
System Requirements Specification

Index

For

**Policy Finder
Application**

Version 1.0

TABLE OF CONTENTS

BACKEND-SPRING BOOT RESTFUL APPLICATION	3
1. Project Abstract	3
2. Assumptions, Dependencies, Risks / Constraints	5
2.1. Policy Constraints	5
2.2. Common Constraints	5
3. Business Validations	6
3.1. Business Validations - Policy	6
4. Rest Endpoints	7
4.1. PolicyController	7
5. Template Code Structure	9
5.1. Package: com.policyfinder	9
5.2. Package: com.policyfinder.repository	9
5.3. Package: com.policyfinder.service	10
5.4. Package: com.policyfinder.service.impl	11
5.5. Package: com.policyfinder.controller	11
5.6. Package: com.policyfinder.dto	12
5.7. Package: com.policyfinder.entity	12
5.8. Package: com.policyfinder.exception	13
5.9. Properties Files	13
6. Execution Steps to Follow for Backend	14

POLICY FINDER APPLICATION

System Requirements Specification

BACKEND-SPRING BOOT RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **Policy Finder Application** is implemented using Spring Boot with a MySQL database. The application aims to provide a comprehensive platform for managing and registering different types of volunteers for different types of programs.

Following is the requirement specifications:

	Policy Finder Application
Modules	
1	Policy
Policy Module Functionalities	
1	Create a policy
2	Get policy by id
3	Get all policies
4	Update policy by id
5	Delete policy by id
6	Search policies by name (should be a custom query)
7	Search policies by coverage (should be a custom query)

Overall Application	
1	Actuator support needs to be added in the properties file. Expose all actuator endpoints except beans.
2	In application.properties file expose a property "profile.validate.data" with value as "This is default profile". Create application-qa.properties file (for QA profile) and expose a property "profile.validate.data" with value as "This is qa profile".
3	Create an endpoint in PolicyController with following configurations: 1. Method - GET 2. Endpoint - /profile 3. Return - String The method for this endpoint must read the "profile.validate.data" property file and return its value based on the active profile.

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 a `ResourceNotFoundException` with "Policy not found." message.
- When updating a policy by ID, if the policy ID does not exist, the service method should throw a `ResourceNotFoundException` with "Policy not found." message.
- When deleting a policy by ID, if the policy ID does not exist, the service method should throw a `ResourceNotFoundException` with "Policy not found." message.

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 dto classes only.
- All the database operations must be implemented on entity object only
- Do not change, add, remove any existing methods in the 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 BUSINESS VALIDATIONS - POLICY

- Name should not be blank.
- Tenure should not be null.
- Amount should not be null.
- InterestRate should not be null.
- Coverage should not be blank.

4 REST ENDPOINTS

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

4.1 POLICY CONTROLLER

URL Exposed		Purpose
1. /policies		Creates a new policy
Http Method	POST	
Parameter	The policy data to be created must be received in the controller using @RequestBody.	
Return	PolicyDTO	
2. /policies/{id}		Gets a policy by id
Http Method	GET	
Parameter 1	Long (id)	
Return	PolicyDTO	
3. /policies		Get all policies
Http Method	GET	
Parameter	-	
Return	List<PolicyDTO>	
4. /policies/{id}		Updates a policy by id
Http Method	PUT	
Parameter 1	Long (id)	
	The policy data to be updated must be received in the controller using @RequestBody.	
Return	PolicyDTO	
5. /policies/{id}		Deletes a policy by id
Http Method	DELETE	
Parameter 1	Long (id)	
Return	-	
6. /policies/searchByName		Search all policies by name
Http Method	GET	
Request Param	String (name)	

Return	List<PolicyDTO>	
--------	-----------------	--

7. /policies/searchByCoverage		Searches all policies by coverage
Http Method	GET	
Request Param	String (coverage)	
Return	List<PolicyDTO>	

5 TEMPLATE CODE STRUCTURE

5.1 PACKAGE: COM.POLICYFINDER

Resources

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

5.2 PACKAGE: COM.POLICYFINDER.REPOSITORY

Resources

Class/Interface	Description	Status
PolicyRepository (interface)	<ul style="list-style-type: none"> Repository interface exposing CRUD functionality for Policy Entity. You can go ahead and add any custom methods as per requirements. You need to write a function to find all policies by name. You need to write a function to find all policies by coverage. 	Partially implemented.

5.3 PACKAGE: COM.POLICYFINDER.SERVICE

Resources

Class/Interface	Description	Status
PolicyService (interface)	<ul style="list-style-type: none">Interface to expose method signatures for policy related functionality.Do not modify, add or delete any method.	Already implemented.

5.4 PACKAGE: COM.POLICYFINDER.SERVICE.IMPL

Class/Interface	Description	Status
PolicyServiceImpl (class)	<ul style="list-style-type: none">Implements PolicyService.Contains template method implementation.Need to provide implementation for policy related functionalities.Do not modify, add or delete any method signature	To be implemented.

5.5 PACKAGE: COM.POLICYFINDER.CONTROLLER

Resources

Class/Interface	Description	Status
PolicyController (Class)	<ul style="list-style-type: none">Controller class to expose all rest-endpoints for policy related activities.May also contain local exception handler methods	To be implemented

5.6 PACKAGE: COM.POLICYFINDER.DTO

Resources

Class/Interface	Description	Status
PolicyDTO (Class)	<ul style="list-style-type: none">• Use appropriate annotations for validating attributes/fields of this class.	Partially implemented.

5.7 PACKAGE: COM.POLICYFINDER.ENTITY

Resources

Class/Interface	Description	Status
Policy (Class)	<ul style="list-style-type: none">• This class is partially implemented.• Annotate this class with proper annotation to declare it as an entity class with id as primary key.• Map this class with a policy table.• Generate the id using the IDENTITY strategy	Partially implemented.

5.8 PACKAGE: COM.POLICYFINDER.EXCEPTION

Class/Interface	Description	Status
ResourceNotFoundException (Class)	<ul style="list-style-type: none">• Custom Exception to be thrown when trying to fetch, update or delete the policy info which does not exist.• Need to create Exception Handler for same wherever needed (local or global)	Already implemented.

5.9 PROPERTIES FILES

Resources

Class/Interface	Description	Status
application.properties	<ul style="list-style-type: none">• This file is treated as the default properties file for this application.• You need to write properties to add actuator support.• You need to write property to expose all endpoints.• You need to write property to exclude /beans endpoint.• Add "profile.validate.data" property with value as "This is default profile".	Partially implemented.
application-qa.properties	<ul style="list-style-type: none">• This file is treated as the qa properties file for this application.• You need to write properties to add actuator support.• You need to write property to expose all endpoints.• You need to write property to exclude /beans endpoint.• Add "profile.validate.data" property with value as "This is qa profile".	Partially implemented.

6 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:
 - i. **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:
 - i. **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**
 - c. **mysql -u root -p**

- i. **The last command will ask for password which is 'pass@word1'**

13. **Mandatory:** Before final submission run the following command:

- i. **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.