Spring MVC (MVC)

Exercise MVC.1

The Setup:

In this exercise we will create a simple CRUD (Create, Retrieve, Update, Delete) application with Spring MVC. Start by downloading **W3D1-MVC** and add the following dependencies:

For the web:

```
        <dependency>
            <groupId>jakarta.servlet.jsp.jstl</groupId>
            <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
            <version>3.0.2</version>
        </dependency>
        <dependency>
            <groupId>org.glassfish.web</groupId>
            <artifactId>jakarta.servlet.jsp.jstl</artifactId>
            <version>3.0.1</version>
        </dependency>
```

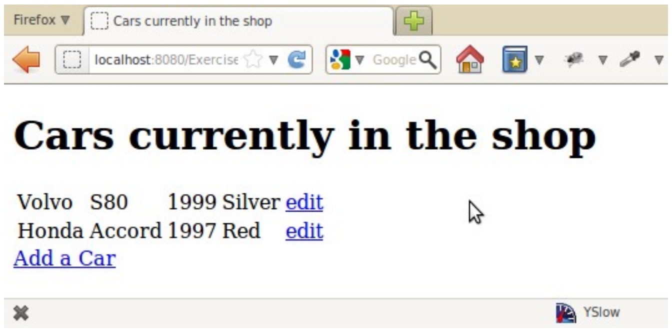
For Spring:

```
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>6.1.14</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>6.1.14</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>6.1.14</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-orm</artifactId>
            <version>6.1.14</version>
        </dependency>
```

For Hibernate:

```
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>6.6.1.Final</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.4.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-core</artifactId>
            <version>2.24.1</version>
        </dependency>
```

Once everything is setup running the project on the tomcat server should open the following page in your browser:



Cars currently in the shop

Volvo S80 1999 Silver edit
Honda Accord 1997 Red edit
Add a Car

The Application:

The provided code is reasonably simple, **CarController** uses **CarService**, which in turn uses **CarDao** to create, retrieve, update and delete **Car** objects, after which the controller forwards to one of the views.

The Exercise:

The goal of this exercise is to make a Book CRUD application similar to the provided Car CRUD application. I recommend creating a new / separate application. Use a book class similar to:

```
public class Book {
	private Integer id;
	private String title;
	private String ISBN;
	private String author;
	private double price;
}
```

The core items that you will need to make are the **BookController** class, and a new set of views related to the book store e.g. BookList, BookDetail, AddBook. And of course a BookService and BookDao (not a lot of code involved). You may also want to update the import.sql file.

An important aspect in the code is how both the viewAdd() method and get() method in the controller utilize the same carDetail view. Be sure that you understand how that works.

You can either copy paste many of the files from the car application and update them, or for a greater learning experience you can start from scratch.

Spring Data (SPD)

Exercise SPD.1

The Setup:
The goal of this exercise is to test using Spring Data for DAO generation. Start by making a copy of your **W3D1-MVC** project (either the Book or Car is fine, although I'll describe Book here), and add the following dependencies:

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>3.3.5</version>
</dependency>
```

The Exercise:
a) Generating a the DAO

   Add @EnableJpaRepositories to your Config.java. Optionally it can take a package name to scan for your packages (adding this can fix issues where it's not finding the repositories).

   Delete the BookDao class, then make IBookDao extend JpaRepository and delete all the methods from the interface.

   Update BookService to use the JpaRepository methods instead of the methods that we used to have on the interface. Important: inside the get() method of the BookService use bookDao.findById(id).get() instead of using bookDao.getById(id).

   If everything went well you should now be able to run the application with your generated DAO.

b) Change to using .getById(id) in your BookService. Running the application now should result in a lazy loading exception. Do you understand why this is? (.getById() returned a proxy).

   To fix this we can add the OpenEntityManagerInViewFilter to the MyWebAppInitializer