

LOAN MANAGEMENT APPLICATION

IIHT

Time To Complete: 2 hr

CONTENTS

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

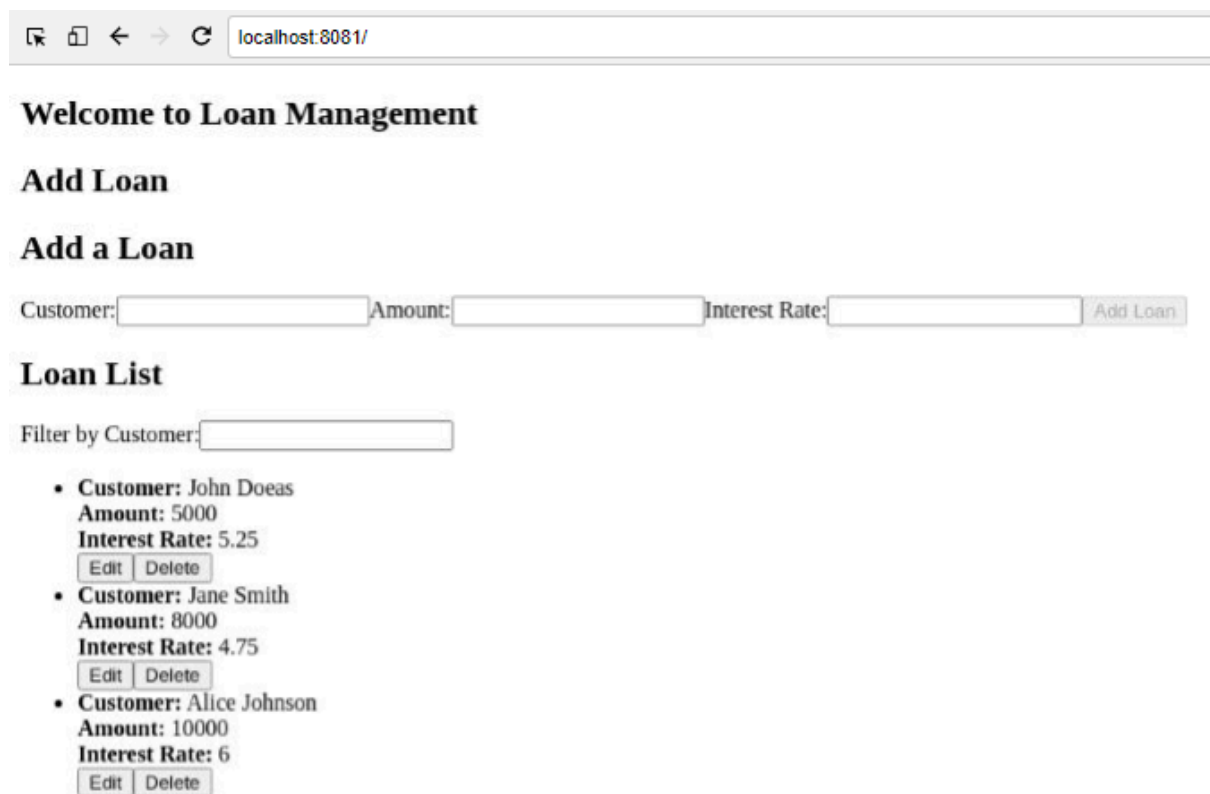
1 PROBLEM STATEMENT

"Loan Management Application" is a Single Page Application (SPA) that empowers banks to have a record of all loans taken by all different users with details.

2 PROPOSED LOAN MANAGEMENT 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 set to `localhost:8081/`. The page content includes a heading "Welcome to Loan Management", a section titled "Add Loan" with a form for Customer, Amount, and Interest Rate, and a "Loan List" section with a filter and a list of three loans.

Welcome to Loan Management

Add Loan

Add a Loan

Customer: Amount: Interest Rate:

Loan List

Filter by Customer:

- **Customer:** John Doeas
Amount: 5000
Interest Rate: 5.25
- **Customer:** Jane Smith
Amount: 8000
Interest Rate: 4.75
- **Customer:** Alice Johnson
Amount: 10000
Interest Rate: 6

3 BUSINESS-REQUIREMENT:

As an application developer, develop the Loan Management 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 - App Component</h3> <ol style="list-style-type: none">1. Display a heading: "Welcome to Loan Management" in h22. Provide a form section under the heading "Add Loan" in h2:<ul style="list-style-type: none">- Used to create or edit a loan.3. Display a section under the heading "Loan List" in h2:<ul style="list-style-type: none">- Show all existing loans.- Allow the user to edit or delete a loan.4. All actions should update the backend and reflect changes in the UI instantly. <h3>State & Props Overview</h3> <p>State Variables:</p> <ul style="list-style-type: none">- loans (array): Stores the list of all loans from the backend.- editLoan (object null): Stores the loan to be edited, or null if not editing. <h3>Functions & Responsibilities</h3> <ol style="list-style-type: none">1. fetchLoans: Fetches all loans on mount using axios GET request using "http://localhost:4000/loans".2. addLoan(loan): Adds a new loan using POST request and updates local state using "http://localhost:4000/loans".3. deleteLoan(loanId): Deletes loan using DELETE request and filters it from the UI using "http://localhost:4000/loans/\${loanId}".4. updateLoan(loan): Updates a loan using PUT request and refreshes

		<p>that item in state using “http://localhost:4000/loans/\${loan.id}”.</p> <h2>HTML Structure</h2> <p>App Component Layout:</p> <ul style="list-style-type: none"> - <div>: Wrap all content - <h2>: “Welcome to Loan Management” - <h2>: “Add Loan” followed by <LoanForm /> - <h2>: “Loan List” followed by <LoanList /> <h2>LoanForm Component (expected UI)</h2> <ul style="list-style-type: none"> - Form with inputs for loan details (amount, borrower, etc.) - Shows 'Update Loan' button if editing, else 'Add Loan' - Submit button disabled if form is incomplete - Calls addLoan or updateLoan on submit <h2>LoanList Component (expected UI)</h2> <ul style="list-style-type: none"> - Renders a list of loans using - Each shows Type, Interest, Borrower - Includes Delete and Edit buttons <h2>Dynamic Behavior</h2> <ul style="list-style-type: none"> - Initial Load: GET request populates the list - Add: POST request adds a new loan - Edit: Loads loan into form, PUT request on submit - Delete: DELETE request and state update
--	--	---

		<h1>LoanForm</h1> <h2>Acceptance Criteria</h2> <ol style="list-style-type: none"> 1. Display a heading: <ul style="list-style-type: none"> - “Add a Loan” when creating a new loan. - “Edit Loan” when updating an existing loan. 2. Render a form with the following fields: <ul style="list-style-type: none"> - Customer (text) - Amount (number) - Interest Rate (number) 3. The Submit button should: <ul style="list-style-type: none"> - Be disabled unless all fields are filled. - Call addLoan if it's a new loan. - Call updateLoan if editing an existing one. 4. After submission: <ul style="list-style-type: none"> - Reset the form fields. <h2>State & Props Overview</h2> <p>Props Received:</p> <ul style="list-style-type: none"> - addLoan: Function to handle adding a new loan. - editLoan: Object containing loan data to be edited. - updateLoan: Function to handle updating an existing loan. <p>State Variables:</p> <ul style="list-style-type: none"> - loan: <ul style="list-style-type: none"> - Type: object - Contains: <ul style="list-style-type: none"> - customer: string - amount: string - interestRate: string <p>Logic & Behavior:</p> <ul style="list-style-type: none"> - useEffect: <ul style="list-style-type: none"> - When editLoan changes: <ul style="list-style-type: none"> - If present, populate the form with existing loan values. - If not, reset the form to default empty fields. - isEditForm: <ul style="list-style-type: none"> - Boolean indicating if editing is active. - isFormIncomplete: <ul style="list-style-type: none"> - Used to disable the submit button if any input is empty. - handleSubmit(e):
--	--	--

		<ul style="list-style-type: none"> - Prevents default form submission. - Calls either addLoan or updateLoan. - Resets form fields after submission. <h2>HTML Structure</h2> <ul style="list-style-type: none"> - Use a <div> to wrap the form. - Add a <h2> with dynamic heading: <ul style="list-style-type: none"> - “Add a Loan” or “Edit Loan” <p>Inside a <form>:</p> <ul style="list-style-type: none"> - Customer input: <ul style="list-style-type: none"> - Use <label> for “Customer:” - Input field of type="text" bound to loan.customer - Amount input: <ul style="list-style-type: none"> - Use <label> for “Amount:” - Input field of type="number" bound to loan.amount - Interest Rate input: <ul style="list-style-type: none"> - Use <label> for “Interest Rate:” - Input field of type="number" bound to loan.interestRate - Submit button: <ul style="list-style-type: none"> - Type: submit - Label: “Add Loan” or “Update Loan” - Disabled if form is incomplete <h2>Dynamic Behavior</h2> <ul style="list-style-type: none"> - When the editLoan prop is received: <ul style="list-style-type: none"> - The form is prefilled with that loan’s data. - The submit button label changes to “Update Loan” - If no editLoan, the form is blank and adds a new loan on submit. - On submission: <ul style="list-style-type: none"> - Calls the correct function (addLoan or updateLoan) - Clears the input fields
--	--	--

		<h1>LoanList</h1> <h2>Acceptance Criteria</h2> <ul style="list-style-type: none"> · Display a filter input labeled “Filter by Customer” to filter loans by customer name. · Show a list of loans that match the filter. · For each loan, display: Customer name, Loan amount, Interest rate. · Include two buttons for each loan: Edit and Delete. · Show a fallback message if no loans match the filter: “No loans found”. <h2>State & Props Overview</h2> <p>Props Received:</p> <ul style="list-style-type: none"> - loans: An array of loan objects to display. - deleteLoan: Function to delete a loan by its id. - setEditLoan: Function to set the selected loan for editing. <p>State Used:</p> <ul style="list-style-type: none"> - filters: Object used to filter the loans ({ customer: " }) <h2>Functions & Responsibilities</h2> <p>handleDelete(id): Calls deleteLoan prop with the loan’s ID.</p> <p>handleEdit(loan): Passes the selected loan object to setEditLoan for editing.</p> <p>filteredLoans: A derived list of loans that match the customer filter text (case-insensitive).</p> <h2>HTML Structure</h2> <ul style="list-style-type: none"> · Use a top-level <div> to wrap everything. · Add a filtering section: · Use a <label> and an <input>:
--	--	--

		<ul style="list-style-type: none"> · Label: “Filter by Customer.” · Input value tied to filters.customer · Display loan results using a element: · If loans match the filter: <ul style="list-style-type: none"> • Loop through filteredLoans and render each inside a • Use for labels like Customer, Amount, Interest Rate • Include Edit and Delete buttons - If no matching loans: <ul style="list-style-type: none"> • Render with message “No loans found” <p>Dynamic Behavior</p> <ul style="list-style-type: none"> · Typing into the filter input dynamically filters the loans list. · Clicking Edit triggers the LoanForm to switch to edit mode and fill in the selected loan's values. · Clicking Delete removes the loan both from the UI and the backend (handled in parent).
--	--	---

4 VALIDATIONS

- All required fields must be fulfilled with valid data.
- In the starting “Add Loan” button should be disabled.
- Only after validating fields, “Add Loan” button should be enabled.
- Once we click on “Add Loan”, a new loan 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 Loan” button, a new loan must be added with entered fields.
- “Add Loan” button will be disabled until all validations are fulfilled.

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. 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.
5. 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.
6. 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.
7. 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
8. 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
9. You may also run “`npm run jest`” while developing the solution to refactor the code to pass the test-cases.
10. 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.
11. 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.