Major Project Report
Software Engineering 14:332:452:01
Group 2 - Robert Reid, Kiernan King, Daniel Mitchinson, Brian Jimenez, Wiktor Fedorowiat
May 2nd, 2022

*Drafted by Robert Reid and Kiernan King

Introduction

This report serves to highlight the software engineering processes taken by our group in developing our business application which we called FreshTime. This system's purpose is to provide companies that utilize delivery couriers to provide customer service a way to attain useful metrics in aiding in financial planning.

Final Requirements Table

Functional Requirements
Must allow input of a database of locations of restaurants and points within the area of service.
Must now allow databases with less than 25 restaurants.
Must assume that a maximum of 1000 orders will be served at any given time.
Should calculate the average wait time per order.
Should output a mapping of restaurant orders and drivers that achieve the target average wait time.
Must have an interface for uploading a database or inputting database connection credentials.
Must include middleware for interfacing with routing service API
Must be able to get rid of invalid locations within the area of service based on organization parameters.

Non Functional Requirements
Must have a document instructing users on how to format their restaurant and area of service database.
Must be easy to use.
Must display data graphically.
Must be able to output data to a local file system.

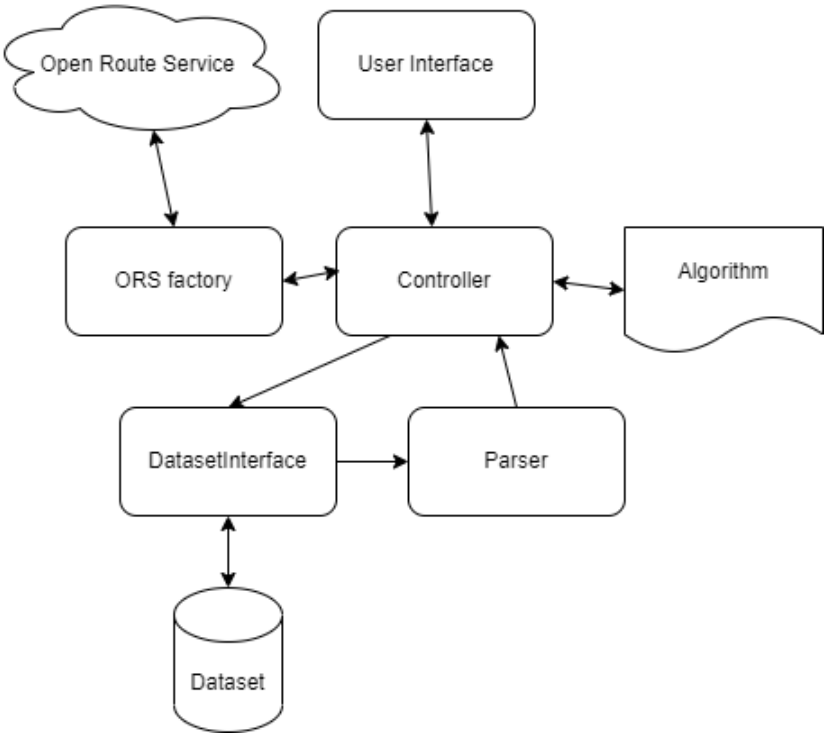
Design Features Specification

3.3 System Features (From Software Specification Document)

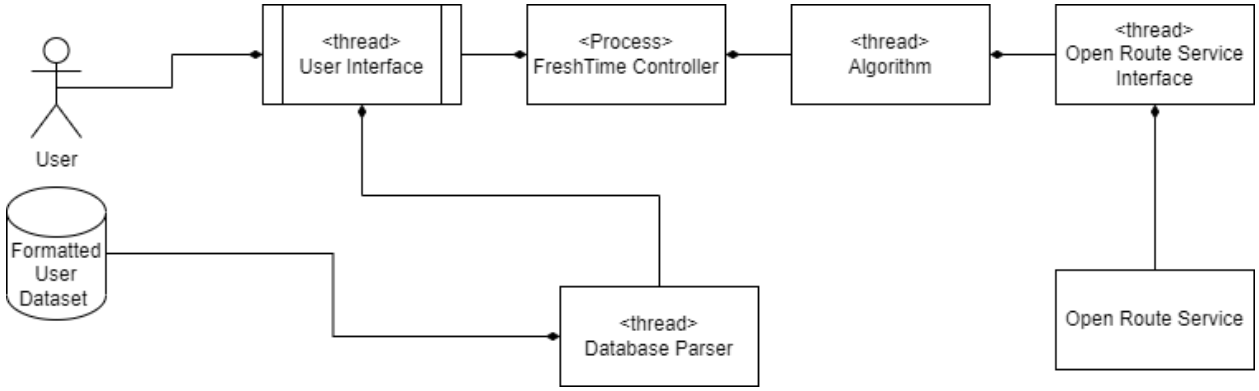
- Application User Interface
- Database Management Integration
- System outputs optimal matching of drivers to restaurants in order for customers to have an average wait time lower than organization needs.
- Display average time assuming 1000 deliveries per hour, and a graph showing how the number of drivers changes as the number or restaurants is doubled from a sample set of 25.
- Support for different databases
    - MongoDB
    - mySQL
    - Local file upload

Final System Architectural Diagrams

- **Conceptual View Diagram**



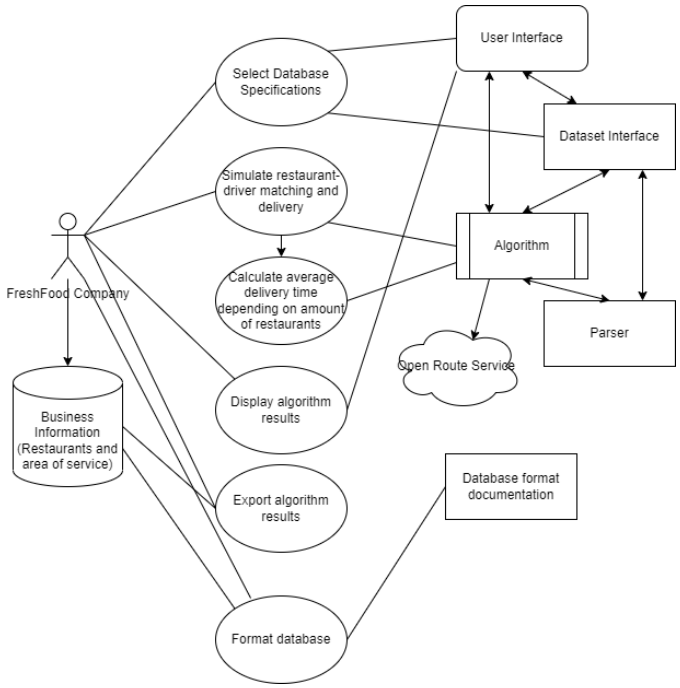
- **Process View Diagram**



The conceptual view diagram features the components we identified would be required to fulfill the functional and nonfunctional requirements. Initial version of the conceptual view diagram did not include a dataset interface component or a dedicated software controller component. The appearance of the dataset interface component in sprint three reflects the additional feature of support for multiple database solutions. The process view diagram shows how the system consists of a single process that initializes a running user interface thread and subsequently the other software components at runtime. In the process view diagram it can be clearly seen that the Open Route Service, User, and Formatted User Dataset are third party components of the system and how they are included in the behavior of the system.

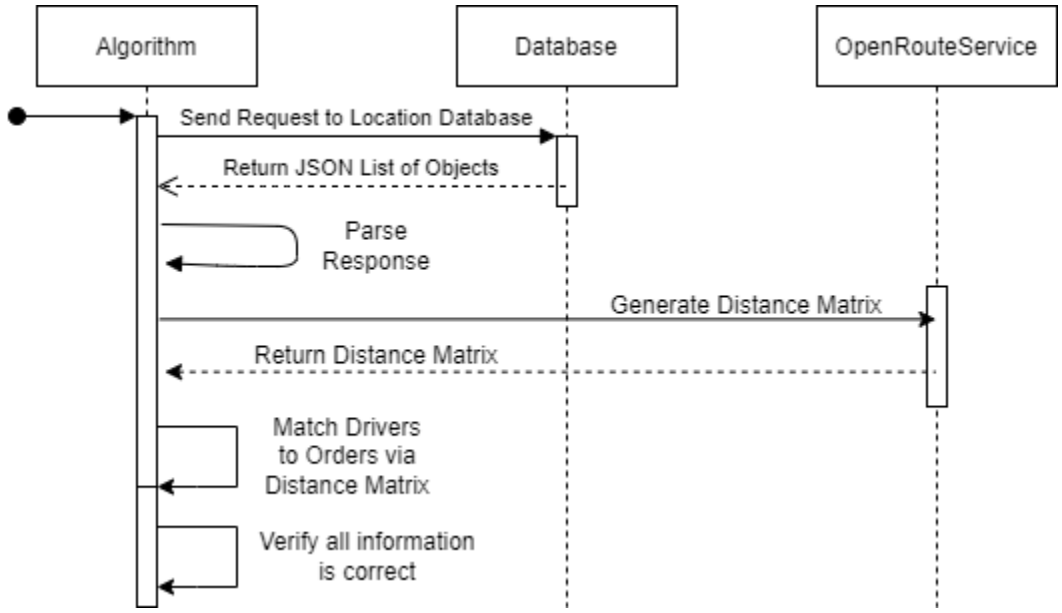
Final System Modeling Diagrams

- **Use Case Diagram**



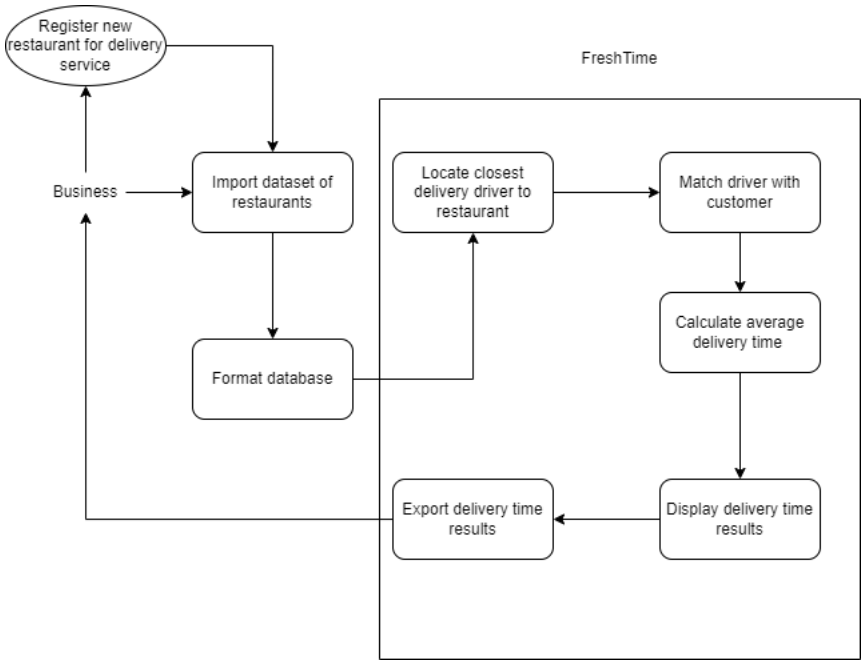
The above use case diagram reflects the typical use cases of the FreshTime system. The user is intended to be a business or otherwise organization that has a delivery service with a set of 25 from a single point to an area of service. The FreshFood company utilizes at least 25 restaurants to be distributed over an area of service. We decided to make our own dataset using Jersey City NJ, which consists of over 150 restaurants. In addition, our team developed an algorithm in excel to generate over 4500 locations around the area that indicate this area of service. Extensive documentation and instruction on how a business can generate this area of service is available to customers. Customers of FreshTime should take their dataset and generate a set of data points using our specifications, and then upload this using our supported database solutions. These solutions include mongoDB, a local file upload, and projected mySQL support. The user can then simulate the restaurant driver matching and delivery and export the metrics for their own purposes.

- **Sequence Diagram**



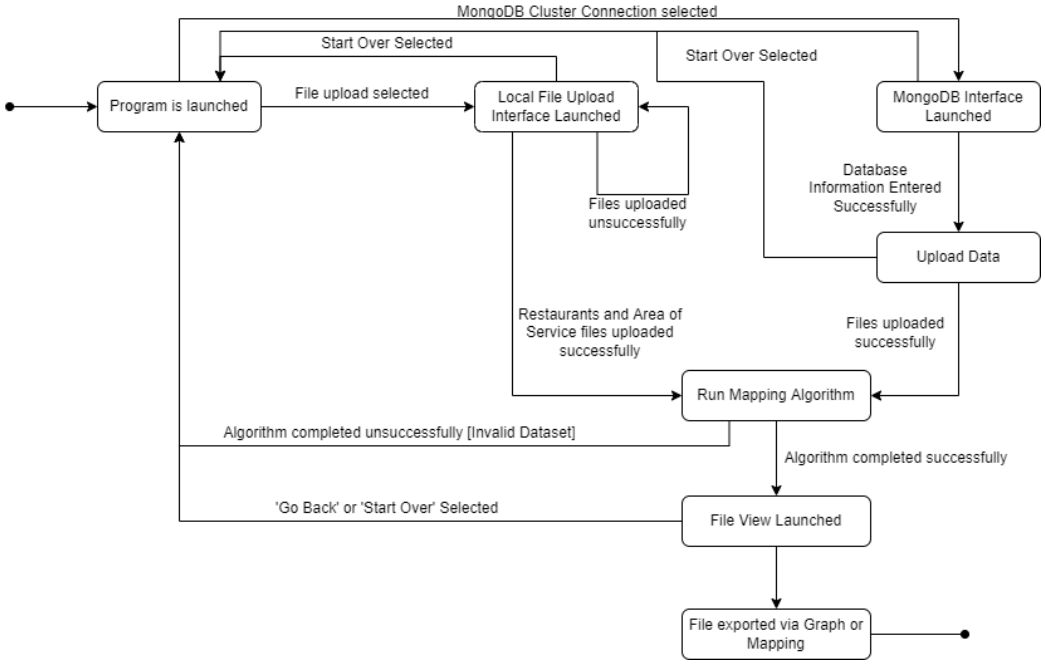
The above sequence diagram depicts the operations performed by the algorithm as it returns the distance matrix from the routing service API, Open Route Service.

- **Business Diagram**



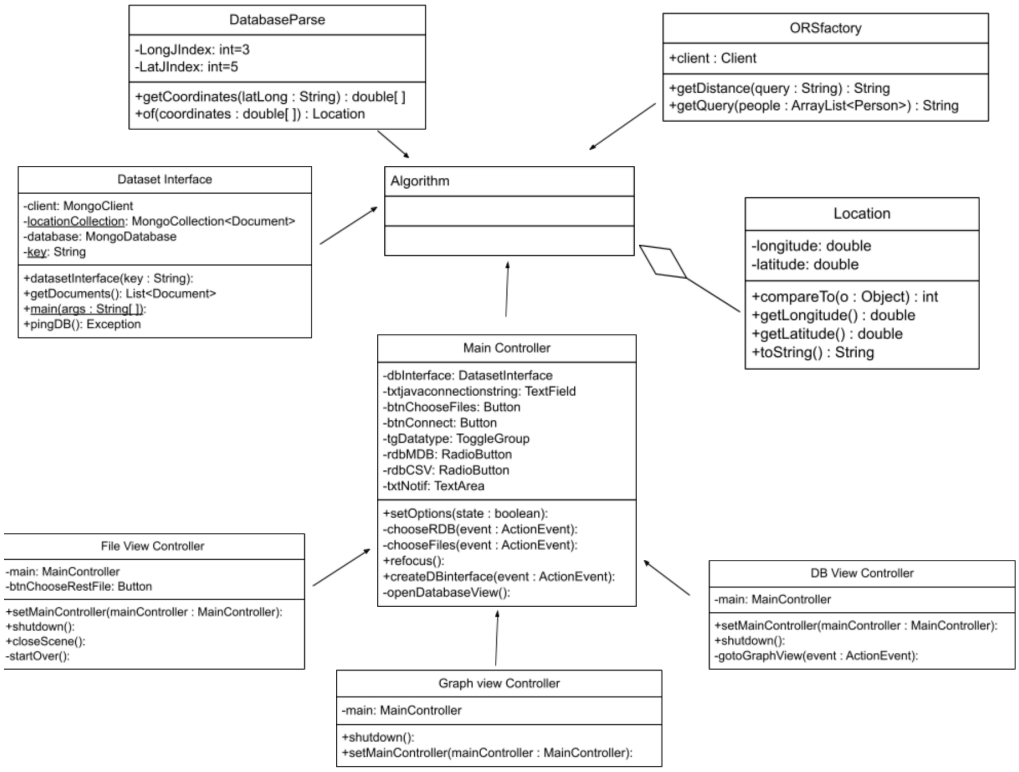
The above business diagram reflects how the FreshTime application process integrates into organizational patterns and assists the company in automatically generating useful metrics based on their restaurant delivery service.

- **State Diagram**



The above state diagram reflects the current FreshTime application operation as seen from the user's perspective.

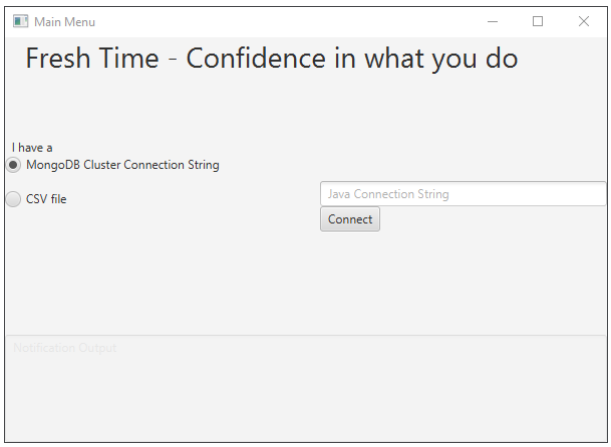
- **Class Diagram**

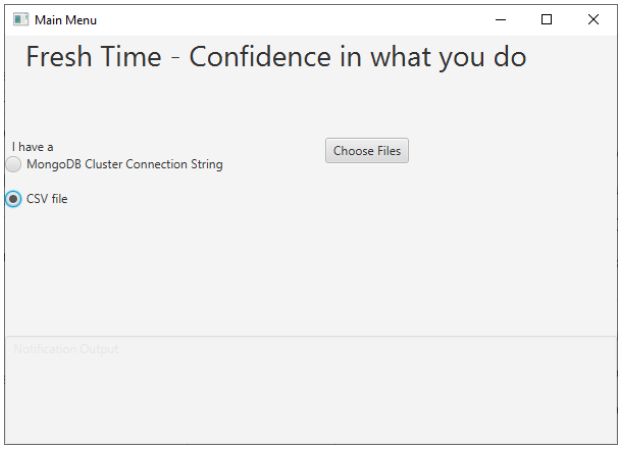


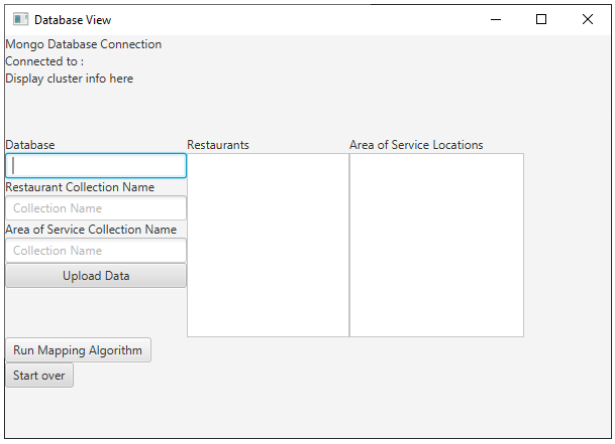
The current class diagram reflects the code available for navigating the different user interfaces, and connecting to a mongoDB.

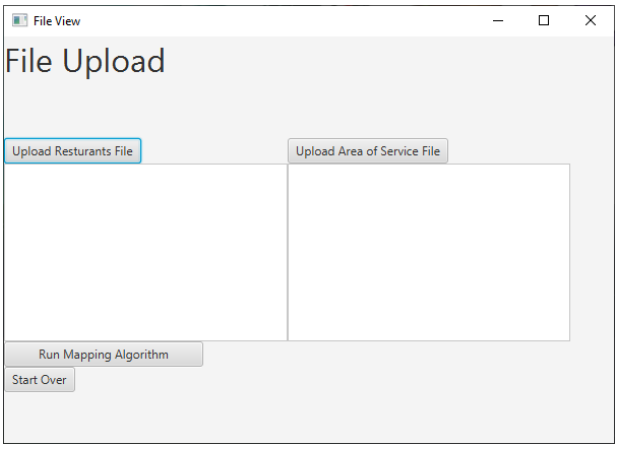
Final Design Implementation

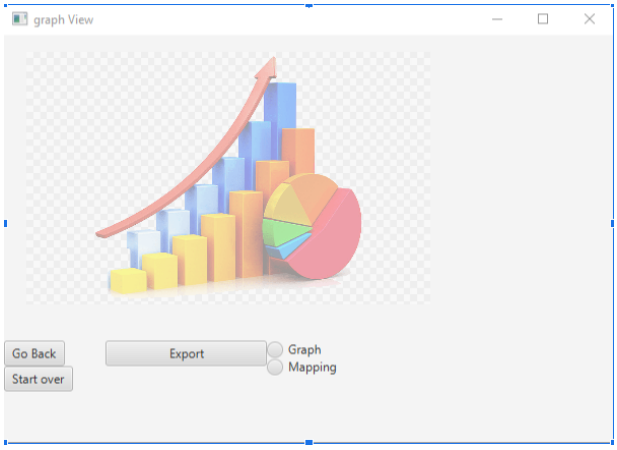
- **User Interface Pages**











- Documentation

	Lat	Long	Adjusted Lat	Adjusted Long
			(add/sub .1degrees)	(add/sub .1degrees)
Max	40.75857694	-74.03179529	40.85857694	-73.93179529
Min	40.69521011	-74.09850153	40.59521011	-74.19850153
Difference	0.06336683	0.06670624	0.26336683	0.26670624
Difference (ft)	18262.32041	19224.73837	75902.32041	76864.73837
Difference (mi)	3.458772804	3.641048933	14.37543947	14.5577156
Grid Spacing			0.003762383286	0.003810089143
Grid Spacing (8 decimal places)			0.00376238	0.00381009
Grid Dimensions = 70 x 70				

Team created an excel algorithm to generate the area of service data points based on the farthest restaurants from each other. This has accompanying documentation on how to use the algorithm for companies to generate a dataset based on their restaurants they will service with couriers.

- Open Route Service middleware (ORSfactory)

```
average time: 3.7589699999999997
```

The algorithm developed by the team was able to generate an http query for the Open Route Service API and capture and parse the distance matrix response. Above shows the result taken from a distance matrix of just two locations, so it returns the average time which is the time taken simply to get from point A to point B.

- Dataset Interface

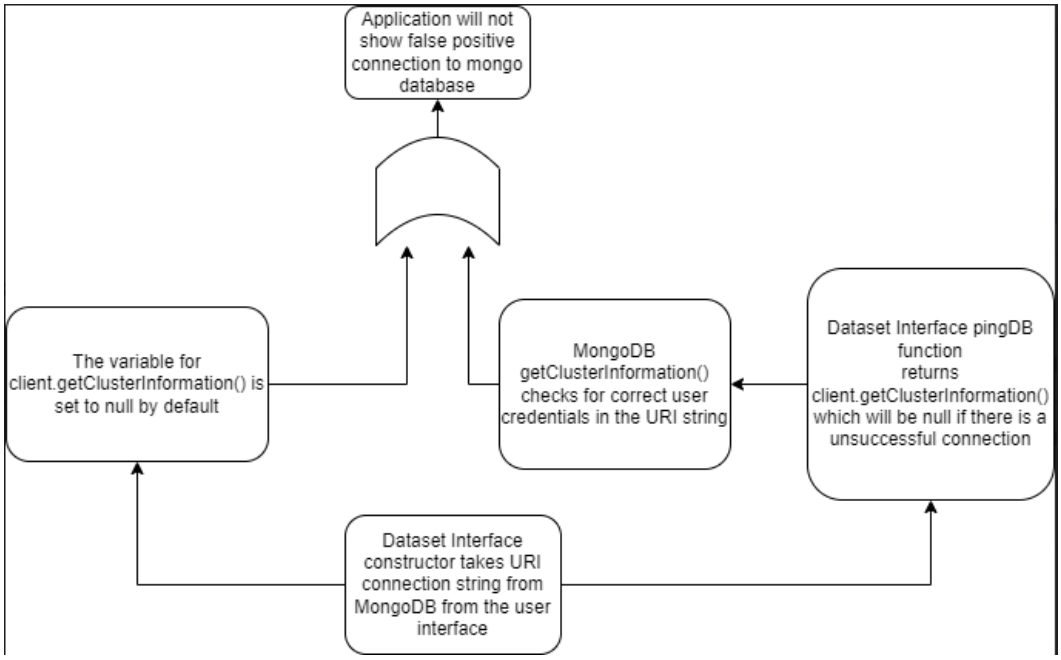
```
97 Passenger - From: -74.43708264,40.48421918 To: -74.45944339,40.52478453
98 Passenger - From: -74.43087266,40.52126556 To: -74.43740331,40.47633882
99 Passenger - From: -74.46344543,40.51610473 To: -74.43374151,40.52339661
100 Passenger - From: -74.45817843,40.52543859 To: -74.44171739,40.52557131
```

The team was able to upload the database to mongoDB and pull 100 random locations and display them to the console. This utilized not just the dataset interface but the parser class which parses the response, JSON in this case, so we could print it to console.

Our team chose to focus on these aspects of the design to implement because we thought it would be best to demonstrate individual functions of the program before integrating them wholly together by developing the main algorithm. We developed this application with scalability in mind, and perhaps not knowing the format that this service would be offered. This also proved to be useful because we were able to incrementally design certain components such as the dataset interface and the user interface once more features were identified in the requirements engineering process.

Dependability Engineering Process

- **Security**



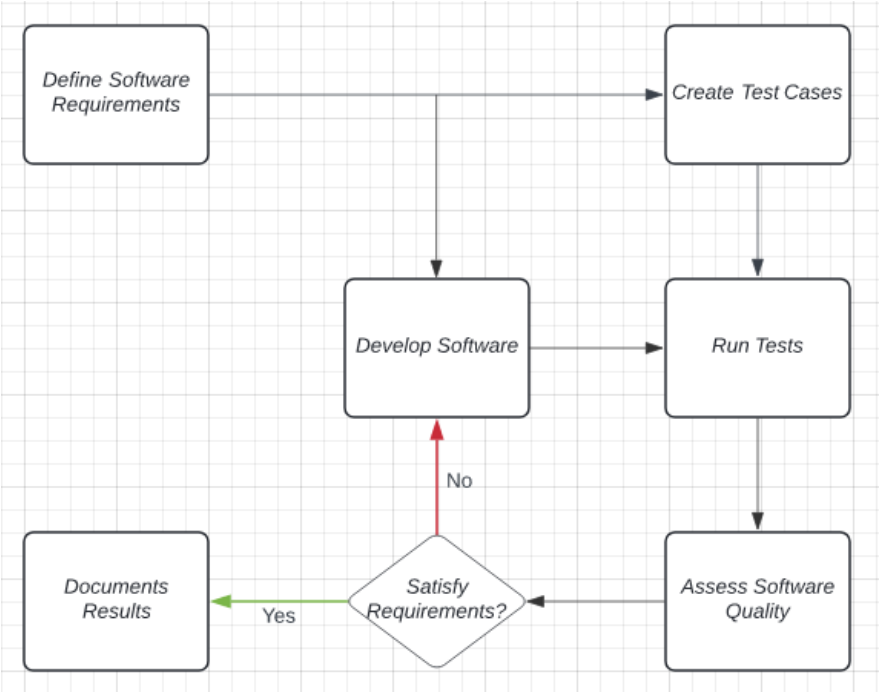
The above diagram is a safety diagram that shows that the application will not provide connection to mongoDB clusters with incorrect credentials. This code adds data security to our program by allowing the application to establish a direct connection with the third party dataset host. To do this we were able to use mongoDB provided API commands to ping the database with the credentials supplied by the user, thus utilizing their direct credential manager.

- **Availability**

Trying to improve from the minor project and overcome the constraints placed on the system by the free routing API, Open Route Service, our team tried to install a local version of the API using docker. This required a lot of research and culminated in the ORS_backend_installation document in sprint 3. We believe this added to the availability of our system because it eliminated functional constraints that impeded the availability metric of our application.

Project Management Process

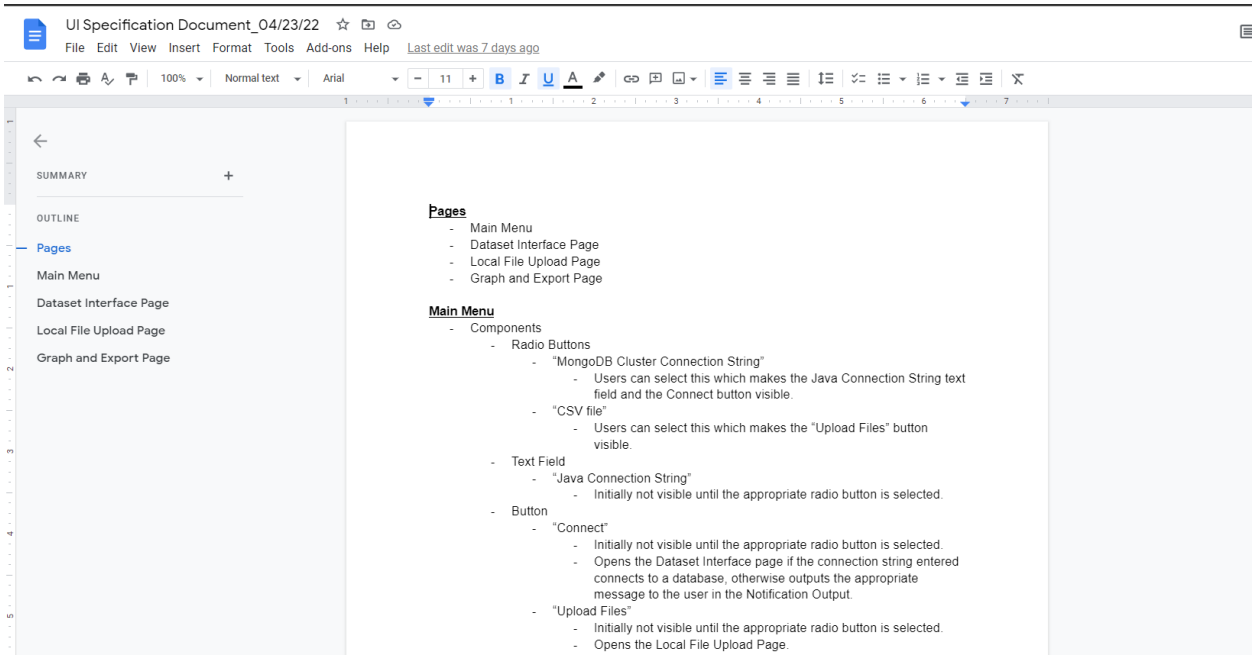
- **Quality Assurance Plan**
  The diagram below is of our **Generalized Review Process** which is a main part of our software quality assurance plan.



The Review Process goes hand in hand with Design Inspection of the following areas of software design using checklists catered to each portion of software that is being Inspected. This quality assurance plan was drafted after the user interface specification document, however program specification documents will be authored in assessing the quality of the software stage of our quality assurance plan.

- General requirements and design
- Functional and Interface specifications
- Requirement traceability
- Logic
- Performance
- Error handling and recovery
- Testability, extensibility

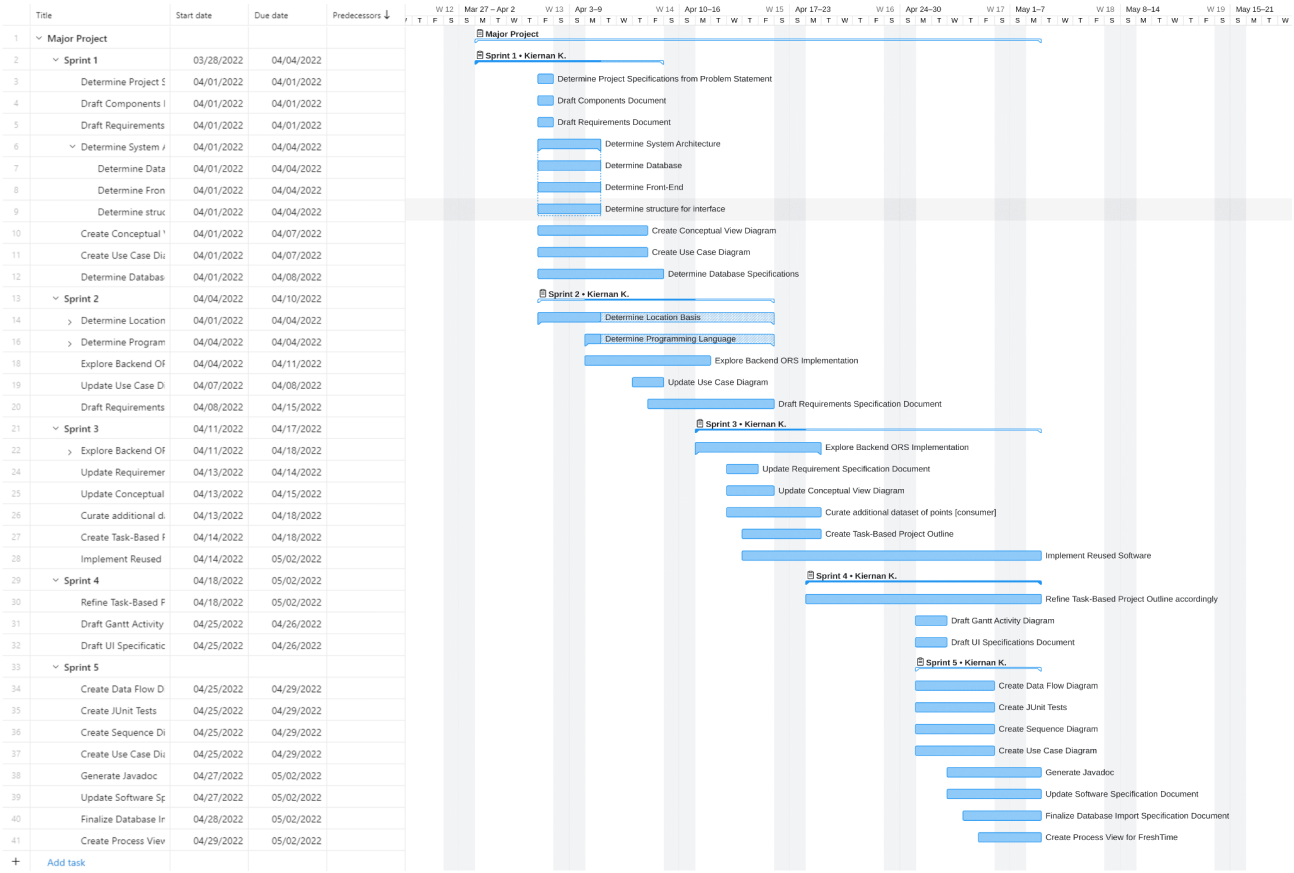
- **Specification Documents**



Program specification documents are going to be used to assess the quality of code that is being produced throughout the project. Our team started this to demonstrate niche functionality each

sprint while also tying this to the general use cases and system functional and nonfunctional requirements. The goal of these documents is to ensure quality code that meets the requirements of the projects. Above is an example document which analyzes the components of the user interface, and includes sections with descriptions on customer interactions throughout each use case.

- **Gantt Diagram**



- **Current Project Plan**

	FILLING THINGS THAT ARE COMPLETED IN WITH GREEN							
Component	Task	Requirement Specification	Application Feature	Time Estimate (Hours)	SWE Activity			Component List
Open Route Service	Reuse Byfar Code for getting JSON response from service.	Getting distance matrix between set of locations.		Completed	Software Reuse			Open Route Service
	Write JUnit tests for time distances from response.			3	Software Testing			
	Perform quality assurance test using ORSfactory			1	Software Quality			
User Interface	Develop architecture for the document screen from use cases.		Application User Interface	1				
	Write a document on how the user interface fulfills each usecase.			1	Software Specification			
	Design user selection of database from supported set of databases.	Allow input of mongo database for location index of orders and drivers.		2				User Interface
	Design user input of database connection credentials.			1				Dataset Interface
	Check for database status.			Completed	Software Reliability			Algorithm
	Design user input for run curate dataset from database.			4				Documentation
	Design user input for run algorithm.			4				Parser
	Design user input for exporting results.			4				
	Design display of results to screen.	Display the average time to delivery.		4				
	Graphically display the difference in average wait time vs drivers.			5				
	Check for correct database size. [Software]	Must be a minimum 25 restaurants available to order from.		2				
	Create a testing document for user interface.			3	Software Testing			
	Perform quality assurance test.			1	Software Quality			
Dataset Interface	Research area of interest and define area of service for FreshFood. (DECIDED ON JERSEY CITY)		Database Management Integration					
	Curate dataset of 25-50 restaurants. (WE HAVE 168 RESTAURANTS)	Must be a minimum 25 restaurants available to order from.						
	Curate dataset of 1000 possible dropoff locations. (WE HAVE 4900 LOCATIONS)	Must be able to handle up to 1000 hours at a time.						
	Create standardized database format.	Documentation for Fresh Food interaction with the system.	Support for different databases					
	Write code for getting collection from user database.	Application interface for uploading to database the company wishes to use.						
	Define methods for interacting with supported databases and returning JSON.							
	Implement methods for notifying user of incorrect data input or otherwise failure to interface with database or open file.				Dependable Engineering			
Parser	Define methods for parsing JSON response from Dataset Interface and returning usable data.		Support for different databases					
	Research database JSON response for MongoDB and other supported database.							
	Write code for parsing JSON response for supported databases.							
Algorithm	Generate workflow for algorithm behavior.							
	Write code for creating query for ORS.			Completed.				

Documentation	Generate database format document for supported databases.							
	Generate javadoc.							
	Class Diagram							
	Develop a use case diagram for FreshTime.			Completed				
	Draft MongoDB Format Document							
	Generate Gantt Diagram							
	Generate State Diagram							
	Generate Sequence Diagram							

Our project plan was created early on as a way to keep track of the tasks that were required for each component of our system in order to build our first baseline. The tasks from this table are also used in our Gantt diagram which was used to plan out our sprints. In addition to these project management processes, we made it our goal to include at least one meeting per week. The meetings included a dedicated recap period for everyone to discuss developments and issues they encountered the previous week.

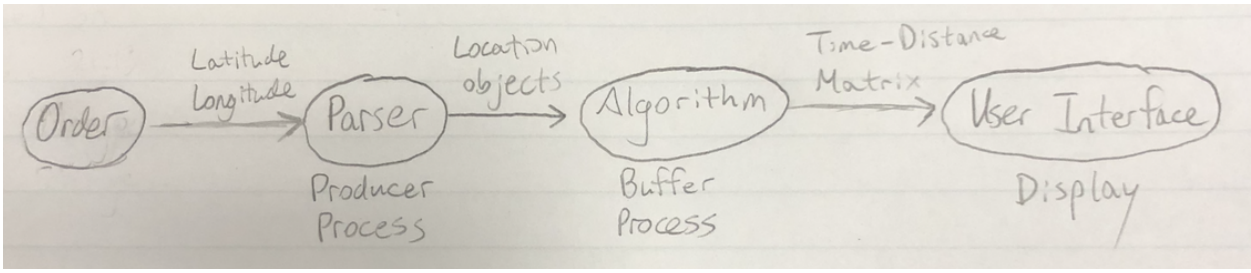
Advanced Software Engineering Topics

- **Real-time Software Engineering**

Architectural patterns for Real Time Software Engineering discovered through analysis of our system.

1. *Observe and React*
2. *Environmental Control*
3. *Process Pipeline*

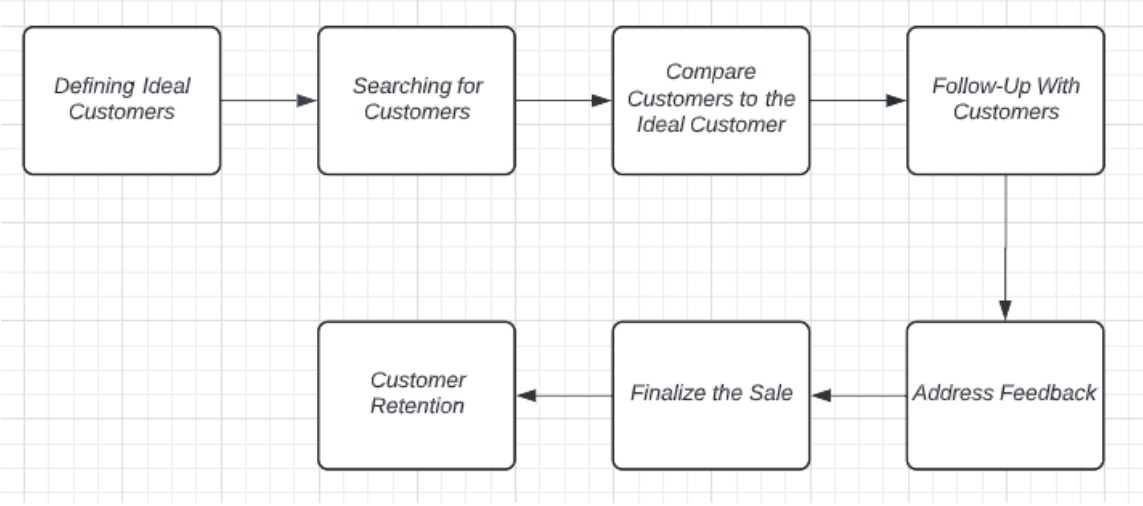
Process Pipeline Process Structure based on figure 21.13:



The above diagram shows how the process is actually a real time pipeline of data that originates at the user ends and ends at the user interface display. We identified the time sensitive components to be the parser and the algorithm.

- **Software as a service**

We identified possible scalability of the application to include software as a service techniques and considerations. These can be found in our software as a service plan, but largely resulted in the identification of our process for implementing this plan for addressing customer feedback.



Testing Process

- **Testings Documents**

From:	To:	Google Distance	Freshtime Distance
40.62268264,-74.18296424	40.62268264,-74.12227135	19 minutes	10.8 minutes
40.62268264,-74.09728016	40.62268264,-74.12941169	8 minutes	6.28 minutes
40.62268264,-74.05086795	40.62268264,-74.03658727	6 minutes	4.99 minutes
40.62268264,-73.96161370	40.62566009,-74.18296424	35 minutes	31.51 minutes
40.62566009,-74.17225373	40.62566009,-74.14012220	8 minutes	7.33 minutes
40.62566009,-74.11870118	40.62566009,-74.09370999	7 minutes	5.21 minutes
40.62566009,-74.06514863	40.62566009,-74.05800829	5 minutes	1.67 minutes
40.62566009,-74.03658727	40.62566009,-73.98660489	16 minutes	8.58 minutes
40.62863754,-74.18653441	40.62566009,-73.96161370	28 minutes	30.44 minutes
40.62863754,-74.17582390	40.62863754,-74.15440288	5 minutes	5.53 minutes
40.62863754,-74.13655203	40.62863754,-74.09013982	11 minutes	9.05 minutes
40.62863754,-74.07228897	40.62863754,-74.00445574	17 minutes	14.69 minutes
40.62863754,-73.99017506	40.62863754,-73.96875404	10 minutes	7.02 minutes
40.63161499,-74.19010458	40.63161499,-74.17225373	4 minutes	3.60 minutes
40.63161499,-74.11513101	40.63161499,-74.08656965	8 minutes	5.80 minutes
40.63161499,-74.07228897	40.63161499,-74.05086795	19 minutes	17.11 minutes
40.63161499,-74.02230659	40.63161499,-74.01159608	5 minutes	2.00 minutes
40.63161499,-73.98660489	40.63161499,-73.96518387	10 minutes	5.14 minutes
40.63459244,-74.17225373	40.63459244,-74.14726254	8 minutes	5.32 minutes
40.63459244,-74.10799067	40.63459244,-74.02944693	20 minutes	18.44 minutes

The above testing document was curated from the data points which represent the area of service in Jersey City which was calculated from our excel algorithm. These points were input into google maps and the distance between them was required in minutes. These values were then compared to the values returned from parsing the distance matrix returned from Open Route Service. These tests prove that our middleware for interfacing with Open Route Service works as well as validates the accuracy of Open Route Service and our choice to use it as our chosen routing API service.

- **User interface document**

MainView.fxml

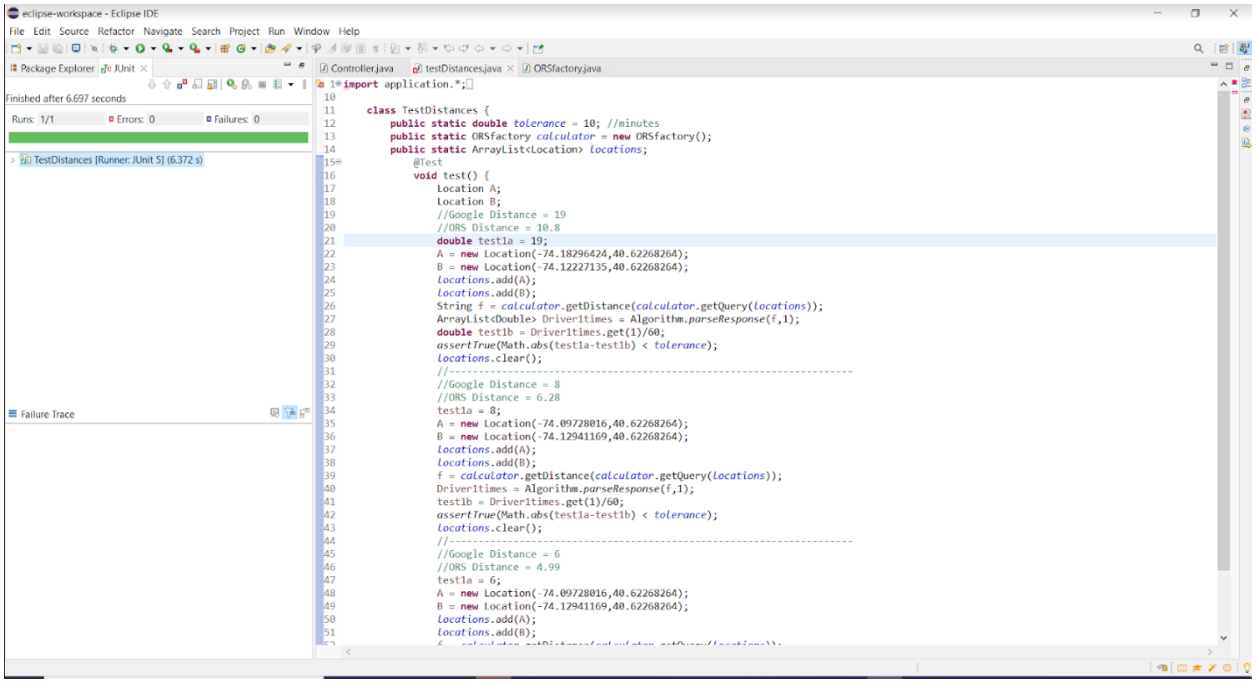
Test Case #	Requirement	Test Description	Expected Output
1	This method shall not accept an empty string	- Empty Connection String - Test data: empty string	Error Message: Please enter a MongoDB Cluster Connection String
2	This method shall accept a valid MongoDB Cluster Connection string	- Valid Connection String - Test data: enter valid MongoDB Cluster Connection string	Confirmation: Valid string
3	This method shall not accept an incorrect MongoDB Cluster Connection string	- Invalid Connection String - Test data: enter invalid Connection string	Error Message: Please enter a valid MongoDB Cluster Connection String
4	This method shall not accept an empty CSV file	- Empty CSV file - Test data: upload empty CSV file	Error Message: Please upload a CSV file
5	This method shall accept a valid CSV file	- Valid CSV file - Test data: upload	Confirmation: Valid CSV file
6	This method shall not accept a file that is not comma-separated values.	- Invalid CSV file - Test data: upload a file that is not comma-separated values	Error Message: Please upload a valid CSV file

FileView.fxml

Test Case #	Requirement	Test Description	Expected Output
1	This method shall not accept an empty restaurant file	- Empty restaurant file - Test data: empty restaurant file	Error message: Please enter a restaurant file
2	This method shall accept a valid restaurant file	- Valid restaurant file - Test data: enter valid restaurant file	Confirmation: Valid restaurant file
3	This method shall not accept a file that is not a restaurant file	- Invalid restaurant file - Test data: upload a file that is not a restaurant file	Error Message: Please upload a valid restaurant file
4	This method shall not accept an empty Area of Service file	- Empty Area of Service file - Test Data: empty Area of Service file	Error message: Please enter a Area of Service file
5	This method shall accept a valid Area of Service file	- Valid Area of Service file - Test Data: enter valid Area of Service file	Confirmation: Valid Area of Service file
6	This method shall not accept a file that is not a Area of Service	- Invalid Area of Service file - Test data: upload a file that is not a Area of Service file	Error Message: Please upload a valid Area of Service file
7	This method shall not allow a user to run Mapping Algorithm with an empty or incorrect Restaurant file	- Empty or Invalid Restaurant file - Test data: Upload empty or invalid Restaurant file	Error message: Please upload a valid Restaurant file before Mapping Algorithm
8	This method shall not allow a user to run Mapping Algorithm with an empty or incorrect Area of Service file	- Empty or Invalid Area of Service file - Test data: Upload empty or invalid Area of Service File	Error Message: Please upload a valid Area of Service file before Mapping Algorithm
9	This method shall allow a user to run Mapping Algorithm with a valid Restaurant file	- Valid Restaurant file - Test data: Upload a valid Restaraunt file	Confirmation: Valid restaurant file uploaded
10	This method shall allow a user to run Mapping Algorithm with a valid Area of Service file	- Valid Area of Service file - Test data: Upload a valid Area of Service file	Confirmation: Valid Area of Service file uploaded

This above document was created to test the GUI component of FreshTime. The purpose of this test is to ensure the functionalities of the program and ensure it fulfills all the use cases.

- **JUnit tests**



This above screenshot of the JUnit tests was conducted to test Rob's getDistance method of the ORSfactory class which passed with a tolerance of 10 minutes.

Evaluation of Processes

The team agrees our organization processes worked much better this time than that of the minor project. This is because we started early on with a plan that included subtasks required to complete each individual component's functionality. Focussing on identifying the correct use cases and accompanying that with a software specification document, again early in development, proved to be justifiable, even in the presence of frequent changes, because we were able to keep a clear goal and vision of what the final application behavior would look like. Finally, this project we focussed on the future of our application more. This included researching how we could build our software with real time systems engineering in mind. This proved to build a strong foundational understanding of what our goal was, and a clear understanding of the system we are building.

Final Contributions:

- Robert Reid
    - Set up the initial file structure and identified the components from the last project that would be reused. Authored the software reuse documents, software quality assurance plan, the project plan, and several system modeling and architecture diagrams. Was initially responsible for programming the general algorithm, the middleware interface for the routing service API, but ended up also being responsible for the user interface, as well as all of the accompanying documentation. Led the project meetings, and kept track of the weekly sprint logs that detail work done each week.
- Kiernan King
    - Kiernan and Rob helped develop a majority of the content for this project. Was in charge of everything database-related (MongoDB), generated a variety of diagrams (system modeling, architecture, etc.) and documentation, updated necessary documents, and helped keep track of weekly sprints and the notes detailing what was done each week. Also focused on the software reuse (from the minor project) in pertinence to the database, and was in charge of creating comments for and generating the Javadoc. Took time to teach Danny how to use JavaFX and create a user interface testing document.
- Brian Jimenez
    - Detailed the software quality assurance plan and drew up the Generalized Review Process diagram. Conducted research on the future possibility of providing our Software as a Service. Formed the SaaS plan including the Customer Interaction Plan Diagram. Gathered testing data points for restaurants in Jersey City and generated an interactable and customizable excel sheet where users could input restaurant Lat-Long coordinates and auto generate 4900 potential customer test coordinates to then input into our software and verify reliability.
- Danny Mitchinson
    - Was initially responsible for doing research on supporting more databases as well as other documentation and system modeling tasks. Completed a business process diagram, and I curated the initial testing data set from google maps that would be used to validate the result from Rob's ORSfactory class. Automated these tests by using the Junit 4 framework in eclipse. Authored a user interface testing document that tests the functionality of the FreshTime user interface.
- Wiktor Federowiat
    - Was initially responsible for research on databases and programming the parser algorithm. Created initial version of conceptual and use case diagram. Generated a javadoc based on Kiernan and Rob's javadoc comments. Researched and developed a process pipeline diagram and the real time system analysis document.