**Dot Net Phase 3 - EMS Full Stack App Requirement**

Phase-End Project Problem Statement



**Phase-End Project 3**

**EMS Full Stack App-Requirement**

**Prerequisites:** Basics of C#, MS-SQL Server, ASP.Net MVC, and Web API

**Case Study:**

Simplona Tech. Solutions require a Full Stack application EMS (Employee Management System) to maintain their employee profile details.

You need to create a Full Stack Application to maintain the details of employees using the multi-layered application described in the system overview.

**1. System Overview:**

Diagram

Description automatically generated

**A. Data Access Layer (DAL):** This layer will be a type of Class Library. This application will create a database using Entity Framework Code First Approach and implement Database functionalities such as Saving Employee details, Get All Employee Details, Get Employee by Code, Update Employee Details, and Delete Employee Details.

Consider the below Entity Model classes to create a database for the same.

1. Add entity class **DeptMaster** and given public properties

|  |  |
| --- | --- |
| **Property Name** | **Type** |
| DeptCode | int |
| DeptName | string |
| virtual EmpProfiles | ICollection<EmpProfile> |

1. Add entity class **EmpProfile** and the following public properties

|  |  |
| --- | --- |
| **Property Name** | **Type** |
| EmpCode | int |
| DateOfBirth | DateTime |
| EmpName | string |
| Email | string |
| DeptCode | Int |
| virtual DeptMaster | DeptMaster |

1. After creating the above entity classes, create the Context class by inheriting **DbContext** class
2. Add one more class by inheriting **DropCreateDatabaseIfModelChanges<ContextClass>** and add some default data into the **DeptMaster** table by overriding the Seed method
3. Write database functionalities for **EmpProfile** entity using Data Repository Pattern: Saving Employee details, Get All Employee Details, Get Employee by Code, Update Employee Details, and Delete Employee Details

**B. Business Logic Layer (BLL):** This layer will be a type of Class Library. This application will invoke functionalities from the DAL class.

1. Add a DAL library reference to this project
2. Add a class to invoke functionalities from the DAL class

**C. App Service Layer:** This layer will be a type of ASP.Net Web Application (Web API).  
This application contains RESTful services to consume functionalities from BLL Class.

1. Add Web API controller and write action methods to issue a GET and POST, PUT and DELETE request perform: Saving Employee details, Get All Employee Details, Get Employee by Code, Update Employee Details, and Delete Employee Details
2. Use attribute-based routing while implementing the above functionalities
3. Enable Swagger support for documentation to test this application as shown in the Output

**D. App UI Layer:** This layer will be the type of Angular or ReactJS Application that issues requests to consume RESTful services from the App Service Layer.

(This layer is out of scope for this system. You may implement this as part of your Angular or ReactJS Project).

**Sample input/output:  
  
1. To test all functionalities using Swagger (index page):**

Graphical user interface, application

Description automatically generated

1. **To test save functionality using Swagger:**

To save the name of the first employee:

Timeline

Description automatically generated with medium confidence

To save the name of the second employee:

Table

Description automatically generated with low confidence

1. **To test get all functionality using Swagger:**

Graphical user interface, text, application, email

Description automatically generated

1. **To test get by code functionality using Swagger:**

Graphical user interface, text, application, email

Description automatically generated

1. **To Test Update Functionality using Swagger:**

A picture containing timeline

Description automatically generated

After updating Employee Details, issue a Get by Code to confirm:

Graphical user interface, text, application, email

Description automatically generated

1. **To test delete functionality using Swagger:**

Graphical user interface

Description automatically generated with medium confidence

After Deleting Employee, issue a Get by Code to confirm:

Graphical user interface, text, application

Description automatically generated