

POLICY FINDER APPLICATION

IIHT

Time To Complete: 2 hr

CONTENTS

1	Problem Statement	3
2	Proposed Policy Finder Application Wireframe	4
2.1	Welcome Page	4
3	Business-Requirement	7
4	Validations	7
5	Constraints	8
6	Mandatory Assessment Guidelines	8

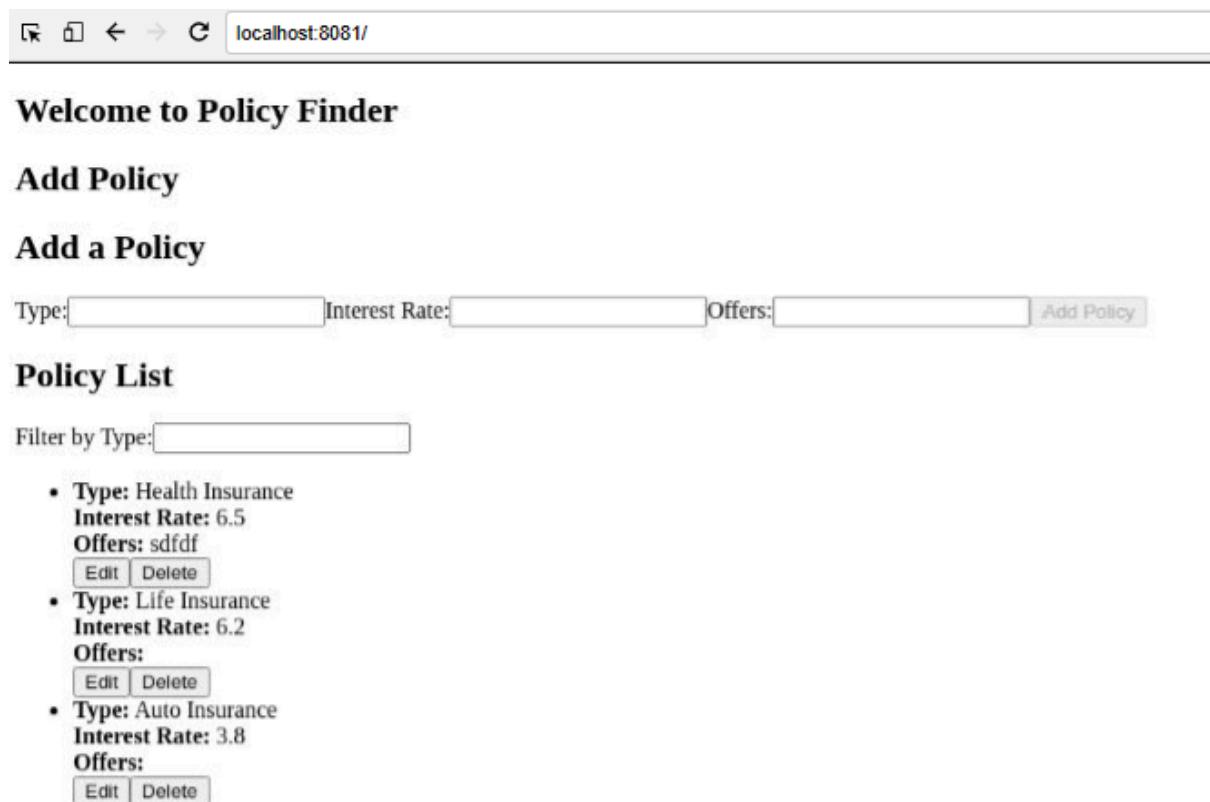
1 PROBLEM STATEMENT

"Policy Finder Application" is a Single Page Application (SPA) that empowers an insurance company to list all their different types of insurances.

2 PROPOSED POLICY FINDER APPLICATION WIREFRAME

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

2.1 WELCOME PAGE



The wireframe shows a web browser window with the address bar displaying 'localhost:8081/'. The page content includes a heading 'Welcome to Policy Finder', a sub-heading 'Add Policy', and a form with three input fields labeled 'Type:', 'Interest Rate:', and 'Offers:', followed by an 'Add Policy' button. Below this is a section titled 'Policy List' with a 'Filter by Type:' input field. A list of three policy items is displayed, each with a bullet point, a 'Type' label, an 'Interest Rate' value, an 'Offers' label, and 'Edit' and 'Delete' buttons.

Welcome to Policy Finder

Add Policy

Add a Policy


Type: Interest Rate: Offers:

Policy List

Filter by Type:

- Type: Health Insurance
Interest Rate: 6.5
Offers: sdfdf
- Type: Life Insurance
Interest Rate: 6.2
Offers:
- Type: Auto Insurance
Interest Rate: 3.8
Offers:

SCREEN SHOTS


localhost:8081/

Welcome to Policy Finder

Add Policy

Edit Policy

Type:
Interest Rate:
Offers:

Policy List

Filter by Type:

- Type:** Health Insurance
Interest Rate: 6.5
Offers: sdfdf
- Type:** Life Insurance
Interest Rate: 6.2
Offers:
- Type:** Auto Insurance
Interest Rate: 3.8
Offers:

3 BUSINESS-REQUIREMENT:

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

User Story #	User Story Name	User Story
US_01	Welcome Page	<p>As a user I should be able to visit the welcome page as default page.</p> <h3>Acceptance Criteria</h3> <ol style="list-style-type: none"> 1. Display a heading "Welcome to Policy Finder". 2. Provide a section labeled "Add Policy" / "Edit Policy" to create or edit a policy. 3. Display a "Policy List" below the form. 4. Support these operations:

		<ul style="list-style-type: none"> - View all policies (fetched from an API) - Add a new policy - Delete an existing policy - Edit and update a policy <h2>State & Props Overview</h2> <h3>App Component</h3> <p>State Used:</p> <ul style="list-style-type: none"> - policies: Type: array Purpose: Holds all the policies fetched from the backend. - editPolicy: Type: object null Purpose: Holds the selected policy for editing. <p>Responsibilities:</p> <ul style="list-style-type: none"> - Fetch policies from http://localhost:4000/policies using <code>useEffect</code>. - Add a new policy using <code>addPolicy(policy)</code>. - Delete a policy using <code>deletePolicy(policyId)</code>. - Update a policy using <code>updatePolicy(policy)</code>. - Pass necessary props to <code>AddPolicy</code> and <code>PolicyList</code> components. <h3>AddPolicy Component</h3> <p>Props Received:</p> <ul style="list-style-type: none"> - <code>addPolicy</code>: Function to add a new policy. - <code>editPolicy</code>: Current policy data if editing.
--	--	---

		<ul style="list-style-type: none"> - updatePolicy: Function to update the selected policy. <p>Responsibilities:</p> <ul style="list-style-type: none"> - Display input fields for creating/updating policy details. - If editPolicy is present, prefill form with existing data. - Call the appropriate function on form submission. <p>PolicyList Component</p> <p>Props Received:</p> <ul style="list-style-type: none"> - policies: Array of policy objects to display. - deletePolicy: Function to delete a selected policy. - setEditPolicy: Function to set a policy for editing. <p>Responsibilities:</p> <ul style="list-style-type: none"> - Loop through and display all policies. - Include buttons to delete or edit each policy. <p>HTML Structure</p> <p>App Component</p> <ul style="list-style-type: none"> - Use a top-level <div> to wrap the page. - Include the following headings: - <h2> with: "Welcome to Policy Finder" - <h2> with: "Add Policy" - <h2> with: "Policy List"
--	--	--

		<ul style="list-style-type: none"> - Below "Add Policy", render the AddPolicy form. - Below "Policy List", render the PolicyList. <h3>AddPolicy Component (UI Expectations)</h3> <ul style="list-style-type: none"> - A form containing fields like: <ul style="list-style-type: none"> - Policy Name - Policy Type - Premium Amount (or similar) - A submit button: <ul style="list-style-type: none"> - Labeled "Add Policy" for new - Labeled "Update Policy" if editing - Input values are controlled via state. - Calls the corresponding prop function (addPolicy or updatePolicy) on submit. <h3>PolicyList Component (UI Expectations)</h3> <ul style="list-style-type: none"> - Loop over policies and render each inside a container. - For each policy: <ul style="list-style-type: none"> - Show relevant details (e.g., name, type, premium) - Include a Delete button → calls deletePolicy(policy.id) - Include an Edit button → calls setEditPolicy(policy)
--	--	---

		<h2>AddPolicy</h2> <h3>Acceptance Criteria</h3> <ol style="list-style-type: none"> 1. Display a heading: “Add a Policy” when creating a new policy. “Edit Policy” when editing an existing one. 2. Render a form with input fields for: Type, Interest Rate, Offers. 3. Validate that all fields are filled before allowing submission. 4. On submit: <ul style="list-style-type: none"> - If editing, call the updatePolicy function. - If adding, call the addPolicy function. - Reset the form after submission. <h3>State & Props Overview</h3> <p>Props Received: addPolicy, editPolicy, updatePolicy.</p> <p>State: policy object with type, interestRate, offers.</p> <p>Behavior: useEffect to sync editPolicy to state, handleSubmit to manage add/update logic.</p> <h3>HTML Structure</h3> <ul style="list-style-type: none"> - <div> with heading <h2>: “Add a Policy” or “Edit Policy” - <form> with input fields for Type, Interest Rate, Offers - Submit <button>: 'Add Policy' or 'Update Policy'
--	--	--

		<h2>PolicyList</h2> <h3>Acceptance Criteria</h3> <ol style="list-style-type: none"> 1. Display a text field to filter policies by type. 2. Show a list of policies that match the filter. 3. For each policy, display Type, Interest Rate, Offers with Edit and Delete buttons. 4. Edit button populates AddPolicy. Delete button removes policy. <h3>State & Props Overview</h3> <p>Props: policies, deletePolicy, setEditPolicy.</p> <p>State: filters object with 'type'.</p> <p>Behavior: filter list based on type, handleEdit and handleDelete actions.</p> <h3>HTML Structure</h3> <ul style="list-style-type: none"> - <div> with a filter <label> and <input> for Type - to display policies - Each includes Type, Interest Rate, Offers, and Edit/Delete buttons
--	--	---

4 VALIDATIONS

- All required fields must be fulfilled with valid data.
- In the starting “Add Policy” button should be disabled.
- Only after validating fields, “Add Policy” button should be enabled.
- Once we click on “Add Policy”, a new policy must be added to the list.

5 CONSTRAINTS

- You should be able to press the “TAB” key and “SHIFT + TAB” to navigate from top field to bottom field and vice-versa.
- On clicking the “Add Policy” button, a new policy must be added with entered fields.
- “Add Policy” button will be disabled until all validations are fulfilled.
- If no policies are there, “No policies found” should be printed in in the PolicyList component.

6 MANDATORY ASSESSMENT GUIDELINES

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 editor Auto Saves the code.
4. 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.
5. 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.
6. This is a web-based application, to run the application 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

7. You can follow series of command to setup React 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:8080/8081 to open project in browser -> takes 2 to 3 min
 - c. npm run json-start -> As we are using a json server to mimic our db.json file as a database. So, this command is useful to start a json server.
 - a. npm run jest -> to run all test cases and see the summary. It takes 5 to 6 min to run.
 - d. npm run test -> to run all test cases. **It is mandatory to run this command before submission of workspace -> takes 5 to 6 min**
8. You may also run “npm run jest” while developing the solution to refactor the code to pass the test-cases.
9. Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on **“Submit Assessment”** after you are done with code.