System Requirements Specification Index

For

Home Insurance Application

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 Restful Web API 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; }
```

3.4 Property Entity:

```
public int PropertyId { get; set; }
public int QuoteId { get; set; }
public int UserId { get; set; }
public double MarketValue { get; set; }
public int YearBuild { get; set; }
public string SqureFootValue { get; set; }
public string DwellingStyle { get; set; }
public string RoofMaterial { get; set; }
public string GarageType { get; set; }
public int NumFullBaths { get; set; }
public int NumHalfBaths { get; set; }
public bool HasSwimmingPool { get; set; }
```

3.4 Residence Entity:

```
public string QuoteId { get; set; }
public string ResidenceType { get; set; }
public string AddressLine1 { get; set; }
public string AddressLine2 { get; set; }
public string City { get; set; }
public int Zip { get; set; }
public string ResidenceUse { get; set; }
public string UserName { get; set; }
```

4. REST ENDPOINTS

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

4.1 AdminController

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

4.2 CustomerController

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

5. TEMPLATE CODE STRUCTURE

5.1 Package: HomeInsurance
Resources

Names	Resource	Remarks	Status
Package Structure			
controller	Admin 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

5.2 Package: HomeInsurance.BusinessLayer
Resources

Names	Resource	Remarks	Status
Package Structure			
Interface	IAdminServices interface ICustomerServices interface	Inside all these interface files contains all business validation logic functions.	Already Implemented
Service	Admin 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.	Partially Implemented
Repository	IAdminRepository Admin Repository ICustomerRepository CustomerRepository CS file and interface.	All these interfaces and class files contain all CRUD operation code for InMemory Database.	Partially Implemented

Names	Resource	Remarks	Status
ViewModels	PolicyModel	Contain all view Domain entities for show and bind data.	Already Implemented

5.3 Package: HomeInsurance.DataLayer
Resources

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

5.4 Package: HomeInsurance.Entities
Resources

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

5.5 Package: HomeInsurance.Tests
Resources

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

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>).

4. To connect SQL server from terminal:
   (HomeInsurance/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-
     (HomeInsurance /dotnet-ef database update)

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

7. To build your project use command:
   (HomeInsurance /dotnet build)

8. To launch your application, Run the following command to run the application:
   (HomeInsurance /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 web-based applications on a browser, use the internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.**

    Note: The application will not run in the local browser

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