#### Safe travel for London 'n shiet

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

January 2, 2023

### Contents

Ι	Introduction	2
II	Feasibility	4
1	Data sources 1.1 Transport for London	<b>5</b> 5
II	I Design	7
2	Main Actors	9
3	Functional requirements	10
4	CAP Theorem issue	11
5	Use cases	12
6	Data model6.1 Document database6.2 Graph database6.3 Key-value database	13 13 13 13
7	Distributed database	14
8	Overall platform architecture	15
9	Framework used	16
ΙV	/ Implementation	17
V	Conclusion	18
V	I Manual	19
$\mathbf{A}$	User	20
В	Statistician	21

# 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

#### 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.

```
1 id:ID,latitude:double,longitude:double,coord:point{crs:WGS-84},:LABEL
2 78112,51.526976,-0.1457924,"{latitude:51.526976, longitude:-0.1457924}","Point"
3 99878,51.524358,-0.1529847,"{latitude:51.524358, longitude:-0.1529847}","Point"
4 99879,51.5248246,-0.1532934,"{latitude:51.5248246, longitude:-0.1532934}","Point"
5 99880,51.5250847,-0.1535802,"{latitude:51.5250847, longitude:-0.1535802}","Point"
```

Figure 1.1: Example of intersections

Part III

Design

bla bla bla

## Main Actors

## Functional requirements

## CAP Theorem issue

Use cases

#### 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.

## Distributed database

## Overall platform architecture

## Framework used

# Part IV Implementation

# Part V Conclusion

Part VI

Manual

## Appendix A

User

## Appendix B

## Statistician