Online Auctioning

Technical Manual

**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

**SpringBoot Dynamic SQL Import**

In Spring Boot, initializing your database on startup is made easy through the utilization of the import.sql file. By incorporating an import.sql file into the src/main/resources directory, Spring Boot will automatically execute it during startup, provided Hibernate serves as the JPA implementation.

Steps to Implement Dynamic SQL Import:

* Create the import.sql File

Begin by placing the import.sql file within the src/main/resources directory of your Spring Boot application. This file is typically generated by exporting the current database using MySQL Workbench.

* Validate SQL Statements
* Address Foreign Key Constraints

When dealing with relational databases, particularly those with foreign key relationships, the sequence of insertion becomes crucial. To streamline the import process, you can temporarily deactivate foreign key checks, under certain conditions where data integrity is assured. This serves several purposes:

Bulk Insertion Order: Disabling checks enables flexibility in organizing insertions without concerns about reference integrity.

Performance Enhancement: Bulk insertions can occur more swiftly without the overhead of checking foreign key constraints.

Data Integrity Confirmation: If confidence in the data's integrity exists, disabling checks will not compromise integrity.

To toggle foreign key checks in MySQL:

SET foreign\_key\_checks = 0;

-- Insert your SQL statements here

SET foreign\_key\_checks = 1;

* Configuration

Spring Boot, coupled with Hibernate, will automatically identify and execute the import.sql file upon startup. However, it's essential to specify Hibernate properties for creation.

spring.jpa.hibernate.ddl-auto=create

Following these steps facilitates the importation of initial database requirements upon application startup.

**Spring Profiles**

The system is set up to employ Spring Profiles for seamless management of environment-specific configurations. By utilizing Spring Profiles, the project can effortlessly transition between various configurations depending on the environment it operates in, such as development (dev) or production (prod).

Choose between dev or prod profiles

spring.profiles.active=dev

**Configuration Setup:**

Default Profile Configuration

Inside the resources folder, the application.properties file is arranged to enable users to specify a default active Spring profile.

**Profile-Specific Configuration Files:**

Within the profiles directory, separate properties files are generated for each environment: application-dev.properties for development and application-prod.properties for production.

**Example of Profile Configuration:**

Below is a glimpse of the application-prod.properties file:

spring.datasource.data=classpath:prod-data.sql

spring.jpa.hibernate.ddl-auto=update

**Profile Activation and Usage:**

Activating Profiles:

The active profile is determined by the spring.profiles.active property set in the application.properties file. Assigning this property to dev or prod activates the corresponding profile and loads the configurations from either application-dev.properties or application-prod.properties.

Loading Configurations:

During system initialization, it reads the spring.profiles.active property to identify the active profile. It then fetches the configurations from the respective profile-specific properties file situated in the profiles folder.

Database Configuration:

In the provided example for production, the spring.datasource.data property is configured to reference a SQL script file (prod-data.sql) containing the data intended for loading into the database. Furthermore, the spring.jpa.hibernate.ddl-auto property is set to update, facilitating Hibernate to adjust the database schema as needed.

**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 Diagram:**



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.

**State Chart Diagram:**



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.

**Sequence Diagram:**



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.

**Summary**

Software Operational Requirements:

* Eclipse IDE for Enterprise Java and Web Developers – Version 2023-06.
* MySQL Server – Version 8.0 or higher.
* MySQL Workbench – Version 8.0 or higher.
* MySQL Shell - Version 8.0 or higher.
* Operating System compatible with the above software.

For detailed installation instructions and initial setup steps, consult the 'Install Manual' in the Program Documents.

After setup, launch the program by right-clicking the 'sellingwidgets' folder in Eclipse and selecting RunAs -> Spring Boot Application.

Upon program initialization, users are directed to the index page featuring various options such as 'login,' 'signup,' 'browse,' 'motto,' 'FAQ,' and 'contact us.'

Refer to the 'User Manual' for more information on how to navigate the project.