WEB ROUTING

Final Report

SLIPPERY ROCK UNIVERSITY

Thomas Haley – [tjh1019@sru.edu](mailto:tjh1019@sru.edu)

Beth Orgovan – [bro0700@sru.edu](mailto:bro0700@sru.edu)

Dakota Myers – [drm1022@sru.edu](mailto:drm1022@sru.edu)

Sinchana Kori – [ssk1002@sru.edu](mailto:ssk1002@sru.edu)

Table of Contents

[1. problem identification 3](#_Toc133355926)

[2. Contribution 3](#_Toc133355927)

[2.1 Data Sanitization 4](#_Toc133355928)

[2.2 Database Backup 4](#_Toc133355929)

[2.3 Shipment Direct Assignment 4](#_Toc133355930)

[2.3 Navigation Bar 4](#_Toc133355931)

[2.4 Auctioning System 4](#_Toc133355932)

[2.5 Freight Rate Table 5](#_Toc133355933)

[2.6 Simulations 5](#_Toc133355934)

[2.7 Google Maps API Integration 5](#_Toc133355935)

[2.8 Log4j 5](#_Toc133355936)

[2.9 Importing and Exporting 5](#_Toc133355937)

[2.10 Shadow Admin 5](#_Toc133355938)

[2.11 Notification System 6](#_Toc133355939)

[2.12 Other Contribution 6](#_Toc133355940)

[3. Complete or Incomplete 6](#_Toc133355941)

[4. File Path names 6](#_Toc133355942)

[5. System & Hardware Requirements 7](#_Toc133355943)

[5.1 System Requirements 7](#_Toc133355944)

[5.2 hardware requirements 7](#_Toc133355945)

[6. setup and installation 7](#_Toc133355946)

[7. Caveats/Minefields 8](#_Toc133355947)

[8. Deployment and Maintenance 8](#_Toc133355948)

[9. code reusability 9](#_Toc133355949)

[9.1 Navigation Bar 9](#_Toc133355950)

[9.2 Templating 9](#_Toc133355951)

[9.3 Services 9](#_Toc133355952)

[9.4 Style sheets 9](#_Toc133355953)

[10. Testing 9](#_Toc133355954)

[10.1 HUMAN-COMPUTER TESTING 10](#_Toc133355955)

[10.2 WHITEBOX TESTING 10](#_Toc133355956)

[10.3 UNIT TESTING 10](#_Toc133355957)

[10.4 INTEGRATION TESTING 10](#_Toc133355958)

[10.5 REGRESSION TESTING 11](#_Toc133355959)

[10.6 STRESS/PARALLEL TESTING 11](#_Toc133355960)

[10.7 BOUNDRY ANALYSIS 11](#_Toc133355961)

[10.8 BLACKBOX TESTING 11](#_Toc133355962)

[11. POST-MORTEM ANALYSIS 11](#_Toc133355963)

[12. UML Diagrams 12](#_Toc133355964)

[12.1 Class Diagrams 12](#_Toc133355965)

[12.2 Sequence diagram 14](#_Toc133355966)

[12.3 Case diagrams 19](#_Toc133355967)

[12.4 Activity diagrams 23](#_Toc133355968)

[12.5 State chart diagram 26](#_Toc133355969)

[REREFERNCES 27](#_Toc133355970)

# 1. problem identification

Traditionally in the shipping industry, the process of sending a shipment to its destination is complex and involves multiple systems interacting across companies. SHIPPERS, who want to ship products to their consumers, typically outsource the shipping process to CARRIERS, who manage fleets of vehicles and handle the process of getting the products to their destination. Examples of SHIPPERS include Amazon, Walmart, and Ebay while examples of CARRIERS include FedEx, UPS, and other contracted carrier companies.

The Web Routing and Auctioning system aims to provide a universal shipment management platform to facilitate communication between SHIPPERS and CARRIERS. This system includes two major aspects: assigning shipments to carriers at appropriate prices and routing those shipments to their destination once assigned. Prices for shipments are determined either by **Auction**, **Freight Rate**, or manual input by the shipper. Routing is handled on the carrier end via Google Maps API. The application also includes Fleet Management tools for Carriers (such as the ability to track all their vehicles and assign Maintenance orders to them) as well as a robust notification/emailing system that will alert both Shippers and Carriers of any status updates to the shipments.

# 2. Contribution

This Section details our group's specific contributions to the project relative to what already existed when we received the project. The auctioning and routing functionality of the original application was extremely barebone, most of the current auctioning and routing systems were built by us.

Previously auctioning was limited to only the creation and accepting of bids, nothing more. We also built a direct assignment system, which allows users to bypass the auction and set shipment prices on their own. Along with this direct assignment system is the freight rate table for determining default prices, which we also built.

The routing system was also heavily reworked. The Google Maps API for routing was not integrated before, it routed just by linking the user to google maps and taking them away from the application entirely. In addition, there was no way to assign shipments to vehicles before we received the application. Shipments had to be pre-injected with a vehicle assigned to use the routing system. We have implemented a way to assign a shipment to the best vehicle based on distance.

Also, we have added several quality-of-life features. The notification system, for updating users as to the status of their current shipment, the database backup and reload system for admins, ShadowAdmin, and all its logging features, and a host of UI improvements and tweaks. A more detailed listing of all our contributions is listed below.

## 2.1 Data Sanitization

The validation service holds methods for taking in locations, vehicles, vehicle types, maintenance orders, technicians, drivers, contacts, and shipments. These methods will validate the objects and return the valid object or display an error message for what went wrong. The validation service holds methods for validating the edit forms and excel files as well.

## 2.2 Database Backup

On startup and every 20 minutes the database is saved into a database backup file that then can be used to reinstate the database at any point. Admins have this feature.

## 2.3 Shipment Direct Assignment

A standalone shipment assigning system that allows shippers to bypass the auctioning system and directly assign shipments to carriers of their choosing. Assigned carriers are sent a notification and an email asking whether they wish to approve or deny the request.

## 2.3 Navigation Bar

The navigation menu now loads from a JavaScript function, which allowed for a huge amount of HTML code to be deleted. It also now has dropdown submenus included in it.

## 2.4 Auctioning System

Shipments can be pushed to auction which means they become available for carriers to bid on them. The carrier with the lowest bid wins. When the shipper accepts the bid, it attaches the winning carrier to the shipment.

## 2.5 Freight Rate Table

Shippers can set default price-per-mile multipliers for shipments. Also allows shippers to pre-generate prices for shipments when they are assigning shipments to carriers outside of the auction. Once a Freight Rate Table has been uploaded, the "Fetch Price from Freight Rate Table" button on the Direct Assignment screen will be enabled.

The system allows shippers to have multiple Freight Rate Tables to allow for different pricing for different carriers. When assigning a price to the shipment using a freight rate table, a query is made to Google Maps API to determine the distance between the shipment start and ending location. That distance is matched against the "Distance Breakpoints" in your freight rate table to determine a price per mile, and then the distance is multiplied by that rate. The resultant price will be automatically filled into the input box.

## 2.6 Simulations

There are two simulations that can be run by the admin. These simulate the direct assignment system and the auctioning system. They involve creating the entities that are required and then setting up the dependencies and moving the entities around into various states of the application. In the end, the entities created are deleted. The console prints out the various actions that were taken during the simulation.

## 2.7 Google Maps API Integration

When creating shipments and locations a request is sent to google maps to find the longitude and latitude of the city and state. This is then attached to location and shipment. When displaying the routes that vehicles are going to take to deliver their shipments a request is sent to google maps to find fastest route by road. When choosing which vehicle to select for assigning it to a shipment, shippers may find the closest vehicle. This sends a request to google maps to find which vehicle is closest to the shipment. The routes page has an embedded map which displays routes.

## 2.8 Log4j

Every controller has an instance of a Log4j object used to log the various database interactions throughout the controller. Logs are written to the console and to a log file.

## 2.9 Importing and Exporting

Implemented shippers and carriers' ability to export and import all their data via Excel. Shadow Admins may export logs. Shippers may import a freight rate table that allows them to fetch a price for every carrier that they interact with.

## 2.10 Shadow Admin

The Shadow Admin can view all database interactions and user login/logout activity. This is made available through the log page. Logs can be filtered based on the User, Level, and Date Range. Shadow Admins may also export the logs via excel.

## 2.11 Notification System

Every role has a notification system. Shippers get notified when there is an interaction with their shipments. Carriers get notified when there is an interaction with a shipment they bid on. Users can see their notifications by traveling to the notification tab on the navigation bar. There they may mark notifications as read.

## 2.12 Other Contribution

1. Auctioneer – changed the name from Master to Auctioneer
   1. Freeze / Unfreeze shipments.
   2. Add / Edit / delete bids.
   3. Remove and push shipments to auction.
   4. Direct Assign Shipments
   5. View all shipment details.
2. Admins can toggle auctioning for shippers and carriers.
3. Implemented HTML sessions to hold errors and redirect locations.
4. Nested Back buttons
5. Carriers can no longer delete other carrier’s bids.
6. Attach vehicles to shipments.
   1. Can select the best vehicle based on distance.
7. Visual Changes
   1. Icons within the tables
   2. Carrier info separated into different columns.
   3. Username, user account icon, alert icon, logout icon moved to top right corner of all pages.
   4. Make edit screens smoother with dropdown bars.
8. Edit pages were reworked.
9. Add pages were reworked.
10. Split the user tables into separate columns for shippers and carriers.
11. Added a function so that shippers can remove their own shipments from auction.
12. Created the All-Shipments Page for MASTER, CARRIER, SHIPPER
13. Removed Slippery Rock references.
14. Confirmation pages replaced with popups using a JavaScript/ajax request.
15. Reworked the Navigation menu.

# 3. Complete or Incomplete

This application is not complete in its entirety. There are standing requirements that have not been met.

# 4. File Path Names

JAVA DOCS - webrouting/Documents/javadocs

MANUALS – webrouting/Documents

OLD DOCUMENTS – webrouting/Documents/OLD

EXCEL FILES – webrouting/Documents/Program Documents

BACKUPS – webrouting/backups

LOGS – webrouting/WebroutingApplication.log

# 5. System & Hardware Requirements

The following sections cover the system and hardware requirements.

5.1 System Requirements

* **Operating System**
* **Required:** Windows
* **Version:** 10
  + - **Link:** https://www.microsoft.com/en-us/software-download/windows10
* **IDE**
* **Required:** Eclipse IDE
* **Version:** 2022-12
  + - **Link:** [Eclipse downloads - Select a mirror | The Eclipse Foundation](https://www.eclipse.org/downloads/download.php?file=/oomph/epp/2021-12/R/eclipse-inst-jre-win64.exe)
* **Database**
* **Required:** MySQL Server & Workbench
* **Version:** 8.0.33
  + - **Link:** [MySQL :: Download MySQL Workbench](https://dev.mysql.com/downloads/workbench/)
* Java
  + **Required:** Version 17
    - Link: [Java Archive Downloads - Java SE 17 (oracle.com)](https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html)
* **Internet Browser**
* **Required:** Any updated Internet Browser
* **Recommended:** Google Chrome
  + - **Link:** [Google Chrome - Download the Fast, Secure Browser from Google](https://www.google.com/chrome/dr/download/?brand=QMKX&geo=US&gclid=CjwKCAjwrpOiBhBVEiwA_473dJ8mM8hU5ugxCAjCm5yPuvtP5Uf5LCf6i9MFF12f40Cy1P204zDnaBoCiVsQAvD_BwE&gclsrc=aw.ds)

5.2 hardware requirements

* **CPU:** Multi-Core 64bit x86 architecture CPU
* **RAM:** 8 GB
* **Display:** 1024x768
* **Storage:** 50 GB

**\*NOTE: Metrics of given Hardware Requirements are for low-end optimal performance.**

# 6. setup and installation

To set up the Web Routing Auctioning System, it must be downloaded onto your machine/system and set up in an Eclipse workspace. To do this, follow this GitHub link: <https://github.com/samthangiah/Spring-2023-Web-Routing-Auctioning.git> and grab the URI from the **Code** tab.

Import the project with Git in Eclipse, entering in both the URI and the Personal Access Token on your GitHub. Branching can be done in the Eclipse IDE or from the GitHub website. This process is detailed further in the Install/Configuration Manual

# 7. Caveats/Minefields

The main caveat of the **Web Routing and Auctioning** system is data dependencies. The SQL database has been designed with an excessive number of foreign keys; every data field is tied to some other data field (often unnecessarily). The application was built like this in previous iterations. Vehicles cannot be added without first adding a “Vehicle Type.” Technicians and Drivers must be associated with a “contact” object first, it is unnecessarily complicated in its data relationships. Maintenance Orders are dependent on **seven** foreign keys. This causes many problems in practice with developing and using the application.

On the user end two major problems manifest because of this. Both problems primarily affect CARRIER account types. The first is the “cannot delete due to dependency conflict” error. The user cannot delete any object that another object is referenced by a foreign key. To delete any object the user must delete all its “dependencies” in a very specific order. To delete a contact for example one must first delete any Technicians associated with it, and to delete a technician you must delete any Maintenance Orders, etc. This is very inconvenient, and an easy solution is not present without redesigning the database.

In a similar vein, the CARRIER must create/upload fields in an extremely specific order to satisfy the foreign key constraints. The user must upload every other field before Maintenance Orders, as Maintenance

Orders are arbitrarily associated with every other field in the database. One must upload all their contacts before any Technicians/Drivers; it is very confusing. The order is detailed in the user manual, but that does not stop it from being a huge inconvenience.

On the development end, these problems are extremely hard to fix without rebuilding the entire database and making the relations between objects less convoluted. Moreover, this is detailed in the Post-Mortem analysis, but in short it would be a very time-consuming process to address this problem.

# 8. Deployment and Maintenance

To install the application, please refer to the installation manual provided. After successfully installing the application, it can be deployed across a network by connecting other computers to the same network as the host machine. Once connected, other users can access the application by navigating to http:// [the host's IP address]:8080, enabling simultaneous usage by multiple users in real-time.

To ensure the smooth operation of the application, it is essential to regularly archive the log files and database backup files. As a backup of the database is taken every 20 minutes, a considerable number of backup files can accumulate quickly. Furthermore, to prevent the log file from becoming too large, it automatically compresses itself after 1GB of data has been written to it. Thus, ensuring that adequate storage space is available to store these files in their storage location is crucial for maintaining the application's functionality.

# 9. code reusability

Code reusability is a fundamental principle of object-oriented programming, and it refers to the ability of software components to be used in multiple contexts without modification. Designing code that can be reused, reduces errors and increases the maintainability of code. A comprehensive list of our efforts to uphold this key principle of object-oriented coding is detailed in this section.

## 9.1 Navigation Bar

The navigation bar was converted into an HTML fragment loaded at the top of all the templates throughout the application. This design removed 11,000 lines of html code. The navigation bar can now be expanded with less effort.

## 9.2 Templating

The All-shipments page is rendered differently based on the role of the user accessing it.

For shippers, it includes all their shipments in any state. For carriers, it includes all shipments that are available to bid on and all the shipments that they have won. For auctioneers, it shows everyone's shipments in every state.

## 9.3 Services

The validation and user details service were deliberately crafted for use by controllers, with helper functions delegated to separate services to be utilized as required.

The validation methods are carefully structured to accommodate various entities, with specific methods for validating individual entities. This allows for seamless integration with various functions such as the edit, add and import functions.

## 9.4 Style sheets

By designing CSS classes to be reusable, we not only saved time but also increased the congruency of the user interface. When users navigate through a web application, they expect a consistent visual experience across different pages and components.

Using reusable CSS classes ensures that containers with similar purposes have the same styling, resulting in a more harmonious and professional-looking user interface. This approach also reduces the likelihood of inconsistencies in the visual design of the application, which can detract from the user experience.

# 10. Testing

Section 10 highlights the testing done within the system.

## 10.1 HUMAN-COMPUTER TESTING

We conducted usability testing, also known as human-computer testing, extensively during the ongoing development of the Web Routing Auctioning System.

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.

## 10.2 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.

## 10.3 UNIT TESTING

Prior to this update, the Web Routing Auctioning System 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.

## 10.4 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 is 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.

## 10.5 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.

## 10.6 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.

## 10.7 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.

## 10.8 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 Web Routing Auctioning System was able to handle the issue, without causing any disruption.

# 11. POST-MORTEM ANALYSIS

There would be many things that would be done differently, given the opportunity to do the project again from the start.

As stated in the Caveats section (section 5), the database was designed in an extremely convoluted manner in previous versions. Our group made the decision to not change the fundamental way the database was designed, and instead opted to work around it. In retrospect this was probably a mistake, we should have rebuilt the domain classes to be more sensible. As you can see from the class diagram below, the classes are associated in a tangled web of relationships rather than a linear tree, and essentially every class is in a convoluted relation with every other class. This caused many issues and lots of wasted time during development, and the excel-uploading system is still confusing for any user to use as a result. It would have been a good idea to spend the first few weeks redesigning the database.

Another thing that should have been done differently is our use of thymleaf. Going into this project none of our team had good knowledge of thymleaf and had to learn it on the fly. Had we known more about its templating features at the beginning we could have reduced the amount of bulk HTML code that is in the application.

A final thing that would have been done differently is our approach to writing the Excel Uploading. We went back and forth between having the Carrier upload all their data at once VS having them upload each of their data types individually. We went with the latter option; this was probably a bad idea. It leads to more confusion with uploading the files in a specific order.

# 12. UML Diagrams

This section will highlight UML diagrams conveying code pertinent to the Web Routing Auctioning System.

## 12.1 Class Diagrams

This section contains Class Diagrams of the Web Routing Auctioning System

Diagram, schematic

Description automatically generated

Diagram, schematic

Description automatically generated

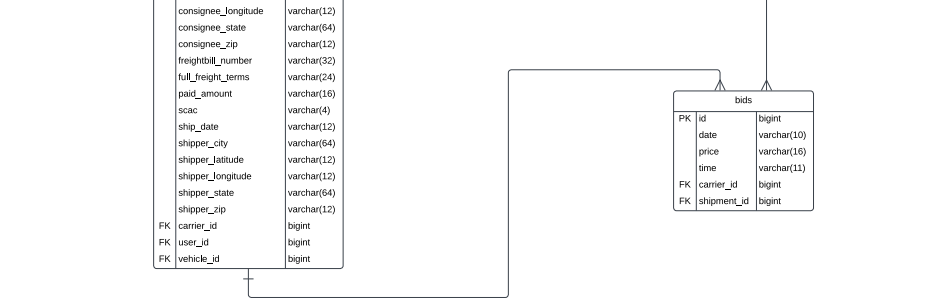


Figure – Class Diagram

## 12.2 Sequence diagram

**Login**

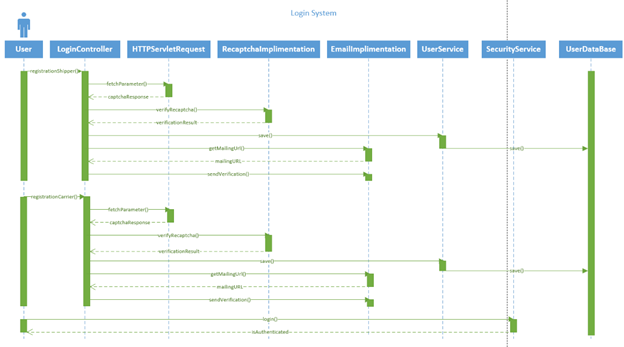


Figure – Login System Sequence Diagram

**Upload Shipments from Excel/Push Shipments to Auction:**

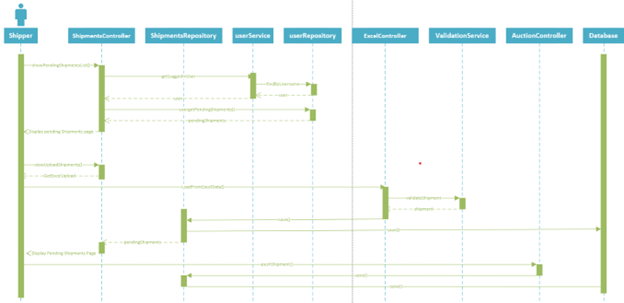


Figure – Upload Shipments from Excel and Push Shipments to Auction Sequence Diagram

**Shipments**

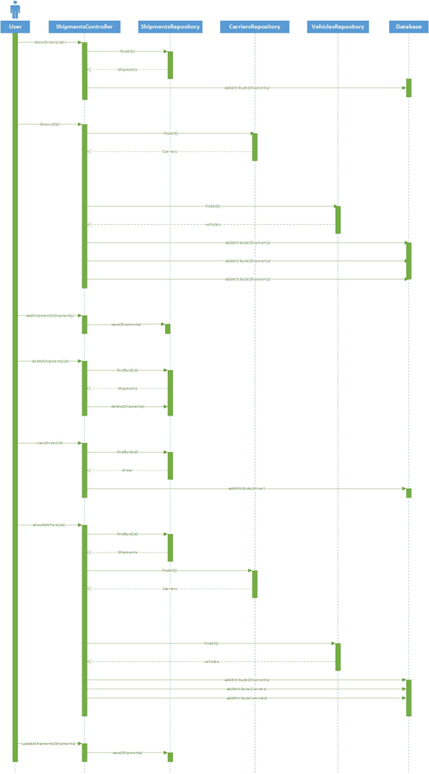


Figure – Shipments Sequence Diagram

**Carrier Create Bid:**

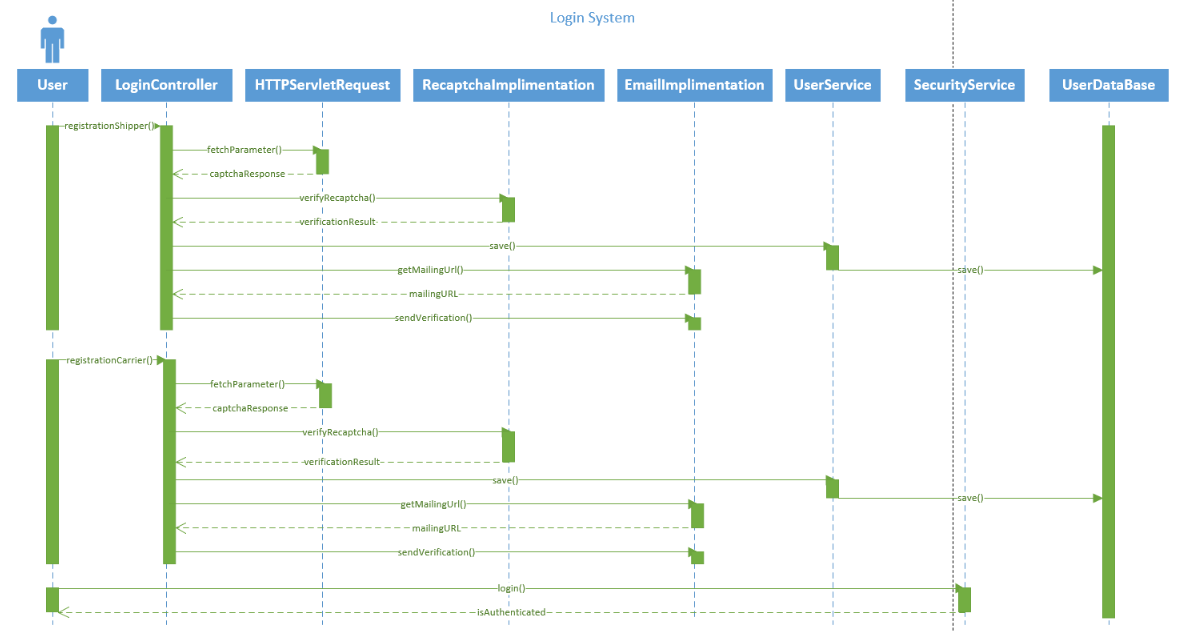


Figure – Carrier create Bid Sequence Diagram

**Shipper Accept Bid:**

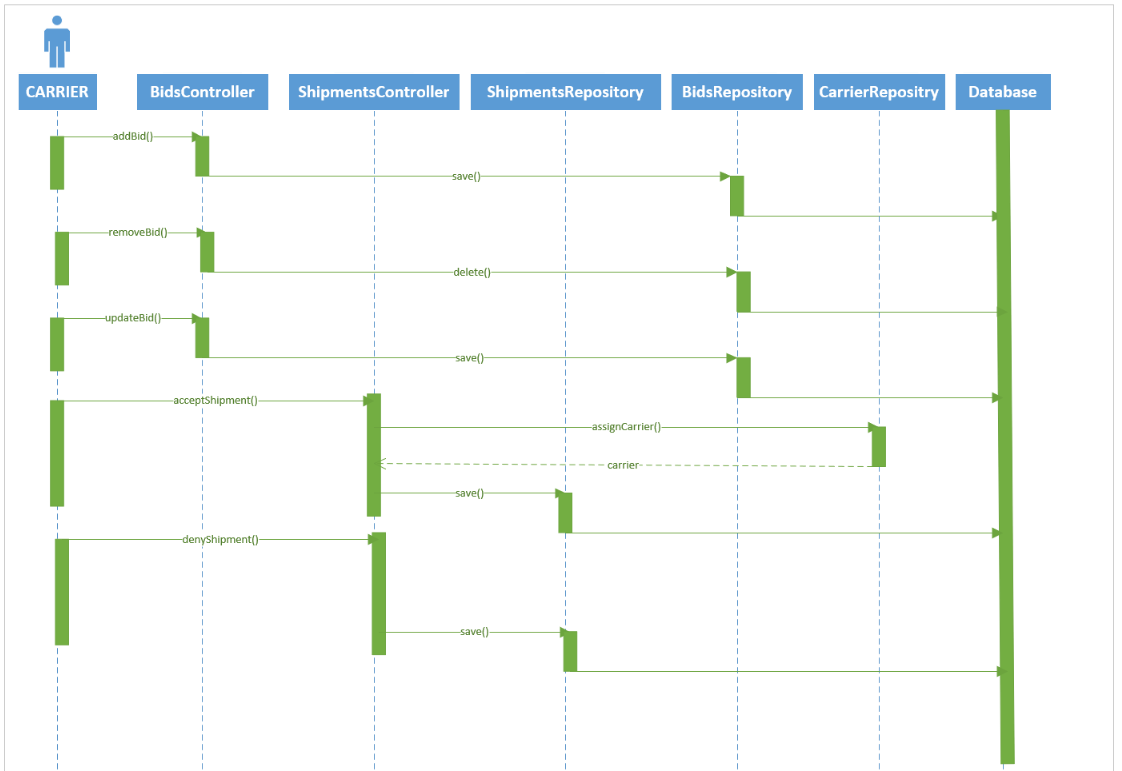


Figure - Accept Bid Sequence Diagram

**Carrier User Auction:**

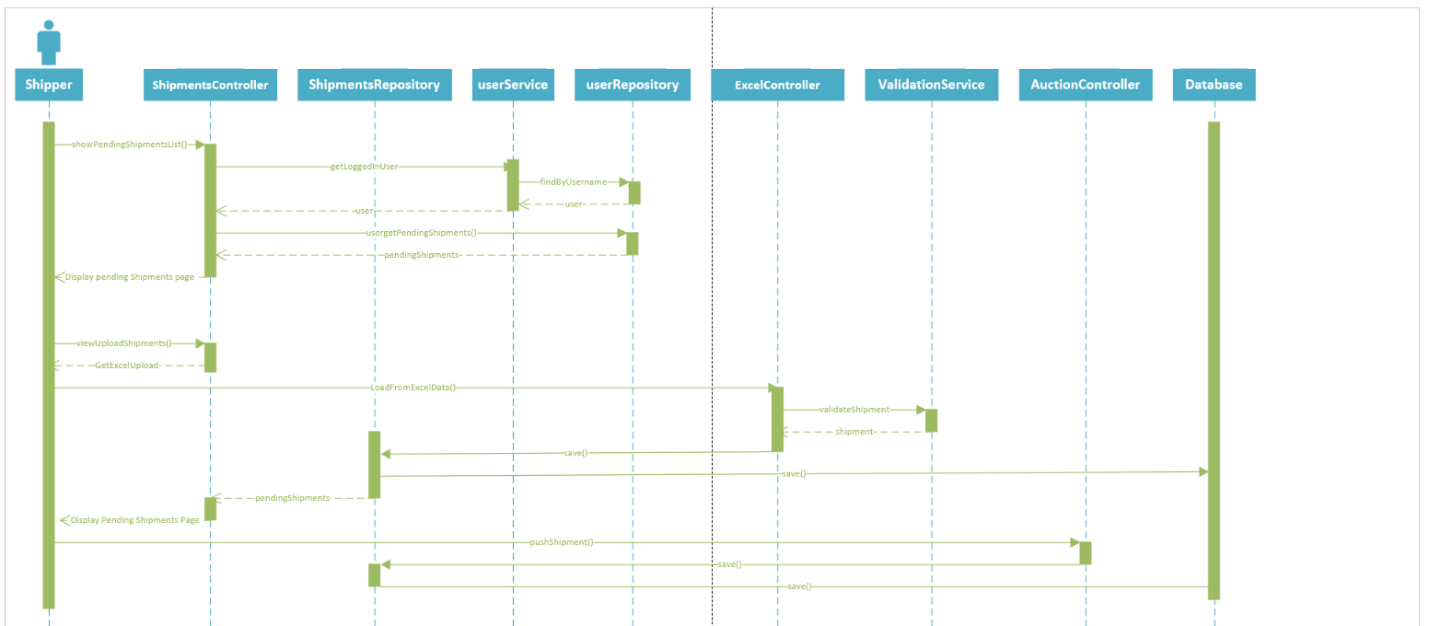
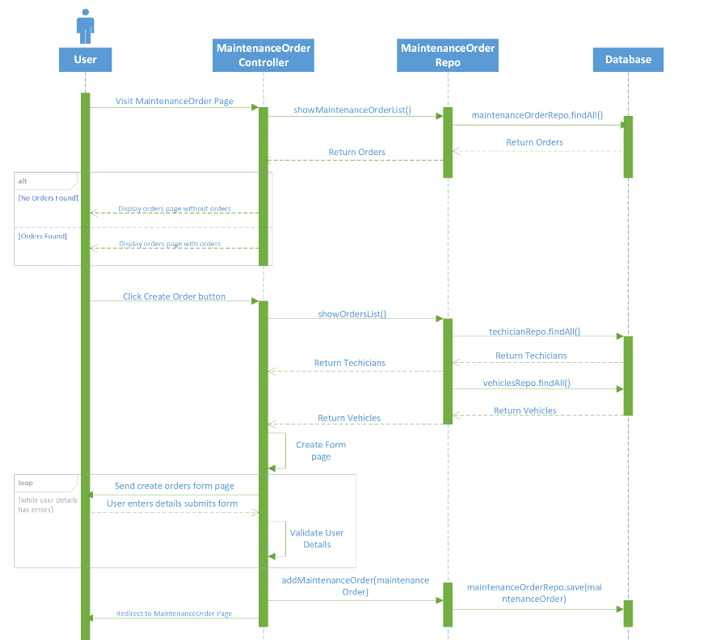


Figure - Carrier User Auction Sequence Diagram

**Maintenance Order:**



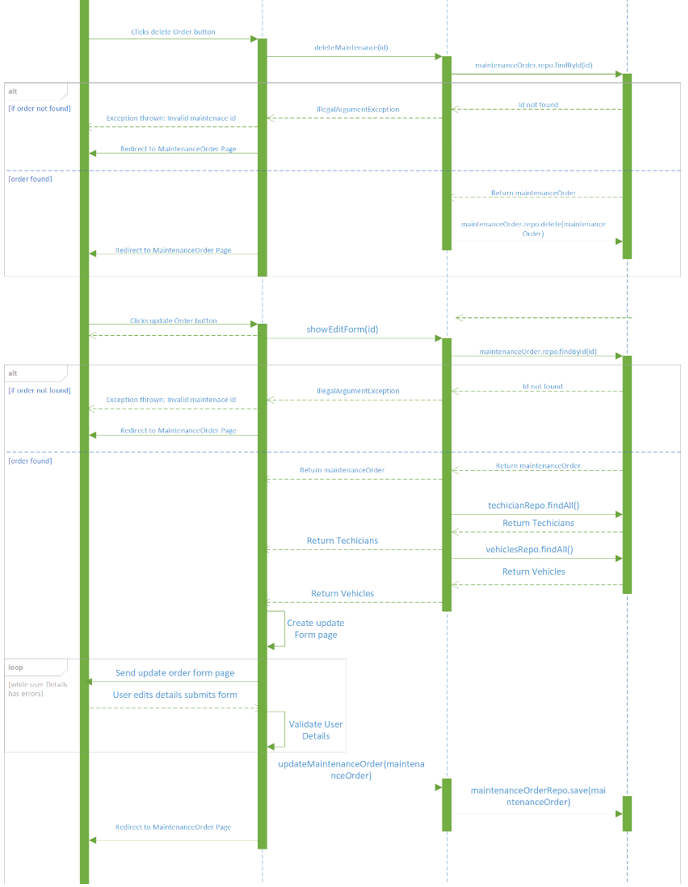


Figure – Maintenance Order Sequence Diagram

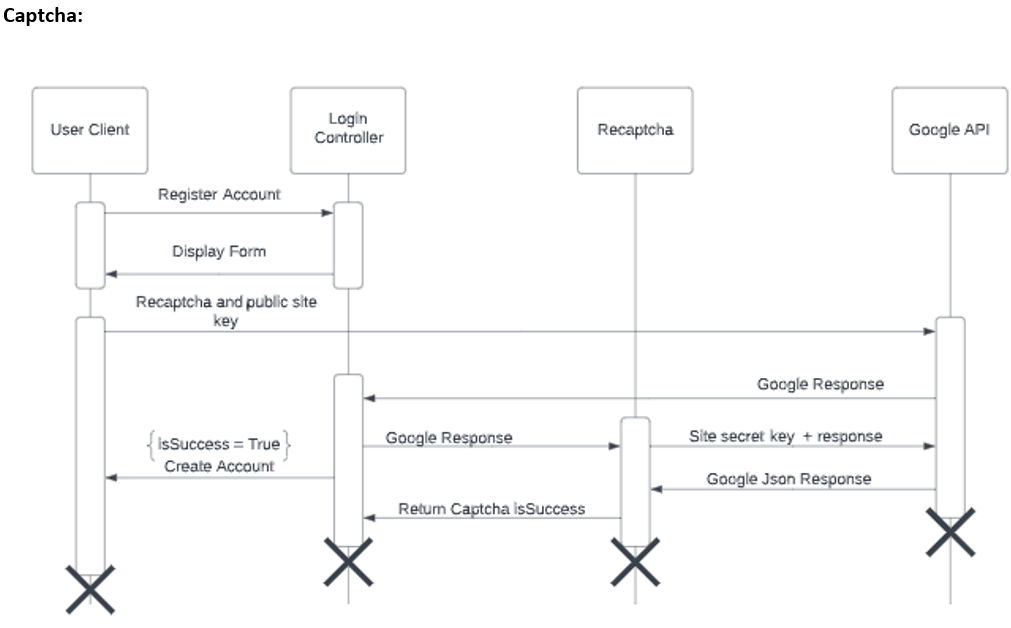


Figure – Captcha State Sequence Diagram

## 12.3 Case diagrams

**Shipper User:**

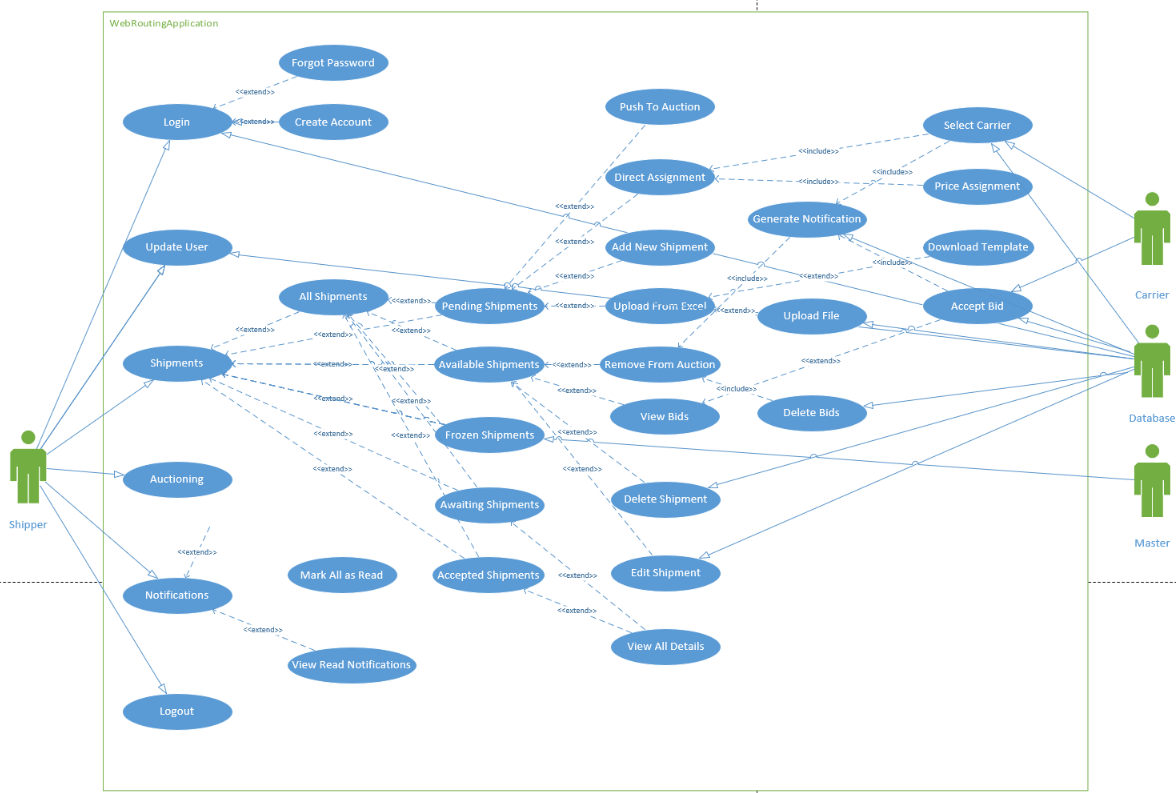


Figure – Shipper User Sequence Diagram

**Auctioneer User:**

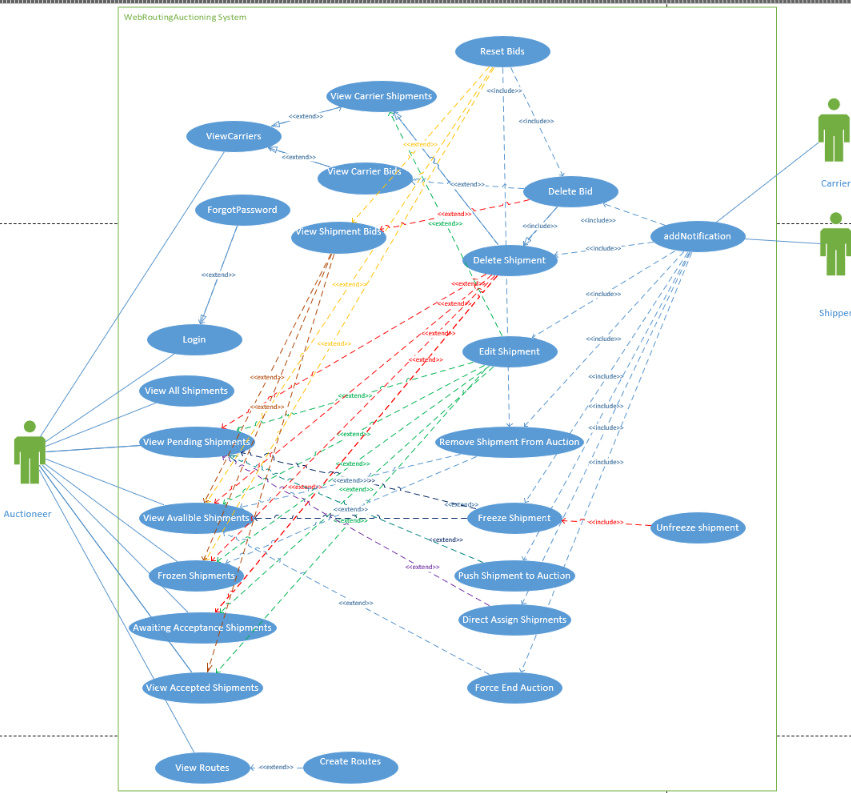
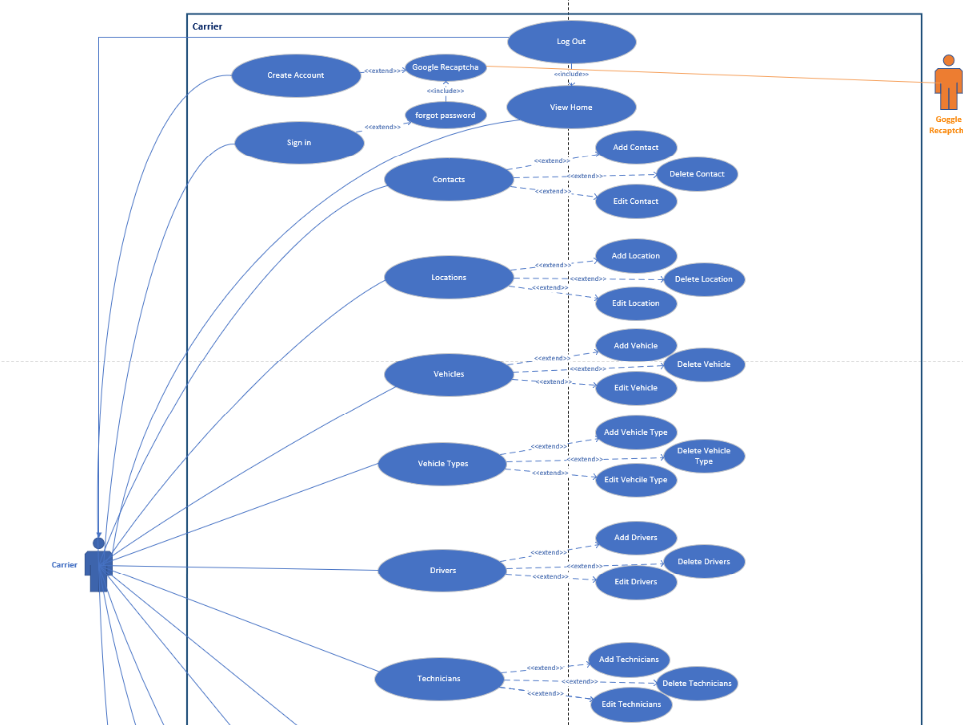


Figure – Auction User Use Case Diagram

**Carrier User:**



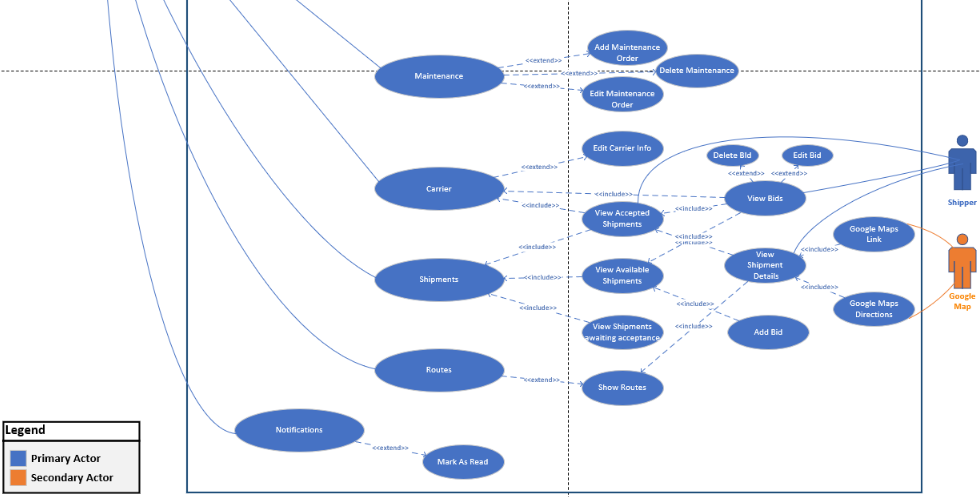
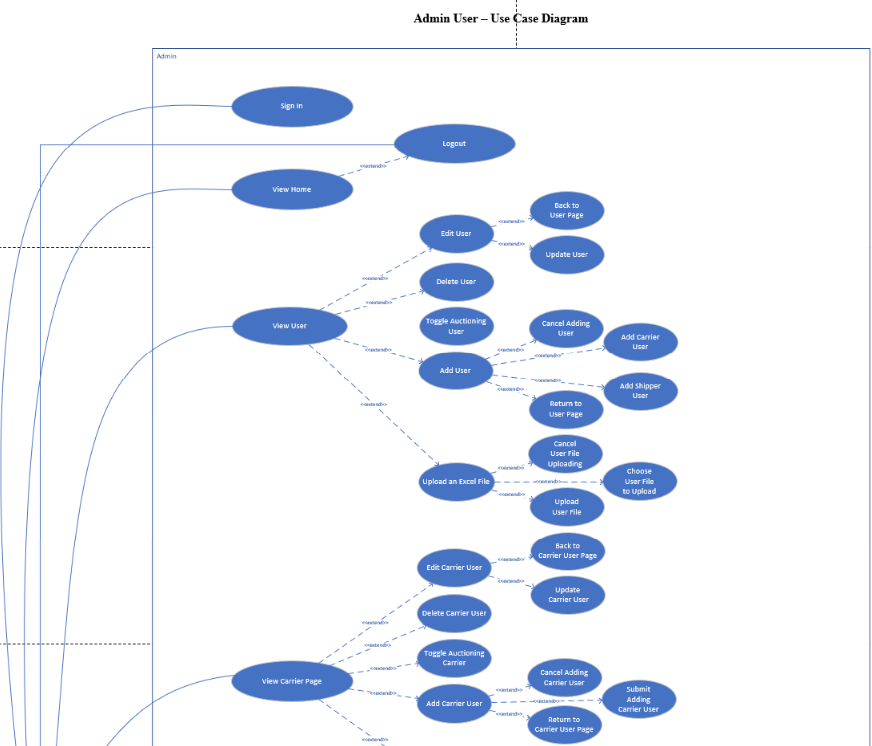


Figure – Carrier User Use Case Diagram

**Admin User:**



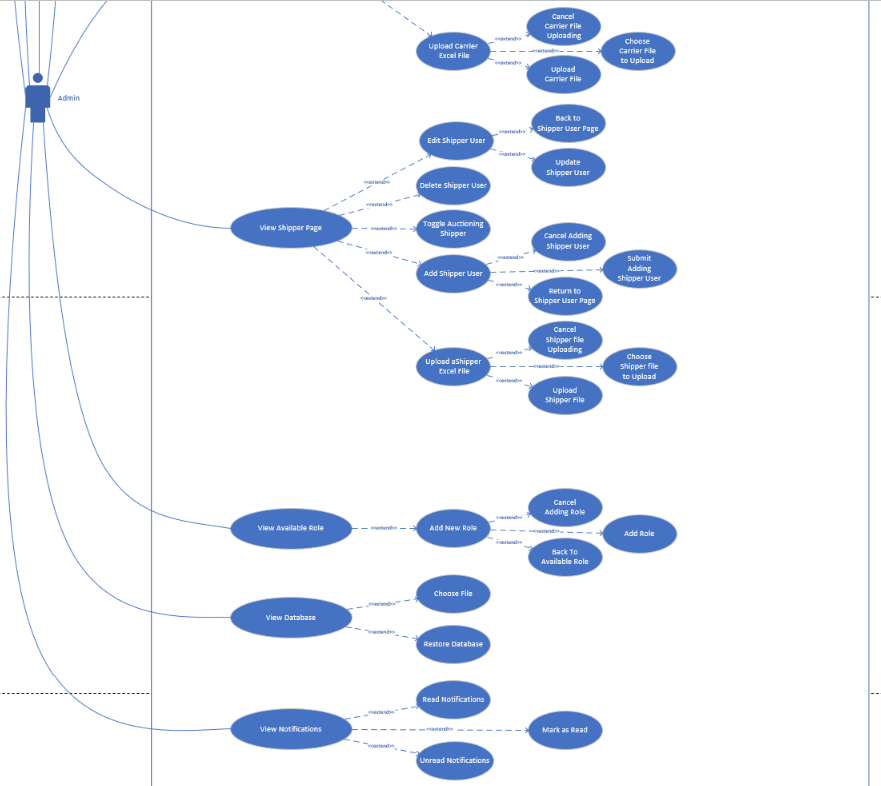


Figure – Admin User Use Case Diagram

## 12.4 Activity diagrams

This section contains activity diagrams of functionality in the system.

**Email Verification:**

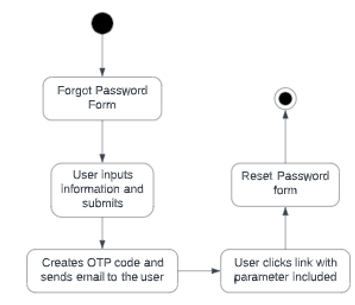


Figure – Email Verification Diagram

**Carrier User**: Applies to Contacts, Drivers, Locations, Maintenance Orders, and Vehicle Types. (Fleet management)

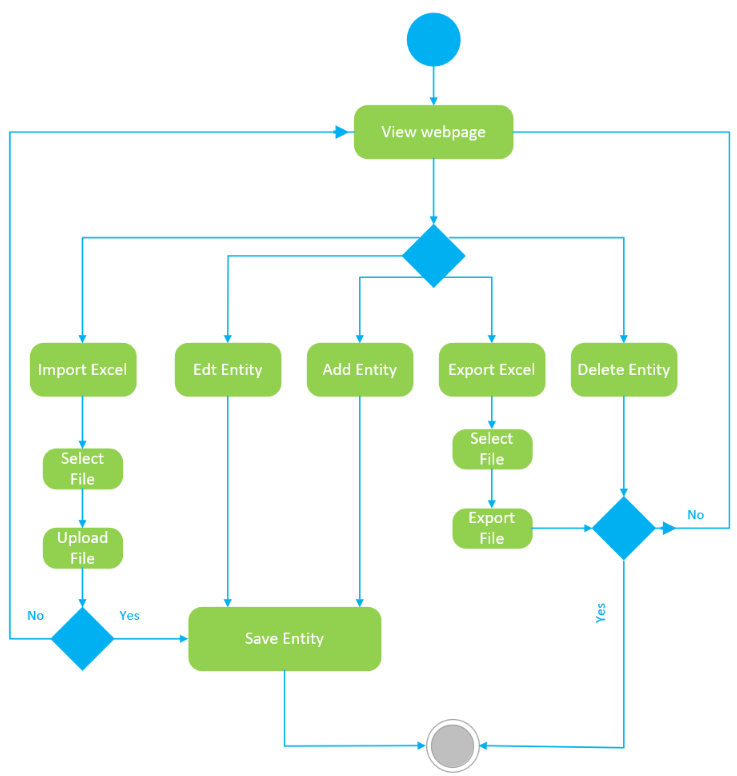


Figure – Carrier Fields Activity Diagram

**Carrier Add Bid:**

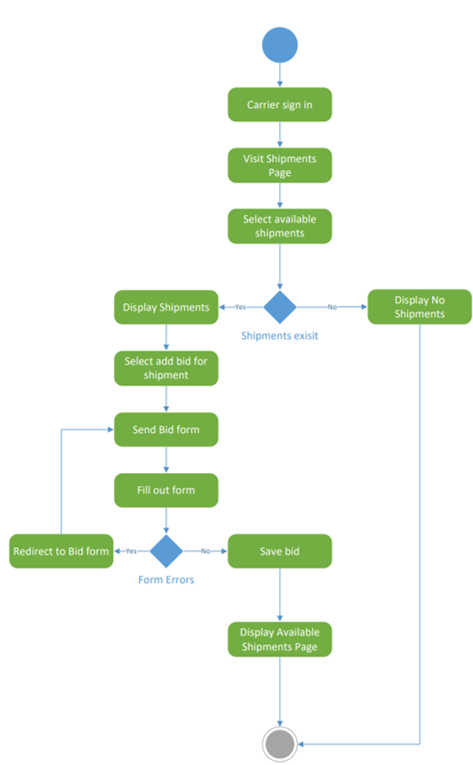


Figure – Add Bid Activity Diagram

**Shipper Accept Bid:**

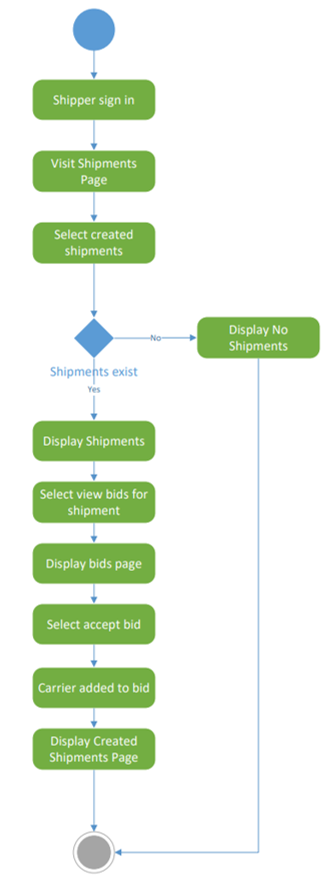


Figure – Accept Bid Activity Diagram

## 12.5 State chart diagram

**Email Verification:**

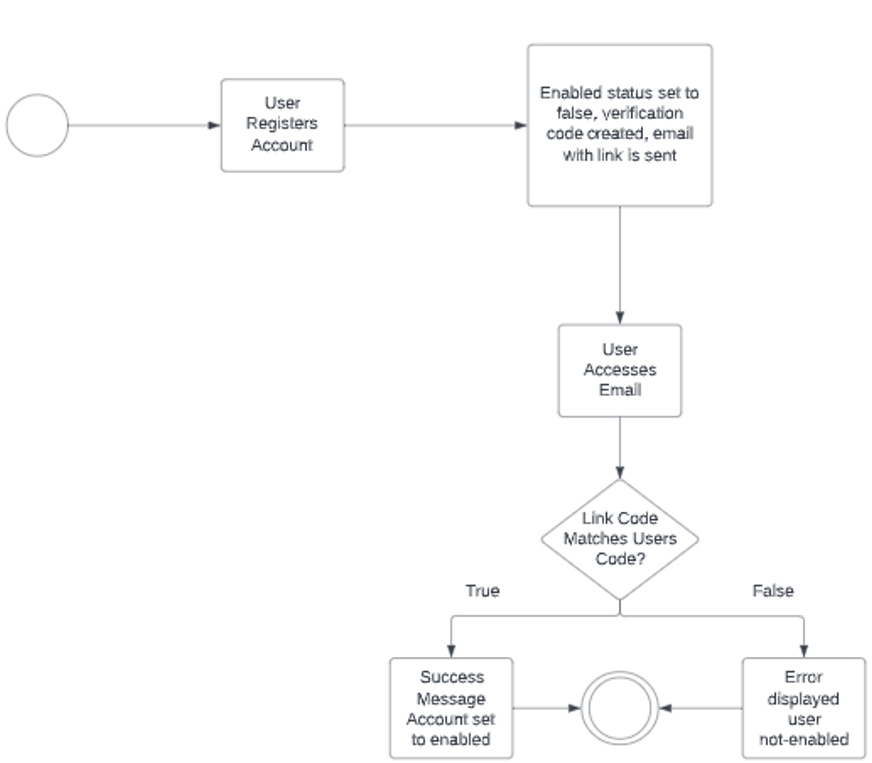


Figure – Email Verification State Chart Diagram

[Figure 1 – Class Diagram 13](#_Toc133358426)

[Figure 2 – Login System Sequence Diagram 14](#_Toc133358427)

[Figure 3 – Upload Shipments from Excel and Push Shipments to Auction Sequence Diagram 14](#_Toc133358428)

[Figure 4 – Shipments Sequence Diagram 15](#_Toc133358429)

[Figure 5 – Carrier create Bid Sequence Diagram 16](#_Toc133358430)

[Figure 6 - Accept Bid Sequence Diagram 16](#_Toc133358431)

[Figure 7 - Carrier User Auction Sequence Diagram 17](#_Toc133358432)

[Figure 8 – Maintenance Order Sequence Diagram 18](#_Toc133358433)

[Figure 9 – Captcha State Sequence Diagram 19](#_Toc133358434)

[Figure 10 – Shipper User Sequence Diagram 19](#_Toc133358435)

[Figure 11 – Auction User Use Case Diagram 20](#_Toc133358436)

[Figure 12 – Carrier User Use Case Diagram 21](#_Toc133358437)

[Figure 13 – Admin User Use Case Diagram 22](#_Toc133358438)

[Figure 14 – Email Verification Diagram 23](#_Toc133358439)

[Figure 15 – Carrier Fields Activity Diagram 23](#_Toc133358440)

[Figure 16 – Add Bid Activity Diagram 24](#_Toc133358441)

[Figure 17 – Accept Bid Activity Diagram 25](#_Toc133358442)

[Figure 18 – Email Verification State Chart Diagram 26](#_Toc133358443)

# REREFERNCES

**Templates for AJAX requests:**

*Introduction to Ajax*. Ajax introduction. (n.d.). Retrieved March 13, 2023, from <https://www.w3schools.com/js/js_ajax_intro.asp>

**Code for changing cursor to loading symbol during requests:**

StanleyStanley 3. (2019, December 1). *Changing cursor to waiting in javascript/jquery*. Stack Overflow. Retrieved April 23, 2023, from <https://stackoverflow.com/questions/9681080/changing-cursor-to-waiting-in-javascript-jquery>

**Templates for Google API requests:**

Google. (n.d.). *Google Maps API Overview*. Google. Retrieved April 3, 2023, from <https://developers.google.com/maps/documentation/javascript/overview>

**Code for custom login success handler (used for logging and directing to landing pages) based off:**

Minh, N. H. (n.d.). *Spring Security Authentication Success Handler Examples*. Codejava. Retrieved January 8, 2023, from <https://www.codejava.net/frameworks/spring-boot/spring-security-authentication-success-handler-examples>

**Template for Springboot email sending:**

baeldung, W. by: (2021, October 28). *Guide to spring email*. Baeldung. Retrieved March 3, 2023, from <https://www.baeldung.com/spring-email>

**Backup System Code:**

Wayne. (2020, September 20). *How to backup and restore mysql databases in Spring Boot*. Wayne's Talk. Retrieved February 28, 2023, from <https://waynestalk.com/en/spring-boot-backup-restore-mysql-en/>

**Chat-GPT (used to generate frameworks for ExcelController, ValidationService, and for various javascipt scripts on the direct assign and freight rate pages):**

OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model.]

**Excel XFFS Code:**

*Busy developers' guide to HSSF and XSSF features*. Apache POI. (n.d.). Retrieved March 10, 2023, from <https://poi.apache.org/components/spreadsheet/quick-guide.html>

**Freight Rate Table Design Reference:**

*JD Edwards World Load and Delivery Management Guide*. Create Freight Tables. (2015, February 23). Retrieved March 24, 2023, from <https://docs.oracle.com/cd/E59116_01/doc.94/e58786/create_freight_tables.htm#WEALD562>

**Springboots general reference:**

*Spring Framework 6.0.8 API*. Overview (Spring Framework 6.0.8 API). (n.d.). Retrieved April 24, 2023, from <https://docs.spring.io/spring-framework/docs/current/javadoc-api/index.html>

**Log4J Reference:**

Goers, R. (n.d.). *Log4J Configuration*. Log4j – configuring log4j 2. Retrieved April 24, 2023, from <https://logging.apache.org/log4j/2.x/manual/configuration.html>

# Figures