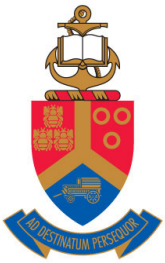

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

COS 301 Software Engineering
Testing the implementation of Broadsword: Navigation

May 5, 2017

Longsword Navigation

Duart Breedt - u15054692
Marin Peroski - u13242475
Nathan Ngobale - u15110045
Azhar Patel - u15052592
Matthew Perry - u13017030
Neo Thokoa - u14163285

GitHub Repository:
`https://github.com/perrymatthew/cos301_Longsword_Navigation`

Contents

1 Non-Functional Requirements **3**

2 Test Models **5**
2.1 Black Box Test Model 5
2.2 White Box Test Model 5

3 Service Contracts **7**

1 Non-Functional Requirements

- **Maintainability:**
  Coding standards were upheld and the code was well documented. However, the lack of compartmentalization of functionality may prove problematic when removing, adding, or editing program features. Furthermore, the location data is kept in a single file with the program therefore if a location should be added it cannot be added with a database query but had to be done manually.
  Mark: 6/10

- **Scalability:**
  MongoDB as a choice of database management is favourable especially with regards scalability. However, the location data is not stored in a MongoDB database but rather in the same navigationLocal.js file as the entire program. This will cause a bottleneck when the amount of location data is increased.
  Mark: 7/10

- **Accessibility:**
  All functions allowed other modules to request information as well as get a result returned if needed. The scope of the functions were adequate, allowing other modules to call needed functions. However, the scope was broad and did allow access functions which other modules should not have access to.
  Mark: 8/10

- **Security:**
  Since the server is not a relational database, and it does not use "get" requests to and from clients. The security is inadvertently good.
  Mark: 9/10

- **Performance:**
  The use of MongoDB and not the use of a relational database improves the efficiency and reduces the overhead of reading from and writing to the database. The functions are designed to be called once per request per user.
  Mark: 8/10

- **Coverage:** The module covers the core cases required. This enhances the non-functional requirements of performance, stability and maintainability whilst giving a good base for future extensibility. However, security cases are lacking and could be improved to prevent exploits.
  Mark: 7/10

- **Efficiency:** Functionality could be extended (by use of more test cases) to enhance the efficiency of this module. The module is effective as it stands, but resources such as time and other technologies could have been used

more effectively to enhance the non-functional requirement of compliance.
Mark: 7/10

2 Test Models

In order to test the broadsword navigation teams code two test models were used, the Black Box and White Box testing models. These two models allow for a fully comprehensive testing environment as each will cover different aspects of depth in the program. To start, the Black Box testing model will be used. With this testing model method the program will be run without analysing at any of the code and the output will be evaluated based on what was expected from the navigation module. After this step the White Box testing model will be used. In this model the code will be examined thoroughly and compared to the corresponding output received when running the program in the Black Box test. With this testing model in mind different aspects of the program can be tested for as each looks at the expected output from different perspectives.

2.1 Black Box Test Model

The steps that were taken in order to get the code running were as follows:

1. Install Node, MongoDB and NSQ
2. Update the Node Package Manager to the latest version
3. Run the code located in the root folder

After the code was run all of the JSON string variables were displayed on the terminal. The output showed all of the routes that were added to the mongo database along with various text describing interactions between the java script code, node and mongo db. At this current stage without having analysed the code, the output seems very cluttered, unformatted and it was difficult to identify major points from all of the text (excluding the JSON output). Focusing on the JSON string outputs however, everything seemed to be in order with regards to what is expected from the navigation module. The script ran through once and was then terminated thus, there was no persistent server (apart from the mongo database storage where the routes are stored and edited).

2.2 White Box Test Model

The code located inside of the navigationLocal.js file was examined in order to determine the correctness of the output that was displayed in the Black Box test. At the very top of the script there were a set of global variables that allowed a connection to be formed to mongo DB however, there was no variable for any NSQ connection. A global data variable was defined which contained a hardcoded JSON formatted string of routes. This variable was then converted into a string and parsed into a JSON object which was used in all of the functions.

The code covered the following functionality:

1. Adding a route

2. Adding user preferences
3. Updating user preferences
4. Calculating the distance between two points
5. Finding a users preferences
6. Finding a route that was added
7. Removing user preferences
8. Removing a route
9. Establishing a connection the a persistent database (Mongo DB in this case)

Taking a second look at the text output that was received from the previous test model phase, the output correlates to each of the functions execution and outcome (connections established, routes added and removed etc). There seem to be no logical errors and the code preforms its required actions which correlate to the output.

3 Service Contracts

- **Save User Route:**
  In this service contract, the code does obliged to all requested pertained in the conditions set out for this service contract. However the code does miss out on catering for the condition not stipulated which is the start saving a start point that is the same location as the end point.

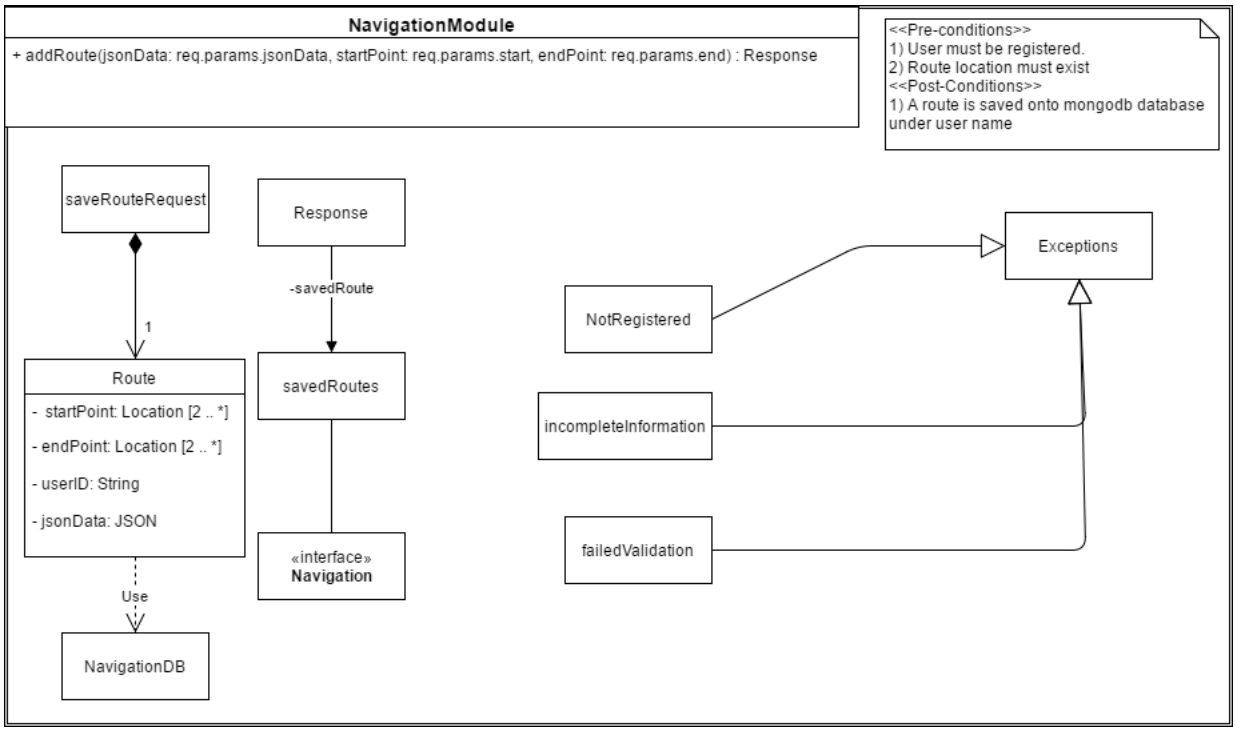

Figure 1: Service Contract: Save User Route

Mark: 8/10

- **Save User Preference:**
  Service contract requires to save different preferences to a specific user when all Pre-Conditions are met. In this test, A discovery was made whereby the request overrides already existing preferences instead of appending new preferences or throw an exception if preference is used. This thus leads to it not fully fulfilling the service contract stipulated.

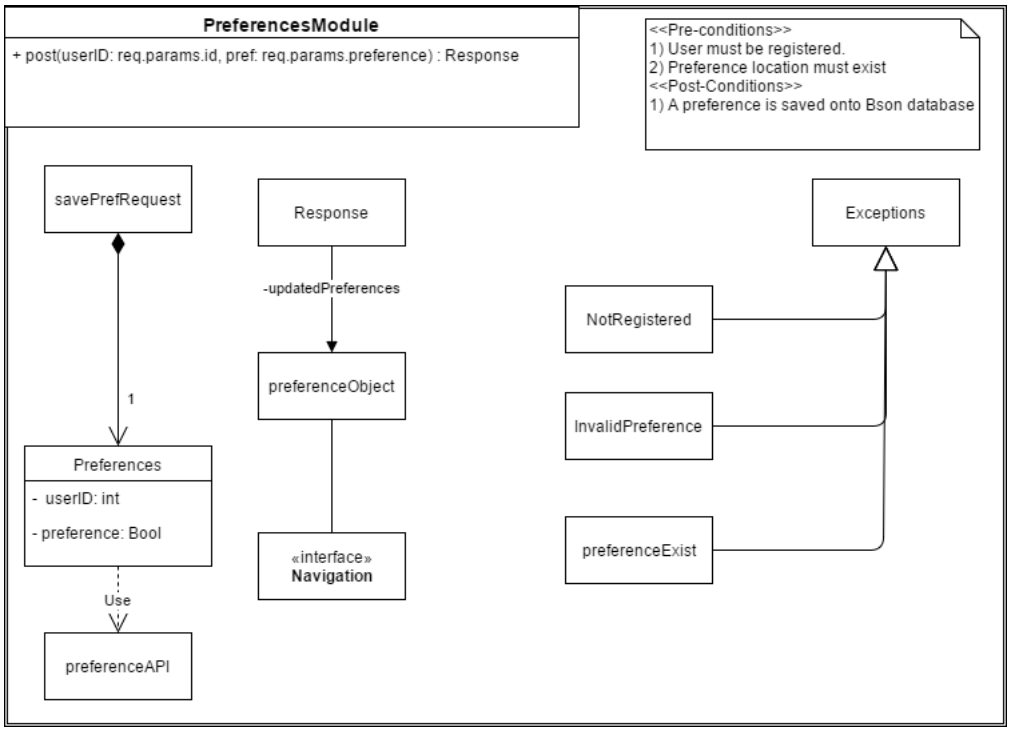

Figure 2: Service Contract: Save User Preference

Mark: 4/10

- **Cache Routes:**
  In this service contract, routes that was saved onto cache is retrieved if route request exists in cache. In all test cases, the cache route is retrieved and if not, a request is made, this fulfils the service contract post condition. To thoroughly validate, we used Postman as an additional third party to validate POST request on database and cached file.

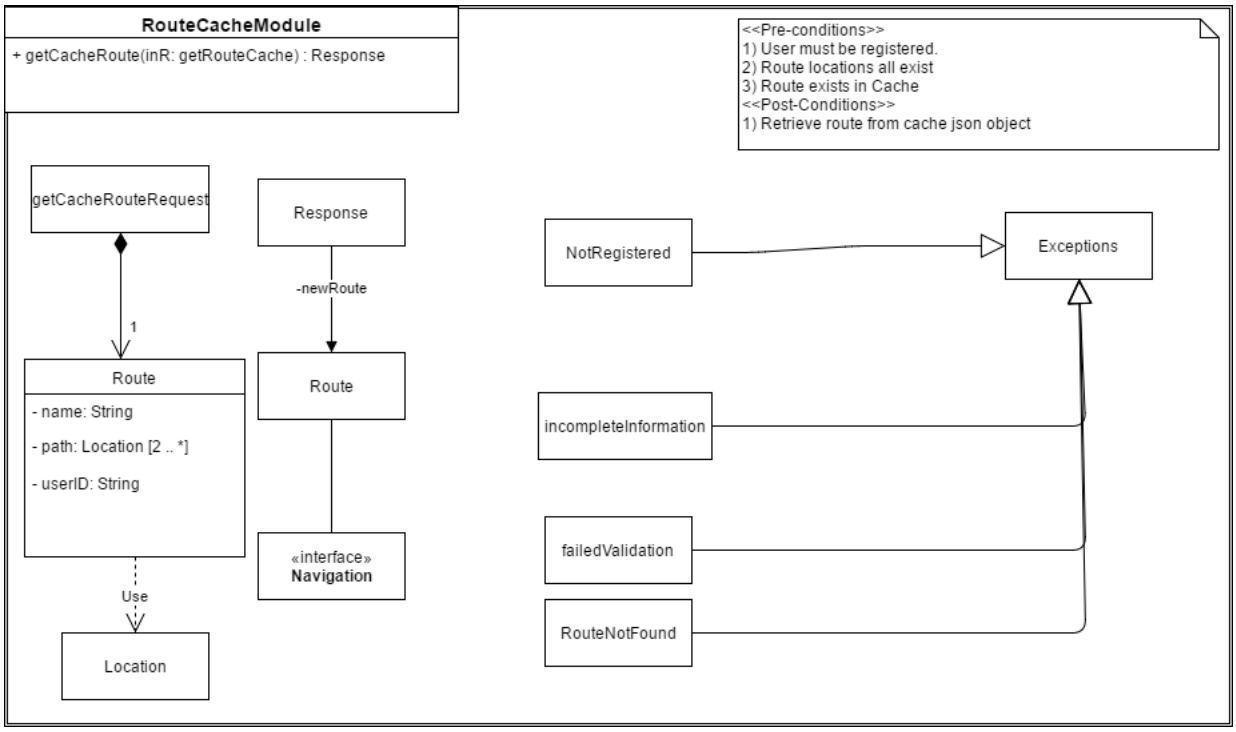

Figure 3: Service Contract: Get Cache Route

Mark: 9/10

- **Route Module:**

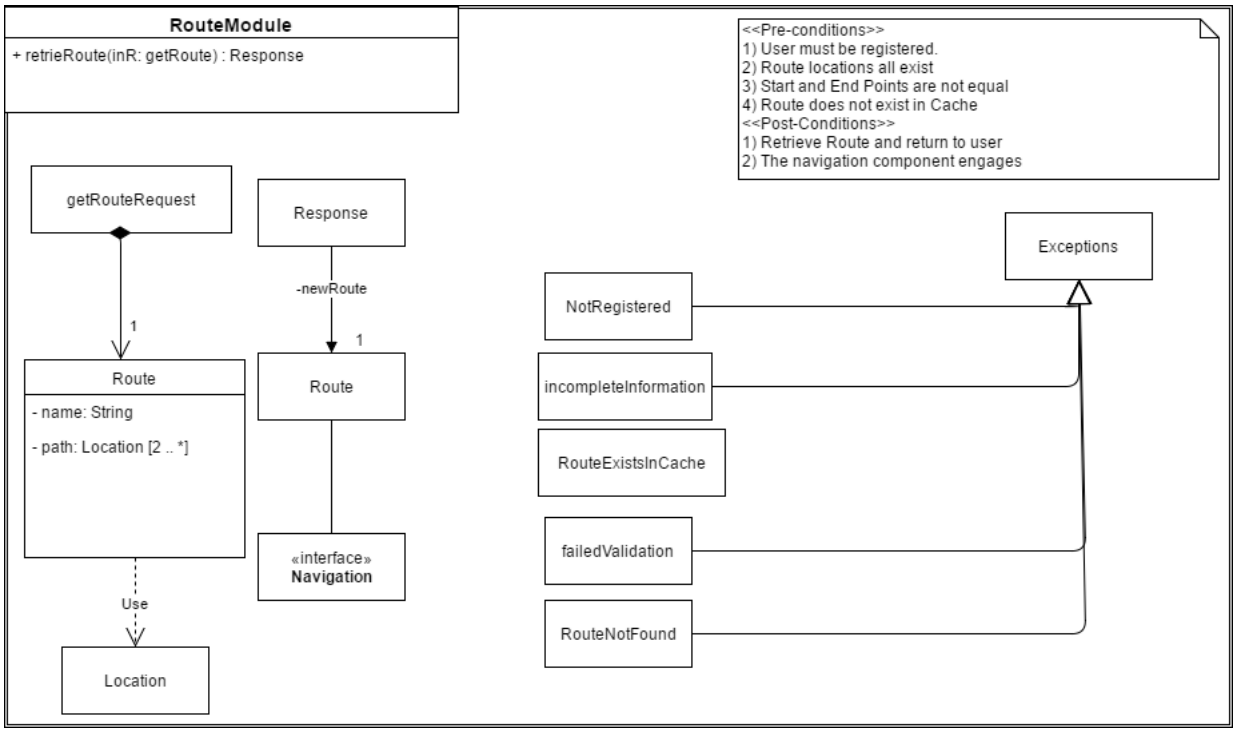

Figure 4: Service Contract: Route Module

Mark: 8/10

- **Add Route:**

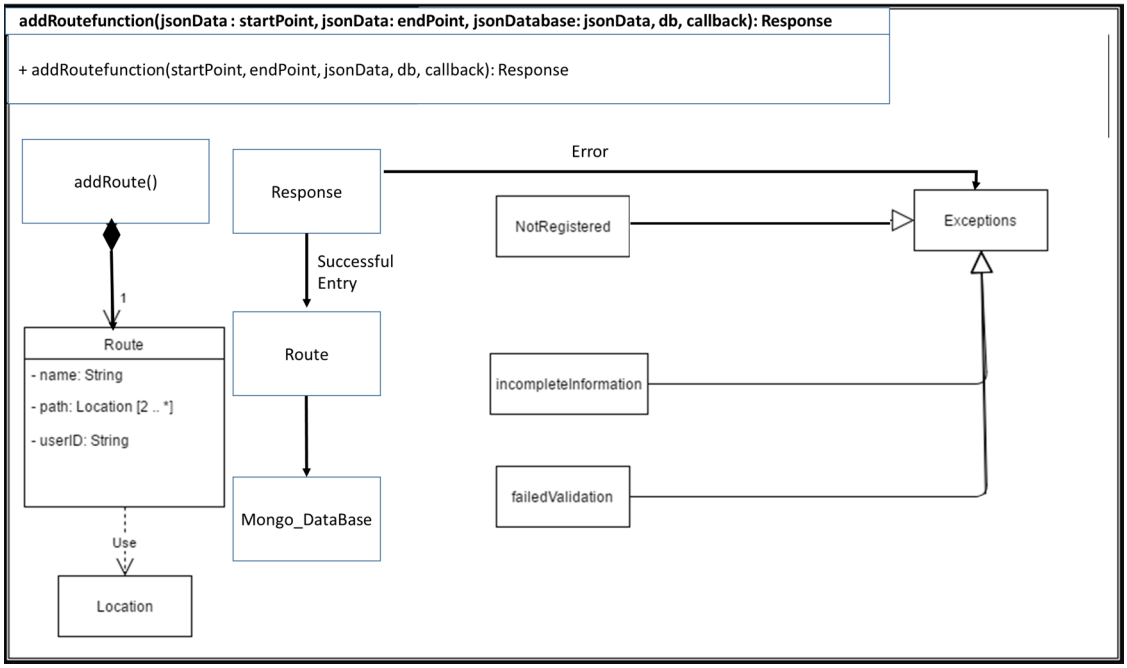

Figure 5: Service Contract: Add Route

Mark: 8/10