System Requirements Specification Index

For

Restaurant Application(Microservices)

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

RESTAURANT APPLICATION

System Requirements Specification

1. Business-Requirement:

1.1 Problem Statement:

Restaurant Application is a C#, .NET Core 3.1 Microservice with MS SQL database. Implements all the checks so that there are no errors when data is added, updated and searched.

1.2 Following is the requirement specification:

	Restaurant Application
Customer module	Add a Customer
	Update a customer by id.
	Get All Customers.
	Get Customer By id.
FoodMenu Module	Add a Menu.
	Update a Menu by id.
	Get All Menu.
	Get Menu By id.
Order Module	Add an Order.
	Update an Order by id.
	Get all Orders.
	Get Order by id.

2. Assumptions, Dependencies, Risks / Constraints

2.1 Common Constraints

- Develop application Restaurant Application using Collections, Classes, Exception handling, in C#, .NET Core 3.1.
- Define appropriate classes and objects for a given scenario.
- Build the application using C# New Features like Default interface methods Nullable reference types.
- Using Entity Framework to manipulate data includes adding, finding,and inserting data
- Use custom exceptions and other built-in exceptions like Database Exceptions, NullReferenceException at required places in applications.
- Create the Entity context class to connect the database and use appropriate methods to execute the CRUD operations for the Restaurant.
- Do not change, add, remove any existing methods in service layer
- In Repository interfaces, custom methods can be added as per requirements.

3. Rest Endpoints

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

3.1 Ocelot.json

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>/gateway/Customer<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>Customers customer</td></tr><tr><td>Return</td><td><Customers></td></tr></table></td><td>Add a Customer</td></tr>
<tr><td>/gateway/Customer<table><tr><td>Http Method</td><td>PUT</td></tr><tr><td>Parameter 1</td><td>Customer customer</td></tr><tr><td>Return</td><td><Customer></td></tr></table></td><td>Update a Customer</td></tr>
<tr><td>/gateway/Customer<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>-</td></tr></table></td><td>Fetches the list of all Customers</td></tr>
</table>

Return	<IEnumerable<Customer>>	
/gateway/Customer?customerId=1 Http Method: GET Parameter 1: int(customerId) Return: <Customer>		Fetches the details of a Customer
/gateway/Menu Http Method: POST Parameter 1: Menu menu Return: <Menu>		Add a Menu
/gateway/Menu Http Method: PUT Parameter 1: Menu menu Return: <Menu>		Update a Menu
/gateway/Menu Http Method: GET Parameter 1: - Return: <IEnumerable<Menu>>		Fetches the list of all Menu
/gateway/Menu?menuId=1 Http Method: GET Parameter 1: int(menuId) Return: <Menu>		Fetches the details of a Menu
/gateway/Order Http Method: POST Parameter 1: Orders order Return: <Orders>		Add a Order
/gateway/Order Http Method: PUT Parameter 1: Orders order Return: <Orders>		Update a Order
/gateway/Order Http Method: GET Parameter 1: -		Fetches the list of all Orders

Return	<IEnumerable<Orders >>	
/gateway/Order?orderId=1 Http Method: GET Parameter 1: int(orderId) Return: <Orders>		Fetches the details of a Order

4. Template Code Structure

4.1 Customer

4.1.1 Package: Customer.WebAPI

Resources

Names	Resource	Remarks	Status
Package Structure			
controllers	CustomerController	This controller handle all application Function, Create/Update/Edit show information	Partially Implemented
Startup.cs	Startup CS file	Contain all Services settings and Db Configuration.	Already Implemented
Properties	launchSettings.json file	All URL Setting for API	Already Implemented

4.1.2 Package: Customer.BusinessLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
Services	CustomerService CS file ICustomerService interface	Inside this interface contains all business validation logic functions and the class file is used to call the repository method.	Partially Implemented

Repository	ICustomer Repository Customer Repository CS file and interface.	All this interface and class files contain all CRUD operation code for the database.	Partially Implemented

4.1.3 Package: Customer.dataLayer

Resources

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

4.1.4 Package: Customer.Entities

Resources

Names	Resource	Remarks	Status
Package Structure			
Models	Customers	All Entities/Domain attribute are used for pass the data in controller	Already Implemented

4.1.5 Package: Customer.Tests

Resources

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

4.2 FoodMenu

4.2.1 Package: FoodMenu.WebAPI

Resources

Names	Resource	Remarks	Status
Package Structure			
controllers	MenuController	This controller handle all application Function, Create/Update/Edit show information	Partially Implemented
Startup.cs	Startup CS file	Contain all Services settings and InMemory Db Configuration.	Already Implemented
Properties	launchSettings.json file	All URL Setting for API	Already Implemented

4.2.2 Package: FoodMenu.BusinessLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
Services	MenuService CS file IMenuService interface	Inside this interface contains all business validation logic functions and the class file is used to call the repository method .	Partially Implemented
Repository	IMenu Repository MenuRepository CS file and interface.	All this interface and class files contain all CRUD operation code for the database.	Partially Implemented

4.2.3 Package: FoodMenu.dataLayer

Resources

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

4.2.4 Package: FoodMenu.Entities

Resources

Names	Resource	Remarks	Status
Package Structure			
Models	Menu	All Entities/Domain attribute are used for pass the data in controller	Already Implemented

4.2.5 Package: FoodMenu.Tests

Resources

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

4.3 Order

4.3.1 Package: Order.WebAPI

Resources

Names	Resource	Remarks	Status
Package Structure			
controllers	OrderController	This controller handle all application Function, Create/Update/Edit show information	Partially Implemented
Startup.cs	Startup CS file	Contain all Services settings and InMemory Db Configuration.	Already Implemented
Properties	launchSettings.json file	All URL Setting for API	Already Implemented

4.3.2 Package: Order.BusinessLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
Services	OrderService CS file IOrderService interface	Inside this interface contains all business validation logic functions and the class file is used to call the repository method .	Partially Implemented

Repository	IOrder Repository Order Repository CS file and interface.	All this interface and class files contain all CRUD operation code for the database.	Partially Implemented

4.3.3 Package: Order.dataLayer

Resources

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

4.3.4 Package: Order.Entities

Resources

Names	Resource	Remarks	Status
Package Structure			
Models	Order	All Entities/Domain attribute are used for pass the data in controller	Already Implemented

4.3.5 Package: Order.Tests

Resources

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

4.4 Package: OcelotGateway

Resources

Names	Resource	Remarks	Status
Package Structure			
Ocelot.json	Json file	Describes the routing of one request to another as a ReRoute.	Already Implemented
Startup.cs	Startup CS file	Contain all Services settings	Already Implemented
Properties	launchSettings.json file	All URL Setting for API	Already Implemented

5. 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. To build your project use command:
   (Project_Folder/dotnet build)

   Project Folder :
   Customer Module - Customer/Customer.WebAPI
   FoodMenu Module - FoodMenu/Foodmenu.WebAPI
   Order Module - Order/Order.WebAPI
   API GateWay - OcelotGateway

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

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

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

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

   1> Use Database_Name
   2> Go

   1> Select * From __EFMigrationsHistory

   2> Go

8. To launch your application, Run the following command to run the application:
   (Project_Folder/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 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.**

12. **To run the test cases in CMD, Run the following command to test the application: (Project_Folder/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.**
