WEB ROUTING

Testing Manual

SLIPPERY ROCK UNIVERSITY

Thomas Haley tjh1019@sru.edu

Sinchana Kori ssk1002@sru.edu

Beth Orgovan bro0700@sru.edu

Dakota Myers drm1022@sru.edu

Table of Contents

[1.Overview 2](#_Toc2064857988)

[2. Testing 2](#_Toc874191750)

[2.1 HUMAN-COMPUTER TESTING 2](#_Toc578945290)

[2.2 WHITEBOX TESTING 2](#_Toc1167346581)

[2.3 UNIT TESTING 2](#_Toc1728354369)

[2.4 Integration testing 2](#_Toc1066410832)

[2.5 Regression testing 3](#_Toc1710353202)

[2.6 Stress/Parallel testing 3](#_Toc45055142)

[2.7 Boundry Analysis 3](#_Toc659369625)

[2.8 Blackbox Testing 3](#_Toc81989225)

[2.8 Crash Recovery 3](#_Toc77198090)

[2.3.1 Running Tests 3](#_Toc807341270)

[2.4 UNIMPLEMENTED TESTS 5](#_Toc1934308762)

[3. Figures 5](#_Toc1132557969)

# 1.Overview

This manual serves as a comprehensive guide to the testing framework of the Webrouting application. It provides detailed instructions on how to execute the tests, the expected results, and a summary of non-programmatic tests conducted on the application. Furthermore, the final section of this manual outlines additional tests that could be implemented into the application to bolster its stability and performance.

# 2. Testing

Section 2 details the different methods of testing that were employed during our development cycle.

2.1 HUMAN-COMPUTER TESTING

We conducted usability testing, also known as human-computer testing, extensively during the ongoing development of the Webrouting Application. Our objective was to identify any usability issues, enhance user experience, detect bugs, validate assumptions, and ensure accessibility. To achieve this, we enlisted the assistance of friends and family members to interact with the system. This approach proved to be highly effective as the users had no preconceived notions or expectations for how the system should work. By doing so, we were able to gain invaluable insights into the user experience, allowing us to refine and optimize the application to provide an enhanced user experience.

2.2 WHITEBOX TESTING

After completing a new system, we conducted whitebox testing to ensure that the system interacted with external entities. Our primary focus during whitebox testing was to ensure the quality, security, and performance of the application. This rigorous testing approach facilitated the early detection of defects such as performance issues, code optimization, security flaws, and comprehensive internal problems such as resource paths and boundary analysis.

2.3 UNIT TESTING

Prior to this update, the Webrouting Application lacked a comprehensive testing framework. To address this, we placed a strong emphasis on testing throughout the project, resulting in the creation of numerous comprehensive tests to verify the efficiency of the program's methods. To achieve this, we implemented Junit 5 into the program, which proved invaluable in facilitating our testing efforts. Additionally, we overcame the challenge of the deprecation of the "run all tests" method in Junit 5 by devising a solution that allows all tests to be run simultaneously by right clicking the src/test/java folder in the project and selecting "Run As" followed by "Junit". With these measures in place, we have been able to ensure the robustness and reliability of the application, providing superior user experience.

2.4 Integration testing

To ensure seamless integration between the various components of the system, we have implemented an integration testing framework in the form of a simulation. This simulation allows for the simultaneous testing of multiple system components, including the Shipments, Bid, and Notification systems. Accessible via the "simulations" tab on the navbar, users can initiate the simulation by logging in as an auctioneer. It's important to note that the current iteration of the simulation is designed to test the integration of the auctioning components of the application, and not the routing component. With this testing approach in place, we can confidently verify that the different components of the system interact seamlessly, delivering an exceptional user experience.

2.5 Regression testing

We implemented a weekly regression testing process as a crucial step in ensuring the stability and reliability of the application. After each new update was merged into the application, each team member was assigned a set of specific systems to test. Our objective was to ensure that all systems performed as expected, regardless of whether changes were made to those systems or not. By adopting this approach, we were able to identify and resolve unexpected errors before they had the chance to escalate into more significant issues, preserving the coherent functionality of the application.

2.6 Stress/Parallel testing

Testing was done to ensure the application works smoothly when multiple users are interacting with the system across multiple computers. The application was hosted on a laptop that was attached via ethernet to the network switch in the robotics lab. All of the other computers in the robolab were then connected to the laptop via its IP in the browser, and our group set about interacting with the application across multiple machines at once. It was shown that the application is stable when many computers are interacting with it at once. This was not a “true” stress test as many more computers would need to be involved. This could potentially be done with virtual machines.

2.7 Boundry Analysis

Throughout the project, we executed a significant overhaul of the data-sanitization component of the application. Our team conducted thorough testing to identify and address any potential boundary issues that could impact the user experience. As part of these efforts, we introduced various measures designed to prevent users from entering bids that exceed a certain limit, input strings that are too long, or those that contain prohibited characters. These safeguards were made possible through the implementation of advanced regex pattern matching techniques, which were integrated into the validationService class.

2.8 Blackbox Testing

We conducted blackbox testing on both the user interface and application programmer's interface calls. By doing so, we enhanced the performance of the application while ensuring that all exceptions were handled appropriately. For instance, we used API calls to Google Maps, and in the event of an error or an unexpected response, the Webrouting Application was able to handle the issue, without causing any disruption.

2.8 Crash Recovery

The application has been thoughtfully designed to handle server-side crashes. Springboot, operating in the background, terminates any POST or GET request that triggers a crash. In response, the template renderer presents a standardized "error" page (the Gazebo Page) to the user. Once the error is resolved, the application seamlessly recovers, and users can continue to use the application without any issues. This proactive approach to error handling minimizes disruptions, enabling a smooth and uninterrupted user experience.

***Tests written:***

* ReceiveCaptchaTest
* RestConfigTest
* All edu.sru.thangiah.webrouting.domain classes (Exception: VehicleTypesTest)
* EmailingImplTest
* UserRepositoryTest
* UserServiceImplTest
* UserValidatorTest
* Simulation feature for Auctioneer

***Note: Approximately 58% of all tests were written.***

### 2.3.1 Running Tests

This section highlights running tests in the **Webrouting Application**, to run a single test class, select a test class in the ***src/test/java*** folder of the project, once finding a test class you’d like to run right click on the class and select ***Run As***, next you will select ***JUnit Test*** (Figure 1 & Figure 2).

A screenshot of a computer

Description automatically generated with medium confidence

Figure 1 – Run As Button

Text

Description automatically generated

Figure 2 – Select Junit Test

If you wish to run all available tests, right click on the ***src/test/java*** folder of the Webrouting project and repeat the same steps for running single tests(Figures 1 & 2). If done correctly you will see the console start to load and process at the bottom of the screen, once finished you will see a Junit tab. To review the tests, click the Junit tab (Figure 3).

Graphical user interface, text, application, Teams

Description automatically generated

Figure 3 – Junit Tab

2.4 UNIMPLEMENTED TESTS

Due to some issues, with getting tests to run and applying values to those, not all tests were finished. However, in aspects of testing the application has seen great improvement. Before this update the application had one test class which was the contacts controller class. It wasn’t a real test class despite that, tests were implemented as time allowed. However, this was not complete. Below are the tests not written:

* RecaptchaTest
* All edu.sru.thangiah.webrouting.controller classes
* VehicleTypesTest
* MailSendingTest
* PopulateDatabaseTest
* MyAccessDeniedHandlerTest
* SecurityServiceImplTest
* UserDetailsServiceImpl

# 3. Figures

Figure 1 – Run As Button 3

Figure 2 – Select Junit Test 4

Figure 3 – Junit Tab 4