System Requirements Specification

Index

For

Book Nest App

Version 1.0

TABLE OF CONTENTS

BOOK NEST APPLICATION

System Requirements Specification

BACKEND-SPRING BOOT RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **Book Nest Application** is implemented using Spring Boot with a MySQL database, designed to enhance a dynamic platform for book enthusiasts. It's tailored to provide an engaging and interactive experience for users to explore, review, and purchase books. The application efficiently organizes books, authors, and categories, making it a comprehensive resource for literary exploration.

You are tasked with crafting an online bookstore where users can effortlessly sign up, manage their profiles, and interact with a vast library of books. The application should facilitate functionalities to manage book inventories, author details, book reviews, and user orders. This includes adding new books and authors, updating book details, posting and managing reviews, and handling book orders. The platform should ensure secure transactions, efficient data handling, and provide users with real-time updates on their orders and new book arrivals, enhancing the user's browsing and shopping experience.

Following is the requirement specifications:

	Book Nest Application
Modules	
1	User
2	Book
3	Author
4	Category
5	Review
6	Order
User Module Functionalities	
1	Register user
2	Get user profile
3	Update user profile

Book Module Functionalities	
1	Get all books
2	Get book by id
3	Add a book
4	Update details of a book

5	Delete a book
6	Search books by their title (or) author

Author Module Functionalities	
1	Get all authors
2	Add a new author

Category Module Functionalities	
1	Get all categories
2	Add a new category

Review Module Functionalities	
1	Add review
2	Get reviews for specific book
3	Update review
4	Delete review
5	Get reviews posted by a specific user
6	Get all reviews

Order Module Functionalities	
1	Place a new order
2	Get order by id
3	Get orders for specific user
4	Update order status
5	Process payment
6	Cancel an order

2 Assumptions, Dependencies, Risks / Constraints

2.1 User Constraints

- When fetching a user profile by ID, if the user ID does not exist, the service method should throw a NotFoundException with the message "User not found".
- When updating a user's profile, if the user ID does not exist, the service method should throw a NotFoundException with the message "User not found".

2.2 Book Constraints

- When fetching a book by ID, if the book ID does not exist, the service method should throw a NotFoundException with the message "Book not found".
- When updating a book:
    1) If the book ID does not exist, the service method should throw a NotFoundException with the message "Book not found".
    2) If any category ID associated with the book does not exist when updating categories, the service method should throw a NotFoundException with the message "Category not found".
- When deleting a book, if the book ID does not exist, the service method should throw a NotFoundException with the message "Book not found".

2.3 Review Constraints

- When updating a review, if the review ID does not exist, the service method should throw a NotFoundException with the message "Review not found".
- When deleting a review, if the review ID does not exist, the service method should throw a NotFoundException with the message "Review not found".

2.4 Order Constraints

- When fetching an order by ID, if the order ID does not exist, the service method should throw a NotFoundException with the message "Order not found with id [orderId]".
- When updating the status of an order, if the order ID does not exist, the service method should throw a NotFoundException with the message "Order not found with id [orderId]".
- When processing a payment for an order, if the order ID does not exist, the service method should throw a NotFoundException with the message "Order not found with id [orderId]".
- When canceling an order, if the order ID does not exist, the service method should throw a NotFoundException with the message "Order not found with id [orderId]".

Common Constraints

- For all rest endpoints receiving @RequestBody, validation check must be done and must throw custom exceptions if data is invalid.
- All the business validations must be implemented in dto classes only.
- All the database operations must be implemented on entity object only.
- Do not change, add, remove any existing methods in the service layer.
- In Repository interfaces, custom methods can be added as per requirements.
- All RestEndpoint methods and Exception Handlers must return data wrapped in **ResponseEntity**.

3 Business Validations

3.1 User

- Id must be of type id.
- Username should not be blank and unique in the system.
- Password should not be blank.
- Email should not be blank, must be of type email and unique in the system.
- Firstname should not be blank.
- Lastname should not be blank.

3.2 Book

- Id must be of type id.
- Title should not be blank.
- Author should not be blank.
- Category IDs should not be null.
- Price should not be null.

3.3 Author

- Id must be of type id.
- Name should not be blank.

3.4 Category

- Id must be of type id.
- Name should not be blank.

3.5 Review

- Id must be of type id.
- User ID should not be null.
- Book ID should not be null.
- Rating should not be null and must be within the range of min 1 to max 5.

3.6 Order

- Id must be of type id.
- User ID should not be null.
- Status should not be null.
- Total Amount should not be null.
- Items should not be null.

3.7 OrderItem

- Id must be of type id.
- Book ID should not be null.
- Quantity should not be null.
- Price should not be null.

4 REST ENDPOINTS

Rest End-points to be exposed in the controller along with method details for the same to be created.

4.1 USERCONTROLLER

URL Exposed		Purpose
1. /api/users/profile/{userId}		Retrieves the profile of a specific user by their id
Http Method	GET	
Parameter 1	Long (userId)	
Return	UserDTO	
2. /api/users		Registers a new user into the system
Http Method	POST **The user data to be created must be received in the controller using @RequestBody.**	
Parameter	-	
Return	UserDTO	
3. /api/users/profile/{userId}		Updates the user profile information for a specified user id
Http Method	PUT **The user data to be updated must be received in the controller using @RequestBody.**	
Parameter 1	Long (userId)	
Return	UserDTO	

4.2 BOOKCONTROLLER

URL Exposed		Purpose
1. /api/books		Adds a new book to the system
Http Method	POST **The book data to be created must be received in the controller using**	

	@RequestBody.	
Parameter 1	-	
Return	BookDTO	
2. /api/books		
Http Method	GET	Retrieves a list of all books available in the system
Parameter 1	-	
Return	List<BookDTO>	
3. /api/books/{id}		
Http Method	GET	Retrieves details of a specific book by its id
Parameter 1	Long (id)	
Return	BookDTO	
4. /api/books/{id}		
Http Method	PUT **The book data to be updated must be received in the controller using @RequestBody.**	Updates details of an existing book
Parameter 1	Long (id)	
Return	BookDTO	
5. /api/books/{id}		
Http Method	DELETE	Deletes a specific book from the system
Parameter 1	Long (id)	
Return	-	
6. /api/books/search		
Http Method	GET	Searches for books based on a title or author
Request Parameter	String (query)	
Return	List<BookDTO>	

4.3 AUTHORCONTROLLER

URL Exposed		Purpose
1. /api/authors		Retrieves a list of all authors in the system
Http Method	GET	
Parameter 1	-	
Return	List<AuthorDTO>	
2. /api/authors		Adds a new author to the system
Http Method	POST **The author data to be created must be received in the controller using @RequestBody.**	
Parameter 1	-	
Return	AuthorDTO	

4.4 CATEGORYCONTROLLER

URL Exposed		Purpose
1. /api/categories		Retrieves a list of all categories in the system
Http Method	GET	
Parameter 1	-	
Return	List<CategoryDTO>	
2. /api/categories		Adds a new category to the system
Http Method	POST **The category data to be created must be received in the controller using @RequestBody.**	
Parameter 1	-	
Return	CategoryDTO	

4.5 REVIEWCONTROLLER

URL Exposed		Purpose
1. /api/reviews/book/{bookId}		Retrieves all reviews associated with a specific book id
Http Method	GET	
Parameter 1	Long (bookId)	

Return	List<ReviewDTO>	
2. /api/reviews		Adds a new review
Http Method	POST **The review data to be created must be received in the controller using @RequestBody.**	
Parameter	-	
Return	ReviewDTO	
3. /api/reviews/{id}		Updates an existing review by its id
Http Method	PUT **The review data to be updated must be received in the controller using @RequestBody.**	
Parameter 1	Long (id)	
Return	ReviewDTO	
4. /api/reviews/user/{userId}		Retrieves all reviews posted by a specific user
Http Method	GET	
Parameter 1	Long (userId)	
Return	List<ReviewDTO>	
5. /api/reviews/{id}		Deletes a review by its id
Http Method	DELETE	
Parameter 1	Long (id)	
Return	-	
6. /api/reviews		Retrieves all reviews from the system
Http Method	GET	
Parameter 1	-	
Return	List<ReviewDTO>	

4.6 ORDERCONTROLLER

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1. /api/orders/{id}<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Long (id)</td></tr><tr><td>Return</td><td>OrderDTO</td></tr></table></td><td>Retrieves details of a specific order by its id</td></tr>
<tr><td>2. /api/orders<table><tr><td>Http Method</td><td>POST

The order data to be created must be received in the controller using @RequestBody.</td></tr><tr><td>Parameter 1</td><td>-</td></tr><tr><td>Return</td><td>OrderDTO</td></tr></table></td><td>Places a new order</td></tr>
<tr><td>3. /api/orders/{id}/status<table><tr><td>Http Method</td><td>PUT</td></tr><tr><td>Parameter 1</td><td>Long (id)</td></tr><tr><td>Request Parameter</td><td>String (status)</td></tr><tr><td>Return</td><td>OrderDTO</td></tr></table></td><td>Updates the status of an existing order</td></tr>
<tr><td>4. /api/orders/{id}<table><tr><td>Http Method</td><td>DELETE</td></tr><tr><td>Parameter 1</td><td>Long (id)</td></tr><tr><td>Return</td><td>-</td></tr></table></td><td>Cancels an existing order by its id</td></tr>
<tr><td>5. /api/orders/user/{userId}<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Long (userId)</td></tr><tr><td>Return</td><td>List<OrderDTO></td></tr></table></td><td>Retrieves all orders associated with a specific user id</td></tr>
<tr><td>6. /api/orders/{id}/payment<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>Long (id)</td></tr><tr><td>Return</td><td>OrderDTO</td></tr></table></td><td>Processes payment for a specific order ID</td></tr>
</table>

5 Template Code Structure

5.1 Package: com.booknest

Resources

BookNestApplication (Class)	This is the Spring Boot starter class of the application.	Already Implemented

5.2 Package: com.booknest.repository

Resources

Class/Interface	Description	Status

UserRepository (interface)	• Repository interface exposing CRUD functionality for User Entity.
	• It must contain the methods for:
	  ○ Finding all users by their username.
	  ○ Finding all users by email.
	• You can go ahead and add any custom methods as per requirements.
		Partially implemented.

BookRepository (interface)	• Repository interface exposing CRUD functionality for Book Entity.
	• It must contain the methods for:
	  ○ Finding all books by title or author.
	  ○ Finding a book by book id.
	• You can go ahead and add any custom methods as per requirements.
		Partially implemented.

AuthorRepository (interface)	• Repository interface exposing CRUD functionality for Author Entity.
	• It must contain the methods for:
	  ○ Finding an author by name.
	  ○ Finding all authors.
	• You can go ahead and add any custom methods as per requirements.
		Partially implemented.

CategoryRepository (interface)	• Repository interface exposing CRUD functionality for Category Entity.
	• It must contain the methods for:
	  ○ Finding category by name.
	  ○ Finding all categories.
	• You can go ahead and add any custom methods as per requirements.
		Partially implemented.

ReviewRepository (interface)	• Repository interface exposing CRUD functionality for Review Entity.
	• It must contain the methods for:
	  ○ Finding all reviews by book id.
	  ○ Finding all reviews by user id.
	  ○ Finding all reviews.
	• You can go ahead and add any custom methods as per
		Partially implemented.

	requirements.	

OrderRepository (interface)	• Repository interface exposing CRUD functionality for Order Entity.
	• It must contain the methods for:
	  ○ Finding all orders by user id.
	  ○ Finding an order by order id.
	  ○ Finding all orders by status.
	• You can go ahead and add any custom methods as per requirements.
		Partially implemented.

5.3 PACKAGE: COM.BOOKNEST.SERVICE

Resources

Class/Interface	Description	Status
UserService (interface)	• Interface to expose method signatures for user related functionality. • Do not modify, add or delete any method.	Already implemented.
BookService (interface)	• Interface to expose method signatures for book related functionality. • Do not modify, add or delete any method.	Already implemented.
AuthorService (interface)	• Interface to expose method signatures for author related functionality. • Do not modify, add or delete any method.	Already implemented.

CategoryService (interface)	• Interface to expose method signatures for category related functionality. • Do not modify, add or delete any method.	Already implemented.
ReviewService (interface)	• Interface to expose method signatures for review related functionality. • Do not modify, add or delete any method.	Already implemented.
OrderService (interface)	• Interface to expose method signatures for order related functionality. • Do not modify, add or delete any method.	Already implemented.

5.4 PACKAGE: COM.BOOKNEST.SERVICE.IMPL

Class/Interface	Description	Status
UserServiceImpl (class)	• Implements UserService. • Contains template method implementation. • Need to provide implementation for user related functionalities. • Do not modify, add or delete any method signature	To be implemented.
BookServiceImpl (class)	• Implements BookService. • Contains template method implementation. • Need to provide implementation for book related functionalities. • Do not modify, add or delete any method signature	To be implemented.

AuthorServiceImpl (class)	• Implements AuthorService. • Contains template method implementation. • Need to provide implementation for author related functionalities. • Do not modify, add or delete any method signature	To be implemented.
CategoryServiceImpl (class)	• Implements CategoryService. • Contains template method implementation. • Need to provide implementation for category related functionalities. • Do not modify, add or delete any method signature	To be implemented.
ReviewServiceImpl (class)	• Implements ReviewService. • Contains template method implementation. • Need to provide implementation for review related functionalities. • Do not modify, add or delete any method signature	To be implemented.
OrderServiceImpl (class)	• Implements OrderService. • Contains template method implementation. • Need to provide implementation for order related functionalities. • Do not modify, add or delete any method signature	To be implemented.

5.5 PACKAGE: COM.BOOKNEST.CONTROLLER

Resources

Class/Interface	Description	Status
UserController (Class)	• Controller class to expose all rest-endpoints for user related activities. • May also contain local exception handler methods	To be implemented
BookController (Class)	• Controller class to expose all rest-endpoints for book related activities. • May also contain local exception handler methods	To be implemented
AuthorController (Class)	• Controller class to expose all rest-endpoints for author related activities. • May also contain local exception handler methods	To be implemented
CategoryController (Class)	• Controller class to expose all rest-endpoints for category related activities. • May also contain local exception handler methods	To be implemented

ReviewController (Class)	• Controller class to expose all rest-endpoints for review related activities. • May also contain local exception handler methods	To be implemented
OrderController (Class)	• Controller class to expose all rest-endpoints for order related activities. • May also contain local exception handler methods	To be implemented

5.6 PACKAGE: COM.BOOKNEST.DTO

Resources

Class/Interface	Description	Status
UserDTO (Class)	Use appropriate annotations for validating attributes of this class.	Partially implemented.
BookDTO (Class)	Use appropriate annotations for validating attributes of this class.	Partially implemented.
AuthorDTO (Class)	Use appropriate annotations for validating attributes of this class.	Partially implemented.
CategoryDTO (Class)	Use appropriate annotations for validating attributes of this class.	Partially implemented.
ReviewDTO (Class)	Use appropriate annotations for validating attributes of this class.	Partially implemented.
OrderDTO (Class)	Use appropriate annotations for validating attributes of this class.	Partially implemented.

OrderItemDTO (Class)	Use appropriate annotations for validating attributes of this class.	Partially implemented.

5.7 PACKAGE: COM.BOOKNEST.ENTITY

Resources

Class/Interface	Description	Status
User (Class)	• This class is partially implemented. • Annotate this class with proper annotation to declare it as an entity class with **id** as primary key. • Map this class with a **users table**. • Generate the **id** using the IDENTITY strategy.	Partially implemented.
Book (Class)	• This class is partially implemented. • Annotate this class with proper annotation to declare it as an entity class with **id** as primary key. • Map this class with a **books table**. • Generate the **id** using the IDENTITY strategy.	Partially implemented.

Author (Class)	• This class is partially implemented. • Annotate this class with proper annotation to declare it as an entity class with **id** as primary key. • Map this class with an **authors table**. • Generate the **id** using the IDENTITY strategy.	Partially implemented.
Category (Class)	• This class is partially implemented. • Annotate this class with proper annotation to declare it as an entity class with **id** as primary key. • Map this class with a **categories table**. • Generate the **id** using the IDENTITY strategy.	Partially implemented.
Review (Class)	• This class is partially implemented. • Annotate this class with proper annotation to declare it as an entity class with **id** as primary key. • Map this class with a **reviews table**. • Generate the **id** using the IDENTITY strategy.	Partially implemented.

Order (Class)	• This class is partially implemented. • Annotate this class with proper annotation to declare it as an entity class with **id** as primary key. • Map this class with an **orders table**. • Generate the **id** using the IDENTITY strategy.	Partially implemented.
OrderItem (Class)	• This class is partially implemented. • Annotate this class with proper annotation to declare it as an entity class with **id** as primary key. • Map this class with an **order_items table**. • Generate the **id** using the IDENTITY strategy.	Partially implemented.
OrderStatus (Class)	-	Already implemented.

5.8 PACKAGE: COM.BOOKNEST.EXCEPTION

Resources

Class/Interface	Description	Status
NotFoundException (Class)	● Custom Exception to be thrown when trying to fetch or delete the user / book / author / category / review / order info which does not exist. ● Need to create Exception Handler for the same wherever needed (local or global).	Already implemented.
ErrorResponse (Class)	● RestControllerAdvice Class for defining global exception handlers. ● Contains Exception Handler for **InvalidDataException** class. ● Use this as a reference for creating exception handler for other custom exception classes.	Already implemented.
RestExceptionHandler (Class)	● RestControllerAdvice Class for defining rest exception handlers. ● Contains Exception Handler for **NotFoundException** class. ● Use this as a reference for creating exception handler for other custom exception classes.	Already implemented.

6 EXECUTION STEPS TO FOLLOW FOR BACKEND

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**
2. **To open the command terminal the test takers need to go to the Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal.**
3. **cd into your backend project folder**
4. **To build your project use command:**

   mvn clean package -Dmaven.test.skip
5. **To launch your application, move into the target folder (cd target). Run the following command to run the application:**

   java -jar <your application jar file name>
6. **This editor Auto Saves the code.**
7. **If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B-command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.**
8. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**
9. **To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN. Please use 127.0.0.1 instead of localhost to test rest endpoints.**
10. **To test any UI based application the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.**
11. **Default credentials for MySQL:**
    a. **Username: root**
    b. **Password: pass@word1**

12. To login to mysql instance: Open new terminal and use following command:

    a. sudo systemctl enable mysql

    b. sudo systemctl start mysql

    NOTE: After typing any of the above commands you might encounter any warnings.

    >> Please note that this warning is expected and can be disregarded. Proceed to the next step.

    c. mysql -u root -p

    The last command will ask for password which is 'pass@word1'

13. Mandatory: Before final submission run the following command:

    mvn test

14. You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.