# Java Development Final Capstone

## Capstone 3: E-Commerce API

Spring Boot API

# Capstone Description

This capstone is an e-commerce application, or an online store, for a company named EasyShop. You should create a new GitHub repository for this project. Give it a meaningful name that describes what the project is about.

Clone your repo into the `C:\pluralsight\LearnToCode_Capstones` folder.

You should create a GitHub project board to manage your work. Use the project requirements below to create user stories on your board so that you can plan and manage your time throughout the project.

For this project you will assume the role of a backend developer for an existing website. The website is already operational and has been published as Version 1. EasyShop is ready to begin development on Version 2.

The website uses a Spring Boot API project for the backend server, with a MySQL database for data storage. You will be given the existing project code to modify. The starter code also includes the database script for the existing data structure.

Since your code changes will all be in the API project you should rely heavily on Postman to test your application endpoints and your logic. However, a front-end website project is also available in your starter code, so that you can test your work, and see how your API is used on the web.

## Bugs

The current API code is functional, but there are a few bugs in the current project. You will need to find and fix those bugs before adding any new features. You should use manual debugging and write unit tests to help fix these bugs.

## New Features

Although you are not developing this project from scratch, you have been asked to develop several new features. You will need to plan, develop, and test the new features as you work on them.

## General

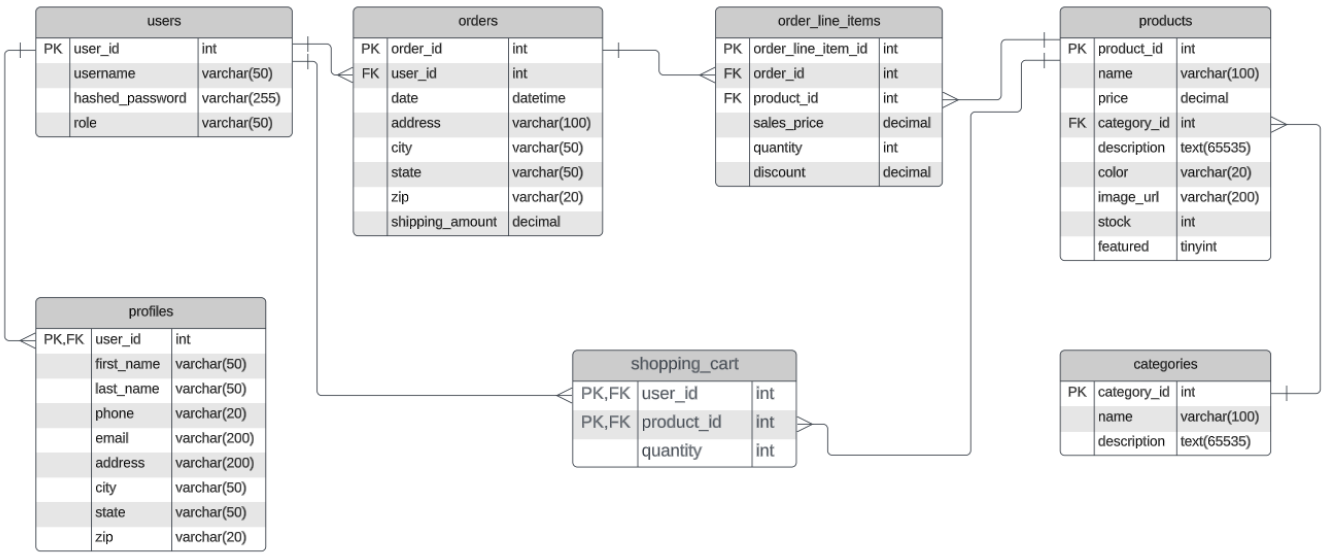
Be sure to use diagramming tools and the GitHub project board to plan your work and to manage your time.

## Demo

During the presentation of your final capstone, you will demonstrate the changes you have made throughout this project. You can do this by running the front-end web application and highlight the new features that you added. You should also be prepared to show some of the Postman scripts that you used to test your API.

# Capstone Setup

The database folder contains a database script (`create_database.sql`). Open this script in MySql Workbench and execute it to create the easyshop database as seen here.




The database script includes several products and 3 sample users (user, admin and george). The password for all demo users is `password`.

### Application structure

The application has uses an AuthenticationController to allow new users to register and existing users to login with the following urls.

### Register

```
POST http://localhost:8080/register

{
  "username": "admin",
  "password": "password",
  "confirmPassword": "password",
  "role": "ADMIN",
}
```

## Login

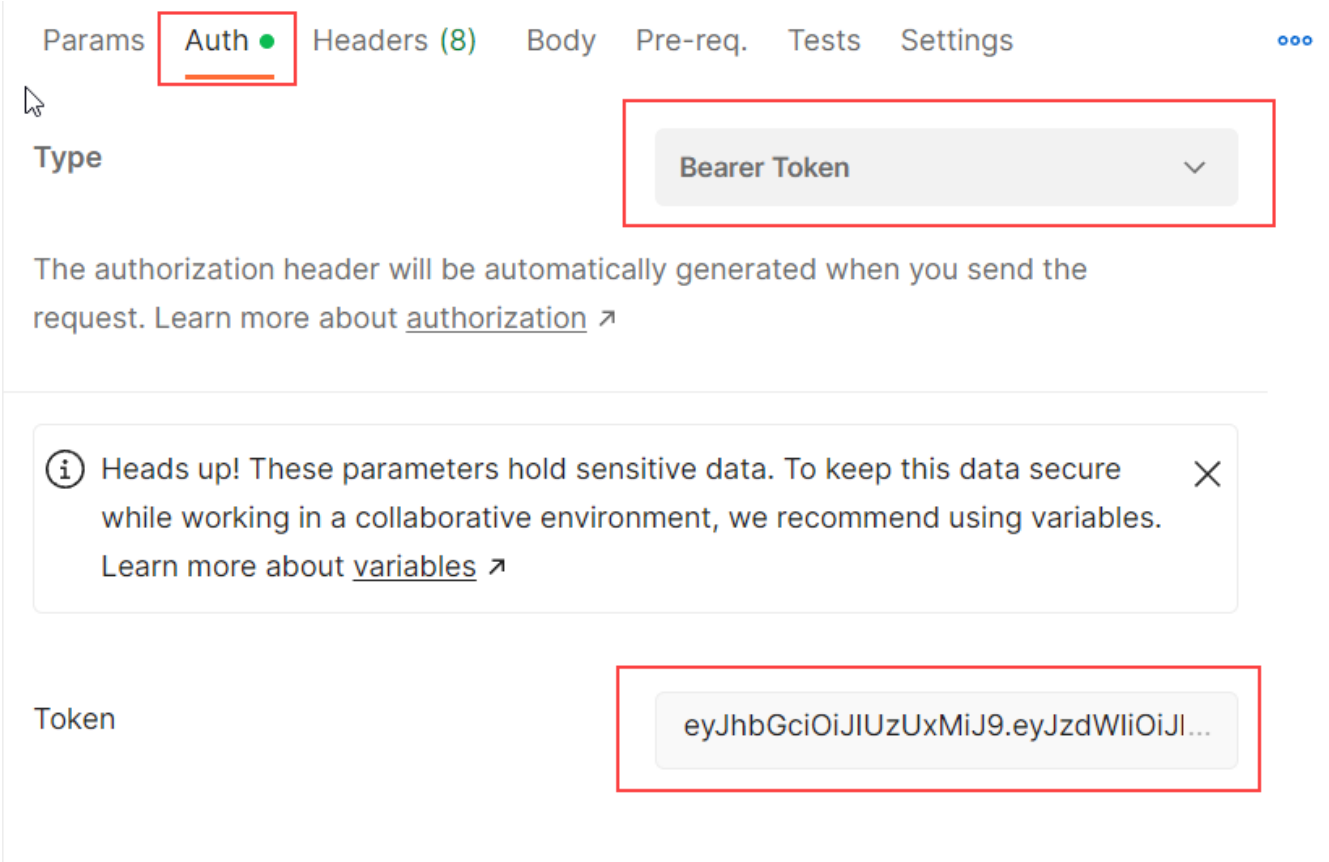
```
POST http://localhost:8080/login

{
  "username": "admin",
  "password": "password"
}
```

When a user logs in the API will return the relevant user information, including a JWT authentication token:


```
1  {
2      "token": "eyJhbGciOiJIUzUxMiJ9.
           eyJzdWIiOiJhZG1pbiIsImF1dGgiOiJST0xFX0FETUlOIiwiZXhwIjoxNjg3NTY5
           OTI5fQ.
           xkslKm7rX4PF0GzzwHSUbaUhwzUnV2ncP006MzHvdIOCEfHkcfnJZTB6JgDEG8LJ
           9EV9VZ6uUCee67WMzJe3Fw",
3      "user": {
4          "id": 2,
5          "username": "admin",
6          "authorities": [
7              {
8                  "name": "ROLE_ADMIN"
9              }
10         ]
11     }
12 }
```


The client application (postman and the sample website) will use this token to certify each request. You will need to add this token to postman requests for **all endpoints** which require a user to be logged in.




After running the database script, run the application and use Postman to verify that the following functions work.

# Application Requirements

Version 1 of this website allows users to browse products in various categories, add them to a shopping cart and check out to order the products. All of the features of the UI are fully functional. The changes and development will only need to be done in the backend Spring Boot Java API project.

The existing API code includes:

- user registration and login
- the ability to display products by category
- search for or filter the products list.

## Bugs

The search/filter functionality will have some bugs and may return incorrect search results. You should test different search and filter criteria to find and fix those bugs. (Include unit tests as part of this process)

# Phase 1 - CategoriesController

The `CategoriesController` class has been created for you but none of the methods have been implemented yet. You must implement the code for each function and add the propper annotations in the controller.

Only administrators (users with the `ADMIN` role) should be allowed to insert, update or delete a category.

The `MySqlCategoriesDao` also has all necessary functions defined, but you need to write the code to implement the functions.

The `Category` model has been defined for you.

## Requirements

The JSON for a category should be in the following format.

```
{
    "categoryId": 1,
    "name": "Electronics",
    "description": "Explore the latest gadgets and ..."
}
```

The following REST methods should be exposed by the API.

| VERB | URL | BODY |
|---|---|---|
| GET | http://localhost:8080/categories | NO body |
| GET | http://localhost:8080/categories/1 | NO body |
| POST | http://localhost:8080/categories | Category |
| PUT | http://localhost:8080/categorids/1 | Category |
| DELETE | http://localhost:8080/categorids/1 | NO body |

# Phase 2 - Fix Bugs

The `ProductsController` endpoint is fully implemented with the following methods.

**NOTE:** only administrators are allowed to insert, update or delete products.

| VERB | URL | BODY |
|---|---|---|
| GET | http://localhost:8080/products | NO body |
| GET | http://localhost:8080/products/1 | NO body |
| POST | http://localhost:8080/products | Category |
| PUT | http://localhost:8080/products/1 | Category |
| DELETE | http://localhost:8080/products/1 | NO body |

## Bug 1

Users have reported that the product search functionality is returning incorrect results. You need to test the search logic to find and fix the bug(s). The search url has the following querystring parameters

| VERB | URL |
|---|---|
| GET | http://localhost:8080/products |

| Key | Type | Filter |
|---|---|---|
| cat | int | categoryId |
| minPrice | BigDecimal | price (lower range) |
| maxPrice | BigDecimal | price (upper range) |
| color | String | color |

**Examples**

```
http://localhost:8080/products?cat=1
http://localhost:8080/products?cat=1&color=red
http://localhost:8080/products?minPrice=25
http://localhost:8080/products?minPrice=25&maxPrice=100
```

## Bug 2

Some users have also noticed that some of the products seem to be duplicated. For example, a laptop is listed 3 times and it appears to be the same product, but there are slight differences, such as the description or the price.

If you look at the 3 laptops you notice that they are the same product. This laptop has been edited twice, the first time you updated the price, the second update was to the description. It appears that instead of updating the product, each time you tried to update, it added a new product to the database.

You need to find this bug and fix it so that administrators can safely update products.

# Phase 3 - Shopping Cart

## New Feature

Users should have the ability to add items to their shopping cart. This is a new feature that has not yet been implemented. The database already has a shopping cart table.

The shopping cart feature should only be available to logged in users. When an item is added to the shopping cart it gets added to the database for the current user. If a user logs out, the next time they log in the items that were added to the cart should still be in the cart.

A class for the `ShoppingCartController` has already been created but you need to add methods for the required REST actions.

| VERB | URL | BODY |
|---|---|---|
| GET | http://localhost:8080/cart | NO body |
| POST | http://localhost:8080/cart/products/15 | NO body |
| PUT | http://localhost:8080/cart/products/15 | has body |
| DELETE | http://localhost:8080/cart | NO body |

The **Get** action should return the shopping cart for the current user, including all shopping cart items (products). A method has been started with sample code to demonstrate how you can get the logged in users' username. You must use the userDao to get the user by username, because you will need the userId to get all the users shopping cart items.

The JSON for a shopping cart should be in the following format:

```
{
    "items": {
        "1": {
            "product": {
                "productId": 1,
                "name": "Smartphone",
                "price": 499.99,
                "categoryId": 1,
                "description": "A powerful and feature-
rich smartphone for all your communication needs.",
                "color": "Black",
                "stock": 50,
                "imageUrl": "smartphone.jpg",
                "featured": false
            },
            "quantity": 2,
            "discountPercent": 0,
            "lineTotal": 999.98
        },
        "15": {
            "product": {
                "productId": 15,
                "name": "External Hard Drive",
                "price": 129.99,
                "categoryId": 1,
                "description": "Expand your storage capacity and backup your import
ant files with this external hard drive.",
                "color": "Gray",
                "stock": 25,
                "imageUrl": "external-hard-drive.jpg",
                "featured": true
            },
            "quantity": 1,
            "discountPercent": 0,
            "lineTotal": 129.99
        }
    },
    "total": 1129.97
}
```

The **POST** action is used to add a new product to the shopping cart. Note that the url is `http://localhost:8080/`**`cart/products/15`** the number is the id of the product that you want to add to the cart. If the current user's shopping cart does not yet include the product id, you should add (insert) the userId and productId to the shopping cart table with a quantity of 1.

If the user already has the specified product in their cart, then you should update the shopping cart table, and update the quantity to increase it by 1.

The **DELETE** method should clear the shopping cart. Delete all of the current user's items from the shopping cart table.

## Optional / Bonus - PUT

The **PUT** method is similar to the POST method, but a body should be included with the Request. The url is `http://localhost:8080/`**`cart/products/15`**. The productId is the id that should be updated in the database. The request body will include the new quantity that should be updated in the database.

Body format:

```
{
  "quantity": 3
}
```

The shopping cart item should only be updated if the user has already added the product to their cart.

# Optional Phase 4 - User Profile

When a user registers for an account, a user profile record is also created (in the `profiles` table). A user should be able to view and update their profile. You need to create a `ProfileController` and add the following methods.

| VERB | URL | BODY |
|---|---|---|
| GET | http://localhost:8080/profile | NO body |
| PUT | http://localhost:8080/profile | Profile body |

A ProfileDao interface and MySqlProfileDao already exist. They are used to create the profile when a user registers.

You will need to update the DAO to add the getByUserId and update methods.

Work with your team to plan and design the requirements of this feature. Then write the implementation code.

# Optional Phase 5 - Checkout

## New Feature

When a user is ready to check out you need to convert their shopping cart into an order. This feature will require a significant amount of planning, because none of the code required for this feature exists in the starter code.

You should create an `OrdersController` that will be used in the checkout process. It should include the following REST method.

| VERB | URL | BODY |
|---|---|---|
| POST | http://localhost:8080/orders | NO body |

The **POST** method does not require a body. The method should retrieve the current user's shopping cart. It should create and insert a new Order into the orders table, then create an OrderLineItem for each shopping cart item and add each item to the database.

Once the order has been created you should clear the shopping cart.

# Future Versions

Take some time with your team to discuss what other features might be worth developing in future versions. Consider other e-commerce websites that you have used (such as Amazon) and think about features that are available.

There is no need to fully plan or implement these features but make a list of these features and rank them in order of priority.

What changes do you think would need to be made to your project to implement these features?

# Other General Project Requirements

Your project must also meet the following requirements:

## Git Repository

- Your code must be in a public GitHub repository
- The repository must contain an appropriate Git commit history
  - **Minimally**, you should have a commit for each meaningful piece of work completed
- It must contain an informative README file that:
  - Describes your project
  - Includes images of your application screens
  - Describes/shows one interesting piece of code from your project

## Class Demonstrations

Each student will be given 10 minutes to demonstrate their project to the class on "project demonstration day". During this time, you will:

- Present your application - run through the website ordering process
  - You should also demonstrate how you can use Postman to interact with and test your API
- Describe / show one interesting piece of code that you wrote
- Answer questions from the audience if time permits