System Requirements Specification Index

For

Online-Auction-System

Version 4.0

IIHT Pvt. Ltd.
IIHT Ltd, No: 15, 2nd Floor, Sri Lakshmi Complex, Off MG Road, Near SBI LHO,
Bangalore, Karnataka – 560001, India
fullstack@iiht.com

ONLINE-AUCTION SYSTEM
System Requirements Specification

1. Project Abstract

1.1 Online Auction System Application is .Net Core 3.1 RESTful API application with MS SQL Server , where it allows the sellers to Manage Products, Customers can place a bid on the products before the last date of the bidding.

1.2 Following is the requirement specifications:

	Online Auction System
USERS	
1	Product
2	Customer
Product Module Functionalities	
	1.Create a Product
	2.Update the Product
	3.Can delete a product
	4.Get Product by id
	5.Fetch all products
	6.Get the Products based on the category
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

2. Assumptions, Dependencies, Risks / Constraints

2.1 Product Constraints

- While deleting the Product details, if productId 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 deleting the Customer details, 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 model 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.**

- **All business logic CRUD operations under repository class and write your business logic validation in Services class and related validation use proper user defined exceptions mentioned in above document.**
- **Controller must validate before processing any logic on the database.**

3. Business Validations

3.1 Customer Class Entities

- Customer Id (long) is not null, key attribute.
- Username (string) is not null, min 3 and max 100 characters.
- Password (string) is not null, min 3 and max 100 characters.
- Email (string) is not null, min 3, max 100 characters and should be email format.
- Phone number (string) is not null, min 10 and max 10 digits only.
- Address (string) is not null, min 3 and max 100 characters.
- IsDeleted (bool).

3.2 Product Class Entities.

- ProductId (long) is not null, key attribute
- Product name (string) is not null, min 3 and max 100 characters.
- Product description (string) is not null, min 3 and max 100 characters.
- Product quantity (int) is not null.
- Product start bidding amount (decimal) is not null.
- Product price (decimal) is not null
- Product last date of bidding (datetime) is not null, it should be in 'yyyy-mm-dd' format and future date
- Product category (int) is not null, min 3 and max 100 characters
- Product predefined categories should be [Mobiles, Electronics, Clothing, Home]

4. Considerations

For Role of application users 2 possible values must be used: -

- Product
- Customer

For Category of product following 4 possible Enum values must be used.

• Mobile= 1,
• Electronics= 2,
• Home= 3,
• Clothing=4,

5. Rest Endpoints

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

5.1 ProductController

URL Exposed		Purpose
/products/register		Register a new product.
Http Method	Post	
Parameter 1	Product product	
Return	`HttpResponse status code`	
/products/update		Update product by productId.
Http Method	PUT	
Parameter 1	RegisterProductViewModel model	
Return	`HttpResponse status code`	
/products/delete/{productId}		Delete a Product.
Http Method	DELETE	
Parameter 1	productId	
Return	HttpResponse status code	
/products/get/{productid}		Fetch the details of a Product.
Http Method	GET	
Parameter 1	productId	
Return	`<Product>`	
/products/get/by-category/{categoryid}		Fetch the details of all the products registered under a category.
Http Method	GET	
Parameter 1	categoryId	
Return	`<IEnumerable<Product>>`	
/products/get/all		Fetches all saved Products.
Http Method	GET	
Parameter 1	-	
Return	<IEnumerable<Product>>	

5.2 CustomerController

URL Exposed		Purpose
/customers/register		Register a new Customer.
Http Method	POST	
Parameter 1	Customer customer	
Return	HttpResponse status code	
/customers/update		Update a customer by customerId.
Http Method	PUT	
Parameter 1	RegisterCustomerViewModel model	
Return	HttpResponse status code	
/customers/delete/{customerId}		Delete a customer by customerId.
Http Method	Delete	
Parameter 1	customerId	
Return	HttpResponse status code	
/customers/get/{customerId}		Fetch customer details by customerId.
Http Method	GET	
Parameter 1	customerId	
Return	<IEnumerable<Customer>>	
/customers/get/all		Fetch all registered customers.
Http Method	GET	
Parameter 1	customerId	
Return	<IEnumerable<Customer>>	

6. Template Code Structure

6.1 Package: Online-Auction

Resources

Names	Resource	Remarks	Status
Package Structure			
controller	Product Controller Customer Controller	Controller class to expose all rest-endpoints for auction related activities.	Partially implemented
Startup.cs	Startup CS file	Contain all Services settings and SQL server Configuration.	Already Implemented
Properties	launchSettings.json file	All URL Setting for API	Already Implemented
	appsettings.json file	Contain connection string for database	Already Implemented

6.2 Package: Online-Auction.BusinessLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
Interface	IProductServices interface ICustomerServices interface	Inside all these interface files contains all business validation logic functions.	Already Implemented
Service	Product Services CS file Customer Services CS file	Using this all class we are calling the Repository method and use it in the program and on the controller.	Partially Implemented
Repository	IProductRepository, Product Repository,ICustomerRepository,Customer Repository CS file and interface.	All these interfaces and class files contain all CRUD operation code for the database. Need to provide implementation for service related functionalities	Partially Implemented

ViewModels	RegisterCustomerViewModel, RegisterProductViewModel.	Contain all view Domain entities for show and bind data. All the business validations must be implemented.	Partially Implemented

6.3 Package: Online-Auction.DataLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
DataLayer	OnlineAuctionDBContext cs file	All database Connection and collection setting class	Already Implemented

6.4 Package: Online-Auction.Entities

Resources

Names	Resource	Remarks	Status
Package Structure			
Entities and Enum	Product, Customer CS file and Category Enum file	All Entities/Domain attribute are used for pass the data in controller and check the various constants using enum. Annotate this class with proper annotation to declare it as an entity class with **Id** as primary key. Generate the **Id** using the **IDENTITY** strategy	Partially Implemented

6.5 Package: Online-Auction.Tests

Resources

The Online-Auction.Tests project contains all test case classes and functions for code evaluation. Don't edit or change anything inside this project.

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 the Application menu (Three horizontal lines at left top) Terminal → New Terminal.

3. On command prompt, cd into your project folder (cd <Your-Project-folder>).

4. To connect SQL server from terminal:
   (Online-Auction /sqlcmd -S localhost -U sa -P pass@word1)
   - To create database from terminal -

     ```
     1>  Create Database OnlineAuction_Db
     2>  Go
     ```

5. Steps to Apply Migration(Code first approach):
   - Press Ctrl+C to get back to command prompt
   - Run following command to apply migration-
     (Online-Auction /dotnet-ef database update)

6. To check whether migrations are applied from terminal:
   (Online-Auction /sqlcmd -S localhost -U sa -P pass@word1)

   ```
   1> Use OnlineAuction_Db
   2> Go
   1> Select * From __EFMigrationsHistory
   2> Go
   ```

7. To build your project use command:
   (Online-Auction /dotnet build)

8. To launch your application, Run the following command to run the application:
   (Online-Auction /dotnet run)

9. This editor Auto Saves the code.

10. 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.

11. **To test web-based applications 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

12. **To run the test cases in CMD, Run the following command to test the application: (Online-Auction /dotnet test --logger "console;verbosity=detailed")**
    (You can run this command multiple times to identify the test case status,and refactor code to make maximum test cases passed before final submission)

13. **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.**

14. **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.**

15. **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.**