Lab: Creating Bookhut Web Site using Servlets and JSP

This **tutorial** provides step-by-step guidelines to build a **"Bookhut" app** in Java, Servlets, JSP and JSTL. The app should implement **home** / **sing up** / **sing in** / **add-book** / **shelves** pages. No database will be used.

Project Specification

Design and implement a **"Bookhut" web application** in Java, Servlets, JSP and JSTL. Create 5 **JSP** pages with the following functionality:

- **Home**
  - Home page of the bookhut site
  - Should be able to redirect to all other pages
- **Sign Up**
  - Register a new user
- **Sign In**
  - In case of success, a session should be created.
  - Sign out option that will **invalidate** the session
- **Add Book**
  - Simple page to create new books
- **Shelves**
  - Page that will **list** all the books
  - Has possibility to **edit** each book
  - Has possibility to **delete** each book

1. Project Setup

Create a new maven project called Bookhut. Add Java EE Web Application Framework support to the project. This is a recommended pom file:

pom.xml

```
<properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>

<dependencies>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-api</artifactId>
        <version>7.0</version>
    </dependency>
    <dependency>
        <groupId>jstl</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>
    <dependency>
        <groupId>org.modelmapper</groupId>
        <artifactId>modelmapper</artifactId>
        <version>0.7.5</version>
    </dependency>
</dependencies>
```

2. Create Models

Create two main entities for the application:

- **User**
  - Id
  - Username
  - Password
- **Book**
  - Id
  - Title
  - Author
  - Pages
  - Creation Date

We would need **three models** that will serves as **DTO**.

Binding Models:

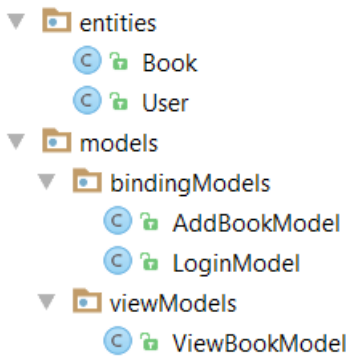
- **LoginModel**
  - Username
  - Password

- **AddBookModel**
  - Title
  - Author
  - Pages

View Models:

- **ViewBookModel**
  - Title
  - Author
  - Pages

Here is an overview of the model structure:



3. Create Repositories

For this project **no database is required**. Use simple list implementation to store data.

Create two repositories:

- User Repository

UserRepository.java

```java
public interface UserRepository {

    void createUser(User user);

    User findByUsernameAndPassword(String username, String password);
}
```

- Book Repository

BookRepository.java

```java
public interface BookRepository {

    void saveBook(Book book);

    List<Book> getAllBooks();

    void deleteBookByTitle(String title);

    Book findBookByTitle(String title);
}
```

In order to have a single instance of the lists use Singleton Pattern:

BookRepositoryImpl.java

```java
public class BookRepositoryImpl implements BookRepository {

    private static BookRepositoryImpl bookRepository;

    private List<Book> books;

    private BookRepositoryImpl() {
        this.books = new ArrayList<>();
    }

    public static BookRepository getInstance() {
        if (bookRepository == null) {
            bookRepository = new BookRepositoryImpl();
        }

        return bookRepository;
    }
}
```

4. Create Services

You would need a services for each that will transform entities to models and the other way around. Use **Model Mapper**.

Two services are required:

- **User Service**

UserService.java

```java
public interface UserService {

    void createUser(LoginModel loginModel);

    LoginModel findByUsernameAndPassword(String username, String password);
}
```

- **Book Service**

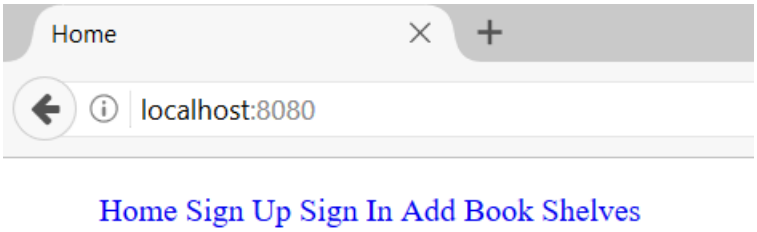
BookService.java

```
public interface BookService {

    void saveBook(AddBookModel addBookModel);

    List<ViewBookModel> getAllBooks();

    ViewBookModel findBookByTitle(String title);

    void deleteBookByTitle(String title);
}
```

5. Create Home Page

Home page should be **blank with simple menu**.

5.1 Create menu.jsp



- Home – should redirect to "/"
- SignUp – should redirect to "/singup"
- SignIn – should redirect to "/singin"
- Add Book – should redirect to "/add"
- Shelves – should redirect to "/shelves"

5.2 Create home.jsp

Include the created menu:

home.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
    <head>
        <title>Home</title>
    </head>
    <body>
        <jsp:include page="menu.jsp"></jsp:include>
    </body>
</html>
```

5.3 Create Home Controller Servlet

HomeController.java

```
@WebServlet("")
public class HomeController extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        request.getRequestDispatcher("templates/home.jsp").forward(request,response);
    }
```

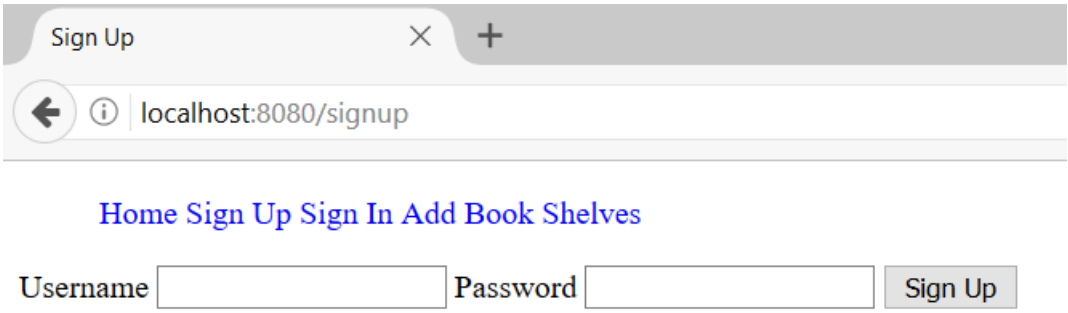
```
}
```

6. Create Sing Up Page

Create Sign Up page that listen to route **/singup**. It should **store a new User** in the list which we use as a database. The **get request** should return a **signup.jsp with a simple form in it**. The post method should read the form parameters, create a new user and redirect to **/signin**.

SingUpController.java

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    LoginModel loginModel = null;
    String signUpText = request.getParameter("signup");
    if(signUpText != null){
        //TODO Implement the logic
        response.sendRedirect("/signin");
    }
}
```



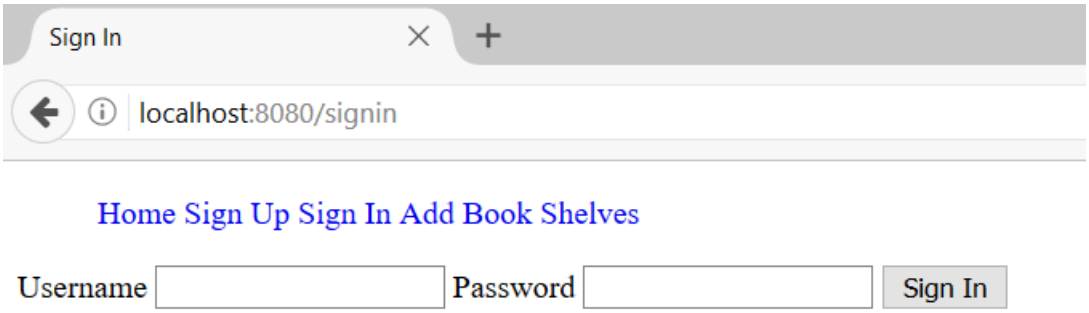
7. Create Sing In Page

Create Sign In page that listen to route **/singin**. Check whether the user exists. If it exists redirect to **/home**, otherwise stay on the **same page**.

SingInController.java

```
@WebServlet("/signin")
public class SignInController extends HttpServlet {

    private UserService userService;

    public SignInController() {
        this.userService = new UserServiceImpl();
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        LoginModel loginModel = null;
        String signInText = request.getParameter("signin");
        if(signInText != null){
            //TODO Implement the logic
        }

        if(loginModel != null) {
            HttpSession session = request.getSession();
            session.setAttribute("LOGIN_MODEL", loginModel);
```

```
            response.sendRedirect("/");
        } else {
            response.sendRedirect("/signin");
        }
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        //TODO Implement the logic
    }
}
```



8. *Create Sing Out Functionality

Crate a SignOutController that will invalidate the current session.

SingOutController.java

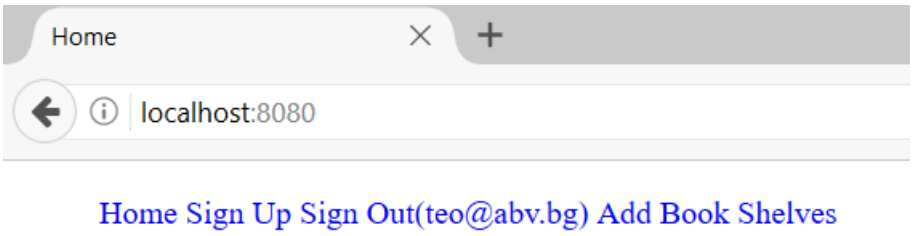
```
@WebServlet("/signout")
public class SignOutController extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        HttpSession session = request.getSession();
        session.invalidate();
        response.sendRedirect("/");
    }
}
```

In order to reach the controller and modify the menu.jsp to redirect to **/signout** when you are signed in.

menu.jsp

```
<ul>
    <li><a href="/">Home</a></li>
    <li><a href="/signup">Sign Up</a></li>
    <%
        LoginModel loginModel = (LoginModel) session.getAttribute(Config.LOGIN_MODEL);
        String username = null;
        //TODO Implement the logic
    %>
    <c:set var="username" value="${USERNAME}" scope="session"/>
    <c:choose>
        <c:when test="${username != null}">
            //TODO Implement the logic
        </c:when>
        <c:otherwise>
            //TODO Implement the logic
        </c:otherwise>
    </c:choose>
    <li><a href="/add">Add Book</a></li>
    <li><a href="/shelves">Shelves</a></li>
```

```
</ul>
```



9. Create Add Book Page

The page should take basic input and save the book in **the list we use as a database**. It should listen to route **/add.**

AddBookController.java

```java
@WebServlet("/add")
public class AddBookController extends HttpServlet {

    private BookService bookService;

    public AddBookController() {
        this.bookService = new BookServiceImpl();
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        //TODO Implement the logic
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        String add = req.getParameter("add");
        if (add != null) {
            //TODO Implement the logic
        }
    }
}
```

10. Create Shelves Page

The page should list all the books in **the list we use as a database**. It should listen to route **/shelves.** Besides ViewBookModel properties add two additional:

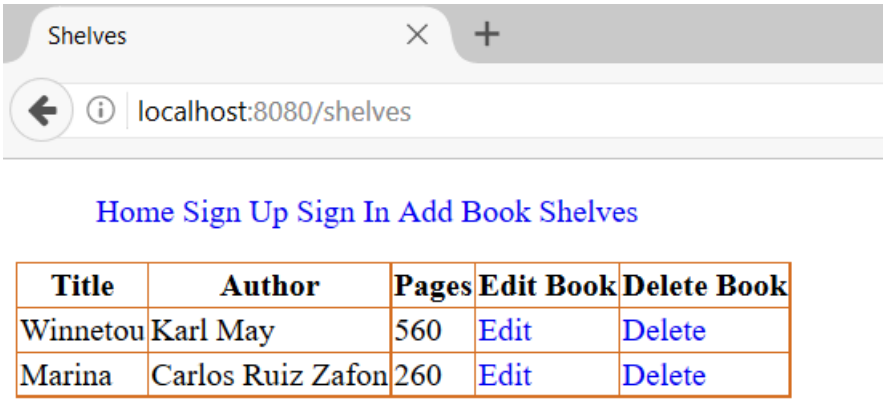
- **Edit – it should refer /shelves/edit/{book name}**
- **Delete – it should refer /shelves/delete/{book name}**

ShelfController.java

```java
@WebServlet("/shelves")
public class ShelfController extends HttpServlet {

    private BookService bookService;

    public ShelfController() {
        this.bookService = new BookServiceImpl();
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        List<ViewBookModel> viewBookModels = this.bookService.getAllBooks();
        //TODO Implement the logic
    }
}
```

shelves.jsp

```jsp
<table class="tb">
    <thead>
        <th>Title</th>
        <th>Author</th>
        <th>Pages</th>
        <th>Edit Book</th>
        <th>Delete Book</th>
    </thead>
    <tbody>
        <c:set var="books" value="${books}" />
        <c:forEach var="book" items="${books}">
            <tr>
                <td>
                    <c:out value="${book.title}"/>
                </td>
                //TODO Implement the logic
                <td>
                    <a href="/shelves/edit/${book.title}">Edit</a>
                </td>
                //TODO Implement the logic
            </tr>
        </c:forEach>
    </tbody>
</table>
```



11. *Create Edit Functionality

Create a controller that will listen to route **/shelves/edit/{book name}**. If the edit is done redirect to **/shelves**.

EditBookController.java

```
@WebServlet("/shelves/edit/*")
public class EditBookController extends HttpServlet {

    private BookService bookService;

    public EditBookController() {
        this.bookService = new BookServiceImpl();
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        String tokens[] = req.getRequestURI().split("/");
        String title = URLDecoder.decode(tokens[3], "UTF-8");
        ViewBookModel viewBookModel = this.bookService.findBookByTitle(title);
        if(viewBookModel != null){
            //TODO Implement the logic
        }
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        String edit = req.getParameter("edit");
        if(edit != null){
            //TODO Implement the logic
        }
    }
}
```

12. *Create Delete Functionality

Create a controller that will listen to route **/shelves/delete/{book name}**. If the edit is done redirect to **/shelves**.

DeleteBookController.java

```
@WebServlet("/shelves/delete/*")
public class DeleteBookController extends HttpServlet {

    private BookService bookService;

    public DeleteBookController() {
        this.bookService = new BookServiceImpl();
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        //TODO Implement the logic
    }
}
```