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. 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.

FRONTEND-ANGULAR SPA

1 PROBLEM STATEMENT

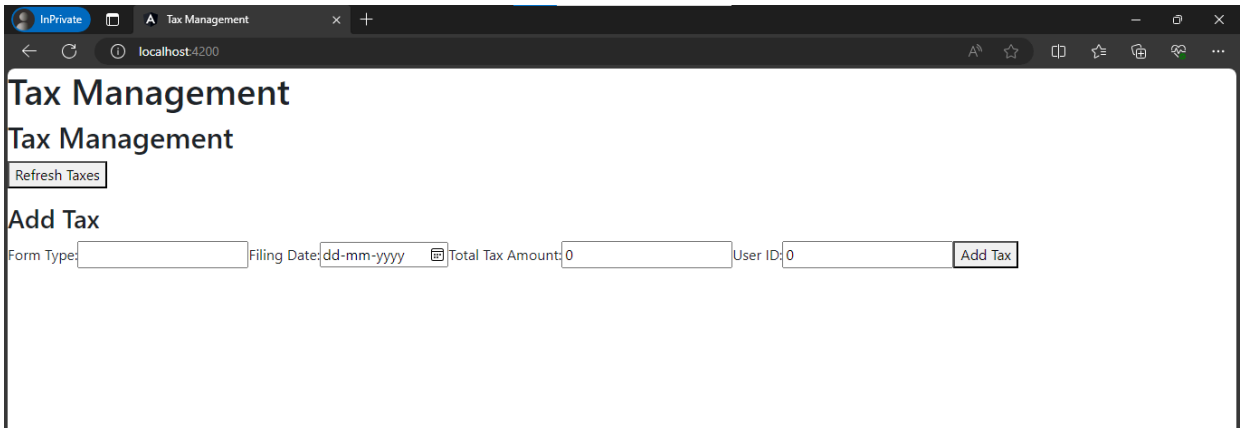
The **Tax Management Application** frontend is a Single Page Application (SPA) built using Angular. Here's what the frontend developers need to achieve:

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 APPLICATION WIREFRAME

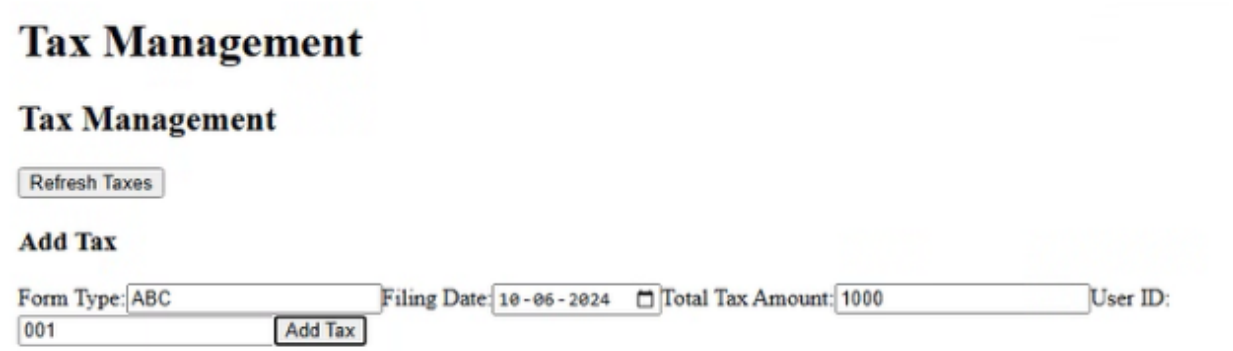
UI needs improvisation and modification as per given use case and to make test cases passed.

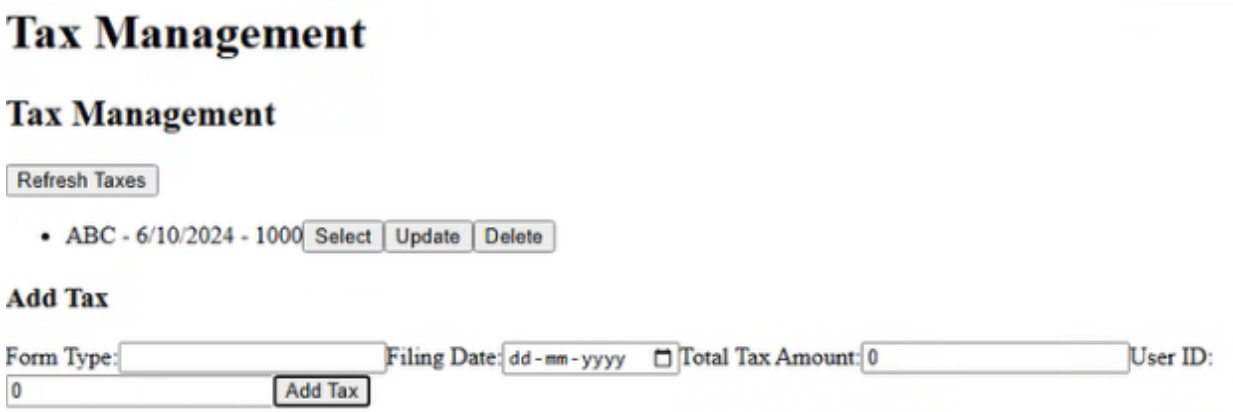
2.1 HOME PAGE



2.2 SCREENSHOTS

***** 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

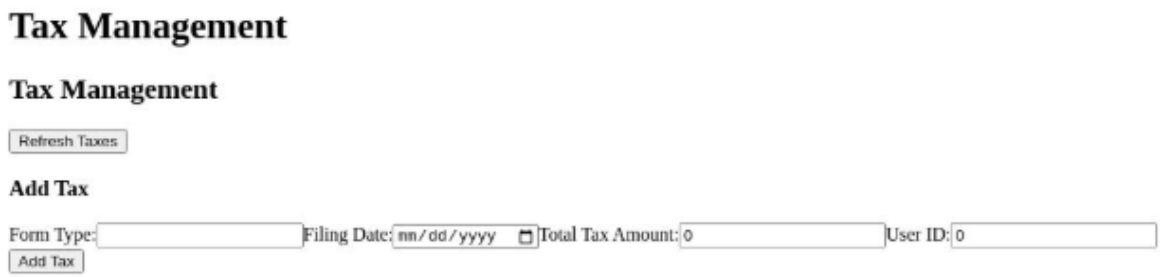
- ABC - 6/10/2024 - 1010 Select Update Delete

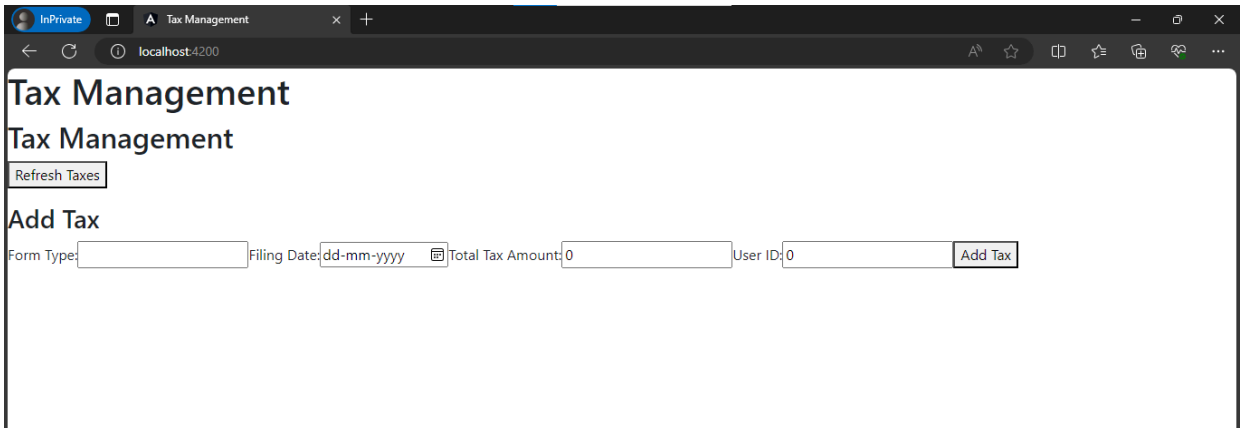
Add Tax

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

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

***** Delete 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: Acceptance criteria: 1. The page should display the title **"Tax Management"** in **h1**. 2. A **Refresh Taxes** button should be present to reload and display the latest tax entries. **> Tax List**: 1. Should show a list of all policies with “Update” & “Delete” button in each of the taxes. 2. The **Tax List** should display all tax records with the following details for each entry: ● Form Type ● Filing Date ● Total Tax Amount ● User ID 3. Each tax record should have **Select**, **Update**, and **Delete** buttons: ● **Select**: Loads the details of the selected tax record into the input fields for viewing. ● **Update**: Opens a form to edit the existing tax record. ● **Delete**: Removes the selected tax record from the list.

		> Add Tax: 1. The page should display the title **"Add Tax"** in **h3**. 2. Users should be able to add a new tax entry by filling in the following fields: ● **Form Type** (text input) ● **Filing Date** (date picker in `dd-mm-yyyy` format) ● **Total Tax Amount** (numeric input) ● **User ID** (numeric input) 3. There should be an **"Add Tax"** button, which should allow you to add only when all fields are filled with valid data. 4. Once the Add tax button is clicked, it should add the tax details to the tax list immediately. **> Select Tax**: 1. Users should be able to view an existing tax entry using the **Select** button. 2. Clicking the **Select** button next to a tax record should load the details into the input fields for viewing. **> Update Tax**: 1. Users should be able to update an existing tax entry by: ● Clicking the **Update** button next to the respective tax record. ● The details of the selected tax record should load into the **Update Tax** form. ● Editing the fields as necessary and clicking the **"Update Tax"** button to save changes. 2. The **"Update Tax"** button should only allow you to update only when all fields are filled and valid. 3. Once updated, the changes should reflect in the tax list immediately.

		> Delete Tax: 1. Users should be able to delete a tax record by clicking the **Delete** button next to the respective entry. 2. Once deleted, the tax record should be removed from the list, and the display should update automatically. **> Refresh Taxes**: 1. Clicking the **Refresh Taxes** button should reload the tax list and display the most up-to-date records without refreshing the entire page. **** 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. **cd into your backend 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. **This is a web-based application, to run the application on a browser, use the internal browser in the environment.**
4. **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**