System Requirements Specification

Index

For

Laptop Store Application

Version 1.0

TABLE OF CONTENTS

LAPTOP STORE APPLICATION
System Requirements Specification

BACKEND-SPRING DATA RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **Laptop Store Application** is implemented using Spring Data with a MySQL database. The application aims to provide a comprehensive platform for finding and exploring all laptops across different configurations.

Following is the requirement specifications:

	Laptop Store Application
Modules	
1	Laptop
Laptop Module Functionalities	
1	List all laptops **(must return all laptops by brand name in ascending order and that also in pages)**
2	Get laptop by id
3	Create laptop **(must be transactional)**
4	Update laptop by id **(must be transactional)**
5	Delete laptop by id
6	Search laptop by name **(must use dynamic method)**
7	Search laptop by price **(must use dynamic method)**
8	Search laptop by brand **(must use custom query)**
9	Search laptop by name, price and brand **(must use custom query)**

2 Assumptions, Dependencies, Risks / Constraints

2.1 LAPTOP constraints

- When fetching a laptop by ID, if the laptop ID does not exist, the service method should throw a ResourceNotFoundException with “Laptop not found.” message.
- When updating a laptop, if the laptop ID does not exist, the service method should throw a ResourceNotFoundException with “Laptop not found.” message.

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

Laptop

- Name must not be null, must not be blank and of minimum 3 characters.
- Price must not be null & must be a positive number.
- Brand must not be null, must not be blank and of minimum 3 characters.
- Storage must not be null, must not be blank and of minimum 3 characters.
- Ram must not be null & must not be blank.
- Processor must not be null, must not be blank and of minimum 3 characters.

4 Rest Endpoints

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

4.1 LaptopController

URL Exposed		Purpose
1. /api/laptops		Fetches all the laptops
Http Method	GET	
Parameter	-	

Return	Page<LaptopDTO>	
2. /api/laptops/{id}		Get laptop by id
Http Method	GET	
Path variable	Long (id)	
Return	LaptopDTO	
3. /api/laptops		Create a new laptop
Http Method	POST **The laptop data to be created must be received in the controller using @RequestBody.**	
Parameter	-	
Return	LaptopDTO	
4. /api/laptops/{id}		Update existing laptop by id
Http Method	PUT **The laptop data to be updated must be received in the controller using @RequestBody.**	
Path variable	Long (id)	
Return	LaptopDTO	

5. /api/laptops/{id}		Deletes a laptop by id
Http Method	DELETE	
Path variable	Long (id)	
Return	LaptopDTO	

6. /api/laptops/search/by-price		Searches a laptop by price
Http Method	GET	
Request Parameter 1	Double (price)	
Return	List<LaptopDTO>	
7. /api/laptops/search/by-name		Searches a laptop by name
Http Method	GET	
Request Parameter 1	String (name)	
Return	List<LaptopDTO>	
8. /api/laptops/search/by-brand		

Http Method	GET	Searches a laptop by brand
Request Parameter 1	String (brand)	
Return	List<LaptopDTO>	

9. /api/laptops/search

Http Method	GET	Searches a laptop by name, price and brand
Request Parameter 1	String (name)	
Request Parameter 2	Double (price)	
Request Parameter 3	String (brand)	
Return	List<LaptopDTO>	

5 TEMPLATE CODE STRUCTURE

5.1 PACKAGE: COM.LAPTOPSTORE

Resources

LaptopApplication(Class)	This is the Spring Boot starterclass of the application.	Already Implemented

5.2 PACKAGE: COM.LAPTOPSTORE.REPO

Resources

Class/Interface	**Description**	**Status**
LaptopRepository (interface)	● Repository interface exposing CRUD functionality for Laptop Entity. ● You can go ahead and add any custom methods as per requirements. ● It must contain the methods for: ○ finding all laptops by name. ○ finding all laptops by price. ○ Search laptop by brand. ○ Search laptops by different criteria like	Partially implemented.

	name, price and brand	

5.3 PACKAGE: COM. LAPTOPSTORE.SERVICE

Resources

Class/Interface	Description	Status
LaptopService (interface)	● Interface to expose method signatures for laptop related functionality. ● Do not modify, add or delete any method.	Already implemented.

5.4 PACKAGE: COM. LAPTOPSTORE.SERVICE.IMPL

Class/Interface	Description	Status
LaptopServiceImpl (class)	● Implements LaptopService. ● Contains template method implementation. ● Need to provide implementation for laptop related functionalities. ● Do not modify or delete any method signature	To be implemented.

5.5 PACKAGE: COM. LAPTOPSTORE.CONTROLLER

Resources

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

5.6 PACKAGE: COM. LAPTOPSTORE.DTO

Resources

Class/Interface	Description	Status
LaptopDTO (Class)	Use appropriate annotations for validating attributes of this class.	Partially implemented.

5.7 PACKAGE: COM. LAPTOPSTORE.ENTITY

Resources

Class/Interface	Description	Status
Laptop (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 **laptops table**. ● Generate the **id** using the IDENTITY strategy	Partially implemented.

5.8 PACKAGE: COM. LAPTOPSTORE.EXCEPTION

Resources

Class/Interface	Description	Status
ResourceNotFoundException (Class)	● Custom Exception to be thrown when trying to fetch or update the entity info which does not exist. ● Need to create Exception Handler for same wherever needed (local or global)	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**

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.