Table of Contents

[1. Introduction - 3 -](#_Toc482145537)

[1.1 Problem drivers - 3 -](#_Toc482145538)

[1.1.1 Problem to be solved - 3 -](#_Toc482145539)

[1.1.2 Project Purpose - 3 -](#_Toc482145540)

[1.1.3 Project Objectives - 4 -](#_Toc482145541)

[1.1.4 Limiting the Project Scope - 4 -](#_Toc482145542)

[1.1.5 Outline of plan - 4 -](#_Toc482145543)

[1.2 Output Summary - 5 -](#_Toc482145544)

[1.2.1 Requirement Specification Document - 5 -](#_Toc482145545)

[1.2.2 Design Specification Document - 5 -](#_Toc482145546)

[1.2.3 Visualisation platform - 5 -](#_Toc482145547)

[2. Literature Review - 6 -](#_Toc482145548)

[2.1 Initial Development - 6 -](#_Toc482145549)

[2.2 Software Visualisation - 6 -](#_Toc482145550)

[2.3 Java Data Structures - 7 -](#_Toc482145551)

[2.4 JavaScript - 7 -](#_Toc482145552)

[3. Methodology - 8 -](#_Toc482145553)

[3.1 Software Development Model - 8 -](#_Toc482145554)

[3.2 Initial Planning - 8 -](#_Toc482145555)

[3.3 Information gathering and Requirement analysis - 9 -](#_Toc482145556)

[3.3.1 Interview with the client - 9 -](#_Toc482145557)

[3.3.2 Requirement specification - 9 -](#_Toc482145558)

[4. Implementation - 10 -](#_Toc482145559)

[4.1 Iteration 1 - 10 -](#_Toc482145560)

[4.1.1 Design Visualisation - 10 -](#_Toc482145561)

[4.1.2 Implementation of the code - 11 -](#_Toc482145562)

[4.1.3 Testing - 11 -](#_Toc482145563)

[4.1.4 Stakeholder feedback - 11 -](#_Toc482145564)

[4.2 Iteration 2 - 12 -](#_Toc482145565)

[4.2.1 Design - 12 -](#_Toc482145566)

[4.2.2 Improve business logic - 12 -](#_Toc482145567)

[4.2.3 Testing - 15 -](#_Toc482145568)

[4.2.4 Stakeholder feedback - 16 -](#_Toc482145569)

[4.3 Iteration 3 - 16 -](#_Toc482145570)

[4.3.1 Design Visualisation - 16 -](#_Toc482145571)

[4.3.2 Improve business logic - 16 -](#_Toc482145572)

[4.3.3 Testing - 16 -](#_Toc482145573)

[4.3 .4 Stakeholder feedback - 16 -](#_Toc482145574)

[5. Results - 16 -](#_Toc482145575)

[5.1 Iterations Results - 17 -](#_Toc482145576)

[5.1.1 Iteration 1 - 17 -](#_Toc482145577)

[5.1.2 Iteration 2 - 21 -](#_Toc482145578)

[5.1.3 Iteration 3 - 21 -](#_Toc482145579)

[5.2 Alternative Results - 21 -](#_Toc482145580)

[5.4 Evaluation - 21 -](#_Toc482145581)

[6. Conclusion and discussion - 21 -](#_Toc482145582)

[6.1 Future work - 22 -](#_Toc482145583)

[6.2 Personal Review and what the author would do differently - 22 -](#_Toc482145584)

[7. References - 23 -](#_Toc482145585)

[8. Appendices - 24 -](#_Toc482145586)

[8.1 Appendix A - 24 -](#_Toc482145587)

[8.1.1 Project Planning - 24 -](#_Toc482145588)

[8.1.2 Aim & Scope - 24 -](#_Toc482145589)

[8.1.3 Stakeholders & Interest - 24 -](#_Toc482145590)

[8.1.4 Requirements - 25 -](#_Toc482145591)

[8.1.5 Risk Assessment - 25 -](#_Toc482145592)

[8.1.6 Project Risks - 25 -](#_Toc482145593)

[8.1.7 Methodology - 26 -](#_Toc482145594)

[8.1.8 Gantt Chart - 26 -](#_Toc482145595)

[Modified Gant Chart - 28 -](#_Toc482145596)

[8.1.9 Ethics form - 29 -](#_Toc482145597)

[8.2 Appendix B - 34 -](#_Toc482145598)

[8.2.1 Java Code Structure - 34 -](#_Toc482145599)

# 1. Introduction

## 1.1 Problem drivers

Organizations using Business Intelligence, use raw data to extract significant information to make decisions, spot business problems and boost their performance. The problem arises when the information is inaccurate and it lacks quality, leading to decrease efficiency in business and production. For such cases, a model should be carefully designed using validation techniques to ensure that the output will meet the appropriate criteria for each case study (Asadallah Najafi, 2011 ). Firstly, the conceptual model must be validated by gathering information from the stakeholders and address the main problem. Then a computerised model is built to implement a solution for the problem. There might be cases when the model’s output behaviour is accurate but there will be times when the model is tested with different input it will produce inaccurate results. The problem is not easy to understand and the results might be misleading. This causes confusion and requires the developer’s team to test the code many times until the main problem is spotted.

### 1.1.1 Problem to be solved

Knowledge engineering group (KEG) is a team of researchers, employed by the University of Brighton to work on large projects worldwide. The group consists a team of developers who were assigned a project to develop a business intelligence platform for a railway company. This project requires the use of raw data such as train timetables, stops and destinations. KEG’s analyst realised the rail company had many issues with their historic data as the trains had many delays or technical problems with their train fleet. A part of this project was to create a route finder application that will be hosted on the rail’s company website. The developers using Java, they have already built a model that computerised the routes using the data provided by the rail company. Unfortunately, their code was not efficient and it produced inaccurate outputs. What irritated them the most was the time they needed to spend debugging the code to identify the problem. Their need was to find a different and most efficient way to understand and identify their problem as the deadline for the project was pressing them.

### 1.1.2 Project Purpose

This is an industrial driven project which its main purpose is to develop a simulation model that illustrates clearly the whole implementation of the client’s code and visualise the whole process. By breaking down the code into different modules and visualising them in a different environment will give a clear understanding to the developers how their code executes using different datasets. The whole model should be re-programmed using JavaScript and D3.js. The code should be translated carefully to serve the main functionality as the original. The main target for this project is not to create a better version of the previous model but to develop a new environment that will visualise the workflow to help the client understand the execution. This will give a clear insight to the group’s developers to identify potential mistakes caused by the operational process or even more identify errors in the data. The new model will use these data for the input and it will produce a new output to be compared and tested with the correct results provided by the KEG’s staff.

### 1.1.3 Project Objectives

Objectives targeted as follows:

* Create an interactive and fully functional visualisation environment that receives a CSV file and produces a new dataset with the correct outputs
* Fully understand and translate Java code and data structures to JavaScript
* Understand the requirements and protocols provided by the KEG’s developers
* Visualise each component of code
* Test every module separately
* Test the system thoroughly and diminish any bugs

• Code the system in such a way so near future changes can be done effortlessly

### 1.1.4 Limiting the Project Scope

The scope of this project was to develop a visualisation environment to add data and illustrate the process until the expected output is extracted. In order to decide upon the functionality, the author researched extensively the available visualisation frameworks to find the best course of action. Using the data from the rail company along with the research done and the frequent meetings with the developer’s team, enable the author to construct a plan using an agile software engineering model. Moreover, by examining the Java code, the author was able to decide and choose more technical stuff like the data structures required to enable the same functionality of the original code, combined with the visualisation components.

### 1.1.5 Outline of plan

**Research:** The first course of action was to collect as much information as possible from the KEG developers to identify their main issue and understand their code. Then the author researched available visualisation frameworks and decided the most suitable for this project.

**Design:** After collecting all the necessary information and deciding upon an agile model, a plan was created to design the basic model of visualisation which included the flow chart of the code transformed into modules. Moreover, functional requirements were created for the system and discussed with the director of Knowledge engineering group until an agreement was reached. JavaFx was used for the creation of the visualisation environment but was replaced with D3.js later when the team decided to host the platform on a web browser.

**Implementation**: The implementation started as soon as the team decided to use JavaScript libraries. The first stages of the implementation were to convert the Java code to JavaScript and use queries to store the output to MySQL. Later, all CSV files were converted to JSON in order to ease the process and avoid relational database systems.

**Testing:** The testing phase consists of two different parts. Firstly, each node on the visualisation platform must be checked if it produces the same result as the original code. After making sure the operational process was functioning properly, train data were used to produce the routes for each train. Then it was compared with the KEG’s output to ensure both workflows were producing the same results. After all checks and tests are completed, KEG will be able to use the simulation model to identify mistakes and improve their code.

## 1.2 Output Summary

In this section of the project, the author will be talking about the major documents produced during the project.

### 1.2.1 Requirement Specification Document

Recipient: Knowledge Engineering group, author

File type : Microsoft Word Document

The first document that was created was the requirement specification document which was crucial in the development of the system since it was created to record and understand the client needs. Moreover, it can be used as a reference point for what the system needs to achieve.

This document contains the requirements specification document with a list of all the functional and non-functional requirements for the system approved by the director of KEG. These were developed after an extensive information gathering and analysis which included methods like interviewing the group members, using a questionnaire to identify their basic needs and understand the process taken to develop the original code for the route finder part of the system.

All the documents created can be found in **APPENDIX A**

### 1.2.2 Design Specification Document

Recipient: Knowledge Engineering group, author

File type : Microsoft Word Document

The second document created was to help even further with the development of the project and again act as a reference for what the system should look like.

This document contains draft designs of the visualisation as well as flow-chart diagrams that were used to structure the functionality of the code. In each iteration, the author created a draft diagram that can be found in the next sections. These were derived from the information gathered by the author and the Java code of the route finder.

all the documents, drafts, diagrams created can be found in **APPENDIX B**.

### 1.2.3 Visualisation platform

Recipient: Knowledge Engineering group

File type : .rar file

This is the finished product that will be used by the group as their visualisation environment to check their code efficiency and execution process in each step, filtered based on each row on the data set and each route created. This folder contains a repository with the raw data, a repository with the original code and the new code written in JavaScript and a repository containing all the files needed for the visualisation including the data in JSON format.

Some of the outputs create are shown on section 5.

# 2. Literature Review

This section will detail what books, articles and resources the author had to read in order to be able to undertake this project. This section has been broken up to into four main categories which include Initial Development, Software Visualisation Java data structures and JavaScript

## 2.1 Initial Development

Project planning and control were one of the modules the author attended during his university studies, therefore after researching development models online and reading books the author decided to work on the Agile model. To do so the user reviewed all the module materials to find more information about development models and their advantages. After reading the book “Software Engineering” (Sommerville, 2011) the author found three development models that all suited his needs and were straightforward to understand. The Waterfall model which was eventually rejected, is the more standard development model used were the developers need to split the project into separate phases considering the fundamental process activities of specification, development, validation and evolution (Sommerville, 2011). An alternative option would have been the increment model which is based on the idea of developing an initial implementation, exposing this user comment and evolving it through several versions until an adequate system has been developed (Sommervile, 2011). This was then considered both by the author and the client due to the fact that constant communication was feasible as both were living in Brighton and they could plan regular meetings. At the end both agreed to use the Agile model as the requirements of this project might change over the time. In contradiction with the waterfall model there was no need to have a strict plan as the development started early and by the regular meeting there was a feedback through the whole process. The quality of the software was improved using the extreme programming methodology by delivering a small version of the final product each week or month to the client and discussing changes or improvements.

## 2.2 Software Visualisation

In order to visualise the execution of the RoutesFinder application, the author researched articles regarding software visualisation. During his research, he came across a great paper called “Software Visualisation Today” (Matttila, 2016). Software visualization means visualizing various aspects and artifacts related to software (Matttila, 2016). To be able to visualise a software, one must study the structure, the behaviour and the evolution of the program. Many details can be extracted by understanding the software development process of an organization which it helps it to manage software projects. The benefits of visualising a software gives the developers a clear understanding on how the software is used by visual means. Instead of reading and trying to understand an algorithm which manipulates large datasets, a visualisation of the system could be easier to human visual perception. This article, lead the author to ask the correct question to KEG’s developers in order to understand the structure of the current application. After understanding the data structure used to manipulate the data, the behaviour of the system was studied. The main operations were removing rows from the data and manipulating cells by changing their values. The interviews with the developers were not enough to start this project, leading the author spending time experimenting with the current system to fully understand the behaviours and its structure. Now that the main aspects of Software Visualisation were covered a tool was needed to implement the process.

While studying visualisation framework online the author considered d3.js as the most suitable tool for this project. D3 is a small library for great visualisations and it’s widely discussed as one of the best techniques to visualise almost anything. Like programming, d3 needs a lot of practice to understand its methods and get familiarised with it. The author during his research process, started spending time studying Mike’s Bostock website which is the author of d3 - Available at : <https://bost.ocks.org/mike/> (Accessed: 27/10/2016). After understanding the fundamentals of this library, the author researched the official website of d3 Available at: <https://d3js.org> (Accessed: 10/11/2016) which provided an amazing documentation for all types of visualisations and their code. By using the code provided by these websites and d3’s GitHub profile, the author experimented with many types of visualisations and decided to use the template provided by Mike Bostock on collapsible trees. By using this tree, the RoutesFinder application could be easily illustrated by breaking it down into modules and visualise them as tree nodes.

## 2.3 Java Data Structures

During the research and design process the author spent a lot of time interviewing KEG’s developers to understand their code. The data structures that were used for the Java code were Lists and ListIterator. Again, the author had previous experience with data structures form his 2nd-year module called Data Structures and Algorithms. To refresh his memory and make the most sensible decision the author spend time reading and experimenting with the Oracle’s documentation about the ListIterator interface and its methods. All the data in the Java code were manipulated using the methods from that interface, such as hasNext(), hasPrevious(), next() and previous(). After understanding these methods, the author tested the code by printing the results on the console to understand clearly their functionality.

## 2.4 JavaScript

The decision to choose d3.js to implement the visualisation for this project led the author to choose JavaScript as the programming language for translating the Java code. D3 is a library written in JavaScript so it will be much easier to combine the logic of the code with the visualisation environment. Because Java is different from JavaScript, there was a risk that the functionality of the system could not meet the requirements. To ensure this problem would not exist, the author researched ways to implement a class in JavaScript having the same functionality as the ListIterator interface. Although the Mozilla developers network provided a documentation for list iterators their methods are not suitable for the requirements of this project. By reading posts in StackOverflow, there were many developers that have recreated this interface using plain JavaScript. After the long discussion the author had with the KEG’s director, they decided to not use this method, instead, the author should create his own way to manipulate data in the list. By reading articles on StackOverflow on how to manipulate data inside lists or arrays the author concluded to use a simple for loop and create variables that will decide which objects in the list will be considered as current, previous and next.

# 3. Methodology

This chapter will describe the methods used and work done by the author over the course of the project to achieve its purpose.

## 3.1 Software Development Model

|  |
| --- |
| *Figure 1: Agile cycle (Sommerville, 2011)* |

The software development model that was used by the author is the Agile model. It is a model that satisfies the stakeholders through early and continuous delivery of valuable software. It welcomes changing requirements, even late in development. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale. (Mike Beedle, 2002). Flexible planning, fast implementation and early product delivery are what defines scrum as agile, repeated in n number of iterations. For example, the author decided to split this project into 4 iterations which each consist of 6 phases: Requirement analysis, Planning, Design, implementation, Testing and stakeholder’s review. This model was ideal for this project as the main problem of the stakeholders was to visualise their code and the end-goal was not clearly defined. The requirements where changing during this process as the members of KEG could decide a new or modify existing features on the visualisation environment. The great communication between the author and the team was ideal for this model as both were living in the same city. One can ask “why not use the increment model then?”. The reason is that agile is both iterative and incremental. There is no need to follow any strict protocol or develop the features at first and then add functionalities to them in every iteration.

Due to the fact that this was the first full-scale individual project the author had to tackle, there was a need for a more “agile” methodology. In principle, the Agile model states that the focus is to develop the full functionality of the product from the beginning and have continued feedback by frequently releasing demos of the product. The review from the stakeholders in each iteration was a critical part of the development of the end-product, as an undergraduate was supported with the appropriate guidance from professionals.

This principle was followed up to a point, however, due to some particular factors like inexperience and changes in the requirements the project had to be adjusted. Therefore, in practice the iterations of the project, overlapped up to a point to provide the author with appropriate feedback, for example during implementation, problems are found and corrected or during the design phase some more requirements are identified and need to be included.

## 3.2 Initial Planning

Time is of the essence in a software development project so the first task the author did in preparation for the project was to develop a work plan. For a better representation of the time spent for each phase of the project, the author created a Gant Chart to allocate time for Requirements Analysis and Research and all iterations of the project. Furthermore, the plan included time taken off project during the examination period in January. However, the initial planning was not well thought and due to some other factors like the stakeholder’s slow response to provide the necessary data and miscalculating the time needed for translating the business logic to JavaScript, the initial work plan had to be adjusted. Some key changes included the re-allocation of every single End Date of the iteration. The first iteration was planned to start from the first week of December but the Java code and data were given in January so the new plan consisted of 3 iterations which they lasted 1 month each. The author spent that time to practice his visualisation code. Both the initial Gant Chart and the changes can be seen in **APPENDIX A, Project Definition Report**

## 3.3 Information gathering and Requirement analysis

This section will focus on the work done by the author to collect the necessary information for the development of the system requirements. The author had a respectable amount of knowledge with object-oriented languages, therefore the stages involving communication with the client were more focused on specific requirements the group had about the Java code they provided to the author, their interaction with the current route finder application as well as their needs to visualise the whole process. All the original documents between the interaction between the author and the client can be seen in **APPENDIX A**

### 3.3.1 Interview with the client

In order to gather more information about the current application and general requirements, the author had an interview with the director of Knowledge engineering group as well with the developers. The director expressed his need to create a basic visualisation platform that visualises the operational process of their code that creates the routes for the trains. Their main requirement was to represent the exact rules as a filter or an SQL command. He mentioned that their existing code is not accurate and it produces wrong outputs. Instead of rewriting a better version of the existing logic, it was necessary to transform this logic into a visualisation environment that will be functioning according to the business protocols according to the client’s case study. The client did not provide enough information about the exact end-product as the features will be discussed and changed between the iterations, emphasizing that the main priority was to visualise the process. The author was responsible for designing a new template that will be used as a simulation model of the current system.

After understanding the main requirements of the project, the author and developers had a long interview discussing the functionalities of the current route finder application. The first task was to understand the data and the expected output. The raw data was many CSV files containing information about the train unit numbers and where they stop and start. In addition, the data contained information unnecessary for the routes. For example, some trains might have technical issues, or they must stop to depot destinations for repairs. The route finder application was written in Java and its main functionality was to remove unnecessary rows from the data or add a new column, pointing which rows were the first or last destination of a specific route.

The developers provided the author with 5 files, the java code, 2 CSV files containing the raw data of 2 unit numbers and 2 CSV files containing their correct output. The first file was the Java code which stored the data into a ListIterator and then it was manipulated inside a long while loop containing many else if statements with the business logic. The CSV files with the correct output were given to the author to allow him to test the new system.

### 3.3.2 Requirement specification

The Requirements Specification was a document that was developed and updated frequently from the beginning of the first phase of the project. The requirements were separated into two sections; functional and non-functional. Functional requirements being statements of services the system should provide, how the system reacts to particular inputs, and how the system should behave in particular situations (Sommerville, 2011) and non-functional requirements being constraints on the services or functions offered by the system (Sommerville, 2011). A general description of the requirement is included along with the date the requirement was written as well as the Priority of the requirement for the system divided into three categories (High, Medium, Low). The full document with both functional and non-functional requirements can be found in **APPENDIX A**

# 4. Implementation

## 4.1 Iteration 1

After the author interviewed the developers and identified the key requirements of the project, this iteration was the most crucial as the first demo was expected at the end of the cycle. Four files were created for this part of the project.

* demo.js
* logic.js
* index.HTML
* style.css

Furthermore, all data received from the stakeholders were converted to JSON format to be accessible by JavaScript and d3.

### 4.1.1 Design Visualisation

|  |
| --- |
| Figure 2: Draft design (Author) |

The first draft was designed on paper displaying shapes that represent methods or filters. (Figure 2, Author). Because the operational procedure is sequential, each shape will contain a unique functionality and some of them will have dependencies. For example, the first shape of the visualisation is a square. This square represents the method that reads data and makes the appropriate checks if the data are valid and doesn’t contain any useless characters. The circles could represent some filters that will receive data from the square as a parameter and filter them. This model will illustrate different shapes as different kind of operations. The users will be able to click on any shape and see the actual data that are manipulated at each phase.

Although this model does not meet all the requirements and doesn’t fit with the business logic, the author wanted to design the first demo to combine the logic and visualisation as the first prototype of something bigger. If the end-product was objectified, the author would characterise it as a large engine with different types of gears. The iterations’ purpose was to build a small gear model that will be used afterward to create similar gears and combine them all together until the engine is completed.

Section 5.1.1 discusses the actual design that was delivered to the client

### 4.1.2 Implementation of the code

The implementation for this iteration contained 2 different stages. The first is the translation of the java file to JavaScript and the second is the visualisation process combined with the logic.

Starting with the translation, the author spent a lot of time reading the Java code to understand it’s structure. Firstly, KEG’s developers decided to store the data into a ListIterator of Strings, so each row of the dataset could be treated as a long string. All the columns were defined as variables in the global scope and with the help of an external class called FastTokenizer and the use of commas, the actual columns were extracted from each row. By using a while statement and the method hasNext(), the team was able to loop through the data and use the business logic as if statements to remove lines or to add a new column to assign the values of first or last. The challenge for the author was to recreate the same process described above into a similar structure in JavaScript to serve both the same functionality and support the visualisation environment of the new platform. After resolving the problem with the data structures for the business logic, the author broke down the Java code into modules and started rewriting the code to the logic.js file.

The full document with the structure of Java code can be found in **APPENDIX B**

Section 5.1.1 discusses the solution for this problem

For the visualisation, the author tried to visualise 2 procedures from the code, reading the data and filtering the data according to their destinations. The first shape will be a square and it will be responsible for reading the data. The second shape is responsible for filtering the data and removing all rows containing the word Depot. When the user clicks the shape, a table will be displayed showing the actual data.

### 4.1.3 Testing

To ensure that the first demo was functioning properly, test data were created by the author to test the procedure. The first demo was launched and the user was able to click on the shapes and display the appropriate data on a table attached to the DOM interface. By using the Firebug extension on Firefox, the author could print results to the console to examine the data before and after the filtering procedure. Moreover, all the operations in the logic.js were commented with a given name by the author; the console was used to test each module of the code by printing the results and the name of each operation to ensure they are returning correct outputs.

### 4.1.4 Stakeholder feedback

Due to the time needed to rewrite the code to JavaScript, the demo did not support many features for the visualisation part. The stakeholders were satisfied with the prototype but they were expecting more features. In addition, there was no need to use a different shape for each operation and at the end of the next iteration, it was expected to visualise the whole process.

## 4.2 Iteration 2

After the meeting with the stakeholders, the author had to make a fast plan to meet the new requirements. While researching d3.js, the author experimented with many visualisation examples and the most suitable for this case was the collapsible tree. All modules from the code could be represented by a simple circle. All dependencies could be illustrated by creating children to the appropriate nodes. Moreover, the business logic now must be changed to functioning with the visualisation environment.

### 4.2.1 Design

|  |
| --- |
| Figure 3: Daft design 2 (Author) |

By using the Java code structure in APPENDIX B, the author was able to rewrite the same structure into a JSON file. This file was used as the data for the collapsible tree. The end nodes of the tree will represent the business protocols which are remove a specific line and add the value of FIRST or LAST to a new column. All nodes’ names will be the given names the author provided the code using comments. The tree will start with all its nodes collapsed. The main idea is to develop a functionality to expand specific nodes while the code runs and display the results of the new and original data file in 2 different tables as can be seen in figure 3. The HTML file should be split in 2 parts. The left part is responsible for hosting the d3 tree and displaying the necessary variables, like the current ***i*** and the second part is to display the results as tables.

### 4.2.2 Improve business logic

In order to achieve the functionality mentioned above, the code should be separated into different files containing the necessary functionality to achieve modularity and improve the testing phase. After designing the second demo for this iteration the author decided to break down the code into 3 new files:

* tree.js
* table.js
* logic.js

**tree.js**

|  |
| --- |
| Figure 3: treeData.json example (Author) |

After splitting the HTML file into different parts, the author started developing the collapsible tree and experimenting with the DOM. The first step was to create new data that will be used to generate the tree. As mentioned before, by using the code structure that was created from the RoutesFinder application, the author created a new JSON file called treeData.json. The basic attributes of the data were the name, parent and children. Each object in the file will be represented as a tree node in the collapsible tree. The code will check if each node has children and it will link them together as children nodes using a diagonal in d3.js. During the implementation, the author decided to use a single variable to define the data as it will improve the performance of the application. To read the data from an external file, it is costly and decreases the complexity of the system as it must call the d3 class to find the data file and extract them to an array of objects. Instead by defining a new variable in the global scope, data can be accessed directly in any part of the application.

|  |
| --- |
| Figure 4: tree.js |

Following the example of the Collapsible tree code (Mike Bostock, 2010) the author had to add some new functions and modifications to fit and combine the business logic with the visualisation. Firstly, some variables had to be defined on the top level, like the dimension of the tree, the tree element from d3, the diagonal that will draw curved lines connecting the nodes and the SVG element that receives the container of the DOM for placing the new tree. All the elements of the tree (nodes and links) will be grouped together using the ***“g”*** attribute.Whenever the user interacts with the tree, or the logic wants to modify the structure of the tree, all elements must be updated. For that reason, a new function was defined with the name update which receives the data of the tree and calculates the new position for each node.

The main behaviour of the tree is to append and display all the circles and text as nodes in the DOM. In addition, the tree must collapse or expand specific nodes. To achieve this behaviour the update function should create specific variables for each process such as nodeEnter, nodeExit and nodeUpdate. Starting by defining the nodes and links, the function will automatically calculate how many elements there are in the tree data and attach them to the tree as a node or a link. NodeEnter will be responsible for appending a circle and its associated text in the “***g”*** attribute to display the nodes when the tree is created. NodeExit will be responsible for removing each element from the tree and nodeUpdate will transition the nodes whenever the tree is updated.

Every time the index.HTML is launched for the first time, the tree will automatically collapse all if its nodes. To achieve that, 2 functions should be created out of the update scope. These methods will receive a parameter as the node that the system wants to collapse and it will check if there are any children. For the collapse function, if the previous statement is true, the children will be stored in a temporary variable and then will be set to null. The same process will be repeated for any child that contains its own children. For the expand method the same process occurs but in a reverse way with the only difference, it must check where the children notes are being stored. These methods work because every time they are called they are changing the content of our data, so when the tree is updated it will not display the associated children nodes as the children variable was set to null. The expand method will be discussed later in this section.

|  |
| --- |
| Figure 5: tree.js (Author) |

**table.js**

While designing the visualisation part for the second iteration it was decided to display 2 tables to represent the output of the workflow. The right section of the index.HTML was split into 2 parts. The id for the first section was ***#table1*** and for the bottom section was ***#table2.*** Table 1 was responsible for displaying the original data and highlighting the changes that will occur while the code executes. When a specific row is removed, it will be highlighted with green colour. When a row is assigned as FIRST it will be highlighted with green and yellow if it is assigned as LAST. The second table will display the clean data. After the execution of the code is completed, the users will be able to review the final data with the correct outputs.

The table.js will use the same function that was used in the first iteration to create the table and display it. The only difference now is that the function will accept a second parameter called tableID to determine where the table should be displayed. Moreover, a new function had been created to highlight the rows in table 1. The method was being called in the logic.js whenever a business protocol was executed. It accepts 2 parameters, the index of the row that should be highlighted and the event such as delete, first or last. By getting the element by tag name and using a switch statement, the author could change the style of the specific row and change its background colour.

**logic.js**

The business logic had to be modified to combine the logic with the visualisation.

The operational process starts with reading the data and then initiates a for loop to extract the correct output, by looping through the rows of the data set. After the for loop is completed, there are 3 final checks before the end of the execution. The author spent most of the time working on the loop as it was the most complicated procedure. As the execution of the program had to be visualised, a different programming paradigm was used called event-driven programming to separate the logic into small components. By creating buttons and transforming each operation of the code to an event, the user will be able to loop through the flow of the program and examine the results on the screen using the tree and the tables. The two major events of this project are the for loop and the 3 final checks on the end. The for loop was replaced with a simple function called ***startLoop()*** that accepts a single parameter and the final checks were placed in a function called ***startChecks().*** Thena new event listener was created with an associated function that it was starting the for loop by sending an

|  |
| --- |
| Figure 6: logic.js(Author) |

incremented index every time the button was clicked. When the index equals the dataset’s size then it means the for loop is completed and the final checks should be called. When the index is more than the size of the data set, the function automatically terminates the execution, as there are not any other procedures left to check. After the code was separated into events, the collapsible tree and table had to be activated whenever a call was made. As the main protocols were to remove rows and change a binary value, a new function was created in the table.js file that accepts the number of the row that will be highlighted and the event like delete, last or first. Whenever a protocol was activated, this function was being called to highlight the rows with an associated colour representing the event. Moreover, the expand function in the tree.js was called to expand the correct node representing the protocols that were activated.

The results of the visualisation can be found in Section 5.1.2

### 4.2.3 Testing

Up to this point, the new visualisation application should accept a new input and all the execution could be illustrated with the tree nodes and the tables as the outputs. Using the correct outputs given by the KEG’s developers, the new output was compared to test the new application. Because these data were too large the author had to convert them into smaller JSON files to be used as test data. Each CSV file had the correct routes marked on the appropriate rows. By using the correct routes outputs, the author used the associated rows between each route from the raw data and created many JSON files to test each route individually. During the testing phase, the author spotted another interesting error that was caused by the business logic of the application. In the previous iteration, each business protocol was checked individually by passing the expected input to execute the process. Everything produced the expected output but they were not tested together. In order to track the process, the author decided to print on the screen the current length, index and the global index. Whenever rows were removed the i was not changing as the dataset’s size was resized automatically. The global i will represent the current row on the table that is currently selected. The error was occurring only whenever the delete protocol was being activated.

|  |
| --- |
| Figure 8: draft example (Author) |

The examples on figures above, illustrated the problem using squares as the object in the list marked with a letter as their name. The first figure illustrated an example when the current element should be assigned with FIRST. When the index is 0 and the current element is B, the value of FIRST is assigned to it and when the index increments the current element becomes the next element located on the 62nd position of the array. When the application wants to remove the current element and the index is 0, the next element automatically becomes the current element. The problem here is that when the index changes and an element was removed, the current element must move to the next position skipping one element. As you can see from the figure on the right, the current element should have been C instead of D. Whenever an element was removed in the visualisation application, the next element would have been unchecked, as the business logic automatically moving to the next element before even the index was incremented. To solve this issue the author had to decrement the index whenever a raw was removed from the data set.

### 4.2.4 Stakeholder feedback

Up to this point, the execution of the RoutesFinder application was successfully visualised. During the demonstration of the current application, the stakeholders were impressed with the new demo. Although the main requirement was successfully implemented, the system was not easy to use, as the data were large and the users cannot move through thousands of lines of data just using a single button to move to the next line. Furthermore, it will be much flexible if validation rules were to be applied to the system in order to analyse the results and automatically spot errors in the workflow. Finally, the client now provided the final requirements needed to finish this project.

## 4.3 Iteration 3

After gathering the requirements from the last meeting, the author had to finish all the features and change the business logic to allow flexibility on the system. The previous button must be implemented and a new layout should be designed to filter the results and find a new way to visualise the output. After analysing the data, verification methods were designed to automatically spot errors in the system according to the business rules.

### 4.3.1 Design Visualisation

### 4.3.2 Improve business logic

### 4.3.3 Testing

### 4.3 .4 Stakeholder feedback

# 5. Results

This section will cover in detail the results of this project, the outputs produced using the methods from section 4. The author will then talk about the importance of each output produced and how significant these results were for the next phases of the project. It will follow a similar structure to section 4 stating the outputs and discussing the result in chronological order like the methods were presented before.

## 5.1 Iterations Results

### 5.1.1 Iteration 1

**Simple Visualisation**

|  |
| --- |
| Figure 9 demo (Author) |

|  |
| --- |
| Figure 10 demo.js (Author) |

The first deliverable for the first iteration was basic and simple. The author created 2 methods which create the shapes. After reading and checking the data, the results were stored to the associated variables for each shape. The square was responsible for filtering the data based on given values such as depot, shed and sidings. Furthermore, while creating the shapes, the onClick() method was added to attach the table with the data on the DOM when the user clicks the associated shape.

|  |
| --- |
| Figure 4: logic.js (Author) |

**Java to JavaScript**

By converting the data into a JSON format it was easier to manipulate them. Every row in the dataset was represented by an object. The columns were the attributes of that object and they could be accessed by simply using the key as the number of the row and the value as the number of the column. For that reason, the author decided to use a simple for loop to represent the ListIterator and replace the ascosiated line in the java code with the while(hasNext()). After the system read and checked the data, they were stored in a variable called ***ds*** (alias for dataset). ListIterator interface use pointers to jump between elements. Methods like next() or previous() return the element next to or before the current element. By defining 3 variables in the local scope called curr, prev and next, the next() and previous() methods of the interface will be replaced by these simple variables that will hold the value of ***i***, depending on their position. Moreover, methods such as hasNext() or hasPrevious() will be eliminated as prev takes the value of null when ***i*** equals 0 and next takes the value of null when ***i*** equals the size of the data set. Whenever the author wants to ensure that the previous or next element is not empty he can simply check if the value of prev or next is not null.

The remove method from the ListIterator interface could be replaced using the splice method by passing the associated variable that defines the position of the element that must be removed. Splice will automatically reduce the size of the length, so there is no need to worry if the index runs out of bounds, as every time the loop ends, the length of ***ds*** will automatically change and the appropriate values will be assigned to the local variables when the loop is accessed again.

**Testing**

During the testing phase, the author spotted many interesting mistakes that were caused by the new logic. To ensure that each part of the code was producing correct outputs, the author had to understand clearly how each block of code was executing in the RoutesFinder.java first. After commenting each block of code with a name, the author was drawing on a paper the procedure that was going to be executed to help him implement the new logic to the logic.js file. Then the outputs were printed to the console and they were compared with the correct outputs in the spreadsheet KEG provided to the author. An example could be seen in figures below.

|  |
| --- |
| Figure 5: RoutsFinder.java (KEG) |

|  |
| --- |
| Figure 6: Execution Process (Author) |

Figure 5 illustrates the process to remove the current and the next element (if it exists) when the current destination is depot and the previous is not depot and contains the START value. It also checks if the element before previous contains the LAST value, if not it assigns the value to the end of the String. To understand how this works, the author draw boxes as the elements as it can be seen in figure 6. The pointer is always located next to the current element when it starts, which now is selected. Whenever the pointer moves to the right or left, it selects the element that has been passed through. Now, when the else if statement executes the pointer moves to the left (it.previous()) and now is located left of the current element. It checks if there is an element to left and moves the pointer to the next element on the left. For example, in figure 6, the pointer now will be located on the left of the previous element. At this point, it checks again if there is an element to left and if it doesn’t contain the value of LAST already, assign it at the end of the string. Then it moves twice to the right, placing the pointer next to the element with the caption Current which is the currently selected element. Now it removes it and checks if there is another element to the right. If this is true, the pointer moves to the right and removes the next element.

|  |
| --- |
| Figure7: logic.js (Author)] |

|  |
| --- |
| Figure 8: test.json (Author) |

As described above, after understanding the execution, the same block of code was re-written in the logic.js. The interesting part with the new logic is that whenever an element is removed, the local variables curr, prev and next are not updated. For example, in that case, when the current row is removed the next element, automatically becomes current. To remove 2 elements in a row, the author must call the splice method with the same variable twice. For this example, the variable is the curr. It’s like a stack of papers, if you remove the second paper, the third automatically becomes the second. To ensure the output was correct the author created test data (figure 8). After commenting all the previous blocks of code, the author was able to examine the results on the console as it can be seen in figure above.

|  |
| --- |
|  |

By repeating the same process, the author managed to rewrite the whole RoutesFinder.java to JavaScript. After testing all outputs on the console, the author proceeded to create the basic visualisation.

### 5.1.2 Iteration 2

### 5.1.3 Iteration 3

## 5.2 Alternative Results

## 5.4 Evaluation

* Complexity problems
* The data list could have been converted into a list of strings like java

# 6. Conclusion and discussion

In this chapter the author will access the work done throughout the project and whether it was enough to achieve the project objectives and leave the client satisfied. Moreover, the author will discuss what they learned while developing the project as well as things that could have done differently. At the end, the author will discuss any possible future developments to the system.

Throughout this period of time, a lot of objectives were set by the client for the author. Some of them were able to be achieved while others not so much. To begin with, the primary objective of this project was to visualise the execution process of the RoutesFinder application that would be used to identify potential errors on the business protocols and help developers to test their code and identify mistakes. This primary objective was overall achieved; the author was able to develop a fully functional visualisation application that can be used to use different inputs and produce the expected output of the execution with verification factors. This application is extremely easy to use even for untrained personnel and will improve the efficiency of the new developers that are working on the current project with the train fleets. In addition to this primary objective, a lot more sub-tasks were requested by the client during each iteration.

Firstly, the client requested for more layouts in order to filter and display the results with different views. Furthermore, it was requested by the author if there was enough time to also add more verification protocols that will display how successful was each operation of the application. These sub-objectives were important for the client but not essential since the primary focus was on the visualisation of the execution process. Both tasks were not finished completely due to time constraints as the verification process needed to use Machine Learning to predict the likelihood of success for each operation.

Overall the author believes the project is a success due to the fact that this was the first time the author worked on a large scale project alone. It was very hard to keep track everything that had to be done for the project as well as finding the right ways to do them. Moreover, this is an ongoing project as the client is in the same city with the author and there are a direct contact and plants to extend this project during the summer. Nonetheless, the author was able to deliver a fully functional product in time that would accommodate the needs of Knowledge Engineering Group.

## 6.1 Future work

## 6.2 Personal Review and what the author would do differently

This project has been a personal success for the author, for having improved in so many different ways. When tackling large scale projects it is necessary to either learn fast or fail therefore the author is happy he was able to improve dramatically some of their skills like programming in JavaScript or creating a detailed plan to follow. Moreover, due to the fact that this was a client based project the author was able to also improve their communicating skills as well as their information gathering abilities and attention to detail. Deadlines and a large workload also helped the author’s time managing abilities.

However, there are still a lot of ways the author can improve even further. For starters, there is the need to learn how to fully extract requirements through the information provided. The author in some cases was unable to fully understand what was asked by the client resulting in losing precious time. Furthermore, there are a lot of different languages to program on therefore there is still a long way to go before the author considers himself a good programmer. In addition, sometimes the author found himself overthinking a specific task instead of directly working on it which again cost them precious time; hence the idea of the agile model states, one must start developing as soon as possible and then review the results.

# 7. References

[1] Mattila, Anna-Liisa et al. "Software Visualization Today".

Proceedings of the 20th International Academic Mindtrek Conference on -

AcademicMindtrek '16 (2016): n. pag. Web. Accessed: (9.Dec. 2016.)

[2] Bostock, Mike. "D3.Js - Data-Driven Documents". *D3js.org* Available at: https://bl.ocks.org/mbostock/4339083 Accessed: (17.Nov. 2016.)

[3]Carson, J.S. "Model Verification And Validation". *Proceedings of the Winter Simulation Conference* n. pag. Web Available at: http://ieeexplore.ieee.org/abstract/document/1172868/?reload=true Accessed: (30 Nov. 2016.)

[4] "Collapsible Tree". *Bl.ocks.org*. N.p., 2017 Web Available at: https://bl.ocks.org/mbostock/4339083 Accessed: (27 Nov. 2016)

[5] "Iterators And Generators". *Mozilla Developer Network*. N.p., 2017. Web Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Iterators\_and\_Generators Accessed: (6 Oct 2016)

[6] "Listiterator (Java Platform SE 7 )". *Docs.oracle.com*. N.p., 2017. Web Available at: https://docs.oracle.com/javase/7/docs/api/java/util/ListIterator.html Accessed: (17 Oct. 2016)

[7] Mattila, Anna-Liisa et al. "Software Visualization Today". *Proceedings of the 20th International Academic Mindtrek Conference on - AcademicMindtrek '16* (2016): n. pag. Web Available at: http://dl.acm.org/citation.cfm?id=2994327&dl=ACM&coll=DL&CFID=934361446&CFTOKEN=38704645 Accessed: ( 9 Dec. 2016)

[8] "Mike Bostock". *Bost.ocks.org*, 2017. Web Available at: https://bl.ocks.org/mbostock Accessed: (14 Nov. 2016)

[9] Najafi, A. "Model Validation And Verification Of Data Mining From The Knowledge Workers Productivity Approach". *Journal of Applied Sciences and Environmental Management* 15.1 (2011): Web Accessed: (13 Oct. 2016)

[10] Sargent, Robert G. "Verification And Validation Of Simulation Models". *2007 Winter Simulation Conference* (2007):. Web Available at: http://www.bioline.org.br/pdf?ja11027 Accessed: ( 28 Nov. 2016)

[11] Sommerville, Ian. *Software Engineering*. 1st ed. Boston: Pearson, 2011. Print. Accessed: (31 Nov.2016)

# 8. Appendices

## 8.1 Appendix A

### 8.1.1 Project Planning

This project aims to develop a visualising environment that will be able to

help the client to make better decisions and identify errors on the early stages of

the process. This work will emphasise on developing a reasoning mechanism for real

time decision making, work on analysis of procedural based data for decision

making, reasoning upon successful signalling and provenance of any suggested

decision, automatic trace-back of any decision made.

### 8.1.2 Aim & Scope

The main purpose of this project is to allow the organisation to trace-back decisions

and examine the results produced by each step, allowing them to examine from

which part of the process the result came from. This will be succeeded by visualising

the process into shapes, indicating them as business “rules”. The ideal deliverable

will be an abstract model that allows organisation to use it in all case studies with

their unique signal protocols.

The supervisor for this project is Stelios Kapetanakis and Aidan Delaney will be the

second reader. The requirements for this project have been gained by the

stakeholders of Knowledge Engineering group. It was agreed that we will have

weekly/monthly meetings with the stakeholders at University of Brighton as we are using the

Agile approach. GitHub will be used to share code and we communicate through

emails.

### 8.1.3 Stakeholders & Interest

The organisation will benefit from this project as it will allow it to use it as an

abstract model for all its cases. On the first meeting, there was a discussion about the main

problem. There were many cases when a project was completed, many mistakes were identified or the deliverables did not meet the requirements. Sometimes the protocols that were used did not fit the case or the information produced from the business rules are

tainted. In most cases, the client can trace-back only on the last signal based protocols

before the final deliverable. This problem leads to consume time and money. This

project will help KEG to quickly identify issues in their workflow and visualise all

decision that have been taken until the last step. Furthermore, we can add an

extension that measures the success of the outcome in each decision to provide

feedback for the organisation. If this project succeeds we already gather thoughts to

create an Intelligent - automatic workflow data verifier that will be able to use

business rules and suggests and compare the final deliverable.

In every weekly iteration, the stakeholder will examine the work and provide

feedback for the project. When the first model will be created, the stakeholders will

test it with datasets from project they have already finished. Every time we will use

different rules, we will discuss what will be needed to add or remove from the model.

### 8.1.4 Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Requirement** | **Type** | **Priority** |
| BR1 | Finish first prototype until February | Business | High |
| BR2 | Make application extendable to accept any business rules | Business | High |
| FR1 | Allow the system to trace previous results from the process | Functional | High |
| FR2 | Validate the results | Functional | High |
| FR3 | Develop a reasoning mechanism for real time decision making | Functional | Medium |
| FR4 | Analyse procedural based data | Functional | Medium |
| FR5 | Software compatible with all Operating Systems | Functional | Medium |
| FR6 | Export model | Functional | Medium |
| UR1 | Visualise the whole workflow using shapes | User | Medium |
| UR2 | Display success rating using Red, Green and Orange colours when ever system is validating the results | User | Medium |
| Q1 | Fast Access (able to trace back fast and load the previous results | Quality | Medium |
| Q2 | Easy to use (design is intuitive and easy to learn) | Quality | Medium |
| Q3 | Result validation is accurate using a well-structured algorithm | Quality | High |

### 8.1.5 Risk Assessment

As in all projects software or not there will always be risks that may affect the project

and make it vulnerable. Even though this is a project for University students, there

are some factors that might affect the project’s process. An assessment was

conducted and the major risks were identified. Each risk describes how it affects the

project and shows an estimate of how likely it’s going to occur. Possible strategies of

how to tackle the problem were included. All these are factors shown in the table

below

### 8.1.6 Project Risks

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Hazard** | **Likelihood**  **(1-5)** | **Impact**  **(1-5)** | **Decision** |
| 1) | Failed to establish a good  communication with the  stakeholders | 4 | 4 | In every meeting make sure you  have understood well what it must  be done. Keep asking questions  until everything i clear to proceed |
| 2) | Power outages | 1 | 3 | Keep your work on cloud or Github |
| 3) | System loss | 2 | 5 | Keep work both on cloud and  external disks |
| 4) | Health issues | 3 | 3 | Be careful from the weather. Also,  set up an extra week that you might  use it in cases like this |
| 5) | University Assignments | 4 | 4 | Make a good plan that will include  all modules. Make sure that you  follow the plan to avoid collisions  with other deadlines |
| 6) | Changes on  requirements | 2 | 3 | Design the project in a way to be  extendable for changes |
| 7) | Errors in testing | 4 | 4 | Weekly tests to avoid  unexpected errors |

### 8.1.7 Methodology

After a palatable level of research led the procedure for the project was picked.

Inquire about was done on different methodologies keeping in mind the end goal to

decide the most proper approach for the extend. Agile, Waterfall, Rapid Application

Development and Spiral were the principal methodologies that were analysed having

at the top of the priority list prerequisites, points and targets. After the investigation

was done it was presumed that every one of the techniques had their points of

interest however the most reasonable model to utilize for this project would be the

Agile method. Agile is a more adaptable and lightweight procedure for programming

projects contrasted with conventional methodologies, agile gives the flexibility rolling

out improvements to the project as it comes, new changes can be actualized at next

to no costs because of the recurrence of the iterative cycles that are conveyed. This

allows the flexibility to change the requirements in any point of the project without

affecting it in a significant way

Unlike the waterfall approach, you do not need to plan and consider everything

before you start. You can identify the basic requirements and start right away,

communicating with the client in iterations, providing you feedback and develop

each step every time.

### 8.1.8 Gantt Chart

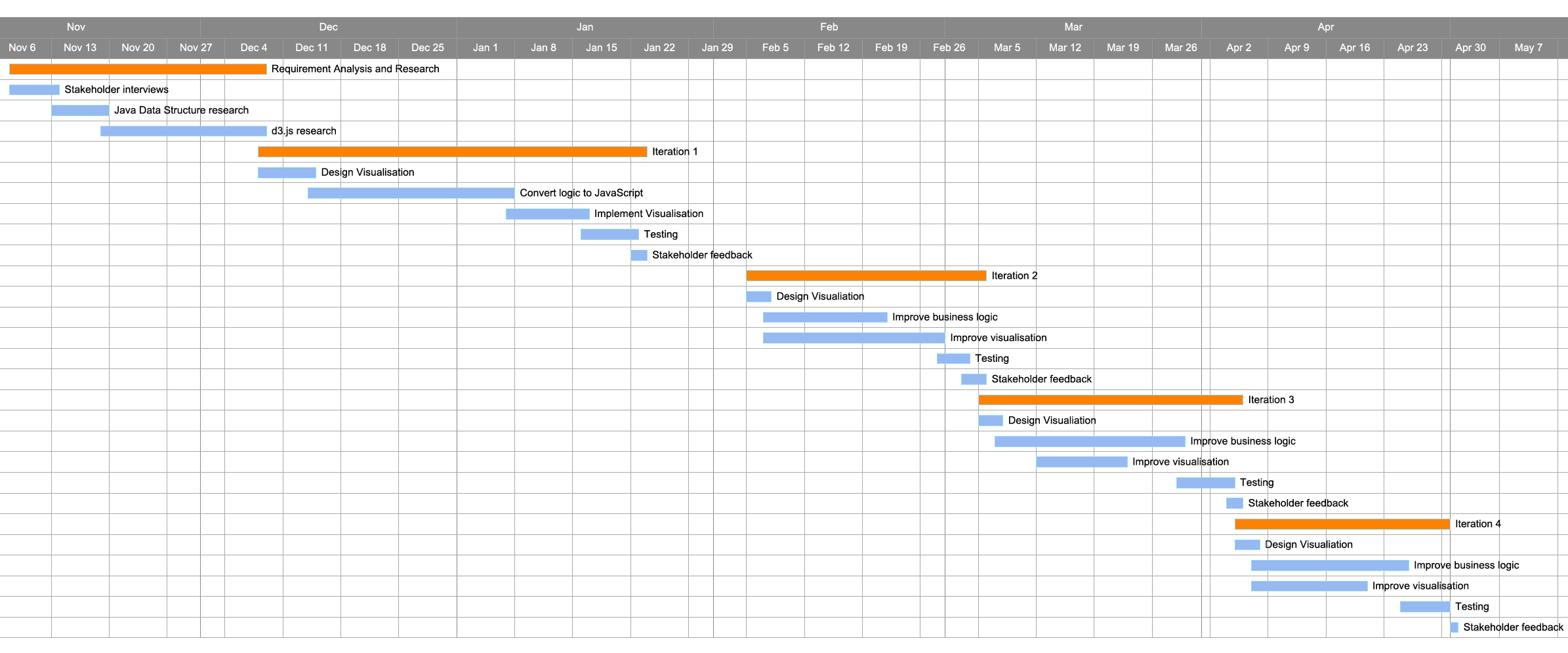
The following charts will describe the process that will be taken to complete this

project. The process is broken into 3 major milestones illustrated in a timeline form.

The implementation will follow an agile methodology that includes some iterations

called sprints. The first Gant chart represents the original planning but after the delay a new one was created.

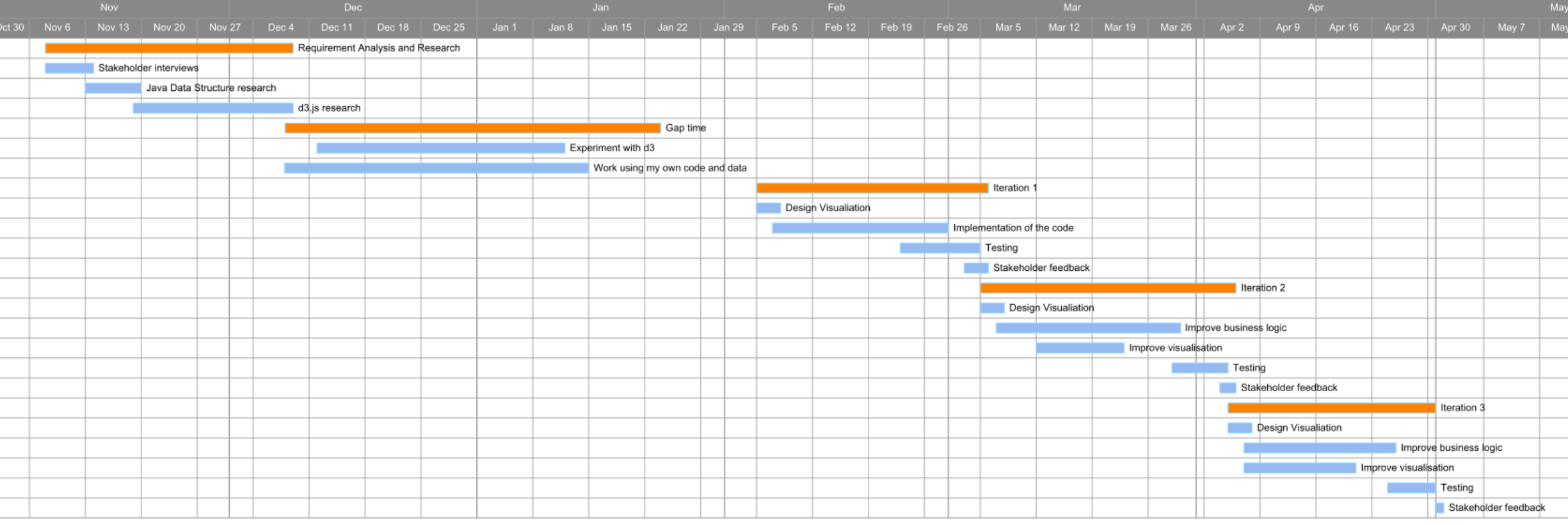




### 

### Modified Gant Chart





### 8.1.9 Ethics form

SCHOOL OF COMPUTING, ENGINEERING & MATHEMATICS

ETHICS FORM

This ethics form is designed to help you quickly and easily identify how you should

approach any ethical issues raised by your project or dissertation. It should be

completed for ALL research projects and dissertations prior to the commencement of

the project. Please do not approach any participants involved in the research until this

have been completed and discussed with your supervisor or member of the CEM ethics

committee (if appropriate).

This form must be completed by the project student or researcher responsible for the

project. Once completed, you should discuss it with your supervisor to ensure that you

take the right follow-up actions.

If you answer ‘No’ to all questions in this form and this is confirmed with

your supervisor (if appropriate) then no further action is required. Please

note that in signing this form you accept that it is still your responsibility for your

project or dissertation module to follow the University’s Guidance on Good Practice in

Research Ethics and Governance, available on StudentCentral. Any significant change

in the question, design or conduct of your project or dissertation that would alter your

answers on this form must be notified to your supervisor who will advise you on

whether you need further action.

If you have answered ‘yes’ to any of the questions in Section B of the

Student Checklist your supervisor will need to make a judgment as to

whether or not the research includes more than a minimum level of risk.

If this is the case then your supervisor will need to email this form to the

CEM ethics committee (CEMethics@brighton.ac.uk) for discussion prior

to the commencement of research. This does not mean that you will not be able

to do the research, but it will need to be considered by the School Research Ethics and

Governance Committee.

Ethics forms, example consent forms/participant information sheets and supporting

guidance are available on the Research Ethics for Projects – CEM area of

StudentCentral.

Signed copies of this completed ethics form must be submitted with your

project or dissertation. Note: the project or dissertation will not be

marked if the completed checklist is not included.

**PROJECT DETAILS**

1. Name of researcher: Leonidas Constantinou

2. Name of supervisor: Dr. Stelios Kapetanakis

3. Title of project: Automatic Workflow Data Verifier

4. Outline of the research (up to 100 words):

This project aims to develop a visualising environment that will be able to

help the client to make better decisions and identify errors on the early stages of

the process. This work will emphasise on developing a reasoning mechanism for real

time decision making, work on analysis of procedural based data for decision

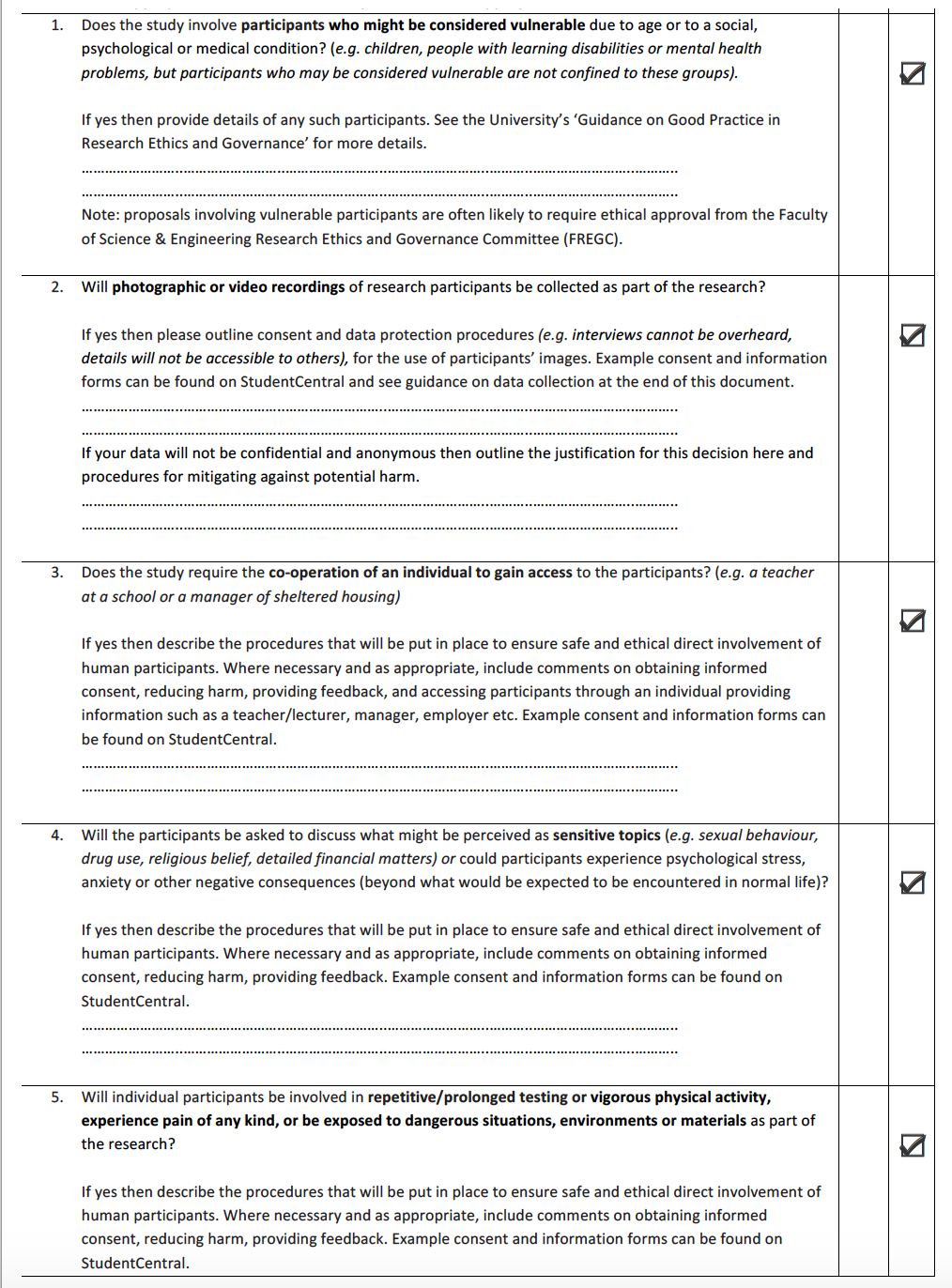
making, reasoning upon successful signalling and provenance of any suggested

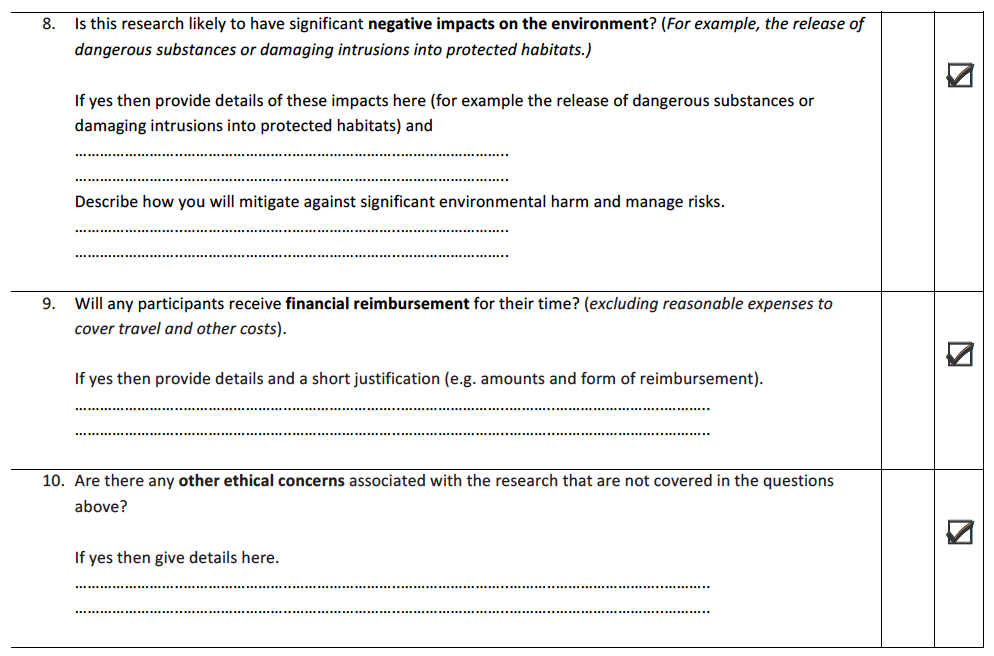
decision, automatic trace-back of any decision made.

5. Location of research: Brighton

8. Email address: lc475@uni.brighton.ac.uk

9. Telephone number: 07519261396





**All Undergraduate and Masters level projects or dissertations in the**

**School of CEM must adhere to the following procedures on data storage**

**and confidentiality.**

All data should be encrypted and stored securely. Documentation should be kept in a

locked cabinet or desk, and electronic data should preferably be kept on a removable

disk or data stick which can be locked away, or if this is not possible on a password

protected computer. Confidential and sensitive data should not be emailed unless it

is encrypted or password protected since emails are centrally archived.

For Undergraduate/Masters projects, normally only the student and supervisor will

have access to the data (see the University’s ‘Guidance on Good Practice in Research

Ethics and Governance for further details). Once a mark for the project or

dissertation has been published, all data must be removed from personal computers,

and original questionnaires and consent forms should be destroyed unless the

research is likely to be published or data re-used. If this is the case a justification for

this should be included where appropriate in this form and in the relevant consent

and participant information forms.

**Student**: Please sign below to confirm that you have completed the Ethics form and

will adhere to these procedures on data storage and confidentiality.

Signed (Student): LEONIDAS CONSTANTINOU

Date: 28/11/2016

**Supervisor**: I confirm that the research **does/does not** (delete as applicable)

include more than **a minimum level of risk.**

Signed (**Supervisor**): ………………..………..……………

Date: ………………..………..……………..

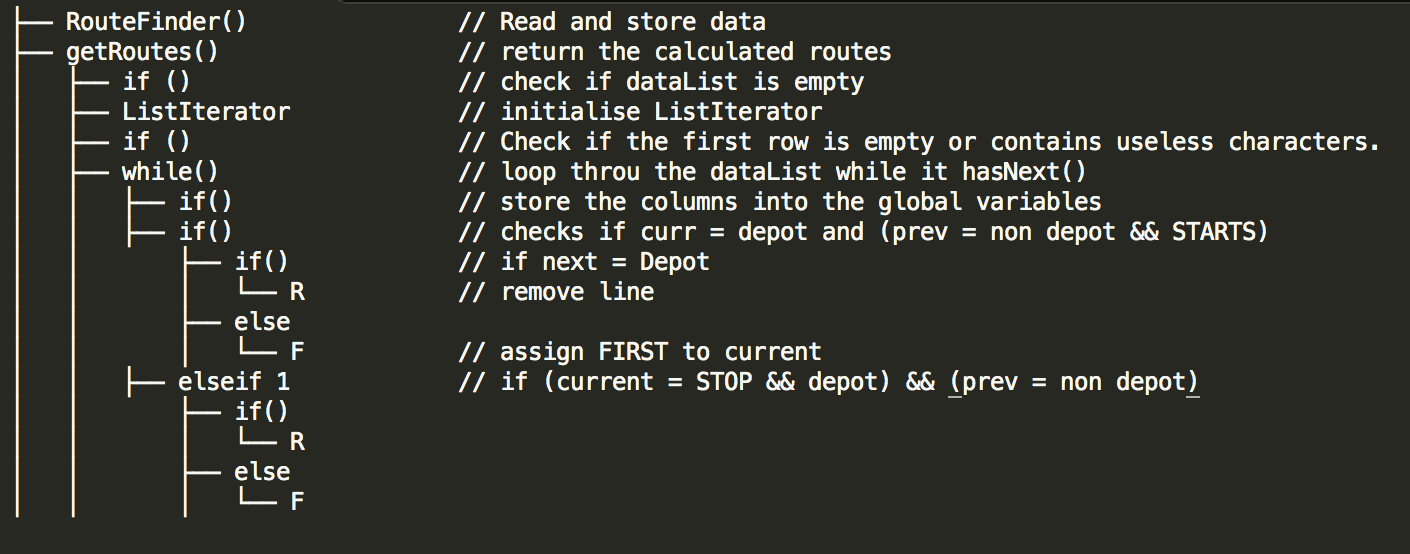
Note: If the **supervisor** judges that there is more than **the minimum level of risk**

then your supervisor will need to email this form to the CEM ethics committee

(CEMethics@brighton.ac.uk) for discussion prior to the commencement of research.

## 8.2 Appendix B

### 8.2.1 Java Code Structure



Because the routes finder application contains 660 lines of code the author tried to structure the code in a simpler format to be more readable to help him understand it clearly and re-write the code into JavaScript.