
System Requirements Specification Index

For

Insurance Policy Management

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

Insurance Policy Management System Requirements Specification

1. BUSINESS-REQUIREMENT:

1.1 PROBLEM STATEMENT:

Insurance Policy Management Application is .Net Core web API 3.1 application integrated with MS SQL Server, 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	
Modules		
1	Insurance Policy	
Insurance Policy Module Functionalities		
1	Create an Insurance Policy	
2	Update the existing Insurance Policy	
3	Get an Insurance Policy by Id	
6	Fetch all Insurance Policies	
7	Delete an existing Insurance Policy	

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.

2.4 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 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

URL Exposed		Purpose
/policy		Create Insurance Policy
Http Method	POST	
Parameter 1	InsurancePolicy model	
Return	HTTP Response StatusCode	
/policy		Update an Insurance Policy
Http Method	PUT	
Parameter 1	Long Id	
Parameter 2	InsurancePolicyView Model model	
Return	HTTP Response StatusCode	
/policies		Fetches the list of all Insurance Policies
Http Method	GET	
Parameter 1	-	
Return	<IEnumerable<Political Party>>	
/policy?id={id}		Fetches the details of an Insurance Policy
Http Method	GET	
Parameter 1	Long (id)	
Return	<InsurancePolicy>	
/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 insurance 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	InsurancePolicyServices 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
Service/ 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)	<p>All Entities/Domain attribute are used for pass the data in controller and status entity to return response</p> <p>Annotate this class with proper annotation to declare it as an entity class with Id as primary key.</p> <p>Generate the Id using the IDENTITY strategy</p>	Partially implemented

7. METHOD DESCRIPTIONS

1. InsurancePolicyService: Method Descriptions

Method	Task	Implementation Details
CreateInsurance Policy	To implement logic to create a new insurance policy.	<ul style="list-style-type: none"> - Accept InsurancePolicy object as parameter - Call <code>_insurancePolicyRepository.CreateInsurancePolicy(insurancePolicy)</code> - Return the result from the repository method
DeleteInsurance PolicyById	To implement logic to delete an insurance policy by its ID.	<ul style="list-style-type: none"> - Accept policy ID as parameter - Call <code>_insurancePolicyRepository.DeleteInsurancePolicyById(id)</code> - Return true if deletion is successful
GetAllInsurance Policies	To implement logic to fetch all insurance policies.	<ul style="list-style-type: none"> - Call <code>_insurancePolicyRepository.GetAllInsurancePolicies()</code> - Return the list of insurance policies

GetInsurancePolicyById	To implement logic to fetch an insurance policy by ID.	<ul style="list-style-type: none"> - Accept policy ID as parameter - Call <code>_insurancePolicyRepository.GetInsurancePolicyById(id)</code> - Return the insurance policy object
UpdateInsurancePolicy	To implement logic to update an existing insurance policy.	<ul style="list-style-type: none"> - Accept <code>InsurancePolicyViewModel</code> as parameter - Call <code>_insurancePolicyRepository.UpdateInsurancePolicy(model)</code> - Return the updated insurance policy object

2. InsurancePolicyRepository: Method Descriptions

Method	Task	Implementation Details
CreateInsurancePolicy	To implement logic for inserting a new <code>InsurancePolicy</code> into the database.	<ul style="list-style-type: none"> - Use try-catch block - In try: Use <code>_dbContext.InsurancePolicies.AddAsync(insurancePolicy)</code> to add new policy - Call <code>SaveChangesAsync()</code> to persist data - Return the created <code>InsurancePolicy</code> object - In catch: throw the caught exception
DeleteInsurancePolicyById	To implement logic to delete an <code>InsurancePolicy</code> by its ID.	<ul style="list-style-type: none"> - Use try-catch block - In try: Use LINQ to find policy with matching <code>PolicyId</code> - Use <code>_dbContext.Remove()</code> to delete the policy - Call <code>SaveChanges()</code> to persist the deletion - Return true if deletion succeeds - In catch: throw the caught exception
GetAllInsurancePolicies	To implement logic to fetch the latest 10 insurance policies.	<ul style="list-style-type: none"> - Use try-catch block - In try: Use <code>_dbContext.InsurancePolicies.OrderByDescending(x => x.PolicyId).Take(10).ToList()</code> to retrieve latest policies - Return the list of <code>InsurancePolicy</code> objects - In catch: throw the caught exception

GetInsurancePolicyById	To implement logic to fetch a specific InsurancePolicy by its ID.	<ul style="list-style-type: none"> - Use try-catch block - In try: Use <code>_dbContext.InsurancePolicies.FindAsync(id)</code> to retrieve the policy - Return the InsurancePolicy object if found - In catch: throw the caught exception
UpdateInsurancePolicy	To implement logic to update an existing InsurancePolicy using data from a ViewModel.	<ul style="list-style-type: none"> - Use try-catch block - In try: <ul style="list-style-type: none"> • Use <code>_dbContext.InsurancePolicies.FindAsync(model.PolicyId)</code> to fetch existing record • Update fields: StartDate, EndDate, PolicyNumber, CustomerId, IsActive, PolicyType • Use <code>_dbContext.InsurancePolicies.Update(policy)</code> to update data • Call <code>SaveChangesAsync()</code> to persist changes - Return the updated InsurancePolicy object - In catch: throw the caught exception

3. InsurancePolicyController: Method Descriptions

Method	Task	Implementation Details
CreateInsurancePolicy	To implement logic to create a new insurance policy with validation.	<ul style="list-style-type: none"> - Request type: POST, URL: <code>/policy</code> - Accept <code>[FromBody]</code> InsurancePolicy model - Call <code>_insurancePolicyService.GetInsurancePolicyById(model.PolicyId)</code> to check if policy already exists - If exists, return StatusCode 500 with message: 'Insurance Policy already exists!' - Else, call <code>_insurancePolicyService.CreateInsurancePolicy(model)</code> - If result is null, return StatusCode 500 with message: 'Insurance Policy creation failed! Please check details and try again.' - If successful, return Ok with message: 'Insurance Policy created successfully!'
UpdateInsurancePolicy	To implement logic to update an	<ul style="list-style-type: none"> - Request type: PUT, URL: <code>/policy</code> - Accept <code>[FromBody]</code> InsurancePolicyViewModel model - Call <code>_insurancePolicyService.UpdateInsurancePolicy(model)</code>

	existing insurance policy.	<ul style="list-style-type: none"> - If result is null, return StatusCode 500 with message: 'Insurance Policy With Id = {model.PolicyId} cannot be found' - Else, return Ok with message: 'Insurance Policy updated successfully!'
DeleteInsurancePolicy	To implement logic to delete an insurance policy by ID with validation.	<ul style="list-style-type: none"> - Request type: DELETE, URL: /policy?id={id} - Accept id as query parameter - Call _insurancePolicyService.GetInsurancePolicyById(id) to check if policy exists - If not found, return StatusCode 500 with message: 'Insurance Policy With Id = {id} cannot be found' - Else, call _insurancePolicyService.DeleteInsurancePolicyById(id) - Return Ok with message: 'Insurance policy deleted successfully!'
GetInsurancePolicyById	To implement logic to retrieve an insurance policy by its ID.	<ul style="list-style-type: none"> - Request type: GET, URL: /policy?id={id} - Accept id as query parameter - Call _insurancePolicyService.GetInsurancePolicyById(id) - If not found, return StatusCode 500 with message: 'Insurance Policy With Id = {id} cannot be found' - Else, return Ok with the policy data
GetAllPolicies	To implement logic to retrieve all insurance policies.	<ul style="list-style-type: none"> - Request type: GET, URL: /policies - Call _insurancePolicyService.GetAllInsurancePolicies() - Return the list of insurance policies

8. 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:
(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.
-

