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. Dr. Smith, 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-DOTNET

1. Business-Requirement:

Tax Management Application is .Net Core web API 3.1 application integrated with MS SQL Server, where it refers to the professional management of various securities and assets to meet specific Tax goals for individuals, institutions, or organizations. This process includes the creation, updating, retrieval, and deletion of tax related properties.

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

	Tax Management
Modules	
1	Tax
Tax Module Functionalities	
1	Create an Tax
2	Update the existing Tax
3	Get an Tax by Id
4	Fetch all Insurance Policies
5	Delete an existing Tax

2. Assumptions, Dependencies, Risks / Constraints

2.1 Tax Constraints:

- While deleting the Tax, if Tax Id does not exist then the operation should throw a custom exception.
- While fetching the Tax details by id, if Tax id does not exist then the operation should throw a custom exception.

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 model 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

Tax Class Entities

- Tax Form Id (long) Not null, Key attribute.
- User Id (int) Not null.
- Form Type (string) is not null, min 3 and max 100 characters.
- Total Tax Amount (decimal) is not null.
- Filling Date (Date)

4. Considerations

- There is no roles in this application
- You can perform the following possible actions

Tax

5. Rest Endpoints

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

5.1 TaxController

URL Exposed		Purpose
/create-tax		Create Tax
Http Method	POST	
Parameter 1	Tax model	
Return	HTTP Response StatusCode	
/update-tax		Update a Tax
Http Method	PUT	
Parameter 1	Long Id	
Parameter 2	TaxViewModel model	
Return	HTTP Response StatusCode	
/get-all-taxes		Fetches the list of all Taxes
Http Method	GET	
Parameter 1	-	
Return	<IEnumerable<Tax >>	
/get-Tax-by-id?id={id}		Fetches the details of a Tax
Http Method	GET	
Parameter 1	Long (id)	
Return	<Tax>	
/delete-Tax?id={id}		Delete a Tax
Http Method	DELETE	
Parameter 1	Long (id)	
Return	HTTP Response StatusCode	

6. Template Code Structure

6.1 Package: TaxManagement

Resources

Names	Resource	Remarks	Status
Package Structure			
controller	TaxController	Controller class to expose all rest-endpoints for auction related activities.	Partially implemented
Startup.cs	Startup CS file	Contain all Services settings and SQL server Configuration.	Already Implemented
Properties	launchSettings.json file	All URL Setting for API	Already Implemented
	appsettings.json	Contain connection string for database	Already Implemented

6.2 Package: TaxManagement.BusinessLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
Interface	ITaxServices interface	Inside all these interface files contains all business validation logic functions.	Already implemented

Service	Tax Services CS file	Using this all class we are calling the Repository method and use it in the program and on the controller.	Partially implemented
Repository	ITax Repository Tax Repository (CS files and interfaces)	All these interfaces and class files contain all CRUD operation code for the database. Need to provide implementation for service related functionalities	Partially implemented
ViewModels	Tax ViewModel	Contain all view Domain entities for show and bind data. All the business validations must be implemented.	Already implemented

6.3 Package: TaxManagement.DataLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
DataLayer	TaxDBContext cs file	All database Connection,collection setting class	Already Implemented

6.4 Package: TaxManagement.Entities

Resources

Names	Resource	Remarks	Status
Package Structure			
Entities	Tax ,Response (CS files)	All Entities/Domain attribute are used for pass the data in controller and status entity to return response Annotate this class with proper annotation to declare it as an entity class with **Id** as primary key. Generate the **Id** using the **IDENTITY** strategy	Already implemented

7. Method Descriptions

1. TaxService: Method Descriptions

Method	Task	Implementation Details
`CreateTax`	To implement logic for creating a new Tax record.	- Input: Tax object - Call _taxRepository.CreateTax(tax) - Return the created Tax object
`DeleteTaxById`	To implement logic for deleting a Tax record by ID.	- Input: long id - Call _taxRepository.DeleteTaxById(id) - Return true if deletion is successful
`GetAllTaxes`	To implement logic for retrieving all Tax records.	- Call _taxRepository.GetAllTaxes() - Return the list of Tax records
`GetTaxById`	To implement logic for fetching a Tax record by its ID.	- Input: long id - Call _taxRepository.GetTaxById(id) - Return the Tax object if found
`UpdateTax`	To implement logic for updating a Tax record.	- Input: TaxViewModel model - Call _taxRepository.UpdateTax(model) - Return the updated Tax object

2. TaxRepository: Method Descriptions

Method	Task	Implementation Details
`CreateTax`	To implement logic for inserting a new Tax record into the database.	- Use try-catch block - In try: Use _dbContext.Taxes.AddAsync(tax) to add the new tax - Call SaveChangesAsync() to save the record - Return the created Tax object - In catch: throw the caught exception

`DeleteTaxById`	To implement logic for removing a Tax record using its ID.	- Use try-catch block - In try: Use LINQ to find the tax with matching TaxFormId - Use _dbContext.Remove() to remove the tax - Call SaveChanges() to commit deletion - Return true if deletion is successful - In catch: throw the caught exception
`GetAllTaxes`	To implement logic to retrieve the latest 10 Tax records.	- Use try-catch block - In try: Use _dbContext.Taxes.OrderByDescending(x => x.TaxFormId).Take(10).ToList() to fetch records - Return the list of tax records - In catch: throw the caught exception
`GetTaxById`	To implement logic for fetching a specific Tax by ID.	- Use try-catch block - In try: Use _dbContext.Taxes.FindAsync(id) to find the tax by ID - Return the Tax object if found - In catch: throw the caught exception
`UpdateTax`	To implement logic to update an existing Tax record.	- Use try-catch block - In try: • Use _dbContext.Taxes.FindAsync(model.TaxFormId) to fetch the existing record • Update fields: FormType, UserId, FilingDate, TotalTaxAmount • Use _dbContext.Taxes.Update(tax) to apply changes • Call SaveChangesAsync() to persist changes - Return the updated Tax object - In catch: throw the caught exception

3. TaxController: Method Descriptions

Method	**Task**	**Implementation Details**
`CreateTax`	To implement logic to create a new tax record with validation.	- Request type: POST, URL: /create-tax - Accept [FromBody] Tax model - Call _taxService.GetTaxById(model.TaxFormId) to check if a tax with the same ID already exists - If exists, return StatusCode 500 with message: 'Tax already exists!' - Else, call _taxService.CreateTax(model) to create a new tax

		record - If the result is null, return StatusCode 500 with message: 'Tax creation failed! Please check details and try again.' - If successful, return Ok with message: 'Tax created successfully!'
`UpdateTax`	To implement logic to update an existing tax record.	- Request type: PUT, URL: /update-tax - Accept [FromBody] TaxViewModel model - Call _taxService.UpdateTax(model) to update tax data - If result is null, return StatusCode 500 with message: 'Tax With Id = {model.TaxFormId} cannot be found' - If successful, return Ok with message: 'Tax updated successfully!'
`DeleteTax`	To implement logic to delete a tax record by ID with existence check.	- Request type: DELETE, URL: /delete-Tax?id={id} - Accept id as query parameter - Call _taxService.GetTaxById(id) to check if tax exists - If not found, return StatusCode 500 with message: 'Tax With Id = {id} cannot be found' - Else, call _taxService.DeleteTaxById(id) to delete tax - Return Ok with message: 'Tax deleted successfully!'
`GetTaxById`	To implement logic to retrieve a tax record by its ID.	- Request type: GET, URL: /get-Tax-by-id?id={id} - Accept id as query parameter - Call _taxService.GetTaxById(id) to fetch the tax record - If not found, return StatusCode 500 with message: 'Tax With Id = {id} cannot be found' - Else, return Ok with the tax record object
`GetAllTaxes`	To implement logic to retrieve all tax records.	- Request type: GET, URL: /get-all-taxes - Call _taxService.GetAllTaxes() to retrieve the list of all taxes - Return the list as the response

1. PROBLEM STATEMENT

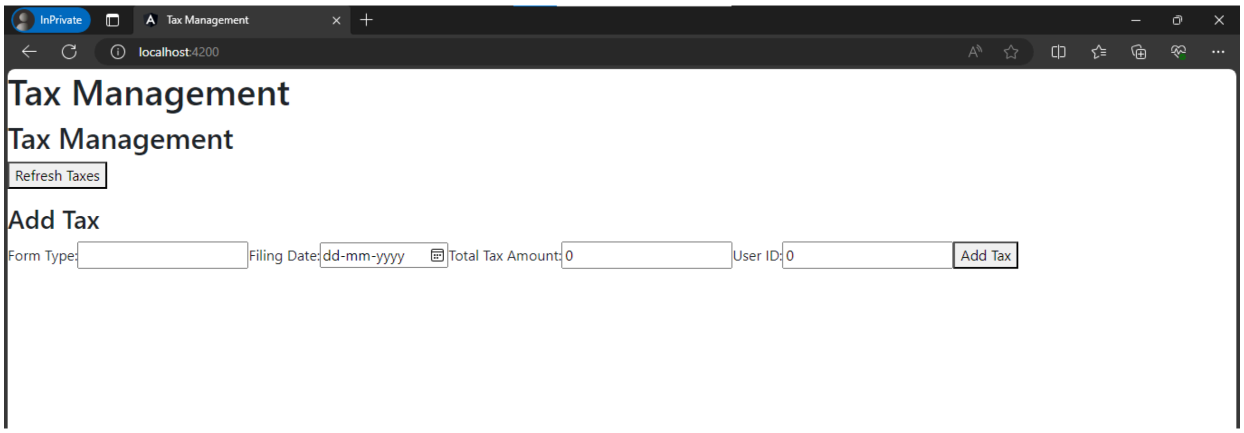
The **Tax Management Application** frontend is a Single Page Application (SPA) built using Angular. Here's what the frontend developers need to achieve:

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 WIREFRAME

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

2.1 Home Page



Tax Management

Tax Management

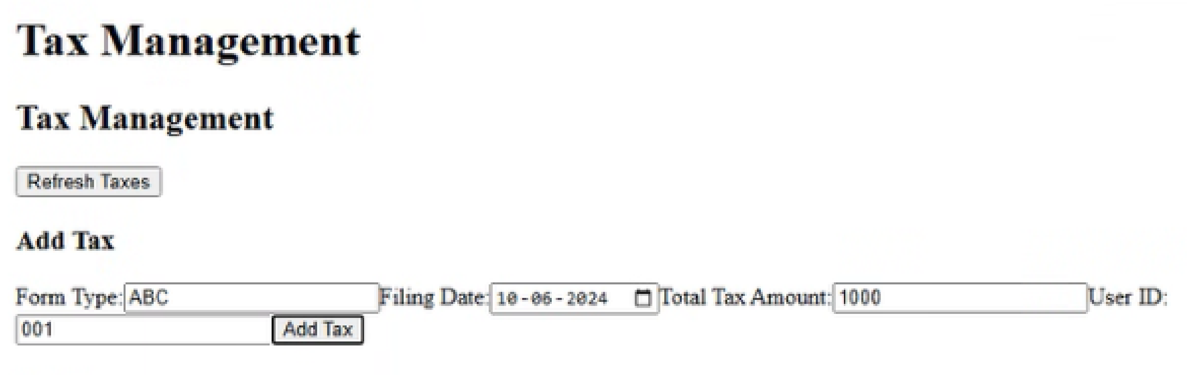
Refresh Taxes

Add Tax

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

2.2 Screenshots

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



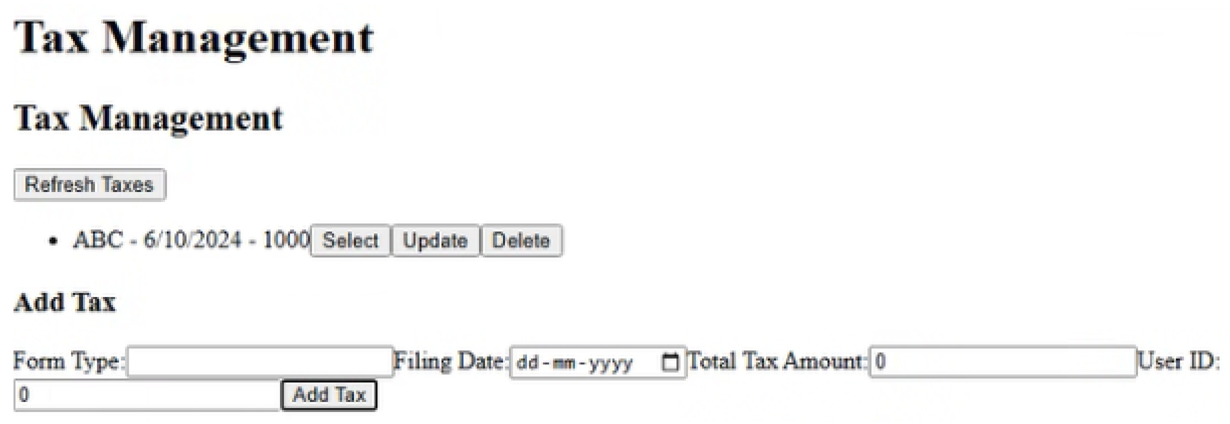
Tax Management

Tax Management

Refresh Taxes

Add Tax

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



Tax Management

Tax Management

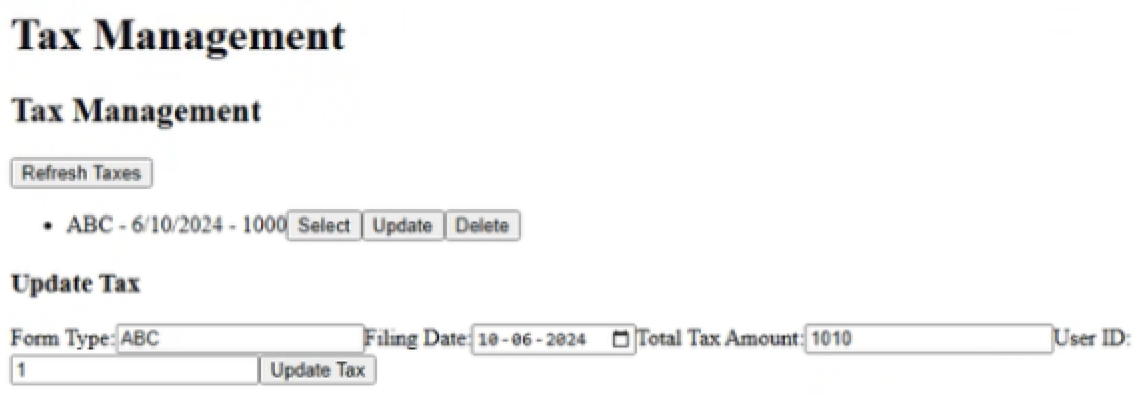
Refresh Taxes

- 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*****



Tax Management

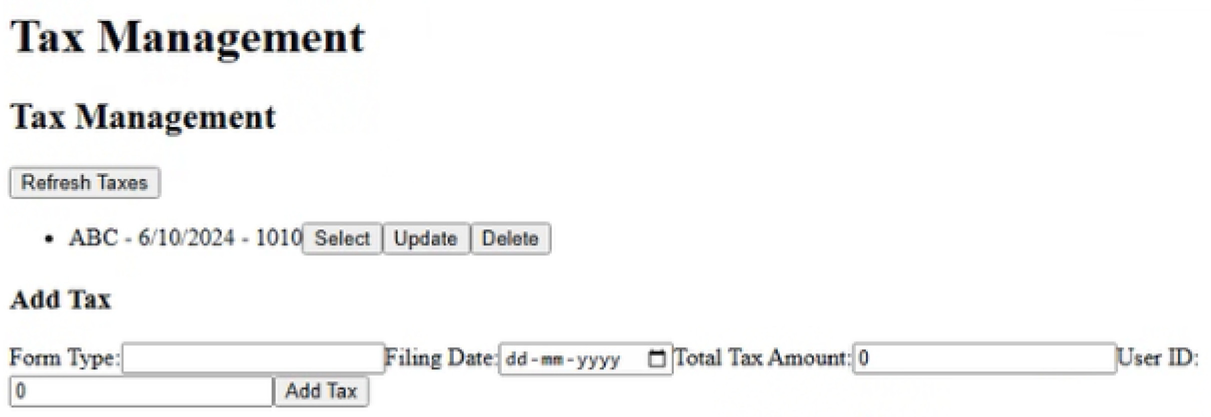
Tax Management

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



Tax Management

Tax Management

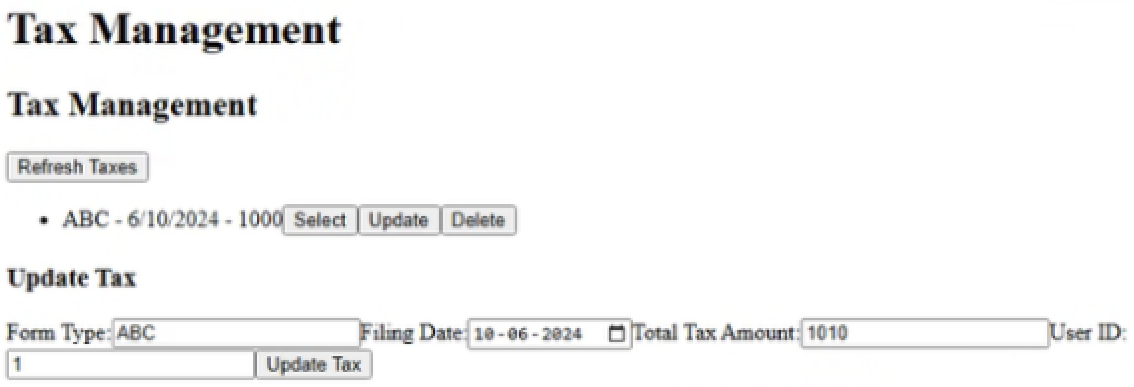
Refresh Taxes

- 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*****


Tax Management

Tax Management

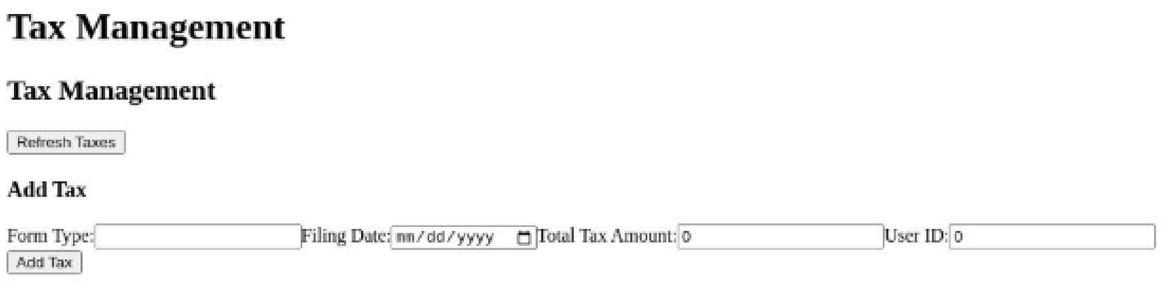
Refresh Taxes

- 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*****

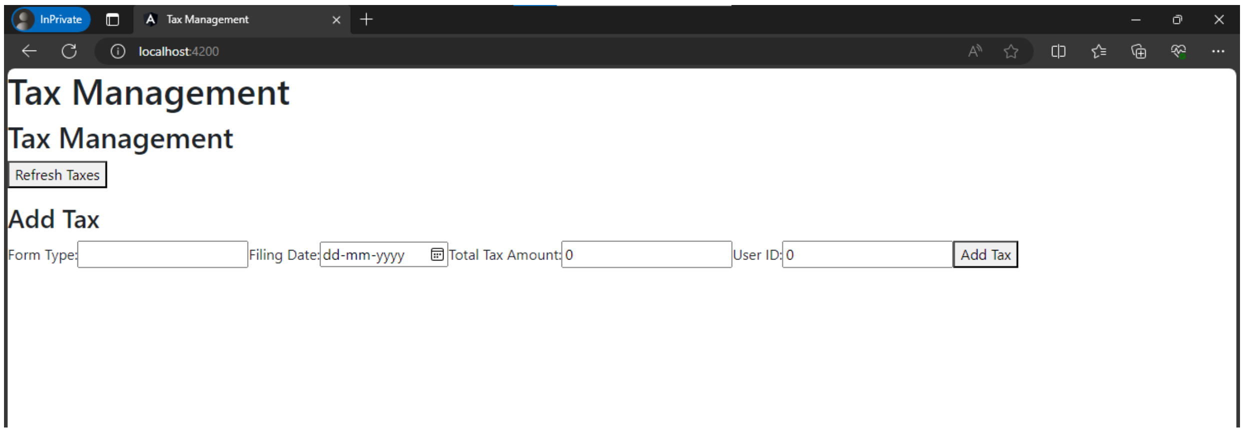

Tax Management

Tax Management

Refresh Taxes

Add Tax

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



Tax Management

Tax Management

Refresh Taxes

Add Tax

Form Type: Filing Date: dd-mm-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 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: As a user I should be able to visit the Home page as the default page. **Acceptance Criteria** **AppComponent** **Purpose** ● Acts as the shell of the application. ● Hosts the `TaxManagementComponent`. **HTML Structure** · Use a top-level `<div>` to wrap everything. · Add a `<h1>` with the text: “Tax Management”. · Render the `<app-tax-management>` component below the heading. **TaxManagementComponent** **Purpose** ● Handles all UI logic for listing, creating, updating, and deleting tax records. ● Handling form input. ● Uses a shared service (`TaxService`) to communicate with the backend.

		State Variables **taxes (array of Tax)** ● Stores the full list of tax records. **selectedTax (Tax object)** ● Represents the current form's tax data, either for creating or updating. **Functions & Responsibilities** **ngOnInit()** ● Calls `loadTaxes()` when the component initializes. **loadTaxes()** ● Sends a `GET` request to retrieve all tax entries. ● On success: updates the `taxes` list. ● On error: ○ Log to console: `console.error('Error loading taxes:', error);` **addTax()** ● Sends a `POST` request with the `selectedTax` object. ● On success: ○ Refreshes tax list via `loadTaxes()` ○ Resets form with `createEmptyTax()` ● On error: ○ Log to console: `console.error('Error adding tax:', error);` **showUpdateForm(id)** ● Finds the tax by ID from the `taxes` array. ● Prefills the form for editing by updating `selectedTax`. **updateTaxApi()**

- Sends a `PUT` request to update the selected tax.
- On success:
  - Refreshes tax list
  - Resets the form
- On error:
  - Log to console:
    ```
    console.error('Error updating tax:',
    error);
    ```

deleteTax(id)

- Sends a `DELETE` request by `taxFormId`.
- On success:
  - Refreshes the list
  - Clears form via `createEmptyTax()`
- On error:
  - Log to console:
    ```
    console.error('Error deleting tax:',
    error);
    ```

selectTax(tax)

- Assigns a selected tax to the form by cloning its values into `selectedTax`.

createEmptyTax()

- Returns a fresh object with default values to reset the form.

HTML Structure

· Use a top-level `<div>` to wrap the full component UI.

· Add a heading `<h2>`: “Tax Management”.

· Add a **Refresh Taxes** button:

- Click triggers `loadTaxes()` to re-fetch data.

· Display all taxes using an `<ul>` list:

- Loop using `*ngFor` over `taxes`

- For each item, show:
  · `formType`
  · `filingDate` (formatted using Angular `date` pipe)
  · `totalTaxAmount`
- Include 3 action buttons:
  · “Select” → sets the selected tax in the form
  · “Update” → fills form with selected tax for editing
  · “Delete” → removes the tax record

· Below the list, add a form with:

- Heading `<h3>`:
  · “Add Tax” if `taxFormId` is 0
  · “Update Tax” if editing an existing one

- Form Fields:
  · **Form Type** – text input
  · **Filing Date** – date input
  · **Total Tax Amount** – number input
  · **User ID** – number input

- Submit Button:
  · Label: “Add Tax” or “Update Tax” based on context
  · Click calls `addTax()` if `taxFormId` is 0, else `updateTaxApi()`

TaxService

Purpose

- Provides reusable HTTP methods to manage tax data from the backend.

Functions & Responsibilities

getAllTaxes()

- Sends a `GET` request to fetch all tax entries.
- Returns an observable `Tax[]`.

getTaxById(id)

- Sends a `GET` request to fetch one tax by ID.

- Returns an observable of a single `Tax`.

createTax(tax)

- Sends a `POST` request with a `Tax` object to create a new record.
- Returns the created object.

updateTax(id, tax)

- Sends a `PUT` request to update an existing tax record.
- Takes the `id` and updated `Tax` object.
- Returns: the updated tax

deleteTax(id)

- Sends a `DELETE` request to remove a tax record by ID.
- Returns: `void`

Dynamic Behavior

- On load: loadTaxes() fetches and displays all tax data.
- Form resets after every successful **Add**, **Update**, or **Delete**
- On **Add**:
  - New entry is sent to the backend.
  - UI refreshes with updated data.
- On **Update**:
  - Form is prefilled with selected tax.
  - PUT request sent on submit.
- On **Delete**:
  - Deletes record and refreshes UI.
- The form dynamically switches between "Add" and "Update" mode based on whether taxFormId is 0 or not.

**** 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 project folder (cd <Your-Project-folder>).**

4. **To connect SQL server from terminal:**
   (TaxManagement /sqlcmd -S localhost -U sa -P pass@word1)
   - **To create database from terminal -**
     ```
     1>  Create Database TaxDb
     2>  Go
     ```

5. **Steps to Apply Migration(Code first approach):**
   - **Press Ctrl+C to get back to command prompt**
   - **Run following command to apply migration-**
     (TaxManagement /dotnet-ef database update)

6. **To check whether migrations are applied from terminal:**
   (TaxManagement /sqlcmd -S localhost -U sa -P pass@word1)
   ```
   1> Use TaxDb
   2> Go
   1> Select * From __EFMigrationsHistory
   2> Go
   ```

7. **To build your project use command:**
   (TaxManagement /dotnet build)

8. **To launch your application, Run the following command to run the application:**
   (TaxManagement /dotnet run)

9. **This editor Auto Saves the code.**

10. **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.**

11. **To test web-based applications 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

12. **To run the test cases in CMD, Run the following command to test the application: (TaxManagement .Tests/dotnet test --logger "console;verbosity=detailed")**
    (You can run this command multiple times to identify the test case status,and refactor code to make maximum test cases passed before final submission)

13. **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.**

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. **Follow the steps below to install and use Node.js version 18.20.3 using nvm:**
   a. **Install nvm:**
   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash
   ```
   b. **Set up nvm environment:**
   ```
   export NVM_DIR="$([ -z "${XDG_CONFIG_HOME-}" ] && printf %s "${HOME}/.nvm" || printf %s "${XDG_CONFIG_HOME}/nvm")" && [ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
   ```
   c. **Verify nvm Installation:**
   ```
   command -v nvm
   ```
   d. **Install Node.js Version 18.20.3:**
   ```
   nvm install 18.20.3
   ```
   e. **Set the installed Node.js version as active:**
   ```
   nvm use 18.20.3
   ```

5. **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**