System Requirements Specification Index

For

Invoice 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

Invoice Management
System Requirements Specification

1. Business-Requirement:

1.1 Problem Statement:

Invoice Management Application is .Net Core web API 3.1 application integrated with MS SQL Server, where it refers to foundation for developing a comprehensive Invoice Management System with CRUD operations. It outlines the key Invoices, requirements, and expectations from the system. Developers can use this problem statement as a guide to creating a solution that meets the specific needs of Finance Corporation.

1.2 Following is the requirement specification:

	Invoice Management
Modules	
1	Invoice
Invoice Module Functionalities	
1	Create an Invoice
2	Update the existing Invoice
3	Get an Invoice by Id
6	Fetch all Invoices
7	Delete an existing Invoices

2. Assumptions, Dependencies, Risks / Constraints

2.1 Invoice Constraints:

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

- Invoice Id (Int) Not null, Key attribute.
- Invoice Number (string) Not null.
- Invoice Name (string) is not null, min 3 and max 100 characters.
- Amount (decimal) is not null.
- Start Date (date)
- End Date(date)
- Customer Id(Int)

4. Considerations

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

Invoice

5. Rest Endpoints

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

5.1 InvoiceController

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>/invoice
<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>Invoice model</td></tr><tr><td>Return</td><td>HTTP Response StatusCode</td></tr></table></td><td>Create Invoice</td></tr>
<tr><td>/invoice
<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>InvoiceViewModel model</td></tr><tr><td>Return</td><td>HTTP Response StatusCode</td></tr></table></td><td>Update a Invoice</td></tr>
<tr><td>/invoices
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>-</td></tr><tr><td>Return</td><td><IEnumerable<Invoice >></td></tr></table></td><td>Fetches the list of all Invoices</td></tr>
<tr><td>/invoice?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><Invoice></td></tr></table></td><td>Fetches the details of a Invoice</td></tr>
<tr><td>/invoice?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 Invoice</td></tr>
</table>

6. Template Code Structure

6.1 Package: InvoiceManagement

Resources

Names	Resource	Remarks	Status
Package Structure			
controller	InvoiveController	Controller class to expose all rest-endpoints for onvoice 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: InvoiceManagement.BusinessLayer

Resources

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

Service	InvoiceServices 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	IInvoice Repository Invoice 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	Invoice ViewModel	Contain all view Domain entities for show and bind data. All the business validations must be implemented.	Partially implemented

6.3 Package: InvoiceManagement.DataLayer

Resources

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

6.4 Package: InvoiceManagement.Entities

Resources

Names	Resource	Remarks	Status
Package Structure			
Entities	Invoice ,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. InvoiceService: Method Descriptions

Method	Task	Implementation Details
`CreateInvoice`	To create a new invoice via the repository.	- Accept an Invoice object as input - Call _InvoiceRepository.CreateInvoice(Invoice) - Return the result from repository
`DeleteInvoiceById`	To delete an invoice by its ID.	- Accept an integer ID as input - Call _InvoiceRepository.DeleteInvoiceById(id) - Return true if deletion succeeds
`GetAllInvoices`	To retrieve all invoice records.	- Call _InvoiceRepository.GetAllInvoices() - Return the list of invoice records

GetInvoiceById	To retrieve a single invoice by its ID.	- Accept an integer ID as input - Call _InvoiceRepository.GetInvoiceById(id) - Return the corresponding Invoice object
UpdateInvoice	To update an existing invoice using view model data.	- Accept InvoiceViewModel object as input - Call _InvoiceRepository.UpdateInvoice(model) - Return the updated Invoice object

2. InvoiceRepository: Method Descriptions

Method	Task	Implementation Details
CreateInvoice	To insert a new invoice record into the database.	- Use try-catch block - In try: Use _dbContext.Invoices.AddAsync(Invoice) to add the record - Call SaveChangesAsync() to persist - Return the created Invoice object - In catch: throw the exception
DeleteInvoiceById	To delete an invoice record by its ID.	- Use try-catch block - In try: Use LINQ to fetch invoice by InvoiceId - Call _dbContext.Remove() and SaveChanges() to delete - Return true if successful - In catch: throw the exception
GetAllInvoices	To fetch the latest 10 invoice records.	- Use try-catch block - In try: Use _dbContext.Invoices.OrderByDescending(x => x.InvoiceId).Take(10).ToList() - Return the list of invoices - In catch: throw the exception
GetInvoiceById	To retrieve an invoice by its ID.	- Use try-catch block - In try: Use _dbContext.Invoices.FindAsync(id) - Return the matching Invoice - In catch: throw the exception
UpdateInvoice	To update an existing invoice	- Use try-catch block - In try: • Use _dbContext.Invoices.FindAsync(model.InvoiceId) to

	record using a view model.	fetch the record • Update necessary fields in the fetched entity • Use _dbContext.Invoices.Update(entity) and SaveChangesAsync() - Return the updated Invoice - In catch: throw the exception

3. InvoiceController: Method Descriptions

- **Make sure that you add correct dependencies in the controller before working on below method logics.**

Method	Task	Implementation Details
`CreateInvoice`	To implement logic to create a new invoice record with validation.	- Request type: POST, URL: /invoice - Accept [FromBody] Invoice model - Call _InvoiceService.GetInvoiceById(model.InvoiceId) to check if the invoice exists - If exists, return StatusCode 500 with message: 'Invoice already exists!' - Else, call _InvoiceService.CreateInvoice(model) - If result is null, return StatusCode 500 with message: 'Invoice creation failed! Please check details and try again.' - Return Ok with message: 'Invoice created successfully!'
`UpdateInvoice`	To implement logic to update an existing invoice record.	- Request type: PUT, URL: /invoice - Accept [FromBody] InvoiceViewModel model - Call _InvoiceService.UpdateInvoice(model) - If result is null, return StatusCode 500 with message: 'Invoice With Id = {model.InvoiceId} cannot be found' - Else, return Ok with message: 'Invoice updated successfully!'
`DeleteInvoice`	To implement logic to delete an invoice record by ID.	- Request type: DELETE, URL: /invoice?id={id} - Accept id as query parameter - Call _InvoiceService.GetInvoiceById(id) to verify existence - If not found, return StatusCode 500 with message: 'Invoice With Id = {id} cannot be found' - Else, call _InvoiceService.DeleteInvoiceById(id) - Return Ok with message: 'Invoice deleted successfully!'

`GetInvoiceById`	To fetch a specific invoice record using its ID.	- Request type: GET, URL: /invoice?id={id} - Accept id as query parameter - Call _InvoiceService.GetInvoiceById(id) - If not found, return StatusCode 500 with message: 'Invoice With Id = {id} cannot be found' - Else, return Ok with the invoice object
`GetAllInvoices`	To implement logic to fetch all invoice records.	- Request type: GET, URL: /invoices - Call _InvoiceService.GetAllInvoices() - Return the list of all invoices

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

4. **To connect SQL server from terminal:**
   (InvoiceManagement /sqlcmd -S localhost -U sa -P pass@word1)
   - **To create database from terminal -**

     1> Create Database InvoiceDb
     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-**
     (InvoiceManagement /dotnet-ef database update)

6. **To check whether migrations are applied from terminal:**
   (InvoiceManagement /sqlcmd -S localhost -U sa -P pass@word1)

   1> Use InvoiceDb
   2> Go
   1> Select * From __EFMigrationsHistory
   2> Go

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

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

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.**

15. **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.**
