System Requirements Specification

Index

For

Insurance Management Application

Version 1.0

TABLE OF CONTENTS

INSURANCE POLICY MANAGEMENT

System Requirements Specification

You need to consume APIs exposed by Backend application in Angular to make application work as FULLSTACK

BACKEND-SPRING BOOT RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **Insurance Policy Management** is a FullStack Application with a backend implemented using Spring Boot with a MySQL database and a frontend developed using Angular. The application aims to provide a comprehensive platform for managing and organizing all insurance policies for a company.

Following is the requirement specifications:

	Insurance Policy Management
Modules	
1	Insurance Policy
Insurance Policy Module Functionalities	
1	Get all policies
2	Get policy by id
3	Create a new policy
4	Update a policy by id
5	Delete a policy by id

2 Assumptions, Dependencies, Risks / Constraints

2.1 POLICY CONSTRAINTS

- When fetching a policy by ID, if the policy ID does not exist, the service method should throw “Insurance Policy not found” message with NotFoundException class.
- When updating a policy, if the policy ID does not exist, the service method should throw “Insurance Policy not found” message with NotFoundException class.
- When removing a policy, if the policy ID does not exist, the service method should throw “Insurance Policy not found” message with NotFoundException class.

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 dto 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

- PolicyNumber should not be null and size must be minimum 3 and maximum 10.
- PolicyType should not be null.
- PremiumAmount should not be null.
- StartDate should not be null.
- EndDate should not be null.

4 REST ENDPOINTS

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

4.1 INSURANACECONTROLLER

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1. /api/policies<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter</td><td>-</td></tr><tr><td>Return</td><td>List<InsurancePolicyDTO></td></tr></table></td><td>Fetches all the policies</td></tr>
<tr><td>2. /api/policies/{id}<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Long (id)</td></tr><tr><td>Return</td><td>InsurancePolicyDTO</td></tr></table></td><td>Fetches a policy by id</td></tr>
<tr><td>3. /api/policies/<table><tr><td>Http Method</td><td>POST

The policy data to be created should be received in @RequestBody</td></tr><tr><td>Parameter</td><td>-</td></tr><tr><td>Return</td><td>InsurancePolicyDTO</td></tr></table></td><td>Creates a new policy</td></tr>
<tr><td>4. /api/policies/{id}<table><tr><td>Http Method</td><td>PUT

The policy data to be updated should be received in @RequestBody</td></tr><tr><td>Parameter 1</td><td>Long (id)</td></tr><tr><td>Return</td><td>InsurancePolicyDTO</td></tr></table></td><td>Updates a policy by id</td></tr>
<tr><td>5. /api/policies/{id}<table><tr><td>Http Method</td><td>DELETE</td></tr><tr><td>Parameter 1</td><td>Long (id)</td></tr><tr><td>Return</td><td>-</td></tr></table></td><td>Deletes a policy by id</td></tr>
</table>

5 TEMPLATE CODE STRUCTURE

5.1 PACKAGE: COM.INSURANCEPOLICY

Resources

insurancePolicyManagementApplication (Class)	This is the Spring Boot starter class of the application.	Already Implemented

5.2 PACKAGE: COM.INSURANCEPOLICY.REPOSITORY

Resources

Class/Interface	Description	Status
InsurancePolicyRepository (interface)	• Repository interface exposing CRUD functionality for insurance policy Entity. • You can go ahead and add any custom methods as per requirements.	Partially implemented.

5.3 PACKAGE: COM.INSURANCEPOLICY.SERVICE

Resources

Class/Interface	Description	Status
InsurancePolicyService (interface)	• Interface to expose method signatures for insurance policy related functionality. • Do not modify, add or delete any method.	Already implemented.

5.4 PACKAGE: COM.INSURANCEPOLICY.SERVICE.IMPL

Class/Interface	Description	Status
InsurancePolicyServiceImpl (class)	• Implements InsurancePolicyService. • Contains template method implementation. • Need to provide implementation for insurance policy related functionalities. • Do not modify, add or delete any method signature	To be implemented.

5.5 PACKAGE: COM.INSURANCEPOLICY.CONTROLLER

Resources

Class/Interface	Description	Status
insurancePolicyController (Class)	• Controller class to expose all rest-endpoints for insurance policy related activities. • May also contain local exception handler methods	To be implemented

5.6 PACKAGE: COM.INSURANCEPOLICY.DTO

Resources

Class/Interface	Description	Status
InsurancePolicyDTO (Class)	• Use appropriate annotations for validating attributes of this class.	Partially implemented.

5.7 PACKAGE: COM.INSURANCEPOLICY.ENTITY

Resources

Class/Interface	Description	Status
InsurancePolicy (Class)	• This class is partially implemented. • Annotate this class with proper annotation to declare it as an entity class with **policyId** as primary key. • Map this class with a **insurance policy** table. • Generate the **policyId** using the IDENTITY strategy	Partially implemented.

5.8 PACKAGE: COM.INSURANCEPOLICY.EXCEPTION

Resources

Class/Interface	Description	Status
NotFoundException (Class)	• Custom Exception to be thrown when trying to fetch, update or delete the insurance policy info which does not exist. • Need to create Exception Handler for same wherever needed (local or global)	Already implemented.

FRONTEND-ANGULAR SPA

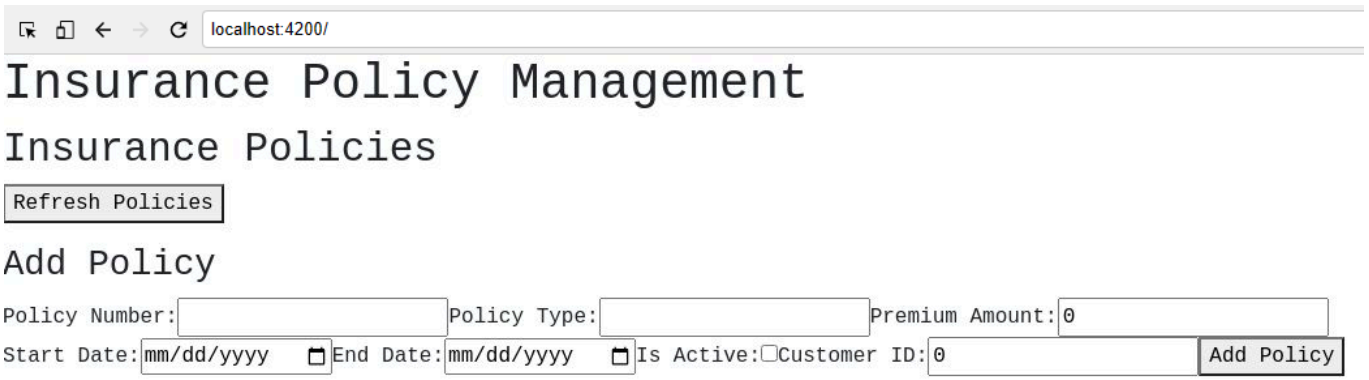
1 PROBLEM STATEMENT

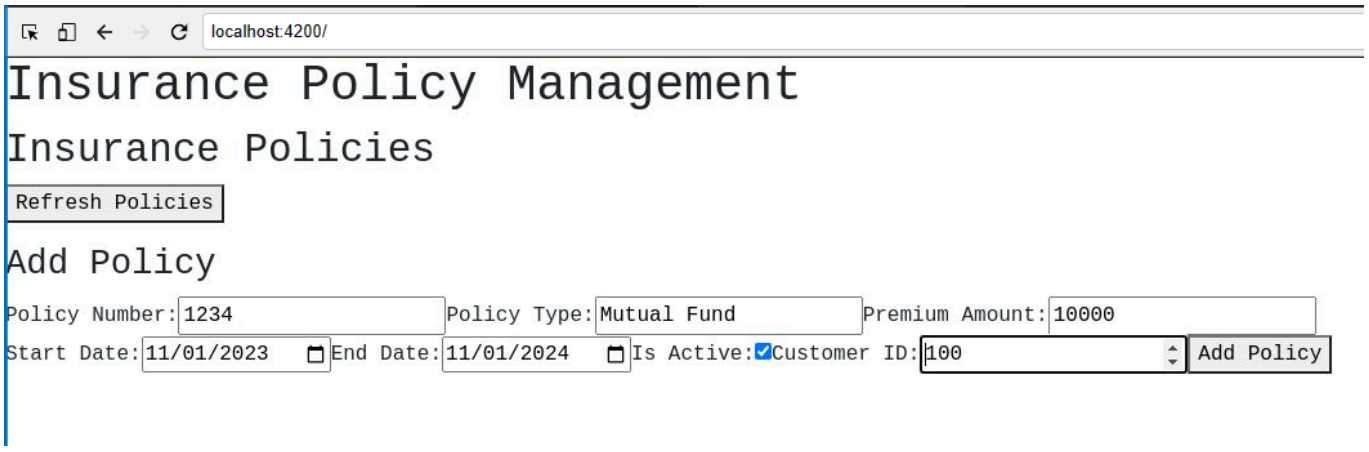
Insurance Policy Application is SPA (Single Page Application), it allows you to add policy details, update policy details, delete policy and get all policies.

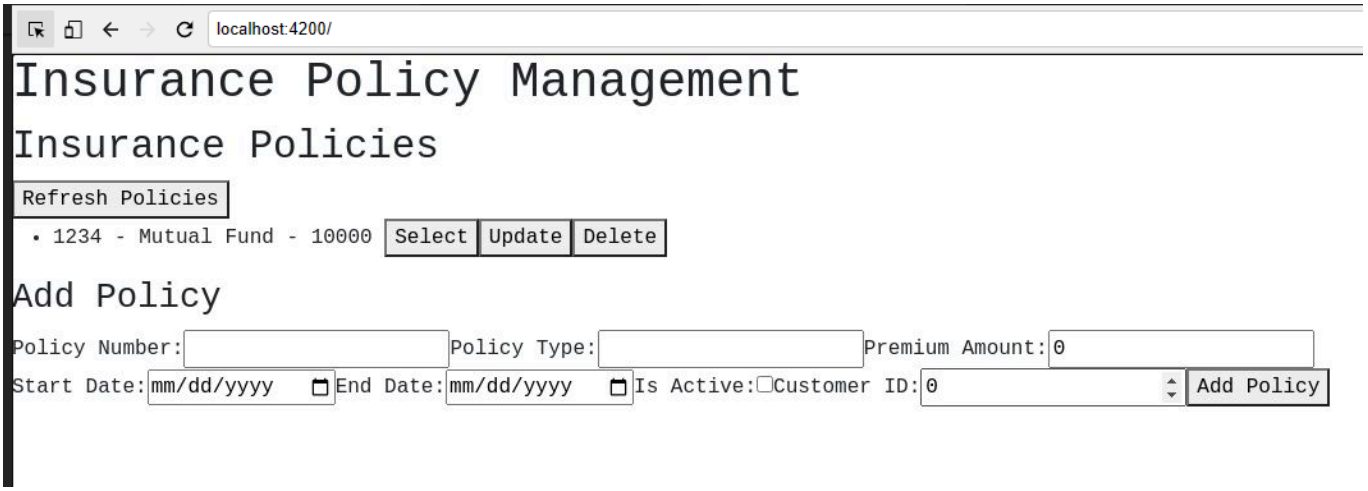
2 PROPOSED INSURANCE POLICY APPLICATION WIREFRAME

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

2.1 HOME PAGE







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 “Refresh Policies” button. 3. Should show 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 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. **cd into your backend project folder**
4. **To build your project use command:**
   mvn clean package -Dmaven.test.skip
5. **To launch your application, move into the target folder (cd target). Run the following command to run the application:**
   java -jar <your application jar file name>
6. **This editor Auto Saves the code.**
7. **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.**
8. **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.**
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.**
10. **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.**
11. **Default credentials for MySQL:**
    a. **Username: root**
    b. **Password: pass@word1**

12. To login to mysql instance: Open new terminal and use following command:
    a. sudo systemctl enable mysql
    b. sudo systemctl start mysql

       NOTE: After typing any of the above commands you might encounter any warnings.

       >> Please note that this warning is expected and can be disregarded. Proceed to the next step.

    c. mysql -u root -p
       The last command will ask for password which is 'pass@word1'
13. Mandatory: Before final submission run the following command:
       mvn test
14. 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.**