System Requirements Specification Index

For

Employee 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

Employee Management SYSTEM

System Requirements Specification

1.BUSINESS-REQUIREMENT:

1.1 PROBLEM STATEMENT:

Employee Management Application is a simple .Net Core 3.1 Razor application.

1.2 FOLLOWING IS THE REQUIREMENT SPECIFICATION:

	Employee Management Application
1	Employee
Employee Module Functionalities	
	1. Add a form with name and email inputs
	2. Add validations for above inputs and add onpost handler for submission
	3. On submit show the validations are firing and submission not happened if validations failed
	4. On submit show the validations are firing and submission not happened if validations failed
	5. Show a plain submission when validations arent failed
	6. Create a onget handler
	7. Create employee class with employeeid and salary property
	8. Add code inside onget handler to return 10 hardcoded employees details
	9. Add a button with caption getemployees which will invoke the onget handler to get the employee details
	10. Add razor page just to display the returned employees details in a tabular format

2. ASSUMPTIONS, DEPENDENCIES, RISKS / CONSTRAINTS

2.1 Employee Constraints

- The employee detail should have key as employee id.

2.2 Common Constraints

- For all operations, validation check must be done and must throw custom exception if data is invalid
- Do not change, add, remove any existing methods

2.4 Visitors can perform the follow actions

- Allows to display all employee information
-

2.5 ToolChain

- .NET Core 3.1, Razor Application

3. BUSINESS VALIDATIONS

3.1 Employee Class Entities

- EmployeeID:
  Data Type: int
  Validation: Since the EmployeeID is generated automatically and should not be provided by the user, no validation message is required for this property.

- Name:
  Data Type: string
  Validation: The Name field is required, meaning it must not be empty.
  Validation Message:
  "Name is required."
  “Email (string) is not null.”

- Email:
  Data Type: string
  Validation: The Email field is required, meaning it must not be empty, and it should be in a valid email address format.
  Validation Messages:
  "Email is required."
  "Invalid email address."

- Salary:
  Data Type: decimal
  Validation: The Salary field should have a numeric value greater than 0.00.
  Validation Message: "Salary must be greater than 0.00."

3.2 Employee Business Logic

1) EmployeeList():

- The EmployeeList method should return a list of Employee objects containing pre-defined sample data.
- The purpose of this method is to create a list of sample employees that can be used to seed the database when it is empty.

2) AddEmployee():

- The AddEmployee method should check if there are any existing employees in the database by querying the employee table.
- If the Employees table is empty (no employees exist), proceed with adding the sample employees.
- The method should use the EmployeeList() data to seed the database with initial employee records.

3)OnGet():

Initialization:

- The OnGet method should initialize the page by calling the AddEmployee function.
- This function is responsible for adding sample employee data to the database if no employees exist.
- It ensures that the database is seeded with some initial data, creating a baseline for the application to work with.

Retrieval of Employee List:

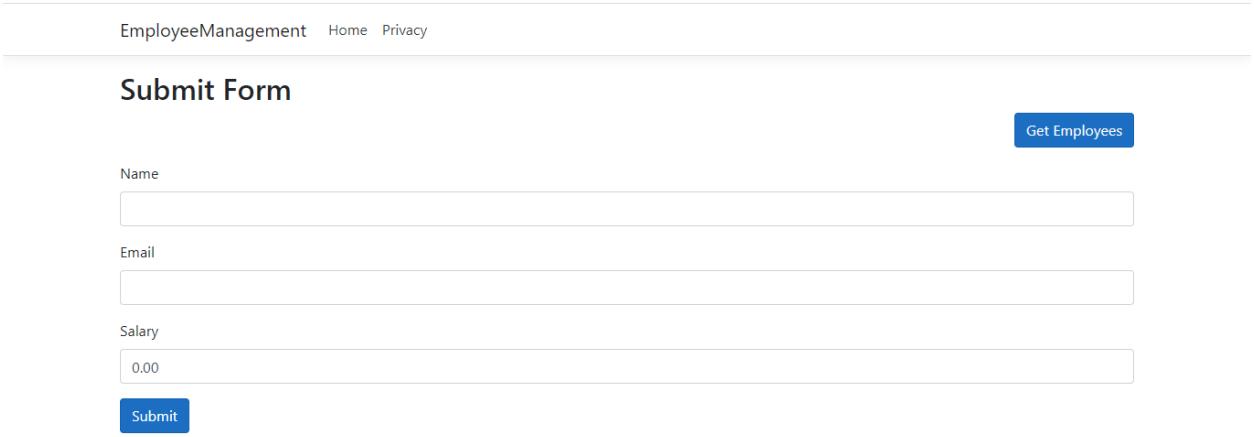
- The OnGet method should retrieve the list of employees from the database using the EmployeeDbContext.
- It should query the database to fetch all the employee records and store them in the Employees property.
- The list of employees will be displayed on the page to show the existing data.

4)OnPost():

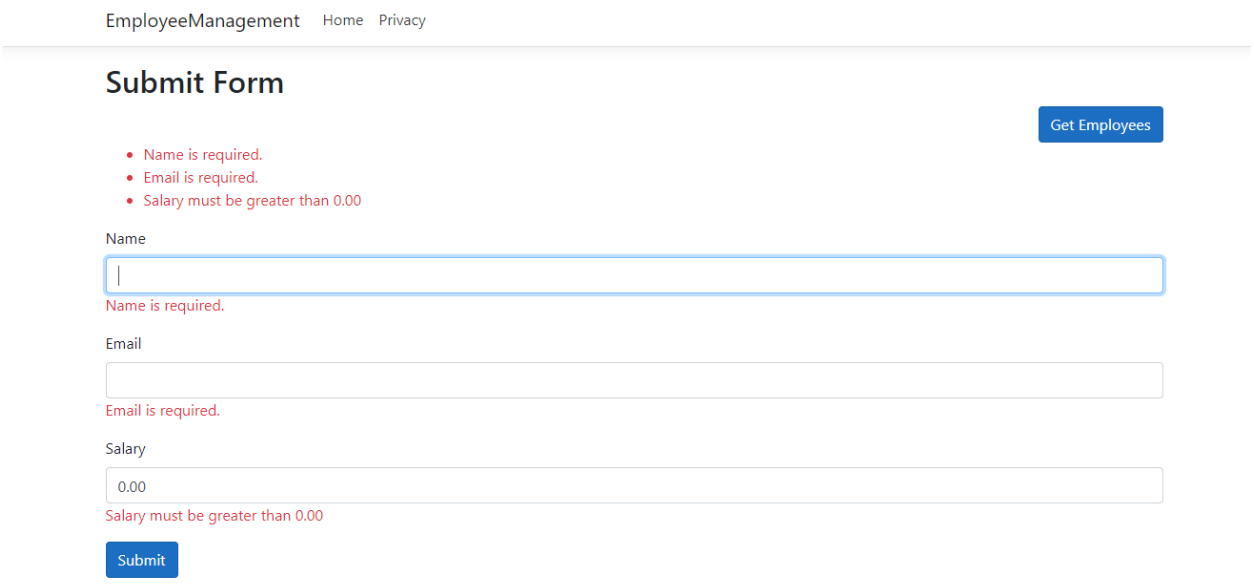
- The OnPost method should validate the form data submitted by the user to ensure its correctness and completeness.
- It should check for any data validation rules defined for the Name, Email, and Salary fields.
- If the form data passes validation, add the new employee to the list of employees.
- Update the list of employees with the newly added employee.
- If the form submission is successful, provide a response message to the user indicating success.
- Display a message like "Form submitted successfully."

4. WIREFRAME

1) Landing Page



2) On click of submit it should show validation summary.



3) On click of valid submission it should return submission content.

4) On click of get employee button it should display employee with salary.

5. TEMPLATE CODE STRUCTURE

5.1 Package: EmployeeManagement

Resources

Names	Resource	Remarks	Status
Package Structure			
Models	Employee.cs	All Entities/Domain attribute are used for pass the data .	Already implemented
Program.cs	Program CS file	Contain all Services settings	Already Implemented
Properties	launchSettings.json file	All URL Setting for API	Already Implemented
Pages	Index.cshtml	UI logic for employee related operations	Partially Implemented
Pages	Index.cshtml.cs	The index.cshtml.cs file is the code-behind file associated with the index.cshtml Razor Page. It contains the server-side logic for handling requests and rendering the view.	Partially Implemented

5.2 Package: EmployeeManagement.Tests

Resources

The EmployeeManagement.Tests project contains all test case classes and functions for code evaluation. Don't edit or change anything inside this project.

6. 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 build your project use command:**
   (EmployeeManagement /dotnet build)

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

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

7. **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.**

8. **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

9. **To run the test cases in CMD, Run the following command to test the application:**
   (EmployeeManagement/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)

10. **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.**

11. **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.**

12. **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.**
