System Requirements Specification Index

For

Grocery e-mart Application (Collaborative)

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

Table of Contents

1.Business-Requirement:

1.1 Problem Statement:

GROCERY EMART Application is .NET CORE 3.1 RESTful API application with MongoDb.where it allows customers to place the product order, and all product maintenance, new addition and complete administration work is performed by admin.

1.2 Following is the requirement specification:

	GROCERYEMART
USERS	
1	Admin
2	Customer
Admin Functionalities	
1	Add new product category
2	Add new product
3	Edit/Update category
4	Edit/Update product
5	Remove product and category
6	Can see all order placed by user
7	Can see all product and category
Customer Functionalities	
1	Can register itself by providing these details (Name, Email, Password, Mobile Number, Pin Code, HouseNo_Building_Name, Road_area, City, State)
2	Can place a product order after login.
3	Can track status of order.
4	Can get the details of all product and category
5	Can update the self-details
6	Can get product by id
7	Can get category by id
8	Get product by name
9	Get product by category id

2. Assumptions, Dependencies, Risks / Constraints

2.1 Admin Constraints:

- While disabling the user by admin, if user id does not exist then the operation should throw a custom exception.
- While enabling the user by admin, if user id does not exist then operation should throw custom exception.
- While deleting the product category by admin, if category id does not exist then operation should throw a custom exception.
- While deleting the product by admin, if product id does not exist then the operation should throw a custom exception.
- While editing/updating the product by admin, if product id does not exist then operation should throw custom exception.
- While editing/updating the product category by admin, if category id does not exist then the operation should throw a custom exception.
- While finding the product/category using name and id by admin, if name and id does not exist then operation should throw custom exception.
- While fetching the product order using id by admin, if product order id does not exist then operation should throw custom exception.

2.2 Customer Constraints

- While placing product orders by customer, if product id does not exist then operation should throw custom exception.
- While placing product orders by customer, if the user doesn't exist then the operation should throw a custom exception.
- While placing product orders by customer, if the user doesn't login then the operation should ask login first or register first otherwise throw a custom exception.
- While placing an order by the customer, if the customer is disabled then the operation should throw a custom exception.
- While fetching product/product-category details, if product/product_category id does not exist then operation should throw custom exception.
- While order details/status, if order id email does not exist then operation should throw custom exception, User must login before getting this information.
- While updating the login user details by customer, if loan customer id/Email does not exist then operation should throw custom exception. User must login before updating his/her details.

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**

3. Business Validations

3.1 User Entity:

- User Name is not null, min 3 and max 100 characters.
- User email is not null, min 3 and max 100 characters and in proper email format.
- Password is not null, min 8 and.
- User mobile is not null, min 10 and max 10 characters.
- Pin Code is not null and must be 6 digits.
- HouseNo_Building_Name is not null and must be completed.
- Road_area is not null must complete,
- City is not a null name as per standard India based.
- State is not null as per India based State.

3.2 Product Category Detail Entity:

- CatId is not null and should be greater than 0
- Url is not null
- OpenInNewWindow must have a Boolean value, true or false.

3.3 Product Detail Entity:

- ProductName is not null.
- Description is not null
- Amount is not null and must be double the value.
- Stock is not null must be < 0 or integer value
- Photo -
- CatId – not null

4. Rest Endpoints

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

4.1 DashboardController

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1. /api/Dashboard
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>-</td></tr><tr><td>Return</td><td><IEnumerable<ProductOrder>></td></tr></table></td><td>Get list of Product order</td></tr>
<tr><td>api/Dashboard/OrderById/{OrderId}
<table><tr><td>Http Method</td><td>Get</td></tr><tr><td>Parameter 1</td><td>OrderId</td></tr><tr><td>Return</td><td>UserDto</td></tr></table></td><td>Get an order details by Id</td></tr>
<tr><td>api/Dashboard/AddCategory
<table><tr><td>Http Method</td><td>Post</td></tr><tr><td>Parameter 1</td><td>CategoryViewModel model</td></tr><tr><td>Return</td><td>HttpStatus Code</td></tr></table></td><td>Add new product category</td></tr>
<tr><td>api/Dashboard/AddProduct
<table><tr><td>Http Method</td><td>Post</td></tr><tr><td>Parameter 1</td><td>ProductViewModel model</td></tr><tr><td>Return</td><td>HttpStatus Code</td></tr></table></td><td>Add new product based on product category</td></tr>
<tr><td>api/Dashboard/Updatecategory/{Id}
<table><tr><td>Http Method</td><td>HTTPPUT</td></tr><tr><td>Parameter 1</td><td>Id, Category Model</td></tr><tr><td>Return</td><td>HttpStatus Code</td></tr></table></td><td>Update an existing product Category</td></tr>
<tr><td>api/Dashboard/UpdateProduct/{ProductId}
<table><tr><td>Http Method</td><td>HTTPPUT</td></tr><tr><td>Parameter 1</td><td>productId, Product model</td></tr><tr><td>Return</td><td>HttpStatus code</td></tr></table></td><td>Update an existing product information</td></tr>
<tr><td>api/Dashboard/RemoveCategory/{Id}
<table><tr><td>Http Method</td><td>HTTPDELETE</td></tr><tr><td>Parameter 1</td><td>Id</td></tr></table></td><td>Delete an existing Product Category</td></tr>
</table>

Return	HttpStatusCode

api/Dashboard/RemoveProduct/{productId}	Delete an existing Product
Http Method	HTTPDELETE
Parameter 1	productId
Return	HttpStatus code

4.2 UserController

URL Exposed	Purpose
api/user	Get list of register User
Http Method: GET	
Parameter 1: -	
Return: `<IEnumerable<ApplicationUser>>`	
api/user/Register	Register new user for application
Http Method: POST	
Parameter 1: `UserViewModel model`	
Return: HttpStatusCode	
api/user/Updateuser/{UserId}	Update an existing user information
Http Method: HTTPPUT	
Parameter 1: UserId, ApplicationUser model	
Return: Http Status Code	

4.3 GroceryController

URL Exposed	Purpose
api/Grocery	Fetches list of product
Http Method: GET	
Parameter 1: -	
Return: `<IEnumerable<Product>>`	
api/grocery/ProductById/{productId}	Get a single product details.
Http Method: GET	
Parameter 1: String(productId)	
Return: Product	
api/grocery/ProductByCategory/{CatId}	Get an existing product category details
Http Method: GET	
Parameter 1: String(CatId)	
Return: Category	

<table>
<tr><td></td><td></td></tr>
<tr><td>api/grocery/ProductByName/{productName}
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>String(productName)</td></tr><tr><td>Return</td><td>List of product match with name</td></tr></table></td><td>Fetches list of product match with name</td></tr>
<tr><td>api/grocery/Placeorder/{ProductId}/{email}/{password}
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>String(productId)</td></tr><tr><td>Parameter 2</td><td>String email</td></tr><tr><td>Parameter 3</td><td>String password</td></tr><tr><td>Return</td><td>OrderId</td></tr></table></td><td>Place a product order validating an existing user.</td></tr>
</table>

5. Template Code Structure

5.1 Package: GroceryEmart

Resources

Names	Resource	Remarks
Package Structure		
controller	User, Dashboard, Grocery Controller	These controllers handle all application Function, update/Edit show information and login existing users.
Startup.cs	Startup CS file	Contain all Services settings and Db Configuration.
Properties	launchSettings.json file	All URL Setting for API

5.2 Package: GroceryEmart.BusinessLayer

Resources

Names	Resource	Remarks
Package Structure		
Interface	IAdminGroceryServices, IGroceryServices, IUserGroceryServices interface	Inside all these cs files contains all business logic functions..
Service	AdminGrocery, Grocery, UserGrocery Services class file	Using this all class we are calling the Repository method and use it in the program and on the controller.
Repository	AdminGrocery, Grocery, IAdminGrocery, IGrocery, IUserGrocery, UserGrocery Repository CS file and interface.	All these interface and class files contain all CRUD operation code for MongoDb.

ViewModels	CategoryViewModel, ProductViewModel, UserViewModel Class file	Contain all view Domain entities for show and bind data.

5.3 Package: GroceryEmart.DataLayer

Resources

Names	Resource	Remarks
Package Structure		
DataLayer	Mongosettings, MongoDBContext, IMongoDBContext cs file	All MongoDb setting class

5.4 Package: GroceryEmart.Entiities

Resources

Names	Resource	Remarks
Package Structure		
Entities	Product, Category, ProductOrder cs file	All Entities/Domain attribute

5.5 Package: GroceryEmart.Tests

Resources

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

6. Considerations

For Role of Users three possible values must be used

1. Admin
2. User

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 <Your-Project-folder>).**

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

- **To launch your application, Run the following command to run the application:**
  (Project_directory/GroceryEmart / 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.**