**Online Bookstore Application Documentation**

The Online Bookstore application is designed to provide a platform for users to browse, purchase books, and leave reviews. It includes functionalities for both regular users and admins, with secure authentication and role-based access control.

**User Management**

UserController Functionalities

1. **List All Users (GET /users, Admin only):**
   * **Functionality:** Retrieves a list of all registered users in the system.
   * **Access Control:** Restricted to users with the 'ADMIN' role.
   * **Process:** Admins can view comprehensive user data, assisting in user management and support.
   * **Error Handling:** Handles unexpected exceptions, potentially returning an Internal Server Error status (500).
2. **Get User by Username (GET /users/{username}):**
   * **Functionality:** Allows retrieval of detailed information for a specific user.
   * **Access Control:** Accessible by the authenticated user for their user details only.
   * **Process:** Validates user existence and returns user details.
   * **Error Handling:** Returns a 404 Not Found status for non-existing usernames.

AuthenticationController Functionalities

1. **User Registration (POST /auth/register):**
   * **Functionality:** Enables new users to create an account in the application by providing a request body, for example:
2. {
3. "email": "user@user.com",
4. "username": "user",
5. "password": "password"
6. }
   * **Process:**
     + Validates the uniqueness of username and email.
     + Encodes the password for security.
     + Registers the user and initializes an empty cart.
   * **Error Handling:** Handles duplicate username or email scenarios, returning a Bad Request status (400) with an appropriate message.
7. **Test Authentication (GET /auth/test-login):**
   * **Functionality:** Checks and confirms the current user's authentication status and roles.
   * **Process:**
     + Useful for testing login status and role assignments.
   * **Error Handling:** Typically, Spring Security handles authentication errors before reaching this point.

**Book Management**

BookController Functionalities

1. **Add Books (POST /books, Admin only):**
   * **Functionality:** Allows admins to add new books to the inventory by giving a request body, for example:
2. {
3. "title": "Frankenstein",
4. "author": "Mary Shelley",
5. "isbn": "978-0486282114",
6. "price": 6.99,
7. "stock": 40,
8. "description": "A novel about a young scientist who creates a sapient creature.",
9. "genre": "Gothic Novel",
10. "publicationYear": 1818
11. }
    * **Access Control:** Restricted to users with the 'ADMIN' role.
    * **Process:**
      + Books are added with details like title, author, genre, and stock.
    * **Error Handling:** Handles data validation errors and unexpected exceptions.
12. **List Books (GET /books):**
    * **Functionality:** Retrieves a list of all available books for users to view.
    * **Process:**
      + Enables browsing of the book inventory without specific search criteria.
    * **Error Handling:** Handles unexpected exceptions, potentially returning an Internal Server Error status (500).
13. **Get Book by Title (GET /books/{title}):**
    * **Functionality:** Retrieves detailed information about a book based on its title.
    * **Error Handling:** Returns a not found status (404) if the book does not exist.
14. **Get Books by Author/Genre (GET /books/author/{author}, GET /books/genre/{genre}):**
    * **Functionality:** Lists books filtered by a specific author or genre.
    * **Error Handling:** Handles cases where no books are found for the given criteria.
15. **Update Book Stock (PUT /books/update/{title}, Admin only):**

* **Functionality**: Allows admins to update the stock of a specific book .
* **Access Control:** Restricted to users with the 'ADMIN' role.
* **Process:**
  + Stock updates adjust the available quantity of a book.
* **Error Handling:** Handles scenarios like non-existent books, returning appropriate error messages and status codes.

1. **Delete Book (DELETE /books/delete/{title}, Admin only):**
   * **Functionality**: Allows admins to remove a book from the inventory.

* **Access Control:** Restricted to users with the 'ADMIN' role.
* **Process:**
  + Book deletion removes the book from the bookstore's listing.
* **Error Handling:** Handles scenarios like non-existent books, returning appropriate error messages and status codes.

**Cart Management**

CartController Functionalities

1. **Add to Cart (POST /users/{username}/cart/add):**
   * **Functionality:** Allows users to add a specific book to their shopping cart by providing a request body, for example:
2. {
3. "title": "Pride and Prejudice",
4. "quantity": 1
5. }
   * **Access Control:** Accessible by the authenticated user matching the **{username}**.
   * **Process:**
     + Verifies stock availability and updates the cart with the selected book and quantity.
   * **Error Handling:** Returns access denied for unauthorized users, and bad request for issues like insufficient stock.
6. **Get Cart (GET /users/{username}/cart):**
   * **Functionality:** Retrieves the contents of the user's shopping cart.
   * **Error Handling:** Returns access denied for unauthorized users, and bad request for non-existent carts.
7. **Empty Cart (POST /users/{username}/cart/empty):**
   * **Functionality:** Empties all items from the user's shopping cart.
   * **Error Handling:** Handles unauthorized access and non-existent cart scenarios.

**Order Management**

OrderController Functionalities

1. **Create Order (POST /orders/{username}/create):**
   * **Functionality:** Allows a logged-in user to create an order from the items in their cart.
   * **Access Control:** This endpoint is accessible only by the authenticated user whose username matches the **{username}** in the path.
   * **Process:**
     + Verifies the user's authentication status.
     + Initiates the order creation process through **OrderService**.
     + Handles scenarios such as empty carts or insufficient stock.
     + Returns the created order details upon successful order placement.
   * **Error Handling:** Returns a 403 status code for unauthorized access, and a 400 status code with an error message for any exceptions during order processing.
2. **Get All Orders (GET /orders, Admin only):**
   * **Functionality:** Retrieves a list of all orders placed in the bookstore.
   * **Access Control:** Restricted to users with the 'ADMIN' role.

* **Process:**
  + The admin user can retrieve a list of all orders without any user-specific filtering.
  + This functionality is particularly useful for order tracking and management from the admin's perspective.
* **Error Handling:** Returns an Internal Server Error status (500) in case of any unexpected exceptions.

1. **Get Orders by Username (GET /orders/{username}/get):**
   * **Functionality:** Allows a logged-in user to retrieve their order history.
   * **Access Control:** Accessible only by the authenticated user whose username matches the **{username}** in the path.
   * **Process:**
     + Verifies the user's authentication status.
     + Retrieves a list of orders specific to the authenticated user.
     + Ensures privacy and security by restricting access to the user's own orders.
   * **Error Handling:** Returns a 403 status code for unauthorized access and a 500 status code in case of any unexpected exceptions.

**Review Management**

ReviewController Functionalities

1. **Create Review (POST /reviews):**
   * **Functionality:** Enables users to leave reviews on books, including a rating and a comment by providing a request body, for example:
2. {
3. "bookTitle": "Pride and Prejudice",
4. "username": "user",
5. "rating": 3,
6. "comment": "boring"
7. }
   * **Process:**
     + Associates the review with the specified book and user.
     + Stores the review in the system for others to view.
   * **Error Handling:** Handles scenarios where the book or user does not exist and returns appropriate error messages.
8. **Get Reviews by Book Title (GET /reviews/book/{title}):**
   * **Functionality:** Retrieves all reviews associated with a specific book.
   * **Process:**
     + Allows users to view what others have said about a book, aiding in purchase decisions.
   * **Error Handling:** Handles cases where the book does not exist or has no reviews.
9. **Delete Review (DELETE /reviews/{reviewId}, Admin only):**
   * **Functionality:** Allows admins to delete a specific review.
   * **Access Control:** Restricted to users with the 'ADMIN' role.
   * **Process:**
     + Removes a review from the system, typically used for moderation purposes.
   * **Error Handling:** Handles scenarios where the review does not exist.

**Security and Authentication**

* The application uses Spring Security to manage authentication and authorization.
* Specific endpoints are secured based on user roles (**USER** and **ADMIN**), ensuring that users can only access data relevant to their account and role.
* Sensitive operations, like order management and review deletion, are restricted to authenticated and authorized users.