System Requirements Specification Index

For

Tax Management App

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

Tax Management
System Requirements Specification

1. Business-Requirement:

1.1 Problem Statement:

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.

1.2 Following is the requirement specification:

	Tax Management
Modules	
1	Tax
Tax Module Functionalities	
1	Create an Tax
2	Update the existing Tax
3	Get an Tax by Id
6	Fetch all Insurance Policies
7	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

3.1 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

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

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 tax 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

Service/ 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.	Partially 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	Partially 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: /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: /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: /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: /tax?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: /taxes - Call _taxService.GetAllTaxes() to retrieve the list of all taxes - Return the list as the response

8. Execution Steps to Follow

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. **Install the .NET SDK 6.0 by running:**
   sudo apt install dotnet-sdk-6.0

   ***If it asks for the password, provide password : pass@word1**

   If it asks: Do you want to continue? [Y/n]

   Type Y and press Enter.

4. **On command prompt, cd into your project folder (cd <Your-Project-folder>).**

5. **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

6. **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)

7. **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
```

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

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

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

11. **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.**

12. **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

13. **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)

14. **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.**
