Online Auctioning

Final Report

**Slippery Rock University**

Spring 2024

Jacob Johnston – jrj1014@sru.edu

Tiffany Morrow – tbm1005@sru.edu

Nicholas Vioral – ncv1005@sru.edu

Wolfgang Zoppelt – wsz1001@sru.edu

**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**

This table represents the tasks to be completed and the status:

|  |  |  |
| --- | --- | --- |
| **Task:** | **Status:** | **Description:** |
| Discussion Board | In Progress | Added new Discussion Board feature complete with titles, keywords, posts, search, and Admin supervision |
| Contact Forms | Complete | Added in Contact Forms complete with contact information. The posts go to new employee type Customer Service as tickets |
| Group Messaging | Complete | Added in new feature to add friends to a group to converse between 2 or more people |
| Improved Messaging | In Progress | Found a way to get all messages to inbox, not just between friends. Still needs work |
| View Count on Listings | Complete | Listings now show how many users have viewed a listing |
| Improved Create and Update Listings | Complete | More categories added to create a listing, can now update most of them once created. |
| Improved the way Employee users are created | Complete | Created Super Admin User that can edit roles of employees and create new user accounts |
| Switched Bids to Offers | In Progress | Instead of Bids added in feature to make offers on listings, the seller can accept/reject offers, only accepted offer may buy at new sell price |
| Improved Data Base and SQL Tables | Complete | SQL data saves properly, Private information no longer displayed in plain text, encryption used in place |

**Contributions**

Although the majority of the code was crafted and authored by the members of group 6, several external sources were consulted as references to aid in accomplishing the aforementioned tasks. Some of the leading contributions were gathered from w3schools, baeldung, thymeleaf, stackoverflow and geeksforgeeks.

Thymeleaf was difficult to navigate at times and required the use of outside sources.

**External Sources**

1. <https://springhow.com/thymeleaf-form-handling/>
2. <https://www.baeldung.com/spring-web-flash-attributes>
3. <https://www.thymeleaf.org/doc/articles/springmail.html>
4. <https://stackoverflow.com/questions/54291420/retreive-blob-from-database-with-image-using-spring-mvc-and-thymeleaf>
5. <https://stackoverflow.com/questions/53102032/thymeleaf-counter-variable-not-incrementing>
6. [https://www.base64-image.de/tutorial#:~:text=You%20can%20use%20the%20base64,be%20copied%20to%20your%20clipboard](https://www.base64-image.de/tutorial)
7. <https://stackoverflow.com/questions/5650457/html-select-form-with-option-to-enter-custom-value>
8. [https://select2.org](https://select2.org/)
9. <https://www.geeksforgeeks.org/how-to-prevent-buttons-from-submitting-forms-in-html/>
10. <https://stackoverflow.com/questions/45480307/bootstrap-modal-how-to-return-true-if-yes-button-is-clicked-customized-confi>
11. <https://www.digitalocean.com/community/tutorials/java-download-file-url>
12. [https://www.kindsonthegenius.com/how-to-implement-pagination-in-spring-boot-with-thymeleaf/#t2](https://www.kindsonthegenius.com/how-to-implement-pagination-in-spring-boot-with-thymeleaf/)
13. <https://www.codehim.com/forms/html-code-for-comment-box-with-reply/>
14. <https://www.codehim.com/html5-css3/html-code-for-discussion-forum/>
15. <https://www.thymeleaf.org/doc/tutorials/3.1/thymeleafspring.html>
16. [https://getbootstrap.com/docs/5.3/components/scrollspy/#via-data-attributes](https://getbootstrap.com/docs/5.3/components/scrollspy/)
17. [https://css-tricks.com/practical-css-scroll-snapping/#:~:text=To%20make%20a%20vertical%20list%20snap%20to%20each,be%20a%20snap%20point%3A.child%20%7B%20scroll-snap-align%3A%20start%3B%20%7D](https://css-tricks.com/practical-css-scroll-snapping/)
18. chat.openai.com

**Glossary**

API, or Application Programming Interface, serves as a comprehensive set of guidelines, protocols, and tools pivotal for constructing software and applications.

A Build Tool automates the conversion of source code into executable applications (such as Maven or Gradle).

Continuous Deployment (CD) is a software release approach that leverages automated testing to verify the correctness and stability of code changes for immediate deployment to a production environment.

Continuous Integration (CI) embodies the practice of frequently merging all developers' working copies into a shared mainline.

Database Initialization entails configuring a database schema and populating it with initial data, crucial for establishing an environment conducive to database operations.

Dependency Management involves handling external libraries necessary for the proper functioning of your project.

Docker comprises a suite of platform-as-a-service products utilizing OS-level virtualization to deliver software through containers.

A Framework provides a foundational platform for developing software applications tailored to a specific platform.

Git is a distributed version-control system that tracks changes in source code during software development.

GitHub Actions with WAR File Artifact Generation automates workflows for building, testing, and deploying projects. In the context of Java web applications, it involves automating the generation of a WAR file, deployable to servlet containers like Tomcat.

Integration Testing involves testing combined units to expose interaction faults, aided by test drivers and stubs.

Java Bean is a reusable software component conforming to specific conventions, encapsulating multiple objects into a single object and adhering to naming conventions, widely used in Java programming, particularly in Java Enterprise Edition and frameworks like Spring.

JDK (Java Development Kit) provides a development environment for Java applications, comprising tools such as the Java Runtime Environment, compiler, archiver, and documentation generator.

JUnit is a unit testing framework for Java.

Live Server Deployment on Tomcat involves deploying a web application to a Tomcat server for access in a production environment.

Logger is a utility for logging information during program execution, crucial for debugging and monitoring operational states.

Maven is a project management and comprehension tool, offering developers a comprehensive build lifecycle framework.

Microservices architecture structures applications as a collection of loosely coupled, independently deployable services organized around business capabilities.

Mockito is a Java-based mocking framework facilitating test-driven and behavior-driven development by creating and configuring mock objects.

REST API establishes rules for constructing web services allowing interaction with RESTful web services.

Spring Boot simplifies the creation of stand-alone, production-grade Spring applications.

Spring Security is an authentication and access-control framework highly customizable for Spring applications.

UAT Testing (User Acceptance Testing) is the final phase of software testing, where end-users ensure the software meets specifications in real-world scenarios.

Unit Testing tests individual software components separately to verify correct functionality, typically automated and integral to test-driven development.

WebSocket is a communication protocol enabling full-duplex communication over a single TCP connection, often used for real-time data transfer between client and server.

**Problem Resolution**

This semester's objective was to revamp Offerly and transition it into a social marketplace application. We introduced functionalities enabling customers to engage and communicate while buying and selling goods. Additionally, we enhanced the process of creating, viewing, updating, and selling listings. Moreover, significant expansions and overhauls were made to the website's structure.

Our solution was to add several avenues for clients to be able to communicate and interact with one another. We updated most of the social aspects that already existed while adding in new features such as discussion posts and group messaging. This provides a way for users to socialize with everyone, not just friends. It also is a good way to promote business and sales in the commerce facet.

We also modernized the way that listings are created, updated and interacting with to boost the experience and satisfaction of consumers and merchandisers. Instead of employing a hands-off bidding approach, we decided to implement a fresh method of offer requests. Customers now have the option to send an offer amount directly to merchants for the product, and merchants can choose to accept or reject these offers. This makes the experience more interactive and simpler.

A method of being able to message merchants has been implemented as well, so that customers can ask vendors about products, quantity and other aspects of commodities.

The discussion posts feature facilitates users in engaging with various market-related topics. Here, users can share their questions, concerns, inquiries, and even advertise products, thereby fostering a vibrant community atmosphere. This interactive platform not only promotes the site itself but also enhances visibility for products listed on Offerly, creating a dynamic space for exchange and promotion within the marketplace.

**Scope:**

Products:

The online auction platform will be a publicly accessible web-hosted application catering to ecommerce activities for its user base.

Technology Stack:

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

Recommended Features:

* User registration and authentication
* Listing products for sale
* Purchase through offered prices and bidding
* Watchlist for products
* Payment processing
* Product search, filtering, and categorization
* Administrative functionalities and permissions

Terms:

* User: Standard user account with regular selling and purchasing permissions.
* Admin: Administrator account overseeing most system operations.
* Super admin: Highest-level administrator overseeing admins and monitoring system revenue.
* Offer price purchasing: User purchases an item at its listed price.
* Bidding: User competes with others by placing bids on items.
* Auto bid: User specifies a maximum bid value for automatic bidding.

**System Requirements**

Specifications:

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

**UML Diagrams**

The project's documentation folder contains a comprehensive set of pertinent UML diagrams. Within this document, you'll find examples of UML diagrams illustrating some of the platform's key features.

**Class Diagrams:**



The FriendsController class is designed to manage social interactions within a Java web application, handling functionalities related to friendships, group activities, and messaging. It operates with various services and repositories to manage friends, respond to friend requests, create and manage groups, archive or unarchive groups, and facilitate user searches and conversations, both on an individual and group level.

**Use Case Diagram**:



The above image shows a Use Case Diagram displaying all the different options/actions available to a User and Seller on a Listing Page. The Buyer can send offers, buy the product at asking price, view/watchlist an item and send a message to the Seller. The Seller can list an item, view offers, and respond to messages. The Customer Service Representitive can Notify users and offer help on listing/product information.

**Sequence Diagrams:**



This sequence diagram shows the flow of data required to load the pages of the "Browse All Items" page. The user enters the page with a certain page number, 1 being the default if they are loading it for the first time. This calls the findPage function with that page number which goes to the MarketListingService. The service then finds a page of listings through the MarketListingRepository. After finding all of the listings on the page it then searches the WidgetImageRepository with those listings to find all of the necessary listing images. Once all of the necessary data is found, the data is sent to the view to be displayed on the browseWidgets.html page.

**StateChart Diagrams:**



This StateChart diagram outlines how the user interacts with the discussion board system. Before navigating to the discussion board page, the user is required to login. Once on the page they have two primary actions that can be performed. These are either make a new post or comment on an existing post, after making their own post they have the option to archive it. Archiving a post makes the post still available to view but no longer be commented on.

**Caveats/Minefields**

Throughout our journey, we encountered a multitude of challenges. Among the most significant were issues pertaining to our database, dependencies, and the deployment process onto a Tomcat server. Here, we offer concise overviews of the problems encountered throughout the development phase.

Working with Thymeleaf, while powerful for server-side Java web applications, can present its fair share of challenges. One of the primary difficulties lies in its learning curve, especially for developers accustomed to more traditional templating engines or frontend frameworks. Thymeleaf's syntax, though expressive, can initially be complex to grasp, leading to potential errors and inefficiencies during development. Additionally, integrating Thymeleaf with complex frontend technologies like JavaScript frameworks may require extra effort, as managing the interplay between server-side rendering and client-side interactions demands careful coordination. Moreover, debugging Thymeleaf templates can be time-consuming, particularly when dealing with dynamic content or complex template structures. Despite these hurdles, mastering Thymeleaf unlocks a robust tool for building dynamic, data-driven web applications in Java environments.

An issue we ran into while working with the database side of things was ensuring that default listings, users, and other important information was being inserted to the database upon launch of the program. A major problem we ran into was when we would make updates to our classes, the tables for said classes would change in the database. If this change was not reflected in the database import file, it would not allow for correct imports.

Another issue that we ran into was the deployment process onto a Tomcat server. The main problem we ran into was getting the project and the server to run on the Secure Socket Layer. After many hours of research, we were able to overcome this hurdle and get the application to deploy with these parameters. If our Tomcat Deployment Manual is followed, these issues should not occur.

**Documentation**

The necessary documentation for the online auction system is available within the Documents folder.

Included documents are:

* Technical Manual
* Security Manual
* Installation Manual
* User Manual
* Evaluation Manual
* Login Manual
* Tomcat Deployment Manual
* Hardware/Software Requirements

Furthermore, all relevant UML diagrams are located in the UML Diagrams folder within the same directory.

In our project, we have prioritized clarity and accessibility for future developers. For the new functions we have introduced, we have incorporated comments directly into the code. These comments not only explain the purpose of each function but also provide insights into why we chose specific approaches. This approach aids new developers in swiftly understanding the functionality and context of our implementations.

Additionally, we have meticulously documented our error-handling processes, outlining potential error scenarios and their corresponding resolutions within the code. Our aim is to furnish a clear roadmap for anyone undertaking future work on the project. This documentation ensures that future developers can comprehend not only the functionality but also the rationale behind our design decisions and our approach to addressing potential issues.

**File Path Names**

* Application Configuration File (application.properties):

Purpose: Serves as a central repository for application-level configurations, encompassing 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, specifically tailored for building and deploying the application as a WAR (Web Application Archive) file, automating the continuous integration and delivery processes.

* Configuration Directory (.configuration):

Purpose: Houses classes and resources pertinent to the application's configuration, such as database configurations, connections to external services, and application-specific settings.

* Controller Directory (.controller):

Purpose: Manages MVC controllers responsible for handling incoming HTTP requests, preparing model data, and rendering views.

* Database Initialization File (import.sql):

Purpose: Facilitates the initialization of the database with necessary data upon startup, typically containing SQL statements for data insertion.

* Data File (StateTaxes.xlsx):

Purpose: Likely contains specific data relevant to the application, such as state tax information, utilized for data import, analysis, or reference.

* Domain Directory (.domain):

Purpose: Hosts domain models embodying the application's core business logic, including entities, enums, and data transfer objects (DTOs).

* Project Object Model File (pom.xml):

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

* Repository Directory (.repository):

Purpose: Focuses on data persistence, housing interfaces for data access layers, typically implementing Spring Data JPA for database interactions.

* Security Directory (.secure):

Purpose: Devoted to the security configurations of the application, encompassing authentication, authorization, and other security aspects.

* Service Directory (.service):

Purpose: Contains the service layer, encapsulating business logic, executing operations on domain objects, and interfacing with repositories.

* Utility Directory (.util):

Purpose: Stores utility or helper classes furnishing common functionalities utilized across the application, such as date/time utilities and string manipulation tools.

**Code Reusability**

During development, the team pursued the goal of constructing a scalable platform. As a result, prioritizing code reusability was paramount. Object-oriented programming was employed due to its innate ability to facilitate code reuse extensively. Additionally, the adoption of the MVC pattern in the project significantly enhances code reusability, with almost all Java classes within Offerly designed for reuse.

In implementing the online auction platform, the team embraced Scrum development, acknowledging the frequent evolution of requirements. To accommodate this dynamic environment, newly introduced methods were meticulously crafted with reusability in mind, ensuring their adaptability to support future features. Moreover, efforts were dedicated to refactoring older, inherited methods and classes to augment their reusability potential.

One instance of this scenario is the enum status within the Market Listing Status. These status settings have the flexibility to serve various purposes. Similarly, the discussion board layout has potential beyond merely posting topics. Additionally, there could be coding solutions addressing challenges related to Thymeleaf.

Throughout the project, instances of reusability are evident, as numerous complex features are built upon relatively straightforward methods and procedures. The emphasis on code reusability has played a crucial role in enabling our team to swiftly develop new features, avoiding the duplication of effort and unnecessary resource expenditure on redundant functionalities.

**Testing**

Objectives and Scope: The purpose of this testing plan is to guarantee the functionality, performance, and security of our Spring Boot application. It encompasses unit testing of separate components, integration testing of interconnected components, and user acceptance testing to confirm the overall performance and usability of the application.

Testing Strategy: Our strategy adopts a phased methodology. We commence with unit testing utilizing JUnit and Mockito, proceed to integration testing to assess component interactions, and culminate with user acceptance testing for comprehensive evaluation of the application's end-to-end functionality.

**Unit Testing:**

Setting Up the Test Environment: We configured the unit testing environment using the Eclipse JUnit plugin and employed Mockito to generate mock objects, facilitating the isolation of individual component testing.

Test Cases and Coverage: Our unit testing strategy encompassed the creation of test cases for essential functionalities, such as setters and getters for domain entities. These tests spanned both positive and negative scenarios to achieve thorough coverage, extending to domain classes, form classes, and DTO objects.

**Integration Testing:**

Integration Approach: Our integration testing methodology emphasized examining the interaction between controllers, services, and entities, aiming to replicate real-world usage scenarios.

Test Scenarios: Key test scenarios comprised listing items for sale, logging in, and purchasing items, with a primary objective of ensuring robust and error-free component interactions.

**Deployment/Maintenance**

**Overview:**

This portion of the report provides an overview of the deployment and upkeep procedures for our Spring Boot application. Our deployment strategy makes use of GitHub Actions for automating the build processes, resulting in the creation of a WAR file that is subsequently manually deployed onto a Tomcat server. This method ensures consistent and replicable builds, while also permitting meticulous control and monitoring during the deployment phase.

**Deployment Procedure:**

Automated Building via GitHub Actions:

* Setup: GitHub Actions is set up to initiate a build process upon every push to the main branch or when a pull request is merged. The .github/workflows directory houses the workflow configuration file outlining the build steps.
* Execution of Build: Once triggered (via merge to master), GitHub Actions proceeds with the predefined workflow. This entails actions such as code checkout, dependency installation, test execution, and code compilation to generate the WAR file.
* Artifact Creation: The resulting WAR file serves as an outcome of this procedure, subsequently accessible for download via the GitHub Actions run.

Manual Deployment to Tomcat Server:

* Pre-Deployment Preparation: Before deploying, a pre-deployment checklist ensures the readiness of the Tomcat server. This includes confirming server availability, reviewing network configurations, and ensuring the backup of current deployments.
* Deployment Process: The WAR file undergoes manual upload to the Tomcat server. This entails 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.
* Post-Deployment Validation: Following deployment, the application undergoes rigorous testing in the production environment to confirm all features are functioning as intended. This involves examining system logs, monitoring application performance, and conducting sanity checks on critical functionalities.

**Maintenance:**

Monitoring and Logging:

* Continuous monitoring is established to monitor the performance and health of the application on the Tomcat server. This encompasses tracking CPU usage, memory consumption, and response times.
* Logging is set up for the application, offering comprehensive insights into runtime operations, errors, and warnings. These logs are periodically reviewed to detect and resolve potential issues.

Updates and Patches:

* The application receives regular updates through the same GitHub Actions pipeline. This guarantees that any modifications, including bug fixes and feature enhancements, undergo the same meticulous build and deployment process.
* Patches and critical fixes are expedited through a defined hotfix process, involving swift deployment procedures while ensuring minimal disruption to the live environment.

Backup and Recovery:

* Our Source Control Management process in GitHub enables regular backups.
* Restoring and rolling back to a previous version of our application is easily achievable.

For additional information, refer to the Install, Technical, Github and Tomact manuals in the Documents folder.

**Future Works**

**Issues:**

* This is a list of features that need adjusted and attended to.
* Market Listing Statuses need to be implemented further, some of them are not currently in use
* Messaging works for all users now, not just friends, however the solution is only a temporary fix and needs to be further improved upon
* There is a method to mark messages as read, though it is never called.
* Offers work and are functional, though a timer needs to be implemented on pending listings and on offer notifications for sellers.
* Admin view of the discussion board needs to be completed; comments associated with a post should appear in a lower table when the button “show comments” is clicked.
* The discussion board has the option for the admin to delete posts that haven't had activity in a defined amount of time, ideally a timer to automatically archive and then delete after a certain amount of time would be a much-needed improvement.

**Cleanup:**

* This is a list of suggestions and possible solutions to the issues stated above.
* When Market listings are created, they automatically get set to an Active status, when an offer on the listing has been accepted, the status is set to a Pending status where no changes can be made. If a listing is sold, deleted, or flagged status the corresponding status should be set.
* A method to retrieve all messages should be implemented and conversations should start as they happen.
* Read Messages Method needs to be called once a conversation between users is opened.
* A timer can be set from the ‘offer accepted on date’ and ‘offer received on date’ and can be monitored/changed by Admin.
* The issues with comments loading in the admin view seems to be just a communication issue between thyme leaf and dynamically loading it JavaScript.
* The admin ability to delete all posts that have been inactive for a set amount of time could be improved with a timer that would automatically archives then deletes, therefore decreasing the amount of maintenance it would require.

**Expansion and Improvement:**

* This is a list of possible future implementations that can be added and enhanced.
* Method to clear conversations could be added.
* Accepting an Offer on a listing could be improved, as of now a user that’s offer was accepted must navigate back to the listing page to buy, instead a link could be sent with the notification.
* A user should not have to sign in to view individual listings, instead if a user clicks on an action/button, it should then prompt user to sign in.
* View counts were added and watchlists implemented, browsing items could be listed by popularity.
* The Discussion Board could have a section for tracking the user’s created posts and notifying the user when a new comment is added. As well as the ability to notify an admin/customer service representative if a post is inappropriate.

**Port-Mortem Analysis**

The journey of expanding an online auction platform introduced our team to an abundance of new technologies, many of which were previously uncharted territory for us. Despite this initial unfamiliarity, our collective effort resulted in delivering a product that encompassed nearly all the envisioned functionality and features we sought to achieve throughout the project's evolution.

While we successfully met most project requirements, it's undeniable that we encountered various challenges, ranging from minor inconveniences to significant hurdles. Looking back, we can now pinpoint areas where proactive measures could have averted certain issues and identify opportunities for improvement in our approach. Nevertheless, our team's collaborative spirit enabled us to navigate through these challenges, finding innovative solutions along the way.

It has been an exhilarating and profoundly rewarding experience to work collaboratively on this project as a cohesive group. From the initial stages of brainstorming ideas to the final execution, each step has been imbued with a sense of shared enthusiasm and dedication. The project, along with our individual development, has traversed a considerable distance, and observing its evolution and formation as we achieved each milestone was a deeply satisfying experience.