**.Net Basic Project With Dapper**

Here’s a step-by-step guide to creating a simple project in ASP.NET MVC using Dapper ORM with SQL Server. This document will help you set up the project, explain each line of code, and guide you through database interaction using Dapper. I'll break down everything into manageable steps with explanations to help you understand the process.

**Step 1: Set up Your ASP.NET MVC Project**

1. **Open Visual Studio**: Launch Visual Studio on your system.
2. **Create a new ASP.NET MVC project**:
   * Go to **File** > **New** > **Project**.
   * Select **ASP.NET Web Application (.NET Framework)**.
   * Choose **MVC** as the template and name your project (e.g., MvcDapperProject).
   * Click **OK** to create the project.

**Step 2: Install Dapper**

To use Dapper, we need to install it via NuGet.

1. **Open NuGet Package Manager Console**:
   * Go to **Tools** > **NuGet Package Manager** > **Package Manager Console**.
2. **Install Dapper** by running the following command:

Install-Package Dapper

**Step 3: Create Your Database**

Set up a SQL Server database and a table to work with.

1. **Open SQL Server Management Studio** (SSMS).
2. **Create a new database**:

CREATE DATABASE EmployeeDB;

1. **Create a table for employees**:

USE EmployeeDB;

CREATE TABLE Employees (

Id INT PRIMARY KEY IDENTITY,

Name NVARCHAR(100),

Age INT,

Department NVARCHAR(100)

);

**Step 4: Create a Database Connection String**

1. **Open Web.config** and add your database connection string:
   * Find the <connectionStrings> section and update it to include your database connection string.

<connectionStrings>

<

add name="EmployeeDBConnection" connectionString="DataSource=YOUR\_SERVER\_NAME;Initial Catalog=EmployeeDB;Integrated Security=True" providerName="System.Data.SqlClient"

/>

</connectionStrings>

* + Replace YOUR\_SERVER\_NAME with your actual SQL Server name.

**Step 5: Create the Data Access Layer (DAL)**

We’ll now set up Dapper to access the database.

1. **Create a new folder named DAL (Data Access Layer)**:
   * Right-click on the project > **Add** > **New Folder** > Name it DAL.
2. **Create a DapperContext.cs file in the DAL folder**:
   * This class will handle the connection to the database.

using System;

using System.Data;

using System.Data.SqlClient;

using System.Configuration;

// DapperContext class manages the connection to the database

public class DapperContext {

private readonly string \_connectionString;

// Constructor to fetch the connection string from Web.config

public DapperContext() {

\_connectionString = ConfigurationManager.ConnectionStrings["EmployeeDBConnection"].ConnectionString;

}

// Method to open and return the database connection

public IDbConnection CreateConnection()

{

return new SqlConnection(\_connectionString); // Returning an open connection to SQL Server

}

}

**Step 6: Create the Model**

1. **Create a Models folder** and add a class Employee.cs to represent the Employee entity:

namespace MvcDapperProject.Models

{

// Model for Employee entity

public class Employee

{

public int Id { get; set; } // Primary Key

public string Name { get; set; } // Employee name

public int Age { get; set; } // Employee age

public string Department { get; set; } // Employee department

}

}

**Step 7: Create the Repository Layer**

The repository will interact with the database using Dapper.

1. **Create an EmployeeRepository.cs file in the DAL folder**:

using System.Collections.Generic;

using System.Linq;

using Dapper;

using MvcDapperProject.Models;

// Repository class for Employee-related database operations

public class EmployeeRepository

{

private readonly DapperContext \_context;

// Constructor to inject the Dapper context

public EmployeeRepository()

{

\_context = new DapperContext();

}

// Method to get all employees

public IEnumerable<Employee> GetAllEmployees()

{

// Query to select all employees

var query = "SELECT \* FROM Employees";

// Using the connection, execute the query using Dapper's Query method

using (var connection = \_context.CreateConnection())

{

var employees = connection.Query<Employee>(query).ToList(); // Fetching the list of employees

return employees;

}

}

// Method to insert a new employee

public void AddEmployee(Employee employee)

{

// SQL Insert statement

var query = "INSERT INTO Employees (Name, Age, Department) VALUES (@Name, @Age, @Department)";

// Execute the query to insert a new employee

using (var connection = \_context.CreateConnection())

{

connection.Execute(query, employee); // Executing the Insert query with employee data

}

}

}

**Step 8: Create the Controller**

Now, let’s create a controller to interact with our repository.

1. **Create a Controllers folder** and add an EmployeeController.cs:

using System.Web.Mvc;

using MvcDapperProject.DAL;

using MvcDapperProject.Models;

// Controller to manage employee-related views and actions

public class EmployeeController : Controller

{

private readonly EmployeeRepository \_repository;

// Constructor to initialize the repository

public EmployeeController()

{

\_repository = new EmployeeRepository();

}

// Action to display the list of employees

public ActionResult Index()

{

var employees = \_repository.GetAllEmployees(); // Fetching all employees

return View(employees); // Passing data to the view

}

// Action to display the form for adding a new employee

public ActionResult Create()

{

return View();

}

// Action to handle the form submission and insert a new employee

[HttpPost]

public ActionResult Create(Employee employee)

{

if (ModelState.IsValid)

{

\_repository.AddEmployee(employee); // Adding the new employee to the database

return RedirectToAction("Index");

}

return View(employee); // If validation fails, reload the form with data

}

}

**Step 9: Create the Views**

1. **Create Views for Index and Create** in Views/Employee.
2. **Create Index.cshtml**:

@model IEnumerable<MvcDapperProject.Models.Employee>

<h2>Employee List</h2>

<table class="table">

<thead>

<tr>

<th>Id</th>

<th>Name</th>

<th>Age</th>

<th>Department</th>

</tr>

</thead>

<tbody>

@foreach (var employee in Model)

{

<tr>

<td>@employee.Id</td>

<td>@employee.Name</td>

<td>@employee.Age</td>

<td>@employee.Department</td>

</tr>

}

</tbody>

</table>

<a href="@Url.Action("Create", "Employee")">Add New Employee</a>

1. **Create Create.cshtml**:

@model MvcDapperProject.Models.Employee

<h2>Add New Employee</h2>

@using (Html.BeginForm())

{

<div>

@Html.LabelFor(m => m.Name)

@Html.TextBoxFor(m => m.Name)

</div>

<div>

@Html.LabelFor(m => m.Age)

@Html.TextBoxFor(m => m.Age)

</div>

<div>

@Html.LabelFor(m => m.Department)

@Html.TextBoxFor(m => m.Department)

</div>

<input type="submit" value="Create" />

}

**Step 10: Run the Project**

Now that everything is set up:

1. **Run the project** by pressing F5 or clicking **Start** in Visual Studio.
2. Navigate to /Employee/Index to view the list of employees.
3. Use the "Add New Employee" button to add new employees.

**Summary of Key Concepts**

* **Dapper ORM**: Dapper is a lightweight ORM (Object-Relational Mapping) tool for .NET that helps map database rows to C# objects without the overhead of a full ORM like Entity Framework.
* **SQL Queries**: Dapper allows you to execute raw SQL queries directly.
* **Repository Pattern**: This pattern is used to separate the data access logic, improving code maintainability and testability.
* **Controller**: Handles incoming requests, interacts with the data (via repositories), and returns views to the user.

By following these steps, you’ve created a simple MVC application using Dapper with SQL Server, allowing you to practice the tech skills you've learned.