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 RESTful Web API application.

1.2 FOLLOWING IS THE REQUIREMENT SPECIFICATION:

	Employee Management Application
1	Employee
Employee Module Functionalities	
	1. Define an endpoint to show its ending section read as "/employees"
	2. Create a model class as employee with required properties
	3. Hardcode 10 employee model collection as a return type of the above endpoint
	4.Fetch all employees

2. ASSUMPTIONS, DEPENDENCIES, RISKS / CONSTRAINTS

2.1 Employee Constraints

- The employee detail should have key as employee id.

2.2 Common Constraints

- For all rest endpoints receiving @RequestBody, validation check must be done and must throw custom exception if data is invalid
- 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

- **All business logic CRUD operations under repository class and write your business logic validation in Services class and related validation use proper user defined exceptions mentioned in above document.**
- **Controller must validate before processing any logic on the database.**

2.4 Visitors can perform the follow actions

- Allows to display all employee information
-

2.5 ToolChain

- .NET Core 3.1, RESTful Web API

3. BUSINESS VALIDATIONS

3.1 Employee Class Entities

- Employee Id (int) must be not null and unique
- Employee Name (string) is not null, min 3 and max 100 characters.
- Salary (string) is not null, should be greater than 0.

4. REST ENDPOINTS

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

4.1 EmployeeController

URL Exposed		Purpose
/employees		Fetches all employees
Http Method	GET	
Parameter 1	-	
Return	`<IEnumerable<Employee>>`	

5. TEMPLATE CODE STRUCTURE

5.1 Package: EmployeeManagement

Resources

Names	Resource	Remarks	Status
Package Structure			
controller	Employee Controller	Controller class to expose all rest-endpoints for employee related activities.	Partially implemented
Program.cs	Program CS file		Already Implemented
Properties	launchSettings.json file	All URL Setting for API	Already Implemented
Startup.cs	Startup.cs file	Contain all Services settings	Already Implemented

5.2 Package: EmployeeManagement.BusinessLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
Interface	IEmployeeService interface	Inside all these interface files contains all business validation logic functions.	Already Implemented
Service	EmployeeService 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	IEmployeeRepository EmployeeRepository CS file and interface.	All these interfaces and class files contain all CRUD operation code for the database. Need to provide implementation for service related functionalities	Partially Implemented

5.3 Package: EmployeeManagement.Entities

Resources

Names	Resource	Remarks	Status

Package Structure			
Entities	Employee CS file	All Entities/Domain attribute are used for pass the data in controller.	Already Implemented

5.4 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:**
   (CollegeManagement/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.**
