# System Requirements Specification Index

### For

# Online-Auction-System

### Version 4.0

# ONLINE-AUCTION SYSTEM
## System Requirements Specification

---

## 1. PROJECT ABSTRACT

**Online Auction System** Application is .Net Core 3.1 RESTful API application with InMemoryDB, 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 |
|---|---|---|
| | | |
| 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 |
|---|---|---|
| api/product/products/register | | Register a new product. |
| Http Method | Post | |
| Parameter 1 | Product product | |
| Return | HttpResponse status code | |
| api/product/products/update | | Update product by productId. |
| Http Method | PUT | |
| Parameter 1 | RegisterProductViewModel model | |
| Return | HttpResponse status code | |
| api/product/products/delete/{productId} | | Delete a Product. |
| Http Method | DELETE | |
| Parameter 1 | productId | |
| Return | HttpResponse status code | |
| api/product/products/get/{productid} | | Fetch the details of a Product. |
| Http Method | GET | |
| Parameter 1 | productId | |
| Return | <Product> | |
| api/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>> | |
| api/product/products/get/all | | Fetches all saved Products. |
| Http Method | GET | |
| Parameter 1 | - | |
| Return | <IEnumerable<Product>> | |

## 5.2 CustomerController

| URL Exposed | | Purpose |
|---|---|---|
| api/customer/customers/register | | Register a new Customer. |
| Http Method | POST | |
| Parameter 1 | Customer customer | |
| Return | HttpResponse status code | |
| | | |
| api/customer/customers/update | | Update a customer by customerId. |
| Http Method | PUT | |
| Parameter 1 | RegisterCustomerViewModel model | |
| Return | HttpResponse status code | |
| | | |
| api/customer/customers/delete/{customerId} | | Delete a customer by customerId. |
| Http Method | Delete | |
| Parameter 1 | customerId | |
| Return | `HttpResponse status code` | |
| | | |
| api/customer/customers/get/{customerId} | | Fetch customer detail by customerId. |
| Http Method | GET | |
| Parameter 1 | customerId | |
| Return | `<IEnumerable<Customer>>` | |
| | | |
| api/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 |
|---|---|---|
| Package Structure | | |
| controller | Product Controller,Customer Controller | These controllers handle all application Function, Create/Update/Edit show information |
| Startup.cs | Startup CS file | Contain all Services settings and InMemory Db Configuration. |
| Properties | launchSettings.json file | All URL Setting for API |

## 6.2 Package: Online-Auction.BusinessLayer

**Resources**

| Names | Resource | Remarks |
|---|---|---|
| Package Structure | | |
| Interface | IProductServices interface ICustomerServices interface | Inside all these interface files contains all business validation logic functions.. |
| 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. |
| Repository | IProductRepository, Product Repository,ICustomerRepository,Customer Repository CS file and interface. | All these interfaces and class files contain all CRUD operation code for InMemoryDb. |
| ViewModels | RegisterCustomerViewModel, RegisterProductViewModel. | Contain all view Domain entities for show and bind data. |

## 6.3 Package:  Online-Auction.DataLayer

**Resources**

| Names | Resource | Remarks |
|---|---|---|
| Package Structure | | |
| DataLayer | OnlineAuctionDBContext cs file | All database Connection and collection setting class |

## 6.4 Package:  Online-Auction.Entities

**Resources**

| Names | Resource | Remarks |
|---|---|---|
| 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 |

## 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

- **All actions like build, compile, running application, running test cases will be through Command Terminal.**

- **To open the command terminal the test takers need to go to the Application menu (Three horizontal lines at left top)  Terminal → New Terminal.**

- **On command prompt, cd into your project folder (cd &lt;Your-Project-folder&gt;).**

- **To build your project use command:**
  **(Project_directory/OnlineAuction / dotnet build)**

- **To launch your application, Run the following command to run the application:**
  **(Project_directory/OnlineAuction / dotnet run)**

- **This editor Auto Saves the code.**

- **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.**

- **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.**

- **To run the test cases in CMD, Run the following command to test the application:**
  **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)**

- **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.**

- **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.**

- **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.**