System Requirements Specification Index

For

Home Insurance 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

HOME INSURANCE APPLICATION

System Requirements Specification

1. Business-Requirement:

1.1 Problem Statement:

Home Insurance Application is .Net Core 3.1 Microservice with MS SQL Server, where it provides the ability to retrieve an already existing quote and print the quote summary. Only registered users can get, retrieve and print quotes.

The Home Insurance application is expected to be used by home owners interested in buying home insurance on an e-commerce site having below functions:

1.2 Following is the requirement specification:

	Home Insurance Application
User Roles	
1	Admin
2	Customer
Admin Module Functionalities	
1	Register a user
2	Renew a policy - Admin can renew the policy for the user.
3	Cancel a policy - Admin can cancel the policy for the user.
4	Add a quote
5	Retrieve an existing quote.
6	View a policy
7	Search for a user
Customer Module Functionalities	
1	Register a user
2	Customer can buy a policy
3	Customer can view policy
4	Can retrieve existing quotes

2. Assumptions, Dependencies, Risks / Constraints.

2.1 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:

```
public int UserId { get; set; }
public string FirstName { get; set; }
public string LastName { get; set; }
public string UserName { get; set; }
public string Mobile { get; set; }
public string Password { get; set; }
public string Email { get; set; }
public DateTime DOB { get; set; }
public bool IsRetired { get; set; }
public bool IsValid { get; set; }
public int SocialSecurityNumber { get; set; }
public string UserType { get; set; }
```

3.2 Policy Entity:

```
public string PolicyKey { get; set; }
public int QuoteId { get; set; }
public DateTime PolicyEffectiveDate { get; set; }
public DateTime PolicyEndDate { get; set; }
public int PolicyTerm { get; set; }
public string PolicyStatus { get; set; }
```

3.3 Quote Entity:

```
public int QuoteId { get; set; }
public double Premium { get; set; }
public double Dwelling { get; set; }
public double DetachedStructure { get; set; }
public double PersonalProperty { get; set; }
public double AdditionalLivingproperty { get; set; }
public double MedicalExpense { get; set; }
public double Deductable { get; set; }
public long UserId { get; set; }
```

4. Rest Endpoints

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

4.1 Ocelot.json

URL Exposed	Purpose
/gateway/admin/signup Http Method: POST Parameters: User user Return: HTTP Response StatusCode	Register a user
/gateway/admin/add-quote Http Method: POST Parameter 1: Quote quote Return: HTTP Response StatusCode	Add a quote
/gateway/admin/renew-policy/{policyKey} Http Method: PUT Parameter 1: String(policyKey) Return: <Policy>	Renew a policy using policyKey
/gateway/admin/cancel-policy/{policyKey} Http Method: PUT Parameter 1: String (policyKey) Return: <Policy>	Cancel a policy using policyKey
/gateway/admin/view-policy/{policyKey} Http Method: GET Parameter 1: String(policyKey) Return: <Policy >	Get details of a policy
/gateway/admin/retrieve-quote/{userId} Http Method: GET Parameter 1: Int(userId) Return: <Quote>	Get a quote details
/gateway/customer/signup Http Method: POST	Register a user

<table>
<tr><td><table>
<tr><td>Parameters</td><td>User user</td></tr>
<tr><td>Return</td><td>HTTP Response StatusCode</td></tr>
</table></td><td></td></tr>
<tr><td>/gateway/customer/view-policy/{policyKey}<table>
<tr><td>Http Method</td><td>GET</td></tr>
<tr><td>Parameter 1</td><td>String(policyKey)</td></tr>
<tr><td>Return</td><td><Policy></td></tr>
</table></td><td>Get details of a policy</td></tr>
<tr><td>/gateway/customer/buy-policy/{quoteId}<table>
<tr><td>Http Method</td><td>GET</td></tr>
<tr><td>Parameter 1</td><td>Int(quoteId)</td></tr>
<tr><td>Parameter 2</td><td>Policy policy</td></tr>
<tr><td>Return</td><td><Policy></td></tr>
</table></td><td>Buy a policy</td></tr>
<tr><td>/gateway/customer/retrieve-quote/{userId}<table>
<tr><td>Http Method</td><td>GET</td></tr>
<tr><td>Parameter 1</td><td>Int(userId)</td></tr>
<tr><td>Return</td><td><Quote></td></tr>
</table></td><td>Get details of an existing quote</td></tr>
</table>

5. Template Code Structure

5.1 CustomerServices

5.1.1 Package: Customer

Resources

Names	Resource	Remarks	Status
Package Structure			
controller	Customer Controller	These controllers handle all application Function, Create/Update/Edit show information	Partially Implemented
Startup.cs	Startup CS file	Contain all Services settings and MS SQL Db Configuration.	Already Implemented
Properties	launchSettings.json file	All URL Setting for API	Already Implemented
	appsettings.json	Contain connection string for database	Already Implemented

5.1.2 Package: Customer.BusinessLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
Interface	ICustomerServices interface	Inside all these interface files contains all business validation logic functions.	Already Implemented
Service	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	ICustomerRepository CustomerRepository CS file and interface.	All these interfaces and class files contain all CRUD operation code for MS SQL Database.	Partially Implemented
ViewModels	PolicyModel	Contain all view Domain entities for show and bind data.	Already Implemented

5.1.3 Package: Customer.DataLayer

Resources

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

5.1.4 Package: Customer.Entities

Resources

Names	Resource	Remarks	Status
Package Structure			
Entities	User ,Policy, Quote CS file	All Entities/Domain attribute are used for pass the data in controller	Already Implemented

5.1.5 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.

5.2 AdminServices

5.2.1 Package: Admin

Resources

Names	Resource	Remarks	Status
Package Structure			
controller	Admin Controller	These controllers handle all application Function, Create/Update/Edit show information	Partially Implemented
Startup.cs	Startup CS file	Contain all Services settings and MS SQL Db Configuration.	Already Implemented

Properties	launchSettings.json file	All URL Setting for API	Already Implemented
	appsettings.json	Contain connection string for database	Already Implemented

5.2.2 Package: Admin.BusinessLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
Interface	IAdminServices interface	Inside all these interface files contains all business validation logic functions.	Already Implemented
Service	Admin 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	IAdminRepository Admin Repository CS file and interface.	All these interfaces and class files contain all CRUD operation code for MS SQL Database.	Partially Implemented
ViewModels	PolicyModel	Contain all view Domain entities for show and bind data.	Already Implemented

5.2.3 Package: Admin.DataLayer

Resources

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

5.2.4 Package: Admin.Entities

Resources

Names	Resource	Remarks	Status
Package Structure			
Entities	User, ,Policy,Quote, CS file	All Entities/Domain attribute are used for pass the data in controller	Already Implemented

5.2.5 Package: Admin.Tests

Resources

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

5.3 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

6. 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>).
   Service Project Folders :
   - (cd CustomerServices/Customer)
   - (cd AdminServices/Admin)

   API Gateway:
   - (cd OcelotGateway)

4. To connect SQL server from terminal:
   (CustomerServices/Customer/sqlcmd -S localhost -U sa -P pass@word1)
   - To create database from terminal -
     ```
     1> Create Database HomeInsurance_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-
     (CustomerServices/Customer/dotnet-ef database update)

6. To check whether migrations are applied from terminal:
   (CustomerServices/Customer/sqlcmd -S localhost -U sa -P pass@word1)
   ```
   1> Use HomeInsurance_Db
   2> Go
   1> Select * From __EFMigrationsHistory
   2> Go
   ```

7. Run one of the Service and API Gateway.
   Open two terminals and use the "dotnet run" command to start them.
   - (<Service-Project-Folder>/dotnet run)
   - (OcelotGateway/dotnet run)

8. This editor Auto Saves the code.

9. **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.**

   (NOTE: Copy URL of API Gateway to test Rest End Points - https://localhost:5001)

10. **To run the test cases in CMD, Run the following command to test the application:**
    (<Service-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)

11. **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.**

12. **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.**

13. **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.**
