System Requirements Specification

Index

For

Inventory Management Application

Version 1.0

TABLE OF CONTENTS

INVENTORY MANAGEMENT APPLICATION

System Requirements Specification

BACKEND-SPRING BOOT RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **Inventory Management Application** is implemented using Spring Boot with a MySQL database. The application aims to provide a comprehensive platform for managing and organizing all products along with their sell records.

Following is the requirement specifications:

	Inventory Management Application
Modules	
1	Product
2	Sell
Product Module Functionalities	
1	List all products
2	Get product by id
3	Create product
4	Update product by id
5	Delete product by id

Sell Module Functionalities	
1	List all sells
2	Get sell by id
3	Create sell
4	Get list of product sold in last 30 days

2 Assumptions, Dependencies, Risks / Constraints

2.1 PRODUCT CONSTRAINTS

- When fetching a Product by ID, if the product ID does not exist, the operation should throw a Product not found exception.
- When updating a Product, if the product ID does not exist, the operation should throw a Product not found exception.
- When removing a Product, if the product ID does not exist, the operation should throw a Product not found exception.

2.2 SELL CONSTRAINTS

- When fetching a Sell by ID, if the sell ID does not exist, the operation should throw a Sell not found exception.

Common Constraints

- For all rest endpoints receiving @RequestBody, validation check must be done and must throw custom exception 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 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 - Product

- Name is not blank, min 3 and max 255 characters.
- Price should not be null and should accept decimal values up-to 2 digits.

4 Business Validations - Sell

5 ProductId is not null.

6 Quantity should not be null, should not exceed 1000.

7 OrderDate should be past or present only.

8 REST ENDPOINTS

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

8.1 PRODUCTCONTROLLER

URL Exposed		Purpose
1. /api/products		
Http Method	GET	Fetches all the products
Parameter	-	
Return	List<Product>	
2. /api/products/{id}		
Http Method	GET	Get a product by id
Parameter 1	Long (id)	
Return	Product	
3. /api/products		
Http Method	POST	Create a new product
Parameter	-	
Return	Product	
4. /api/products/{id}		
Http Method	PUT	Updates existing product by id
Parameter 1	Long (id)	
Return	Product	
5. /api/products/{id}		
Http Method	DELETE	Deletes a product by id
Parameter 1	Long (id)	
Return	-	

8.2 SELLCONTROLLER

URL Exposed		Purpose
1. /api/sells		
Http Method	GET	Fetches all the sells

Parameter	-	
Return	List<Sell>	
2. /api/sells/{id}		Get a sell by id
Http Method	GET	
Parameter 1	Long (id)	
Return	Sell	
3. /api/sells		Create a new sell
Http Method	POST	
Parameter	-	
Return	Sell	
4. /api/sells/products-sold-last-month		Fetches the list of all products with name and quantity sold in last 30 days
Http Method	GET	
Parameter	-	
Return	List<Sell>	

9 TEMPLATE CODE STRUCTURE

9.1 PACKAGE: COM.LAPTOPSTORE

Resources

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

9.2 PACKAGE: COM.INVENTORYMANAGEMENT.REPOSITORY

Resources

Class/Interface	Description	Status

ProductRepository (interface)	• Repository interface exposing CRUD functionality for Product Entity. • You can go ahead and add any custom methods as per requirements.	Partially implemented.
SellRepository (interface)	• Repository interface exposing CRUD functionality for Sell Entity. • You can go ahead and add any custom methods as per requirements.	Partially implemented.

9.3 PACKAGE: COM.INVENTORYMANAGEMENT.SERVICE

Resources

Class/Interface	Description	Status
ProductService (interface)	• Interface to expose method signatures for product related functionality. • Do not modify, add or delete any method.	Already implemented.
SellService (interface)	• Interface to expose method signatures for sell related functionality. • Do not modify, add or delete any method.	Already implemented.

9.4 PACKAGE: COM.INVENTORYMANAGEMENT.SERVICE.IMPL

Class/Interface	Description	Status
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.
SellServiceImpl (class)	• Implements SellService. • Contains template method implementation. • Need to provide implementation for sell related functionalities. • Do not modify, add or delete any method signature	To be implemented.

9.5 PACKAGE: COM.INVENTORYMANAGEMENT.CONTROLLER

Resources

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

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

9.6 PACKAGE: COM.INVENTORYMANAGEMENT.DTO

Resources

Class/Interface	Description	Status
ProductDTO (Class)	Use appropriate annotations from the Java Bean Validation API for validating attributes of this class.	Partially implemented.
SellDTO (Class)	Use appropriate annotations from the Java Bean Validation API for validating attributes of this class.	Partially implemented.

9.7 PACKAGE: COM.INVENTORYMANAGEMENT.ENTITY

Resources

Class/Interface	Description	Status
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 **product table**. • Generate the **id** using the IDENTITY strategy	Partially implemented.
Sell (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 **sell table**. • Generate the **id** using the IDENTITY strategy	Partially implemented.

9.8 PACKAGE: COM.INVENTORYMANAGEMENT.EXCEPTION

Resources

Class/Interface	Description	Status
NotFoundException (Class)	• Custom Exception to be thrown when trying to fetch or delete the product/sell info which does not exist. • Need to create Exception Handler for same wherever needed (local or global)	Already implemented.

1 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.
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
11. To login to mysql instance: Open new terminal and use following command:
    a. sudo systemctl enable mysql
    b. sudo systemctl start mysql
    c. mysql -u root -p

    The last command will ask for password which is 'pass@word1'
12. Mandatory: Before final submission run the following command:

    mvn test
13. 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.