Safe travel for London 'n shiet

Dario Pagani, 585281
Federico Frati, 585281
Ricky Marinsalda, 585281

January 20, 2023

Contents

Part I

Introduction

Every day millions of people find themselves in the troublesome task of computing a route to their desired destination. We'll provide directions for their journey with state of the art routing algorithms in a safe way. The service will provide the following transportation modes when querying a route: car, foot and bicycle. We'll also provide a compact and easy way to view timetables and public transport's routes.

Problem Navigating a city like London is a complicated matter even during normal times, let alone during rush hour. Due to the convoluted nature of London's road network accidents and disruptions are common occurrences that afflict all road users during their journeys [?]. People who use public transportation to travel around the city could be interested in knowing which are the most congested lines in advance in order to avoid delays and other kinds of interference.

Objective We'll provide analytic functionalities to find hotspots in the road network graph, that is to collect information about all users' journeys and visualize the most congested routes/graph's regions. Other information about public transportation's issues will help users to locate and avoid the most critical parts of the transportation network.

Part II

Feasibility

Chapter 1

Data sources

We're combining data from three sources and two organizations:

- Transport for London TIMS
- Transport for London public disruptions API
- OpenStreetMap

Introduction At the very beginning, we made a feasibility study of the project idea, in order to identify both the pros and the critical aspects of the application, considering the time and technical constraints at the time and thinking how to possibly improve our work in future.

1.1 Transport for London

Scraping Transport for London provides data at regular intervals – every five minutes – and in a standardized format; this kind of data, especially when scrapped during an adequate period of time can provide a good amount of information to be analyzed by our application. Those analyses become of particular interest when we consider that Transport for London provides data with a great amount of detail, including things such as minor collisions and broken traffic lights. In our particular case we collected data for circa a month using TLF's JSON API with a simple shell script that was being executed automatically by a SystemD timer every ten minutes.

schema The main challenge was to find a "schema" for the document database to store in an unified way, that is to share the same MongoDB collection, between road disruptions and public transportation disruptions; as it could be useful for certain operations to have an unified view of such data.

1.2 OpenStreetMap

Scraping Since the map's data would be used only for routing purposes, we don't need all the information provided by OpenStreetMap; so, to reduce the dimension of the data and maintain only the useful information, we decided to do some pruning of the network graph, in particular we removed useless nodes such as buildings, waterways, trees, parks and so on.

Schema The main challenge was to find a good representation of OpenStreetMap's data for our chosen graph database (neo4j) because it has to represents facts like one ways streets, access restrictions and the elements' geographical positions; we iterate over several possible schemas for the graph database and over many different ways to import the raw data into it.

Figure 1.1: Example of intersections

Part III

Design

bla bla bla

Chapter 2

Main Actors

The application has three main actors:

- Client
- TIMS
- Statician

Client this user is able to view map and run searches on it and he's also able to ask for directions between two points on the map.

TIMS this user is able to perform CRUD operations on the disruptions.

Statician this user is able to perform analytics on the data generated by TIMS.

Chapter 3

Functional requirements

The application will allow its users to perform the following tasks:

Client:

- **View the map**
- **Move the map in a direction**, with the typical drag 'n drop
- **Increase or decrease the map magnifying**, with either buttons or the mouse's wheel
- **Search for a specific place**, the following method are supported:
  - With an address
  - With a POI's name
  - With WGS84 geographical coordinates
- **Ask for directions** between two intersections on the map
  - Via car
  - Via bicycle
  - On foot

Chapter 4

CAP Theorem issue

Chapter 5

Use cases

Chapter 6

Data model

6.1 Document database

DBMS Our choice for the *database management system* to handle the document database was *MongoDB*, since it is the most popular *DBMS* of its kind and it also provides several functionalities useful for our use case, such as indexes and a powerful query engine.

6.2 Graph database

DBMS Our choice for the *database management system* to handle the graph database was *Neo4j*, since it is ???

6.3 Key-value database

DBMS Our choice for the *database management system* to handle the graph database was *Rédis*, since it provides excellent performance for our use case, that is to act as a cache for most frequent users' queries.

Chapter 7

Distributed database

Chapter 8

Overall platform architecture

Chapter 9

Framework used

Part IV

Implementation

Part V

Conclusion

Part VI

Manual

Appendix A

User

Appendix B

Statistician