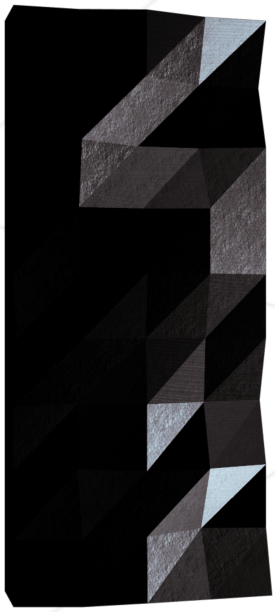

Rage Against The aPi

dup ta da dap

42 Staff pedago@staff.42.fr

Summary: This document is the subject for the API and algorithm rush

Contents

Chapter I

Forewords

Open Data France is a public initiative started in 2013 by some cities in France, including Paris, Lyon, Bordeaux, and the General Council of some departments.
This initiative aims to promote the concept of open data and to make data from local authorities and companies public.

The city of Paris has made a lot of interesting data public, such as:

- Parisian trees (including type, length and circumference)
- The most loaned books from librairies
- Different bars serving one euro coffee
- WiFi hotspots
- Most given first names from 2004 to 2014
- Delivery places
- Public clocks

Chapter II

Introduction

Using the RATP's API, you will have to create your own travel planning application.

Chapter III

Goals

By entering a starting point and an ending point, your application will have to determine the most efficient route using RATP metro lines to get from start to finish.

Chapter IV

General instructions

The official API for RATP, hosted at www.ratp.fr, isn't documented. Use the API at `navitia.io` instead.
Language is up to you, as well as the form of the program: a website, a GUI application, a script, whatever.
You can find the API documentation on Github. Read it.
The endpoint to use is `https://api.navitia.io/v1/`. You'll need to create your own token to query the API.
You are **not authorized** to use the `journeys` resource, the goal here is to reproduce its behavior.
You can use external libraries to parse JSON or make API requests, but everything else has to be done by yourself. You must use your own data structures and implement your own algorithm.

`Your token will limit your number of requests to 3000 per day. It is more than enough, but then again you should maybe cache some data locally...`

Chapter V

Mandatory part

Your program will take two arguments: a departure point and an arrival point. Use the API to detect the metro stations closest to this point.
Your result will include the lines to take and their direction, the transfers, and a timeframe for the whole journey.
The timeframe does not have to be precise but it has to give an idea of the expected travel time.
Of course, the journey starts when a metro train departs from the starting point, not before...

For example:

```
$> python ratp.py 'porte clichy' lyon
0 - Porte de Clichy (Paris)
Choose the station ID:
0
0 - GARE DE LYON (Paris)
Choose the station ID:
0
for requested departure time: 13:41
L.13 from Porte de Clichy to GARE ST LAZARE direction: Chatillon Montrouge at 13:42 total stations: 5
L.14 from GARE ST LAZARE to GARE DE LYON direction: Olympiades at 13:53 total stations: 4
Arrival at: 14:13 Total stations: 9, transfers: 1
```

The "places" resource allows a google-like search. It is a good way to find the station closest to your requested starting point.

Chapter VI

Bonus part

Once your mandatory part is complete, you are free to add some bonuses that enhance your program. Here are a few ideas that you might want to implement:

- Handle the bus, tram, RER, TER lines
- Work on the timeframe and allow the user to add an argument for date/time of departure or arrival
- Open a webpage to Google Maps with every stop marked
- Handle the `trafic_reports` resource to find out about traffic disruptions and modify your itinerary accordingly
- Make a nice user interface

Using external libraries for bonuses is allowed, but it has to be justified during the peer-evaluation.

Chapter VII

Turn-in and peer-evaluation

Turn your work in using your `GiT` repository, as usual. Only what's present on your repository will be graded in defense.