System Requirements Specification Index

For

Grocery E-mart Application (MS SQL)

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.Project Abstract

1.1 GROCERY MART Application is .NET CORE 3.1 RESTful API application with MS SQL, 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 specifications:

	GROCERYEMART
USERS	
1	Admin
2	Customer
Dashboard Controller Functionalities	
1	Get all orders placed by user
2	Get order by id.
3	Add a new category
4	Add a new product
5	Update a category
6	Update a product
7	Remove Category
8	Remove Product
Grocery Controller Functionalities	
1	Get all products
2	Get product details by product id.
3	Get all products by category id.
4	Get product by product name
5	Get all category list.
6	Place order for user.
User Controller Functionality	
1	Display all users
2	Update an existing user

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 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 operation should ask login first or register first otherwise throw 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

URL Exposed		Purpose
/all-order		Get list of Product order
Http Method	GET	
Parameter 1	-	
Return	<IEnumerable<ProductOrder> >	
/order-byI-id/{OrderId}		Get an order details by Id
Http Method	Get	
Parameter 1	OrderId	
Return	UserDto	
/add-category		Add new product category
Http Method	Post	
Parameter 1	CategoryViewModel model	
Return	HttpStatus Code	
/add-product		Add new product based on product category
Http Method	Post	
Parameter 1	ProductViewModel model	
Return	HttpStatus Code	
/update-category/{Id}		Update an existing product Category
Http Method	HTTPPUT	
Parameter 1	Id, Category Model	
Return	HttpStatus Code	
/update-product/{ProductId}		Update an existing product information
Http Method	HTTPPUT	
Parameter 1	productId, Product model	
Return	HttpStatus code	
/remove-category/{Id}		Delete an existing Product Category
Http Method	HTTPDELETE	
Parameter 1	Id	

	Return	HttpStatusCode	
/remove-product/{productId}			Delete an existing Product
	Http Method	HTTPDELETE	
	Parameter 1	productId	
	Return	HttpStatus code	

4.2 UserController

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

4.3 GroceryController

URL Exposed			Purpose
/all-product			Fetches list of product
	Http Method	GET	
	Parameter 1	-	
	Return	`<IEnumerable<Product> >`	
/product-by-id/{productId}			Get a single product details.
	Http Method	GET	
	Parameter 1	String(productId)	
	Return	Product	
/product-by-category/{CatId}			Get an existing product category details
	Http Method	GET	

Parameter 1	String(CatId)	
Return	Category	
/product-by-name/{productName}		Fetches list of product match with name
Http Method	GET	
Parameter 1	String(productName)	
Return	List of product match with name	
/place-order/{ProductId}/{email}/{password}		Place a product order with validating existing user.
Http Method	GET	
Parameter 1	String(productId)	
Parameter 2	String email	
Parameter 3	String password	
Return	OrderId	

5. Template Code Structure

5.1 Package: GroceryEmart

Resources

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

5.2 Package: GroceryEmart.BusinessLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
Interface	IAdminGroceryServices, IGroceryServices, IUserGroceryServices interface	Inside all these cs files contains all business logic functions..	Already Implemented

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.	Partially Implemented
Repository	AdminGrocery, Grocery, IAdminGrocery, IGrocery, IUserGrocery, UserGrocery Repository CS file and interface.	All these interfaces and class files contain all CRUD operation code for Db.	Partially Implemented
ViewModels	CategoryViewModel, ProductViewModel, UserViewModel Class file	Contain all view Domain entities for show and bind data.	Already Implemented

5.3 Package: GroceryEmart.DataLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
DataLayer	GroceryemartDbContext, DataGenerator cs file	All Db setting class	Already Implemented

5.4 Package: GroceryEmart.Entiities

Resources

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

5.5 Package: GroceryEmart.Tests

Resources

Note: - Under the GroceryEmart.Tests contain All Test cases for code evaluation, please don't try to alter and edit it.

6. Considerations

- For Role of Users three possible values must be used
  1.Admin
  2.User

- Your code will also be evaluated for code quality, naming conventions, readability etc.
- Make sure you do not modify existing class and method names and their signatures, else it would severely affect the final result.
- Make sure you do not add any new class or methods, else it would severely affect the final result.
- Make sure you do not modify any test cases, else it would severely affect final result

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>).**

1. **To connect SQL server from terminal:**
   (GroceryEmart/sqlcmd -S localhost -U sa -P pass@word1)
   - **To create database from terminal -**
     1. **Create Database GroceryEmart_Db**
     2. **Go**

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

3. **To check whether migrations are applied from terminal:**
   (GroceryEmart /sqlcmd -S localhost -U sa -P pass@word1)

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

4. **To build your project use command:**
   (GroceryEmart /dotnet build)

5. **To launch your application, Run the following command to run the application:**
   (GroceryEmart /dotnet run)

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

7. **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.**

8. **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.**

9. **To run the test cases in CMD, Run the following command to test the application:**
   (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)
   (GroceryEmart.Tests/dotnet test --logger "console;verbosity=detailed").

10. **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.**

11. **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.**

12. **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.**