Web Routing

Final Report

Slippery Rock University

Fady Aziz [FAA1002@SRU.EDU](mailto:FAA1002@SRU.EDU)

IAN BLACK [IMB1007@SRU.EDU](mailto:IMB1007@SRU.EDU)

LOGAN KIRKWOOD [LLK1005@SRU.EDU](mailto:LLK1005@SRU.EDU)

Contents

[1. References 4](#_Toc101781512)

[2. Completion 4](#_Toc101781513)

[2.1 Complete Modules 4](#_Toc101781514)

[2.2 Incomplete 9](#_Toc101781515)

[3. Contribution 9](#_Toc101781516)

[3.1 Sources Used 9](#_Toc101781517)

[3.2 Our Contribution 10](#_Toc101781518)

[4. Glossary 10](#_Toc101781519)

[5. Problem Explanation 10](#_Toc101781520)

[6. System Requirements 11](#_Toc101781521)

[7. UML Diagrams 12](#_Toc101781522)

[7.1 Sequence Diagrams 12](#_Toc101781523)

[7.2 Activity Diagrams 16](#_Toc101781524)

[7.3 State Diagrams 21](#_Toc101781525)

[8. Data Flow Diagrams 22](#_Toc101781526)

[9. Entity / Relationship Diagrams 23](#_Toc101781527)

[10. Caveats / Minefields 24](#_Toc101781528)

[10.1 Carrier Losing Link to User 24](#_Toc101781529)

[10.2 Reading Large Excel Files 24](#_Toc101781530)

[10.3 Reading a Carrier Excel File 24](#_Toc101781531)

[10.4 Shipment Dates 24](#_Toc101781532)

[10.5 Recovery of Account 24](#_Toc101781533)

[11. Documentation 25](#_Toc101781534)

[12. File Path Names 28](#_Toc101781535)

[13. Code Reusability 35](#_Toc101781536)

[14. Testing 36](#_Toc101781537)

[14.1 Human – Computer Testing 36](#_Toc101781538)

[14.2 Blackbox Testing 36](#_Toc101781539)

[14.3 Whitebox Testing 36](#_Toc101781540)

[14.4 Unit Testing 36](#_Toc101781541)

[14.5 Integration Testing 36](#_Toc101781542)

[14.6 Regression Testing 36](#_Toc101781543)

[14.7 Boundary Analysis 36](#_Toc101781544)

[14.8 Volume Test 36](#_Toc101781545)

[14.9 Stress Test 36](#_Toc101781546)

[14.10 Missing Resources 36](#_Toc101781547)

[14.11 Mis-Management of Resources 36](#_Toc101781548)

[14.12 Recovering from Crashes 36](#_Toc101781549)

[15. Logger 37](#_Toc101781550)

[16. Deployment / Maintenance 38](#_Toc101781551)

[16.1 Downloading Project from GitHub 38](#_Toc101781552)

[16.1.1 Finding the GitHub Repository 38](#_Toc101781553)

[16.1.2 Downloading the Project 38](#_Toc101781554)

[16.1.3 Opening the Project in Eclipse 38](#_Toc101781555)

[16.2 Cloning Project into Eclipse 41](#_Toc101781556)

[16.2.1 Cloning Project via URI 41](#_Toc101781557)

[16.3 Setting up the MySQL Database 45](#_Toc101781558)

[16.3.1 Installing MySQL & MySQL Workbench 45](#_Toc101781559)

[16.3.2 Loading MySQL Data 46](#_Toc101781560)

[16.3.3 Changing Application Properties 47](#_Toc101781561)

[16.4. Starting the Application 47](#_Toc101781562)

[16.4.1 Where to Start Application From 47](#_Toc101781563)

[16.4.2 Where to Access Application From 48](#_Toc101781564)

[17. Post-Mortem Analysis 49](#_Toc101781565)

[18. Figures 50](#_Toc101781566)

# 1. References

Bodnar, J. (2020, September 10). *Spring Boot MySQL*. Retrieved from ZetCode: https://zetcode.com/springboot/mysql/

mkyong. (2017, May 4). *Spring Boot + Spring Security + Thymeleaf*. Retrieved from Mkyong: https://mkyong.com/spring-boot/spring-boot-spring-security-thymeleaf-example/

Ngo, G. (2020, October 28). *Spring Boot Registration Login and Logout Example with Spring Security, Thymeleaf, Spring Data JPA and Hibernate*. Retrieved from hellokoding: https://hellokoding.com/spring-security-login-logout-thymeleaf/

Datsabk. (2016, December 28). Apache POI – Reading and Writing Excel file in Java. Retrieved from Mkyong: https://mkyong.com/java/apache-poi-reading-and-writing-excel-file-in-java/

# 2. Completion

**Completion Status: Incomplete**

## 2.1 Complete Modules

The following modules have been completed.

* Creation of all the entities
  + Bids
  + Carriers
  + Contacts
  + Driver
  + Locations
  + Maintenance Orders
  + Roles
  + Routes
  + Shipments
  + Technicians
  + User
  + Vehicles
  + Vehicle Types
* Creation of all the Controller Classes and their methods
  + Bids Controller
    - BidsController(BidsRepository, ShipmentsRepository, CarriersRepository)
    - acceptBid(long, Model)
    - addBid(Bids, BindingResult, Model)
    - deleteBid(long, Model)
    - getLoggedInUser()
    - showBidList(long, Model, Bids, BindingResult)
    - showEditForm(long, Model)
    - updateBid(long, Bids, BindingResult, Model)
  + Carriers Controller
    - CarriersController(CarriersRepository)
    - addCarrier(Carriers, BindingResult, Model)
    - deleteCarrier(long, Model)
    - getLoggedInUser()
    - ListFromExcelData(Model)
    - showCarriersList(Model)
    - showEditForm(long, Model)
    - showList(Model, Carriers, BindingResult)
    - updateCarrier(long, Carriers, BindingResult, Model)
    - viewCarrier(long, Model)
    - viewCarrierBids(long, Model)
    - viewCarrierShipments(long, Model)
  + Contacts Controller
    - ContactsController(ContactsRepository)
    - addCarrier(Carriers, BindingResult, Model)
    - deleteCarrier(long, Model)
    - getLoggedInUser()
    - ListFromExcelData(Model)
    - showCarriersList(Model)
    - showEditForm(long, Model)
    - showList(Model, Carriers, BindingResult)
    - updateCarrier(long, Carriers, BindingResult, Model)
    - viewCarrier(long, Model)
    - viewCarrierBids(long, Model)
    - viewCarrierShipments(long, Model)
  + Driver Controller
    - DriverController(DriverRepository, ContactsRepository, CarriersRepository, VehiclesRepository)
    - addDriver(Driver, BindingResult, Model)
    - deleteDriver(long, Model)
    - getLoggedInUser()
    - showDriversList(Model)
    - showEditForm(long, Model)
    - showLists(Model, Driver, BindingResult)
    - updateDriver(long, Driver, BindingResult, Model)
    - viewDriver(long, Model)
  + Locations Controller
    - LocationsController(LocationsRepository, CarriersRepository)
    - addLocation(Locations, BindingResult, Model)
    - deleteLocation(long, Model)
    - getLoggedInUser()
    - showCarriersList(Model, Locations, BindingResult)
    - showEditForm(long, Model)
    - showLocationsList(Model)
    - updateLocation(long, Locations, BindingResult, Model)
    - viewLocation(long, Model)
  + Login Controller
    - LoginController(CarriersRepository, ContactsRepository)
    - error403()
    - getLoggedInUser()
    - login(Model, String, String)
    - registrationCarrier(User, BindingResult, Model, String, String, String, String, String, String, String, String, String, String, String, String, String, String, String, String, String)
    - registrationCarrierInitial(Model)
    - registrationHome(Model)
    - registrationLogin(Model, String)
    - registrationShipper(User, BindingResult)
    - registrationShipperInitial(Model)
    - welcome(Model)
  + Maintenance Orders Controller
    - MaintenanceOrderController(MaintenanceOrdersRepository, TechniciansRepository, DriverRepository, VehiclesRepository)
    - addMaintenanceOrder(MaintenanceOrders, BindingResult, Model)
    - deleteMaintenance(long, Model)
    - getLoggedInUser()
    - showEditForm(long, Model)
    - showMaintenanceOrdersList(Model)
    - showOrderList(Model, MaintenanceOrders, BindingResult)
    - updateOrder(long, MaintenanceOrders, BindingResult, Model)
  + Roles Controller
    - RolesController(RoleRepository)
    - showRoleList(Model)
  + Routes Controller
    - RoutesController(VehiclesRepository)
    - getVehiclesWithShipments()
    - getLoggedInUser()
    - showVehicleRoute(Model, String, String, boolean)
    - showVehiclesList(Model)
  + Shipments Controller
    - ShipmentsController(BidsRepository, ShipmentsRepository, CarriersRepository, VehiclesRepository)
    - addShipment(Shipments, BindingResult, Model)
    - deleteShipment(long, Model)
    - getLoggedInUser()
    - ListFromExcelData(Model)
    - LoadFromExcelData(MultipartFile)
    - shipmentsHomeCarrier(Model)
    - shipmentsHomeMaster(Model)
    - shipmentsHomeShipper(Model)
    - showAcceptedShipmentsList(Model)
    - showCreatedShipmentsList(Model)
    - showEditForm(long, Model)
    - showList(Model, Shipments, BindingResult)
    - showShipmentList(Model)
    - updateShipment(long, Shipments, BindingResult, Model)
    - viewCarrierShipments(long, Model)
    - viewShipmentBids(long, Model)
  + Technicians Controller
    - TechniciansController(TechniciansRepository, ContactsRepository)
    - addtechnician(Technicians, BindingResult, Model)
    - deletetechnician(long, Model)
    - getLoggedInUser()
    - showContactList(Model, Technicians, BindingResult)
    - showEditForm(long, Model)
    - showTechList(Model)
    - updateTechnician(long, Technicians, BindingResult, Model)
    - viewMaintenanceOrders(long, Model)
    - viewTechnician(long, Model)
  + User Controller
    - UserController(UserRepository, RoleRepository, CarriersRepository, ContactsRepository)
    - addUser(User, BindingResult, Model)
    - addUserCarrier(User, BindingResult, Model, String, String, String, String, String, String, String, String, String, String, String, String, String, String, String, String, String)
    - deleteUser(long, Model)
    - getLoggedInUser()
    - showCarrierPage(User, Model)
    - showEditForm(long, Model)
    - shownAddHomePage(Model)
    - showOtherPage(User, Model)
    - showUserDetailsForm(Model)
    - showUserList(Model)
    - updateDetails(User, BindingResult, Model)
    - updateUser(long, User, BindingResult, Model, boolean)
  + Vehicles Controller
    - VehiclesController(VehiclesRepository, LocationsRepository, VehicleTypesRepository, CarriersRepository)
    - addVehicle(Vehicles, BindingResult, Model)
    - deleteVehicle(long, Model)
    - getLoggedInUser()
    - showEditForm(long, Model)
    - showLists(Model, Vehicles, BindingResult)
    - showVehicleList(Model)
    - updateVehicle(long, Vehicles, BindingResult, Model)
    - viewVehicle(long, Model)
    - viewVehicleDrivers(long, Model)
    - viewVehicleShipments(long, Model)
  + Vehicle Types Controller
    - VehicleTypesController(VehicleTypesRepository)
    - addVehicleType(VehicleTypes, BindingResult, Model)
    - deleteVehicleType(long, Model)
    - showEditForm(long, Model)
    - showVehicleTypeList(Model)
    - showVehicleTypeSignUpForm(VehicleTypes)
    - updateVehicleType(long, VehicleTypes, BindingResult, Model)
    - viewVehicleType(long, Model)
* Creation of the following Repositories
  + Bids Repository
  + Carriers Repository
  + Contacts Repository
  + Driver Repository
  + Locations Repository
  + Maintenance Orders Repository
  + Roles Repository
  + Routes Repository
  + Shipments Repository
  + Technicians Repository
  + User Repository
  + Vehicles Repository
  + Vehicle Types Repository
* Creation of the following security services and their methods
  + Security Service
    - autoLogin(String, String)
    - isAuthenticated()
  + Security Service Implementation
    - autoLogin(String, String)
    - isAuthenticated()
  + User Details Service Implementation
    - loadUserByUsername(String)
  + User Service
    - findByUsername(String)
    - save(User)
  + Use Service Implementation
    - findByUsername(String)
    - save(User)
* Creation of Error class and method
  + Access Denied Handler
    - handle(HttpServletRequest, HttpServletResponse, AccessDeniedException)
* Creation of authorization and methods
  + Web Security Configuration
    - configure(HttpSecurity)
    - configureGlobal(AuthenticationManagerBuilder)
    - customAuthenticationManager()
    - userValidator()
* Creation of resources needed to run the program
  + Application properties
  + Banner for webpage
  + Validation properties
* Creation of Web Pages for all entities
  + Web Pages with the following tasks
    - View
    - Add
    - Delete
    - Update

## 2.2 Incomplete

The following have not been completed.

* Junit tests for multiple classes.
* Implementation of using google maps to acquire address.
* Reading of excel file for Carrier class

# 3. Contribution

## 3.1 Sources Used

We estimated around 30% of the code was referenced from an outside source. Here are the following sources and what was referenced.

* Mkyong.com
  + This site was referenced for the implementation of Spring security, examples of using Thymeleaf, reading/writing excel files.
  + <https://mkyong.com/spring-boot/spring-boot-spring-security-thymeleaf-example/>
  + <https://mkyong.com/java/apache-poi-reading-and-writing-excel-file-in-java/>
* Zetcode.com
  + This site was used for a reference of writing/implementing an SQL file and creation of a database.
  + <https://zetcode.com/springboot/mysql/>
* Hellokoding.com
  + This site was referenced for the implementation of registration, login, and logout of our system
  + <https://hellokoding.com/spring-security-login-logout-thymeleaf/>
* Enterprise Routing System – 488 Class project - Alex McCracken, Kelly Smith, Mitch Nemitz
  + Our project was a continuation of this previous project in 2013. We used their previous build for making our entities and SQL file. Although the entities and SQL were heavily changed, it was originally built off their files. We also used their CSS formatting in our project for our web pages.
* CPSC 488 Class examples - Dr. Sam Thangiah
  + Our project followed examples that we went over in class. Examples such as testing, MVC coding, and all UML diagram creations.

## 3.2 Our Contribution

We estimated that around 70% of our code was written by ourselves. We used most of our sources for understanding the frameworks. Methods written in our Controller classes were all created by our members. We created all the html pages ourselves (Besides the CSS files). Entities and SQL files were originally based on the previous project, but our group made many significant changes such as adding/removing attributes, creation/deletion of entities, database relationship changes, etc.

# 4. Glossary

* Bids
  + Bids are the offers made from Carriers regarding a certain shipment made by the shippers. The bids have the following attributes id, price, date, time, link to shipment, link to carrier. Once a bid is placed on a shipment the shippers can accept the bid. This gives the shipment to the winning Carrier bid.
* Carriers
  + Carriers are the person or company that handles the conveyance of goods. These goods are labeled shipments in our application. Carriers have the following attributes id, name, SCAC code, LTL (less than truckload), FTL (full truckload), Pallets, weight, link to contact, list of shipments, list of bids, list of locations, list of vehicles, list of drivers.
* Contacts
  + Holds the information needed to communicate with a specific person or business. Contacts have the following attributes first name, last name, middle initial, email address, street address, city, state, zip, primary phone number, work phone number, list of carriers, list of drivers, and list of technicians.
* Driver
  + Holds the information for the individual who will be driving the vehicles for transportation. Driver has the following attributes id, link to contact, link to carrier, link to vehicle, license number, license expiration, license class.
* Locations
  + Holds the information for any location being used by the Carrier. Locations have the following attributes id, name, street address, city, state, zip, latitude, and longitude.
* Maintenance Orders
  + Holds all the information on requests for maintenance on a vehicle. Orders hold the following attributes id, link to technician, scheduled date, details, service type, cost, status of completion, link to vehicle, type of maintenance type.
* Role
  + This is the role of the user. The role could be Admin, Master List, Carrier, or Shipper. Role has the following attributes id, name, and list of users.
* Shipments
  + Holds all the information on a shipment created from a shipper user. This is the main action of shipper users. Shipments have the following attributes id, client, link to carrier, SCAC code, client mode, ship date, freight bill number, paid amount, full freight terms, commodity classification, commodity pieces, commodity paid weight, shipper city, shipper state, shipper zip, shipper latitude, shipper longitude, consignee city, consignee state, consignee zip consignee latitude, consignee longitude, link to vehicle, link to user, and list of bids.
* Technicians
  + Holds the information on technicians employed by Carriers. It holds the following attributes id, link to contact, skill grade, and list of maintenance orders.
* User
  + Holds the account information of someone who has signed up as either a carrier, shipper, admin, or master list. It holds the following attributes id, username, password, email, link to role, link to carrier (if carrier role is selected), and list of shipments.
* Vehicles
  + Holds all the information on the vehicles owned by a Carrier. It has the following attributes id, plate number, VIN number, manufactured year, link to vehicle type, link to location, link to carrier, list of maintenance orders, list of shipments, and list of drivers.
* Vehicle Types
  + Holds the information on the type of vehicle it is associated with. It has the following attributes id, type, sub type, description, make, model, minimum weight, maximum weight, capacity, maximum range, restrictions, height, empty weight, length, minimum cubic weight, maximum cubic weight, and list of vehicles.

# 5. Problem Explanation

The purpose of this application is to provide a way to properly store information for carriers on different shipments they can carry out. Information that the carrier can store, and use includes contacts, drivers, locations, maintenance orders, shipments, technicians, vehicle types, and vehicles. These entities all interact with each other in different ways.

The program also implements an auction functionality where someone logged in as a shipper can create shipments that can be bid on. Someone logged in as a carrier can view the shipments created by a shipper and place bids on those shipments. If a shipper receives a bid on a shipment that they want to accept, they can do so, and the shipment is automatically assigned to the carrier that placed the bid. After a shipment is assigned to a carrier, the carrier can then view and edit the details of that shipment.

Details and information on all parts accessible by the carrier and shipper can be edited and deleted from the program. New users can register as either a shipper or a carrier. Other account types include an admin which can create and edit the details of other users within the program. A master list is another account type that can view all carriers and shipments in the program regardless of their role.

The program at its core is to provide an easy way to auction shipments as well as store important data for carriers.

# 6. System Requirements

**CPU:** Intel® Core i5

**Memory:** 8GB

**OS:** Microsoft® Windows® 10

**Free Space:** 40gb

**Software:**

* Eclipse IDE EE – 2021-12
* MySQL Server 8.0.28
* MySQL Workbench 8.0.28
* Google Chrome v100.0.4896.75

# 7. UML Diagrams

Below are UML diagrams of the most complex parts of our system

## 7.1 Sequence Diagrams

Graphical user interface

Description automatically generated

Figure 1 – Drivers Sequence Diagram

Timeline

Description automatically generated

Figure 2 – Maintenance Orders Diagram

A picture containing graphical user interface

Description automatically generated

Figure 3 – Shipments Sequence Diagram

Graphical user interface, chart

Description automatically generated

Figure 4 – Accept Bid Sequence Diagram

Graphical user interface

Description automatically generated

Figure 5 – Create Bid Sequence Diagram

## 7.2 Activity Diagrams

Diagram

Description automatically generated

Figure 6 – Drivers Activity Diagram

Diagram

Description automatically generated

Figure 7 – Update Maintenance Order Activity Diagram

Diagram

Description automatically generated

Figure 8 – Shipments Activity Diagram

Graphical user interface, application

Description automatically generated

Figure 9 – Add Bid Activity Diagram

Graphical user interface, application, table, Excel

Description automatically generated

Figure 10 – Accept Bid Activity Diagram

## 7.3 State Diagrams

Timeline

Description automatically generated with medium confidence

Figure 11 – Drivers State Diagram

Timeline

Description automatically generated

Figure 12 – Maintenance Orders State Diagram

Diagram

Description automatically generated

Figure 13 – Shipments State Diagram

# 8. Data Flow Diagrams

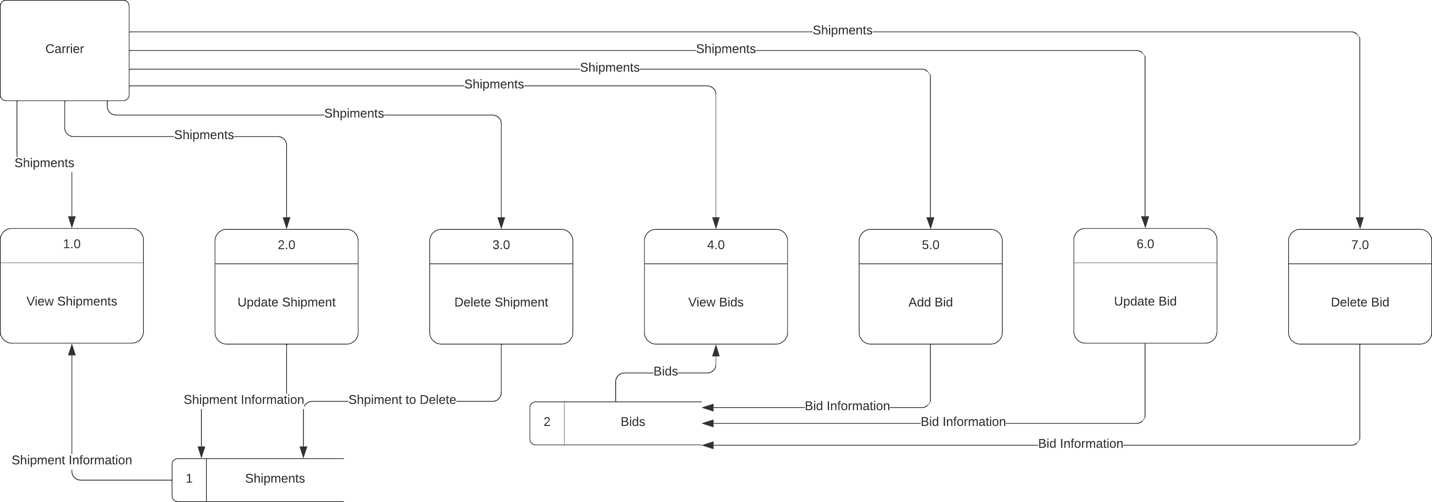


Figure 14 – Shipments Data Flow Diagram as a Carrier

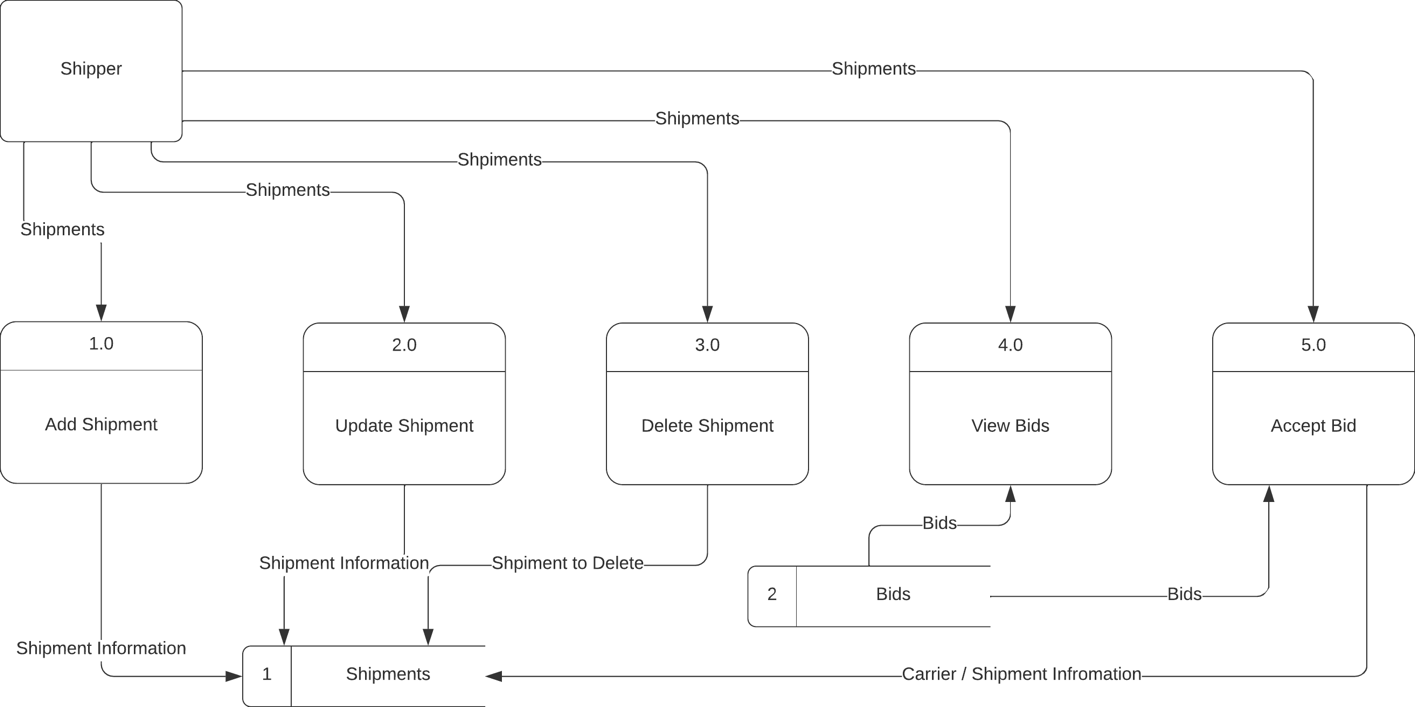


Figure 15 – Shipments Data Flow Diagram as a Shipper

# 9. Entity / Relationship Diagrams

Below is a detailed diagram of all the entities within the program, the values associated with them and the relationship with other entities in the program. All entities have some sort of relationship with each other with only the Role and Carrier entities not having any dependencies.

Timeline

Description automatically generated

Figure 16 – Entity / Relationship Diagram

# 10. Caveats / Minefields

## 10.1 Carrier Losing Link to User

Admins can delete a user associated with a Carrier. This will leave the Carrier without a user and the Carrier cannot be accessed. Admins have the ability to add a Carrier to a user, but there is no current way to create a Carrier role user without a Carrier link.

## 10.2 Reading Large Excel Files

Reading large excel files can take a long time to load. If a user plans on loading a larger excel file, it will take some time. We have an alert alerting this before a file is read. However, we have no other output of how much the file was loaded in real time.

## 10.3 Reading a Carrier Excel File

Carrier excel files cannot be loaded correctly due to Carriers accounts being created during sign up. The file will load into the database. However, the Carriers will not have a user connected to them. Without a link to a user the Carrier will not be accessible. In this current stage loading a Carrier excel file will cause issues. The method is currently commented out of the Carriers controller class.

## 10.4 Shipment Dates

At this current build, shipmate dates are selected using a calendar input. This input allows the use of past dates. If a shipper created a shipment for a date in the past this could cause issues with the Carrier who accepted to deliver the shipment.

## 10.5 Recovery of Account

No recovery method was implemented for our application. If a user were to forget their username/password there would be no way to recover their account. The same could be said for hacked accounts.

# 11. Documentation

**Class description:**

edu.sru.thangiah.webrouting

**WebroutingApplication.java** – Used to start the Spring Application

↳config

**WebSecurityConfig.java** – Sets up the Spring Security with Thymeleaf, redirects user to login page unless page is allowed to a user, handles user authentication.

↳controller

**BidsController.java** – Handles the Thymeleaf controls for the pages dealing with bids.

**CarriersController.java** – Handles the Thymeleaf controls for the pages dealing with carriers.

**ContactsController.java** – Handles the Thymeleaf controls for pages dealing with contacts.

**DriverController.java** – Handles the Thymeleaf controls for the pages dealing with drivers.

**LocationsController.java** – Handles the Thymeleaf controls for the pages dealing with locations.

**LoginController.java** – Handles the Thymeleaf controls for the pages dealing with the login / registration functionality.

**MaintenanceOrderController.java** – Handles the Thymeleaf controls for the pages dealing with maintenance orders.

**RolesController.java** – Handles the Thymeleaf controls for the pages dealing with roles.

**RoutesController.java** – Handles the Thymeleaf controls for the routes page.

**ShipmentsController.java** – Handles the Thymeleaf controls for the pages dealing with shipments.

**TechniciansController.java** – Handles the Thymeleaf controls for the pages dealing with technicians.

**UserController.java** – Handles the Thymeleaf controls for the pages dealing with users.

**VehicleTypesController.java** – Handles the Thymeleaf controls for the pages dealing with Vehicle Types.

**VehicleController.java** – Handles the Thymeleaf controls for the pages dealing with the vehicles.

↳domain

**Bids.java** – Sets up the bids database.

**Carriers.java** – Sets up the carriers database.

**Contacts.java** – Sets up the contacts database.

**Driver.java** – Sets up the drivers database.

**Locations.java** – Sets up the locations database.

**MaintenanceOrders.java** – Sets up the maintenance orders database.

**Role.java** – Sets up the roles database.

**Shipments.java** – Sets up the shipments database.

**Technicians.java** – Sets up the technicians database.

**User.java** – Sets up the users database.

**VehicleTypes.java** – Sets up the vehicle types database.

**Vehicles.java** – Sets up the vehicles database.

↳error

**MyAccessDeniedHandler.java** – Logs when a user accesses a page without permission. Also redirects a user to the 403 access denied page (403.html)

↳repository

**BidsRepository.java** – Sets up the bids repository using CrudRepository

**CarriersRepository.java** – Sets up the carriers repository using CrudRepository

**DriverRepository.java** – Sets up the drivers repository using CrudRepository

**LocationsRepository.java** – Sets up the locations repository using CrudRepository

**MaintenanceOrdersRepository.java** – Sets up the maintenance orders repository using CrudRepository

**RoleRepository.java** – Sets up the roles repository using JpaRepository

**ShipmentsRepository.java** – Sets up the shipments repository using CrudRepository

**TechniciansRepository.java** – Sets up the technicians repository using CrudRepository

**UserRepository.java** – Sets up the users repository using JpaRepository

**VehicleTypesRepository.java** – Sets up the vehicle types repository using CrudRepository

**VehiclesRepository.java** – Sets up the vehicles repository using CrudRepository

↳services

**SecurityService.java** – Sets up the isAuthenticated and autoLogin methods

**SecurityServiceImpl.java** – Implements the SecurityService interface class. Used as a service for Spring Security

**UserDetailsServiceImpl.java** – Implements the UserDetailsService interface class. Used as a service for Spring Security

**UserService.java** – Interface class for the save and findByUser methods

**UserServiceImpl.java** – Implements the UserService interface class. Used as a service for Spring Security

↳web

**UserValidator.java** – Used to validate a user by checking their username and password

# 12. File Path Names

webrouting

↳src

↳main

↳java/edu/sru/thangiah/webrouting

**WebroutingApplication.java**

↳config

**WebSecurityConfig.java**

* accessDeniedHandler : AccessDeniedHandler
* userDetailsService : UserDetailsService

↳controller

**BidsController.java**

* userService : UserService
* securityService : SecurityService
* bidsRepository : BidsRepository
* shipmentsRepository : ShipmentsRepository
* carriersRepository : CarriersRepository

**CarriersController.java**

* userService : UserService
* securityService : SecurityService
* carriersRepository : CarriersRepository

**ContactsController.java**

* contactsRepository : ContactsRepository
* userService : UserService
* securityService : SecurityService

**DriverController.java**

* driverRepository : DriverRepository
* userService : UserService
* securityService : SecurityService

**LocationController.java**

* locationsRepository : LocationsRepository
* userService : UserService
* securityService : SecurityService

**LoginController.java**

* userService : UserService
* securityService : SecurityService
* userValidator : UserValidator
* carriersRepository : CarriersRepository
* contactsRepository : ContactsRepository

**MaintenanceOrderController.java**

* maintenanceOrderRepository : MaintenanceOrderRepository
* techniciansRepository : TechniciansRepository
* driverRepository : DriverRepository
* vehicleRepository : VehicleRepository
* userService : UserService
* securityService : SecurityService

**RolesController.java**

* roleRepository : RoleRepository
* userService : UserService
* securityService : SecurityService

**RoutesController.java**

* vehiclesRepository : VehiclesRepository

**ShipmentsController.java**

* userService : UserService
* securityService : SecurityService
* carriersRepository : CarriersRepository
* shipmentsRepository : ShipmentsRepository
* vehiclesRepository : VehiclesRepository
* bidsRepository : BidsRepository

**TechniciansController.java**

* techniciansRepository : TechniciansRepository
* userService : UserService
* securityService : SecurityService

**UserController.java**

* userService : UserService
* securityService : SecurityService
* userValidator : UserValidator
* userRepository : UserRepository
* roleRepository : RoleRepository
* carriersRepository : CarriersRepository
* contactsRepository : ContactsRepository

**VehicleTypesController.java**

* vehicleTypesRepository : VehicleTypesRepository

**VehiclesController.java**

* vehiclesRepository : VehiclesRepository
* vehicleTypesRepository : VehicleTypesRepository
* userService : UserService
* securityService : SecurityService

↳domain

**Bids.java**

* id : Long
* price : String
* date : String
* time : String
* shipment : Shipments
* carrier : Carriers

**Carriers.java**

* id : Long
* scac : String
* ltl : String
* ftl : String
* pallets : String
* weight : String
* contacts : List<Contacts>
* shipments : List<Shipments>
* bids : List<Bids>
* locations : List<Locations>
* vehicles : List<Vehicles>
* drivers : List<Drivers>
* orders : List<MaintenanceOrders>

**Contacts.java**

* id : Long
* firstName : String
* lastName : String
* middleInitial : String
* emailAddress : String
* streetAddress1 : String
* streetAddress2 : String
* city : String
* state : String
* zip : String
* primaryPhone : String
* workPhone : String
* carrier : Carrier
* drivers : List<Drivers>
* technicians : List<Technicians>

**Driver.java**

* id : Long
* contact : Contacts
* carrier : Carriers
* vehicle : Vehicles
* lisence\_number : String
* lisence\_expiration : String
* lisence\_class : String

**Locations.java**

* id : Long
* name : String
* streetAddress1 : String
* streetAddress2 : String
* city : String
* state : String
* zip : String
* latitude : String
* longitude : String
* carrier : Carriers
* locationType : String
* vehicles : List<Vehicles>

**MaintenanceOrders.java**

* id : Long
* technician : Technicians
* scheduled\_date : String
* details : String
* service\_type\_key : String
* cost : String
* status\_key : String
* vehicle : Vehicles
* maintenance\_type : String
* carrier : Carriers

**Role.java**

* id : Long
* name : String
* users : List<User>

**Shipments.java**

* id : Long
* client : String
* carrier : Carriers
* scac : String
* clientMode : String
* shipDate : String
* freightbillNumber : String
* paidAmount : String
* fullFreightTerms : String
* commodityClass : String
* commodityPieces : String
* commodityPaidWeight : String
* shipperCity : String
* shipperState : String
* shipperZip : String
* shipperLatitude : String
* shipperLongitude : String
* consigneeCity : String
* consigneeState : String
* consigneeZip : String
* consigneeLatitude : String
* consigneeLongitude : String
* vehicle : Vehicles
* user : User
* bids : List<Bids>

**Technicians.java**

* id : Long
* contact : Contacts
* skill\_grade : String
* orders : List<MaintenanceOrders>

**User.java**

* id : Long
* username : String
* password : String
* email : String
* role : Role
* carrier : Carriers
* shipments : List<Shipments>

**VehicleTypes.java**

* id : Long
* type : String
* subType : String
* description : String
* make : String
* model : String
* minimumWeight : int
* maximumWeight : int
* capacity : String
* maximumRange : int
* restrictions : String
* height : int
* emptyWeight : int
* length : int
* minimumCubicWeight : int
* maximumCubicWeight : int
* vehicles : List<Vehicles>

**Vehicles.java**

* id : Long
* plateNumber : String
* vinNumber : String
* manufacturedYear : String
* vehicleType : VehicleTypes
* location : Locations
* carrier : Carriers
* orders : List<MaintenanceOrders>
* shipments : List<Shipments>
* drivers : List<Drives>

↳error

**MyAccessDeniedHandler.java**

* logger : Logger

↳repository

**BidsRepository.java**

**CarriersRepository.java**

**ContactsRepository.java**

**DriverRepository.java**

**LocationsRepository.java**

**MaintenanceOrdersRepository.java**

**RoleRepository.java**

**ShipmentsRepository.java**

**TechniciansRepository.java**

**UserRepository.java**

**VehicleTypesRepository.java**

**VehiclesRepository.java**

↳services

**SecurityService.java**

**SecurityServiceImpl.java**

* authenticationManager : AuthenticationManager
* userDetailsService : UserDetailsService
* logger : Logger

**UserDetailsServiceImpl.java**

* userRepository : UserRepository

**UserService.java**

**UserServiceImpl.java**

* userRepository : UserRepository
* bCryptPasswordEncode : BCryptPasswordEncoder

↳web

**UserValidator.java**

* userService : UserService

# 13. Code Reusability

We believe that our program was designed to allow code reusability. Our domain classes are built with the idea of multiple different entities of the same type. It would be easy to add or remove attributes belonging to an entity. Our controller classes also work well with any domains with the same name. Some changes may need to be made in certain parameters passed in certain methods, depending on the attributes of the entity. Even with this slight change our controllers' classes should be easily implemented with other entities.

# 14. Testing

## 14.1 Human – Computer Testing

## 14.2 Blackbox Testing

## 14.3 Whitebox Testing

## 14.4 Unit Testing

## 14.5 Integration Testing

## 14.6 Regression Testing

## 14.7 Boundary Analysis

## 14.8 Volume Test

## 14.9 Stress Test

## 14.10 Missing Resources

## 14.11 Mis-Management of Resources

## 14.12 Recovering from Crashes

# 15. Logger

The logger class was implemented in both testing as well as the final project. In the final project, the logger class is implemented into the SecurityServiceImpl class as well as the MyAccessDeniedHandler class. In the SecurityServiceImpl class, the logger is implemented to signify when the auto login method is called so it logs when the user logs in. In the MyAccessDeniedHandler class, the logger is used to log when a user attempts to connect to a page that they do not have access to. The logger displays the username and the URL they tried to connect to. Implementing these logs has proven to be beneficial in terms of security as they log the most important things and identify that the program is working as it should.

When implementing new features into the program and testing them, the logger class was used many times when testing different algorithms. For example, when implementing the shipments class, all the shipments must be looped through to ensure they do not have a carrier associated with them. When implementing that method, it was important to use the logger to see what shipments, and how many shipments were displayed to the user. The logger class has been used many other times during testing and implementation as well to ensure that different parts of the program are working properly before deploying the application. The logger class proved to be a critical part of the development of this application.

# 16. Deployment / Maintenance

To get started with setting up this application, we must first download it. There are several ways to do this which will be discussed in this section.

## 16.1 Downloading Project from GitHub

The first method we will discuss is downloading the application straight from GitHub. We will be downloading a zip folder that contains all the source code to the application which then can be opened in Eclipse or any other IDE of your choosing.

### 16.1.1 Finding the GitHub Repository

Finding the GitHub repository containing this application can simply be found by clicking [here](https://github.com/samthangiah/Spring-2022---Spring-Web-Routing) or by clicking this link: <https://github.com/samthangiah/Spring-2022---Spring-Web-Routing>

**Note:** A GitHub account may be required in order to access and download this repository!

### 16.1.2 Downloading the Project

After accessing the GitHub repository, make sure you have the ***Code*** tab selected at the top. In the middle of your screen, you should see a green ***Code*** button with an arrow beside it (**Figure 17**). Click that button and choose ***Download ZIP*** (**Figure 18**). A zip file will then be downloaded containing all of the source code for the application.



Figure 17 – Code Button

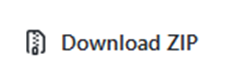


Figure 18 – Download ZIP Button

### 16.1.3 Opening the Project in Eclipse

Now that the zip file is downloaded, navigate to the location where you downloaded it to and extract that zip file. You should be left with a folder titled ***Spring-2022---Spring-Web-Routing***

Inside of that folder, they may be another folder inside titled the exact same thing. Simply move the contents of that folder to the parent folder so when you open ***Spring-2022-Spring-Web-Routing*** you will see ***.metadata***, ***webrouting***, and ***.project***.

Next, open Eclipse IDE. When Eclipse asks for a workspace, navigate to and use the ***Spring-2022-Web-Routing*** folder as the workspace and choose ***Launch*** (**Figure 19**)

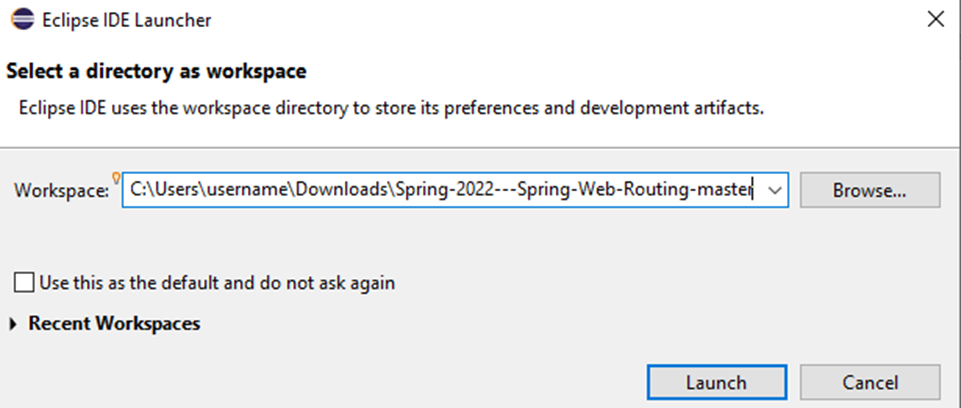


Figure 19 – Launch Eclipse Workspace

Eclipse will then launch and you will see the webrouting application on the side (Figure 20). Before proceeding, you may need to update the project’s dependencies. To do so, right-click on ***webrouting*** select ***Maven*** then choose ***Update Project*** and press ***OK*** (**Figure 21**). The dependencies for the project will then be downloaded and updated if necessary.

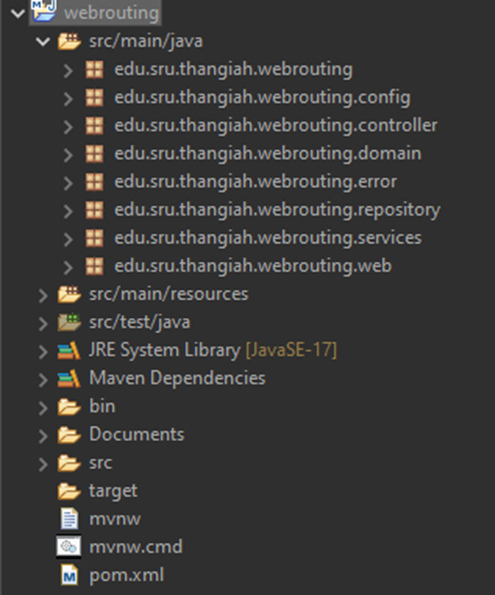


Figure 20 – Webrouting Application Inside of Eclipse

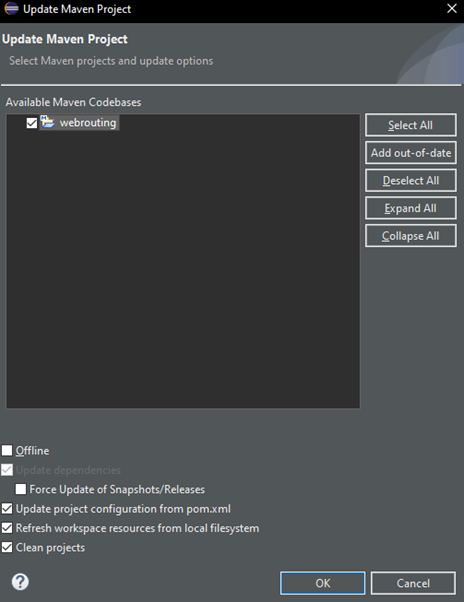


Figure 21 – Update Project Menu

## 16.2 Cloning Project into Eclipse

Another option to get the application files is to clone the GitHub repository straight into Eclipse. The method outlined in this section is specific to the Eclipse IDE and will not work for any other IDE that you may be using.

### 16.2.1 Cloning Project via URI

Start Eclipse and launch any workspace of your choosing. In the top left corner, select ***File*** then choose ***Import*** (**Figure 22**). From there, the ***Import*** window will open up. Choose ***Git*** and select ***Projects from Git (with smart import)*** then choose ***Next*** (**Figure 23**)

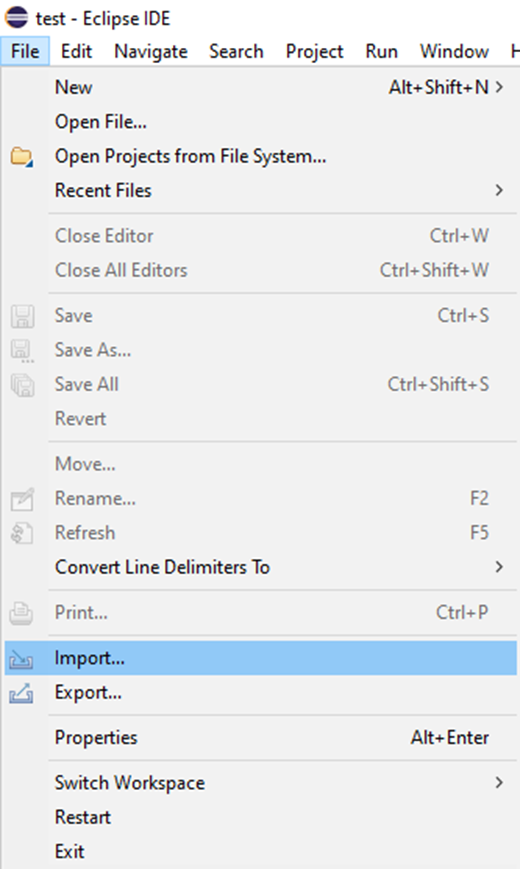


Figure 22 – Location of the Import Button

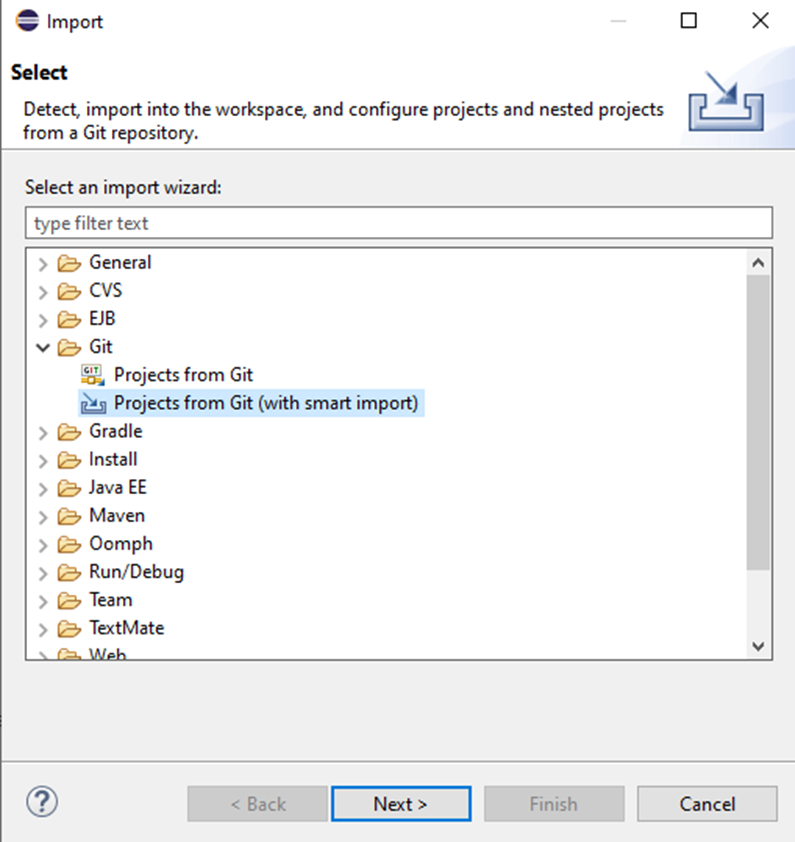


Figure 23 – Location of the Git Import Button

In the ***Select Repository Source*** window, choose ***Clone URI*** then select ***Next***.

The ***Source Git Repository*** window will open and in the ***URI*** field enter this URI: [***https://github.com/samthangiah/Spring-2022---Spring-Web-Routing.git***](https://github.com/samthangiah/Spring-2022---Spring-Web-Routing.git)

The ***Host*** and ***Repository path*** will automatically be filled with the correct information. The ***Protocol*** will also automatically be set to ***HTTPS***. The ***Port*** field does not need to be filled. The only fields left to be filled out are the ***User*** and ***Password***. For these, just enter your GitHub credentials. Make sure all of the information is correct then choose ***Next*** (**Figure 24**)

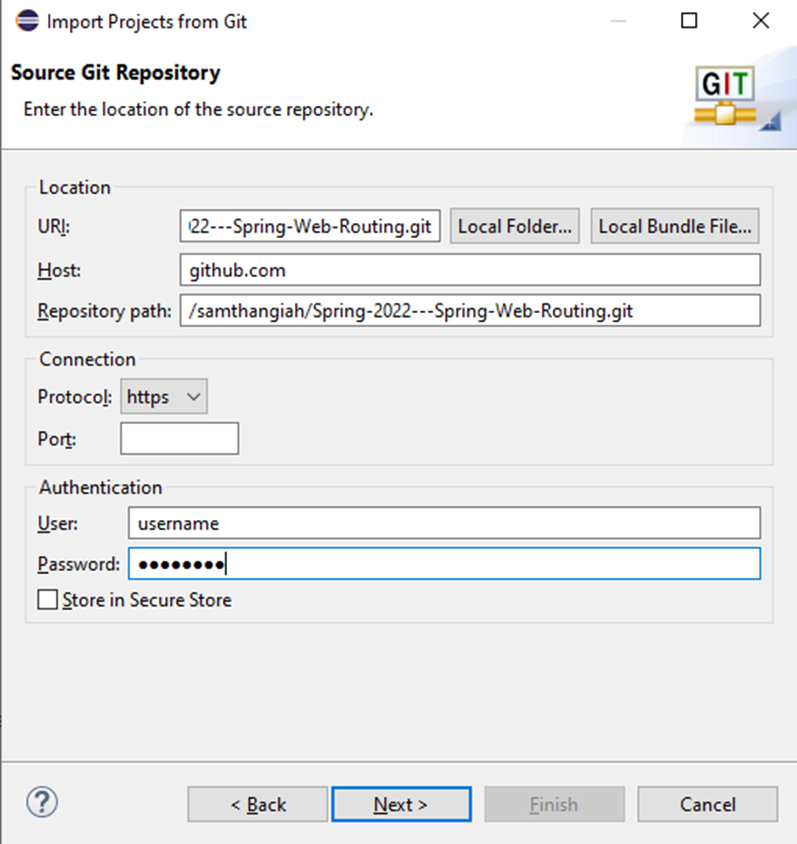


Figure 24 – Source Git Repository Menu

On the ***Branch Selection*** page, make sure all branches are selected and select ***Next*** (**Figure 25**)

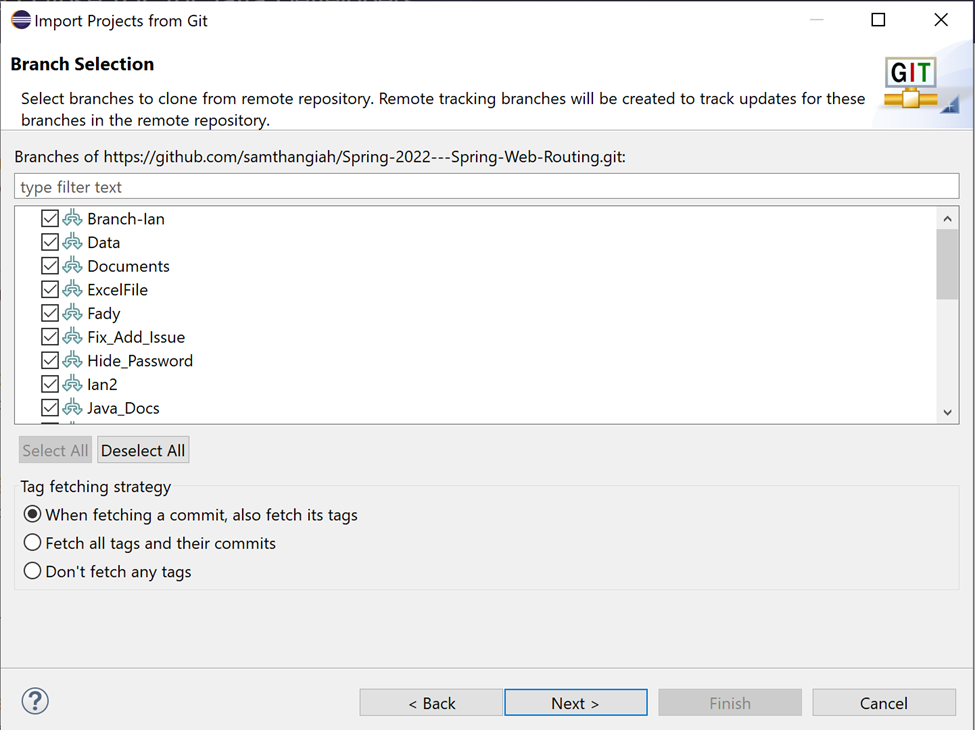


Figure 25 – Branch Selection Page

On the ***Local Destination*** page, choose the directory that the repository clone will be saved in and select ***Next***.

The project will then be cloned to the destination. Make sure that both ***Spring-2022---Spring-Web-Routing*** AND ***Spring-2022---Spring-Web-Routing\webrouting*** are selected and choose ***Finish***

2 Projects will be added to the ***Package Explorer*** section of Eclipse. To access the application, choose the ***webrouting***. It may be a good idea to update the project. To do so, right-click on ***webrouting*** select ***Maven*** then choose ***Update Project*** and press ***OK***. The dependencies for the project will then be downloaded and updated if necessary.

## 16.3 Setting up the MySQL Database

Setting up a MySQL database is crucial to getting this project to work properly. Without it being setup properly, the application will not load and you will be completely unable to access the application. The following guide will show you how to install MySQL and MySQL Workbench. Of course, there are other options to do this but the overall setup should be very similar.

## 16.3.1 Installing MySQL & MySQL Workbench

Download the MySQL installer from <https://dev.mysql.com/downloads/>

Ensure that version 8.0.28 is downloaded.

Run the installer and when choosing a setup type, be sure to select ***Custom***

On the ***Select Products*** page, make sure to choose ***MySQL Server 8.0.28***, ***MySQL Workbench 8.0.28***, and ***MySQL Shell 8.0.28***

The next options such as ***Port***, ***Root Password***, and other settings can be set at your own discretion. For a more detailed guide on installing MySQL Server and MySQL Workbench, visit [this link.](https://www.simplilearn.com/tutorials/mysql-tutorial/mysql-workbench-installation)

### 16.3.2 Loading MySQL Data

To load the MySQL data, start ***MySQL Workbench*** and login with your credentials. After logging in, click ***File*** and choose ***Open SQL Script***. (**Figure 26**) Alternatively, you can use ***CTRL + SHIFT + O***

Choose the ***webrouting.sql*** file located in the ***Documents/*** folder of the application.

After opening, that file press the lightning bolt at the top of the screen (**Figure 27**)

The script will then run and the database will be successfully set up and ready to be configured in your application properties.

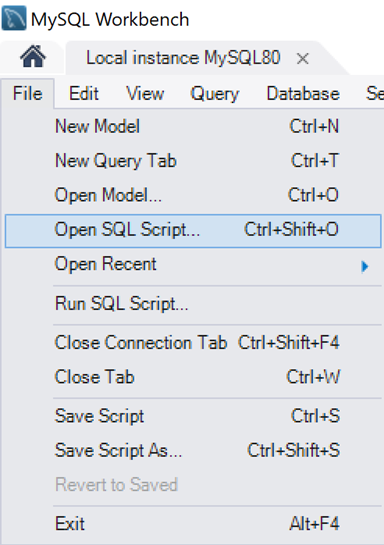


Figure 26 – Open SQL Script



Figure 27 – Run Script Button

### 16.3.3 Changing Application Properties

To access the application.properties file, go to the ***src/main/resources/application.properties*** file and open it either in Eclipse or with a text editor.

Within this folder, you will want to be sure that the ***datasource url***, ***username***, and ***password*** are changed to be properly used with your MySQL Server (**Figure 28**).

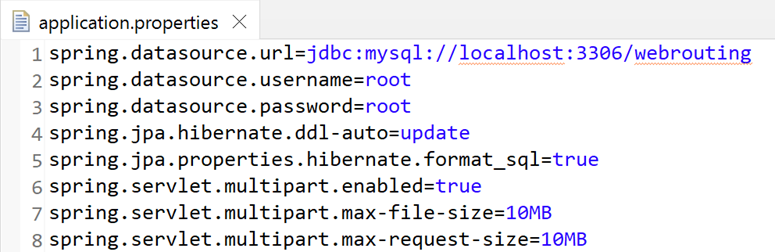


Figure 28 – Application Properties File

Once changing the properties within this file, you should be able to successfully start the application and access the application.

## 16.4. Starting the Application

This section will outline how to start the application through Eclipse as well as the proper location to access the application from when it is started.

### 16.4.1 Where to Start Application From

Within Eclipse, navigate to ***src/main/java/edu.sru.thangiah.webrouting/WebroutingApplication.java***

Right-click on that file and choose ***Run As*** then ***Java Application*** (**Figure 29**).

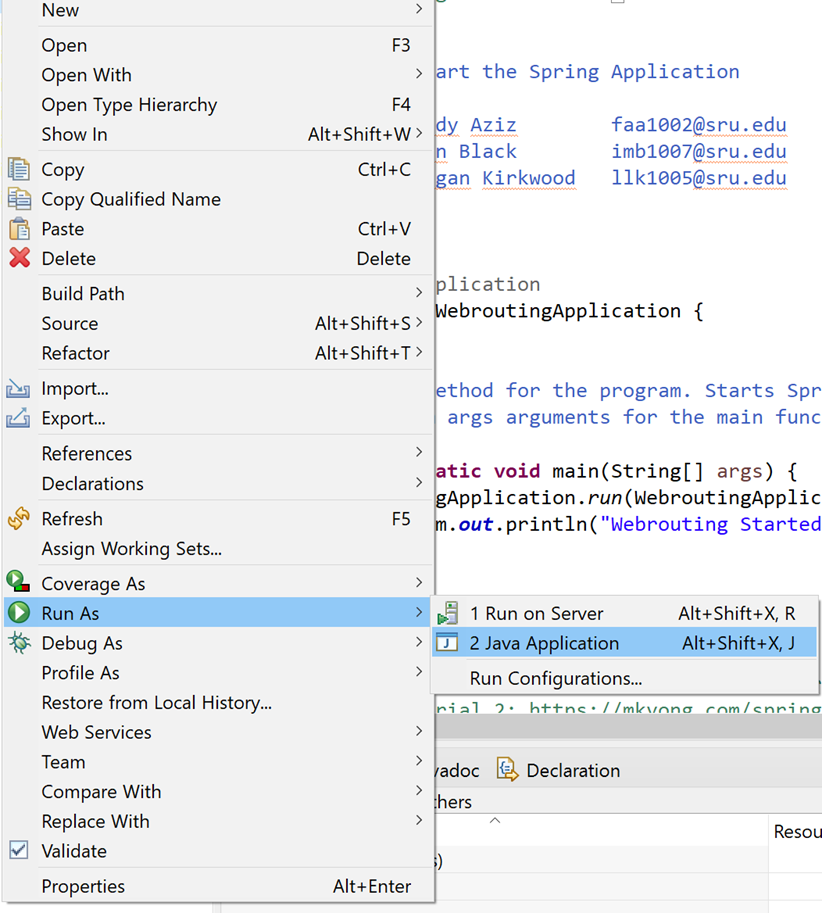


Figure 29 – Run Webrouting Application

The application will then be started. Keep an eye on the console within Eclipse and ensure the message ***Webrouting Started!*** Is displayed before trying to properly access the application.

### 16.4.2 Where to Access Application From

To access the application, open up any web browser of your choice and navigate to <http://localhost:8080/>

After navigating to that page, you will be shown the ***Enterprise Routing System*** home page (**Figure 30**)



Figure 30 – Home Page of Enterprise Routing System

# 17. Post-Mortem Analysis

Our group would have liked to implement more on the security of our program. We currently have no account recovery methods for users who lost their accounts. Our project also had more of a clear purpose near the end of the semester. Our entities did not have a clear purpose until we added a bidding system. Some entities were changed significantly when we were assigned the bidding system. We believe that we should have spent more time learning how other routing systems work. We originally had issues with getting the previous project to run. If we had had more time testing out the old project, we would have had a better idea of our goal.

The start of the project consisted more of learning new technologies than using our creative ideas. As the project went further into development we started to test and improve the design of our project off our own ideas and creativity. With more of an understanding of these technologies we would love to start completely over in order to have more time to create a bigger and much more efficient program.

# 18. Figures

[Figure 1 – Drivers Sequence Diagram 12](#_Toc101781481)

[Figure 2 – Maintenance Orders Diagram 13](#_Toc101781482)

[Figure 3 – Shipments Sequence Diagram 14](#_Toc101781483)

[Figure 4 – Accept Bid Sequence Diagram 15](#_Toc101781484)

[Figure 5 – Create Bid Sequence Diagram 16](#_Toc101781485)

[Figure 6 – Drivers Activity Diagram 16](#_Toc101781486)

[Figure 7 – Update Maintenance Order Activity Diagram 17](#_Toc101781487)

[Figure 8 – Shipments Activity Diagram 18](#_Toc101781488)

[Figure 9 – Add Bid Activity Diagram 19](#_Toc101781489)

[Figure 10 – Accept Bid Activity Diagram 20](#_Toc101781490)

[Figure 11 – Drivers State Diagram 21](#_Toc101781491)

[Figure 12 – Maintenance Orders State Diagram 21](#_Toc101781492)

[Figure 13 – Shipments State Diagram 21](#_Toc101781493)

[Figure 14 – Shipments Data Flow Diagram as a Carrier 22](#_Toc101781494)

[Figure 15 – Shipments Data Flow Diagram as a Shipper 22](#_Toc101781495)

[Figure 16 – Entity / Relationship Diagram 23](#_Toc101781496)

[Figure 17 – Code Button 38](#_Toc101781497)

[Figure 18 – Download ZIP Button 38](#_Toc101781498)

[Figure 19 – Launch Eclipse Workspace 39](#_Toc101781499)

[Figure 20 – Webrouting Application Inside of Eclipse 40](#_Toc101781500)

[Figure 21 – Update Project Menu 41](#_Toc101781501)

[Figure 22 – Location of the Import Button 42](#_Toc101781502)

[Figure 23 – Location of the Git Import Button 43](#_Toc101781503)

[Figure 24 – Source Git Repository Menu 44](#_Toc101781504)

[Figure 25 – Branch Selection Page 45](#_Toc101781505)

[Figure 26 – Open SQL Script 46](#_Toc101781506)

[Figure 27 – Run Script Button 46](#_Toc101781507)

[Figure 28 – Application Properties File 47](#_Toc101781508)

[Figure 29 – Run Webrouting Application 48](#_Toc101781509)

[Figure 30 – Home Page of Enterprise Routing System 49](#_Toc101781510)