System Requirements Specification

Index

For

Shop Ease App

Version 1.0

TABLE OF CONTENTS

SHOP EASE APPLICATION
System Requirements Specification

BACKEND-SPRING BOOT RESTFUL APPLICATION

1 Project Abstract

The **Shop Ease Application** is implemented using Spring Boot with a MySQL database, designed to enhance a comprehensive e-commerce platform. This app ensures dynamic interactions and transactional data processing to facilitate a smooth shopping experience from browsing products to managing orders.

You are tasked with developing functionalities that enable seamless user registration, product management, and inventory control. The application should allow users to register, update their profiles, and track their orders. Additionally, it should support comprehensive product and inventory management systems where products can be added, updated, deleted, and reviewed. The Order Module will handle all aspects of order processing including creation, updates, cancellations, and returns, ensuring that each step is managed dynamically and supports transactional integrity to maintain accurate and real-time order and inventory statuses.

Following is the requirement specifications:

	Shop Ease Application
Modules	
1	User
2	Product
3	Inventory
4	Order
User Module Functionalities	
1	Register a user
2	Get user by id
3	Update an user by id
4	Delete an user by id
5	Get user orders

Product Module Functionalities	
1	Add a product
2	Get all products
3	Get product by id
4	Update a product
5	Delete a product

6	Get product reviews
7	Add review for a product
8	Search products

Inventory Module Functionalities	
1	Add an item to the inventory
2	Get all inventory items
3	Get inventory item by id
4	Update inventory item
5	Delete inventory item
6	Get inventory status of products
7	Update inventory details for a product

Order Module Functionalities	
1	Create a new order
2	Get all orders
3	Get order by id
4	Update order
5	Cancel order
6	Get order status
7	Update the status of an order
8	Initiate a return process for an order

2 Assumptions, Dependencies, Risks / Constraints

2.1 User Constraints

- When fetching a user by ID, if the user ID does not exist, the service method should throw a NotFoundException with the message "User not found for this id : [id]".
- When updating a user's details, if the user ID does not exist, the service method should throw a NotFoundException with the message "User not found for this id : [id]".
- When deleting a user, if the user ID does not exist, the service method should throw a NotFoundException with the message "User not found for this id : [id]".

2.2 Product Constraints

- When fetching a product by ID, if the product ID does not exist, the service method should throw a NotFoundException with the message "Product not found for this id : [id]".
- When updating a product's details, if the product ID does not exist, the service method should throw a NotFoundException with the message "Product not found for this id : [id]".

- When deleting a product, if the product ID does not exist, the service method should throw a NotFoundException with the message "Product not found for this id : [id]".
- When fetching reviews for a product, if the product ID does not exist, the service method should throw a NotFoundException with the message "Product not found for this id : [productId]".
- When adding a review to a product, if the product ID does not exist, the service method should throw a NotFoundException with the message "Product not found for this id : [productId]".

2.3 Inventory Constraints

- When fetching an inventory item by ID, if the inventory item ID does not exist, the service method should throw a NotFoundException with the message "Inventory item not found for this id : [id]".
- When updating an inventory item's details, if the inventory item ID does not exist, the service method should throw a NotFoundException with the message "Inventory item not found for this id : [id]".
- When deleting an inventory item, if the inventory item ID does not exist, the service method should throw a NotFoundException with the message "Inventory item not found for this id : [id]".
- When updating the inventory of a specific product, if the product ID does not exist, the service method should throw a NotFoundException with the message "Product not found for this id : [productId]."

2.4 Order Constraints

- When creating an order:
    1) If the user ID does not exist, the service method should throw a NotFoundException with the message "User not found for this id : [id]".
    2) If the product ID does not exist, the service method should throw a NotFoundException with the message "Product not found for this id : [id]".
- When fetching an order, if the order ID does not exist, the service method should throw a NotFoundException with the message "Order not found for this id : [id]".
- When updating an order, if the order ID does not exist, the service method should throw a NotFoundException with the message "Order not found for this id : [id]".
- When canceling an order, if the order ID does not exist, the service method should throw a NotFoundException with the message "Order not found for this id : [id]".
- When fetching 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 for this id : [id]".
- 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 for this id : [id]".
- When initiating the return, if the order ID does not exist, the service method should throw a NotFoundException with the message "Order not found for this id : [id]".

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, min 3 and max 50 characters and unique in the system.
- Password should not be blank and must be at least 6 characters long.
- Email should not be blank and must be of type email.
- Firstname should not be blank.
- Lastname should not be blank.

3.2 Product

- Id must be of type id.
- Name should not be blank and size can be of max 100 characters.
- Description size can be of max 2000 characters.
- Price should not be null.

3.3 Inventory

- Id must be of type id.
- Location should not be blank.
- StockQuantity should not be null.

3.4 Order

- Id must be of type id.
- User should not be null.
- Product should not be null.
- Orderdate should not be null.
- Status should not be blank.

3.5 Review

- Id must be of type id.
- Comment should not be blank.
- Rating should not be null and min 1 and max 5 should be allowed.
- Product should not be null.
- User 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/{id}		Retrieves details of a user by their ID
Http Method	GET	
Parameter 1	Long (id)	
Return	UserDTO	
2. /api/users/register		Register a new user
Http Method	POST **The user data to be created must be received in the controller using @RequestBody.**	
Parameter	-	
Return	UserDTO	
3. /api/users/{id}		Updates the details of an existing user by their id
Http Method	PUT **The user data to be updated must be received in the controller using @RequestBody.**	
Parameter 1	Long (id)	
Return	UserDTO	
4. /api/users/{id}		Deletes a user from the system by their ID
Http Method	DELETE	
Parameter 1	Long (id)	
Return	-	
5. /api/users/{id}/orders		Retrieves all orders placed by a specific user
Http Method	GET	
Parameter 1	Long (id)	
Return	List<OrderDTO>	

4.2 ProductController

URL Exposed		Purpose
1. /api/products		Adds a new product to the catalog
Http Method	POST **The product data to be created must be received in the controller using @RequestBody.**	
Parameter 1	-	
Return	ProductDTO	
2. /api/products		Retrieves a list of all products in the catalog
Http Method	GET	
Parameter 1	-	
Return	List<ProductDTO>	
3. /api/products/{id}		Retrieves details of a specific product by its ID
Http Method	GET	
Parameter 1	Long (id)	
Return	ProductDTO	
4. /api/products/{id}		Updates details of a specific product
Http Method	PUT **The product data to be updated must be received in the controller using @RequestBody.**	
Parameter 1	Long (id)	
Return	ProductDTO	
5. /api/products/{id}		Deletes a specific product
Http Method	DELETE	
Parameter 1	Long (id)	
Return	-	
6. /api/products/{id}/reviews		Retrieves all reviews associated with a specific product
Http Method	GET	
Parameter 1	Long (id)	
Return	List<ReviewDTO>	

7. /api/products/{id}/reviews		Adds a review to a specific product
Http Method	POST **The review data to be created must be received in the controller using @RequestBody.**	
Parameter 1	Long (id)	
Return	ReviewDTO	

8. /api/products/search		Searches products based on given name
Http Method	GET	
Request Parameter 1	String (criteria)	
Return	List<ProductDTO>	

4.3 InventoryController

URL Exposed		Purpose
1. /api/inventory		Retrieves all items in the inventory
Http Method	GET	
Parameter 1	-	
Return	List<InventoryDTO>	
2. /api/inventory		Adds a new item to the inventory
Http Method	POST **The inventory data to be created must be received in the controller using @RequestBody.**	
Parameter 1	-	
Return	InventoryDTO	
3. /api/inventory/{id}		Retrieves details of a specific inventory item by its ID
Http Method	GET	
Parameter 1	Long (id)	
Return	InventoryDTO	

4. /api/inventory/{id}		
Http Method	PUT	

	The inventory data to be updated must be received in the controller using @RequestBody.	Updates an existing inventory item
Parameter 1	Long (id)	
Return	InventoryDTO	
5. /api/inventory/{id}		Removes an inventory item
Http Method	DELETE	
Parameter 1	Long (id)	
Return	-	
6. /api/inventory/products		Retrieves inventory status for all products
Http Method	GET	
Parameter 1	-	
Return	List<ProductDTO>	
7. /api/inventory/products/{productId}		Updates the inventory details for a specific product
Http Method	PUT **The inventory data to be updated must be received in the controller using @RequestBody.**	
Parameter 1	Long (productId)	
Return	InventoryDTO	

4.4 OrderController

URL Exposed		Purpose
1. /api/orders		Retrieves a list of all orders
Http Method	GET	
Parameter 1	-	
Return	List<OrderDTO>	
2. /api/orders		Places a new order
Http Method	POST **The order data to be created must be received in the**	

	controller using @RequestBody.	
Parameter 1	-	
Return	OrderDTO	

3. /api/orders/{id}		
Http Method	PUT **The order data to be updated must be received in the controller using @RequestBody.**	Updates an existing order
Parameter 1	Long (id)	
Return	OrderDTO	

4. /api/orders/{id}		
Http Method	DELETE	Cancels an existing order
Parameter 1	Long (id)	
Return	-	

5. /api/orders/{id}		
Http Method	GET	Retrieves details of a specific order by its ID
Parameter 1	Long (id)	
Return	OrderDTO	

6. /api/orders/{id}/status		
Http Method	GET	Retrieves the status of a specific order
Parameter 1	Long (id)	
Return	String	

7. /api/orders/{id}/status		
Http Method	PUT	Updates the status of an existing order
Parameter 1	Long (id)	
Request Parameter	String (status)	
Return	Boolean	

8. /api/orders/{id}/return		
Http Method	POST	Initiates a return process for an order
Parameter 1	Long (id)	
Return	Boolean	

5 Template Code Structure

5.1 Package: com.shopease

Resources

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

5.2 Package: com.shopease.repository

Resources

Class/Interface	**Description**	**Status**
InventoryRepository (interface)	● Inventory interface exposing CRUD functionality for Inventory Entity. ● It must contain the methods for: ○ Finding all inventories by location. ○ Finding all low stock (a number should be passed in argument for comparing the quantity) inventory items. ● You can go ahead and add any custom methods as per requirements.	Partially implemented.
OrderRepository (interface)	● Repository interface exposing CRUD functionality for Order Entity. ● It must contain the methods for: ○ Find all orders by user id. ○ Find all orders by status. ● You can go ahead and add any	Partially implemented.

	custom methods as per requirements.	
ProductRepository (interface)	● Repository interface exposing CRUD functionality for Product Entity. ● It must contain the methods for: ○ Find all products by their name. ○ Find all products which are cheaper than passed value in argument. ● You can go ahead and add any custom methods as per requirements.	Partially implemented.
ReviewRepository (interface)	● Repository interface exposing CRUD functionality for Review Entity. ● You can go ahead and add any custom methods as per requirements.	Partially implemented.
UserRepository (interface)	● Repository interface exposing CRUD functionality for User Entity. ● It must contain the methods for: ○ Find all users by their username. ○ Find all users by email. ● You can go ahead and add any custom methods as per requirements.	Partially implemented.

5.3 Package: com.shopease.service

Resources

Class/Interface	Description	Status
InventoryService (interface)	● Interface to expose method signatures for inventory 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.
ProductService (interface)	● Interface to expose method signatures for product related functionality. ● Do not modify, add or delete any method.	Already implemented.
UserService (interface)	● Interface to expose method signatures for user related functionality. ● Do not modify, add or delete any method.	Already implemented.

5.4 Package: com.shopease.service.impl

Class/Interface	Description	Status
InventoryServiceImpl (class)	● Implements InventoryService. ● Contains template method implementation. ● Need to provide implementation for inventory 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.
ProductServiceImpl (class)	● Implements ProductService. ● Contains template method implementation. ● Need to provide implementation for product related functionalities. ● Do not modify, add or delete any method signature	To be implemented.
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.

5.5 Package: com.shopease.controller

Resources

Class/Interface	**Description**	**Status**
InventoryController (Class)	● Controller class to expose all rest-endpoints for inventory 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
ProductController (Class)	● Controller class to expose all rest-endpoints for product related activities. ● May also contain local exception handler methods	To be implemented
UserController (Class)	● Controller class to expose all rest-endpoints for user related activities. ● May also contain local exception handler methods	To be implemented

5.6 PACKAGE: COM.SHOPEASE.DTO

Resources

Class/Interface	**Description**	**Status**
InventoryDTO (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.
ProductDTO (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.

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

5.7 Package: com.shopease.entity

Resources

Class/Interface	**Description**	**Status**
Inventory (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 **inventory 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 a **orders table**. ● Generate the **id** using the IDENTITY strategy.	Partially implemented.

Product (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 **products 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.
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.

5.8 Package: com.shopease.exception

Resources

Class/Interface	**Description**	**Status**
NotFoundException (Class)	● Custom Exception to be thrown when trying to fetch or delete the inventory/ order/product/review/user 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.