System Requirements Specification Index

For

Online Auction System

Version 1.0

Table of Contents

Online Auction APPLICATION

System Requirements Specification

1 Project Abstract

Online Auction System Application is Spring boot RESTful application with MySQL, where it allows the sellers to Manage Products, Customers can place a bid on the products before the last date of the bidding.

Following is the requirement specifications:

	Online Auction System
Modules	
1	Seller
2	Customer
Seller Module Functionalities	
1	Register Itself
2	Can add a new product based on predefined categories
3	Can delete a product
4	Get Seller by id
5	Fetch all registered sellers
6	Delete an existing Seller
7	Can View details of bids placed on a particular product
8	Can view list of all products added for selling
Customer Module Functionalities	
1	Customer can register itself
2	Customer can update its information
3	Get customer by Id
4	Fetch all registered customers
5	Get All the Products
6	Get the product by id
7	Can view all product placed for bidding based on category
8	Customer can Place a bid on specific product
9	Customer can view the all bids placed on a product (only after last date)

2 Assumptions, Dependencies, Risks / Constraints

2.1 Seller Constraints:

- While deleting the seller details, if sellerId does not exist then the operation should throw a custom exception.
- While fetching the Seller details by id, if sellerId does not exist then the operation should throw a custom exception.
- While fetching the Product details by id, if productId does not exist then the operation should throw a custom exception.
- While deleting the Product details, if productId does not exist then operation should throw custom exception

2.2 Customer Constraints

- While deleting a customer, if the id does not exist then the operation should throw a custom exception.
- While fetching the customer details by id, if id does not exist then the operation should throw a custom exception.
- While placing a bid if customer , if id does not exist then operation should throw custom exception.

2.3 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

- Seller name is not null, min 3 and max 100 characters.
- Seller email is not null, min 3, max 100 characters and should be email format
- Seller address is not null, min 3 and max 100 characters.
- Seller phone number is not null, min 10 and max 10 digits only
- Product name is not null, min 3 and max 100 characters.
- Product description is not null, min 3 and max 100 characters.
- Product quantity is not null.
- Product start bidding amount is not null.
- Product price is not null
- Product last date of bidding is not null, it should be in 'yyyy-mm-dd' format and future date
- Product category is not null, min 3 and max 100 characters
- Product predefined categories should be [Mobiles, Electronics, Clothing, Home]
- Customer username is not null, min 3 and max 100 characters
- Customer password is not null, min 3 and max 100 characters
- Customer email is not null, min 3, max 100 characters and should be email format
- Customer phone number is not null, min 10 and max 10 digits only
- Customer address is not null, min 3 and max 100 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 SellerController

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1. /sellers/register<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>SellerDto</td></tr><tr><td>Return</td><td>SellerDto</td></tr></table></td><td>Register a seller</td></tr>
<tr><td>/sellers/update<table><tr><td>Http Method</td><td>PUT</td></tr><tr><td>Parameter 1</td><td>SellerDto</td></tr><tr><td>Return</td><td>SellerDto</td></tr></table></td><td>Update a Seller</td></tr>
<tr><td>/sellers/get/all<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>-</td></tr><tr><td>Return</td><td>List<SellerDto></td></tr></table></td><td>Fetches the list of all registered Sellers</td></tr>
<tr><td>/sellers/get/{sellerId}<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Long (sellerId)</td></tr><tr><td>Return</td><td>SellerDto</td></tr></table></td><td>Fetches the details of a Seller</td></tr>
<tr><td>/sellers/delete/{sellerId}<table><tr><td>Http Method</td><td>DELETE</td></tr><tr><td>Parameter 1</td><td>Long (sellerId)</td></tr><tr><td>Return</td><td>Boolean</td></tr></table></td><td>Delete a seller</td></tr>
<tr><td>/sellers/get/bids-on-product/{produtId}<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Long (productId)</td></tr><tr><td>Return</td><td>List<BidsDto></td></tr></table></td><td>Get Bids on a Products</td></tr>
</table>

4.2 ProductController

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>/products/register<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>ProductDto</td></tr><tr><td>Return</td><td>ProductDto</td></tr></table></td><td>Register a Product</td></tr>
</table>

/products/update Http Method: PUT Parameter 1: ProductDto Return: ProductDto	Update the Product
/products/get/all Http Method: GET Parameter 1: - Return: List<ProductDto >	Fetches all saved Products
/products/get/{productId} Http Method: GET Parameter 1: Long (productId) Return: ProductDto	Fetch the details of a Product
/products/get/by-seller/{sellerId} Http Method: GET Parameter 1: Long (sellerId) Return: List<ProductDto >	Fetches the details of all the Products registered by a seller
/products/delete/{productId} Http Method: DELETE Parameter 1: Long (productId) Return: Boolean	Delete a Product
/products/get/by-category/{categoryId} Http Method: GET Parameter 1: Long (categoryId) Return: List<ProductDto >	Fetch the details of all the products registered under a category

4.3 CustomerController

URL Exposed	Purpose
/customers/register Http Method: POST Parameter 1: CustomerDto Return: CustomerDto	Register a Customer
/customers/update Http Method: PUT Parameter 1: CustomerDto Return: CustomerDto	Update an existing Customer
/customers/get/all	Fetches all the registered customers

Http Method	GET	
Parameter 1	-	
Return	List<CustomerDto >	
/customers/get/{id}		Fetch the details of a Customer
Http Method	GET	
Parameter 1	Long(id)	
Return	List<CustomerDto>	
/customers/delete/{id }		Deletes an existing customer
Http Method	DELETE	
Parameter 1	Long(id)	
Return	Boolean	
/customers/place-bid		Places a bid on the product by the customer
Http Method	POST	
Parameter 1	BidsDto	
Return	BidsDto	
/customers/get/all-bids-on-product/{productId}		Customer can get all the bids on a product after the bid ends.
Http Method	GET	
Parameter 1	Long(productid)	
Return	List<BidsDto>	

5 Template Code Structure

5.1 Package: com.niit.training.auction

Resources

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

5.2 Package: com.niit.training.auction.entity

Resources

Class/Interface	Description	Status
SellerEntity (class)	o Annotate this class with proper annotation to declare it as an entity class with **sellerId** as primary key.	Partially implemented.

	o Map this class with **sellers** table. o Generate the **sellerId** using **IDENTITY** strategy	
ProductEntity(class)	o This class is partially implemented. o Annotate this class with proper annotation to declare it as an entity class with **productId** as primary key. o Map this class with **products** table. o Generate the **productId** using the **IDENTITY** strategy	Partially implemented.
CustomerEntity(class)	o This class is partially implemented. o Annotate this class with proper annotation to declare it as an entity class with **id** as primary key. o Map this class with **customers** table. o Generate the **id** using the **IDENTITY** strategy	Partially implemented.
BidsEntity(class)	o This class is partially implemented. o Annotate this class with proper annotation to declare it as an entity class with **id** as primary key. o Map this class with **bids** table. o Generate the **id** using the **IDENTITY** strategy o	Partially implemented.

5.3 PACKAGE: COM.NIIT.TRAINING.AUCTION.DTO

Resources

Class/Interface	Description	Status

SellerDto (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attributes of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.
ProductDto (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attributes of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.
CustomerDto (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attributes of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.
BidsDto (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attributes of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.

5.4 PACKAGE: COM.NIIT.TRAINING.AUCTION.MODEL.EXCEPTION

Resources

Class/Interface	Description	Status
ExceptionResponse (class)	Object of this class is supposed to be returned in case of exception through exception handlers	Already implemented.

5.5 Package: com.niit.training.auction.repository

Resources

Class/Interface	Description	Status
SellerRepository (interface)	1. Repository interface exposing CRUD functionality for **SellerEntity** Entity. 2. You can go ahead and add any custom methods as per requirements	Partially implemented
ProductRepository (interface)	1. Repository interface exposing CRUD functionality for **ProductEntity** Entity. 2. You can go ahead and add any custom methods as per requirements	Partially implemented
CustomerRepository (interface)	1. Repository interface exposing CRUD functionality for **Customer** Entity. 2. You can go ahead and add any custom methods as per requirements	Partially implemented
BidsRepository (interface)	1. Repository interface exposing Bids functionality for **Bids** Entity. 2. You can go ahead and add any custom methods as per requirements	Partially implemented

5.6 Package: com.niit.training.auction.service

Resources

Class/Interface	Description	Status

SellerService (interface)	Interface to expose method signatures for political party related functionality. Do not modify, add or delete any method	Already implemented.
ProductService (interface)	Interface to expose method signatures for political leader related functionality. Do not modify, add or delete any method	Already implemented.
CustomerService (interface)	Interface to expose method signatures for Developments related functionality. Do not modify, add or delete any method	Already implemented.
BidsService (interface)	Interface to expose method signatures for Developments related functionality. Do not modify, add or delete any method	Already implemented.

5.7 PACKAGE: COM.NIIT.TRAINING.AUCTION.SERVICE.IMPL

Resources

Class/Interface	Description	Status
SellerServiceImpl (class)	• Implements **SellerService**. Contains template method implementation.	To be implemented.

	• Need to provide implementation for seller related functionalities • Add required repository dependency • Do not modify, add or delete any method signature	
ProductServiceImpl (class)	• Implements **ProductService**. Contains template method implementation. • Need to provide implementation for product related functionalities • Add required repository dependency • Do not modify, add or delete any method signature	To be implemented.
CustomerServiceImpl (class)	• Implements **CustomerService**. Contains template method implementation. • Need to provide implementation for Customer related functionalities • Add required repository dependency • Do not modify, add or delete any method signature	To be implemented.
BidsServiceImpl (class)	• Implements **BidsService**. Contains template method implementation. • Need to provide implementation for **Bids** related functionalities	To be implemented.

	• Add required repository dependency • Do not modify, add or delete any method signature	

5.8 Package: com.niit.training.auction.exception

Resources

Class/Interface	Description	Status
GlobalHandler (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	Partially implemented.

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

	fetch or delete Product info which does not exist. ● Need to create Exception Handler for same wherever needed (local or global)	
CustomerNotFoundException (Class)	● Custom Exception to be thrown when trying to fetch or delete a Customer info which does not exist. ● Need to create Exception Handler for same wherever needed (local or global)	Already created.

5.9 PACKAGE: COM.NIIT.TRAINING.AUCTION.CONTROLLER

Resources

Class/Interface	Description	Status
SellerController (Class)	● Controller class to expose all rest-endpoints for Seller 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

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

6 CONSIDERATIONS

A. There are 2 roles in this application

Seller
Customer

B. You can perform the following 4 possible actions

Seller Actions
Product Actions
Customer Actions
Bids on Products

7 Execution Steps to Follow

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 Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal.
3. To build your project and run test cases use command:
   mvn clean package
4. To launch your application, move into the target folder (cd target). Run the following command to run the application:
   java -jar online-auction-system-0.0.1-SNAPSHOT.jar
5. This editor Auto Saves the code.
6. 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.
7. 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.
8. 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.
9. This is a web-based application, to run the application on a browser, use the internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.

   Note: The application will not run in the local browser
10. 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.**