Online Auction

Technical Manual

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

Docker

Docker is a platform used to develop, ship, and run applications inside containers. A container is a lightweight, standalone, and executable software package that includes everything needed to run a piece of software, including the code, runtime, system tools, libraries, and settings. Containers are isolated from each other and the host system.

In the context of the Online Auction project, Docker is used to ensure that the application and its dependencies are packaged together and can be consistently deployed across various environments.

Prerequisites for Docker

1. Ensure you have Docker installed on your machine. Please refer to the install manual for instructions.
2. Ensure Docker Engine is running (start the Docker Desktop app).
3. Have the Project cloned to your machine from GitHub

Dockerfile Explanation

The `Dockerfile` is a script used by Docker to automate the building of container images. Here's a breakdown of the provided Dockerfile:

Base Image

```
FROM maven:3.8.4-openjdk-17
```

This line specifies that the base image for this container will be the official Maven image with Java 17. This means that the container will have both Maven and Java pre-installed.

Working Directory

```
WORKDIR /OnlineAuction
```

This sets the working directory inside the container to /OnlineAuction.

Copying Project Files

```
COPY pom.xml .
COPY src ./src
```

These lines copy the `pom.xml` (Maven project file) and the src directory (source code) into the container's working directory.

Build Project

```
RUN mvn clean install -DskipTests
```

This runs the Maven command to clean any previous builds, install the project dependencies, and compile the source code. Tests are skipped in this process.

Expose Port

```
EXPOSE 8080
```

This exposes port 8080 from the container, which is the default port for Spring Boot applications.

Run Command

```
EXPOSE 8080
```

This is the command that will be executed when the container starts. It runs the Spring Boot application.

Docker Compose Explanation

Docker Compose is a tool for defining and running multi-container Docker applications. The provided `docker-compose.yml` file defines the services, networks, and volumes for the Online Auction project.

Services

app

This service represents the main application. It depends on the `mysql` service, builds the image using the provided Dockerfile, maps port 6868 on the host to port 8080 on the container, and sets various environment variables for Spring Boot configuration.

The `volumes` directive is used to mount the Maven local repository from the host to the container, ensuring that dependencies are cached and reused between builds.

The `stdin_open` and `tty` options are set to true, allowing you to interact with the app service as if it were running in the foreground.

mysql

This service uses the official MySQL 8.0 image. It sets the root password and initializes a database named `online-auction`. The `import.sql` file from the project is copied to the container's initialization directory, ensuring that the database schema and initial data are loaded when the container starts.

The `volumes` directive is used to persist the MySQL data on the host machine, ensuring that data is not lost when the container is stopped or removed.

Volumes

```
volumes:
  online-auction-db:
```

This defines a named volume `online-auction-db` which is used to persist the MySQL database data.

Compose

To build and start the services defined in the `docker-compose.yml` file, use the following command:

```
docker-compose up --build
```

Once the services are up and running, you can access the Online Auction application by navigating to:

```
http://localhost:6868
```

To stop the running services, press CTRL+C in the terminal where docker-compose up is running. Alternatively, you can run the following command in another terminal:

```
docker-compose down
```

Dynamic SQL Import in SpringBoot

Spring Boot offers a simple way to initialize your database on startup through the import.sql file. By placing an import.sql file in the src/main/resources directory, Spring Boot will execute it on startup if Hibernate is used as the JPA implementation.

Steps to Use Dynamic SQL Import

1. Create import.sql

Firstly, we the place import.sql file inside the src/main/resources directory of our Spring Boot application. The import.sql is generated by exporting the current database using MySQL Workbench.

2. Confirm SQL Statements

3. Handle Foreign Key Constraints

When working with relational databases, especially those having foreign key relationships, the order of insertion matters. If you are certain about the integrity of the data being imported, you can temporarily disable foreign key checks to ease the import process. There are free reasons we do this:

- Bulk Insertion Order: By disabling checks, you are free to order your inserts in any way without worrying about reference integrity.
- Performance: Bulk insertions can be faster without the overhead of checking foreign key constraints.
- Data Integrity Assurance: If you're sure of the integrity of your data, i.e., the data being imported maintains referential integrity by itself, then disabling checks will not lead to integrity problems.

To disable/enable foreign key checks in MySQL:

```
SET foreign_key_checks = 0;

-- Your SQL INSERT statements go here

```

```
SET foreign_key_checks = 1;
```

4. Configuration

Spring Boot with Hibernate will pick up and execute import.sql on startup. However, we need to specify the property of Hibernate to `create`.

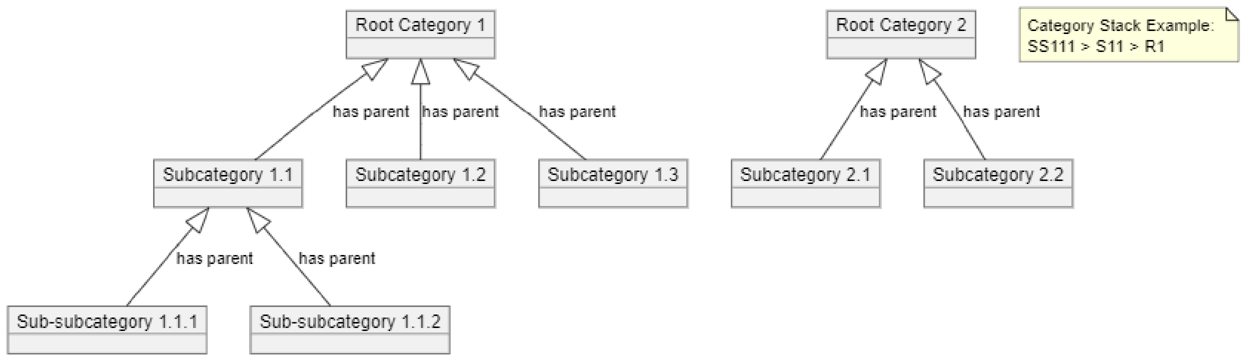
```
spring.jpa.hibernate.ddl-auto=create
```

Using these steps allows use to import the initial requirements of our database upon starting up the application.

Category Tree & Attribute Recommendation System

Category Tree

The Category Tree is a hierarchical representation of product or item classifications, where each category can have multiple subcategories. These categories help organize products or items in a structured manner, making it easier for users to navigate and locate specific items.



Tree Structure

- *Root Category*: The top-level category that forms the base of the tree. This category doesn't have a parent.

- *Subcategory*: Categories that fall under the root or another subcategory. Each subcategory has a parent category to which it is directly associated.

- *Further Subdivisions*: Subcategories can further be divided into more granular categories, creating multiple layers in the hierarchy.

- *Category Stack*: This represents the breadcrumb trail of categories for a given subcategory or item. For example, for an item in the "SS111" sub-subcategory, its category stack might be "SS111 > S11 > R1", indicating it falls under "SS111", which is under "S11", which in turn is under the root "R1".

Tree Features

- *Self-Join*: Categories have a parent-child relationship with themselves, allowing the creation of an n-level hierarchy.
- *Flexibility*: The tree can grow dynamically. As the inventory expands, more categories or subcategories can be added seamlessly.

Data

- For this application, basic ecommerce data was downloaded from Google to populate our database with categories.
- Link: https://support.google.com/merchants/answer/6324436?hl=en-GB

Attribute Recommendation System

In addition to the Category Tree, the platform also incorporates an Attribute Recommendation System. This system suggests attributes that are pertinent to specific categories, helping users define and refine their product descriptions.

Recommendation Structure

- *Attribute*: Represents a specific characteristic or property of an item. For instance, the attribute could be "Color" for clothing, or "Storage Capacity" for electronic devices.

- *Attribute Recommendation*: Associates an attribute with a category. Each association also carries a recommendation level, indicating its relevance or importance.

Recommendation Features

- *Relevance Score*: Each attribute associated with a category has a recommendation level. The higher the score, the more relevant the attribute is deemed to be for that category.

- *Dynamic Recommendations*: As products are added or removed, and as the market evolves, the recommendation levels for attributes can change, ensuring the system remains relevant and timely. Whenever a user uses a recommendation, the score associated with that attribute increases. Whenever they remove and attribute that was recommended, the score decreases.

- *Top Recommendations*: For any category, it's possible to retrieve the most recommended attributes, allowing users to quickly see the most relevant characteristics for products within that category. These recommendations can also be fetched from higher up the Category Stack.

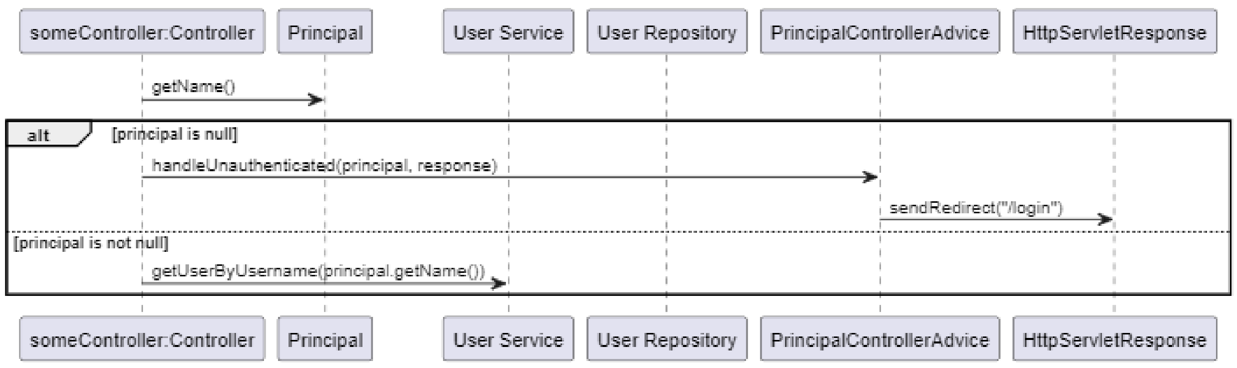
Integration

The Category Tree and Attribute Recommendation System work hand in hand. When a product is classified under a category, the Attribute Recommendation System can provide the most relevant attributes for that category, guiding the user in detailing the product specifications efficiently.

Session-based User Information Retrieval

Overview:

In Spring-based applications, the `Principal` interface represents the current authenticated user. It provides a mechanism to verify authentication and access user-specific details.



Key Concepts:

Principal:

- Represents the authenticated user in Spring.
- Used to check if a user is authenticated and fetch their details, e.g., `principal.getName()`.

Custom Exception for Unauthenticated Users:

- `UnauthenticatedUserException` is thrown when principal is null, indicating a lack of authentication.

Global Exception Handling:

- `PrincipalControllerAdvice` uses `@ControllerAdvice` to handle exceptions globally.
- The method `handleUnauthenticated` redirects unauthenticated users to the login page when the custom exception is thrown.

Usage in Controller:

- Within controllers in the project, Principal checks user authentication. If null, the custom exception is thrown.
- If authenticated, the user's details are retrieved using `principal.getName()`.

Takeaway:

Principal is a crucial tool in Spring for verifying user authentication and accessing user data. Proper handling, such as redirecting unauthenticated users, ensures secure and effective application functionality.

Shipment Tracking with Shippo

Integrating Shippo into the system significantly enhanced shipment tracking capabilities. Shippo provides a seamless API that can be utilized to fetch real-time tracking information from various carriers. This integration provides users with accurate and timely updates regarding their shipments.

Prerequisites

- A valid Shippo API key
- Shippo API dependency in pom.xml file:

```
<dependency>
      <groupId>com.goshippo</groupId>
      <artifactId>shippo-java-client</artifactId>
      <version>3.2.0</version> <!-- Use the latest version -->
    </dependency>
```

Integration Steps

1. API Key Configuration

A Shippo API key is obtained and configured within the system, stored as a variable in application.properties.

```
shippo.api.key=shippo_live_key_1234
```

2. Creating Shipment Tracking Service

A service class named ShippoTrackingService is established to manage interactions with the Shippo API. Within this class, the getTrackingInformation method is crafted to fetch tracking information.

```
@Service
public class ShippoTrackingService {

    public TrackDTO getTrackInformation(String carrier, String trackingNum){
        Shippo.setApiKey(shippoApiKey);
        // ... rest of the code
    }
    // ... rest of the code
}
```

3. Handling Exceptions

Exception handling is implemented to catch and manage any ShippoException that may arise during interactions with the Shippo API. Encountered ShippoException is wrapped in a RuntimeException.

```
catch (ShippoException e) {
    throw new RuntimeException("Failed to get tracking information", e);
}
```

4. Mapping Shippo Responses to DTOs

The response obtained from Shippo is mapped to Data Transfer Objects (DTOs) to facilitate its utilization within the application. Methods are created to map Track, Track.Address, and Track.TrackingEvent objects to their corresponding DTOs.

```
private TrackDTO mapTrackToTrackDTO(Track track) {
    // ... code to map track to TrackDTO
}
// ... rest of the mapping methods
```

5. Creating a Controller

A controller class named TrackingController is created to handle HTTP requests for tracking information. The ShippoTrackingService is utilized to fetch and return tracking information based on a transaction ID.

```
@RestController
@RequestMapping("/api/tracking")
```

```
public class TrackingController {
    @Autowired
    private ShippoTrackingService shippoTrackingService;
    // ... rest of the code
}
```

6. Fetching and Displaying Tracking Information

The /api/tracking/{transId} endpoint is devised to fetch and display tracking information for a particular transaction. It's ensured that the transaction exists and possesses valid tracking information before proceeding.

```
@GetMapping("/{transId}")
@ResponseBody
public TrackDTO getTrackingInformation(@PathVariable("transId") long transId)
{
    // ... rest of the code
}
```

Spring Profiles

The system is configured to utilize Spring Profiles to manage environment-specific configurations seamlessly. By leveraging Spring Profiles, the project can easily switch between different configurations based on the environment it is running in, namely development (dev) or production (prod).

```
# Set profile to dev or prod
spring.profiles.active=dev
```

Configuration Setup:

Default Profile Setting

Within the resources folder, the application.properties file is configured to allow users to define a default active Spring profile.

Profile-Specific Configuration Files:

Under the profiles folder, two properties files are created for each environment: application-dev.properties for development and application-prod.properties for production.

Profile Configuration Example:

Here's an example of what the application-prod.properties file looks like:

```
spring.datasource.data=classpath:prod-data.sql
spring.jpa.hibernate.ddl-auto=update
```

Profile Activation and Usage:

Profile Activation:

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

Configuration Loading:
When the system starts up, it reads the spring.profiles.active property to determine the active profile. It then loads the configurations from the corresponding profile-specific properties file located in the profiles folder.

Database Configuration:
In the provided example for prod, the spring.datasource.data property is set to point to a SQL script file (prod-data.sql) which contains the data to be loaded into the database. Additionally, the spring.jpa.hibernate.ddl-auto property is configured to update, allowing Hibernate to update the database schema whenever necessary.

Synopsis

Requirements to Operate Software:

Eclipse IDE for Enterprise Java and Web Developers – 2023 – 06.
MySQL Server – 8.0 or above.
MySQL Workbench – 8.0 or above.
MySQL Shell - 8.0 or above.
Operating System capable of running software above.

*Reference the 'Updated Install Manual' within Program Documents for more details on the process of installing each product and the steps for first time setup.

After setup, the program can be run by right clicking the 'sellingwidgets' folder within eclipse and selecting RunAs -> Springboot Application.

Upon initialization of the program, the user is brought to the index page where there are a variety of options including 'login,' 'signup,' 'browse,' 'motto,' 'FAQ,' and 'contact us.'

The software contains to accounts in which are utilized for testing, they include:

Regular User Account:
Username: userName
Password: testPass

Administrative User Account:
Username: useradminwidget
Password: useradmin

*Each of the functions within the program are described in detail as well as the operations in which they can perform. Reference the ‘User Manual’ within Program Documents.