DAT152: Lab – Exercise #3

Spring Web MVC Framework and Security

For this lab, you will extend the library service built with the Spring Web MVC framework to include two additional features.

1. Add Author
2. Delete Book

Preambles: Download and unzip *Lab3a.zip*. Then, import the maven project into your preferred IDE.
Some ways to run your project:

1. Run the project from a command prompt (e.g., Mac Terminal). Navigate to the root folder of your project and run the maven command: ./mvnw spring-boot:run
2. From your IDE: Right click on 'LibraryApplication.java' in the "no.hvl.dat152.main" package and "Run As" Java Application.

If everything goes well, point your url to "http://localhost:8090/" (default server port=8090)
You can change the port and other settings in the 'application.properties' file located under "src/main/resources"

Task #1: **Add Author**:

Presentation Layer: Create addauthor.html (folder = src/main/resources/templates)
Tasks: Create a view named "addauthor.html in "src/main/resources/templates" folder. The view should have a form element with two *input* elements – firstname and lastname. When the submit button on the form is pushed/pressed, it should **post** the submitted data to the URL path: **/addauthor**

Service Layer: update AuthorService.java (package = no.hvl.dat152.service)
You should use inject the AuthorRepository class into the AuthorService class. Don't forget to use @Autowired.
Tasks: Write a method "saveAuthor(Author author)". The simple task the method should do is to call the 'save' method on AuthorRepository object and save the 'author' data. The return type of the method should be Author.

Controller: update AuthorController.java (package = no.hvl.dat152.controller)
Don't forget to annotate the class with the @Controller annotation.
Tasks: Write two methods to handle GET and POST requests.

1. create(). This controller method should return the view "addauthor". Remember you must use @RequestMapping annotation and set the value (/addauthor) and the method (GET) attributes.
2. create(firstname, lastname). This method should redirect to the home page when it's done performing its job. That is return "redirect:/". Set the value (/addauthor) and the method (POST) attributes in your @RequestMapping. Also, remember to use the @RequestParam annotation for each of the method arguments to map the form's attributes to your method's arguments.

Task #2: Delete Book

Presentation Layer: update 'viewbooks.html'
Task: Add a hyperlink to send a GET request to the URL '/deletebook?id='bookid

Service Layer: update BookService.java
Tasks: implement the method "deleteBookById(id). This method throws BookNotFoundException, therefore, you need to first check whether the book exist before you perform a delete operation.

Controller: update BookController.java
Tasks: Create "delete(id, Model)" method. This method will accept two arguments, the 'id' of the book and the 'Model' object.

- Map the id of the GET request to the id of the method by using @RequestParam
- Perform a delete operation by calling the deleteBookById of the BookService class. You will need to throw BookNotFoundException.
- Find all books by calling the findAll() method of the BookService. Add the results as "books" attribute to the model object to give 'viewbooks.html' access to the data.
- This controller method should then return the view "viewbooks".

Task 3: Security Layer

Download and unzip *Lab3b.zip*. Run the application and see that the basic features of login and logout function.

Task 3.1: Custom Login Form

- View
    - Create your own custom login form. Your custom login form will have two fields, username and password.
    - Form method will be POST and the action can should be /login.

- Controller
    - You'll need a controller to process a GET request that returns the login form.
- Security Configuration
    - In the WebAppSecurityConfig.java, configure your securityfilter for formlogin, and logout.
    - Configure your securityfilter to secure the views/endpoints based on roles

Task 3.2: Authorization decision in Thymeleaf

- View
    - Thymeleaf has custom tags and expression language for authorization decisions in the views. Do some search and find out the security/authorization tag(s) or EL you can use to display 'Add Book' and 'Add Author' menu ONLY when the admin (i.e. with ADMIN role) has logged in. Other users should only see the 'Home' and 'View Books' menu.