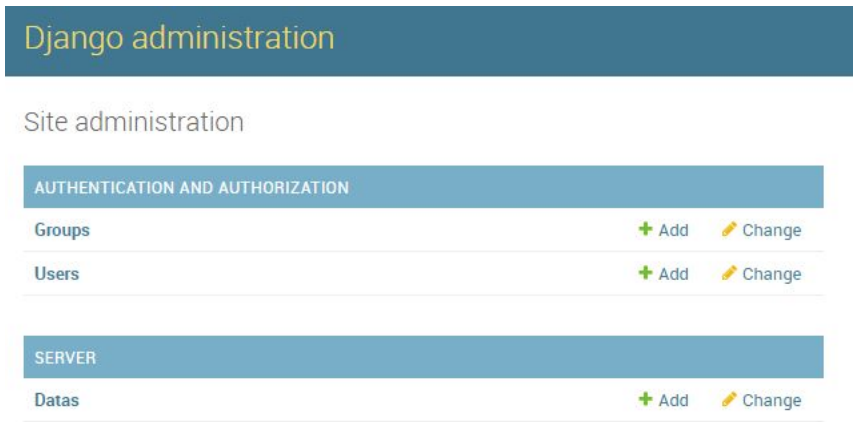
- Data detail:

The only action allowed is **PUT**.
It is used to update the “country” attribute with the information gathered by the geocoder service launched in the client. Once the attribute is populated the client won’t call the geocoder service anymore.
It is accessible via “*http://127.0.0.1:8000/server/data/*N” url, where N is the primary key identifying the data.



2.1.3 Admin panel

The admin panel is a graphical representation of the database, offered by Django, and it allows the manipulation of the data stored in the database.
From this interface it is possible to add fake data into the system, simulating the addition that should usually be automatic whenever a user downloads the application.



2.2 Client

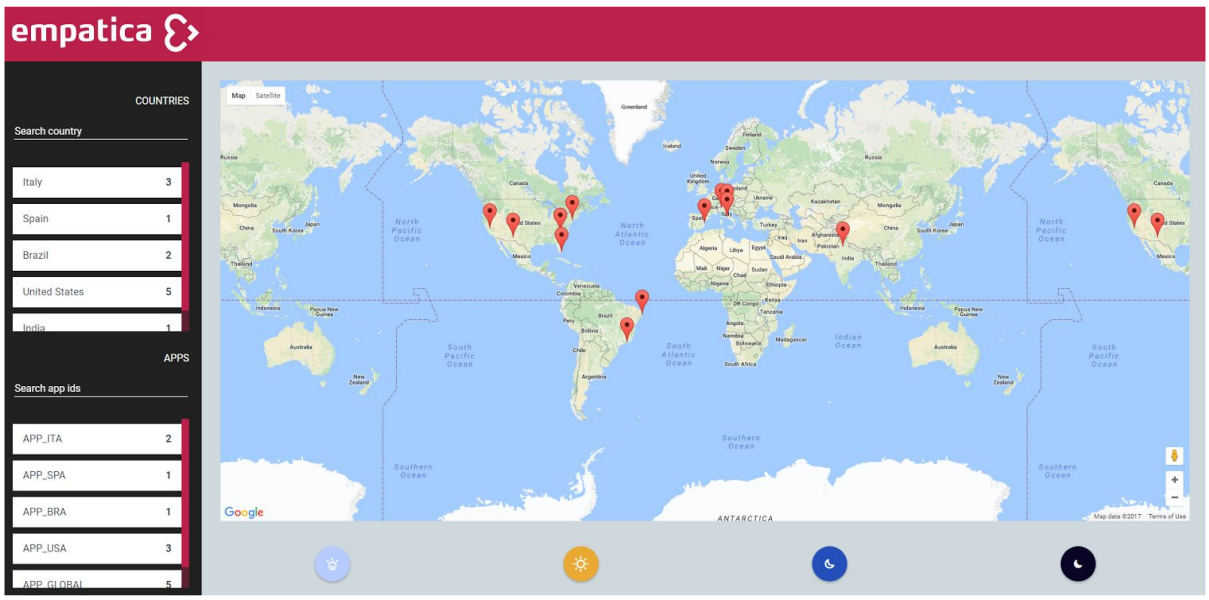
The client consists in a single page application created in HTML, styled with basic CSS and populated using javascript, relying on the AngularJS framework. Additionally I used the angular material library, used to give a material-like style to the application.

The main module, called “dashboard” is initialized in the file “app.js”. In the same file there are the theme configurations.

2.2.1 Model

The model of the data are gathered by the Api service contained in the api.js file, which is responsible of the interrogation to the API exposed by the server.
As can be stated, it queries the same urls described in paragraph 2.1.2.

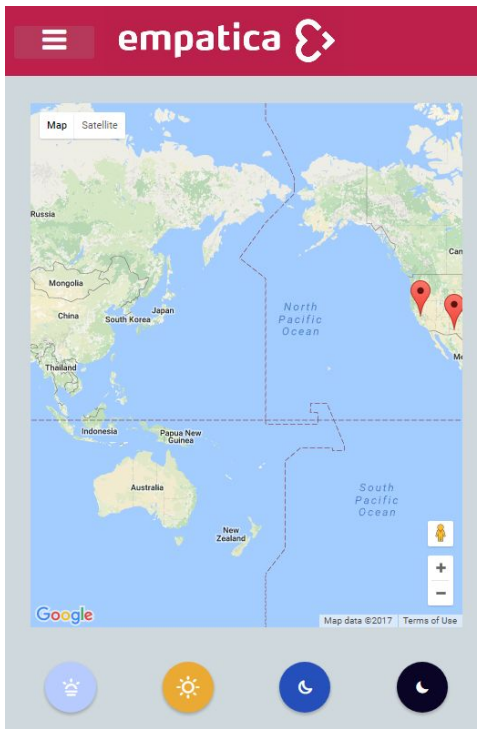
2.2.2 View



The view of the application is encapsulated in the single file called “index.html” containing all the html layout. It includes the scripts used in the application and it makes a wide use of Angular expressions to dynamically populate the graphical interface.
There are three main sections:

- A header, containing the logo and a “Reset filter” button that appears only when some filter is applied by the user
- A sidenav, in which it is possible to see the list of countries and the relative amount of downloads and the list of applications downloaded. From this section it is possible to filter the data displayed via a “search” functionality or via “click” on the desired item (country or app id)

- A main content in which the map is displayed and in which it is possible to filter the data displayed by time of the day



2.2.2.1 Filter by searching

As briefly described in the paragraph above, the sidenav of the dashboard presents the list of countries in which the applications have been downloaded and another list displaying all the app ids of the downloaded app.
Above the two lists there is a search box in which it is possible to type the name of the country or the application interested. Typing any letter will produce a change both to the lists displayed and to the marker on the map. In particular, the amount of download is re-computed in order to indicate the actual amount of download available for the given filters.

2.2.2.2 FIlter by selecting

A faster way to filter the result is to actually select the desired country (or app id) and, in this way, obtain an immediate view of the situation in the selected country (or app id).

2.2.2.3 Filter by time of the day

The four icons placed under the map, allow the user to filter the results by time of the day, respectively morning, afternoon, evening and night.

2.2.2.4 Reset filters

Whenever a filter is applied, a dedicated button will appear in the top right corner of the dashboard, allowing the user to immediatly clear all the filter applied and restoring the initial representation.

2.2.3 Controller

The single controller, populating the index.html is used to gather the information from the server (via the API service) and to display them in the interface.
It is logically divided into three sections:

- Interaction with google maps (interaction with the marker in the map and geocoding of the locations)
- Filters: in this section there are several functions used to dynamically filter the data displayed in the map and in the lists in the sidenav. The avoidance of the “filter” function in the Angular “ng-repeat” directive is made on purpose considering that, every interaction with the items displayed causes changes not only to a single list, but to all the others.
- UI interaction: it simply handles the sidenav visibility under a certain screen resolution.

2.2.4 Style

The angular material library offers the possibility to easily implement responsive design, using the flex directive and specifying the percentage of space that a certain element will cover on the screen. In addition it is possible to specify different behaviors depending on the screen size and resolution.
In addition I provided a *style.css*, which is not organized by elements, but by action. This means that, apart from certain borderline cases, there won't be attributes related to every single element in the page, but there will be general class, specifying height, font size, padding and margin that, when needed, could be added to the required element.

3. How to run

In this section I will guide you through the steps needed to run the application.

1. Install python 2 and pip
2. From the command line, type the following string
    - *pip install django*
    - *pip install djangorestframework*
    - *pip install django-cors-headers*
3. If you are using windows, click con the start.sh file to run the server.
4. Otherwise

    - from the command line, move to the folder Dashboard/Server/dashboard
    - From the command line, run the following code
        i. *python manage.py runserver*
5. Click on the index.html file contained in the folder Dashboard/Client

4. Expansion

How would you design the solution for the following requirement: Data must be visualised in real time in the map. That means that as soon as new data are saved in the database, the map should reflect the change.

The first, more straightforward and resource consuming answer that came to my mind, involved the continuous polling, meaning the continuous sending of AJAX requests to the server in order to get the most updated data from it. This approach, of course, is as realtime as the developer decides it to be (a request every 30 seconds, means that data could be refreshed every 30 seconds, meaning that they could be out-of-date for at most 29 seconds). Moreover, this approach is clearly resource consuming, requiring continuous requests to the server, even when no update is available.
A slight variation to this approach is the long-polling, that held a connection open until new data are provided by the server.

Another approach take under consideration involves the HTML5 server-sent events, in which the focus moves from the client requesting update to the server sending them whenever available. This approach is less consuming, but, on the other hand, being a relatively new standard, it is not fully supported by all browsers (specially in their older versions).

Assignment notes*: unfortunately I did not have the time to consider the dockerization, but i really want to thank you for giving me the opportunity to get to know this software. During the weekend I downloaded it and quickly read the starting guide. Unfortunately, I did not have much time and, stuck in the phase of running the server, I had to abandon the configuration of the Dockfile. For the whole week I was in Rome for work, far from my personal computer, so I could not go further with its configuration.*