ASHESI UNIVERSITY COLLEGE

BUILDING A MAP-BASED TRANSIT PLANNER FOR THE TRO-TRO SYTEM IN ACCRA

APPLIED PROJECT

B.Sc. Computer Science

Andrew Abbeyquaye

2017

ASHESI UNIVERSITY COLLEGE

BUILDING A MAP-BASED TRANSIT PLANNER FOR THE TRO-TRO SYTEM IN ACCRA

APPLIED PROJECT

Applied Project submitted to the Department of Computer Science, Ashesi University College in partial fulfilment of the requirements for the award of Bachelor of Science degree in Computer Science.

Andrew Abbeyquaye

2017

DECLARATION

I hereby declare that this applied project is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

……

Candidate's Name:

……

Date:

……

I hereby declare that preparation and presentation of this applied project were supervised in accordance with the guidelines on supervision of applied projects laid down by Ashesi University College.

Supervisor's Signature:

……

Supervisor's Name:

……

Date:

……

Acknowledgements

To my supervisor, Dr Ayorkor Korsah whose encouragement and academic advice helped me undertake this project.

Abstract

Technology has been a useful tool in the way we transcend our limitations as human beings. In the transport sector, we have seen remarkable improvements such as the variety of modes through which we travel as well as the applications that make planning our journey more easy and effective. Transit planning applications have been used in developing countries to make sense of their transportation systems and to help people travel in more effective ways. However, most of these applications have limited coverage or reduced functionality when used in developing countries such as Ghana since the data these functions depend on are usually inexistent. In this project, a transit planning application is implemented to make use of tro-tro data to help users obtain a travel itinerary for their journeys within Accra while minimizing travel distance and time.

Table of Contents

Chapter 1: Introduction

1.1 Introduction

Technology is continually being used to improve our lives. Various consumer technologies have made it possible for us to interact and explore our world in smarter ways. The pervasiveness and ubiquity of present-day technology is responsible for the fact that today, we possess much computing power in our personal devices than computers did some years ago. This has enabled individuals to personally improve the way they work, travel, interact with others among others.

Of interest to this project is how technology has improved the way we transport ourselves from one location to another. The power to know beforehand the best mode of transport for a given journey exists at our fingertips through geospatial routing applications that are accessible via our personal computers and smartphones. Today, it is possible for one to use a map application (say Google Maps) to get the best route by public transport from a given start point to a specified destination. These applications often work well for developed countries. However, for most developing countries such as Ghana, these applications come with reduced functionality.

For example, if your start and stop points when using Google Maps are within the borders of Accra, you do not get the option of a transit itinerary via public transportation. This has been the case because data on our public transport system was unavailable in the past. Today, this data exists and is available online for free download ("Accra TroTro Apps Challenge - the Datahub", 2016). However, it has yet to be used to solve this informational gap either by integrating it with an existing application or incorporating it in a new one.

In this paper, I present a solution that uses the data on our local bus service (known as 'tro-tro' or 'troski') to the provide an option for transit planning within the Accra metropolis.

1.2 Background

A large number of Accra residents are very mobile, often commuting to work, school, market centres, hospitals among other places. The modes of transportation commonly used include cycling, walking, chartering a taxi, driving as well as using the local bus service (tro-tro). Of all these modes, the most used is the tro-tro service which caters to 70% of all city resident commuters. 11% prefer to commute by foot, 10% prefer driving their own cars whereas 8% makes use of taxis. In addition to the above, only 0.3% of commuters prefer the Metro Mass Transit buses. (World Bank, 2010)

A subsequent report on transport indicators by the Ghana Statistical Service in 2013 revealed that about 78.7% of respondents in the Greater Accra Region have commuted by bus before with 67.2% of them having made up to 5 trips using tro-tro in the week leading up to the survey. (Ghana Statistical Service, 2013).

Metro Mass Transit buses are to be distinguished from tro-tros. Tro-tros are privately owned minivans or minibuses operating under the Ghana Private Road Transport Union of the Trades' Union Congress (GPRTU of TUC). These buses seat about 16 to 24 persons where boarding is conducted at stations and thereafter they travel fixed routes. On the other hand, the Metro Mass Transit buses are operated by the Metro Mass Transit company whose largest shareholder is the Government of Ghana. These buses seat more passenger but are very few. This makes them less available than tro-tros. (MMT, 2013)

Using the tro-tro service requires some knowledge about the routes they ply as well

as the various stops they make during each journey. Journeying to a place for the first time can be hectic if one does not know which buses to take. To avoid this, one could charter a taxi which would be costly. Also, even if one knew which buses to take, there is no immediate or guaranteed way of knowing that your chosen route would get you to your destination fastest when compared to alternate routes.

These concerns present an excellent opportunity where technology can be used to effectively bridge the inherent informational gaps. By building an application that uses the available data on tro-tros, I hope to enable tro-tro passengers make their journeys in a more effective way while choosing to minimize travel time, reduce bus changes as well as helping them get to their destination using the shortest possible path.

1.3 Related Work

1.3.1 Google Maps Transit

Google Maps Transit is a service provided by Google Maps that gives users routing information for selected areas. The service calculates routes, travel time as well costs and can compare alternate modes and routes of transportation. Google created the General Transit Feed Specification (GTFS) formerly Google Transit Feed Specification, as a simple specification for sharing transit data. GTFS data is required for routing information to be provided. This data contains routes, schedules, fares, trips, stops as well as stop times. This information is supplied to Google's routing engine when a request is made and the resulting path/route is displayed on the maps interface. (McHugh, 2013)

The coverage for Google Transit is publicly available. However, Google relies on transit agencies to upload their GTFS data before options for routing can be made

available to their users. Individuals are not allowed to upload transit data per the current policies. (Google, 2017)

1.3.2 The Digital Matatus Project

The Digital Matatus Project was conceived out of a collaboration between Kenyan and American universities as well as the technology sector in Nairobi, Kenya. The project was aimed at collecting public transit data for Nairobi, making this data freely available to the technologists for routing applications to be built and a transit map to be designed for the city. The data was collected using cellphone technology and ordered per the GTFS data structure. A transit map was also designed and released in tandem with the GTFS transit data.

Workshops were organized to brief the technology community on how to gain insights into the data after which five applications (Ma3route, Flashcast, sonar, digitalmatatu, matatumap) were developed to provide routing application to the public. (Williams et al., 2015)

1.4 Significance

The above applications highlight how transit data provided in the GTFS format are used to provide routing information to the public. In the Digital Matatus project, the data collection process is detailed to show how the various informalities associated with the bus system in Nairobi were accounted for.

Also, Google's Transit application would serve as a useful design precedent worth learning from for this project. Google Transit's architecture differentiates between the data

model and the view model. This setup allows for easy replacement and improvements of application components.

Chapter 2: Requirement Specification

2.1 Overview

This chapter gives a detailed analysis of the scope of the project as well as the functionality that it would offer to end users. This project, as introduced earlier would make use of transit data on tro-tros in Accra to offer a transit itinerary to users based on their given start and destination bus stops. The requirements that this project seeks to meet would have to be ascertained from the application's potential users.

However, in Cherry et al. (2006) various guidelines are given for the requirements that routing planning applications should meet. These guidelines are based on studies of earlier works and activities that mimic the decision-making processes of users. The findings from these studies forms the bedrock for the requirement specifications of this project. These requirements would then be extended to accommodate peculiar information that is location-specific to Accra.

2.2 User Identification and Use Cases

This transit planning application is meant to be used primarily by tro-tro users and others who wish to use tro-tros but do not know which routes these buses ply. Since the application would make use of services available on the internet, potential users of this application would also need to have some basic technological knowledge of how to use the internet. To offer more insights into how the application may be used, below are some use cases.

1. A student going to a university for the first time in Accra wishes to get there using tro-tro. He knows which bus stop to begin his journey and the closest bus stop to

the school. Before leaving home, he logs on to the application and requests for an itinerary that gives him step-by-step instructions on how to get there.

2. A group of tourists visiting Accra for the second-time attempt to use Google Maps to find directions to their hotel from the Airport. They feel they wasted enough money on taxis during their last visit. They want to use Google Maps' direction to board a tro-tro but find out that there is no option for transit via tro-tro. They log on to this application to get the tro-tro itinerary for their journey.

2.3 Requirement Analysis

From Cherry et al. (2006), the main function of a map-based itinerary planner is to provide a transit user with an intuitive and interactive trip-planning through the web. An important step in preparing requirements is to identify what types of input are required from the user. Most common to transit planners, the primary inputs are the start (or origin) location and the destination location as well as the circumstances that should govern the resulting planning solution.

The provision of user interaction must be achieved by providing a geographical information map. The report also specifies the need for a user interface as a medium for collecting a user's input as well as communicating the resulting itinerary back to the user. The specification of such interactions makes it possible to provide this service through a standard web browser making it possible for a user with internet access to use the application.

Below are details regarding the scope of the project as well as the functional and non-functional requirements that are sourced and built upon from the guidelines highlighted above from Cherry et al. (2006).

2.4 Scope of Project

This project would consist of two main parts: firstly, a web-based client side that would accept users' inputs and then a server-side that would find the optimal route based on the user's inputs. The users' inputs would specify their origin and destination locations as well as a trip planning option (either least distance or least travel time). The client side would also display the optimum route returned by server with detailed step-by-step instructions for moving from the origin to the destination.

2.5 Functional Requirements

1. Accepting user input

- Users should be able to type out the name of their start and destination bus stops.
- Users should be able to specify trip planning options (least distance and least travel time)

2. Displaying result

- The user should be able to zoom and pan on the result displayed on the rendered map.
- The user should be able to view a map polyline corresponding to a single instruction of the returned result.

2.6 Non-Functional Requirements

1. Accepting user input

- The system must give predictive suggestions of the names of location as the user types.

2. Generating routes

- The system must generate at least one optimal route based on a user's query information.
- The system should be able to find transfer node(s) if the origin and destination do not share a common route.
- The system should be able to reduce computational difficulty by avoiding the exploration of routes that do not yield optimal results.

3. Displaying result

- The optimal route generated by the routing engine should be displayed on a map on the client side.
- The optimal route should be split into step-by-step instructions that the user can follow through to the destination.

Chapter 3: Architecture and Design

The steps outlined in the image below (Figure 3.1) depict the high-level architecture of this project's transit planning application.

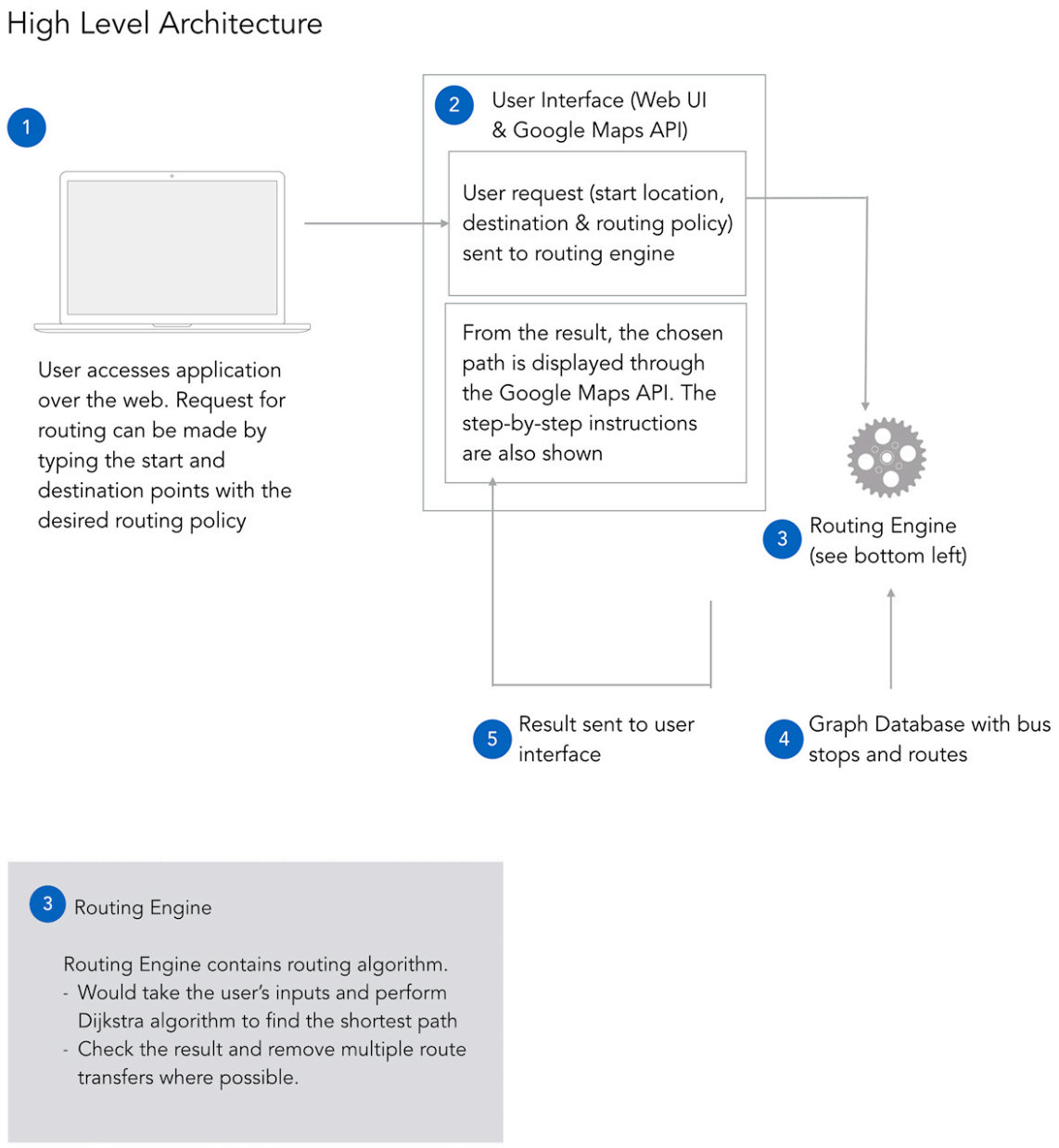

Figure 3.1 High-Level Architecture

3.1 Key Modules in the Architecture/Design

In Sun et al. (2011) an open and interoperable service-oriented architecture that makes use of existing software components was specified for a web-based trip planning system. The display and data manipulation model of the resulting application was built in a loosely coupled manner. From the above work, the main guideline for this application's architecture is to build each component individually and connect them in a manner that allows each component to be easily improved or replaced without affecting the entire system's functionality.

This tro-tro transit application has three key modules/components in the architecture. These are the database, routing engine and front-end web application.

1. Database

   The database containing information about the nodes and stops would be built using Neo4j, an open source graph database that takes advantage of not just data but its relationships when executing queries. Neo4j is optimized to provide constant real-time performance. The database entities can easily be modelled using graphs making it a viable to use neo4j as it would be faster than using a relational database. (Vukotic, Watt, Abedrabbo, Fox, & Partner, 2015)

   The GTFS data would be transformed into nodes and hosted on a webserver through Neo4j. Interaction with the database would be accessible through the database's REST (Representational State Transfer) API service provided by Neo4j. This API exposes the database through a range of HTTP (Hypertext Transfer Protocol) requests that elicit a JSON (JavaScript Object Notation) formatted response.

2. Routing Engine

The routing engine would be built in JavaScript. The routing engine would make use of a Dijkstra's algorithm procedure already available in the database application in finding the desired route per the user's query information. The routing engine would do this through the neo4j RESTful API. The routing engine would extract useful information from the resulting JSON response, making them available to the front-end web application.

3. Front-end Web application

The front-end of the application would be built with web technologies including HTML, CSS and JavaScript. Also, the Google Maps API would be used in visualizing both the start and stop points provided by the user as well the resulting path from the routing engine. The unique key for using the Google Maps API has been generated for this application. A function would be written that would parse the user's query to the routing engine. Another function would be written that would take the JSON result from the routing engine and render it on the map along with the given step-by-step instructions.

Chapter 4: Implementation

4.1 Overview

This section describes in detail the processes involved in implementing the tro-tro transit planner application. The chapter is divided into three major subsections that deal with the implementation of the database, the routing engine and the front-end application. For each software component, the libraries (if any) are described, useful program routines are also highlighted and screenshots are provided to give a rich detail of the overall implementation.

4.2 Database Implementation

4.2.1 The Data

The GTFS data used for this project was provided by the Meltwater Entrepreneurial School of Technology (MEST) in partnership with the Accra Metropolitan Assembly (AMA) and the French Agency for Development (AFD) through a hackathon dubbed the Accra TroTro Apps Challenge on May 27th and 28th, 2016. The hackathon saw over 100 hackers present and 7 schools represented at the event. (Baffoe, 2017)

The data used was available online prior to the event and consisted of over 300 active routes and over 2500 stops. The GTFS data is presented as shapefiles and comma separated records stored in text files. The descriptions for each text (.txt) file included in the data is given below (Table 4.1).

Table 4.1 Descriptions for GTFS files ("Overview | Static Transit | Google Developers", 2017)

File Name	Description	GTFS Specifications	Accra Data
agency.txt	Information on different transit operators	Required	Available
calendar.txt	List of days included in assignment period	Required	Available
routes.txt	List of bus lines	Required	Available
stop_times.txt	Arrival times at bus stops during data collection period	Required	Available
stops.txt	List of bus stops	Required	Available
trips.txt	List of trips	Required	Available
shapes.txt	Rules for drawing the linear shape of a bus line itinerary on a map	Optional	Available
fare_attributes.txt	Fare information.	Optional	Available
fare_rules.txt	Rules for applying fare information to tro-tro routes.	Optional	Available

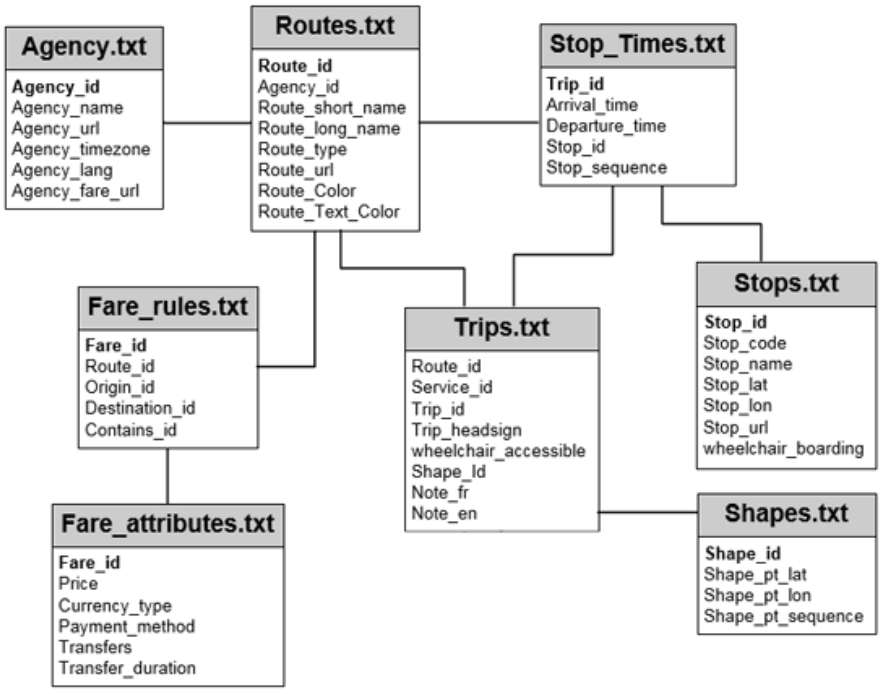

Figure 4.1 Schema for how GTFS files are linked ("Available data description", 2017)

4.2.2 Preparing the data

The goal in building the database was to get all the stops as nodes and routes between these nodes as edges or connections to build the graph in neo4j.

The nodes were extracted from the stops.txt file into 'stops-no-dp.csv' with the columns ***stop_id*** – the stop's unique identification number, ***stop_lat*** – the latitude coordinate for the stop, ***stop_lon*** – the stop's longitude coordinate and ***stop_name*** – the name of the stop. A java console application 'stopAndInOutRoutes.java' was written to create two files 'incoming.csv' and 'outgoing.csv', both containing key-value pairs (***stop_id, incoming route***) and (***stop_id, outgoing route***) respectively. These files show each stop and routes arriving at and leaving from it so that each node has this data included in its properties when the database is being created.

The steps below were used in generating the edges between stops that would represent the routes in the networks:

1. A script 'edgeScript.java' was written to extract pairs of nodes that were connected in each route along with the route name. The resulting file was named 'edgelist-with-tripname.csv' having the columns ***trip, stopA, stopB***.
2. The 'edgelist-with-tripname.csv' file was expanded to include the latitude and longitude coordinates for both stops. The resulting file was converted to JSON file named 'output.json' using a Node.js library called **'csv-to-json'.** (Kam, 2017)
3. A script 'edgeweight(Distance).js' was written to compute the distance (in meters) and travel time (in seconds) for each edge in the 'output.json' file using the Google Maps Distance Matrix API (Google, 2017). This API enables querying Google Maps' database for the road distance and travel time between two locations given

their latitude/longitude coordinates. The output from this step was stored in 'output(with weights).json'.

4. The file 'output(with weights).json' was converted to a comma-separated file using an online tool (json-csv.com, 2012). The resulting file was named 'final-edgelist.csv' only maintaining the columns ***trip, stopA, stopB, distance, time.***

4.2.3 Building the Database

As highlighted in Chapter 3, the data is better adapted to fit a graph database than a relational database. Therefore, to build the graph database, Neo4j, an open source graph database solution was used. Neo4j allows one to set up a graph database using a query language known as Cypher. By creating nodes and connecting them through edges, networks or graphs can be built. The database can be visualized using the Neo4j web interface. Neo4j also has a built-in REST API that allows other applications to communicate with it through queries and algorithms with it over HTTP requests. (Neo4j, 2017)

Neo4j was installed on standalone laptop computer running the Mac OS X operating system. A new database instance was created and the file 'stops.csv' was imported. In the import process, each line in the file was split into its component columns and used in creating a database object named 'Stop'. All stops have the properties ***stop_lat, stop_long, id*** and ***stop_name*** which were assigned their respective values from the column in the csv file.

A separate import statement was used to load the 'final-edgelist.csv' file into the database where two stops with ***id***s present in each record of the file were identified for

a connection named ‘IS_CONNECTED_TO’ to be created with the ***route***, ***distance*** and ***time*** columns used as properties for the edge connection.

Two further imports for the files ‘incoming.csv’ and ‘outgoing.csv’ were made to assign each node their respective incoming and outgoing route properties (***in_route*** & ***out_route***). With all this set up, the database was checked for connections that were redundant (i.e. between a given node and itself). These were removed. Below (Figure 4.2) is diagram showing a section of the database.

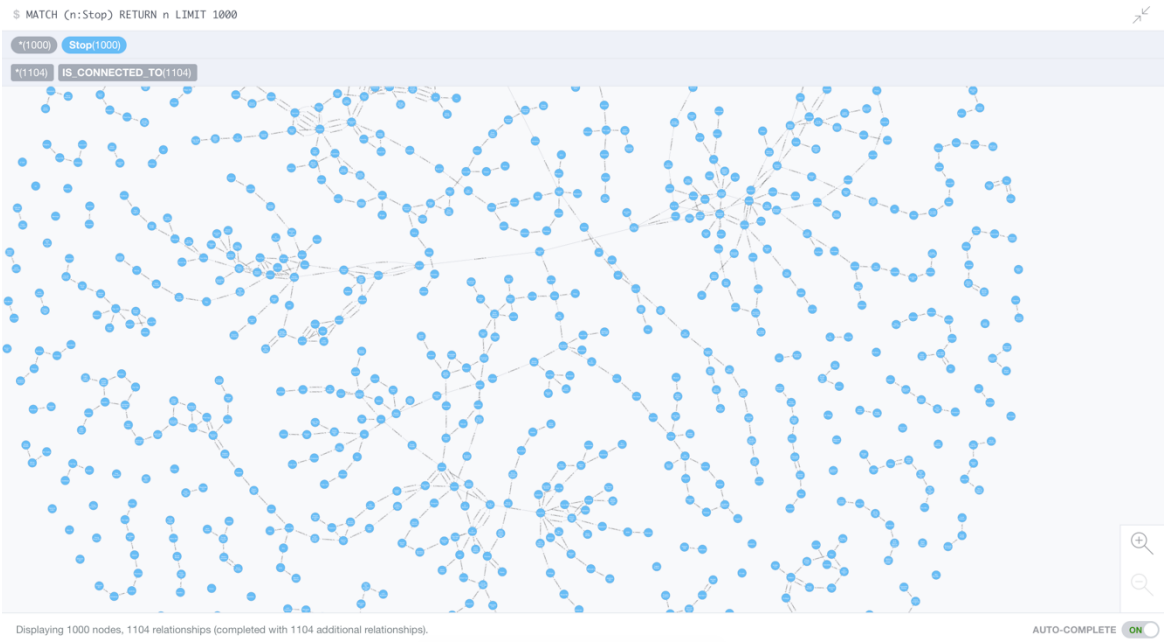

Figure 4.2 Screenshot of Database showing some nodes and connections.

4.3 Routing Engine

The routing engine was built entirely in JavaScript. The engine is made up of functions capable of making HTTP requests to the database server. A function ***getNode()*** was written to return a node object given an id as the parameter. Another function ***getNodeUsingMetaID()*** was written to return a node object based on the metadata id that is assigned by the database application during its creation.

The routing engine as highlighted in the high-level architecture is supposed to perform Dijkstra's algorithm in finding the shortest path between the supplied start and destination locations. To achieve this, a function ***dijkstra()*** was written to prepare and execute the HTTP request that would use the database's built-in Dijkstra's algorithm to compute the shortest path between two stops that are supplied as parameters. This request can compute the shortest path using the distance or time as the cost property.

Another method ***getShortestPath()*** was written to implement an algorithm for how the shortest path would be computed between two stops. This function made use of the ***dijkstra()*** function in its operation. The algorithm can be seen below.

ALGORITHM

1. If start stop = destination stop, no routing necessary. Return either the start or stop location
2. If start and destination are on the same tro-tro route(s) using the ***findSharedRoute()*** function, find the route(s). Return the itinerary that optimizes the routing criteria.
3. If start and destination do not share the same route, execute ***dijkstra()*** function using start and destination as parameters. With the resulting path:
   i. Starting with the destination node and ending with the start node search through the rest of the result for nodes that are on the same route and eliminate intermediary nodes between them.
   ii. Prepare a shortened result ('shortenedResult') if any intermediaries have been eliminated.
   iii. With the result (or 'shorternedResult') compute the total journey distance or journey time
   iv. Format the result in an html form and return this as a string.

Figure 4.3 Algorithm for computing shortest path between two stops

When a user supplies a request for transit planning, the routing engine builds an array of start points based on the user's supplied start point by mapping all stops that have matching names with it. The same is done for destination point. With these two arrays, multiple start/destination pairs are generated and ***getShortestPath()*** is run on all of them. The routing engine reviews all the results and selects the one that optimizes the routing policy specified. For the least distance policy, the cost when performing Dijkstra's algorithm is the distance and for the least travel time, the cost is the time property of the 'IS_CONNECTED_TO' relationship between nodes.

4.3 Front-end Web Application Implementation

4.3.1 User Interface Design

The front-end web application is the primary interface through which users of this application would get their planned paths. This calls for a user-centered approach in the design of the application. From Cherry et. al. (2006) and the software requirements highlighted earlier in Chapter 2, the application should provide a medium for entering the start and destination locations as well as a map-based interface for interaction with the generated path.

These two major requirements form the basis for the interface's design where the screen is split into a sidebar and a rendered map area as shown below (Figure 4.4)

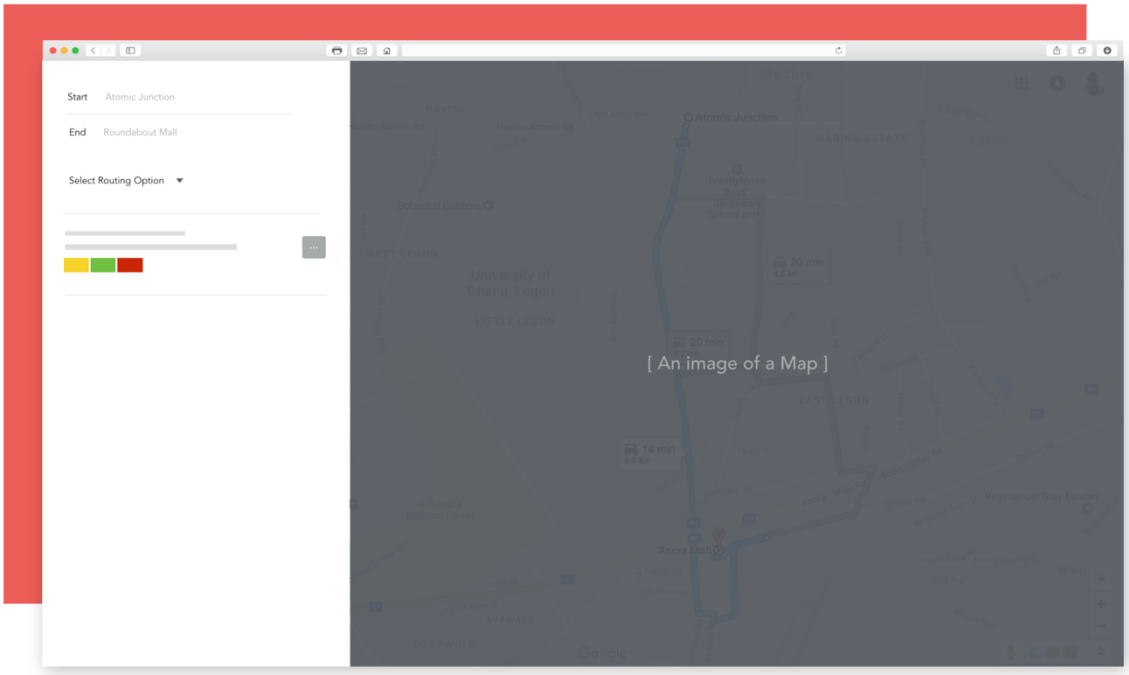

Figure 4.4 User Interface for routing application

The sidebar provides two inputs for the start and destination locations respectively and a dropdown menu for choosing the preferred routing criteria. The portion of the sidebar beneath these input fields serves as an area for the displaying the returned result as a summary and also the results as broken down step-by-step instructions. (shown below in Figure 4.5)

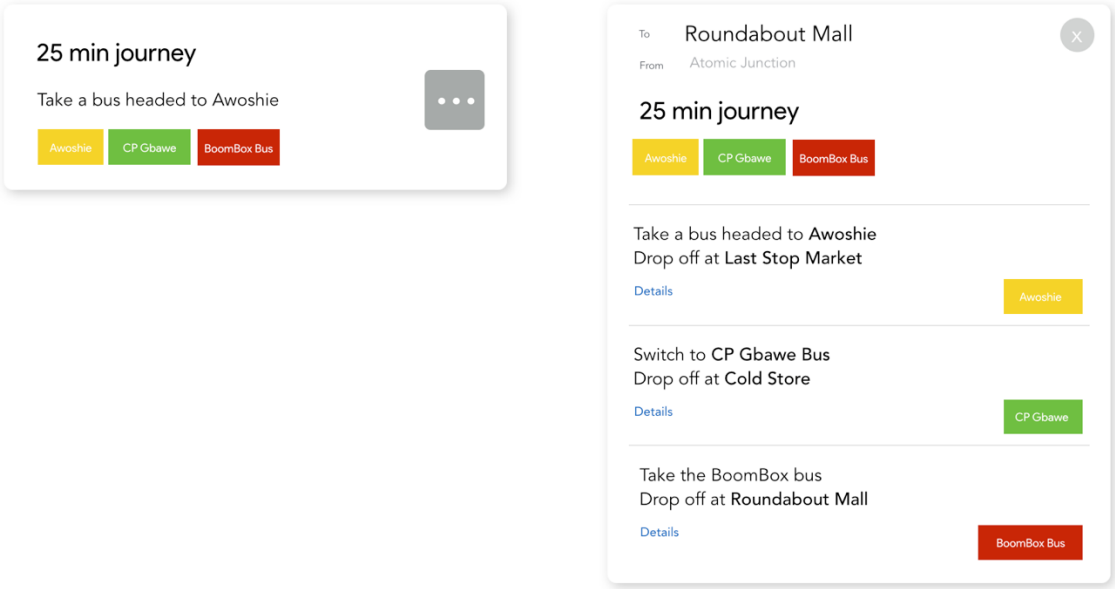

Figure 4.5 Left Image: showing result in a summary form. Right Image: showing result in the broken-down step-by-step-instructions.

The summarized result contains the total cost of the journey, the first action and a summary of all routes used on the trip. These routes are colored based on the order in which they appear.

4.3.2 Development

The development of the front-end web application was done using HTML, CSS and JavaScript technologies. HTML is the markup language that enabled the creation of the webpage elements. CSS (Cascading Style Sheets) was used to apply styles to the HTML elements. JavaScript was used to define the interaction of elements on the page as well as sending and receiving information from the routing engine.

To speed up the development process, Bootstrap (Otto, 2017), an HTML, CSS and JavaScript framework was used to define the basic input elements (input boxes and

dropdown list). From these elements, custom CSS and JavaScript scripts were written to provide the appropriate styling for the interface to look exactly as they had been specified in the user interface and experience design (see Chapter 4.4).

A button with the label ‘GO’ was built into the user interface to be used as a trigger for the transit planning procedure. This button only appears when the right inputs are supplied.

The rendered map area was built using a Google Maps API instance. The map was programmed to initialize Accra as its center anytime the application is accessed. When a request is made for transit planning, the results returned from the routing engine include the latitude/longitude coordinates of all stops in the result. These coordinates are plotted on the map area using a direction service in the API. This service allows one to plot a road route between two points while specifying attributes such as waypoints. The start is labelled ‘A’ and the destination is depicted with a red marker. All intermediary stops (if any) are stored in the waypoints of the plotted route and are labelled in the order in which they occur starting with the letter ‘B’

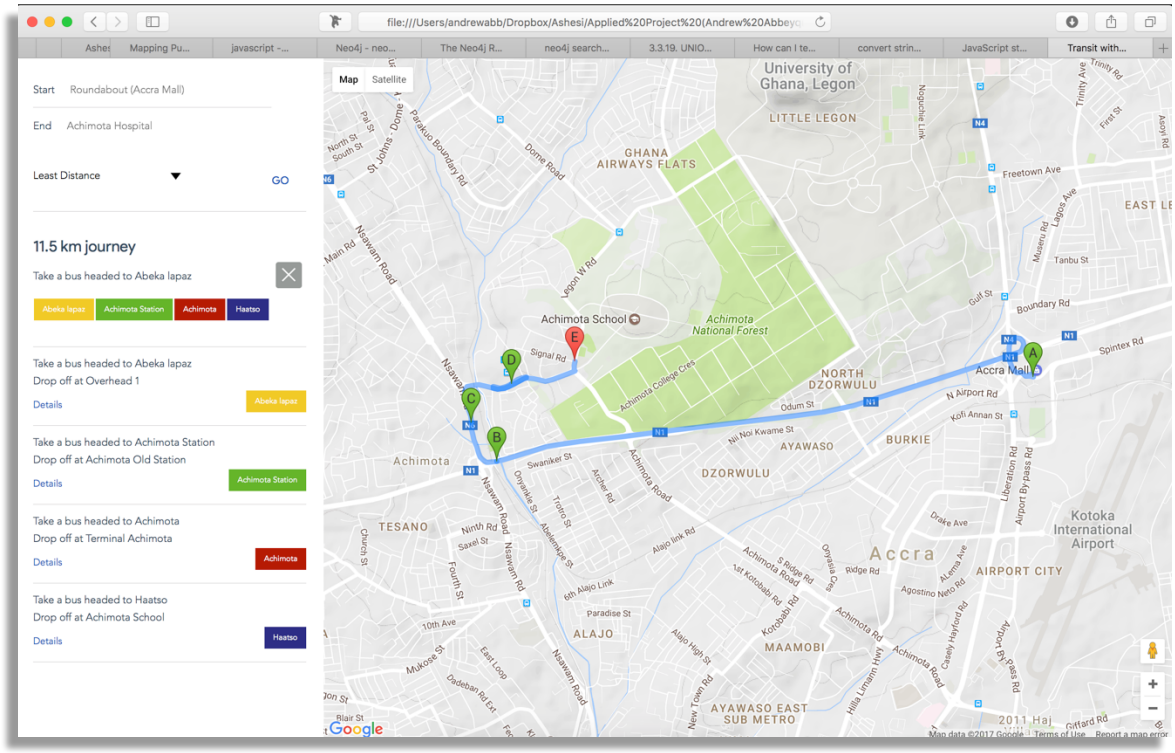

Figure 4.6 Screenshot of Application showing front-end interface after transit itinerary has been returned.

Chapter 5: Testing

This chapter describes how the application was tested to ascertain that it had met the requirement specifications.

5.1 Test Description

Most users would use this application through the front-end in getting their transit itinerary. For this reason, the testing is split into two: one dealing with the User Interface (Front-end) where various categories of inputs are supplied to the system and the other test of the system dealing with the quality of results returned by the routing engine when compared to the transit itinerary users are already accustomed to.

5.2 User Interface (Front End) Testing

The basic interaction with the application occurs through the input fields. As such, a variety of test values would be supplied and the results observed to see how best the program works. The following are test scenarios for the front end:

1. Routing without providing any input
2. Routing with wrong inputs
3. Routing with right inputs

For 1 and 2, the front-end application is built such that the ‘GO’ button which triggers the routing procedure is only visible when all inputs are verified and supplied to their respective fields. Also, inputs are selected from a dropdown list of autocompleted bus stop names such that only verified and predefined locations are permissible.

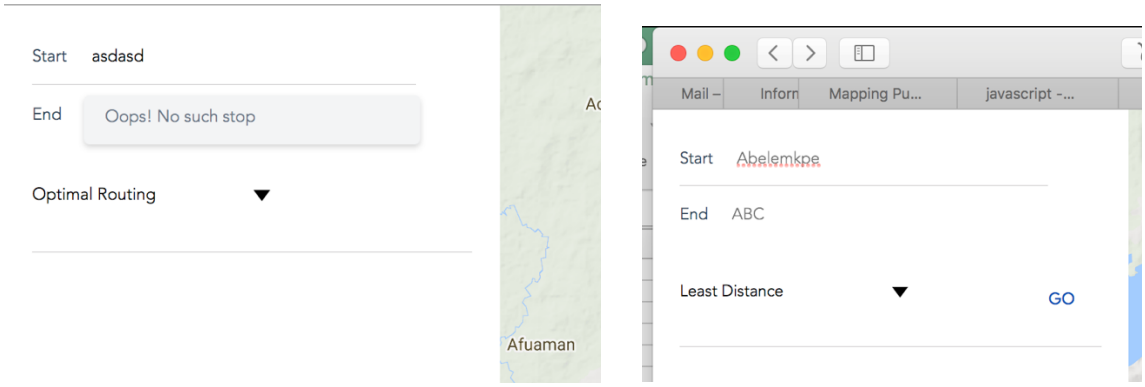

Figure 5.1 and 5.2 showing that the 'GO' button only appears when correct stop locations are entered.

For 3, when correct bus stop names were entered into the input fields, the 'GO' button became visible allowing the user to issue the request for routing to begin. (See Figure 5.2)

5.3 System Testing

From the user interface testing, the results showed that the transit planning procedure can only be triggered via the 'GO' button when valid bus stops are supplied as inputs. For the system testing, valid bus stop names were supplied to the application and the results checked against what a user would normally use when travelling that same journey. This meant that the tests could only be carried out by people who had travelled the same journey they were testing using tro-tro in a real-life situation.

A questionnaire (as indicated in Appendix A) was prepared to first introduce the project to users and then offer them instructions as to how they would fill it out. This questionnaire required respondents to enter a start and destination bus stop and then judge based on the results returned and their tro-tro travel experience how easy it was to find the

correct names for the start and destination points they wanted to use, if the results returned were different from what they would normally use and if so, how the results differed.

The questionnaires were distributed to 31 Ashesi University students. These students were selected if they knew how to make a single journey using tro-tros within Accra. The students were supplied with a laptop with the application already loaded into a browser. The students performed their tests before answering the questions.

5.4 System Test Results

As shown in the previous section, users judged the application based of 3 criteria namely:

1. How easy or difficult it was finding the correct name for the start and destination bus stops they wanted to use
2. If the returned result differed from what respondents would normally use
3. How results differed from respondents normally used.

Responding to criterion 1, 55% or respondents said it was either easy or too easy finding the names of the start and destination bus stops that they wanted to use. 19% found this task difficult whereas 26% concluded that it was a task of moderate difficulty. No respondent found it “too difficult”.

Table 5.1: Table showing how difficult it was for respondents to find the correct start and destination stop names.

	Q1 Options	No of Respondents	Percentage
1	Very Difficult	0	0%
2	Difficult	6	19%
3	Moderate	8	26%
4	Easy	13	42%
5	Too easy	4	13%
		31	100%

From the responses given to the question 2, over 77% of respondents indicated that the returned result was different from what they would have used. The main difference (accounting for 83% of varying returned results) that these respondents identified was that the returned result was longer than what they would have used. 17% found the difference between their choice itinerary and the returned result as not being significant whereas no respondent found the result as being shorter than their preferred itinerary.

Table 5.2: Showing the number and percentage of respondents who found the returned results being different from what they would normally use.

	Q2 Options	No of Respondents	Percentage
1	Yes	24	77%
2	No	7	23%
		31	100%

Table 5.3 Showing the number and percentage of respondents as well as the corresponding differences they found between the returned result and the itinerary they would have used.

	Q3 Options	No of Respondents	Percentage
1	Longer	20	83%
2	Not so different	4	17%
3	Shorter	0	0%
		24	100%

5.5 Test Conclusions

From the above results, it is evident that although the application succeeds in returning a tro-tro transit itinerary when given any two stops, it does not necessarily succeed in providing a solution that is either similar or better than the itinerary that users would rather

use. There are various reasons that can be attributed to this phenomenon. These reasons can be divided into those that occur because of the current algorithm in place and those that can be attributed to the structure of data.

1. The Data

   The data includes multiple stops mapping to the same physical location. In preparing the database, duplicates stop objects (stops with the same id) were eliminated. However, from the results of the system tests, it turns out that some stops can have different ids but map to the same bus physical stop location (sometimes with their latitude/longitude values differing by a walking distance)

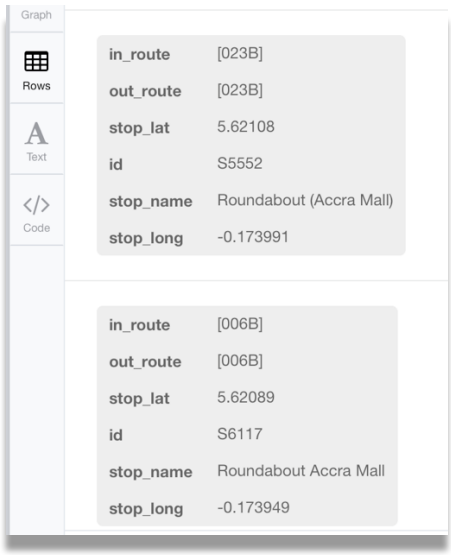

Figure 5.3 Showing two stops that map to the same physical bus stop location but have different stop ids and different stop names.

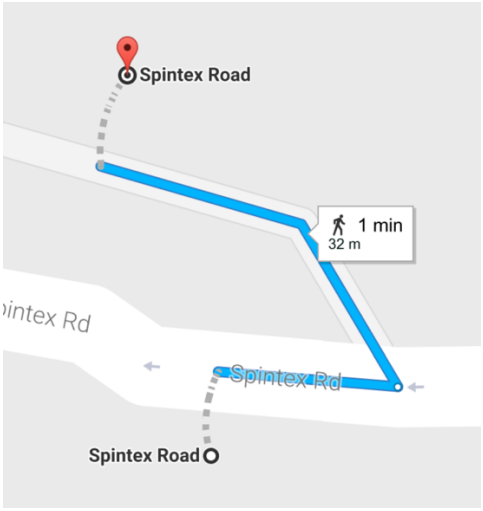

Figure 5.4 showing that the latitude/longitude values have a walking distance between them.

2. The Algorithm

The current algorithm performs the routing by using a combination of stops when finding the shortest path between two stops. When a user supplies a start point, the routing engine finds all other stops with the exact name. The same happens for the destination. When these array of start points and destination points are found, the algorithm performs various shortest path simulations using multiple pairs of start and stop points till the routing policy is optimized. This becomes the solution.

With this arrangement, certain stops in the database that point to the same physical location for a bus stop or terminal may be omitted in the array of start or destination points if their names do not match what's supplied in the input field. As a result of the algorithm's current setup, a user may receive a Dijkstra solution that is different from what they would normally use because one vital stop was not included in the array of start or stop points. (See Figure 5.3)

Chapter 6: Conclusion and Recommendations

The user application built in this project illustrates how technology can be used in bridging the information gaps within our transportation system. With this application, one can obtain the tro-tro transit itinerary between one's start and destination locations. Also, one can get step-by-step instruction for how to undertake one's journey from the start point to the destination.

However, as show in the test results, these results are often longer to what tro-tro users currently use. This presents a creative challenge where this application can be improved to provide user satisfaction.

6.1 Future Work

This project lays the foundation for exciting improvements to be made in understanding our tro-tro system. Some ways this can happen is through:

1. Using proximity to determine the appropriate start and stop points that optimize a chosen routing policy. As highlighted in test results, using the stop id or stop name to find an array of start/destination bus stops per the user's input may not be exhaustive as stops mapping to the same location may be named differently. The algorithm can be extended to use how close other stops are from the user's input before deciding which of them optimizes the routing policy. This improvement would introduce walking distances into the transit itinerary.
2. Integrating live traffic feed in route calculations. The current travel time associated each route does not represent real time information. Since travel time is often a function of what time the journey is being undertaken, it would be more efficient to introduce live traffic information when calculation the travel time for a journey.

3. Enable users to use GPS in their navigation along the returned routes. The current system can be extended such that users can have a feedback of where exactly they are in their journey while following the transit itinerary.
4. Integrating fare information to make cost effective path planning possible. The routing policies may be extended to include fares such that users would be able to make their journey in most cost-effective manner.

References:

Accra TroTro Apps Challenge - the Datahub. (2016). Datahub.io. Retrieved 16 April 2017, from https://datahub.io/dataset/accra-trotro-apps-challenge

Available data description. (2017). Stm.info. Retrieved 24 March 2017, from http://www.stm.info/en/about/developers/available-data-description

Baffoe, A. (2017). The TroTro Apps Challenge – MEST, May 27-28 -. Meltwater.org. Retrieved 24 March 2017, from http://meltwater.org/the-trotro-apps-challenge-mest-may-27-28/

Cherry, C., Hickman, M., & Garg, A. (2006). Design of a Map-Based Transit Itinerary Planner. *Journal of Public Transportation*, *9*(2), 45-68. doi:10.5038/2375-0901.9.2.3

Ghana Statistical Service. (2013). *Ghana - Transport Indicator Database Survey 2012, Second Round.* Retrieved from http://www.statsghana.gov.gh/nada/index.php/catalog/82

Google. (2017). *Transit-Google Maps.* Retrieved March 21, 2017 from https://maps.google.com/landing/transit/index.html

Google. (2017). *Google Maps Distance Matrix API | Google Developers*. Retrieved from https://developers.google.com/maps/documentation/distance-matrix/

json-csv.com. (2012). *JSON to CSV Online Converter.* Retrieved from https://json-csv.com

Kam, A. (2017). csv-to-json. npm. Retrieved 24 March 2017, from https://www.npmjs.com/package/csv-to-json

McHugh, B. (2013). Pioneering open data standards: The GTFS story. *Beyond transparency: open data and the future of civic innovation*, 125-135.

Metro Mass Transit Ltd (MMT). (2013). The establishment of MMT. Retrieved from http://www.metromass.com/mmt/establish_mmt.htm

Neo4j. (2017). *Neo4j Graph Database: Unlock the Value of Data Relationships.* Retrieved from https://neo4j.com/product/

Otto, M. (2017). *Bootstrap · The world's most popular mobile-first and responsive front-end framework*. *Getbootstrap.com*. Retrieved 17 April 2017, from http://getbootstrap.com

Overview | Static Transit | Google Developers. (2017). Google Developers. Retrieved 24 March 2017, from https://developers.google.com/transit/gtfs/reference/

Sun, D., Peng, Z. R., Shan, X., Chen, W., & Zeng, X. (2011). Development of web-based transit trip-planning system based on service-oriented architecture. *Transportation Research Record: Journal of the Transportation Research Board*, (2217), 87-94.

Vukotic, A., Watt, N., Abedrabbo, T., Fox, D., & Partner, J. (2015). Neo4j in action. Manning.

Williams, S., White, A., Waiganjo, P., Orwa, D., & Klopp, J. (2015). The digital matatu project: Using cell phones to create an open source data for Nairobi's semi-formal bus system. *Journal of Transport Geography*, *49*, 39-51.

World Bank. (2010). City of Accra, Ghana consultative citizens' report card. Retrieved from http://documents.worldbank. org/curated/en/540521468249314253/ City-of-Accra-Ghana-consultative-citizens-report-card

Appendix

Appendix A

Hello!

Tro-tro is a very popular transportation service, catering to 70% of Accra's commuters according to a World Bank study in 2010. Using this service requires a lot of knowledge about bus stops and the routes buses ply. For those who don't know them, the service is very difficult to use. Also, even if one knew which buses to take, there is no immediate way of asserting that one's travel plan is optimal in reducing one's travel time or travel distance.

The application you're about to use seeks to bridge the above informational gaps. The application takes a start and end destination bus stops within Accra as well as a routing criteria and returns a tro-tro itinerary with step-by-step instructions for undertaking the journey.

This is a questionnaire that would aid in comparing the results of the tro-tro transit planning application to the plans people have intuitively used as a useful test of the application.

STEPS

1. Select a start and end destination stop for a journey you already know how to make using tro-tros.
2. Select routing criteria
3. Compare the results from the application to what you would normally do by answering the questions on the right.

Start

End

Tick one for each question

1. From difficult to easy, rate how it was finding the stops

- O Very Difficult
- O Difficult
- O Moderate
- O Easy
- O Too easy

2. Is the results returned different from what you would normally use?

- O Yes
- O No

3. If it was, describe the results using one of the responses below

- O Longer than what I normally use
- O Not so different from what I normally use
- O Shorter than what I normally use