System Requirements Specification Index

For

Insurance Management System

Version 1.0

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

Insurance Policy Management
System Requirements Specification

1. BUSINESS-REQUIREMENT:

1.1 PROBLEM STATEMENT:

Insurance, a critical component of financial services, is undergoing a transformative phase with the advent of digital technology. This evolution mirrors the broader trends in the financial technology (fintech) sector, which is revolutionizing the way we interact with financial services, making them more accessible, efficient, and tailored to individual needs. Fintech innovations are streamlining operations, enhancing customer experience, and offering new avenues for managing financial risks and opportunities.

In the realm of insurance, fintech is playing a pivotal role by introducing tools and platforms that automate and simplify the management of insurance policies. From policy issuance and administration to claims processing, technology is making insurance more adaptable to the changing needs of consumers and businesses alike.

Our organization, leveraging the latest in fintech innovations, is introducing the Insurance Policy Management Application, a cutting-edge .Net Core web API 3.1 application integrated with MS SQL Server and Angular. This application is designed to revolutionize the insurance sector by providing a comprehensive solution for the systematic administration and handling of insurance policies throughout their lifecycle. It encompasses the creation, updating, retrieval, and deletion of insurance policies, thereby ensuring accurate and secure management of policy-related information.

Insurance Policy Management Application is .Net Core web API 3.1 application integrated with MS SQL Server and frontend being implemented in Angular, where it involves the systematic administration and handling of insurance policies throughout their lifecycle. This process includes the creation, updating, retrieval, and deletion of insurance policies, ensuring accurate and secure management of policy-related information.

1.2 FOLLOWING IS THE REQUIREMENT SPECIFICATION:

	Insurance Policy Management
	This core module is dedicated to enabling comprehensive insurance policy management, facilitating efficient handling of policy details from inception through to conclusion.
Modules	
1	Insurance Policy
Insurance Policy Module Functionalities	
1	**Create an Insurance Policy:** Enables the initiation and addition of new insurance policies into the system.
2	**Update the existing Insurance Policy:** Allows for modifications to existing policies to ensure they are up-to-date.
3	**Get an Insurance Policy by Id:** Users can quickly access specific policy details using a unique identifier.
4	**Fetch all Insurance Policies:** Provides a holistic view of all policies for effective tracking and oversight.
5	**Delete an existing Insurance Policy:** Facilitates the removal of policies from the system, keeping the portfolio current.

2. ASSUMPTIONS, DEPENDENCIES, RISKS / CONSTRAINTS

2.1 Insurance Policy Constraints:

- While deleting the policy, if policy Id does not exist then the operation should throw a custom exception.
- While fetching the policy details by id, if policy id does not exist then the operation should throw a custom exception.

2.2 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

Insurance Policy Class Entities

- Policy Id (long) Not null, Key attribute.
- Customer Id (int) Not null.
- Policy Number (string) is not null, min 3 and max 100 characters.
- Premium Amount (decimal) is not null.
- Start Date (Date)
- End Date (Date)
- Policy Type (string) Not null.
- Is Active bool

4. CONSIDERATIONS

- There is no roles in this application
- You can perform the following 3 possible actions

Insurance Policy

5. REST ENDPOINTS

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

5.1 InsurancePolicyController

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>/create-policy

<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>InsurancePolicy model</td></tr><tr><td>Return</td><td>HTTP Response StatusCode</td></tr></table></td><td>Create Insurance Policy</td></tr>
<tr><td>/update-policy</td><td></td></tr>
</table>

Http Method	PUT	Update an Insurance Policy
Parameter 1	Long Id	
Parameter 2	InsurancePolicyView Model model	
Return	HTTP Response StatusCode	
/get-all-policies		Fetches the list of all Insurance Policies
Http Method	GET	
Parameter 1	-	
Return	<IEnumerable<PoliticalParty>>	
/get-policy-by-id?id={id}		Fetches the details of an Insurance Policy
Http Method	GET	
Parameter 1	Long (id)	
Return	<InsurancePolicy>	
/delete-policy?id={id}		Delete an Insurance Policy
Http Method	DELETE	
Parameter 1	Long (id)	
Return	HTTP Response StatusCode	

6. TEMPLATE CODE STRUCTURE

6.1 Package: InsurancePolicyManagement

Resources

Names	Resource	Remarks	Status
Package Structure			
Controller	InsurancePolicyController	Controller class to expose all rest-endpoints for auction related activities.	Partially implemented
Startup.cs	Startup CS file	Contain all Services settings and SQL server Configuration.	Already Implemented
Properties	launchSettings.json file	All URL Setting for API	Already Implemented
	appsettings.json	Contain connection string for database	Already Implemented

6.2 Package: InsurancePolicyManagement.BusinessLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
Interface	IInsurancePolicyServices interface	Inside all these interface files contains all business validation logic functions.	Already implemented
Service	InsurancePolicy 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	IInsurancePolicy Repository InsurancePolicy Repository (CS files and interfaces)	All these interfaces and class files contain all CRUD operation code for the database. Need to provide implementation for service related functionalities	Partially implemented
ViewModels	InsurancePolicy ViewModel	Contain all view Domain entities for show and bind data. All the business validations must be implemented.	Partially implemented

6.3 Package: InsurancePolicyManagement.DataLayer

Resources

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

6.4 Package: InsurancePolicyManagement.Entities

Resources

Names	Resource	Remarks	Status
Package Structure			
Entities	InsurancePolicy ,Response (CS files)	All Entities/Domain attribute are used for pass the data in controller and status entity to return response Annotate this class with proper annotation to declare it as an entity class with **Id** as primary key. Generate the **Id** using the **IDENTITY** strategy	Partially implemented

FRONTEND-ANGULAR SPA

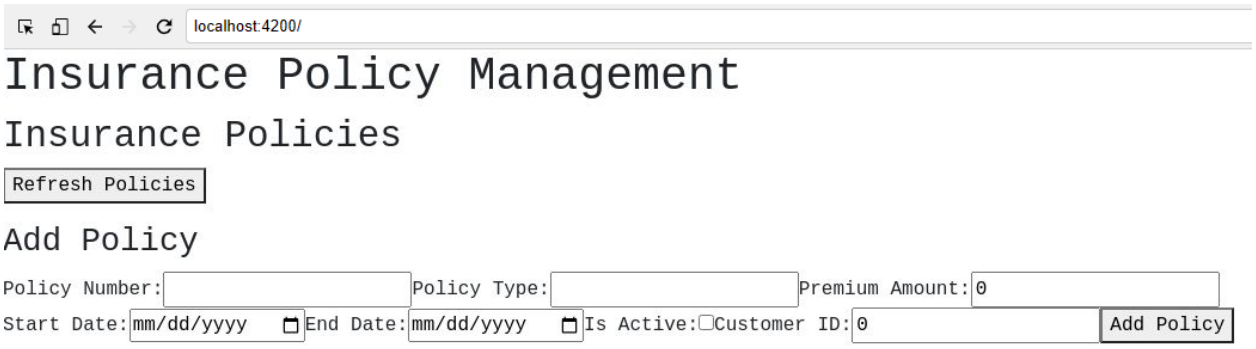
1. PROBLEM STATEMENT

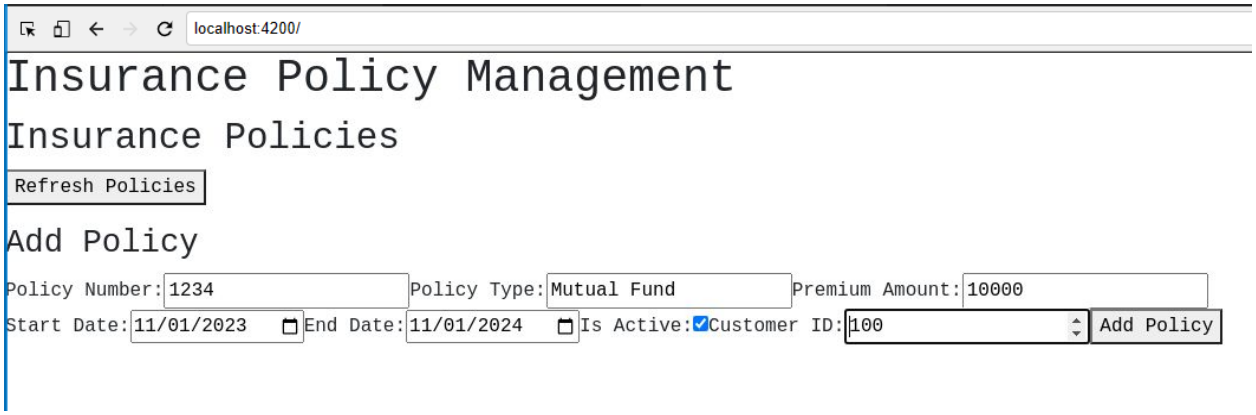
Insurance Policy Application is SPA (Single Page Application), it allows you to add policy details, update policy details, delete policy and get all policies.

2. PROPOSED INSURANCE POLICY APPLICATION WIREFRAME

UI needs improvisation and modification as per given use case and to make test cases passed.

2.1 HOME PAGE




localhost:4200/
Insurance Policy Management
Insurance Policies
Refresh Policies
Add Policy
Policy Number: 1234
Policy Type: Mutual Fund
Premium Amount: 10000
Start Date: 11/01/2023
End Date: 11/01/2024
Is Active:
Customer ID: 100
Add Policy


localhost:4200/
Insurance Policy Management
Insurance Policies
Refresh Policies
• 1234 - Mutual Fund - 10000
Select
Update
Delete
Add Policy
Policy Number:
Policy Type:
Premium Amount: 0
Start Date: mm/dd/yyyy
End Date: mm/dd/yyyy
Is Active:
Customer ID: 0
Add Policy

3. BUSINESS-REQUIREMENT:

As an application developer, develop the Insurance Policy Management (Single Page App) with below guidelines:

User Story #	User Story Name	User Story
US_01	Home Page	As a user I should be able to visit the Home page as the default page.

US_01	Home Page	As a user I should be able to see the homepage and perform all operations: Acceptance criteria: 1. Add “Insurance Policies” as heading in h2. 2. Should have a “Refresh Policies” button. 3. Should show a list of all policies with “Update” and “Delete” button in each of the policy. 4. As a user I should be able to furnish the following details at the time of creating a policy. 1.1 Policy Number 1.2 Policy Type 1.3 Premium Amount 1.4 Start Date 1.5 End Date 1.6 Is Active 1.7 Customer ID 5. All fields should be a required fields to add a policy.

4. Execution Steps to Follow For Backend

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:**
   (InsurancePolicyManagement /sqlcmd -S localhost -U sa -P pass@word1)
   - **To create database from terminal -**

     1> Create Database InsuranceDb
     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-**
     (InsurancePolicyManagement /dotnet-ef database update)

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

   1> Use InsuranceDb
   2> Go
   1> Select * From __EFMigrationsHistory
   2> Go

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

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

---

5. EXECUTION STEPS TO FOLLOW FOR FRONTEND

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 Application menu (Three horizontal lines at left top) -> Terminal ->New Terminal.**
3. **This is a web-based application, to run the application on a browser, use the**

internal browser in the environment.

4. **You can follow series of command to setup Angular environment once you are in your project-name folder:**
    a. **npm install -> Will install all dependencies -> takes 10 to 15 min**
    b. **npm run start -> To compile and deploy the project in browser. You can press <Ctrl> key while clicking on localhost:4200 to open project in browser -> takes 2 to 3 min**
    c. **npm run test -> to run all test cases. It is mandatory to run this command before submission of workspace -> takes 5 to 6 min**
5. **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.**
