🧩 Module 1: Introduction to ASP.NET

1. What is ASP.NET?

Think of it as a cooking system by Microsoft that helps you build dynamic websites—like a restaurant menu that changes based on what customers want.

2. What is PostBack?

Like filling a form and hitting submit—it sends the same form back to the server to process your input.

3. What are Server-side controls?

These are tools like TextBox and Button that work on the server. Example: You type in a name, click a button, and the server remembers and processes it.

4. What is AutoPostBack?

Like a smart form field—if you select something from a dropdown, it instantly reacts and sends data to the server without clicking submit.

🎨 Module 2: UI Design with Controls and Master Page

1. What is a Master Page?

It’s like a template. For example, a school website with the same header/footer on every page—only the center content changes.

2. What are validation controls?

They check user input. Example: If you forget to enter your email, RequiredFieldValidator shows an error.

3. What is AdRotator?

It shows different images/ads each time you reload the page—like changing banners on a shopping website.

🗄️ Module 3: Database with ADO.NET

1. What is ADO.NET?

It’s a toolkit that helps your website talk to databases. Example: Showing user data from a table.

2. Connected vs Disconnected architecture?

Connected: Like a live call (SqlDataReader).

Disconnected: Like a recorded message (DataSet).

3. Two data-bound controls?

GridView, Repeater – they show data like Excel tables on web pages.

4. How to call a stored procedure?

Use SqlCommand, set type to StoredProcedure, and run it.

🧠 Module 4: State Management & AJAX

1. What is State Management?

It’s about remembering user data between pages. Like remembering your cart items while shopping.

2. Types of State Management?

Client: Stored in browser (ViewState, Cookies)

Server: Stored in server memory (Session, Application)

3. What is AJAX?

It updates parts of a page without reloading. Like typing in Google search and getting suggestions instantly.

4. What is ViewState?

It stores values like a form's content so it doesn’t reset on refresh.

🌐 Module 5: Web Services and WCF

1. What is a Web Service?

It shares data with other apps. Like a weather app using data from a weather service.

2. Web Service vs WCF?

Web Service is basic. WCF is advanced—it can use more protocols (not just HTTP).

3. What is SOAP?

It’s like a letter format (in XML) for sending data between computers.

🏗️ Module 6: ASP.NET Core MVC

1. What is MVC?

Model–View–Controller = Brains + Design + Controller. It keeps the code neat and separate.

2. What are Razor Views?

Web pages where HTML and C# mix together. Example: You display a name using @Model.Name.

3. What is Entity Framework?

It connects your code to the database easily. You work with classes instead of SQL directly.

4. CRUD in MVC?

Create (Add), Read (View), Update (Edit), Delete (Remove) data using MVC + Entity Framework.

✅ Experiments (With Real Examples)

1. Registration Form

User signup with TextBoxes, DropDownList, etc. Use validators for checking input.

2. Master Page Website

One layout for all pages. Header/footer same, only body content changes.

3. Connection-Oriented Architecture

Use SqlConnection, SqlCommand to read live data from database.

4. Disconnected Architecture

Use DataSet to pull data, then close DB. Work with it offline.

5. Data-Bound Controls

Show data in GridView, Repeater—like showing employee details in a table.

6. Simple Stored Procedure

Create a stored procedure in SQL and call it to show results in GridView.

7. Parameterized Stored Procedure

Pass user input (like ID) and fetch matching data using stored procedure.

8. Using LINQ

Query data using C# like SQL. Example: from e in employees where e.Salary > 5000 select e;

9. Entity Framework

Use EF to create classes that link to DB tables and work with them easily.

10. Client-Side Session

Use Cookies/ViewState to keep data in the browser (like theme settings).

11. Server-Side Session

Use Session to store username so it stays same across pages.

12. AJAX Controls

Use UpdatePanel to refresh only part of a page—like live search results.

13. Web Service

Create a WebMethod that returns employee data, use it in another web app.

14. MVC Framework App

Use MVC pattern to build app—create models, write controllers, design views.

15. MVC + Entity Framework

Connect MVC app to DB using EF, display data on Razor pages.

16. CRUD in MVC

Make a complete form to add, show, edit, and delete users using Razor + EF.

Practical 1

**step-by-step guide** to perform the **ASP.NET Web Application practical** for a **Registration Form with Advanced Controls** using **ASP.NET Web Forms** in **Visual Studio**.

**🔹 Step 1: Create ASP.NET Web Forms Project**

**✅ What to do:**

1. Open **Visual Studio**.
2. Go to **File > New > Project**.
3. Select:
   * Templates > Visual C# > Web > ASP.NET Web Application (.NET Framework)
4. Name the project: OrganizationRegistration
5. Click **OK**.
6. In the next dialog, choose **Web Forms**, then click **Create**.

**🔹 Step 2: Add a New Web Form Page**

**✅ What to do:**

1. Right-click the project in **Solution Explorer**.
2. Choose Add > Web Form.
3. Name it: Registration.aspx.

**🔹 Step 3: Design the Registration Form (Frontend)**

**✅ What to do:**

1. Open Registration.aspx.
2. Replace the code with the following:

aspx

CopyEdit

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Registration.aspx.cs" Inherits="OrganizationRegistration.Registration" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Registration Form</title>

</head>

<body>

<form id="form1" runat="server">

<div style="width: 400px; margin: auto; padding: 20px; border: 1px solid #ccc;">

<h2>Registration Form</h2>

<asp:Label runat="server" Text="Full Name:" /><br />

<asp:TextBox ID="txtName" runat="server" /><br /><br />

<asp:Label runat="server" Text="Email:" /><br />

<asp:TextBox ID="txtEmail" runat="server" TextMode="Email" /><br /><br />

<asp:Label runat="server" Text="Phone:" /><br />

<asp:TextBox ID="txtPhone" runat="server" /><br /><br />

<asp:Label runat="server" Text="Gender:" /><br />

<asp:RadioButtonList ID="rblGender" runat="server">

<asp:ListItem Text="Male" Value="Male" />

<asp:ListItem Text="Female" Value="Female" />

</asp:RadioButtonList><br />

<asp:Label runat="server" Text="Role:" /><br />

<asp:DropDownList ID="ddlRole" runat="server">

<asp:ListItem Text="Select Role" Value="" />

<asp:ListItem Text="Admin" />

<asp:ListItem Text="Employee" />

<asp:ListItem Text="Manager" />

</asp:DropDownList><br /><br />

<asp:Label runat="server" Text="Skills:" /><br />

<asp:CheckBoxList ID="cblSkills" runat="server">

<asp:ListItem Text="C#" />

<asp:ListItem Text="JavaScript" />

<asp:ListItem Text="SQL" />

<asp:ListItem Text="ASP.NET" />

</asp:CheckBoxList><br />

<asp:Label runat="server" Text="Upload Resume:" /><br />

<asp:FileUpload ID="fuResume" runat="server" /><br /><br />

<asp:Label runat="server" Text="Password:" /><br />

<asp:TextBox ID="txtPassword" runat="server" TextMode="Password" /><br /><br />

<asp:Label runat="server" Text="Confirm Password:" /><br />

<asp:TextBox ID="txtConfirmPassword" runat="server" TextMode="Password" /><br /><br />

<asp:Label runat="server" Text="Address:" /><br />

<asp:TextBox ID="txtAddress" runat="server" TextMode="MultiLine" Rows="3" Columns="40" /><br /><br />

<asp:Button ID="btnSubmit" runat="server" Text="Submit" OnClick="btnSubmit\_Click" />

<asp:Button ID="btnReset" runat="server" Text="Reset" OnClick="btnReset\_Click" /><br /><br />

<asp:Label ID="lblResult" runat="server" ForeColor="Green" />

</div>

</form>

</body>

</html>

**🔹 Step 4: Add Backend Code (CodeBehind)**

**✅ What to do:**

1. Open Registration.aspx.cs
2. Replace all code with:

csharp

CopyEdit

using System;

using System.Linq;

using System.Web.UI.WebControls;

namespace OrganizationRegistration

{

public partial class Registration : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

}

protected void btnSubmit\_Click(object sender, EventArgs e)

{

if (txtPassword.Text != txtConfirmPassword.Text)

{

lblResult.ForeColor = System.Drawing.Color.Red;

lblResult.Text = "Passwords do not match.";

return;

}

string selectedSkills = string.Join(", ", cblSkills.Items.Cast<ListItem>()

.Where(i => i.Selected)

.Select(i => i.Text));

string resumeName = fuResume.HasFile ? fuResume.FileName : "No file uploaded";

lblResult.ForeColor = System.Drawing.Color.Green;

lblResult.Text = $"<b>Registration Successful!</b><br/>" +

$"Name: {txtName.Text}<br/>" +

$"Email: {txtEmail.Text}<br/>" +

$"Phone: {txtPhone.Text}<br/>" +

$"Gender: {rblGender.SelectedValue}<br/>" +

$"Role: {ddlRole.SelectedValue}<br/>" +

$"Skills: {selectedSkills}<br/>" +

$"Address: {txtAddress.Text}<br/>" +

$"Resume: {resumeName}";

}

protected void btnReset\_Click(object sender, EventArgs e)

{

txtName.Text = "";

txtEmail.Text = "";

txtPhone.Text = "";

txtPassword.Text = "";

txtConfirmPassword.Text = "";

txtAddress.Text = "";

ddlRole.SelectedIndex = 0;

rblGender.ClearSelection();

cblSkills.ClearSelection();

lblResult.Text = "";

}

}

}

**🔹 Step 5: (Optional) Create Uploads/ Folder**

If you want to save uploaded files:

1. Right-click your project > Add > New Folder
2. Name it Uploads
3. Modify this line in btnSubmit\_Click:

csharp

CopyEdit

if (fuResume.HasFile)

{

string path = Server.MapPath("~/Uploads/" + fuResume.FileName);

fuResume.SaveAs(path);

}

**🔹 Step 6: Run the Application**

**✅ What to do:**

1. Press Ctrl + F5 (Run without debugging)
2. Fill out the form
3. Test:
   * Validations (password match)
   * Drop-down, checkboxes, radio buttons
   * Resume upload simulation
   * Reset button
4. See the result below the form

**🎓 Summary**

| **Feature** | **Control Used** |
| --- | --- |
| Full Name | TextBox |
| Email | TextBox (TextMode=Email) |
| Phone | TextBox |
| Gender | RadioButtonList |
| Role | DropDownList |
| Skills | CheckBoxList |
| Resume | FileUpload |
| Address | TextBox (MultiLine) |
| Submit/Reset | Button |
| Result | Label |

**📥 Deliverables for Submission**

* Screenshot of the working form
* Screenshot of the form after submission
* Code from Registration.aspx and Registration.aspx.cs
* Optional: ZIP of your project folder

Create a website using the master page concept.

**ChatGPT said:**

Creating a website using the **Master Page** concept in **ASP.NET Web Forms** is a practical and efficient way to maintain a consistent layout across multiple pages.

**✅ PRACTICAL TITLE**

**"Create a Website using the Master Page Concept in ASP.NET Web Forms."**

**🧭 Step-by-Step Guide with Code**

**🔹 Step 1: Create ASP.NET Web Forms Project**

**✅ What to do:**

1. Open **Visual Studio**.
2. Go to File > New > Project.
3. Select:
   * **Visual C# > Web > ASP.NET Web Application (.NET Framework)**
4. Name your project (e.g., MasterPageWebsite), then click **OK**.
5. Choose **Web Forms**, click **Create**.

**🔹 Step 2: Add a Master Page**

**✅ What to do:**

1. Right-click your project > Add > New Item.
2. Choose **Master Page**, name it Site.master, and click **Add**.

**✅ Replace Site.master code with:**

aspx

CopyEdit

<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site.master.cs" Inherits="MasterPageWebsite.Site" %>

<!DOCTYPE html>

<html>

<head runat="server">

<title><asp:ContentPlaceHolder ID="head" runat="server" /></title>

<style>

body { font-family: Arial; }

.header { background: #007acc; color: white; padding: 10px; }

.menu { background: #ddd; padding: 10px; }

.content { padding: 20px; }

.footer { background: #007acc; color: white; padding: 10px; text-align: center; }

</style>

</head>

<body>

<form id="form1" runat="server">

<div class="header">

<h1>Organization Website</h1>

</div>

<div class="menu">

<a href="Home.aspx">Home</a> |

<a href="About.aspx">About</a> |

<a href="Contact.aspx">Contact</a>

</div>

<div class="content">

<asp:ContentPlaceHolder ID="MainContent" runat="server" />

</div>

<div class="footer">

&copy; 2025 My Organization

</div>

</form>

</body>

</html>

**🔹 Step 3: Add Content Pages (Linked to Master Page)**

**✅ What to do:**

Repeat this for each page you want (e.g., Home, About, Contact):

1. Right-click project > Add > Web Form using Master Page.
2. Name the page: Home.aspx, About.aspx, etc.
3. In the next screen, choose Site.master as the master page.

**🔹 Step 4: Edit Content Pages**

**✅ Example: Home.aspx**

aspx

CopyEdit

<%@ Page Title="Home" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true" CodeBehind="Home.aspx.cs" Inherits="MasterPageWebsite.Home" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">

<title>Home - Organization</title>

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">

<h2>Welcome to our Organization</h2>

<p>This is the homepage of our sample ASP.NET website using master pages.</p>

</asp:Content>

**✅ Example: About.aspx**

aspx

CopyEdit

<%@ Page Title="About" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true" CodeBehind="About.aspx.cs" Inherits="MasterPageWebsite.About" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">

<title>About - Organization</title>

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">

<h2>About Us</h2>

<p>We are a fictional organization built for ASP.NET practicals.</p>

</asp:Content>

**🔹 Step 5: Run the Website**

**✅ What to do:**

1. Set Home.aspx as the **Start Page** (right-click > Set as Start Page).
2. Press Ctrl + F5 to run.
3. Click navigation links to test the shared layout and individual content pages.

**🎯 Master Page Concept Benefits**

* Centralized layout control (e.g., header, menu, footer).
* Reduces redundancy across multiple pages.
* Easy updates to site-wide UI.

**📥 Deliverables for Submission**

* Screenshots of all content pages in browser
* Code of:
  + Site.master
  + Home.aspx, About.aspx, etc.
* Project ZIP file (optional)

**PRACTICAL TITLE:**

**"Design a Web Page to Demonstrate Connection-Oriented Architecture using ASP.NET and SQL Server."**

**📌 TOOLS REQUIRED:**

* **Visual Studio** (any version)
* **SQL Server** or **SQL Server Express / LocalDB**
* **ASP.NET Web Forms**

**🧭 STEP-BY-STEP IMPLEMENTATION**

**🔹 STEP 1: Create the Database in SQL Server**

**✅ What to Do:**

1. Open **SQL Server Management Studio** or **SQL Server Express**.
2. Run the following script to create a database and table:

sql

CopyEdit

-- Create database

CREATE DATABASE OrganizationDB;

-- Use the database

USE OrganizationDB;

-- Create Employees table

CREATE TABLE Employees (

Id INT PRIMARY KEY IDENTITY(1,1),

Name NVARCHAR(100),

Department NVARCHAR(100),

Salary INT

);

-- Insert some records

INSERT INTO Employees (Name, Department, Salary) VALUES

('Alice', 'HR', 35000),

('Bob', 'IT', 50000),

('Charlie', 'Finance', 45000);

**🔹 STEP 2: Create an ASP.NET Web Forms Project**

**✅ What to Do:**

1. Open **Visual Studio**.
2. Go to File > New > Project.
3. Select:
   * **ASP.NET Web Application (.NET Framework)**
   * Name it: ConnectionOrientedDemo
4. Click **OK**.
5. Select **Web Forms** in the next dialog and click **Create**.

**🔹 STEP 3: Add a New Web Form**

**✅ What to Do:**

1. Right-click the project in **Solution Explorer**.
2. Click Add > Web Form.
3. Name it ConnectionDemo.aspx.

**🔹 STEP 4: Add Frontend Code (Design the Page)**

**✅ Open ConnectionDemo.aspx and replace the content with:**

aspx

CopyEdit

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="ConnectionDemo.aspx.cs" Inherits="ConnectionOrientedDemo.ConnectionDemo" %>

<!DOCTYPE html>

<html>

<head runat="server">

<title>Connection-Oriented Example</title>

</head>

<body>

<form id="form1" runat="server">

<div style="width: 600px; margin: auto;">

<h2>Employee Details (Connection-Oriented Architecture)</h2>

<asp:Button ID="btnLoadData" runat="server" Text="Load Employees" OnClick="btnLoadData\_Click" />

<br /><br />

<asp:GridView ID="gvEmployees" runat="server" AutoGenerateColumns="true" />

</div>

</form>

</body>

</html>

**🔹 STEP 5: Add Backend Code (ADO.NET Logic)**

**✅ Open ConnectionDemo.aspx.cs and add the following code:**

csharp

CopyEdit

using System;

using System.Data.SqlClient;

using System.Data;

namespace ConnectionOrientedDemo

{

public partial class ConnectionDemo : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e) { }

protected void btnLoadData\_Click(object sender, EventArgs e)

{

// Step 1: Define the connection string

string connectionString = "Data Source=.;Initial Catalog=OrganizationDB;Integrated Security=True";

// Step 2: Write SQL Query

string query = "SELECT \* FROM Employees";

// Step 3: Create connection and command

using (SqlConnection conn = new SqlConnection(connectionString))

{

SqlCommand cmd = new SqlCommand(query, conn);

// Step 4: Open connection

conn.Open();

// Step 5: Execute reader (Connection-Oriented)

SqlDataReader reader = cmd.ExecuteReader();

// Step 6: Load data into DataTable

DataTable dt = new DataTable();

dt.Load(reader); // Connection must remain open during this read

// Step 7: Bind data to GridView

gvEmployees.DataSource = dt;

gvEmployees.DataBind();

// Step 8: Close connection

conn.Close();

}

}

}

}

**🔹 STEP 6: Run and Test the Web Page**

**✅ What to Do:**

1. Right-click ConnectionDemo.aspx → Set as Start Page.
2. Press Ctrl + F5 to run.
3. Click **"Load Employees"** button.
4. You will see data loaded from the SQL Server database using a **connection-oriented** method.

**🧠 Explanation of Architecture:**

| **Step** | **Description** |
| --- | --- |
| SqlConnection | Establishes connection to the database (Connection-Oriented) |
| SqlCommand | Holds the SQL query |
| SqlDataReader | Reads data while connection is open |
| DataTable.Load() | Loads data into memory during active connection |
| GridView | Binds and displays data on the web page |

**✅ Deliverables for Your Practical:**

* Screenshot of SQL table with data
* Screenshot of the web page before and after clicking **Load Employees**
* Code files:
  + ConnectionDemo.aspx
  + ConnectionDemo.aspx.cs
* SQL Script for OrganizationDB and Employees table
* ZIP of Visual Studio project (optional)

**"Design a Web Page to Demonstrate Disconnected Architecture using ASP.NET and ADO.NET."**

**📌 TOOLS REQUIRED:**

* Visual Studio
* SQL Server (Express or full)
* ASP.NET Web Forms

**🧭 STEP-BY-STEP IMPLEMENTATION**

**🔹 STEP 1: Create Database and Table in SQL Server**

**What to do:**

Open SQL Server and run:

sql

CopyEdit

-- Create database

CREATE DATABASE DisconnectedDemoDB;

-- Use it

USE DisconnectedDemoDB;

-- Create table

CREATE TABLE Products (

Id INT PRIMARY KEY IDENTITY(1,1),

ProductName NVARCHAR(100),

Price DECIMAL(10, 2)

);

-- Insert sample data

INSERT INTO Products (ProductName, Price) VALUES

('Keyboard', 1200.00),

('Mouse', 500.00),

('Monitor', 7000.00);

**🔹 STEP 2: Create an ASP.NET Web Forms Project**

1. Open Visual Studio.
2. File > New > Project.
3. Select **ASP.NET Web Application (.NET Framework)**.
4. Name it DisconnectedArchitectureDemo.
5. Select **Web Forms**, click **Create**.

**🔹 STEP 3: Add Web Form**

1. Right-click on the project → Add > Web Form.
2. Name it: DisconnectedDemo.aspx.

**🔹 STEP 4: Front-End Design (DisconnectedDemo.aspx)**

Replace the code in DisconnectedDemo.aspx:

aspx

CopyEdit

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="DisconnectedDemo.aspx.cs" Inherits="DisconnectedArchitectureDemo.DisconnectedDemo" %>

<!DOCTYPE html>

<html>

<head runat="server">

<title>Disconnected Architecture Demo</title>

</head>

<body>

<form id="form1" runat="server">

<div style="padding: 20px;">

<h2>Product List (Disconnected Architecture)</h2>

<asp:Button ID="btnLoad" runat="server" Text="Load Products" OnClick="btnLoad\_Click" />

<br /><br />

<asp:GridView ID="gvProducts" runat="server" AutoGenerateColumns="true" />

</div>

</form>

</body>

</html>

**🔹 STEP 5: Backend Code (DisconnectedDemo.aspx.cs)**

Replace code in DisconnectedDemo.aspx.cs:

csharp

CopyEdit

using System;

using System.Data;

using System.Data.SqlClient;

namespace DisconnectedArchitectureDemo

{

public partial class DisconnectedDemo : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e) { }

protected void btnLoad\_Click(object sender, EventArgs e)

{

// 1. Connection string to SQL Server

string connectionString = "Data Source=.;Initial Catalog=DisconnectedDemoDB;Integrated Security=True";

// 2. SQL Query

string query = "SELECT \* FROM Products";

// 3. Set up ADO.NET disconnected objects

SqlDataAdapter adapter = new SqlDataAdapter(query, connectionString);

DataSet ds = new DataSet();

// 4. Fill DataSet (fetches and disconnects)

adapter.Fill(ds, "Products");

// 5. Bind data to GridView

gvProducts.DataSource = ds.Tables["Products"];

gvProducts.DataBind();

}

}

}

⚠️ Replace Data Source=. with your actual server name if needed.

**🔹 STEP 6: Run and Test the Page**

1. Right-click DisconnectedDemo.aspx → Set as Start Page.
2. Press Ctrl + F5 to run the website.
3. Click **"Load Products"** to see the data loaded from the database using a disconnected architecture.

**🧠 Key Concept: Disconnected Architecture**

| **Feature** | **Description** |
| --- | --- |
| SqlDataAdapter | Acts as a bridge between DB and DataSet |
| DataSet | Stores data in-memory, no need to keep connection open |
| Connection Closed | After fetching data, app works offline |
| Ideal For | Reducing DB load, disconnected environments (e.g. offline mode) |

**✅ Practical Deliverables**

* Screenshot of:
  + SQL table with data
  + Web page before/after loading data
* Code files:
  + DisconnectedDemo.aspx
  + DisconnectedDemo.aspx.cs
* SQL script for Products table
* Visual Studio project (optional ZIP)

**Create a Web Page that Demonstrates the Use of Data-Bound Controls in ASP.NET."**

**📌 GOAL:**

Use ASP.NET **data-bound controls** like:

* GridView
* DropDownList
* DetailsView

These controls bind directly to data sources such as SqlDataSource, DataTable, or DataSet.

**🧭 STEP-BY-STEP IMPLEMENTATION**

**🔹 STEP 1: Create Database in SQL Server**

**SQL Script to run in SQL Server:**

sql

CopyEdit

-- Create database

CREATE DATABASE DataBoundDemoDB;

-- Use it

USE DataBoundDemoDB;

-- Create Products table

CREATE TABLE Products (

ProductId INT PRIMARY KEY IDENTITY,

ProductName NVARCHAR(100),

Category NVARCHAR(50),

Price DECIMAL(10, 2)

);

-- Insert sample data

INSERT INTO Products (ProductName, Category, Price) VALUES

('Laptop', 'Electronics', 55000),

('Printer', 'Electronics', 12000),

('Office Chair', 'Furniture', 4500),

('Desk', 'Furniture', 8000);

**🔹 STEP 2: Create ASP.NET Web Forms Project**

1. Open **Visual Studio**.
2. Go to File > New > Project.
3. Select **ASP.NET Web Application (.NET Framework)**.
4. Name it: DataBoundControlsDemo
5. Select **Web Forms**, click **Create**.

**🔹 STEP 3: Add Web Form**

1. Right-click the project → Add > Web Form
2. Name it: DataBoundControls.aspx

**🔹 STEP 4: Design UI Using Data-Bound Controls**

**Replace DataBoundControls.aspx with the following:**

aspx

CopyEdit

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="DataBoundControls.aspx.cs" Inherits="DataBoundControlsDemo.DataBoundControls" %>

<!DOCTYPE html>

<html>

<head runat="server">

<title>ASP.NET Data Bound Controls</title>

</head>

<body>

<form id="form1" runat="server">

<div style="padding: 20px;">

<h2>Product List (Using Data Bound Controls)</h2>

<asp:DropDownList ID="ddlCategory" runat="server" AutoPostBack="true" OnSelectedIndexChanged="ddlCategory\_SelectedIndexChanged">

</asp:DropDownList>

<br /><br />

<asp:GridView ID="gvProducts" runat="server" AutoGenerateColumns="true" />

<br /><br />

<asp:DetailsView ID="dvProduct" runat="server" AutoGenerateRows="true" DefaultMode="ReadOnly" />

</div>

</form>

</body>

</html>

**🔹 STEP 5: Add Code Behind (DataBoundControls.aspx.cs)**

**Replace DataBoundControls.aspx.cs with:**

csharp

CopyEdit

using System;

using System.Data;

using System.Data.SqlClient;

namespace DataBoundControlsDemo

{

public partial class DataBoundControls : System.Web.UI.Page

{

string connectionString = "Data Source=.;Initial Catalog=DataBoundDemoDB;Integrated Security=True";

protected void Page\_Load(object sender, EventArgs e)

{

if (!IsPostBack)

{

BindCategoryDropdown();

}

}

private void BindCategoryDropdown()

{

using (SqlConnection conn = new SqlConnection(connectionString))

{

SqlCommand cmd = new SqlCommand("SELECT DISTINCT Category FROM Products", conn);

conn.Open();

SqlDataReader reader = cmd.ExecuteReader();

ddlCategory.DataSource = reader;

ddlCategory.DataTextField = "Category";

ddlCategory.DataValueField = "Category";

ddlCategory.DataBind();

ddlCategory.Items.Insert(0, "-- Select Category --");

}

}

protected void ddlCategory\_SelectedIndexChanged(object sender, EventArgs e)

{

string selectedCategory = ddlCategory.SelectedValue;

if (selectedCategory != "-- Select Category --")

{

BindProductsGrid(selectedCategory);

}

}

private void BindProductsGrid(string category)

{

using (SqlConnection conn = new SqlConnection(connectionString))

{

SqlCommand cmd = new SqlCommand("SELECT \* FROM Products WHERE Category = @Category", conn);

cmd.Parameters.AddWithValue("@Category", category);

SqlDataAdapter adapter = new SqlDataAdapter(cmd);

DataTable dt = new DataTable();

adapter.Fill(dt);

gvProducts.DataSource = dt;

gvProducts.DataBind();

// Bind to DetailsView if there is at least one record

if (dt.Rows.Count > 0)

{

dvProduct.DataSource = dt;

dvProduct.DataBind();

}

else

{

dvProduct.DataSource = null;

dvProduct.DataBind();

}

}

}

}

}

✅ Replace Data Source=. with your actual SQL Server name if needed.

**🔹 STEP 6: Run and Test**

1. Right-click DataBoundControls.aspx → Set as Start Page.
2. Press Ctrl + F5 to run the app.
3. Select a **Category** from the dropdown.
4. **GridView** and **DetailsView** will show the related product data.

**🧠 Explanation of Data-Bound Controls**

| **Control** | **Description** |
| --- | --- |
| DropDownList | Bound to product categories from the database |
| GridView | Displays multiple products in tabular format |
| DetailsView | Shows details of a selected product (single record) |

**✅ Deliverables for Practical Submission**

* Screenshot of SQL table and sample records.
* Screenshot of UI with data-bound controls in action.
* Code:
  + DataBoundControls.aspx
  + DataBoundControls.aspx.cs
* SQL Script for Products table
* Project ZIP (optional)

**Design a Web Page to Demonstrate the Working of a Simple Stored Procedure in ASP.NET."**

**🧰 TOOLS REQUIRED:**

* **SQL Server**
* **Visual Studio** with ASP.NET Web Forms
* **ADO.NET** (for database access)

**🧭 OVERVIEW OF WHAT WE WILL DO:**

1. Create a database and a stored procedure in SQL Server.
2. Create a Web Form in ASP.NET to call the stored procedure.
3. Display the results using a GridView.

**🔹 STEP 1: Create SQL Server Database and Stored Procedure**

**🧾 SQL Script to Execute:**

sql

CopyEdit

-- Create the database

CREATE DATABASE StoredProcDemoDB;

GO

-- Use the database

USE StoredProcDemoDB;

GO

-- Create table

CREATE TABLE Employees (

Id INT PRIMARY KEY IDENTITY,

Name NVARCHAR(100),

Department NVARCHAR(100),

Salary INT

);

GO

-- Insert sample data

INSERT INTO Employees (Name, Department, Salary) VALUES

('Alice', 'HR', 30000),

('Bob', 'IT', 50000),

('Carol', 'Finance', 45000);

GO

-- Create a stored procedure

CREATE PROCEDURE GetEmployeesByDepartment

@Department NVARCHAR(100)

AS

BEGIN

SELECT \* FROM Employees WHERE Department = @Department;

END;

GO

**🔹 STEP 2: Create an ASP.NET Web Forms Project**

1. Open **Visual Studio**.
2. Create a new project:
   * Choose: ASP.NET Web Application (.NET Framework)
   * Name: StoredProcDemo
   * Select **Web Forms**, click **Create**

**🔹 STEP 3: Add a Web Form**

1. Right-click project → Add > Web Form
2. Name it: StoredProcPage.aspx

**🔹 STEP 4: Front-End Code (StoredProcPage.aspx)**

Replace the contents of StoredProcPage.aspx with:

aspx

CopyEdit

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="StoredProcPage.aspx.cs" Inherits="StoredProcDemo.StoredProcPage" %>

<!DOCTYPE html>

<html>

<head runat="server">

<title>Stored Procedure Demo</title>

</head>

<body>

<form id="form1" runat="server">

<div style="padding:20px;">

<h2>View Employees by Department (Stored Procedure)</h2>

<asp:Label ID="Label1" runat="server" Text="Select Department: " />

<asp:DropDownList ID="ddlDepartment" runat="server">

<asp:ListItem Text="-- Select --" Value="" />

<asp:ListItem Text="HR" Value="HR" />

<asp:ListItem Text="IT" Value="IT" />

<asp:ListItem Text="Finance" Value="Finance" />

</asp:DropDownList>

<asp:Button ID="btnLoad" runat="server" Text="Load Employees" OnClick="btnLoad\_Click" />

<br /><br />

<asp:GridView ID="gvEmployees" runat="server" AutoGenerateColumns="true" />

</div>

</form>

</body>

</html>

**🔹 STEP 5: Backend Code (StoredProcPage.aspx.cs)**

Replace the contents of StoredProcPage.aspx.cs with:

csharp

CopyEdit

using System;

using System.Data;

using System.Data.SqlClient;

namespace StoredProcDemo

{

public partial class StoredProcPage : System.Web.UI.Page

{

string connectionString = "Data Source=.;Initial Catalog=StoredProcDemoDB;Integrated Security=True";

protected void btnLoad\_Click(object sender, EventArgs e)

{

string selectedDept = ddlDepartment.SelectedValue;

if (string.IsNullOrEmpty(selectedDept)) return;

using (SqlConnection conn = new SqlConnection(connectionString))

{

using (SqlCommand cmd = new SqlCommand("GetEmployeesByDepartment", conn))

{

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@Department", selectedDept);

SqlDataAdapter da = new SqlDataAdapter(cmd);

DataTable dt = new DataTable();

da.Fill(dt);

gvEmployees.DataSource = dt;

gvEmployees.DataBind();

}

}

}

}

}

⚠️ Adjust "Data Source=." if you're using a named SQL Server instance or another server.

**🔹 STEP 6: Run and Test**

1. Set StoredProcPage.aspx as the start page.
2. Press Ctrl + F5 to run.
3. Select a department from the dropdown.
4. Click **"Load Employees"**.
5. You will see results fetched via the **stored procedure** and displayed in the GridView.

**✅ PRACTICAL EXPLANATION**

| **Component** | **Purpose** |
| --- | --- |
| Stored Procedure | Server-side logic to fetch filtered data |
| SqlCommand | Used with CommandType.StoredProcedure to call SP |
| SqlDataAdapter | Fills DataTable with result from stored procedure |
| GridView | Binds to DataTable to display the data |

**✅ PRACTICAL DELIVERABLES**

* Screenshot of SQL Server table and stored procedure
* Screenshot of the web page with employee data
* Files:
  + StoredProcPage.aspx
  + StoredProcPage.aspx.cs
* SQL Script used for DB and stored procedure
* ZIP of Visual Studio project (optional)

Would you like a **printable report format (DOC or PDF)** for journal submission?

4o

Top of Form

Bottom of Form

**Design a Web Page to Demonstrate the Working of Parameterized Stored Procedures in ASP.NET."**

**🧰 REQUIREMENTS:**

* **SQL Server** for database & stored procedure
* **Visual Studio** with ASP.NET Web Forms
* **ADO.NET** for database access

**🧭 WHAT THIS PRACTICAL COVERS:**

* Creating a stored procedure with parameters
* Accepting user input (like Employee ID)
* Fetching specific data from the DB using parameters
* Displaying the result using DetailsView

**🔹 STEP 1: Create SQL Server Database and Stored Procedure**

**🧾 SQL Script:**

sql

CopyEdit

-- Create the database

CREATE DATABASE ParamStoredProcDB;

GO

USE ParamStoredProcDB;

GO

-- Create the Employees table

CREATE TABLE Employees (

Id INT PRIMARY KEY IDENTITY,

Name NVARCHAR(100),

Department NVARCHAR(100),

Salary INT

);

GO

-- Insert sample data

INSERT INTO Employees (Name, Department, Salary) VALUES

('John Doe', 'IT', 50000),

('Jane Smith', 'HR', 45000),

('Tom Brown', 'Finance', 47000);

GO

-- Create parameterized stored procedure

CREATE PROCEDURE GetEmployeeById

@EmpId INT

AS

BEGIN

SELECT \* FROM Employees WHERE Id = @EmpId;

END;

GO

**🔹 STEP 2: Create an ASP.NET Web Forms Project**

1. Open **Visual Studio**.
2. Go to: File > New > Project.
3. Select: **ASP.NET Web Application (.NET Framework)**.
4. Name it: ParameterizedStoredProcDemo.
5. Choose **Web Forms** template → click **Create**.

**🔹 STEP 3: Add a Web Form**

1. Right-click the project → Add > Web Form.
2. Name it: EmployeeSearch.aspx.

**🔹 STEP 4: Front-End Code (EmployeeSearch.aspx)**

Replace everything with:

aspx

CopyEdit

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="EmployeeSearch.aspx.cs" Inherits="ParameterizedStoredProcDemo.EmployeeSearch" %>

<!DOCTYPE html>

<html>

<head runat="server">

<title>Employee Search - Parameterized Stored Procedure</title>

</head>

<body>

<form id="form1" runat="server">

<div style="padding:20px;">

<h2>Search Employee by ID (Parameterized Stored Procedure)</h2>

<asp:Label ID="Label1" runat="server" Text="Enter Employee ID: " />

<asp:TextBox ID="txtEmpId" runat="server"></asp:TextBox>

<asp:Button ID="btnSearch" runat="server" Text="Search" OnClick="btnSearch\_Click" />

<br /><br />

<asp:DetailsView ID="dvEmployee" runat="server" AutoGenerateRows="true" />

</div>

</form>

</body>

</html>

**🔹 STEP 5: Backend Code (EmployeeSearch.aspx.cs)**

Replace with:

csharp

CopyEdit

using System;

using System.Data;

using System.Data.SqlClient;

namespace ParameterizedStoredProcDemo

{

public partial class EmployeeSearch : System.Web.UI.Page

{

string connectionString = "Data Source=.;Initial Catalog=ParamStoredProcDB;Integrated Security=True";

protected void btnSearch\_Click(object sender, EventArgs e)

{

int empId;

if (!int.TryParse(txtEmpId.Text, out empId))

{

// Show validation message if input is not valid

dvEmployee.DataSource = null;

dvEmployee.DataBind();

return;

}

using (SqlConnection conn = new SqlConnection(connectionString))

{

SqlCommand cmd = new SqlCommand("GetEmployeeById", conn);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@EmpId", empId);

SqlDataAdapter adapter = new SqlDataAdapter(cmd);

DataTable dt = new DataTable();

adapter.Fill(dt);

dvEmployee.DataSource = dt;

dvEmployee.DataBind();

}

}

}

}

✅ Replace Data Source=. with your actual SQL Server name if different.

**🔹 STEP 6: Run and Test**

1. Right-click EmployeeSearch.aspx → **Set as Start Page**
2. Press Ctrl + F5 to launch
3. Enter an employee ID like 1, 2, or 3
4. Click **Search** to see the record fetched using the **parameterized stored procedure**

**🧠 CONCEPT REVIEW**

| **Concept** | **Description** |
| --- | --- |
| Stored Procedure | Predefined SQL logic on the server |
| Parameterized | Uses input like @EmpId to prevent SQL injection |
| SqlCommand | Executes the SP with CommandType.StoredProcedure |
| AddWithValue() | Sends the user input as a parameter |

**✅ PRACTICAL DELIVERABLES**

* Screenshot of:
  + Stored procedure in SQL Server
  + Web page before/after searching
* Files:
  + EmployeeSearch.aspx
  + EmployeeSearch.aspx.cs
* SQL script for DB/table/procedure
* ZIP of Visual Studio project (optional)

**Design a Web Page to Display the Use of LINQ in ASP.NET Web Forms."**

**🧰 REQUIREMENTS:**

* **Visual Studio**
* **ASP.NET Web Forms Project**
* No database is needed for basic LINQ (we’ll use in-memory data)

**🧭 WHAT THIS PRACTICAL COVERS:**

* Create a list of objects in C#
* Use LINQ to query, sort, and filter the list
* Display the result in a GridView

**🔹 STEP 1: Create ASP.NET Web Forms Project**

1. Open **Visual Studio**
2. Go to: File > New > Project
3. Select: **ASP.NET Web Application (.NET Framework)**
4. Name it: LINQDemoWeb
5. Choose **Web Forms**, click **Create**

**🔹 STEP 2: Add a Web Form**

1. Right-click project → Add > Web Form
2. Name it: LinqDemo.aspx

**🔹 STEP 3: Design the Web Page (LinqDemo.aspx)**

Replace the default code with:

aspx

CopyEdit

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="LinqDemo.aspx.cs" Inherits="LINQDemoWeb.LinqDemo" %>

<!DOCTYPE html>

<html>

<head runat="server">

<title>LINQ Demo in ASP.NET</title>

</head>

<body>

<form id="form1" runat="server">

<div style="padding:20px;">

<h2>Employees - LINQ Query Result</h2>

<asp:GridView ID="gvEmployees" runat="server" AutoGenerateColumns="true" />

</div>

</form>

</body>

</html>

**🔹 STEP 4: Add LINQ Logic in Code-Behind (LinqDemo.aspx.cs)**

Replace the code in LinqDemo.aspx.cs with:

csharp

CopyEdit

using System;

using System.Collections.Generic;

using System.Linq;

namespace LINQDemoWeb

{

public partial class LinqDemo : System.Web.UI.Page

{

// Define an Employee class

public class Employee

{

public int Id { get; set; }

public string Name { get; set; }

public string Department { get; set; }

public int Salary { get; set; }

}

protected void Page\_Load(object sender, EventArgs e)

{

if (!IsPostBack)

{

DisplayEmployees();

}

}

private void DisplayEmployees()

{

// Create a list of employees

List<Employee> employees = new List<Employee>

{

new Employee { Id = 1, Name = "Alice", Department = "IT", Salary = 50000 },

new Employee { Id = 2, Name = "Bob", Department = "HR", Salary = 40000 },

new Employee { Id = 3, Name = "Charlie", Department = "IT", Salary = 55000 },

new Employee { Id = 4, Name = "Diana", Department = "Finance", Salary = 45000 }

};

// Use LINQ to get employees from IT department sorted by salary

var result = from emp in employees

where emp.Department == "IT"

orderby emp.Salary descending

select emp;

// Bind to GridView

gvEmployees.DataSource = result.ToList();

gvEmployees.DataBind();

}

}

}

**🔹 STEP 5: Run and Test**

1. Right-click LinqDemo.aspx → Set as Start Page
2. Press Ctrl + F5 to run
3. You’ll see a GridView showing employees from the **IT department** sorted by **Salary (descending)**

**🧠 CONCEPT REVIEW**

| **Concept** | **Description** |
| --- | --- |
| LINQ | Language-Integrated Query for querying data using C# |
| from ... where ... select | Query syntax for filtering data |
| List<T> | Collection of employee objects |
| GridView.DataSource | Binds LINQ query result to the UI |

**✅ PRACTICAL DELIVERABLES**

* Screenshot of the webpage displaying LINQ output
* Code:
  + LinqDemo.aspx
  + LinqDemo.aspx.cs
* ZIP of Visual Studio project (optional)

**🔄 BONUS EXTENSION (Optional)**

You can add:

* A DropDownList to filter by department
* Use LINQ method syntax (.Where(), .OrderByDescending())

**"Build a Website to Demonstrate the Working of Entity Framework in ASP.NET."**

**🧰 REQUIREMENTS:**

* **Visual Studio** (any version with Web Forms + EF support)
* **SQL Server** (Express or LocalDB)
* **Entity Framework (EF 6.x)** – used for data access

**🧭 OVERVIEW:**

You will:

1. Create a database (EFDB) with a table (Students)
2. Use **Entity Framework Database First Approach**
3. Generate model using EF Designer
4. Use a GridView to display student records

**🔹 STEP 1: Create SQL Server Database and Table**

Open **SQL Server Management Studio** and run:

sql

CopyEdit

-- Create database

CREATE DATABASE EFDB;

GO

USE EFDB;

GO

-- Create table

CREATE TABLE Students (

StudentId INT PRIMARY KEY IDENTITY,

FullName NVARCHAR(100),

Age INT,

Email NVARCHAR(100)

);

-- Insert sample data

INSERT INTO Students (FullName, Age, Email) VALUES

('Alice Johnson', 20, 'alice@example.com'),

('Bob Smith', 22, 'bob@example.com'),

('Clara White', 21, 'clara@example.com');

**🔹 STEP 2: Create a New ASP.NET Web Forms Project**

1. Open **Visual Studio**
2. File > New > Project
3. Choose: **ASP.NET Web Application (.NET Framework)**
4. Name: EFWebDemo
5. Template: **Web Forms**
6. Click **Create**

**🔹 STEP 3: Install Entity Framework**

If not already included:

1. Open **Tools > NuGet Package Manager > Manage NuGet Packages for Solution**
2. Install **EntityFramework** (v6.x)

**🔹 STEP 4: Add Entity Data Model**

1. Right-click project → Add > New Item
2. Select **ADO.NET Entity Data Model**
3. Name it: EFModel.edmx
4. Choose: **EF Designer from Database**
5. Click **Next**

**Select Database Connection:**

* Click **New Connection**
* Enter your server name and database: EFDB
* Test connection > OK

Check **"Yes, include sensitive data in the connection string"**

**Choose Objects:**

* Expand Tables → Check **Students**
* Finish the wizard

✅ EF will generate Student model and a EFDBEntities context class.

**🔹 STEP 5: Add a Web Form**

1. Right-click project → Add > Web Form
2. Name it: StudentList.aspx

**🔹 STEP 6: Frontend Code (StudentList.aspx)**

Replace content with:

aspx

CopyEdit

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="StudentList.aspx.cs" Inherits="EFWebDemo.StudentList" %>

<!DOCTYPE html>

<html>

<head runat="server">

<title>Entity Framework Demo - Student List</title>

</head>

<body>

<form id="form1" runat="server">

<div style="padding:20px;">

<h2>Students (Loaded via Entity Framework)</h2>

<asp:GridView ID="gvStudents" runat="server" AutoGenerateColumns="true" />

</div>

</form>

</body>

</html>

**🔹 STEP 7: Backend Code (StudentList.aspx.cs)**

Replace the code-behind with:

csharp

CopyEdit

using System;

using System.Linq;

namespace EFWebDemo

{

public partial class StudentList : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

if (!IsPostBack)

{

LoadStudents();

}

}

private void LoadStudents()

{

using (var context = new EFDBEntities())

{

var students = context.Students.ToList();

gvStudents.DataSource = students;

gvStudents.DataBind();

}

}

}

}

**🔹 STEP 8: Run and Test**

1. Right-click StudentList.aspx → Set as Start Page
2. Press Ctrl + F5
3. You’ll see a list of students loaded from SQL Server using **Entity Framework**

**🧠 CONCEPT REVIEW**

| **Concept** | **Description** |
| --- | --- |
| **Entity Framework** | ORM tool that simplifies database operations |
| **Database First** | Generates model from existing SQL DB |
| **EDMX** | Visual model of tables and relationships |
| **LINQ** | Used to query the database in C# |
| **DbContext** (EFDBEntities) | Provides access to tables |

**✅ DELIVERABLES**

* Screenshot of:
  + EFModel.edmx model diagram
  + Web page with student list
* Files:
  + StudentList.aspx
  + StudentList.aspx.cs
  + EFModel.edmx
* SQL script for database & table
* ZIP of Visual Studio project (optional)

**🚀 BONUS (Optional)**

Want to expand the project?

* Add Create, Update, Delete buttons using FormView or DetailsView
* Use EF Code First or EF Core
* Add filtering using DropDownLists with LINQ

Let me know if you'd like:

* **A DOC/PDF version** of this practical for submission
* Help to **add insert/update/delete** via Entity Framework

4o

Top of Form

Tools

Bottom of Form

**"Design Web Applications using Client-Side Session Management in ASP.NET."**

**🧰 WHAT YOU NEED:**

* Visual Studio (ASP.NET Web Forms)
* Basic understanding of cookies and query strings

**🧭 WHAT THIS PRACTICAL DEMONSTRATES:**

Client-side session management using:

1. **Cookies**
2. **Query Strings**
3. (Optional) Hidden Fields and ViewState

**🔹 APPROACH 1: Using Cookies**

**📄 Page 1: Login.aspx (User enters their name)**

**Frontend: Login.aspx**

aspx

CopyEdit

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Login.aspx.cs" Inherits="ClientSessionDemo.Login" %>

<!DOCTYPE html>

<html>

<head runat="server">

<title>Login with Cookies</title>

</head>

<body>

<form id="form1" runat="server">

<div style="padding:20px;">

<h2>Login Page (Using Cookies)</h2>

<asp:Label ID="Label1" runat="server" Text="Enter Your Name: " />

<asp:TextBox ID="txtName" runat="server" />

<asp:Button ID="btnLogin" runat="server" Text="Login" OnClick="btnLogin\_Click" />

</div>

</form>

</body>

</html>

**Backend: Login.aspx.cs**

csharp

CopyEdit

using System;

namespace ClientSessionDemo

{

public partial class Login : System.Web.UI.Page

{

protected void btnLogin\_Click(object sender, EventArgs e)

{

HttpCookie userCookie = new HttpCookie("UserName");

userCookie.Value = txtName.Text;

userCookie.Expires = DateTime.Now.AddMinutes(10);

Response.Cookies.Add(userCookie);

Response.Redirect("Welcome.aspx");

}

}

}

**📄 Page 2: Welcome.aspx (Reads name from cookie)**

**Frontend: Welcome.aspx**

aspx

CopyEdit

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Welcome.aspx.cs" Inherits="ClientSessionDemo.Welcome" %>

<!DOCTYPE html>

<html>

<head runat="server">

<title>Welcome Page</title>

</head>

<body>

<form id="form1" runat="server">

<div style="padding:20px;">

<h2>Welcome Page</h2>

<asp:Label ID="lblMessage" runat="server" />

</div>

</form>

</body>

</html>

**Backend: Welcome.aspx.cs**

csharp

CopyEdit

using System;

namespace ClientSessionDemo

{

public partial class Welcome : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

if (Request.Cookies["UserName"] != null)

{

string name = Request.Cookies["UserName"].Value;

lblMessage.Text = "Welcome, " + name + "!";

}

else

{

lblMessage.Text = "No cookie found. Please login first.";

}

}

}

}

**🔹 APPROACH 2: Using Query Strings (Alternative Method)**

Modify Login.aspx.cs:

csharp

CopyEdit

protected void btnLogin\_Click(object sender, EventArgs e)

{

string userName = txtName.Text;

Response.Redirect("Welcome.aspx?name=" + Server.UrlEncode(userName));

}

Modify Welcome.aspx.cs:

csharp

CopyEdit

protected void Page\_Load(object sender, EventArgs e)

{

string name = Request.QueryString["name"];

lblMessage.Text = "Welcome, " + name + "!";

}

**🧠 CONCEPT REVIEW**

| **Method** | **Stored In** | **Pros** | **Cons** |
| --- | --- | --- | --- |
| **Cookies** | Client browser | Persistent (time-based), supports multiple pages | Can be blocked or cleared |
| **Query Strings** | URL | Simple and visible | Limited size, not secure |
| **Hidden Fields / ViewState** | Inside page | Easy for postbacks | Not secure across pages |

**✅ DELIVERABLES**

* Screenshot of Login.aspx and Welcome.aspx
* Code files:
  + Login.aspx, Login.aspx.cs
  + Welcome.aspx, Welcome.aspx.cs
* ZIP of Visual Studio project (optional)

**🔄 BONUS IDEAS**

* Add logout feature (clear cookies)
* Use hidden fields and ViewState for form-based session storage
* Compare with server-side session management

**Design Web Applications Using Server-Side Session Management Techniques in ASP.NET."**

**🧰 TOOLS NEEDED:**

* **Visual Studio (ASP.NET Web Forms Template)**
* **.NET Framework (any version that supports Web Forms)**

**🧭 WHAT THIS PRACTICAL COVERS:**

**You will:**

1. **Create a login form**
2. **Store user data in Session**
3. **Access session data on another page**
4. **Clear session on logout**

**🔹 STEP 1: Create ASP.NET Web Forms Project**

1. **Open Visual Studio**
2. **Go to File > New > Project**
3. **Choose ASP.NET Web Application (.NET Framework)**
4. **Name it: ServerSessionDemo**
5. **Select Web Forms, click Create**

**🔹 STEP 2: Add Two Web Forms**

* **Login.aspx — to input data and set session**
* **Welcome.aspx — to access session**
* **(Optional) Logout.aspx — to destroy session**

**🔹 STEP 3: Design Login Page**

**📄 Login.aspx**

**aspx**

**CopyEdit**

**<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Login.aspx.cs" Inherits="ServerSessionDemo.Login" %>**

**<!DOCTYPE html>**

**<html>**

**<head runat="server">**

**<title>Login Page - Session Example</title>**

**</head>**

**<body>**

**<form id="form1" runat="server">**

**<div style="padding:20px;">**

**<h2>Login Page</h2>**

**<asp:Label ID="Label1" runat="server" Text="Enter Your Name: " />**

**<asp:TextBox ID="txtName" runat="server" />**

**<br /><br />**

**<asp:Button ID="btnLogin" runat="server" Text="Login" OnClick="btnLogin\_Click" />**

**</div>**

**</form>**

**</body>**

**</html>**

**📄 Login.aspx.cs**

**csharp**

**CopyEdit**

**using System;**

**namespace ServerSessionDemo**

**{**

**public partial class Login : System.Web.UI.Page**

**{**

**protected void btnLogin\_Click(object sender, EventArgs e)**

**{**

**string name = txtName.Text;**

**Session["UserName"] = name; // Set server-side session**

**Response.Redirect("Welcome.aspx");**

**}**

**}**

**}**

**🔹 STEP 4: Design Welcome Page**

**📄 Welcome.aspx**

**aspx**

**CopyEdit**

**<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Welcome.aspx.cs" Inherits="ServerSessionDemo.Welcome" %>**

**<!DOCTYPE html>**

**<html>**

**<head runat="server">**

**<title>Welcome Page - Server Session</title>**

**</head>**

**<body>**

**<form id="form1" runat="server">**

**<div style="padding:20px;">**

**<h2>Welcome Page</h2>**

**<asp:Label ID="lblWelcome" runat="server" />**

**<br /><br />**

**<asp:Button ID="btnLogout" runat="server" Text="Logout" OnClick="btnLogout\_Click" />**

**</div>**

**</form>**

**</body>**

**</html>**

**📄 Welcome.aspx.cs**

**csharp**

**CopyEdit**

**using System;**

**namespace ServerSessionDemo**

**{**

**public partial class Welcome : System.Web.UI.Page**

**{**

**protected void Page\_Load(object sender, EventArgs e)**

**{**

**if (Session["UserName"] != null)**

**{**

**lblWelcome.Text = "Welcome, " + Session["UserName"].ToString();**

**}**

**else**

**{**

**Response.Redirect("Login.aspx");**

**}**

**}**

**protected void btnLogout\_Click(object sender, EventArgs e)**

**{**

**Session.Clear(); // Or Session.Abandon();**

**Response.Redirect("Login.aspx");**

**}**

**}**

**}**

**🧠 CONCEPT REVIEW**

| **Feature** | **Description** |
| --- | --- |
| **Session["Key"]** | **Stores user-specific data on the server** |
| **Session.Clear()** | **Removes all session values** |
| **Session.Abandon()** | **Ends the current session** |
| **Server-side session** | **More secure, not exposed in URL or client** |

**✅ DELIVERABLES**

* **Code for:**
  + **Login.aspx, Login.aspx.cs**
  + **Welcome.aspx, Welcome.aspx.cs**
* **Screenshot showing:**
  + **Login form**
  + **Welcome message**
* **ZIP of Visual Studio project (optional)**

**🔄 BONUS IDEAS (Optional Enhancements)**

* **Add session timeout in web.config:**

**xml**

**CopyEdit**

**<sessionState timeout="10"></sessionState>**

* **Track login count using session variables**
* **Use master page to display session info on all pages**

**Build a Web Page Using AJAX Controls in ASP.NET."**

**🧰 TOOLS REQUIRED:**

* **Visual Studio (ASP.NET Web Forms)**
* **.NET Framework**
* **System.Web.Extensions (for AJAX support)**

**🧭 WHAT YOU'LL DO:**

**You will:**

* **Use AJAX to update part of the web page without full reload**
* **Use ScriptManager, UpdatePanel, and a Button**
* **Dynamically show current time on button click without refreshing the page**

**🔹 STEP 1: Create ASP.NET Web Forms Project**

1. **Open Visual Studio**
2. **File > New > Project**
3. **Choose: ASP.NET Web Application (.NET Framework)**
4. **Name it: AJAXDemo**
5. **Select Web Forms, click Create**

**🔹 STEP 2: Add Web Form**

1. **Right-click project → Add > Web Form**
2. **Name it: AjaxPage.aspx**

**🔹 STEP 3: Add AJAX Controls to the Page**

**📄 AjaxPage.aspx**

**aspx**

**CopyEdit**

**<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="AjaxPage.aspx.cs" Inherits="AJAXDemo.AjaxPage" %>**

**<!DOCTYPE html>**

**<html>**

**<head runat="server">**

**<title>AJAX Demo Page</title>**

**</head>**

**<body>**

**<form id="form1" runat="server">**

**<!-- Required to use ASP.NET AJAX -->**

**<asp:ScriptManager ID="ScriptManager1" runat="server" />**

**<div style="padding:30px;">**

**<h2>AJAX UpdatePanel Example</h2>**

**<!-- This panel will update without refreshing the full page -->**

**<asp:UpdatePanel ID="UpdatePanel1" runat="server">**

**<ContentTemplate>**

**<asp:Label ID="lblTime" runat="server" Font-Size="Large" />**

**<br /><br />**

**<asp:Button ID="btnGetTime" runat="server" Text="Get Current Time" OnClick="btnGetTime\_Click" />**

**</ContentTemplate>**

**</asp:UpdatePanel>**

**<br /><hr /><br />**

**<p>The rest of the page won't refresh during AJAX updates!</p>**

**</div>**

**</form>**

**</body>**

**</html>**

**🔹 STEP 4: Backend Logic to Update Time**

**📄 AjaxPage.aspx.cs**

**csharp**

**CopyEdit**

**using System;**

**namespace AJAXDemo**

**{**

**public partial class AjaxPage : System.Web.UI.Page**

**{**

**protected void Page\_Load(object sender, EventArgs e)**

**{**

**}**

**protected void btnGetTime\_Click(object sender, EventArgs e)**

**{**

**lblTime.Text = "Current Server Time: " + DateTime.Now.ToString("hh:mm:ss tt");**

**}**

**}**

**}**

**🧠 HOW IT WORKS**

| **Control** | **Role** |
| --- | --- |
| **ScriptManager** | **Enables AJAX features** |
| **UpdatePanel** | **Updates only part of the page asynchronously** |
| **ContentTemplate** | **Section inside the panel that will update** |
| **Button + Label** | **On click, label updates without full page reload** |

**✅ OUTPUT EXPECTED**

1. **Page loads with a button.**
2. **On clicking "Get Current Time", only the time label updates (via AJAX).**
3. **The rest of the page remains unchanged—no full reload.**

**✅ DELIVERABLES**

* **Files: AjaxPage.aspx, AjaxPage.aspx.cs**
* **Screenshot of working time update**
* **ZIP of Visual Studio project (optional)**

**🔄 BONUS: Add More AJAX Controls**

**You can also try:**

* **UpdateProgress (shows loading animation)**
* **Timer (auto update)**
* **AutoCompleteExtender (AJAX Control Toolkit)**

**Design a Web Application to Produce and Consume a Web Service in ASP.NET."**

**🧰 TOOLS REQUIRED:**

* Visual Studio (.NET Framework)
* ASP.NET Web Forms
* SOAP Web Service (ASMX)

**🧭 WHAT YOU’LL DO:**

1. Create a **Web Service** (ASMX)
2. Host and test it locally
3. Create a **Web Application** that consumes this service

**🔹 PART 1: Create the Web Service (ASMX)**

**✅ Step 1: Create New Web Site (Service)**

1. Open Visual Studio
2. File > New > Website
3. Choose **ASP.NET Empty Website**, name it: MyWebService
4. Right-click → Add New Item → Choose **Web Service (ASMX)**, name it: MathService.asmx

**✅ Step 2: Write Web Service Code**

📄 **MathService.asmx.cs**

csharp

CopyEdit

using System;

using System.Web.Services;

namespace MyWebService

{

[WebService(Namespace = "http://myservice.org/")]

[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1\_1)]

public class MathService : WebService

{

[WebMethod]

public int Add(int a, int b)

{

return a + b;

}

[WebMethod]

public int Multiply(int x, int y)

{

return x \* y;

}

}

}

📄 **MathService.asmx**

aspx

CopyEdit

<%@ WebService Language="C#" CodeBehind="MathService.asmx.cs" Class="MyWebService.MathService" %>

**✅ Step 3: Run and Test the Service**

1. Set the project as StartUp
2. Run the project (F5)
3. Navigate to: http://localhost:[port]/MathService.asmx
4. You should see: Add and Multiply methods listed

**🔹 PART 2: Create the Web Application (Consumer)**

**✅ Step 4: Create a New Web Forms App**

1. In the same solution (or new one), add another project
2. Template: **ASP.NET Web Application (.NET Framework)**
3. Name: WebServiceConsumer
4. Choose **Web Forms**

**✅ Step 5: Add Web Reference to the Service**

1. Right-click WebServiceConsumer > Add Service Reference...
2. Click **"Advanced…" > Add Web Reference**
3. Enter the URL: http://localhost:[port]/MathService.asmx
4. Name the reference: MathReference
5. Click **Add Reference**

**✅ Step 6: Design the Consumer Page**

📄 **Default.aspx**

aspx

CopyEdit

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="WebServiceConsumer.Default" %>

<!DOCTYPE html>

<html>

<head runat="server">

<title>Web Service Consumer</title>

</head>

<body>

<form id="form1" runat="server">

<div style="padding:20px;">

<h2>Consume Web Service</h2>

<asp:TextBox ID="txtA" runat="server" />

<asp:TextBox ID="txtB" runat="server" />

<br /><br />

<asp:Button ID="btnAdd" runat="server" Text="Add" OnClick="btnAdd\_Click" />

<asp:Button ID="btnMultiply" runat="server" Text="Multiply" OnClick="btnMultiply\_Click" />

<br /><br />

<asp:Label ID="lblResult" runat="server" Font-Bold="true" />

</div>

</form>

</body>

</html>

**✅ Step 7: Code to Consume Web Service**

📄 **Default.aspx.cs**

csharp

CopyEdit

using System;

namespace WebServiceConsumer

{

public partial class Default : System.Web.UI.Page

{

protected void btnAdd\_Click(object sender, EventArgs e)

{

var service = new MathReference.MathService();

int a = int.Parse(txtA.Text);

int b = int.Parse(txtB.Text);

lblResult.Text = "Sum = " + service.Add(a, b).ToString();

}

protected void btnMultiply\_Click(object sender, EventArgs e)

{

var service = new MathReference.MathService();

int a = int.Parse(txtA.Text);

int b = int.Parse(txtB.Text);

lblResult.Text = "Product = " + service.Multiply(a, b).ToString();

}

}

}

**✅ OUTPUT**

* User enters two numbers
* Clicks Add or Multiply
* Web page calls web service
* Result is shown without server logic in the page itself

**🧠 KEY CONCEPTS**

| **Term** | **Meaning** |
| --- | --- |
| ASMX | Old-style SOAP web service in .NET |
| WebMethod | Makes method callable over the web |
| Web Reference | Allows your app to call service methods like local methods |
| Consumer | The app that uses the service |

**✅ DELIVERABLES**

* Code files:
  + MathService.asmx, .cs
  + Default.aspx, .cs
* Screenshot of:
  + Web service URL
  + Consumer working
* Project ZIP for submission

**🔄 BONUS IDEAS**

* Add more methods (e.g., Subtract, Divide)
* Use try-catch for error handling
* Convert to **WCF** or **Web API** service for modern architecture

**Design a Web Application Using the MVC Framework in ASP.NET."**

**🧰 TOOLS NEEDED:**

* **Visual Studio 2019 or later**
* **ASP.NET MVC (.NET Framework or .NET Core)**

**🧭 WHAT YOU’LL DO:**

You will:

1. Create an ASP.NET MVC Project
2. Design a Model (data)
3. Create a Controller (logic)
4. Build Views (UI)
5. Use Razor syntax to display data dynamically

**🔹 STEP 1: Create ASP.NET MVC Project**

1. Open **Visual Studio**
2. File > New > Project
3. Choose: **ASP.NET Web Application (.NET Framework)** (or .NET Core)
4. Name it: MvcStudentApp
5. Select **MVC** and click **Create**

**🔹 STEP 2: Create a Model**

Right-click on Models folder → Add > Class → Name it Student.cs

**📄 Models/Student.cs**

csharp

CopyEdit

namespace MvcStudentApp.Models

{

public class Student

{

public int Id { get; set; }

public string Name { get; set; }

public string Course { get; set; }

}

}

**🔹 STEP 3: Create a Controller**

Right-click Controllers → Add > Controller → Choose **MVC 5 Controller - Empty** → Name: StudentController

**📄 Controllers/StudentController.cs**

csharp

CopyEdit

using System.Collections.Generic;

using System.Web.Mvc;

using MvcStudentApp.Models;

namespace MvcStudentApp.Controllers

{

public class StudentController : Controller

{

// GET: Student

public ActionResult Index()

{

var students = new List<Student>

{

new Student { Id = 1, Name = "Amit", Course = "C#" },

new Student { Id = 2, Name = "Neha", Course = "ASP.NET" },

new Student { Id = 3, Name = "Ravi", Course = "MVC" }

};

return View(students);

}

}

}

**🔹 STEP 4: Create a View**

Right-click inside Index() method → Add View → Name it Index, Template: **List**, Model class: Student

Or manually add:

📄 Views/Student/Index.cshtml

c

CopyEdit

@model IEnumerable<MvcStudentApp.Models.Student>

@{

ViewBag.Title = "Student List";

}

<h2>Student List</h2>

<table border="1">

<tr>

<th>ID</th>

<th>Name</th>

<th>Course</th>

</tr>

@foreach (var s in Model)

{

<tr>

<td>@s.Id</td>

<td>@s.Name</td>

<td>@s.Course</td>

</tr>

}

</table>

**🔹 STEP 5: Run the Application**

1. Set StudentController as the start page:
   * In RouteConfig.cs, update:

csharp

CopyEdit

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Student", action = "Index", id = UrlParameter.Optional }

);

1. Run the app (F5)
2. You should see a **list of students** displayed using the MVC pattern.

**✅ OUTPUT**

| **ID** | **Name** | **Course** |
| --- | --- | --- |
| 1 | Amit | C# |
| 2 | Neha | ASP.NET |
| 3 | Ravi | MVC |

**🧠 MVC ARCHITECTURE RECAP**

| **Component** | **Role** |
| --- | --- |
| Model | Represents data (e.g., Student) |
| View | UI to present data (HTML/Razor) |
| Controller | Handles logic and data flow |

**✅ DELIVERABLES**

* Code files:
  + Student.cs (Model)
  + StudentController.cs (Controller)
  + Index.cshtml (View)
* Screenshot of:
  + Student list output in browser
* (Optional) Project ZIP for submission

**🔄 BONUS IDEAS**

* Add a form to create new students
* Connect to a database using Entity Framework
* Use TempData, ViewBag, and ModelState

**Design an MVC Application Using Entity Framework**

**🧰 TOOLS REQUIRED:**

* Visual Studio 2019 or later
* ASP.NET MVC (.NET Framework or .NET Core)
* SQL Server (Express/LocalDB)

**🧭 WHAT YOU WILL DO:**

1. Create MVC Project
2. Create Database and Entity Model (Code First or Database First)
3. Scaffold Controller & Views using EF
4. Perform CRUD operations

**🔹 STEP 1: Create ASP.NET MVC Project**

1. Open Visual Studio
2. File > New > Project
3. Select **ASP.NET Web Application (.NET Framework)** or **ASP.NET Core Web Application**
4. Name: MvcEFApp
5. Choose **MVC** template, click Create

**🔹 STEP 2: Create Database**

Create a database called **StudentDB** with a table Students:

sql

CopyEdit

CREATE TABLE Students (

Id INT PRIMARY KEY IDENTITY,

Name NVARCHAR(100),

Course NVARCHAR(100)

);

You can also let EF create it via Code First (see Step 3).

**🔹 STEP 3: Add Entity Framework to Project**

* If not installed, install EF NuGet Package:
  + For .NET Framework: EntityFramework
  + For .NET Core: Microsoft.EntityFrameworkCore.SqlServer

**🔹 STEP 4: Create Model and DbContext (Code First Approach)**

**Add folder Models**

Add Student.cs in Models folder:

csharp

CopyEdit

namespace MvcEFApp.Models

{

public class Student

{

public int Id { get; set; }

public string Name { get; set; }

public string Course { get; set; }

}

}

Add StudentContext.cs in Models folder:

csharp

CopyEdit

using System.Data.Entity;

namespace MvcEFApp.Models

{

public class StudentContext : DbContext

{

public StudentContext() : base("name=StudentContext")

{

}

public DbSet<Student> Students { get; set; }

}

}

**🔹 STEP 5: Configure Connection String**

Open **Web.config**, add inside <configuration><connectionStrings>:

xml

CopyEdit

<connectionStrings>

<add name="StudentContext" connectionString="Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=StudentDB;Integrated Security=True" providerName="System.Data.SqlClient" />

</connectionStrings>

**🔹 STEP 6: Enable Migrations (Optional)**

Open **Package Manager Console**:

powershell

CopyEdit

Enable-Migrations

Add-Migration InitialCreate

Update-Database

This creates the database and table from your model.

**🔹 STEP 7: Scaffold Controller and Views**

1. Right-click **Controllers** → Add > Controller
2. Choose **MVC 5 Controller with views, using Entity Framework**
3. Model class: Student
4. Data context class: StudentContext
5. Controller name: StudentController
6. Click Add

This auto-generates CRUD views and controller code.

**🔹 STEP 8: Run and Test Application**

* Set default route to Student Controller:

In RouteConfig.cs:

csharp

CopyEdit

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Student", action = "Index", id = UrlParameter.Optional }

);

* Run app (F5), CRUD UI for Students appears.

**✅ EXPECTED FEATURES:**

* List all students
* Create new student
* Edit student details
* Delete student
* Details view

**🔹 Sample StudentController (auto-generated):**

csharp

CopyEdit

public class StudentController : Controller

{

private StudentContext db = new StudentContext();

// GET: Student

public ActionResult Index()

{

return View(db.Students.ToList());

}

// GET: Student/Create

public ActionResult Create()

{

return View();

}

// POST: Student/Create

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create(Student student)

{

if (ModelState.IsValid)

{

db.Students.Add(student);

db.SaveChanges();

return RedirectToAction("Index");

}

return View(student);

}

// Other actions: Edit, Delete, Details similarly

}

**🧠 SUMMARY**

| **Component** | **Role** |
| --- | --- |
| Entity Framework | ORM to interact with database |
| DbContext | Session between code and database |
| Models | Represent data structure |
| Controller | Handle requests, use EF to query database |
| Views | Display data and forms to user |

**✅ DELIVERABLES:**

* Models/Student.cs
* Models/StudentContext.cs
* Scaffolded StudentController.cs and Views
* Web.config with connection string
* Screenshot of running CRUD app

**🔄 BONUS:**

* Add validation attributes on Model
* Use async EF methods
* Implement search/filter on student list

**Design an MVC Application to Perform CRUD Operations"**

**🧰 TOOLS REQUIRED:**

* Visual Studio (2017/2019/2022)
* ASP.NET MVC (.NET Framework)
* Entity Framework (Code First)

**🧭 WHAT YOU WILL DO:**

* Create MVC Project
* Create a Model representing data
* Create DbContext with EF
* Scaffold Controller and Views for CRUD
* Test CRUD operations on a browser

**Step-by-step Guide**

**🔹 STEP 1: Create New MVC Project**

1. Open Visual Studio.
2. File > New > Project
3. Select **ASP.NET Web Application (.NET Framework)**.
4. Name it: MvcCrudApp.
5. Select **MVC** template and click **Create**.

**🔹 STEP 2: Add Model**

Add a folder named Models (if not already there).

Add a class Product.cs inside Models:

csharp

CopyEdit

namespace MvcCrudApp.Models

{

public class Product

{

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

public string Name { get; set; }

public decimal Price { get; set; }

public string Description { get; set; }

}

}

**🔹 STEP 3: Add DbContext**

Add a new class ProductContext.cs inside Models:

csharp

CopyEdit

using System.Data.Entity;

namespace MvcCrudApp.Models

{

public class ProductContext : DbContext

{

public ProductContext() : base("name=ProductContext") { }

public DbSet<Product> Products { get; set; }

}

}

**🔹 STEP 4: Configure Connection String**

Add in Web.config inside <connectionStrings>:

xml

CopyEdit

<connectionStrings>

<add name="ProductContext"

connectionString="Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=ProductDB;Integrated Security=True"

providerName="System.Data.SqlClient" />

</connectionStrings>

**🔹 STEP 5: Enable Migrations (Optional, for Code First DB Creation)**

Open **Package Manager Console** (Tools > NuGet Package Manager > Package Manager Console) and run:

powershell

CopyEdit

Enable-Migrations

Add-Migration InitialCreate

Update-Database

**🔹 STEP 6: Scaffold Controller with Views (CRUD)**

1. Right-click Controllers folder > Add > Controller.
2. Choose **MVC 5 Controller with views, using Entity Framework**.
3. Model class: Product (from Models).
4. Data context class: ProductContext.
5. Controller name: ProductController.
6. Click **Add**.

Visual Studio auto-generates controller actions and strongly typed views for CRUD.

**🔹 STEP 7: Set Default Route to Product Controller**

Edit RouteConfig.cs to start with Product controller:

csharp

CopyEdit

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Product", action = "Index", id = UrlParameter.Optional }

);

**🔹 STEP 8: Run and Test**

1. Press F5 or Ctrl + F5 to run.
2. The app should open the list of products.
3. You can **Create**, **Edit**, **Details**, and **Delete** products via the web UI.

**Sample Controller Snippet (Auto-generated):**

csharp

CopyEdit

public class ProductController : Controller

{

private ProductContext db = new ProductContext();

public ActionResult Index()

{

return View(db.Products.ToList());

}

public ActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create(Product product)

{

if (ModelState.IsValid)

{

db.Products.Add(product);

db.SaveChanges();

return RedirectToAction("Index");

}

return View(product);

}

// Edit, Details, Delete methods follow similar patterns

}

**Summary of Operations**

| **Action** | **Purpose** |
| --- | --- |
| Index | Display list of records |
| Create | Insert new record |
| Edit | Update existing record |
| Details | View details of a record |
| Delete | Remove record from database |

**Optional Enhancements**

* Add validation attributes to Model properties (like [Required], [Range])
* Style views using Bootstrap or CSS
* Add search/filter functionality