System Requirements Specification Index

For

Tax Management Application

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

Table of Contents

Tax Management Application
System Requirements Specification

Project Abstract

In the world of financial management, there's a pressing need to modernize tax handling. The CFO of a leading financial institution, challenges a team of developers to create a Fullstack Tax Management Application.

Your task is to develop a digital solution that seamlessly manages tax calculations and related specifications, providing users with an intuitive platform for effective tax management.

BACKEND-JAVA

1. Problem Statement

The **Tax Management Application** is a Java-based RESTful Web API utilizing Spring Boot, with MySQL as the database. The application aims to provide a comprehensive platform for managing and organizing all tax related data for a company.

To build a robust backend system that effortlessly handles tax calculations. Here's what the developers need to accomplish:

Following is the requirement specification:

	Tax Management Application
1	Tax
Tax Module Functionalities	
1	Get all taxes
2	Get tax by id
3	Create a new tax
4	Update a tax by id
5	Delete a tax by id

2. Assumptions, Dependencies, Risks / Constraints

2.1 Tax Constraints

- When fetching tax by id, if tax ID does not exist, the service method should throw a "Tax not found" message in the ResourceNotFoundException class.
- When updating a tax, if tax ID does not exist, the service method should throw a "Tax not found" message in the ResourceNotFoundException class.
- When removing a tax , if the tax ID does not exist, the service method should throw a "Tax not found" message in the ResourceNotFoundException class.

2.2 Common Constraints

- For all rest endpoints receiving @RequestBody, validation check must be done and must throw custom exceptions 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

- FormType should not be blank.
- FillingDate should not be null and must be current or past date.
- TotalTaxAmount should not be null and the value must be a positive number.
- UserID should not be null.

4. Database Operations

- Tax entity class must have a table name as "taxes" in the database.
- taxFormId field must have column name as "tax_form_id" in the database. It must be considered as id and should be generated through identity strategy.
- formType field must have a column name as "form_type" in the database and it must not be null.
- fillingDate field must have a column name as "filling_date" in the database and it must not be null.
- totalTaxAmount field must have a column name as "total_tax_amount" in the database and it must not be null.
- userId field must have a column name as "user_id" in the database and it must not be null.

5. Rest Endpoints

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

4.1 TaxController

URL Exposed			Purpose
1. /api/taxes			Fetches all the taxes
	Http Method	GET	
	Parameter	-	
	Return	List<TaxDTO>	
2. /api/taxes/{id}			Fetches a tax by id
	Http Method	GET	
	Parameter 1	Long (id)	
	Return	TaxDTO	
3. /api/taxes			Creates a new tax
	Http Method	POST **The tax data to be created should be received in @RequestBody**	
	Parameter	-	
	Return	TaxDTO	
4. /api/taxes/{id}			Updates a tax by id
	Http Method	PUT **The tax data to be updated should be received in @RequestBody**	
	Parameter 1	Long (id)	
	Return	TaxDTO	
5. /api/taxes/{id}			Deletes a tax by id
	Http Method	DELETE	
	Parameter 1	Long (id)	
	Return	-	

5. Template Code Structure

5.1 Package: com.taxmanagement

Resources

Class/Interface	Description	Status
TaxManagementApplication (Class)	This is the Spring Boot starter class of the application.	Already implemented.

5.2 Package: com.taxmanagement.repository

Resources

Class/Interface	Description	Status
TaxRepository (interface)	● Repository interface exposing CRUD functionality for tax Entity. ● You can go ahead and add any custom methods as per requirements.	Already implemented.

5.3 Package: com.taxmanagement.service

Resources

Class/Interface	Description	Status
TaxService (interface)	● Interface to expose method signatures for tax related functionality. ● Do not modify, add or delete any method.	Already implemented.

5.4 Package: com.taxmanagement.service.impl

Resources

Class/Interface	Description	Status
TaxServiceImpl (class)	● Implements TaxService. ● Contains template method implementation. ● Need to provide implementation for tax related functionalities. ● Do not modify, add or delete any method signature	To be implemented.

5.5 Package: com.taxmanagement.controller

Resources

Class/Interface	Description	Status
TaxController (Class)	● Controller class to expose all rest-endpoints for tax related activities. ● Should also contain local exception handler methods	To be implemented

5.6 Package: com.taxmanagement.dto

Resources

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

Response (Class)	• DTO created for response object.	Already implemented.

5.7 Package: com.taxmanagement.entity

Resources

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

5.8 Package: com.taxmanagement.exception

Resources

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

6. Method Descriptions

6.1 TaxServiceImpl Class - Method Descriptions

-> Add taxRepository as dependency.

Method	Task	Implementation Details
`createTax`	To create and save a new tax entry	- Use `modelMapper.map(taxDTO, Tax.class)` to convert DTO to entity. - Call `taxRepository.save(tax)` to persist the entity. - Convert the saved entity back to DTO using `modelMapper.map(savedTax, TaxDTO.class)`. - Return the resulting DTO.
`deleteTaxById`	To delete a tax entry by ID	- Use `taxRepository.existsById(id)` to check if tax exists. - If exists, call `taxRepository.deleteById(id)` and return `true`. - If not found, throw `ResourceNotFoundException` with message `"Tax not found"`.
`getAllTaxes`	To retrieve all tax records	- Fetch all taxes using `taxRepository.findAll()`. - Use Java Streams to map each `Tax` entity to `TaxDTO` using `modelMapper`. - Return the resulting list of DTOs.
`getTaxById`	To retrieve a single tax by ID	- Use `taxRepository.findById(id)`. - If found, map to `TaxDTO` and return. - If not found, throw `ResourceNotFoundException` with message `"Tax not found"`.
`updateTax`	To update an existing tax entry	- Get tax ID from `taxDTO.getTaxFormId()`.

		- Check if tax exists with `taxRepository.existsById(taxId)`. - If exists, map DTO to entity and save using `taxRepository.save(tax)`. - Map updated entity back to DTO and return. - If not found, throw `ResourceNotFoundException` with message `"Tax not found"`.

6.2 TaxController Class - Method Descriptions

-> Add taxService as dependency

Method	**Task**	**Implementation Details**
`createTax`	To implement logic to handle tax creation via API	- Request type: POST with URL `/api/taxes` - Method name: `createTax` and returns `ResponseEntity<TaxDTO>` - Use `@Valid @RequestBody` to accept and validate `TaxDTO` - Call `taxService.createTax(model)` - Return created tax with status `HttpStatus.CREATED`
`updateTax`	To implement logic for updating existing tax	- Request type: PUT with URL `/api/taxes/{id}` - Method name: `updateTax` returning `ResponseEntity<TaxDTO>` - Use `@Valid @RequestBody` for input - Call `taxService.updateTax(model)` - Return `HttpStatus.NOT_FOUND` if null, else `HttpStatus.OK` with updated object

`deleteTax`	To implement logic for deleting a tax by ID	- Request type: DELETE with URL `/api/taxes/{id}` - Method name: `deleteTax` returning `ResponseEntity<Void>` - Use `@PathVariable` to extract ID - Call `taxService.deleteTaxById(id)` - Return `HttpStatus.NOT_FOUND` if false, else `HttpStatus.NO_CONTENT`
`getTaxById`	To implement logic to retrieve a tax by ID	- Request type: GET with URL `/api/taxes/{id}` - Method name: `getTaxById` returning `ResponseEntity<TaxDTO>` - Use `@PathVariable` for ID input - Call `taxService.getTaxById(id)` - Return the tax with status `HttpStatus.OK`
`getAllTaxes`	To implement logic to retrieve all tax entries	- Request type: GET with URL `/api/taxes` - Method name: `getAllTaxes` returning `ResponseEntity<List<TaxDTO>>` - Call `taxService.getAllTaxes()` - Return the list with status `HttpStatus.OK`

FRONTEND-REACT SPA

1 PROBLEM STATEMENT

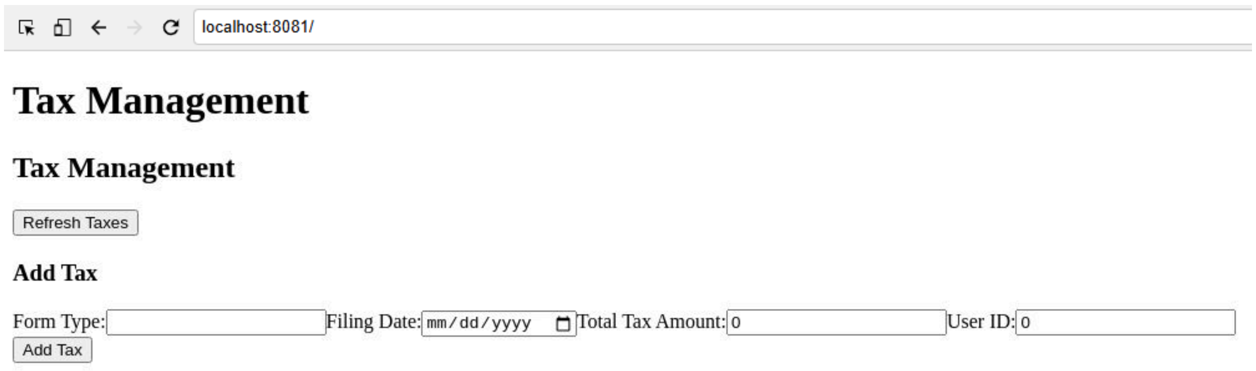
The **Tax Management Application** frontend is a Single Page Application (SPA) built using React. Here's what the frontend developers need to achieve:

The frontend should provide a user-friendly interface for users to manage tax-related tasks like: add tax details, update tax details, delete tax and get all taxes.

2 PROPOSED TAX MANAGEMENT APPLICATION WIREFRAME

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

2.1 HOME PAGE


localhost:8081/

Tax Management

Tax Management

Refresh Taxes

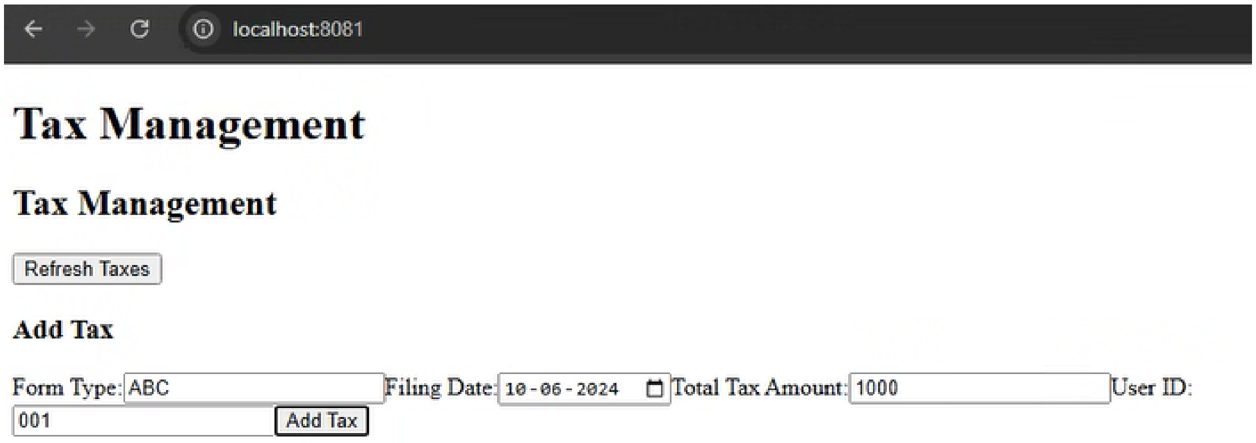
Add Tax

Form Type: Filing Date: mm/dd/yyyy Total Tax Amount: 0 User ID: 0

Add Tax

2.2 SCREENSHOTS

***** Add Tax*****


localhost:8081

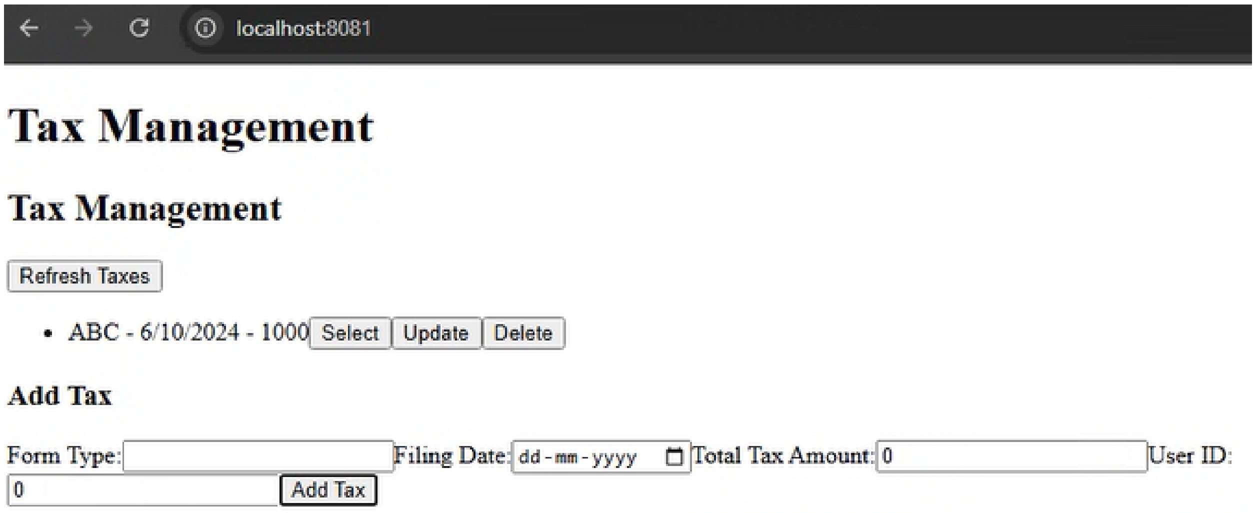
Tax Management

Tax Management

Refresh Taxes

Add Tax

Form Type: ABC Filing Date: 10-06-2024 Total Tax Amount: 1000 User ID: 001 Add Tax


localhost:8081

Tax Management

Tax Management

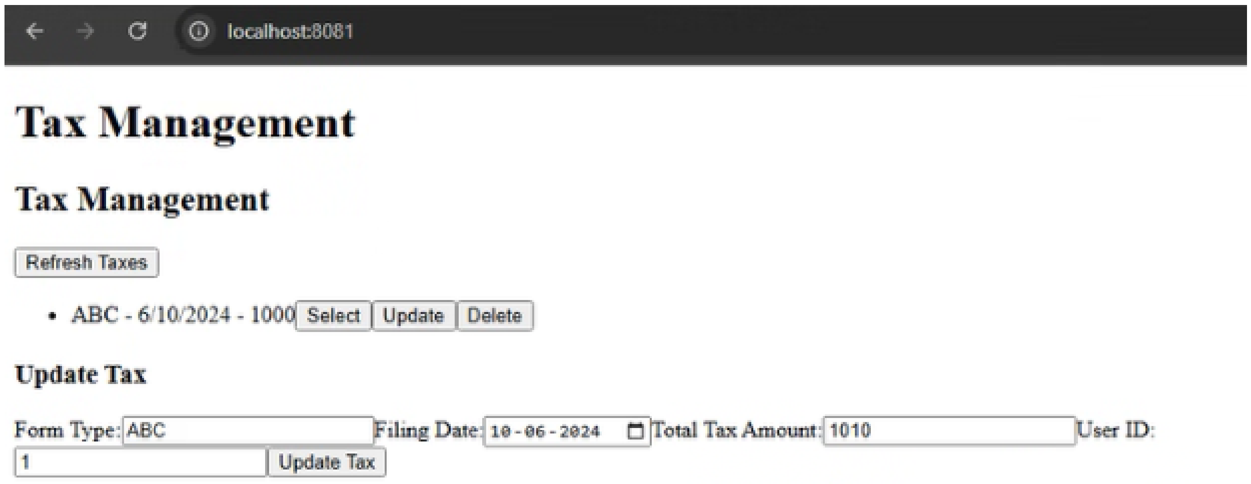
Refresh Taxes

- ABC - 6/10/2024 - 1000 Select Update Delete

Add Tax

Form Type: Filing Date: dd-mm-yyyy Total Tax Amount: 0 User ID: 0 Add Tax

***** Update Tax*****


localhost:8081

Tax Management

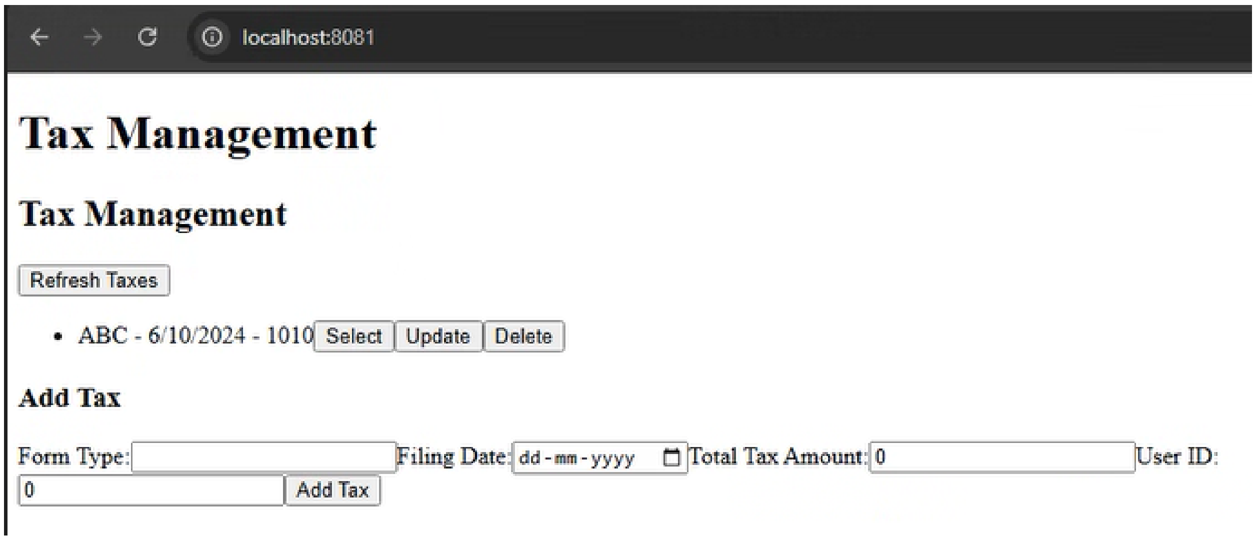
Tax Management

Refresh Taxes

- ABC - 6/10/2024 - 1000 Select Update Delete

Update Tax

Form Type: ABC Filing Date: 10-06-2024 Total Tax Amount: 1010 User ID: 1 Update Tax


localhost:8081

Tax Management

Tax Management

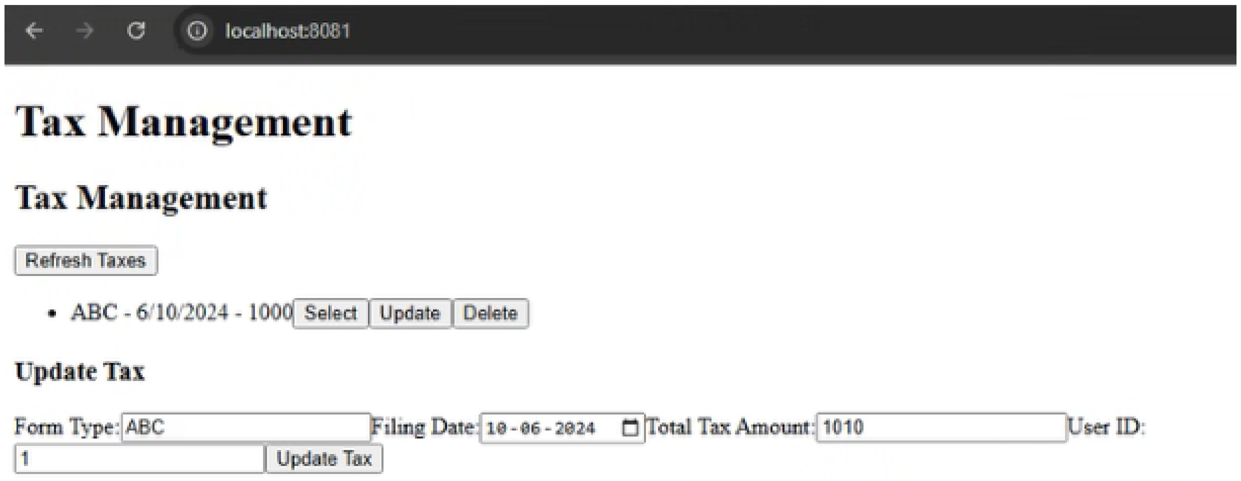
Refresh Taxes

- ABC - 6/10/2024 - 1010 Select Update Delete

Add Tax

Form Type: Filing Date: dd-mm-yyyy Total Tax Amount: 0 User ID: 0 Add Tax

***** Select Tax*****


localhost:8081

Tax Management

Tax Management

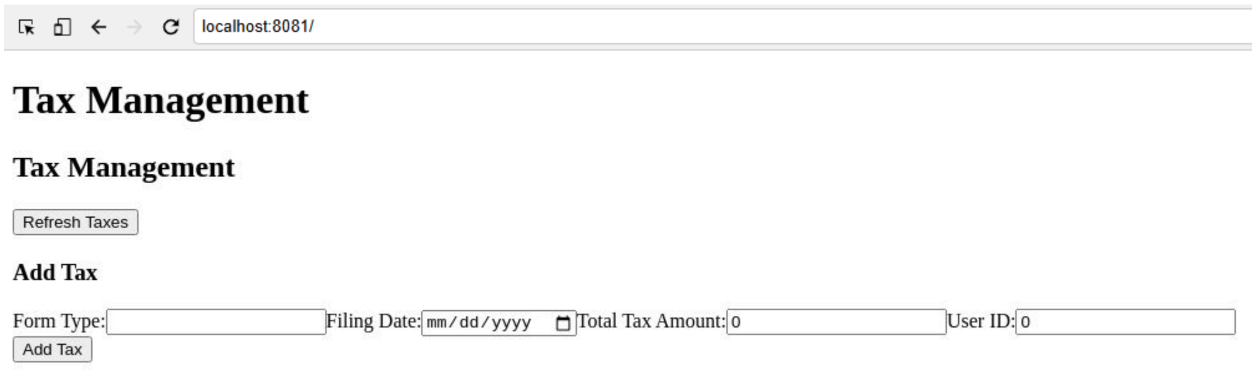
Refresh Taxes

- ABC - 6/10/2024 - 1000 Select Update Delete

Update Tax

Form Type: ABC Filing Date: 10-06-2024 Total Tax Amount: 1010 User ID: 1 Update Tax

***** Delete Tax*****


localhost:8081/

Tax Management

Tax Management

Refresh Taxes

Add Tax

Form Type: Filing Date: mm/dd/yyyy Total Tax Amount: 0 User ID: 0 Add Tax

3 BUSINESS-REQUIREMENT:

As an application developer, develop the Tax Management Application (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 welcome page as default page. **Acceptance Criteria - App Component** 1. Display a heading: “Tax Management” in `<h1>`. 2. Render the `TaxManagement` component below the heading. **HTML Structure** 1. Use a top-level `<div>` to wrap all content. 2. Inside it, add a heading `<h1>` with text: “Tax Management”. 3. Below the heading, include the TaxManagement component. **TaxManagement Component** **Acceptance Criteria** 1. Display a heading: “Tax Management” in `<h2>`. 2. Add a button labeled “Refresh Taxes” to reload the tax list. 3. Display a list of all existing tax entries. ● Each item should show: Form Type, Filing Date, Total Tax Amount. ● Include **Select**, **Update**, and **Delete** buttons for each item. 4. Provide a form under the heading: ● Display “Add Tax” when creating a new tax. ● Display “Update Tax” when editing an existing tax. 5. The form should contain the following fields: ● Form Type (text input) with label “Form Type:” ● Filing Date (date input) with label “Filling Date:”

- Total Tax Amount (number input) with label “Total Tax Amount:”
- User ID (number input) with label “User ID:”

6. The submit button should:
   - Show label “Add Tax” when adding a new entry.
   - Show label “Update Tax” when updating an existing one.
   - Trigger form submission.
   - Reset the form after successful submission.

State & Props Overview

State Variables:

- `taxes` (array): Stores the list of tax entries fetched from the backend.
- `selectedTax` (object): Holds the tax form currently being added or edited.

Functions & Responsibilities

1. loadTaxes:
Fetches all tax records from the backend using a GET request to `http://127.0.0.1:8081/taxmanagement/api/taxes`. Updates the `taxes` state with the response data.

2. addTax:
Sends a POST request to the backend with `selectedTax` data to create a new tax entry. On success, reloads the tax list and resets the form by updating `selectedTax` to its default empty state.

3. updateTax:
Sends a PUT request to the backend using `selectedTax.taxFormId` and the form data. After a successful update, it reloads the tax list and resets the form fields.

4. deleteTax:
Sends a DELETE request to remove a tax by ID. After deletion, it reloads the tax list and clears the form state.

5. showUpdateForm:
Accepts a tax form ID, finds the corresponding tax in the `taxes` list,

and sets `selectedTax` to that tax to populate the form for editing.

6. handleInputChange:
Triggered on input field changes. Updates the `selectedTax` object dynamically by field name and value.

7. createEmptyTax:
Returns a default tax object with initial values. Used to reset the form.

HTML Structure

- Use a top-level `<div>` to wrap everything.
- Add a heading `<h2>`: “Tax Management”.
- Add a button labeled “Refresh Taxes” to reload the list from backend.
- Display tax list using a `<ul>` element.
    - ➔ For each tax in the list:
        - ➢ Show **Form Type**, **Filing Date**, and **Total Tax Amount**.
        - ➢ Add three buttons: **Select**, **Update**, and **Delete**.
    - ➔ If the list is empty, show a `<li>` with message “No tax entries found”.
- Below the list, add a form section:
    - ➔ Add a heading `<h3>`:
        - ➢ Show “Add Tax” if creating a new entry.
        - ➢ Show “Update Tax” if editing an existing one.
    - ➔ Inside the form:
        - ➢ Add a `<label>` and an input for “Form Type”.
        - ➢ Add a `<label>` and an input of type “date” for “Filing Date”.
        - ➢ Add a `<label>` and an input of type “number” for “Total Tax Amount”.
        - ➢ Add a `<label>` and an input of type “number” for “User ID”.
        - ➢ Add a `<button>`:
            - ❖ Label: “Add Tax” or “Update Tax” depending on whether editing is active.
            - ❖ Button should trigger either

`addTax` or `updateTax`.

Dynamic Behavior

- On component mount, the list of taxes is loaded from the backend via `loadTaxes()`.
- When the user clicks “Refresh Taxes”, it re-fetches the list and updates the UI.
- Clicking “Select” on any tax simply sets it in the form but does not enable editing.
- Clicking “Update” sets the tax in the form and enables edit mode by updating `selectedTax`.
- If `selectedTax.taxFormId` is non-zero, the form shows “Update Tax” and calls `updateTax` on submission.
- If `selectedTax.taxFormId` is zero, the form shows “Add Tax” and calls `addTax` on submission.
- After each add or update, the form fields reset to default via `createEmptyTax()`.
- Clicking “Delete” removes the tax from backend and refreshes the UI list.

TaxService Class

Purpose: Handles all HTTP requests to the backend for tax-related operations. Centralizes API communication logic to keep the UI components clean.

Functions & Responsibilities

constructor():
- Sets the base URL for tax-related API endpoints: `http://127.0.0.1:8081/taxmanagement/api/taxes`.

getAllTaxes():
- Sends a GET request to retrieve all tax records from the backend.
- If successful, returns a JSON array of tax objects.
- If the request fails, logs and throws an error: “Error fetching taxes”.

getTaxById(id):

- Sends a GET request to fetch a single tax entry by its ID.
- URL: `http://127.0.0.1:8081/taxmanagement/api/taxes/{id}`
- Returns the tax object on success.
- Logs and throws an error if the fetch fails: “Error fetching tax”.

createTax(tax):

- Sends a POST request to create a new tax record.
- Includes tax data in the request body as JSON.
- Headers: `Content-Type: application/json`.
- On success, returns the newly created tax object.
- Logs and throws an error if creation fails: “Error creating tax”.

updateTax(id, tax):

- Sends a PUT request to update an existing tax record by ID.
- URL: `http://127.0.0.1:8081/taxmanagement/api/taxes/{id}`
- Includes updated tax data in the request body as JSON.
- Headers: `Content-Type: application/json`.
- On success, returns the updated tax object.
- Logs and throws an error if the update fails: “Error updating tax”.

deleteTax(id):

- Sends a DELETE request to remove a tax record by ID.
- URL: `http://127.0.0.1:8081/taxmanagement/api/taxes/{id}`
- Does not return any content on success.
- Logs and throws an error if deletion fails: “Error deleting tax”.

**** Kindly refer to the screenshots for any clarifications. ****

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. On command prompt, cd into your backend project folder (cd <Your-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. 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.
8. 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.
9. 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.
10. Default credentials for MySQL:
    a. Username: root
    b. Password: pass@word1
11. 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'

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

    mvn test

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. **On command prompt, cd into your frontend project folder (cd <Your-Project-folder>).**
4. **This is a web-based application, to run the application on a browser, use the internal browser in the environment.**
5. **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:8082 to open project in browser -> takes 2 to 3 min**
    c. **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**
6. **You may also run "npm run jest" while developing the solution to re-factor the code to pass the test-cases.**

7. **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.**