**Employee Management Web Application**

**Practical 27:** Create a sample application with the following functionality in .Net MVC.

* Need to create Employee Management web application.
* Need to create one MVC project, one Web API project, and one class library for a database.
* The hierarchy of requests is that the cursor will go to the MVC controller using an AJAX call, from controller to Web API, and from Web API to the class library to save and manage the data.
* Need to create the following tables into a local database.
  + Employee Designation
    - Id
    - Designation
  + Employee Details
    - Id
    - Name
    - Designation ID (store designation Id in the column)
    - Profile Picture (Save picture into local machine and store that path into the database)
    - Salary
    - Date Of Birth
    - Email
    - Address

**Architecture Overview:**

* Employee Management Web Application is built using .NET MVC Framework, Which follows a **Three-tier Architecture**. The Application consists of main four components: An **MVC** project for the presentation layer, **Services Class Library** that represents the business logic and acts as an intermediary between controller and repository. It provides the necessary operation and functionality for manipulating employee data, the **Repository Class library** for interacting with the database, and the Employee models class library that represent the data management, validation, and manipulation

**The hierarchy of request flow is as follows**:

* User interface -> Controller ->Services layer -> Repository ->database

1. **User Interface(UI):**

* The user interacts with the Application through the UI, which typically consists of view, forms, and user input elements.
* In this case, the UI is implemented using ASP.NET MVC and consists of the views and actions defined in the “Employee Controller”.

1. **Controller:**

* The controller in MVC pattern the receive the user’s request performs necessary action s and determine the appropriate response.
* In this case, the controller handles the HTTP request related to employee management, such as adding, updating, deleting, and retrieving employees.

1. **Services Layer:**

* The services layer encapsulates the business logic of the application.
* It receives the request from the controller and processes them and interacts with the repository to perform data access operations

1. **Repository:**

* The repository is responsible for data access operation, interaction with the database, or any other data storage mechanism.
* It provides a method to perform CRUD operations on employee data.
* In this case, the Employee repository class handles the database interactions using the Dapper to execute the store procedure and retrieve and update employee data

1. **Database:**

* The actual database stores the employee data persistently.
* In this case, a SQL server database is used, and the connection string is provided in the repository class
* In summary, the request flows start from the UI, where the user interacts with the application the controller receives the user request, and delegates appropriate action to the services layer. The service layer, in turn, interact with the repository to perform data access operations. Finally, the repository communicates with the database to retrieve or update the employee data. The response flows back in the reverse order, from the repository to the service layer then the controller, and ultimately to the UI for rendering the appropriate view or response to the user.

**Project References:**

1. **EmployeeMvc**:

1. EmployeeModels

2. EmployeeServices

1. **EmployeeRepository:**
2. EmployeeModels
3. **EmployeeServices:**
4. EmployeeModels
5. EmployeeRepository

**Dependency Injection:**

Dependency injection in ASP.NET MVC is a Design Pattern and technique used to implement loose coupling and improve the maintainability and testability of an application .it allows the dependencies of a class to be resolved and provided by an external entity, Typically a DI container, instead of class itself is responsible for creating its dependency.

In ASP.NET MVC Di is commonly used to inject dependency into controllers, and services. Repositories, and other components of an application. The di container takes care of creating and managing instances of these dependencies and ensures that they are available when needed

**To Implement DI in ASP.NET MVC, you follow these steps:**

1. Define The Dependencies: identify the dependencies that a class requires to perform its tasks. These dependencies can be other classes, interfaces, or services.
2. Configure The DI Container: Configure to DI container to map the interfaces or abstract classes to their concrete implementation. This mapping tells the container which implementation to provide when a dependency is requested.
3. Specify dependencies in Contractors: Declare the dependency as a constructor parameter in classes that require them. This allows the DI container to inject the appropriate dependencies when an instance of the class is created
4. Use the dependencies: Access and use the injected dependencies within the class to perform the required operations.
5. Resolve Dependencies: Let The DI Container handle the creation and injection of dependency when an instance of the class is needed. The container is automatic resolve the dependencies based on the configured mappings.

**In These ASP.NET MVC Projects, we use the Unity DI Container**

**Dapper:**

1. **What is dapper and how to dapper in ASP.NET MVC?**

Dapper usually refers to the Dapper ORM (Object-Relational-Mapper) framework for database access in .NET applications it is the same as the entity framework.

Add dapper in your project install Nuget package=Dapper.

**Unity:**

Unity refers to the unity dependency injection (DI) framework.

Install Unity in your project: install unity nugget packages

Configure Unity Container: configure in **global.asax.cs** file in the **Application\_start** method

**Project on GitHub:**

git@github.com:jigar-prajapati1/EmployeeMamagementWebApplication.git