**CRUD with Dapper**

step-by-step guide to create a simple CRUD (Create, Read, Update, Delete) operation using .NET MVC, Dapper, and SQL Server.

**Prerequisites**

* Install Visual Studio
* Install SQL Server or use SQL Server Express
* Install Dapper via NuGet
* Basic knowledge of C#, MVC pattern, and SQL

**Step 1: Set up the Database**

Create a sample database and table in SQL Server.

CREATE DATABASE SampleDB;

USE SampleDB;

CREATE TABLE Employees (

Id INT PRIMARY KEY IDENTITY(1,1),

Name NVARCHAR(100),

Position NVARCHAR(100),

Salary DECIMAL(18, 2)

);

This SQL script creates an Employees table with three fields: Id, Name, Position, and Salary.

**Step 2: Create a new .NET MVC Project**

1. Open Visual Studio.
2. Create a new project:
   * Select **ASP.NET Web Application (.NET Framework)**.
   * Choose **MVC**.

This will set up a new MVC project with the necessary structure (Controllers, Models, and Views).

**Step 3: Install Dapper**

1. In **Solution Explorer**, right-click on your project and select **Manage NuGet Packages**.
2. Search for Dapper and install it.

**Step 4: Create the Database Connection**

In your project, locate the Web.config file and add the connection string inside the <configuration> section:

<connectionStrings>

<add name="DbConnection" connectionString="Data Source=.;Initial Catalog=SampleDB;Integrated Security=True" providerName="System.Data.SqlClient" />

</connectionStrings>

* Data Source=. means it will connect to your local SQL Server instance.
* Initial Catalog=SampleDB specifies the database.
* Integrated Security=True uses Windows authentication.

**Step 5: Create the Model**

Create a simple Employee model that corresponds to the Employees table:

public class Employee

{

public int Id { get; set; }

public string Name { get; set; }

public string Position { get; set; }

public decimal Salary { get; set; }

}

This model class represents the data structure of the Employees table.

**Step 6: Create the Repository for CRUD operations**

Dapper uses raw SQL queries for data access. To keep things clean, create a repository to handle all CRUD operations.

Create a Repositories folder in the project and add an EmployeeRepository class:

using Dapper;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

public class EmployeeRepository

{

private string connectionString;

public EmployeeRepository()

{

connectionString = ConfigurationManager.ConnectionStrings["DbConnection"].ConnectionString;

}

// Get a connection to the database

private IDbConnection Connection

{

get { return new SqlConnection(connectionString); }

}

// Get All Employees

public IEnumerable<Employee> GetAll()

{

using (IDbConnection dbConnection = Connection)

{

string query = "SELECT \* FROM Employees";

dbConnection.Open();

return dbConnection.Query<Employee>(query);

}

}

// Get Employee by Id

public Employee GetById(int id)

{

using (IDbConnection dbConnection = Connection)

{

string query = "SELECT \* FROM Employees WHERE Id = @Id";

dbConnection.Open();

return dbConnection.QueryFirstOrDefault<Employee>(query, new { Id = id });

}

}

// Add new Employee

public void Add(Employee employee)

{

using (IDbConnection dbConnection = Connection)

{

string query = "INSERT INTO Employees (Name, Position, Salary) VALUES (@Name, @Position, @Salary)";

dbConnection.Open();

dbConnection.Execute(query, employee);

}

}

// Update Employee

public void Update(Employee employee)

{

using (IDbConnection dbConnection = Connection)

{

string query = "UPDATE Employees SET Name = @Name, Position = @Position, Salary = @Salary WHERE Id = @Id";

dbConnection.Open();

dbConnection.Execute(query, employee);

}

}

// Delete Employee

public void Delete(int id)

{

using (IDbConnection dbConnection = Connection)

{

string query = "DELETE FROM Employees WHERE Id = @Id";

dbConnection.Open();

dbConnection.Execute(query, new { Id = id });

}

}

}

**Explanation:**

* Connection: Establishes a connection to the database using the connection string.
* GetAll(): Fetches all employees from the Employees table.
* GetById(int id): Retrieves a specific employee by their Id.
* Add(Employee employee): Inserts a new employee into the database.
* Update(Employee employee): Updates an existing employee’s details.
* Delete(int id): Deletes an employee from the database.

**Step 7: Create the Controller**

Now, create a controller that will handle the requests and responses related to Employee data.

1. Right-click on the Controllers folder, and select **Add > Controller**.
2. Choose **MVC 5 Controller – Empty** and name it EmployeeController.

Add the following code to the EmployeeController:

using System.Web.Mvc;

public class EmployeeController : Controller

{

private EmployeeRepository repository = new EmployeeRepository();

// GET: Employee

public ActionResult Index()

{

var employees = repository.GetAll();

return View(employees);

}

// GET: Employee/Details/5

public ActionResult Details(int id)

{

var employee = repository.GetById(id);

return View(employee);

}

// GET: Employee/Create

public ActionResult Create()

{

return View();

}

// POST: Employee/Create

[HttpPost]

public ActionResult Create(Employee employee)

{

try

{

repository.Add(employee);

return RedirectToAction("Index");

}

catch

{

return View();

}

}

// GET: Employee/Edit/5

public ActionResult Edit(int id)

{

var employee = repository.GetById(id);

return View(employee);

}

// POST: Employee/Edit/5

[HttpPost]

public ActionResult Edit(Employee employee)

{

try

{

repository.Update(employee);

return RedirectToAction("Index");

}

catch

{

return View();

}

}

// GET: Employee/Delete/5

public ActionResult Delete(int id)

{

var employee = repository.GetById(id);

return View(employee);

}

// POST: Employee/Delete/5

[HttpPost, ActionName("Delete")]

public ActionResult DeleteConfirmed(int id)

{

try

{

repository.Delete(id);

return RedirectToAction("Index");

}

catch

{

return View();

}

}

}

**Step 8: Create Views**

1. Right-click on the Views folder, add a folder named Employee.
2. Add the following Razor Views (Index.cshtml, Create.cshtml, Edit.cshtml, Details.cshtml, and Delete.cshtml) inside this folder.

Here’s a basic template for Index.cshtml:

@model IEnumerable<Employee>

<h2>Employees</h2>

<p>

@Html.ActionLink("Create New", "Create")

</p>

<table class="table">

<tr>

<th>Name</th>

<th>Position</th>

<th>Salary</th>

<th></th>

</tr>

@foreach (var item in Model)

{

<tr>

<td>@item.Name</td>

<td>@item.Position</td>

<td>@item.Salary</td>

<td>

@Html.ActionLink("Edit", "Edit", new { id = item.Id }) |

@Html.ActionLink("Details", "Details", new { id = item.Id }) |

@Html.ActionLink("Delete", "Delete", new { id = item.Id })

</td>

</tr>

}

</table>

For other views, you can follow a similar pattern for creating forms to input employee data.

**Step 9: Run the Application**

Press **F5** in Visual Studio to run the application. You should be able to:

* View all employees.
* Add new employees.
* Edit employee details.
* Delete employees.

**Summary**

* **Dapper** is used for data access with raw SQL queries.
* The **EmployeeRepository** class handles CRUD operations with SQL Server.
* The **EmployeeController** manages the flow between views and database operations.
* Views display and interact with employee data using Razor syntax.

This project structure follows the MVC pattern with Dapper for lightweight data access.

Bottom of Form