Offerly Online Auction

Final Report



**Slippery Rock University of Pennsylvania**

Contributions by:

**Fall 2023**

Timothy Holtz – tth1003@sru.edu

Douglas Maxwell – dsm1015@sru.edu

Luke McElligott – lpm1006@sru.edu

Table of Contents

[References 4](#_Toc152182424)

[Completion 5](#_Toc152182425)

[Contribution 6](#_Toc152182426)

[External Sources 7](#_Toc152182427)

[Glossary 10](#_Toc152182428)

[Problem Explanation 12](#_Toc152182429)

[Solution 12](#_Toc152182430)

[Scope 13](#_Toc152182431)

[Products: 13](#_Toc152182432)

[Technology Stack: 13](#_Toc152182433)

[Recommended Features: 13](#_Toc152182434)

[Terms: 13](#_Toc152182435)

[System Requirements 14](#_Toc152182436)

[Specifications: 14](#_Toc152182437)

[UML Diagrams 15](#_Toc152182438)

[Class Diagrams: 15](#_Toc152182439)

[Use Case Diagrams: 15](#_Toc152182440)

[Activity Diagrams: 15](#_Toc152182441)

[Sequence Diagrams: 16](#_Toc152182442)

[DFD Diagrams 18](#_Toc152182443)

[Entity/Relationship Diagrams 18](#_Toc152182444)

[User 19](#_Toc152182445)

[Bid 20](#_Toc152182446)

[MarketListing 21](#_Toc152182447)

[Widget 22](#_Toc152182448)

[Social Messaging Websocket 23](#_Toc152182449)

[Caveats/Minefields 24](#_Toc152182450)

[Caveats with working solutions 24](#_Toc152182451)

[E-Commerce Categories 24](#_Toc152182452)

[Shipment Tracking & Address Validation 24](#_Toc152182453)

[Tomcat Deployment 24](#_Toc152182454)

[User Sessions 24](#_Toc152182455)

[Websockets 25](#_Toc152182456)

[Context Paths 25](#_Toc152182457)

[Minefields 25](#_Toc152182458)

[Documentation 26](#_Toc152182459)

[File Path Names 27](#_Toc152182460)

[OO Class Design of File Paths 28](#_Toc152182461)

[Code Reusability 29](#_Toc152182462)

[Testing 30](#_Toc152182463)

[Testing Plan 30](#_Toc152182464)

[Unit Testing 30](#_Toc152182465)

[Integration Testing 30](#_Toc152182466)

[Automated Tests Results 30](#_Toc152182467)

[User Acceptance Testing 31](#_Toc152182468)

[Logger 33](#_Toc152182469)

[User Action Logging 33](#_Toc152182470)

[Logging in Deployed Environment 33](#_Toc152182471)

[Deployment/Maintenance 34](#_Toc152182472)

[Overview: 34](#_Toc152182473)

[Deployment Process: 34](#_Toc152182474)

[Automated Build with GitHub Actions: 34](#_Toc152182475)

[Manual Deployment to Tomcat Server: 34](#_Toc152182476)

[Maintenance: 35](#_Toc152182477)

[Monitoring and Logging: 35](#_Toc152182478)

[Updates and Patches: 35](#_Toc152182479)

[Backup and Recovery: 35](#_Toc152182480)

[Port-Mortem Analysis 38](#_Toc152182481)

# References

Braun, D., Crider, G., Morrow, N., Thangiah, S. Dr. (2023). *Evaluation Manual*. Completion documentation, Slippery Rock University of Pennsylvania, Slippery Rock.

Braun, D., Crider, G., Morrow, N., Sanders, N., Thangiah, S. Dr. (2023). *Install Manual*. Installation documentation, Slippery Rock University of Pennsylvania, Slippery Rock.

Braun, D., Crider, G., Morrow, N., Sanders, N., Thangiah, S. Dr. (2023). *Security Manual*. Security documentation, Slippery Rock University of Pennsylvania, Slippery Rock.

Braun, D., Crider, G., Morrow, N., Sanders, N., Thangiah, S. Dr. (2023). *Technical* *Manual*. Technical documentation, Slippery Rock University of Pennsylvania, Slippery Rock.

Braun, D., Crider, G., Morrow, N., Thangiah, S. Dr. (2023). *Testing Document*. Testing documentation, Slippery Rock University of Pennsylvania, Slippery Rock.

Braun, D., Crider, G., Morrow, N., Thangiah, S. Dr. (2023). *Testing Manual*. Testing documentation, Slippery Rock University of Pennsylvania, Slippery Rock.

Braun, D., Crider, G., Morrow, N., Thangiah, S. Dr. (2023). *User Manual*. User documentation, Slippery Rock University of Pennsylvania, Slippery Rock.

# Completion

Please see the following table for our completion status on modules for this project.

|  |  |  |
| --- | --- | --- |
| **Module** | **Status** | **Description** |
| Billing | Complete | Includes credit cards, direct deposit, billing addresses, state taxes, and website percentages for sales |
| Market Listing | Complete | Includes widgets, categories, attributes, images, and the user’s ability list these widgets |
| Watchlist | Complete | Includes adding listing to watchlist, getting updates, and removing listing. |
| Transaction | Complete | Includes the user’s ability to purchase listings and request refunds. |
| Auction | Complete | Includes the ability to bid and autobid on listings. |
| Shipment Tracking | Complete | After making a transaction, seller can upload shipping label and buyer can track the shipment. |
| Admin | Complete | Includes managing accounts, listings, viewing logs, and tickets |
| Ticketing | Complete | Includes IT staff assignment, user ticket submission, status, and messaging. |
| Social | Complete | Includes friends, messaging, and friend search. |
| Profile Management | Complete | Includes the ability to update user profile, image, and payment/shipping details. |
| Security | Complete | Includes 2FA, email verification, user roles, login mechanism, database hashing, and other security configurations |
| Testing | In-Progress | Includes unit-testing, integration testing, UAT, and black/white box testing. Incomplete do to lack of code coverage. |

# Contribution

While the group made use of external resources when unaware of how to go about solving a problem or for guidance, nearly all of the code written was developed by our group members. Resources and external sources were mostly used in order to help our group arrive at the correct solution rather than giving the group the right answer. Because of this, it is our estimate that 95% of the code for the online auction was written by our group with 5% of the code being from external sources.

Our group had not relied on code from external sources and only resorted to external code when implementing certain features. One example of this was when our group was attempting to implement a magnifying class feature when viewing a market listing. This magnifying glass would recognize where the user’s cursor was on the screen and when hovering over a listing image, the system would magnify the image for easier viewing. Because our team did not have the relevant JavaScript experience to accomplish this task easily, we turned to an external source (<https://www.w3schools.com/howto/howto_js_image_magnifier_glass.asp>). While this feature did not end up making the final version of the application, this was an example of how our team turned to external sources.

Another external source that had greatly helped with the development of the project was ChatGPT (<https://chat.openai.com/chat>). Our team found ChatGPT to be a helpful resource when troubleshooting and debugging our code. The typical use case for including ChatGPT in our team’s workflow was to develop code individually or by referring to other resources first. If a suitable solution was not found, we would turn to ChatGPT with our question. This often led us to the solution of the problem. Another use case would be when we encounter problems during or after the implementation of a feature. ChatGPT could be used to discover potential bugs or errors in our code. While we found ChatGPT to be useful for explaining code or concepts, it was certainly not the end-all solution in the majority of cases. ChatGPT was used mostly for gaining knowledge and deeper understanding of the problem that we needed to solve.

The following section includes a complete list of the external sources used by our team. While code was not directly taken from every one of these sources, if our team did utilize code from an outside source, it is cited below.

# External Sources

andrewJames. “Answer to ‘Dynamically Create Variable in Thymeleaf.’” *Stack Overflow*, 2 Dec. 2021, <https://stackoverflow.com/a/70199965>.

Andrius. “Java - Java.Lang.NoSuchMethodException.” *Stack Overflow*, 11 Nov. 2013, <https://stackoverflow.com/q/19913970>.

*Authentication Persistence and Session Management :: Spring Security*. <https://docs.spring.io/spring-security/reference/servlet/authentication/session-management.html>. Accessed 27 Nov. 2023.

“Automatic Bidding.” *eBay*, <https://www.ebay.com/help/buying/bidding/automatic-bidding?id=4014>. Accessed 27 Nov. 2023.

baeldung. *Testing in Spring Boot | Baeldung*. 26 Apr. 2017, <https://www.baeldung.com/spring-boot-testing>.

*Build CI/CD Pipeline for Java Maven Using GitHub Actions #devops #technology #tech*. Directed by Thetips4you, 2023. *YouTube*, <https://www.youtube.com/watch?v=BqCe-nSXSGI>

*Chevron Down*. <https://icons.getbootstrap.com/icons/chevron-down/>. Accessed 27 Nov. 2023.

*Chevron Up*. <https://icons.getbootstrap.com/icons/chevron-up/>. Accessed 27 Nov. 2023.

Cocuthemyth. “Error during Execution of Processor ‘Org.Thymeleaf.Spring5.Processor.SpringInputGeneralFieldTagProcessor.’” *Stack Overflow*, 4 Nov. 2022, <https://stackoverflow.com/q/53743806>.

*ControllerAdvice (Spring Framework 6.1.1 API)*. <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/ControllerAdvice.html>. Accessed 27 Nov. 2023.

*CSS Object-Fit Property*. <https://www.w3schools.com/css/css3_object-fit.asp>. Accessed 27 Nov. 2023.

“Getting Started | Handling Form Submission.” *Getting Started | Handling Form Submission*, <https://spring.io/guides/gs/handling-form-submission/>. Accessed 27 Nov. 2023.

*Google Product Category [Google\_product\_category] - Google Merchant Center Help*. <https://support.google.com/merchants/answer/6324436?hl=en-GB>. Accessed 27 Nov. 2023.

*Guide on Loading Initial Data with Spring Boot | Baeldung*. <https://www.baeldung.com/spring-boot-data-sql-and-schema-sql>. Accessed 27 Nov. 2023.

"Hibernate ORM Documentation - 5.4." *Hibernate*, hibernate.org/orm/documentation/5.4/. Accessed 27 Nov. 2023.

*How Does Bidding and Automatic Bidding on eBay Work?* <https://techboomers.com/t/how-does-bidding-on-ebay-work>. Accessed 27 Nov. 2023.

“How to Call Private Method from Another Class in Java with Help of Reflection API?” *GeeksforGeeks*, 17 June 2019, <https://www.geeksforgeeks.org/how-to-call-private-method-from-another-class-in-java-with-help-of-reflection-api/>.

*How To Create Next and Previous Buttons*. <https://www.w3schools.com/howto/howto_css_next_prev.asp>. Accessed 27 Nov. 2023.

*How To Make a Modal Box With CSS and JavaScript*. <https://www.w3schools.com/howto/howto_css_modals.asp>. Accessed 27 Nov. 2023.

*How to Remove All Unused Imports in a Java File - Eclipse Shortcut Example*. <https://javarevisited.blogspot.com/2018/09/eclipse-shortcut-to-remove-all-unused-imports-java.html#axzz8KICFPusB>. Accessed 27 Nov. 2023.

*HTML Input Types*. <https://www.w3schools.com/html/html_form_input_types.asp>. Accessed 27 Nov. 2023.

*Introduction to Spring MVC HandlerInterceptor | Baeldung*. <https://www.baeldung.com/spring-mvc-handlerinterceptor>. Accessed 27 Nov. 2023.

Mark Otto, Jacob Thornton, and Bootstrap. *Cards*. <https://getbootstrap.com/docs/4.0/components/card/>. Accessed 27 Nov. 2023.

Mark Otto, Jacob Thornton, and Bootstrap. *Tables*. <https://getbootstrap.com/docs/4.0/content/tables/>. Accessed 27 Nov. 2023.

Maurya, Shirshak. “How to Add a Date Picker in HTML?” *Scaler Topics*, 12 Oct. 2022, <https://www.scaler.com/topics/date-picker-in-html/>.

“Method: validateAddress | Address Validation API.” *Google for Developers*, <https://developers.google.com/maps/documentation/address-validation/reference/rest/v1/TopLevel/validateAddress>. Accessed 27 Nov. 2023.

*MySQL :: MySQL 8.0 Reference Manual :: 11.2 Date and Time Data Types*. <https://dev.mysql.com/doc/refman/8.0/en/date-and-time-types.html>. Accessed 27 Nov. 2023.

OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model] <https://chat.openai.com/chat>.

*PlantUML Language Reference Guide*. *PlantUML*, 2023, <https://pdf.plantuml.net/PlantUML_Language_Reference_Guide_en.pdf>.

RS, Senthil. “Mask Email in Javascript.” *Stack Overflow*, 2 Aug. 2022, <https://stackoverflow.com/q/39247866>

“Selling Fees.” *eBay*, <https://www.ebay.com/help/selling/fees-credits-invoices/selling-fees?id=4822>. Accessed 27 Nov. 2023.

“Singleton Method Design Pattern in Java.” *GeeksforGeeks*, 5 Apr. 2017, <https://www.geeksforgeeks.org/singleton-class-java/>.

“Sequence Diagram Syntax and Features.” *PlantUML.Com*, <https://plantuml.com/sequence-diagram>. Accessed 27 Nov. 2023.

*Track & Confirm API.* <https://www.usps.com/business/web-tools-apis/track-and-confirm-api_files/track-and-confirm-api.htm>. Accessed 27 Nov. 2023.

*Tracking 101 | eBay Selling*. Directed by eBay for Business AU, 2023. *YouTube*, <https://www.youtube.com/watch?v=6ixSG8MvJgA>

*Tutorial: Using Thymeleaf*. <https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html>. Accessed 27 Nov. 2023.

*Thymeleaf Page Layouts - Thymeleaf*. <https://www.thymeleaf.org/doc/articles/layouts.html>. Accessed 27 Nov. 2023.

“Use API Keys with Address Validation API.” *Google for Developers*, <https://developers.google.com/maps/documentation/address-validation/get-api-key>. Accessed 27 Nov. 2023.

Vejani, Pathik. “Time Input Validation with Start Time and End Time Using Javascript.” *Stack Overflow*, 12 Aug. 2019, <https://stackoverflow.com/q/57455835>.

# Glossary

**API:** application programming interface is a set of rules, protocols, and tools for building software and applications.

**Build Tool:** A tool used for automating the creation of executable applications from source code (like Maven or Gradle).

**Continuous Deployment (CD):** A software release process that uses automated testing to validate if changes to a codebase are correct and stable for immediate autonomous deployment to a production environment.

**Continuous Integration (CI):** The practice of merging all developers' working copies to a shared mainline several times a day.

**Database Initialization:** The process of setting up a database schema and populating it with initial data. This process is crucial for preparing an environment where applications can perform database operations.

**Dependency Management:** The process of handling external libraries your project needs to function correctly.

**Docker:** A set of platform-as-a-service products that use OS-level virtualization to deliver software in packages called containers.

**Framework:** A platform for developing software applications. It provides a foundation on which software developers can build programs for a specific platform.

**Git:** A distributed version-control system for tracking changes in source code during software development.

**GitHub Actions with WAR File Artifact Generation:** A CI/CD feature in GitHub that automates workflows for building, testing, and deploying projects. In the context of Java web applications, this might involve automating the generation of a WAR (Web Application Archive) file, which can then be deployed to a servlet container like Tomcat.

**Integration Testing:** A level of software testing where individual units are combined and tested as a group. The purpose is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in integration testing.

**Java Bean:** A reusable software component in Java that follows a particular convention. It's a class that encapsulates many objects into a single object (the bean) and adheres to specific naming conventions. Java Beans are serializable, have a no-argument constructor, and allow access to properties using getter and setter methods. They are widely used in Java programming, particularly in Java Enterprise Edition, for encapsulating data and business logic, and they play a significant role in various frameworks like Spring.

**JDK (Java Development Kit):** A software development environment used for developing Java applications and applets. It includes the Java Runtime Environment (JRE), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), and other tools needed in Java development.

**JUnit:** A unit testing framework for the Java programming language.

**Live Server Deployment on Tomcat**: The process of deploying a web application to a Tomcat server so it's accessible in a production environment. Tomcat is a popular Java servlet container that's used in the Java enterprise ecosystem.

**Logger:** A tool or utility in software development used for logging information during the execution of a program. Loggers are crucial for debugging and monitoring the software's operational state. They provide insights into the behavior of an application by recording its runtime activities.

**Maven:** A software project management and comprehension tool that provides developers with a complete build lifecycle framework.

**Microservices:** An architectural style that structures an application as a collection of services that are highly maintainable and testable, loosely coupled, independently deployable, and organized around business capabilities.

**Mockito:** A popular Java-based mocking framework used in the development of test-driven and behavior-driven Java applications. It's used primarily for unit testing Java applications, allowing developers to create and configure mock objects.

**REST API:** A set of rules for building web services that allow for interaction with RESTful web services.

**Spring Boot:** A Java-based framework used for creating stand-alone, production-grade Spring applications with minimal fuss.

**Spring Security:** A powerful and highly customizable authentication and access-control framework for Spring applications. It's the de-facto standard for securing Spring-based applications.

**UAT Testing (User Acceptance Testing)**: The last phase of the software testing process, where end-users test the software to ensure it can handle required tasks in real-world scenarios, according to specifications.

**Unit Testing:** The process of testing individual components of the software separately to ensure that each part is working correctly. Unit testing is typically automated and is a fundamental part of test-driven development.

**WebSocket:** A communication protocol providing full-duplex communication channels over a single TCP connection, often used for real-time data transfer between a client and a server.

# Problem Explanation

Today, ecommerce is an incredibly lucrative and important industry. Much of the world relies on online applications for their everyday lives. The Online Auction platform was developed as a means to offer a centralized market for users. What separates our auction from others is the prioritization of transparency. Many other ecommerce platforms are interested in user data. Our goal was to put customer satisfaction first.

Originally, the goal was to develop an all-encompassing online retailer for a chemical company. When discussing this idea, our group was made aware that an online auction application had been worked on previously. It was recommended that we inherit the auction project as it shared many similarities with the original chemical company site we had discussed.

## Solution

Our solution involves the development of a complete web-based auction application. This platform features innovative and user-friendly transaction processes, making buying and selling as easy as possible. Simply, the solution to the group’s problem was to deliver a web-based application that allows users to sign up, list items for sale, and purchase items. We then built off this idea to deliver a complete ecommerce system. We wanted to implement features that helped the user with their auction experience.

Making the platform with the user in mind has been the highest priority and the application demonstrates this. Some of the key features include:

* Personalized user profiles
* Real-time and automatic bidding
* Offer price purchasing
* Social messaging
* Dynamic category tree
* 2-Factor authentication

Prioritizing user experience has resulted in the delivery of a top-of-the-line auction platform. By conducting weekly meetings, the most important features to the user were prioritized and eventually realized in the final project. The group had looked at similar platforms in order to determine what does and does not work.

We also made some great expansions with the DevOps aspect of the application. These areas include:

* Live Server deployment on Tomcat
* Migration to Spring Boot 3
* Migration to Spring Security 6
* Database initialization
* Testing
* Docker
* GitHub Actions with war file artifact generation

## Scope

### Products:

The online auction platform will be a web-hosted application made available to the public. The system will be capable of handling ecommerce activity for its user base.

### Technology Stack:

* HTML, CSS, and JavaScript
* Java
* Spring Boot
* Maven
* Thymeleaf
* SQL
* Tomcat
* GitHub

### Recommended Features:

* User registration and authentication
* List products for sale
* Offer price purchasing and bidding
* Watchlist for products
* Payment acceptance
* Product searching, filtering, and categorization
* Administrative tasks and permissions

### Terms:

* User – standard user account with normal selling/purchasing permissions.
* Admin – administrator account that oversees most of the system’s operations.
* Super admin – highest level administrator account to oversee admin and check the system’s revenue.
* Offer price purchasing – when a user purchases an item at its listed price.
* Bidding – a user can place a bid on an item and compete with other users to purchase a product.
* Auto bid – a user can specify the max value that they are willing to bid, so that automatic bidding can occur.

# System Requirements

### Specifications:

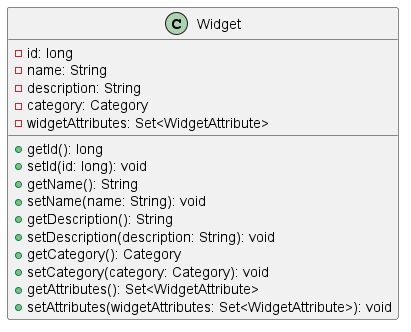
* Windows OS
* MySQL Workbench 8.x
* JDK: 17
* Tomcat 10.0.x
* Spring Boot 3.1.x

# UML Diagrams

A complete collection of relevant UML diagrams can be found within the project’s documentation folder. Included in this document are examples of UML diagrams for some of the most important features of the platform.

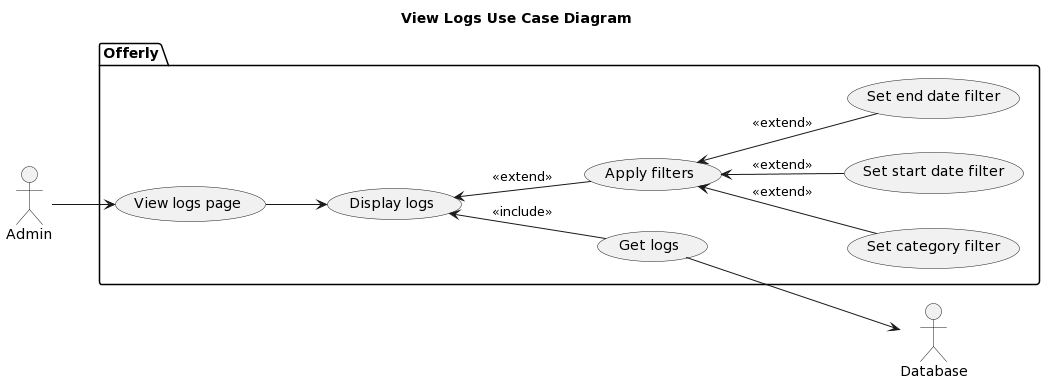
### Class Diagrams:

Class diagrams outline the Java class name, attributes, and methods. The example below is for the Widget class.



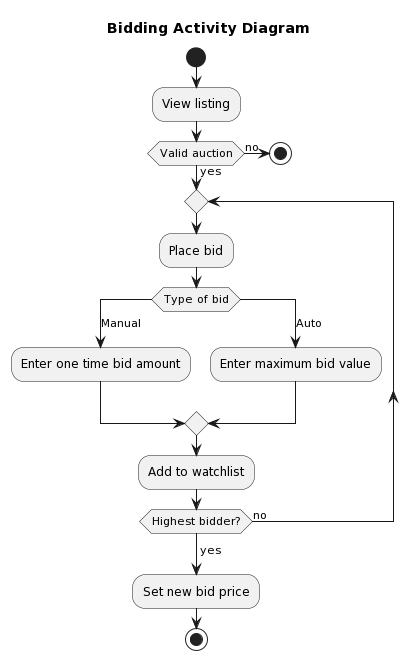
### Use Case Diagrams:

These diagrams explain how the system works from a high-level. The example below illustrates how an admin can view logs.



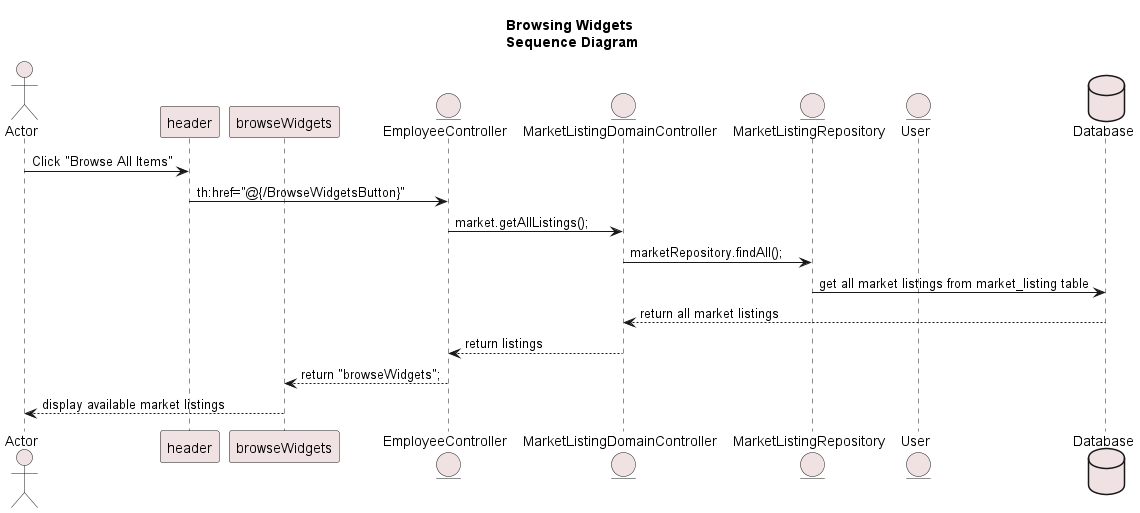
### Activity Diagrams:

Activity diagrams illustrate the flow of actions within the system. The example below outlines how bidding works for users.

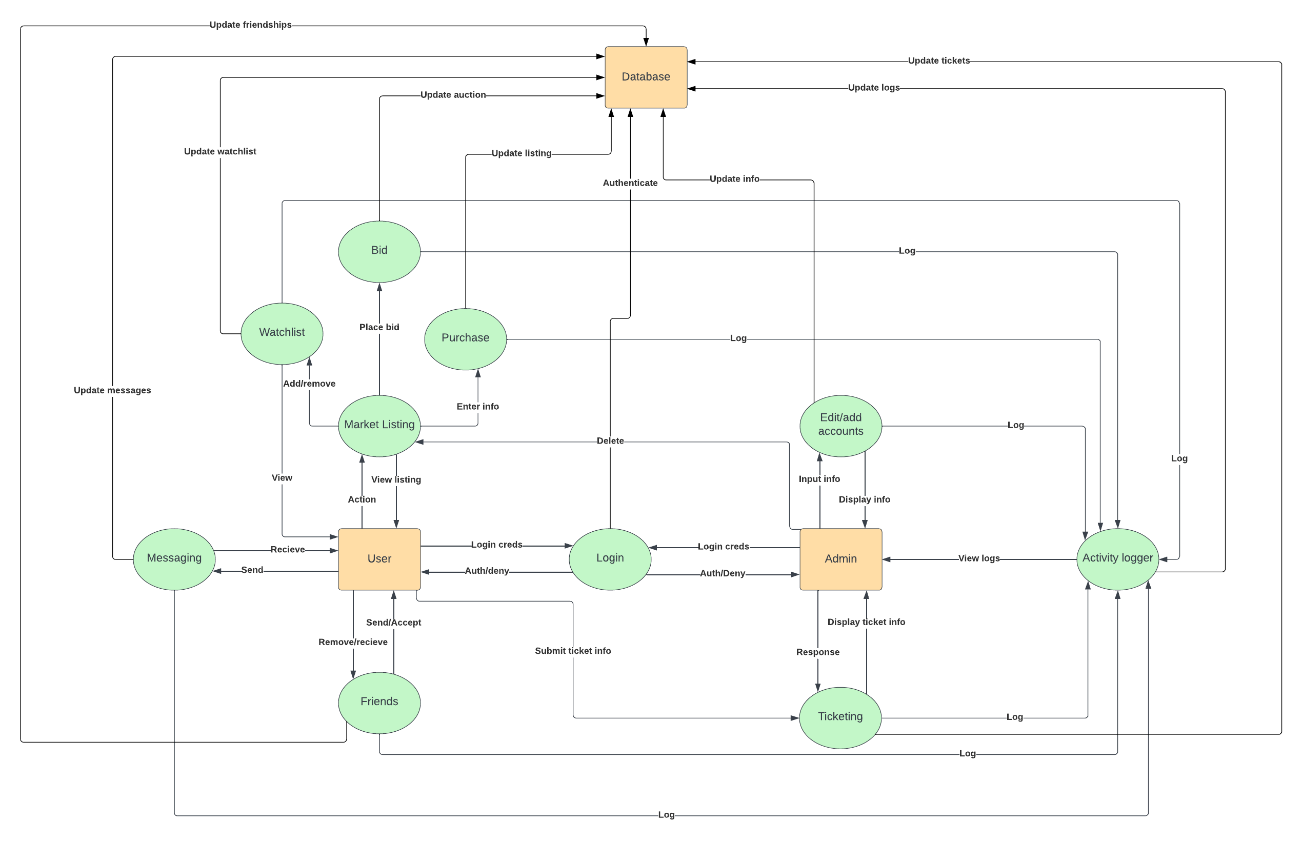


### Sequence Diagrams:

Sequence diagrams help illustrate the interactions and orders of the various components of our system working together. For this example, this sequence diagram helps illustrate how an actor can browse widgets (market listings) on our system.



# DFD Diagrams



# Entity/Relationship Diagrams

## User

A screenshot of a computer

Description automatically generated

## Bid

A screenshot of a computer

Description automatically generated

## MarketListing

A screenshot of a computer

Description automatically generated

## Widget

A screenshot of a diagram

Description automatically generated

## Social Messaging Websocket

A screenshot of a computer

Description automatically generated

# Caveats/Minefields

We ran into numerous issues along the way. Some of our largest issues stemmed from areas including our database, dependencies, and deploying to a Tomcat server. Brief outlines of the problems encountered during development are summarized here.

## Caveats with working solutions

These are areas that we were able to overcome after many hours of trial and error. The solutions could be improved in the future.

### E-Commerce Categories

**Problem:**

In the code that we received, the e-commerce categories for the application were “hardcoded” into database, html files, and domain/services. This limitation made it incredibly difficult to scale and add more categories to the platform, as there were so many locations that these categories existed. For example, “appliances” was an existing category with the two subcategories of “fridges” and “microwaves.” If we wanted to remove the “fridges” category, we would need to visit every single page and location of that code and remove it by hand.

**Solution:**

To solve this, we created a Dynamic Category tree data structure. This data structure allowed us to scale and add categories and maintain relationships between them. It also allowed us to have much deeper levels than just category and subcategory.

### Shipment Tracking & Address Validation

**Problem**:

Shipment tracking is limited to our knowledge from the carrier. Many carriers offer their own API for package tracking, but our application supports multiple carriers. So, we need some all-in-one solution to track packages, no matter the carrier. Additionally, we needed a solution for address validation, since our Smarty Streets API was no longer functional.

**Solution:**

We found a free API called from Shippo which allowed us to submit shipping labels and receive information about the status of that parcel. The Shippo API also had support for validating addresses. We used this for shipping addresses, local pickup addresses, and billing addresses.

### Tomcat Deployment

**Problem:**

As we made several attempts to deploy to Tomcat on the Robotics server, there were several instances where we had to either update to another version of Spring Boot or JDK. The compatibility issues between outdated versions of the tools that we were using made it particularly challenging to deploy to the live server.

**Solution:**

Our solution was to convince IT to upgrade the Robotics lab with JDK 17. This was necessary because Tomcat 10 required us to upgrade to Spring Boot 3.1 (From 2.6). Spring Boot 3.1 required us to use JDK 17.

### User Sessions

**Problem:**

In many instances of the application, the controllers host global attributes with information about the current user using that controller. This solution prevented us in many ways from successfully deploying to a live server with several users using the application at once.

**Solution:**

In order to solve this problem, we used the “Principal” request parameter in order to identify the user making a request to our controllers. In turn, we were able to identify specific users and their sessions, rather than storing temporary information globally.

### Websockets

**Problem:**

Websockets were an especially difficult issue to overcome. This arose in the social messaging page. A websocket was used to subscribe to incoming messages and send outgoing messages to friends. This websocket was difficult to setup and maintain on deployment to Tomcat.

**Solution:**

To help with websockets, we used the STOMP protocol along with stomp.js. This allowed the messaging “topic” for that user’s session to be the driving component for displaying and sending messages.

### Context Paths

**Problem:**

When deploying to Tomcat, many of URLs, redirects, and resources became broken. This was because the context path had changed. Locally the context path was “localhost:8080/” but in Tomcat, it became “192.xxx.xxx.xxx:8080/OnlineAuction”.

**Solution:**

To fix this, we used the ServletContext bean to fetch the relative context path for whatever environment the application was running in. We also needed to manually go through and change all URLs that used absolute pathing.

## Minefields

These are areas where no solution was able to be created.

* **SLL encryption**
  + Limited by IATS configuration
* **Database initialization for Tomcat deployment**
  + We were not able to create a solution for initializing the database automatically when deploying a war file to Tomcat. Instead, we either entered the data manually through the UI, or imported data using a .sql file.
* **Paypal**
  + We were able to provide Paypal plugins, but we later deprecated this due to issues testing and lack of support.
* **Docker Database**
  + We made it as far as creating a docker image for the SpringBoot application, but supporting and initializing the MySQL database became very challenging.

# Documentation

The online auction system’s required documentation can be found within the Documents folder.

The following documentation are included with the project:

* Technical manual
* Security Manual
* Installation manual
* User manual
* Evaluation manual
* Login manual
* Tomcat deployment manual
* Hardware/software requirements

Additionally, the complete collection of relevant UML diagrams can be found within the UML Diagrams folder in the same directory.

In our project, we've made a conscious effort to keep things clear and accessible for future developers. For most of the new functions we've introduced, we've added comments right into the code, explaining not just what each function does but also why we chose a specific approach. This helps new developers quickly grasp the purpose and context behind our implementations. Additionally, we've taken care to document our error-handling processes, detailing possible error scenarios and how they're handled within the code. By doing so, we aim to provide a roadmap for anyone who might work on the project in the future, ensuring that they can understand not only the functionality but also the reasoning behind our design decisions and how we handle potential issues.

# File Path Names

**Application Configuration File (application.properties):**

* Purpose: Central location for application-level configurations, including database connections, server settings, and other Spring Boot specific configurations.

**CI/CD Configuration Files (GitHub Actions for WAR Generation):**

* Purpose: Define workflows for GitHub Actions, especially for building and deploying the application as a WAR (Web Application Archive) file, automating the continuous integration and delivery processes.

**Configuration Directory (.configuration):**

* Purpose: Contains classes and resources related to the application's configuration, such as database configurations, external service connections, and application-specific settings.

**Controller Directory (.controller):**

* Purpose: Holds MVC controllers responsible for handling incoming HTTP requests, preparing a model, and returning the view.

**Database Initialization File (import.sql):**

* Purpose: Used for initializing the database with necessary data upon startup, typically containing SQL statements for data insertion.

**Data File (StateTaxes.xlsx):**

* Purpose: Likely contains specific data related to your application, such as state tax information, used for data import, analysis, or as a reference.

**Domain Directory (.domain):**

* Purpose: Used for domain models, containing the application's core business logic, including entities, enums, and data transfer objects (DTOs).

**Project Object Model File (pom.xml):**

* Purpose: A key file in Maven-based projects, defining the project's structure, dependencies, build configurations, and other details.

**Repository Directory (.repository):**

* Purpose: Focuses on data persistence, containing interfaces for data access layers, usually implementing Spring Data JPA for database interactions.

**Security Directory (.secure):**

* Purpose: Dedicated to security configurations of the application, including authentication, authorization, and other security aspects.

**Service Directory (.service):**

* Purpose: Houses the service layer, encapsulating business logic, performing operations on domain objects, and interacting with repositories.

**Utility Directory (.util):**

* Purpose: Contains utility or helper classes that provide common functions used across the application, like date/time utilities and string manipulation tools.

## OO Class Design of File Paths

public class ApplicationFilePaths {  
  
 private String applicationConfig = "application.properties"; // Application Configuration File  
 private String ciCdConfig = "GitHub Actions Files"; // CI/CD Configuration Files  
 private String configurationPath = ".configuration"; // Configuration Directory  
 private String controllerPath = ".controller"; // Controller Directory  
 private String dbInitScript = "import.sql"; // Database Initialization File   
 private String dataFile = "StateTaxes.xlsx"; // Data File   
 private String domainPath = ".domain"; // Domain Directory   
 private String projectModel = "pom.xml"; // Project Object Model File   
 private String repositoryPath = ".repository"; // Repository Directory   
 private String securityPath = ".secure"; // Security Directory   
 private String servicePath = ".service"; // Service Directory   
 private String utilityPath = ".util"; // Utility Directory

}

# Code Reusability

Building a scalable platform was one of the goals the team strived for during development. Because of this, code reusability was a top priority. Object oriented programming was used as it easily allows developers to reuse code as many times as necessary. The MVC pattern implemented in the project also allows code to be reused very easily. Nearly all of the Java classes within Offerly can be reused.

While developing the online auction platform, the team utilized Scrum development where requirements are frequently changing. Since the team was aware of this, any of the new methods implemented into the project have been designed in a way that makes them reusable in the event that they are needed to help with another feature. Most of the older, inherited methods and classes have also been refactored in order to boost their reusability.

For example, the watchlist feature allows a user to add a product listing to their watchlist for future viewing. When developing this feature, a lot of new methods and classes needed to be created. Rather than building a one-time use class that did not offer much use outside of that specific example, the classes and methods were broken down in a way that would allow them to be used later with easy implementation. This proved helpful when it was decided that when a user places a bid on a product, it will be added to their watchlist if it was not previously. Because these methods were developed with reusability in mind, implementation of this new feature was quick and hassle-free.

These examples of reusability can be seen throughout the project, as many of the more complex features still rely on relatively simple methods and procedures. Bidding is another excellent example of this. While manual and automatic bidding behave differently, they still rely on many of the same methods.

Code reusability has been a key component in allowing our team to develop new features fast as we do not waste unnecessary resources developing the same functionalities more than once.

# Testing

## Testing Plan

**Objectives and Scope:** This testing plan aims to ensure the functionality, performance, and security of our Spring Boot application. The scope covers unit testing of individual components, integration testing of combined components, and user acceptance testing to validate overall application performance and usability.

**Testing Strategy**: Our strategy involves a phased approach: beginning with unit testing using JUnit and Mockito, followed by integration testing to evaluate component interactions, and concluding with user acceptance testing for end-to-end application evaluation.

**Tools and Technologies:** We utilize JUnit for unit/integration testing and Mockito for mocking dependencies. UAT is conducted in a simulated production environment in our Robotics lab. We also utilized GitHub Issues to keep track of bugs that were revealed during the UAT process.

## Unit Testing

**Test Environment Setup:** The unit testing environment was configured with the Eclipse JUnit plugin. Mock objects were created using Mockito to isolate the testing of individual components.

**Test Cases and Coverage:** We implemented unit test cases covering key functionalities such as setters and getters for our domain entities. These tests include both positive and negative scenarios to ensure comprehensive coverage. Unit tests were created for all domain classes, form classes, and DTO objects.

## Integration Testing

**Integration Strategy**: Our integration testing focused on the interaction between controllers, services, and entities. The tests were designed to simulate real-world usage scenarios.

**Test Scenarios:** Key scenarios included listing items for sale, logging in, and purchasing items. Our goal was to ensure that component interactions were robust and error-free.

**Challenges and Resolutions:** We encountered challenges related to user authorization for requests which were resolved through mocking the user role and principal.

## Automated Tests Results

We had a total of 317 tests. 259/317 tests ran successfully, with 38 errors and 5 failures.

Passing percentage: 81.7%

A screenshot of a computer

Description automatically generated

## User Acceptance Testing

User Acceptance Testing was conducted on the live server in the Robotics Lab. The test was a check of the following components using the UI:

1. Create user account
2. Add payment details
3. Add shipping details
4. Create Admin/IT accounts
5. Create Listing
6. Purchase Listing
7. Bid/Autobid on Listing
8. Track Listing
9. Social Messaging
10. Update user details
11. Create/update Ticket
12. Test multiple user sessions
13. Admin functionalities (delete users, update listings, view logs)

While conducting tests, bugs/items to addressed were marked in a GitHub issue:

A screenshot of a computer

Description automatically generated

# Logger

## User Action Logging

Logging user actions was achieved using our statistics class. These logged actions are saved to both a text file and to our database. Admins can then view and filter logged actions through the admin portal:

A screenshot of a computer

Description automatically generated

## Logging in Deployed Environment

For system logging, we mostly used system prints to the console. When running in a Tomcat Environment we could then view these logs in the /logs directory. This could be improved in the future by using a logging package such as log4j.

The following logs are how we interpreted our system’s activity (x’s represent dates):

* Tomcat logs – **catalina.xxxx-xx-xx.log**
* Requested resource logs – **localhost\_access\_log.xxxx-xx-xx.txt**
* Spring Boot Application logs – **tomcat10-stdout.xxxx-xx-xx.log**

# Deployment/Maintenance

## Overview:

This section of the report outlines the deployment and maintenance process for our Spring Boot application. Our deployment strategy leverages GitHub Actions for automated build processes, generating a WAR file which is then manually deployed to a Tomcat server. This approach ensures consistent and repeatable builds, while also allowing for careful control and monitoring during the deployment phase.

## Deployment Process:

### Automated Build with GitHub Actions:

1. Configuration: GitHub Actions is configured to trigger a build process on every push to the main branch or when a pull request is merged. The .github/workflows directory contains the workflow configuration file defining the build steps.
2. Build Execution: On trigger (merge to master), GitHub Actions executes the predefined workflow. It includes steps for code checkout, dependency installation, running tests, and compiling the code to generate the WAR file.

A screenshot of a computer

Description automatically generated

1. Artifact Generation: The resulting WAR file is an artifact of this process, which is then available for download from the GitHub Actions run.

A screenshot of a computer

Description automatically generated

### Manual Deployment to Tomcat Server:

1. Pre-Deployment Checklist: Prior to deployment, a pre-deployment checklist is completed to ensure the Tomcat server is ready. This includes verifying server availability, checking network configurations, and ensuring backup of current deployments.
2. Deployment: The WAR file is manually uploaded to the Tomcat server. This involves accessing the server, typically via SSH or a direct server interface, navigating to the webapps directory, and replacing the existing WAR file with the new one.
3. Post-Deployment Verification: After deployment, the application is thoroughly tested in the production environment to verify that all features are working as expected. This includes checking system logs, monitoring application performance, and conducting sanity checks of critical functionalities.

## Maintenance:

### Monitoring and Logging:

* Continuous monitoring is set up to track the application's performance and health on the Tomcat server. This includes monitoring CPU usage, memory consumption, and response times.
* Logging is configured for the application, providing detailed insights into runtime operations, errors, and warnings. These logs are periodically reviewed to identify and address potential issues.

### Updates and Patches:

* Regular updates to the application are managed through the same GitHub Actions pipeline. This ensures that any changes, including bug fixes and feature enhancements, undergo the same rigorous build and deployment process.
* Patches and urgent fixes are expedited through a defined hotfix process, which includes rapid deployment steps while ensuring minimal disruption to the live environment.

### Backup and Recovery:

* Regular backups are possible with our Source Control Management process in GitHub.
* We can easily recover and rollback to a previous iteration of our application.

# Future Works

## Problems:

Below are the components of the program that are as of now present, however, are not fully operational as of the time of submission.

* Make bidding more authorized. Users can bid on listings, even if they do not have payment information set up. During the process of creating a new account user have the option to skip payment information.
* Resetting password. Trying to reset the password generates a HTTP 400 error, but the process of validating security questions works.
* Viewing a Ticket ID on the live build is not functioning. On the local build, users can view their tickets to see the conversation and details about their ticket.

## Cleanup:

Below are potential fixes to resolve the problems in the previous section.

* When a user goes to bid on a listing, generate a pop up that makes them fill in payment information if they do not have any.
* Fix the /resetPassword method and check if everything is connected properly between the front-end and the backend.
* Similar to messaging on the live build, try to configure the Tomcat webapps functions for viewing Ticket ID’s to make sure there is a seamless connection. Using, “Inspect” on the browser and looking at the “Network” tab may help with finding a solution.

## Improvement/ Expansion:

Below are potential future implementations that could be added to the software to improve ease of use and software functionality.

* Fixing the UI for messaging. Our team’s main concern was building up the foundation for this future. By making it more modern it could improve user experience.
* Configuring SSL on the live build. With this in place, logins and transactions can be more secure.
* Rework the functionality for how a user wins an auction. As of right now, when an auction ends, the highest bidder wins. The highest bidder can then click a button for the listing they won, after the auction ends, to purchase the item. Instead of this, maybe have it so when an auction ends, it notifies the highest bidder and just charges them. Having a button is not convenient as the user may forget
* When an auction ends, it still populates the page. To better user experience, set a timer on a listing so it deletes an auction that has ended (maybe a couple hours/days after).
* Filter Options on the Browse page. (Filter for Category, Price, Quantity, Location). This could help users find items.
* Filter options for the admin/ employee tickets page. This can help filter tickets to show one’s with higher priorities versus the time the ticket was sent.

# Port-Mortem Analysis

Developing this online auction platform introduced our team to plenty of new technology, many of which we had little to no experience with previously. Despite this, our team was able to deliver the product with nearly all of the functionality and features that had come about during the project’s lifetime.

Although most of the requirements of the project were delivered, the group undoubtedly encountered difficulties that could have been avoided. Some of these problems were minuscule while others were quite troublesome. In hindsight, we can effectively identify which problems could have been avoided and what we could have done differently. For example, towards the end of the project’s lifetime, we learned about design patterns and how they can be implemented to solve problems using an object-oriented design. Looking back at our project, there are several key parts in our code, like messaging, that would greatly benefit if we incorporated design patterns. As work on the project progressed, we began to implement design patterns when we could, although implementing them from the start would likely make the project more scalable.

Another key area of improvement is testing from the beginning. The handed-off code came with numerous tests, with most of them being broken as soon as we dug into the code. Additionally, our environment was so fast paced that it made it difficult to maintain these tests and create new ones for our new features. We believe that if we could have slowed down the number of new features that we were getting, and use the time saved to focus on continuous testing, we would be in a much better position. The pressure for testing eventually became a very deep hole that we struggled with in the end.