DEPSTAR

Devang Patel Institute of Advance Technology and Research

(A Constitute Institute of CHARUSAT)

Certificate

This is to certify that

Mr./Mrs. Garala Dhruvi

of 5CSE-1 *Class,*

ID. No. 23DCS029 *has satisfactorily completed*

his/ her term work in .Net Core Programming (CSE307) *for*

the ending in Oct 2025/2026

Date : 6/10/2025

K.S. Patel

Sign. of Faculty

Head of Department

PRACTICAL: 1

AIM:

Design a console-based application in C# to track employee attendance within an organization. The system should allow HR personnel to record daily attendance, display all attendance entries, and calculate the number of days an employee was present. A user-friendly console interface should manage all interactions, with data stored in-memory during runtime

CODE:

EmployeeAttendance.cs using
System;
namespace employeeattendence

```
{    public class
EmployeeAttendance
   {

      public int EmployeeId { get; set; }
public string EmployeeName { get; set; }
public DateTime Date { get; set; }        public
bool IsPresent { get; set; }


      public EmployeeAttendance(int employeeId, string employeeName, DateTime date, bool isPresent)

      {

         EmployeeId = employeeId;

         EmployeeName = employeeName;

         Date = date;

         IsPresent = isPresent;

      }

   }

}
```

AttendanceManager.cs using
employeeattendence; using
System; using
System.Collections.Generic;

```
namespace employeeattendence

{    public class
AttendanceManager
   {

      private List<EmployeeAttendance> attendanceRecords;

      public AttendanceManager()

      {

         attendanceRecords = new List<EmployeeAttendance>();

      }

      public void AddAttendance(EmployeeAttendance record)

      {

         attendanceRecords.Add(record);

         Console.WriteLine("Attendance record added successfully!\n");        }

      public void DisplayAllRecords()

      {

         if (attendanceRecords.Count == 0)

         {

            Console.WriteLine("No attendance records found.\n");

            return;

         }

         Console.WriteLine("\n===== Attendance Records =====");
foreach (var record in attendanceRecords)
         {

            Console.WriteLine($"ID: {record.EmployeeId}, Name: {record.EmployeeName}, Date:
{record.Date.ToShortDateString()}, Present: {record.IsPresent}");

         }

         Console.WriteLine();

      }
```

```
    public void GetTotalPresentDays(int employeeId)

    {           int count = 0;          foreach (var
record in attendanceRecords)
        {

            if (record.EmployeeId == employeeId && record.IsPresent)

            {
count++;
            }

        }



        Console.WriteLine($"Total days present for Employee ID {employeeId}: {count}\n");

    }

  }

}
```

Program.cs using
employeeattendence; using
System;

```
namespace EmployeeAttendanceTracker

{   class
Program
  {       static void Main(string[]
args)
    {

        AttendanceManager manager = new AttendanceManager();

        bool exit = false;


do
        {

            Console.WriteLine("===== Employee Attendance Tracker =====");
Console.WriteLine("1. Add Attendance Record");

            Console.WriteLine("2. View All Attendance Records");
```

```
            Console.WriteLine("3. Get Total Present Days by Employee ID");
            Console.WriteLine("4. Exit");
            Console.Write("Enter your choice: ");

            string choiceInput = Console.ReadLine();
            int choice;                 if
(!int.TryParse(choiceInput, out choice))
            {
               Console.WriteLine("Invalid input! Please enter a valid number.\n");
               continue;
            }

            switch (choice)
            {
case 1:
                  AddAttendanceRecord(manager);
                  break;
               case 2:
                  manager.DisplayAllRecords();
                  break;
               case 3:
                  GetTotalPresentDays(manager);
                  break;
               case 4:
                  exit = true;
                  Console.WriteLine("Exiting... Goodbye!");
                  break;
               default:
                  Console.WriteLine("Invalid choice. Please select a valid option.\n");
```

```
                    break;
                }
            } while (!exit);
        }
        static void AddAttendanceRecord(AttendanceManager manager)
        {
try
            {
 Console.Write("Enter Employee ID: ");
int empId = int.Parse(Console.ReadLine());

Console.Write("Enter Employee Name: ");
string empName = Console.ReadLine();
Console.Write("Enter Date (yyyy-MM-dd): ");
string dateInput = Console.ReadLine();
DateTime date;
 if (!DateTime.TryParse(dateInput, out date))
                {
                    Console.WriteLine("Invalid date format! Please use yyyy-MM-dd.\n");
                    return;
                }
                Console.Write("Is Employee Present? (Y/N): ");
  string presenceInput = Console.ReadLine();
bool isPresent = false;


                if (presenceInput.Equals("Y", StringComparison.OrdinalIgnoreCase))
                {
isPresent = true;
                }
                else if (presenceInput.Equals("N", StringComparison.OrdinalIgnoreCase))
                {
isPresent = false;
                }
else
                {
```

```
                Console.WriteLine("Invalid input! Please enter Y or N.\n");
                return;
            }

            EmployeeAttendance record = new EmployeeAttendance(empId, empName, date, isPresent);
manager.AddAttendance(record);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error: {ex.Message}\n");
        }
    }

    static void GetTotalPresentDays(AttendanceManager manager)
    {
try
        {
            Console.Write("Enter Employee ID: ");
int empId = int.Parse(Console.ReadLine());
manager.GetTotalPresentDays(empId);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error: {ex.Message}\n");
        }
    }
  }
}
```

OUTPUT:

```
===== Employee Attendance System =====
1. Record Attendance
2. Display All Attendance Records
3. Calculate Days Present
4. Exit
Enter your choice: 1
Enter Employee ID: 123
Enter Employee Name: dhruvi
Enter Date (yyyy-mm-dd): 04-10-20
Is employee present on this date? (y/n): y
? Attendance recorded successfully!

===== Employee Attendance System =====
1. Record Attendance
2. Display All Attendance Records
3. Calculate Days Present
4. Exit
Enter your choice: 1
Enter Employee ID: 456
Enter Employee Name: krishna
Enter Date (yyyy-mm-dd): 06-08-20
Is employee present on this date? (y/n): n
? Attendance recorded successfully!
```

```
===== Employee Attendance System =====
1. Record Attendance
2. Display All Attendance Records
3. Calculate Days Present
4. Exit
Enter your choice: 2

===== Attendance Records =====
04-10-2020 - ID: 123 - Name: dhruvi - Status: Present
06-08-2020 - ID: 456 - Name: krishna - Status: Absent

===== Employee Attendance System =====
1. Record Attendance
2. Display All Attendance Records
3. Calculate Days Present
4. Exit
Enter your choice: 3
Enter Employee ID to calculate days present: 123
? Employee ID 123 was present for 1 day(s).

===== Employee Attendance System =====
1. Record Attendance
2. Display All Attendance Records
3. Calculate Days Present
4. Exit
Enter your choice: 4
Exiting program. Goodbye!
```

Conclusion:

The console-based system successfully records, displays, and calculates employee attendance using an EmployeeAttendance class and AttendanceManager. It ensures separation of concerns, proper input validation, and a user-friendly menu-driven interface, fulfilling all attendance management requirements in-memory.

PRACTICAL: 2

AIM:

The Faculty Profile Management System is a web-based application designed to streamline the process of recording and managing academic and professional details of faculty members in an educational institution. The system provides an intuitive user interface where users can input, review, and maintain essential profile information in a structured format.

All modules of the application are integrated within a consistent layout to simulate a real-world enterprise system. The solution ensures form validation, secure access, automated identifier generation, and smooth navigation across different functional pages. Designed to operate entirely without database dependency, this system leverages memory-based data handling and logical structures to simulate record storage and access behavior effectively.

I: Faculty Profile Form

Design and implement an interactive form that captures essential details about faculty members for institutional records.

Fields to Capture:

- Full Name
- Employee ID (Auto-generated)
- Department
- Highest Qualification
- Specialization
- Experience (in years)
- Email ID
- Mobile Number

Implementation:

- The form will be developed using an ASP.NET Web Form (.aspx) and its associated code-behind file.
- Standard web controls like textboxes, dropdowns, and radio buttons will be used to capture inputs.
- ASP.NET validation controls will be applied to ensure correctness, including checks for required fields, valid email/mobile formats, and a proper range for numeric values.
- Faculty data entered will be temporarily stored in session to preserve user input across multiple pages or refresh actions.
- ViewState will be used to retain control-level values during postbacks.
- A unique Employee ID will be automatically generated based on the faculty's initials and system timestamp to ensure uniqueness.

II: Master Page Integration Ensure a consistent, professional, and navigable layout across all system pages using a master page structure.

Implementation:

- A single Master Page will be created to define a uniform layout for the application.
- The layout will include:
- A header with the institute's logo and title
- A navigation menu for module switching (e.g., profile entry, list view, logout)
- A footer with dynamic content such as the current year and contact details
- A welcome banner displaying the logged-in user's name
- The master page will dynamically highlight the active page in the navigation menu for improved user guidance.
- Session validation will be integrated to restrict access to authenticated users only. Unauthorized users will be redirected to the login page.
- A ContentPlaceHolder will be used to inject page-specific content without altering the master layout, promoting code reuse and consistency.

III: Session-Based Simulated Login

Implement a basic login mechanism that controls access to the system and simulates role-based authorization without requiring a database.

Implementation:

- A dedicated login page will allow users to enter a predefinedusername and password combination to gain access.
- Authentication will be based on hardcoded credentials, simulating a basic admin login experience.
- Upon successful authentication, the user's identity will be stored in a session variable and made available throughout the system.
- If authentication fails, users will receive an appropriate message prompting them to retry.
- A logout option will be available, which clears the session and redirects the user to the login page, ensuring proper session handling and security.

CODE:

Site.Master:

```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site.master.cs"
Inherits="FacultyProfileSystem_prac2.SiteMaster" %>

<!DOCTYPE html>
<html lang="en">
<head runat="server">
    <meta charset="utf-8" />
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title><%: Page.Title %> - Faculty Profile Management System</title>

    <asp:PlaceHolder runat="server">
        <%: Scripts.Render("~/bundles/modernizr") %>
    </asp:PlaceHolder>

    <webopt:bundlereference runat="server" path="~/Content/css" />
    <link href="~/favicon.ico" rel="shortcut icon" type="image/x-icon" />
</head>
<body>
    <form runat="server">
        <asp:ScriptManager runat="server">
            <Scripts>
                <asp:ScriptReference Name="MsAjaxBundle" />
                <asp:ScriptReference Name="jquery" />
                <asp:ScriptReference Name="WebForms.js" Assembly="System.Web"
Path="~/Scripts/WebForms/WebForms.js" />
                <asp:ScriptReference Name="WebUIValidation.js" Assembly="System.Web"
Path="~/Scripts/WebForms/WebUIValidation.js" />
                <asp:ScriptReference Name="MenuStandards.js" Assembly="System.Web"
Path="~/Scripts/WebForms/MenuStandards.js" />
                <asp:ScriptReference Name="GridView.js" Assembly="System.Web"
Path="~/Scripts/WebForms/GridView.js" />
                <asp:ScriptReference Name="DetailsView.js" Assembly="System.Web"
Path="~/Scripts/WebForms/DetailsView.js" />
                <asp:ScriptReference Name="TreeView.js" Assembly="System.Web"
Path="~/Scripts/WebForms/TreeView.js" />
                <asp:ScriptReference Name="WebParts.js" Assembly="System.Web"
Path="~/Scripts/WebForms/WebParts.js" />
                <asp:ScriptReference Name="Focus.js" Assembly="System.Web"
Path="~/Scripts/WebForms/Focus.js" />
                <asp:ScriptReference Name="WebFormsBundle" />
            </Scripts>
        </asp:ScriptManager>

        <!-- Navigation Bar -->
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-dark bg-
dark">
            <div class="container">
                <a class="navbar-brand" runat="server" href="~/">
                    <img src="https://colleges18.s3.ap-south-
1.amazonaws.com/Charotar_University_of_Science_and_Technology_CHARUSAT_An
and_logo_jpg_eb45ac6cc5.webp"
                         alt="Logo" style="height:70px; vertical-align: middle;" />
                    <span style="margin-left: 10px;">Faculty Profile Management
System</span>
                </a>
                <button type="button" class="navbar-toggler" data-bs-toggle="collapse" data-
bs-target=".navbar-collapse"
```

```
                    title="Toggle navigation" aria-controls="navbarSupportedContent" aria-
expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse d-sm-inline-flex justify-content-
between">
                <ul class="navbar-nav flex-grow-1">
                    <li class="nav-item">
                        <a class="nav-link" runat="server" href="~/FacultyForm.aspx">Profile
Entry</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" runat="server" href="~/FacultyList.aspx">List
View</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="#" runat="server"
onserverclick="Logout_Click">Logout</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>

    <!-- Page Body -->
    <div class="container body-content mt-3">
        <!-- Welcome Message -->
        <asp:Label ID="lblWelcome" runat="server" CssClass="text-success font-
weight-bold" Font-Size="Large"></asp:Label>
        <br /><br />

        <!-- Content Placeholder -->
        <asp:ContentPlaceHolder ID="MainContent" runat="server" />
        <hr />

        <!-- Footer -->
        <footer class="text-center mb-3">
            <p>&copy; <%: DateTime.Now.Year %> - Faculty Profile System | Contact:
info@institute.edu</p>
        </footer>
    </div>
</form>

<asp:PlaceHolder runat="server">
    <%: Scripts.Render("~/Scripts/bootstrap.js") %>
</asp:PlaceHolder>
</body>
</html>
```

Faculty Form:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="FacultyForm.aspx.cs"
Inherits="FacultyProfileSystem_prac2.FacultyForm" MasterPageFile="~/Site.Master"
%>

<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="server">
   <h2>Faculty Profile Entry</h2>

   <asp:Label Text="Full Name:" runat="server" AssociatedControlID="txtFullName"
/>
   <asp:TextBox ID="txtFullName" runat="server" /><br /><br />

   <asp:Label Text="Department:" runat="server"
AssociatedControlID="ddlDepartment" />
   <asp:DropDownList ID="ddlDepartment" runat="server">
      <asp:ListItem Text="--Select--" Value="" />
      <asp:ListItem Text="CSE" />
      <asp:ListItem Text="IT" />
      <asp:ListItem Text="ECE" />
   </asp:DropDownList><br /><br />

   <asp:Label Text="Highest Qualification:" runat="server"
AssociatedControlID="txtQualification" />
   <asp:TextBox ID="txtQualification" runat="server" /><br /><br />

   <asp:Label Text="Specialization:" runat="server"
AssociatedControlID="txtSpecialization" />
   <asp:TextBox ID="txtSpecialization" runat="server" /><br /><br />

   <asp:Label Text="Experience (Years):" runat="server"
AssociatedControlID="txtExperience" />
   <asp:TextBox ID="txtExperience" runat="server" /><br /><br />

   <asp:Label Text="Email ID:" runat="server" AssociatedControlID="txtEmail" />
   <asp:TextBox ID="txtEmail" runat="server" /><br /><br />

   <asp:Label Text="Mobile Number:" runat="server"
AssociatedControlID="txtMobile" />
   <asp:TextBox ID="txtMobile" runat="server" /><br /><br />

   <asp:Button ID="btnSubmit" runat="server" Text="Submit"
OnClick="btnSubmit_Click" /><br /><br />

   <asp:Label ID="lblResult" runat="server" Text=""></asp:Label>
</asp:Content>
```

FacultyForm.aspx.cs:

```
using System;
using System.Web.UI;

namespace FacultyProfileSystem_prac2
{
   public partial class FacultyForm : Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         if (!IsPostBack)
         {
            // Only generate on initial load
            txtFullName.Focus();
         }
      }

      protected void btnSubmit_Click(object sender, EventArgs e)
      {
         if (Page.IsValid)
         {
            string fullName = txtFullName.Text;
            string initials = GetInitials(fullName);
            string empID = initials + DateTime.Now.Ticks.ToString().Substring(10); //
unique ID

            var faculty = new
            {
               EmployeeID = empID,
               FullName = fullName,
               Department = ddlDepartment.SelectedValue,
               Qualification = txtQualification.Text,
               Specialization = txtSpecialization.Text,
               Experience = txtExperience.Text,
               Email = txtEmail.Text,
               Mobile = txtMobile.Text
            };

            Session["Faculty"] = faculty;

            lblResult.Text = $"Faculty Profile Submitted Successfully!<br />Generated ID:
{empID}";
         }
      }

      private string GetInitials(string name)
      {
         string[] parts = name.Split(' ');
         string initials = "";
```

```
            foreach (var part in parts)
            {
                if (!string.IsNullOrWhiteSpace(part))
                    initials += part[0];
            }
            return initials.ToUpper();
        }
    }
}
```

Login.aspx.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace FacultyProfileSystem_prac2
{
    public partial class WebForm3 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            // Page load logic can go here if needed
        }

        protected void btnLogin_Click(object sender, EventArgs e)
        {
            string username = txtUsername.Text;
            string password = txtPassword.Text;

            if (username == "admin" && password == "admin123")  // hardcoded credentials
            {
                Session["User"] = username;
                Response.Redirect("FacultyForm.aspx");
            }
            else
            {
                lblMessage.Text = "Invalid username or password!";
            }
        }
    }
}
```

Login.aspx:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Login.aspx.cs"
Inherits="FacultyProfileSystem_prac2.WebForm3" %>

<!DOCTYPE html>
<html>
<head>
   <title>Login</title>
</head>
<body>
   <form runat="server">
      <h2>Admin Login</h2>
      <asp:Label Text="Username:" runat="server" />
      <asp:TextBox ID="txtUsername" runat="server" /><br /><br />

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

      <asp:Button ID="btnLogin" runat="server" Text="Login"
OnClick="btnLogin_Click" /><br /><br />
      <asp:Label ID="lblMessage" runat="server" ForeColor="Red" />
   </form>
</body>
</html>
```

FacultyList.aspx:

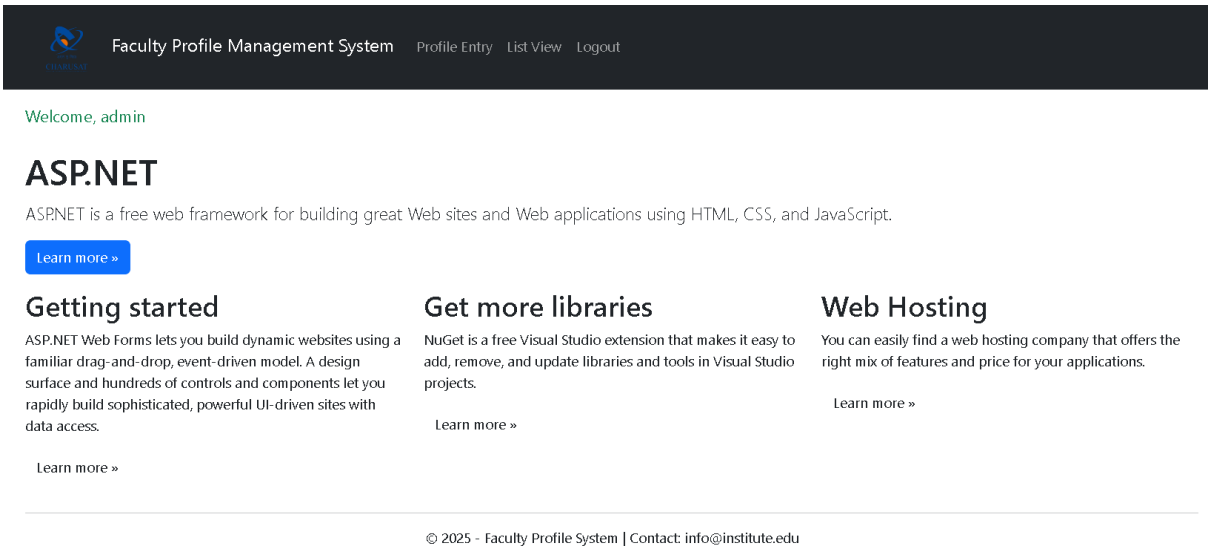
```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="FacultyList.aspx.cs" Inherits="FacultyProfileSystem_prac2.FacultyList"
MasterPageFile="~/Site.Master" %>

<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="server">
   <h2>Faculty Details</h2>
   <asp:Label ID="lblFacultyDetails" runat="server" Font-Size="Large" />
</asp:Content>
```

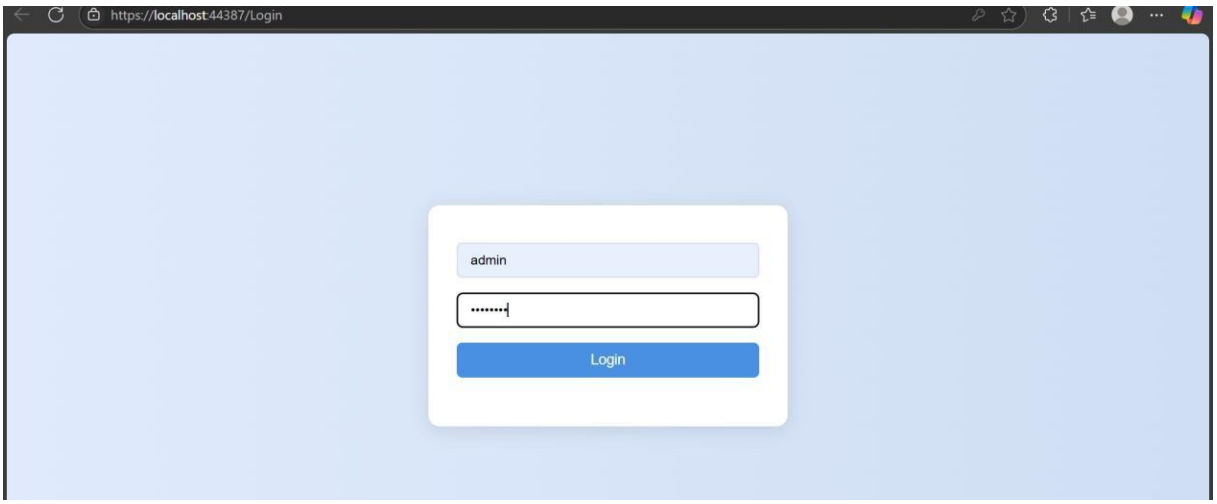
FacultyList.aspx.cs:

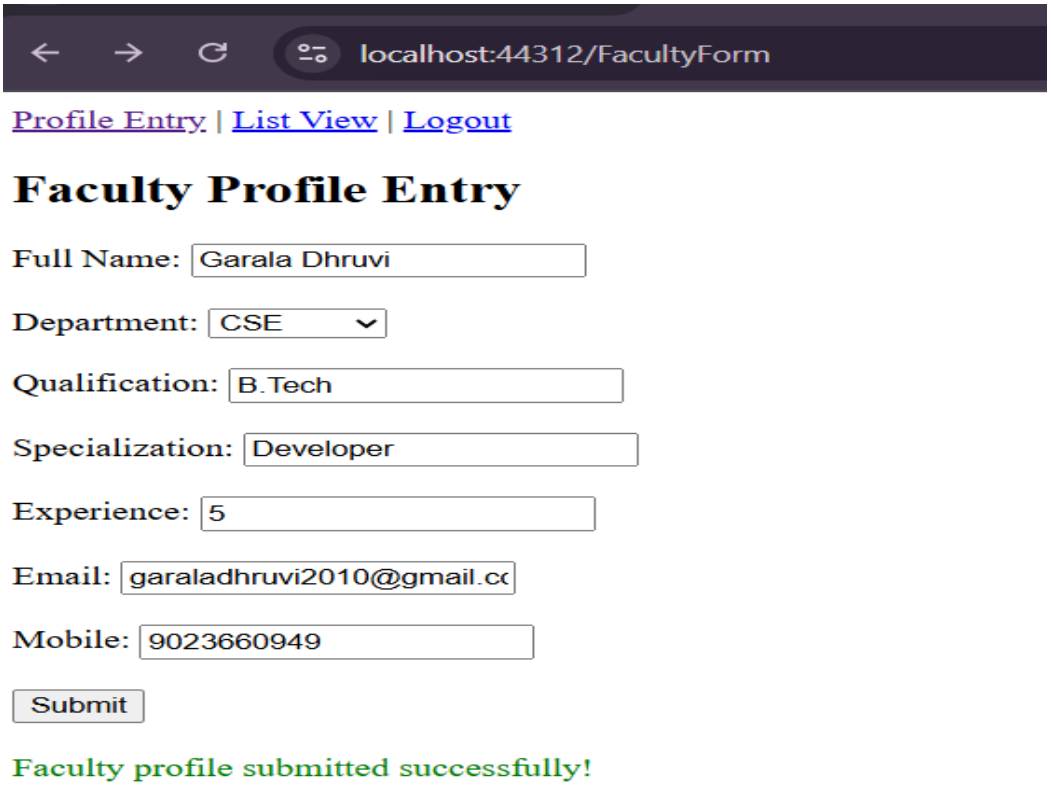
```
using System;
namespace FacultyProfileSystem_prac2
{
   public partial class FacultyList : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         if (Session["Faculty"] != null)
         {
            var faculty = (dynamic)Session["Faculty"];
            lblFacultyDetails.Text = $@"
               <b>Employee ID:</b> {faculty.EmployeeID}<br />
```

```
                <b>Full Name:</b> {faculty.FullName}<br />
                <b>Department:</b> {faculty.Department}<br />
                <b>Qualification:</b> {faculty.Qualification}<br />
                <b>Specialization:</b> {faculty.Specialization}<br />
                <b>Experience:</b> {faculty.Experience} years<br />
                <b>Email:</b> {faculty.Email}<br />
                <b>Mobile:</b> {faculty.Mobile}";
            }
            else
            {
                lblFacultyDetails.Text = "No faculty data found.";
            }
        }
    }
}
```

OUTPUT:







Conclusion:

The Faculty Profile System effectively captures faculty details with validation, auto-generated IDs, and session-based data handling. Master page integration ensures a consistent layout, while session-based login and role simulation provide controlled access, achieving a secure and professional faculty management interface.

PRACTICAL: 3

AIM: Develop a Student Management System using ASP.NET Web Forms and ADO.NET that allows for efficient student data handling, secure session management, and a consistent user interface. The system must align with modular design principles and maintain data integrity through structured validation and efficient data access methods.

System Requirements

I. Student Enrollment Management Module

Design and implement a student enrollment form that captures essential student information and ensures proper validation for mandatory fields including:

- Student Name
- Roll Number
- Course
- Enrollment Date
- Email ID
- Mobile Number

Technical requirements:

- Implement ASP.NET validation controls (e.g., RequiredFieldValidator, RegularExpressionValidator) to enforce input correctness.
- Use Session State to temporarily store form data to handle navigation without losing entered information.
- Use ADO.NET DataSet for inserting and updating data.
- Use ADO.NET DataReader for fast, forward-only retrieval of student records for read-only views.
- Auto-generate a unique StudentID using a logic that combines timestamp and student initials to ensure uniqueness across entries.

II. Admin Dashboard for Student Data Management

Develop an admin panel using GridView for complete student data management, including functionalities for viewing, adding, editing, and deleting records.

Technical requirements:

- Populate GridView using DataReader for display operations and DataSet for data manipulation.
- Enable paging in GridView to manage large volumes of student records.
- Include editing and deleting capabilities with confirmation dialogs.
- Allow data filtering based on specific attributes such as Course or Enrollment Year using dropdowns or text inputs.

Tasks:

● Task 1: Create a New Console Application in C# Named LinqOperationsDemo

● Task 2: Define a Class Product with the appropriate Properties: (Product ID, Name, Category, Price, Stock Quantity)

● Task 3: Create a List of Products and Populate it with at Least 6 Sample Records

● Task 4: Perform the Following LINQ Operations Using Method Syntax:

- Display all products in the list
- Filter products where Price > 500
- Get products that belong to a specific Category (e.g.,
- "Electronics")
- Sort products by Name in ascending order
- Calculate the total stock quantity of all products
- Find the most expensive product

● Task 5: Perform the Following LINQ Operations Using Query Syntax:

- Select only product names from the list
- Group products by Category
- Count the number of products in each category
- Find average price of all products

● Task 6: Display the Output of Each Query in the Console with Proper Labels

CODE:

StudentEnrollment.aspx

```
<%@ Page Title="Student Enrollment" Language="C#" MasterPageFile="~/Site.Master"
    AutoEventWireup="true" CodeBehind="StudentEnrollment.aspx.cs"
    Inherits="StudentManagementSystem.StudentEnrollment" %>

<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="server">
    <h2>Student Enrollment</h2>

    <!-- Student Name -->
    <asp:TextBox ID="txtName" runat="server" placeholder="Student Name"></asp:TextBox>
    <asp:RequiredFieldValidator ID="rfvName" runat="server"
        ControlToValidate="txtName" ErrorMessage="Name is required" ForeColor="Red" />
    <br /><br />

    <!-- Roll Number -->
    <asp:TextBox ID="txtRollNumber" runat="server" placeholder="Roll Number"
        OnTextChanged="AnyFieldChanged" AutoPostBack="true"></asp:TextBox>
```

```
<asp:RequiredFieldValidator ID="rfvRoll" runat="server"
   ControlToValidate="txtRollNumber" ErrorMessage="Roll Number is required"
   ForeColor="Red" />
<br /><br />

<!-- Course Selection -->
<asp:DropDownList ID="ddlCourse" runat="server" AutoPostBack="true"
   OnSelectedIndexChanged="AnyFieldChanged">
   <asp:ListItem Text="Select Course" Value="" />
   <asp:ListItem Text="CSE" Value="CSE" />
   <asp:ListItem Text="CE" Value="CE" />
   <asp:ListItem Text="IT" Value="IT" />
</asp:DropDownList>
<asp:RequiredFieldValidator ID="rfvCourse" runat="server"
   ControlToValidate="ddlCourse" InitialValue=""
   ErrorMessage="Select a course" ForeColor="Red" />
<br /><br />

<!-- Enrollment Date -->
<asp:TextBox ID="txtEnrollmentDate" runat="server" TextMode="Date"
   OnTextChanged="AnyFieldChanged" AutoPostBack="true"></asp:TextBox>
<asp:RequiredFieldValidator ID="rfvDate" runat="server"
   ControlToValidate="txtEnrollmentDate" ErrorMessage="Enrollment Date is required"
   ForeColor="Red" />
<br /><br />

<!-- Email -->
<asp:TextBox ID="txtEmail" runat="server" placeholder="Email ID"
   OnTextChanged="AnyFieldChanged" AutoPostBack="true"></asp:TextBox>
<asp:RequiredFieldValidator ID="rfvEmail" runat="server"
   ControlToValidate="txtEmail" ErrorMessage="Email is required" ForeColor="Red" />
<asp:RegularExpressionValidator ID="revEmail" runat="server"
   ControlToValidate="txtEmail"
   ValidationExpression="^[\w\.-]+@[\w\.-]+\.\w+$"
   ErrorMessage="Invalid email format" ForeColor="Red" />
<br /><br />

<!-- Mobile -->
<asp:TextBox ID="txtMobile" runat="server" placeholder="Mobile Number"
   OnTextChanged="AnyFieldChanged" AutoPostBack="true"></asp:TextBox>
<asp:RequiredFieldValidator ID="rfvMobile" runat="server"
   ControlToValidate="txtMobile" ErrorMessage="Mobile Number is required"
   ForeColor="Red" />
<asp:RegularExpressionValidator ID="revMobile" runat="server"
   ControlToValidate="txtMobile"
   ValidationExpression="^[0-9]{10}$"
   ErrorMessage="Enter a valid 10-digit mobile number" ForeColor="Red" />
<br /><br />

<!-- Submit Button -->
```

```
    <asp:Button ID="btnSubmit" runat="server" Text="Enroll Student"
        OnClick="btnSubmit_Click" />
    <br /><br />

    <!-- Success/Error Label -->
    <asp:Label ID="lblMessage" runat="server" ForeColor="Green"></asp:Label>
</asp:Content>
```

StudentEnrollement.aspx.cs:

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;

namespace StudentManagementSystem
{
    public partial class StudentEnrollment : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                // Restore data if available from session
                if (Session["Name"] != null) txtName.Text = Session["Name"].ToString();
                if (Session["RollNumber"] != null) txtRollNumber.Text =
Session["RollNumber"].ToString();
                if (Session["Course"] != null) ddlCourse.SelectedValue = Session["Course"].ToString();
                if (Session["EnrollmentDate"] != null) txtEnrollmentDate.Text =
Session["EnrollmentDate"].ToString();
                if (Session["Email"] != null) txtEmail.Text = Session["Email"].ToString();
                if (Session["Mobile"] != null) txtMobile.Text = Session["Mobile"].ToString();
            }
        }

        // Triggered when any field with AutoPostBack changes
        protected void AnyFieldChanged(object sender, EventArgs e)
        {
            SaveFormToSession();
        }

        protected void btnSubmit_Click(object sender, EventArgs e)
        {
            // Save form values into session before processing
            SaveFormToSession();

            // Only proceed if all ASP.NET validators are valid
            if (!Page.IsValid)
```

```
            return;

        string studentID = GenerateStudentID(txtName.Text);

        try
        {
            using (SqlConnection con = new
SqlConnection(ConfigurationManager.ConnectionStrings["StudentDBConn"].ConnectionString))
            {
                con.Open();
                string query = @"INSERT INTO Students
                                (StudentID, StudentName, RollNumber, Course, EnrollmentDate, EmailID,
MobileNumber)
                                VALUES
                                (@StudentID, @Name, @Roll, @Course, @Date, @Email, @Mobile)";

                using (SqlCommand cmd = new SqlCommand(query, con))
                {
                    cmd.Parameters.AddWithValue("@StudentID", studentID);
                    cmd.Parameters.AddWithValue("@Name", txtName.Text.Trim());
                    cmd.Parameters.AddWithValue("@Roll", txtRollNumber.Text.Trim());
                    cmd.Parameters.AddWithValue("@Course", ddlCourse.SelectedValue);
                    cmd.Parameters.AddWithValue("@Date",
DateTime.Parse(txtEnrollmentDate.Text));
                    cmd.Parameters.AddWithValue("@Email", txtEmail.Text.Trim());
                    cmd.Parameters.AddWithValue("@Mobile", txtMobile.Text.Trim());

                    cmd.ExecuteNonQuery();
                }
            }

            // Clear session so form doesn't repopulate next time
            Session.Clear();

            // Show alert to user
            Response.Write("<script>alert('Student Enrolled Successfully!');</script>");

            // Optionally also clear fields visually
            ClearForm();
        }
        catch (Exception ex)
        {
            lblMessage.ForeColor = System.Drawing.Color.Red;
            lblMessage.Text = "Error: " + ex.Message;
        }
    }

    // Save all current form values into session
    private void SaveFormToSession()
    {
```

```
            Session["Name"] = txtName.Text;
            Session["RollNumber"] = txtRollNumber.Text;
            Session["Course"] = ddlCourse.SelectedValue;
            Session["EnrollmentDate"] = txtEnrollmentDate.Text;
            Session["Email"] = txtEmail.Text;
            Session["Mobile"] = txtMobile.Text;
        }

        // Generate a unique Student ID using initials + timestamp
        private string GenerateStudentID(string name)
        {
            string initials = name.Length >= 2 ? name.Substring(0, 2).ToUpper() : name.ToUpper();
            string timestamp = DateTime.Now.ToString("yyyyMMddHHmmss");
            return initials + timestamp;
        }

        // Clear all form inputs
        private void ClearForm()
        {
            txtName.Text = "";
            txtRollNumber.Text = "";
            ddlCourse.SelectedIndex = 0;
            txtEnrollmentDate.Text = "";
            txtEmail.Text = "";
            txtMobile.Text = "";
        }
    }
}
```

AdminDashboard.aspx:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="AdminDashboard.aspx.cs"
    Inherits="StudentManagementSystem.AdminDashboard" %>

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Admin Dashboard</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <h2>Admin Dashboard - Student Management</h2>

            <asp:DropDownList
                ID="ddlFilterCourse"
                runat="server"
                AutoPostBack="True"
                OnSelectedIndexChanged="ddlFilterCourse_SelectedIndexChanged">
```

```
        <asp:ListItem Text="All Courses" Value="" />
        <asp:ListItem Text="CSE" Value="IT" />
        <asp:ListItem Text="CE" Value="CE" />
        <asp:ListItem Text="IT" Value="IT" />
    </asp:DropDownList>

      

    <asp:TextBox
        ID="txtYear"
        runat="server"
        placeholder="Enter Year">
    </asp:TextBox>

    <asp:Button
        ID="btnFilterYear"
        runat="server"
        Text="Filter by Year"
        OnClick="btnFilterYear_Click" />

    <br /><br />

    <asp:GridView
        ID="gvStudents"
        runat="server"
        AutoGenerateColumns="False"
        DataKeyNames="StudentID"
        AllowPaging="True"
        PageSize="5"
        OnPageIndexChanging="gvStudents_PageIndexChanging"
        OnRowEditing="gvStudents_RowEditing"
        OnRowCancelingEdit="gvStudents_RowCancelingEdit"
        OnRowUpdating="gvStudents_RowUpdating"
        OnRowDeleting="gvStudents_RowDeleting">

        <Columns>
            <asp:BoundField
                DataField="StudentID"
                HeaderText="Student ID"
                ReadOnly="True" />

            <asp:BoundField
                DataField="StudentName"
                HeaderText="Name" />

            <asp:BoundField
                DataField="RollNumber"
                HeaderText="Roll No" />

            <asp:BoundField
```

```
                DataField="Course"
                HeaderText="Course" />

            <asp:BoundField
                DataField="EnrollmentDate"
                HeaderText="Enrollment Date"
                DataFormatString="{0:yyyy-MM-dd}" />

            <asp:BoundField
                DataField="EmailID"
                HeaderText="Email" />

            <asp:BoundField
                DataField="MobileNumber"
                HeaderText="Mobile" />

            <asp:CommandField
                ShowEditButton="True"
                ShowDeleteButton="True" />
        </Columns>

    </asp:GridView>
  </div>
 </form>
</body>
</html>
```

AdminDashboard.aspx.cs

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
using System.Web.UI.WebControls;

namespace StudentManagementSystem
{
 public partial class AdminDashboard : System.Web.UI.Page
 {
  string connStr =
ConfigurationManager.ConnectionStrings["StudentDBConn"].ConnectionString;

  protected void Page_Load(object sender, EventArgs e)
  {
   if (!IsPostBack)
   {
    BindGrid();
   }
  }

  private void BindGrid(string course = "", string year = "")
```

```
{
    using (SqlConnection con = new SqlConnection(connStr))
    {
        con.Open();
        string query = "SELECT * FROM Students WHERE 1=1";
        if (!string.IsNullOrEmpty(course))
            query += " AND Course = @Course";
        if (!string.IsNullOrEmpty(year))
            query += " AND YEAR(EnrollmentDate) = @Year";

        using (SqlCommand cmd = new SqlCommand(query, con))
        {
            if (!string.IsNullOrEmpty(course))
                cmd.Parameters.AddWithValue("@Course", course);
            if (!string.IsNullOrEmpty(year))
                cmd.Parameters.AddWithValue("@Year", year);

            using (SqlDataReader dr = cmd.ExecuteReader())
            {
                DataTable dt = new DataTable();
                dt.Load(dr);
                gvStudents.DataSource = dt;
                gvStudents.DataBind();
            }
        }
    }
}

protected void ddlFilterCourse_SelectedIndexChanged(object sender, EventArgs e)
{
    BindGrid(ddlFilterCourse.SelectedValue,  txtYear.Text);
}

protected void btnFilterYear_Click(object sender, EventArgs e)
{
    BindGrid(ddlFilterCourse.SelectedValue,  txtYear.Text);
}

protected void gvStudents_PageIndexChanging(object sender, GridViewPageEventArgs e)
{
    gvStudents.PageIndex = e.NewPageIndex;
    BindGrid(ddlFilterCourse.SelectedValue, txtYear.Text);
}

protected void gvStudents_RowEditing(object sender, GridViewEditEventArgs e)
{
    gvStudents.EditIndex = e.NewEditIndex;
    BindGrid(ddlFilterCourse.SelectedValue, txtYear.Text);
}
```

```
    protected void gvStudents_RowCancelingEdit(object sender, GridViewCancelEditEventArgs
e)
    {
        gvStudents.EditIndex = -1;
        BindGrid(ddlFilterCourse.SelectedValue, txtYear.Text);
    }

    protected void gvStudents_RowUpdating(object sender, GridViewUpdateEventArgs e)
    {
        string studentID = gvStudents.DataKeys[e.RowIndex].Value.ToString();

        string name = ((TextBox)gvStudents.Rows[e.RowIndex].Cells[1].Controls[0]).Text;
        string roll = ((TextBox)gvStudents.Rows[e.RowIndex].Cells[2].Controls[0]).Text;
        string course = ((TextBox)gvStudents.Rows[e.RowIndex].Cells[3].Controls[0]).Text;
        string date = ((TextBox)gvStudents.Rows[e.RowIndex].Cells[4].Controls[0]).Text;
        string email = ((TextBox)gvStudents.Rows[e.RowIndex].Cells[5].Controls[0]).Text;
        string mobile = ((TextBox)gvStudents.Rows[e.RowIndex].Cells[6].Controls[0]).Text;

        using (SqlConnection con = new SqlConnection(connStr))
        {
            SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM Students WHERE
StudentID=@StudentID", con);
            da.SelectCommand.Parameters.AddWithValue("@StudentID",   studentID);

            DataSet ds = new DataSet();
            da.Fill(ds);

            if (ds.Tables[0].Rows.Count > 0)
            {
                ds.Tables[0].Rows[0]["StudentName"] = name;
                ds.Tables[0].Rows[0]["RollNumber"] = roll;
                ds.Tables[0].Rows[0]["Course"] = course;
                ds.Tables[0].Rows[0]["EnrollmentDate"] = date;
                ds.Tables[0].Rows[0]["EmailID"] = email;
                ds.Tables[0].Rows[0]["MobileNumber"] = mobile;

                SqlCommandBuilder cb = new SqlCommandBuilder(da);
                da.Update(ds);
            }
        }

        gvStudents.EditIndex = -1;
        BindGrid(ddlFilterCourse.SelectedValue, txtYear.Text);
    }

    protected void gvStudents_RowDeleting(object sender, GridViewDeleteEventArgs e)
    {
        string studentID = gvStudents.DataKeys[e.RowIndex].Value.ToString();
        using (SqlConnection con = new SqlConnection(connStr))
        {
```

```
            con.Open();
            using (SqlCommand cmd = new SqlCommand("DELETE FROM Students WHERE
StudentID=@StudentID", con))
            {
                cmd.Parameters.AddWithValue("@StudentID", studentID);
                cmd.ExecuteNonQuery();
            }
        }
        BindGrid(ddlFilterCourse.SelectedValue,  txtYear.Text);
    }
  }
}
```

Web.config :

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
 <connectionStrings>
  <add name="StudentDBConn"
      connectionString="Data Source=LAPTOP-BOJ0UBU5\SQLEXPRESS;Initial
Catalog=StudentDB;Integrated Security=True"
      providerName="System.Data.SqlClient"/>
 </connectionStrings>

 <system.web>
  <sessionState mode="InProc" timeout="20"></sessionState>
  <compilation debug="true" targetFramework="4.8" />
  <httpRuntime targetFramework="4.8" />
  <pages>
   <namespaces>
    <add namespace="System.Web.Optimization" />
   </namespaces>
   <controls>
    <add assembly="Microsoft.AspNet.Web.Optimization.WebForms"
        namespace="Microsoft.AspNet.Web.Optimization.WebForms"
        tagPrefix="webopt" />
   </controls>
  </pages>
 </system.web>

 <runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
   <dependentAssembly>
    <assemblyIdentity name="Antlr3.Runtime" publicKeyToken="eb42632606e9261f" />
    <bindingRedirect oldVersion="0.0.0.0-3.5.0.2" newVersion="3.5.0.2" />
   </dependentAssembly>
   <dependentAssembly>
    <assemblyIdentity name="Microsoft.Web.Infrastructure"
publicKeyToken="31bf3856ad364e35" />
    <bindingRedirect oldVersion="0.0.0.0-2.0.0.0" newVersion="2.0.0.0" />
```

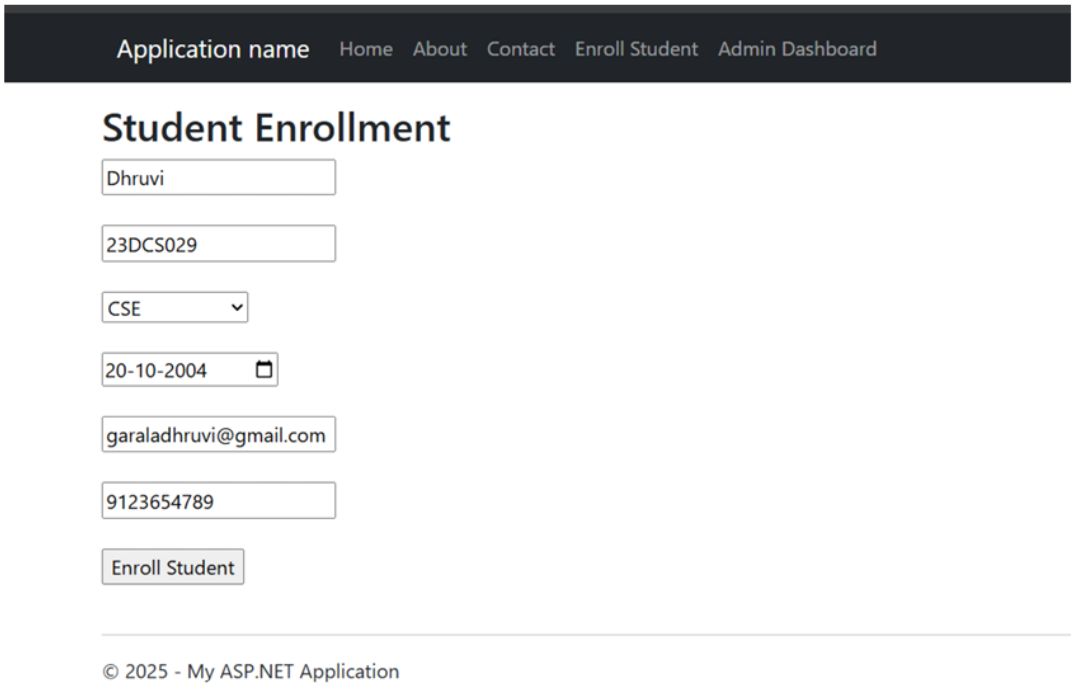
```
    </dependentAssembly>
    <dependentAssembly>
      <assemblyIdentity name="Newtonsoft.Json" publicKeyToken="30ad4fe6b2a6aeed" />
      <bindingRedirect oldVersion="0.0.0.0-13.0.0.0" newVersion="13.0.0.0" />
    </dependentAssembly>
    <dependentAssembly>
      <assemblyIdentity name="WebGrease" publicKeyToken="31bf3856ad364e35" />
      <bindingRedirect oldVersion="0.0.0.0-1.6.5135.21930" newVersion="1.6.5135.21930" />
    </dependentAssembly>
  </assemblyBinding>
 </runtime>

 <system.codedom>
  <compilers>
    <compiler language="c#;cs;csharp" extension=".cs"
          type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.CSharpCodeProvider,
          Microsoft.CodeDom.Providers.DotNetCompilerPlatform, Version=2.0.1.0,
Culture=neutral,
          PublicKeyToken=31bf3856ad364e35" warningLevel="4"
          compilerOptions="/langversion:default /nowarn:1659;1699;1701" />
    <compiler language="vb;vbs;visualbasic;vbscript" extension=".vb"
          type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.VBCodeProvider,
          Microsoft.CodeDom.Providers.DotNetCompilerPlatform, Version=2.0.1.0,
Culture=neutral,
          PublicKeyToken=31bf3856ad364e35" warningLevel="4"
          compilerOptions="/langversion:default /nowarn:41008
/define:_MYTYPE=\&quot;Web\&quot; /optionInfer+" />
  </compilers>
 </system.codedom>
</configuration>
```

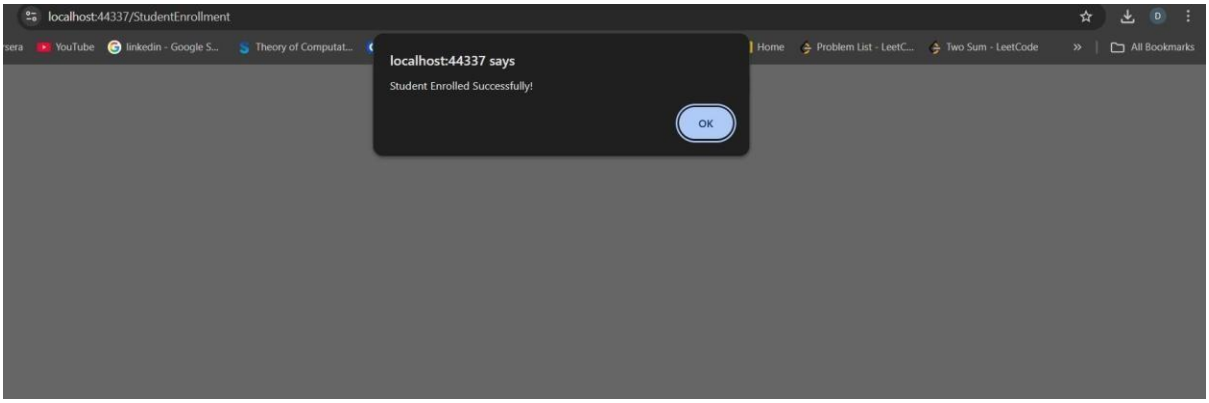
Site.Master:

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

<!DOCTYPE html>
<html lang="en">
<head runat="server">
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title><%: Page.Title %> - Student Management</title>

  <asp:PlaceHolder runat="server">
    <%: Scripts.Render("~/bundles/modernizr") %>
  </asp:PlaceHolder>

  <webopt:bundlereference runat="server" path="~/Content/css" />
  <link href="~/favicon.ico" rel="shortcut icon" type="image/x-icon" />
</head>
<body>
```

```
    <form runat="server">
        <asp:ScriptManager runat="server">
            <Scripts>
                <asp:ScriptReference Name="MsAjaxBundle" />
                <asp:ScriptReference Name="jquery" />
                <asp:ScriptReference Name="WebFormsBundle" />
            </Scripts>
        </asp:ScriptManager>

        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-dark bg-dark">
            <div class="container">
                <a class="navbar-brand" runat="server" href="~/">Student Management</a>
                <button type="button" class="navbar-toggler" data-bs-toggle="collapse"
                        data-bs-target=".navbar-collapse" aria-controls="navbarSupportedContent"
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="collapse navbar-collapse d-sm-inline-flex justify-content-between">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item"><a class="nav-link" runat="server"
href="~/">Home</a></li>
                        <li class="nav-item"><a class="nav-link" runat="server"
href="~/About">About</a></li>
                        <li class="nav-item"><a class="nav-link" runat="server"
href="~/Contact">Contact</a></li>
                        <li class="nav-item"><a class="nav-link" href="StudentEnrollment.aspx">Enroll
Student</a></li>
                        <li class="nav-item"><a class="nav-link" href="AdminDashboard.aspx">Admin
Dashboard</a></li>
                    </ul>
                </div>
            </div>
        </nav>

        <div class="container body-content">
            <asp:ContentPlaceHolder ID="MainContent" runat="server"></asp:ContentPlaceHolder>
            <hr />
            <footer>
                <p>&copy; <%: DateTime.Now.Year %> - Student Management</p>
            </footer>
        </div>
    </form>

    <asp:PlaceHolder runat="server">
        <%: Scripts.Render("~/Scripts/bootstrap.js") %>
    </asp:PlaceHolder>
</body>
</html>
```

OUTPUT:





Admin Dashboard - Student Management

All Courses | Enter Year | Filter by Year

Student ID	Name	Roll No	Course	Enrollment Date	Email	Mobile	
DH20250818161631	dhruvi	23dcs029	CSE	2004-10-20	garaladhruvi@gmail.com	9123654789	Edit Delete
IS20250818193125	Isha	23CE114	CE	2005-06-21	isha89@gmail.com	9123654743	Edit Delete
KR20250818161826	krishna	25DDU036	IT	2006-08-20	garalakrishna@gmail.com	9123654799	Edit Delete
KU20250818193423	Kunj	23DDUCE	CE	2001-06-14	kunjpatil@gmail.com	9123654743	Edit Delete
PR20250818193222	Pranjal	23CE008	IT	2025-06-09	ppranjal@gmail.com	9123654799	Edit Delete

CSE | Enter Year | Filter by Year

Student ID	Name	Roll No	Course	Enrollment Date	Email	Mobile	
DH20250818161631	dhruvi	23dcs029	CSE	2004-10-20	garaladhruvi@gmail.com	9123654789	Edit Delete

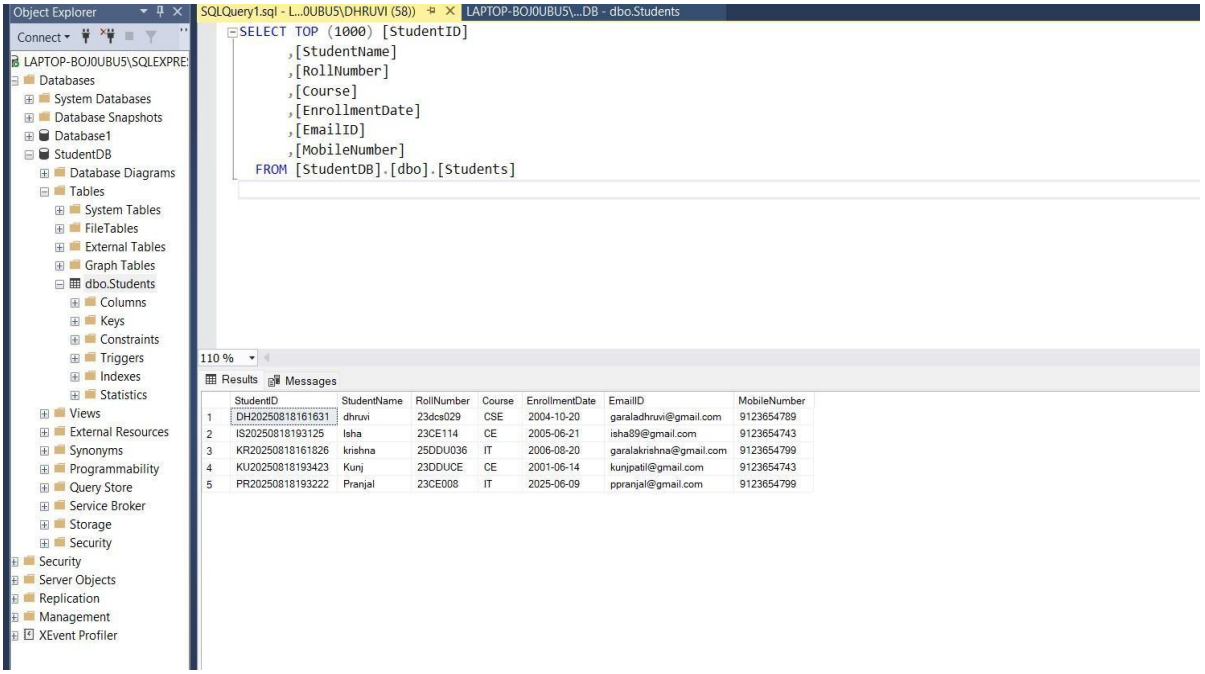
Admin Dashboard - Student Management

CE | Enter Year | Filter by Year

Student ID	Name	Roll No	Course	Enrollment Date	Email	Mobile	
IS20250818193125	Isha	23CE114	CE	2005-06-21	isha89@gmail.com	9123654743	Edit Delete
KU20250818193423	Kunj	23DDUCE	CE	2001-06-14	kunjpatil@gmail.com	9123654743	Edit Delete

All Courses | Enter Year | Filter by Year

Student ID	Name	Roll No	Course	Enrollment Date	Email	Mobile	
DH20250818161631	dhruvi	23dcs029	CSE	20-10-2004 00:00:00	garaladhruvi@gmail.com	9123654789	Update Cancel



Conclusion:

The Student Management System manages enrollment with validation, session handling, and unique IDs. Using ADO.NET's DataSet and DataReader, it enables efficient data operations with GridView for full CRUD functionality, ensuring modular, structured, and interactive student data management.

PRACTICAL: 4

AIM: The Academic Services Department of a university is in need of a secure and scalable web-based solution to manage student enrollments across courses. As a developer in the university's internal development team, you are tasked with building this system using ASP.NET MVC architecture. This system will allow academic administrators to register students, assign them to courses, manage enrollment updates, and maintain a clean view of course-wise student listings.

Functional Requirements:

Admin Tasks:

1. Add new students with validated details.
2. Create and manage course offerings (course name, code, credits).
3. Assign one or more courses to a student during enrollment.
4. View all student records along with enrolled courses.
5. Edit or delete a student's record.
6. Filter students by course, name, or enrollment year.
7. Navigate through the application using well-defined routes.

MVC Implementation Strategy:

Model Layer:

- Define clean domain models: Student, Course, Enrollment
- Use Data Annotations for constraints (e.g., Required, Range, Email format)
- Maintain in-memory collections to simulate a repository

Controller Layer:

- Create StudentController and CourseController
- Implement strongly typed action methods for:
- Index, Create, Edit, Delete, Details
- Include custom actions like:
- AssignCourses(int studentId)
- GetStudentsByCourse(string courseCode)

View Layer (Razor):

- Use strongly typed Razor views for all operations
- Build interactive forms with Html.BeginForm, @Html.EditorFor, etc.

- Display error messages using @Html.ValidationMessageFor
- Use foreach to list student-course enrollments in tabular format
- Apply conditional UI logic (e.g., "Show Courses if Assigned") Routing & Navigation:
- Define custom routes such as:

  /students/enrolled

  /student/edit/102

  /course/assign/CS101
- Utilize RouteConfig.cs for defining logical URL structures
- Use attribute routing for specific controller methods

CODE:

STUDENT CONTROLLER:

```
using System.Linq;
using System.Web.Mvc;
using UniversityEnrollmentSystem.Models;
namespace UniversityEnrollmentSystem.Controllers {
   public class StudentController : Controller {
      public ActionResult Index(){
         return View(Repository.Students);
      }
      public ActionResult Details(int id){
         var student = Repository.GetStudentById(id);
         if (student == null) return HttpNotFound();
         return View(student);
      }
      public ActionResult Create() {
         return View();
      }
```

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create(Student student) {
    if (ModelState.IsValid) {
        student.Id = Repository.Students.Count + 1;
        Repository.Students.Add(student);
        return RedirectToAction("Index");
    }
    return View(student);
}
public ActionResult Edit(int id) {
    var student = Repository.GetStudentById(id);
    if (student == null) return HttpNotFound();
    return View(student);
}
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit(Student updatedStudent) {
    if (ModelState.IsValid) {
        var student = Repository.GetStudentById(updatedStudent.Id);
        if (student == null) return HttpNotFound();
        student.Name = updatedStudent.Name;
        student.Email = updatedStudent.Email;
        student.EnrollmentYear = updatedStudent.EnrollmentYear;
        return RedirectToAction("Index");
    }
    return View(updatedStudent);
}
```

```
        public ActionResult Delete(int id) {
            var student = Repository.GetStudentById(id);
            if (student == null) return HttpNotFound();
            return View(student);
        }
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public ActionResult DeleteConfirmed(int id) {
            var student = Repository.GetStudentById(id);
            if (student == null) {
                return HttpNotFound();
            }
            Repository.Students.Remove(student);
            return RedirectToAction("Index");
        }
    }
}
```

COURSE CONTROLLER:

```
using System.Linq;
using System.Web.Mvc;
using UniversityEnrollmentSystem.Models;
namespace UniversityEnrollmentSystem.Controllers {
    public class CourseController : Controller {
        public ActionResult Index() {
            return View(Repository.Courses);
        }
        public ActionResult Details(string code) {
            var course = Repository.GetCourseByCode(code);
```

```
        if (course == null) return HttpNotFound();
        return View(course);
    }
    public ActionResult Create() {
        return View();
    }
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Create(Course course) {
        if (ModelState.IsValid) {
            Repository.Courses.Add(course);
            return RedirectToAction("Index");
        }
        return View(course);
    }
    public ActionResult Edit(string code) {
        var course = Repository.GetCourseByCode(code);
        if (course == null) return HttpNotFound();
        return View(course);
    }
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Edit(Course updatedCourse) {
        if (ModelState.IsValid) {
            var course = Repository.GetCourseByCode(updatedCourse.Code);
            if (course == null) return HttpNotFound();
            course.Name = updatedCourse.Name;
            course.Credits = updatedCourse.Credits;
            return RedirectToAction("Index");
```

VIEWS UNDER STUDENT CONTOLLER:

Create.cshtml:

```
@model UniversityEnrollmentSystem.Models.Student

@{

ViewBag.Title = "Add Student";

Layout = "~/Views/Shared/_Layout.cshtml"; // Optional: ensure consistent styling

}

<h2 class="mb-4">Add Student</h2>

<div class="card shadow-sm p-4 mb-5 bg-white rounded"> @using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-group mb-3">

@Html.LabelFor(m => m.Name, htmlAttributes: new { @class = "form-label fw-bold" })
@Html.TextBoxFor(m => m.Name, new { @class = "form-control", placeholder =

"Enter full name" })

@Html.ValidationMessageFor(m => m.Name, "", new { @class = "text-danger" })

</div>

<div class="form-group mb-3">

@Html.LabelFor(m => m.Email, htmlAttributes: new { @class = "form-label fw-bold" })
@Html.TextBoxFor(m => m.Email, new { @class = "form-control", placeholder =

"Enter email address" })

@Html.ValidationMessageFor(m => m.Email, "", new { @class = "text-danger" })

</div>

<div class="form-group mb-4">

@Html.LabelFor(m => m.EnrollmentYear, htmlAttributes: new { @class = "form-label fw-
bold" })

@Html.TextBoxFor(m => m.EnrollmentYear, new { @class = "form-control", placeholder =
"e.g. 2023" })

@Html.ValidationMessageFor(m => m.EnrollmentYear, "", new { @class = "text- danger" })

</div>
```

```
<button type="submit" class="btn btn-primary w-100">Save Student</button>
}
</div>
<div class="text-center">
@Html.ActionLink("← Back to List", "Index", null, new { @class = "btn btn-outline-secondary" })
</div>
```

Delete.cshtml:

```
@model UniversityEnrollmentSystem.Models.Student
@{
ViewBag.Title = "Delete Student";
}
<h2>Delete Student</h2>
<p>Are you sure you want to delete this student?</p>
<p><strong>Name:</strong> @Model.Name</p>
<p><strong>Email:</strong> @Model.Email</p>
<p><strong>Enrollment Year:</strong> @Model.EnrollmentYear</p>
@using (Html.BeginForm("Delete", "Student", new { id = Model.Id }, FormMethod.Post))
{
@Html.AntiForgeryToken()
<button type="submit" class="btn btn-danger">Delete</button>
}
<p>
@Html.ActionLink("Back to List", "Index")
</p>
```

Details.cshtml:

```
@model UniversityEnrollmentSystem.Models.Student
@{
ViewBag.Title = "Student Details";
}
<h2>Student Details</h2>
<p><strong>Name:</strong> @Model.Name</p>
<p><strong>Email:</strong> @Model.Email</p>
<p><strong>Enrollment Year:</strong> @Model.EnrollmentYear</p>
<p>
@Html.ActionLink("Back to List", "Index")
</p>
```

Edit.cshtml:

```
@model UniversityEnrollmentSystem.Models.Student
@{
ViewBag.Title = "Edit Student";
Layout = "~/Views/Shared/_Layout.cshtml"; // Optional: ensures consistent styling
}
<h2 class="mb-4">Edit Student</h2>
<div class="card shadow-sm p-4 mb-5 bg-white rounded"> @using (Html.BeginForm())
{
@Html.AntiForgeryToken() @Html.HiddenFor(m => m.Id)
<div class="form-group mb-3">
@Html.LabelFor(m => m.Name, htmlAttributes: new { @class = "form-label fw-bold" })
@Html.TextBoxFor(m => m.Name, new { @class = "form-control", placeholder =
"Enter full name" })
@Html.ValidationMessageFor(m => m.Name, "", new { @class = "text-danger" })
</div>
```

```
<div class="form-group mb-3">

@Html.LabelFor(m => m.Email, htmlAttributes: new { @class = "form-label fw-bold" })
@Html.TextBoxFor(m => m.Email, new { @class = "form-control", placeholder =

"Enter email address" })

@Html.ValidationMessageFor(m => m.Email, "", new { @class = "text-danger" })

</div>



<div class="form-group mb-4">

@Html.LabelFor(m => m.EnrollmentYear, htmlAttributes: new { @class = "form-label fw-
bold" })


@Html.TextBoxFor(m => m.EnrollmentYear, new { @class = "form-control", placeholder =
"e.g. 2023" })

@Html.ValidationMessageFor(m => m.EnrollmentYear, "", new { @class = "text- danger" })

</div>

<button type="submit" class="btn btn-success w-100">Update Student</button>

}

</div>

<div class="text-center">

@Html.ActionLink("← Back to List", "Index", null, new { @class = "btn btn-outline-
secondary" })

</div>
```

Index.cshtml:

```
@model IEnumerable<UniversityEnrollmentSystem.Models.Student>

@{

ViewBag.Title = "Students";

}

<h2>Students</h2>

<p>
```

```
@Html.ActionLink("Add New Student", "Create")
</p>
<table class="table table-bordered">
<tr>
<th>Name</th>
<th>Email</th>
<th>Enrollment Year</th>
<th>Actions</th>
</tr>
@foreach (var student in Model)
{
<tr>
<td>@student.Name</td>
<td>@student.Email</td>
<td>@student.EnrollmentYear</td>
<td>
@Html.ActionLink("Details", "Details", new { id = student.Id }) | @Html.ActionLink("Edit", "Edit", new { id = student.Id }) | @Html.ActionLink("Delete", "Delete", new { id = student.Id })
</td>
</tr>
}
</table>
```

COURSE CONTOLLER:

```
using System.Linq; using System.Web.Mvc;
using UniversityEnrollmentSystem.Models;
namespace UniversityEnrollmentSystem.Controllers
{
public class CourseController : Controller  {
```

```
public ActionResult Index() {
return View(Repository.Courses);
}
public ActionResult Details(string code) {
var course = Repository.GetCourseByCode(code); if (course == null) return HttpNotFound();
return View(course);
}
public ActionResult Create() {
return View();
}
[HttpPost]

[ValidateAntiForgeryToken]
public ActionResult Create(Course course) {
if (ModelState.IsValid) {
Repository.Courses.Add(course); return RedirectToAction("Index");
}
return View(course);
}
public ActionResult Edit(string code) {
var course = Repository.GetCourseByCode(code); if (course == null) return HttpNotFound();
return View(course);
}
[HttpPost] [ValidateAntiForgeryToken]
public ActionResult Edit(Course updatedCourse) {
if (ModelState.IsValid) {
var course = Repository.GetCourseByCode(updatedCourse.Code); if (course == null) return
HttpNotFound();
course.Name = updatedCourse.Name;
```

```
course.Credits = updatedCourse.Credits;
return RedirectToAction("Index");
} return View(updatedCourse);
}
public ActionResult Delete(string code) {
var course = Repository.GetCourseByCode(code); if (course == null) return HttpNotFound();
return View(course);
}
[HttpPost, ActionName("Delete")] [ValidateAntiForgeryToken]
public ActionResult DeleteConfirmed(string code) {
var course = Repository.GetCourseByCode(code); if (course == null) {
return HttpNotFound(); // Proper null check
}
Repository.Courses.Remove(course); return RedirectToAction("Index");
} } }
```

// MODELS :

// Course.cs

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
namespace UniversityEnrollmentSystem.Models {
   public class Course {
      public string Code { get; set; }
      [Required]
      public string Name { get; set; }
      [Range(1, 10, ErrorMessage = "Credits must be between 1 and 10")]
      public int Credits { get; set; }
      public List<Enrollment> Enrollments { get; set; } = new List<Enrollment>();
   } }
```

// Enrollment.cs

```
namespace UniversityEnrollmentSystem.Models {
   public class Enrollment {
      public int Id { get; set; }
      public int StudentId { get; set; }
      public string CourseCode { get; set; }
      public Student Student { get; set; }
      public Course Course { get; set; }
   }  }
```

// Repository.cs

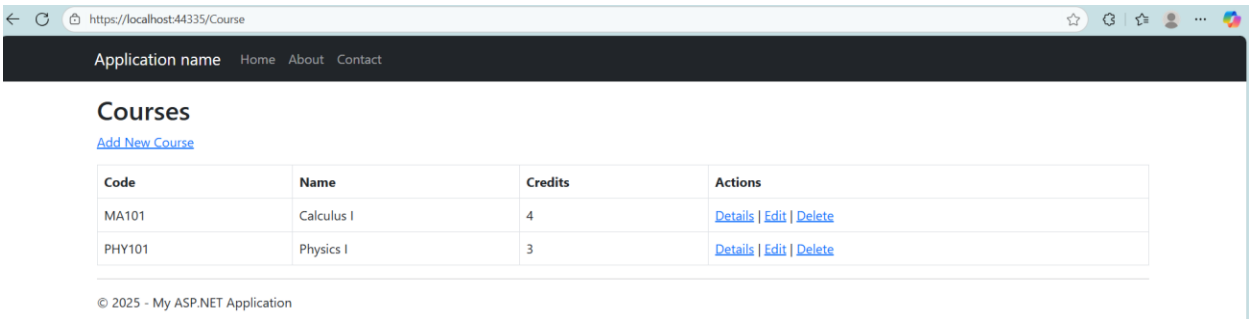
```
using System.Collections.Generic;
using System.Linq;
namespace UniversityEnrollmentSystem.Models {
   public static class Repository {
      public static List<Student> Students = new List<Student>();
      public static List<Course> Courses = new List<Course>();
      public static List<Enrollment> Enrollments = new List<Enrollment>();
      static Repository() {
         Courses.Add(new Course { Code = "CS101", Name = "Introduction to CS", Credits =
3 });
         Courses.Add(new Course { Code = "MA101", Name = "Calculus I", Credits = 4 });
         Courses.Add(new Course { Code = "PHY101", Name = "Physics I", Credits = 3 });
         Students.Add(new Student { Id = 1, Name = "Amit Sharma", Email =
"amit@example.com", EnrollmentYear = 2023 });
         Students.Add(new Student { Id = 2, Name = "Priya Patel", Email =
"priya@example.com", EnrollmentYear = 2022 });
         Enrollments.Add(new Enrollment { Id = 1, StudentId = 1, CourseCode = "CS101" });
         Enrollments.Add(new Enrollment { Id = 2, StudentId = 1, CourseCode = "MA101" });
         Enrollments.Add(new Enrollment { Id = 3, StudentId = 2, CourseCode = "PHY101"
});
```

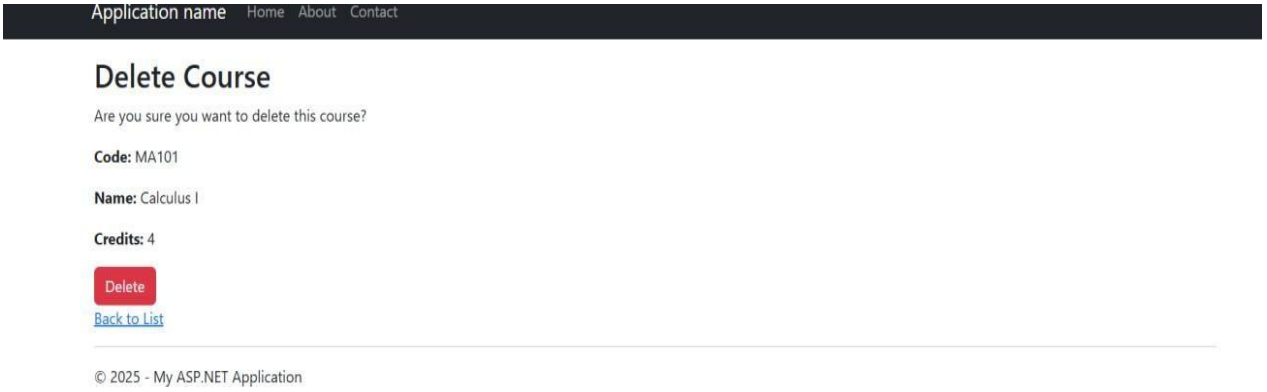
```
  }
      public static Student GetStudentById(int id) => Students.FirstOrDefault(s => s.Id == id);
      public static Course GetCourseByCode(string code) => Courses.FirstOrDefault(c => c.Code == code);
      public static List<Student> GetStudentsByCourse(string courseCode) =>
         Students.Where(s => Enrollments.Any(e => e.StudentId == s.Id && e.CourseCode == courseCode)).ToList();
  } }
```

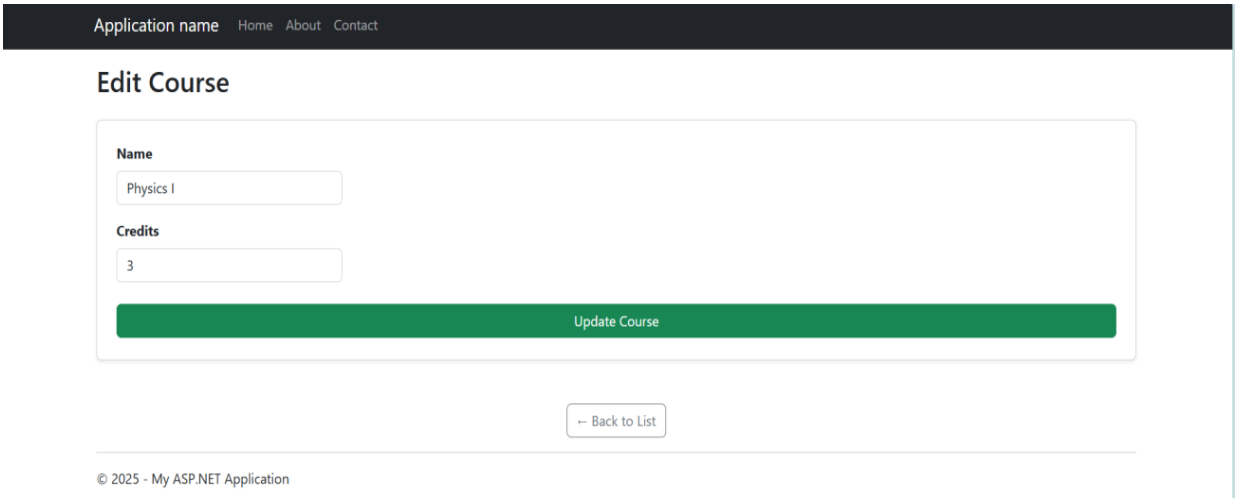
// Student.cs

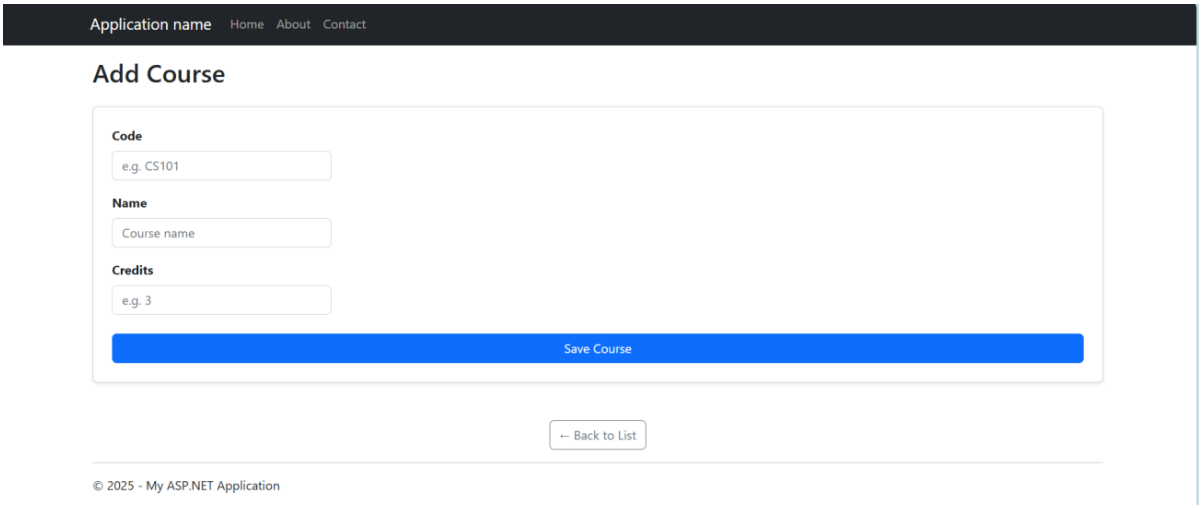
```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
namespace UniversityEnrollmentSystem.Models {
  public class Student {
      public int Id { get; set; }
      [Required(ErrorMessage = "Name is required")]
      public string Name { get; set; }
      [Required, EmailAddress(ErrorMessage = "Enter a valid email")]
      public string Email { get; set; }
      [Range(2000, 2100, ErrorMessage = "Enter valid enrollment year")]
      public int EnrollmentYear { get; set; }
      public List<Enrollment> Enrollments { get; set; } = new List<Enrollment>();
  } }
```

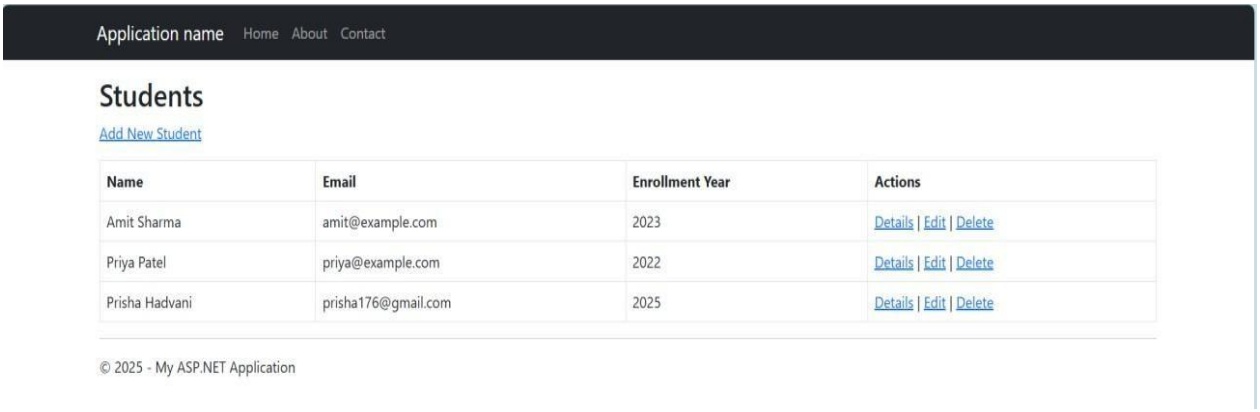
OUTPUT:

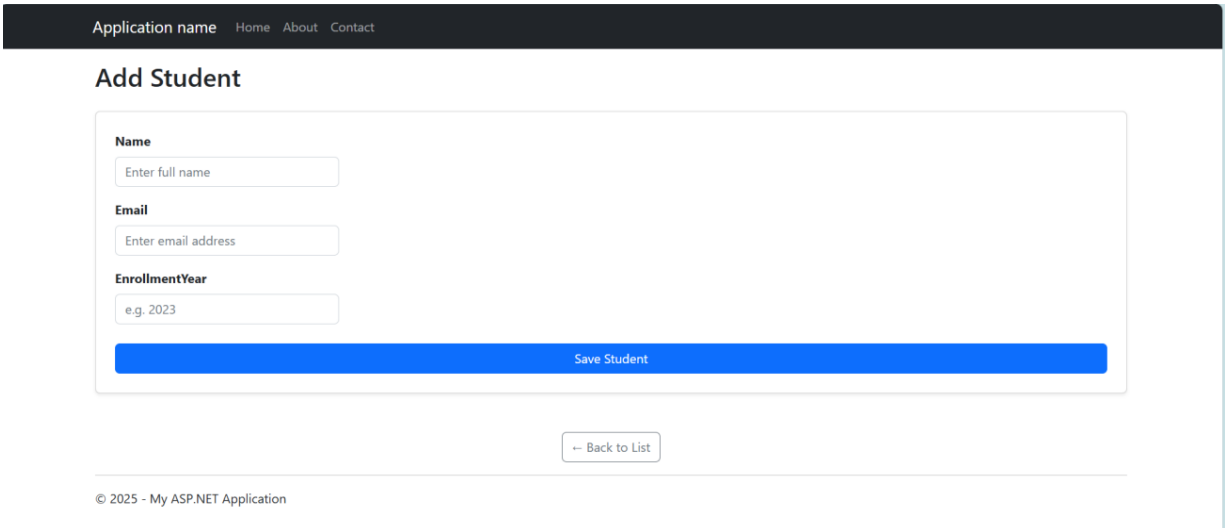

https://localhost:44335/Course
Application name
Home
About
Contact
Courses
Add New Course
Code
Name
Credits
Actions
MA101
Calculus I
4
Details | Edit | Delete
PHY101
Physics I
3
Details | Edit | Delete
© 2025 - My ASP.NET Application

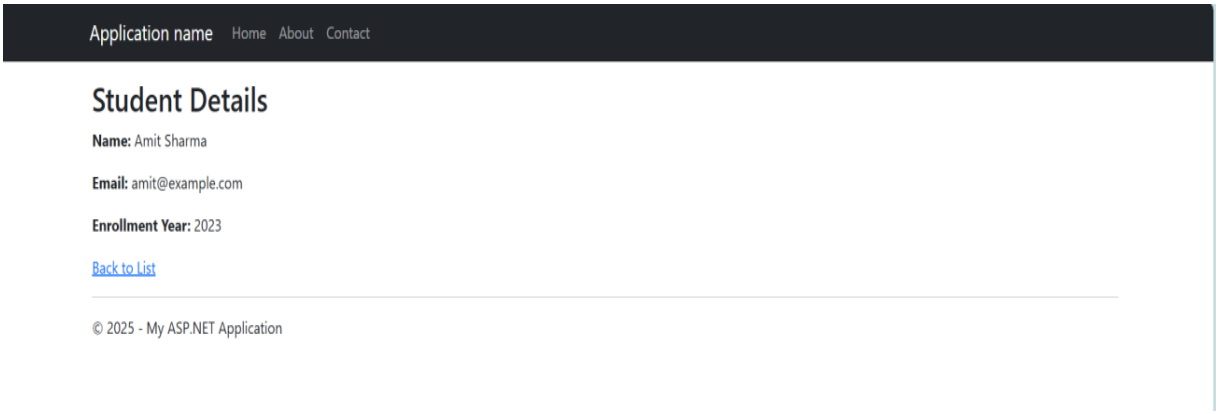

Application name
Home
About
Contact
Delete Course
Are you sure you want to delete this course?
Code: MA101
Name: Calculus I
Credits: 4
Delete
Back to List
© 2025 - My ASP.NET Application


Application name
Home
About
Contact
Edit Course
Name
Physics I
Credits
3
Update Course
← Back to List
© 2025 - My ASP.NET Application


Application name
Home
About
Contact
Add Course
Code
e.g. CS101
Name
Course name
Credits
e.g. 3
Save Course
← Back to List
© 2025 - My ASP.NET Application


Application name
Home
About
Contact
Students
Add New Student
Name
Email
Enrollment Year
Actions
Amit Sharma
amit@example.com
2023
Details | Edit | Delete
Priya Patel
priya@example.com
2022
Details | Edit | Delete
Prisha Hadvani
prisha176@gmail.com
2025
Details | Edit | Delete
© 2025 - My ASP.NET Application


Application name
Home
About
Contact
Add Student
Name
Enter full name
Email
Enter email address
EnrollmentYear
e.g. 2023
Save Student
← Back to List
© 2025 - My ASP.NET Application



Conclusion

The Student Enrollment Management System was developed using ASP.NET MVC, allowing efficient management of students, courses, and enrollments. It provides features to add, edit, delete, and view records with a clean, user-friendly interface, ensuring better organization and maintainability.

PRACTICAL: 5

AIM: Student Record Manager – Display Student Data using MVC Architecture

Scenario

You are working as a software developer in the university's IT department. The faculty has requested a web application that allows them to view student records such as ID, name, department, and CGPA. As part of the initial prototype, your task is to create a simple ASP.NET Core MVC application that will display a list of students using hardcoded data (i.e., not connected to a database).

This prototype should follow the MVC design pattern:

- Model: Define a Student class that encapsulates student attributes including ID, Name, Department, CGPA, and Email.
- Controller: Create a controller (StudentController) that initializes and supplies a hardcoded list of Student objects to the view.
- View: Implement a Razor view to display student records in a structured HTML table with columns for ID, Name, Department, CGPA, and Email.

Task 1: Create a New ASP.NET Core MVC Project

Task 2: Create a Student model class with properties: StudentId, Name, Department, CGPA.

Task 3: Add a StudentController and define an Index() action method that prepares and passes a list of hardcoded students to the view.

Task 4: Design a Student/Index.cshtml view to display the list using an HTML table format, properly binding data from the model.

Task 5: Configure the Startup.cs or Program.cs and appsettings.json (if required) to ensure the default route maps to StudentController -> Index.

Task 6: Build and run the application to test that the student records are displayed correctly.

Task 7:

- Add a new property Email to the Student model.
- Update the controller to include email addresses in the hardcoded list.
- Modify the view to include a new column for displaying the email address.

CODE:

Student.cs

```
namespace StudentRecordManager.Models {
   public class Student
   {
      public int StudentId { get; set; }
      public string Name { get; set; } = string.Empty;
      public string Department { get; set; } = string.Empty;
      public string Email { get; set; } = string.Empty;
      public double CGPA { get; set; }
   }
}
```

StudentController.cs

```
using Microsoft.AspNetCore.Mvc;
using StudentRecordManager.Models;
namespace StudentRecordManager.Controllers {
   public class StudentController : Controller  {
      public IActionResult Index() {
         var students = new List<Student>     {
            new Student { StudentId = 101, Name = "Aeni Patel",    Department = "CSE",
CGPA = 9.1, Email = "aenipatel@gmail.com" },
            new Student { StudentId = 102, Name = "Dishva Vasoya", Department = "ECE",
CGPA = 8.6, Email = "dishvavasoya@gmail.com" },
            new Student { StudentId = 103, Name = "Nishi Naik",    Department = "IT",
CGPA = 8.9, Email = "nishinaik@gmail.com" },
            new Student { StudentId = 104, Name = "Nency Patel",  Department = "ME",
CGPA = 8.2, Email = "nencypatel@gmail.com" },
            new Student { StudentId = 105, Name = "Riya Patel",    Department = "CSE",
CGPA = 9.3, Email = "riyapatel@gmail.com" },
```

```
            new Student { StudentId = 106, Name = "Vishwa Vasoya", Department = "CSE",
CGPA = 9.5, Email = "vishvavasoya@gmail.com" },

            new Student { StudentId = 107, Name = "Kreshi Goti",  Department = "ECE",
CGPA = 8.0, Email = "kreshigoti@gmail.com" },

            new Student { StudentId = 108, Name = "Disha Patel",  Department = "IT",
CGPA = 7.9, Email = "dishapatel@gmail.com" },

            new Student { StudentId = 109, Name = "Astha Patel",  Department = "ME",
CGPA = 8.7, Email = "asthapatel@gmail.com" },

            new Student { StudentId = 110, Name = "Krisha Patel",  Department = "CSE",
CGPA = 9.0, Email = "krishapatel@gmail.com" }

        };

        return View(students);

    }

  }

}
```

Index.cshtml

```
@model IEnumerable<StudentRecordManager.Models.Student>

@{

  ViewData["Title"] = "Students";

}

<h1 class="mt-3 mb-3">Student Records</h1>

<table class="table table-bordered table-striped">

  <thead>

    <tr>

      <th>Student ID</th>

      <th>Name</th>

      <th>Department</th>

      <th>CGPA</th>

      <th>Email</th> @* NEW *@

    </tr>
```

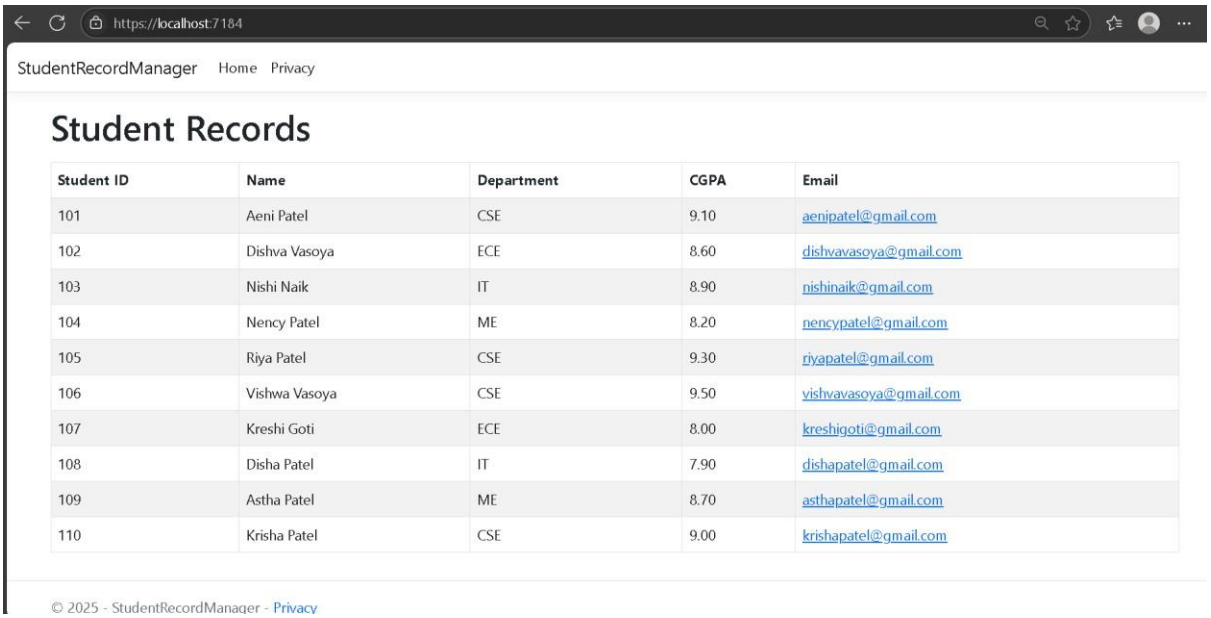
```
    </thead>
    <tbody>
    @foreach (var s in Model)  {
        <tr>
            <td>@s.StudentId</td>
            <td>@s.Name</td>
            <td>@s.Department</td>
            <td>@s.CGPA.ToString("0.00")</td>
            <td>
                <a href="mailto:@s.Email">@s.Email</a>
            </td>
        </tr>
    }
    </tbody>
</table>
```

Program.cs:

```
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllersWithViews();
var app = builder.Build();
if (!app.Environment.IsDevelopment()) {
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts(); // Enforces HTTPS for 30 days (default)
}
app.UseHttpsRedirection();
```

```
app.UseStaticFiles();
app.UseRouting();
app.UseAuthorization();
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Student}/{action=Index}/{id?}");
app.Run();
```

OUTPUT:



Conclusion:

In this practical, we built a simple Student Record Management System using ASP.NET Core MVC. We created a Student model, a controller to manage records, and a Razor view to display student details in a table. The system neatly shows information like Student ID, Name, Department, CGPA, and Email. This helped us understand how models, controllers, and views work together in MVC architecture to create dynamic web applications.

Practical-6

ASP.NET Core MVC – CRUD Employee Information – CRUD Operations Without Database

Scenario

You are working as a software developer in the Human Resources (HR) department of a corporate company. The HR team has requested a simple internal web application to manage employee records during the planning phase. Since this is a prototype, the data will be stored in-memory, not in a database. Your task is to build a basic ASP.NET Core MVC application to perform CRUD (Create, Read, Update, Delete) operations on employee information using a simple List in memory.

The application must follow the MVC design pattern:

- The Model will define the structure of the Employee.
- The Controller will manage the in-memory list and logic for CRUD operations.
- The Views will render user interfaces to interact with the employee data.

Tasks

Task 1: Create a New ASP.NET Core MVC Project

Task 2: Define the Employee Model

Create an Employee class inside the Models folder with the following

properties:

- Id (int)
- Name (string)
- Department (string)
- Designation (string)
- Email (string)
- Salary (decimal)

Task 3: Create the EmployeeController

Add a new EmployeeController that manages an in-memory List<Employee>, declared as a static list or via dependency injection. Implement logic to manipulate this list in memory (without database).

Task 4: Implement CRUD Operations in the Controller

Implement all necessary action methods in the controller:

- Index() – Display the list of employees.
- Details(int id) – Show detailed information of a specific employee.
- Create() – Show form and handle form submission to add a new employee.
- Edit(int id) – Show form and handle update logic.
- Delete(int id) – Confirm and remove an employee from the list.

Task 5: Create Razor Views for All Actions

Design corresponding Razor views using HTML helpers (@Html.EditorFor, @Html.ValidationMessageFor, etc.) for:

- Listing all employees
- Creating a new employee
- Editing existing employee details
- Viewing employee details
- Deleting an employee

Task 6: Implement Form Validation

Use Data Annotations in the Employee model to apply validation rules such as:

- [Required] for mandatory fields
- [EmailAddress] for email validation
- [Range] for salary limits
- Display validation messages in views using built-in Razor syntax.

Task 7: Configure Default Route

Update the routing configuration in Program.cs to ensure the application launches with EmployeeController and Index action.

Task 8: Build and Run the Application

Compile and run the application. Validate the correctness of all CRUD operations in the browser using the in-memory list.

Task 9: Documentation

Capture and submit screenshots of each operation:

- Create
- Read (Index and Details)
- Update
- Delete

Code:

Models/Employee.cs

```
using System.ComponentModel.DataAnnotations;

namespace EmployeeCRUD.Models {

   public class Employee    {

      public int Id { get; set; }

      [Required(ErrorMessage = "Name is required")]
```

```
        public string Name { get; set; }
        [Required(ErrorMessage = "Department is required")]
        public string Department { get; set; }
        [Required(ErrorMessage = "Designation is required")]
        public string Designation { get; set; }
        [Required(ErrorMessage = "Email is required")]
        [EmailAddress(ErrorMessage = "Invalid Email Address")]
        public string Email { get; set; }
        [Required(ErrorMessage = "Salary is required")]
        [Range(1000, 1000000, ErrorMessage = "Salary must be between 1000 and 1000000")]
        public decimal Salary { get; set; }
    }
}
```

Controllers /EmployeeController.cs

```
using Microsoft.AspNetCore.Mvc;
using EmployeeCRUD.Models;
using System.Collections.Generic;
using System.Linq;
namespace EmployeeCRUD.Controllers {
    public class EmployeeController : Controller     {
        // In-memory list to store employees
        private static List<Employee> employees = new List<Employee>();

        // GET: Employee
        public IActionResult Index()  {
            return View(employees);
        }
```

```
// GET: Employee/Details/5
public IActionResult Details(int id)      {
   var emp = employees.FirstOrDefault(e => e.Id == id);
   if (emp == null)
      return NotFound();
   return View(emp);
}

// GET: Employee/Create
public IActionResult Create()   {
   return View();
}

// POST: Employee/Create
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Create(Employee employee)    {
   if (ModelState.IsValid)
   {
      // Generate Id
      employee.Id = employees.Count > 0 ? employees.Max(e => e.Id) + 1 : 1;

      // Add employee to list
      employees.Add(employee);
      return RedirectToAction(nameof(Index));
   }  return View(employee);
}

// GET: Employee/Edit/5
```

```
public IActionResult Edit(int id)
{
   var emp = employees.FirstOrDefault(e => e.Id == id);
   if (emp == null)
      return NotFound();
   return View(emp);
}

// POST: Employee/Edit/5
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Edit(Employee employee)  {
   if (ModelState.IsValid)
   {
      var emp = employees.FirstOrDefault(e => e.Id == employee.Id);
      if (emp == null)
         return NotFound();

      // Update employee details
      emp.Name = employee.Name;
      emp.Department = employee.Department;
      emp.Designation = employee.Designation;
      emp.Email = employee.Email;
      emp.Salary = employee.Salary;
      return RedirectToAction(nameof(Index));
   } return View(employee);
}

// GET: Employee/Delete/5
```

```
        public IActionResult Delete(int id)    {
            var emp = employees.FirstOrDefault(e => e.Id == id);
            if (emp == null)
                return NotFound();
            return View(emp);
        }

        // POST: Employee/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public IActionResult DeleteConfirmed(int id)    {
            var emp = employees.FirstOrDefault(e => e.Id == id);
            if (emp != null)    {
                employees.Remove(emp);
            }     return RedirectToAction(nameof(Index));
        }
    }
}
```

Views/Employess/Index.cshtml

```
@model IEnumerable<EmployeeCRUD.Models.Employee>
<h2>Employees</h2>
<a asp-action="Create">Create New Employee</a>
<table class="table">
    <tr>
        <th>Name</th>
        <th>Department</th>
        <th>Designation</th>
        <th>Email</th>
```

```
      <th>Salary</th>
      <th>Actions</th>
    </tr>
    @foreach (var emp in Model)  {
      <tr>
        <td>@emp.Name</td>
        <td>@emp.Department</td>
        <td>@emp.Designation</td>
        <td>@emp.Email</td>
        <td>@emp.Salary</td>
        <td>
          <a asp-action="Details" asp-route-id="@emp.Id">Details</a> |
          <a asp-action="Edit" asp-route-id="@emp.Id">Edit</a> |
          <a asp-action="Delete" asp-route-id="@emp.Id">Delete</a>
        </td>
      </tr>
    }
</table>
```

Views/Employess/Create.cshtml

```
@model EmployeeCRUD.Models.Employee
@{
    ViewData["Title"] = "Create Employee";
}
<h2>Create New Employee</h2>
<form asp-action="Create" method="post">
    <div class="form-group">
        <label asp-for="Name"></label>
        <input asp-for="Name" class="form-control" />
```

```
        <span asp-validation-for="Name" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Department"></label>
        <input asp-for="Department" class="form-control" />
        <span asp-validation-for="Department" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Designation"></label>
        <input asp-for="Designation" class="form-control" />
        <span asp-validation-for="Designation" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Email"></label>
        <input asp-for="Email" class="form-control" />
        <span asp-validation-for="Email" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Salary"></label>
        <input asp-for="Salary" class="form-control" />
        <span asp-validation-for="Salary" class="text-danger"></span>
    </div>
    <br />
    <button type="submit" class="btn btn-success">Create</button>
    <a asp-action="Index" class="btn btn-secondary">Back to List</a>
</form>
@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
```

Views/Employess/Delete.cshtml

```
@model EmployeeCRUD.Models.Employee

@{
   ViewData["Title"] = "Delete Employee";
}

<h2>Delete Employee</h2>

<h4>Are you sure you want to delete this employee?</h4>
<div>
   <dl class="row">
      <dt class="col-sm-2">Name</dt>
      <dd class="col-sm-10">@Model.Name</dd>

      <dt class="col-sm-2">Department</dt>
      <dd class="col-sm-10">@Model.Department</dd>

      <dt class="col-sm-2">Designation</dt>
      <dd class="col-sm-10">@Model.Designation</dd>

      <dt class="col-sm-2">Email</dt>
      <dd class="col-sm-10">@Model.Email</dd>

      <dt class="col-sm-2">Salary</dt>
      <dd class="col-sm-10">@Model.Salary</dd>
   </dl>
</div>
```

```
<form asp-action="Delete" method="post">
    <input type="hidden" asp-for="Id" />
    <button type="submit" class="btn btn-danger">Delete</button>
    <a asp-action="Index" class="btn btn-secondary">Back to List</a>
</form>
```

Views/Employess/Details.cshtml

```
@model EmployeeCRUD.Models.Employee

@{
    ViewData["Title"] = "Employee Details";
}

<h2>Employee Details</h2>

<div>
    <h4>@Model.Name</h4>
    <hr />
    <dl class="row">
        <dt class="col-sm-2">Department</dt>
        <dd class="col-sm-10">@Model.Department</dd>

        <dt class="col-sm-2">Designation</dt>
        <dd class="col-sm-10">@Model.Designation</dd>

        <dt class="col-sm-2">Email</dt>
        <dd class="col-sm-10">@Model.Email</dd>
```

```
        <dt class="col-sm-2">Salary</dt>
        <dd class="col-sm-10">@Model.Salary</dd>
    </dl>
</div>

<a asp-action="Edit" asp-route-id="@Model.Id" class="btn btn-warning">Edit</a>
<a asp-action="Index" class="btn btn-secondary">Back to List</a>
```

Views/Employess/Edit.cshtml

```
@model EmployeeCRUD.Models.Employee

@{
    ViewData["Title"] = "Edit Employee";
}

<h2>Edit Employee</h2>

<form asp-action="Edit" method="post">
    <input type="hidden" asp-for="Id" />

    <div class="form-group">
        <label asp-for="Name"></label>
        <input asp-for="Name" class="form-control" />
        <span asp-validation-for="Name" class="text-danger"></span>
    </div>

    <div class="form-group">
        <label asp-for="Department"></label>
```

```
        <input asp-for="Department" class="form-control" />
        <span asp-validation-for="Department" class="text-danger"></span>
    </div>

    <div class="form-group">
        <label asp-for="Designation"></label>
        <input asp-for="Designation" class="form-control" />
        <span asp-validation-for="Designation" class="text-danger"></span>
    </div>

    <div class="form-group">
        <label asp-for="Email"></label>
        <input asp-for="Email" class="form-control" />
        <span asp-validation-for="Email" class="text-danger"></span>
    </div>

    <div class="form-group">
        <label asp-for="Salary"></label>
        <input asp-for="Salary" class="form-control" />
        <span asp-validation-for="Salary" class="text-danger"></span>
    </div>

    <br />
    <button type="submit" class="btn btn-success">Save</button>
    <a asp-action="Index" class="btn btn-secondary">Back to List</a>
</form>

@section Scripts {
    <partial name="_ValidationScriptsPartial" />
```

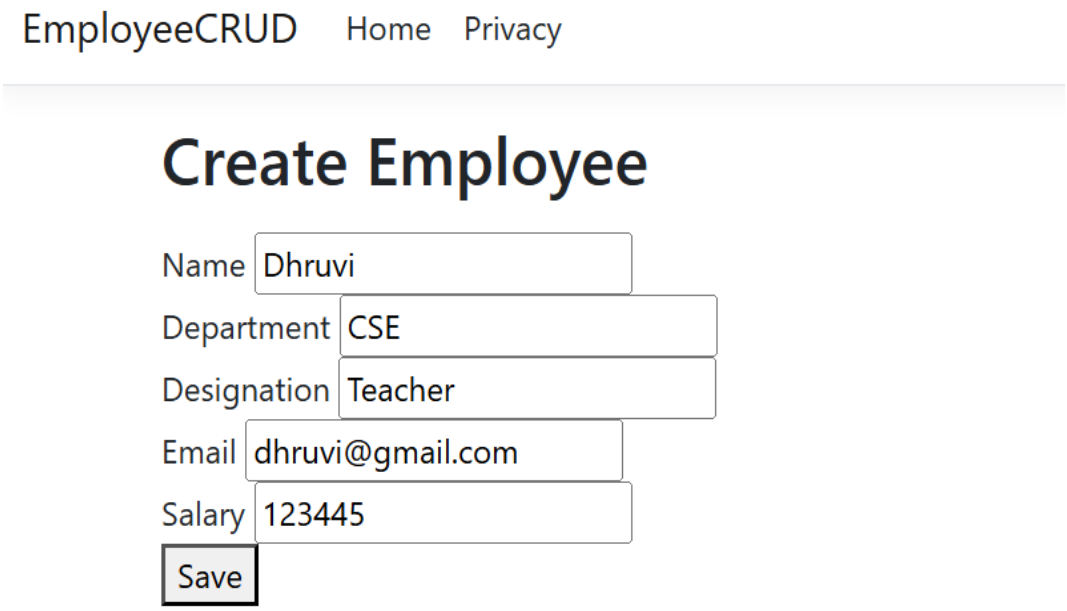
```
}
```

Program.cs

```
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();
```

```
// Configure default route to Employee/Index
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Employee}/{action=Index}/{id?}");

app.Run();
```

Output:



EmployeeCRUD Home Privacy

Employee Details

Name: Dhruvi

Department: CSE

Designation: Teacher

Email: dhruvi@gmail.com

Salary: 123445

Back to List

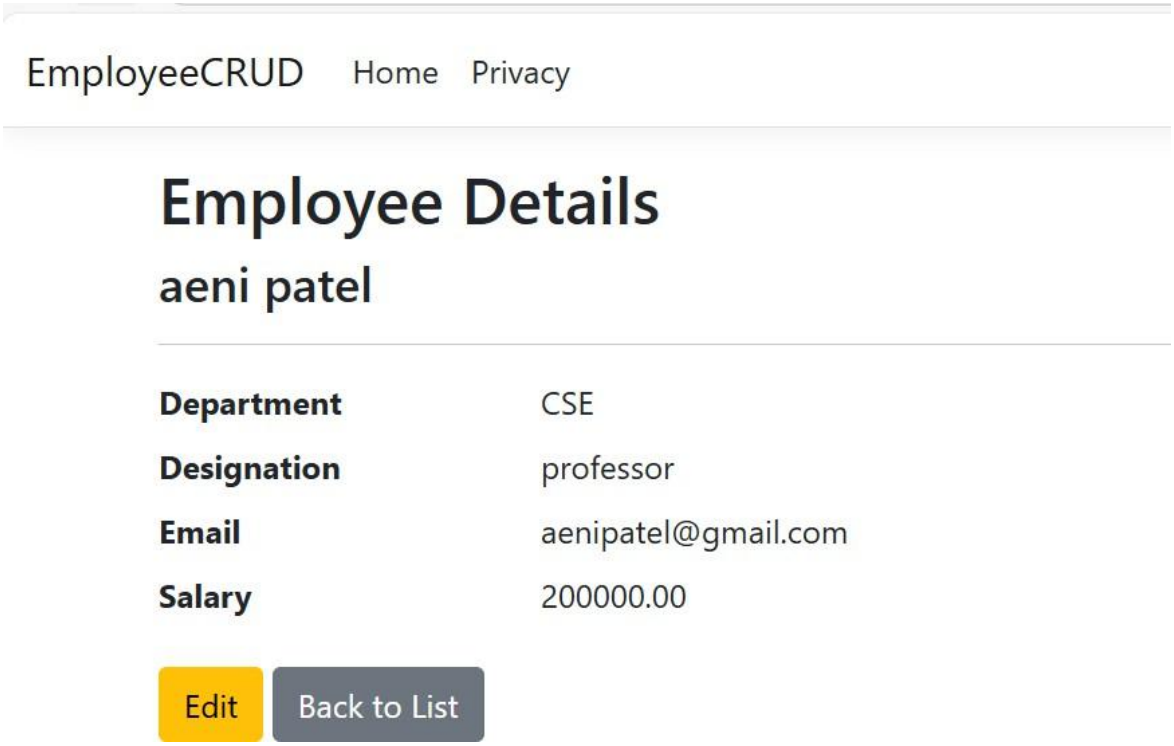
EmployeeCRUD Home Privacy

Employee List

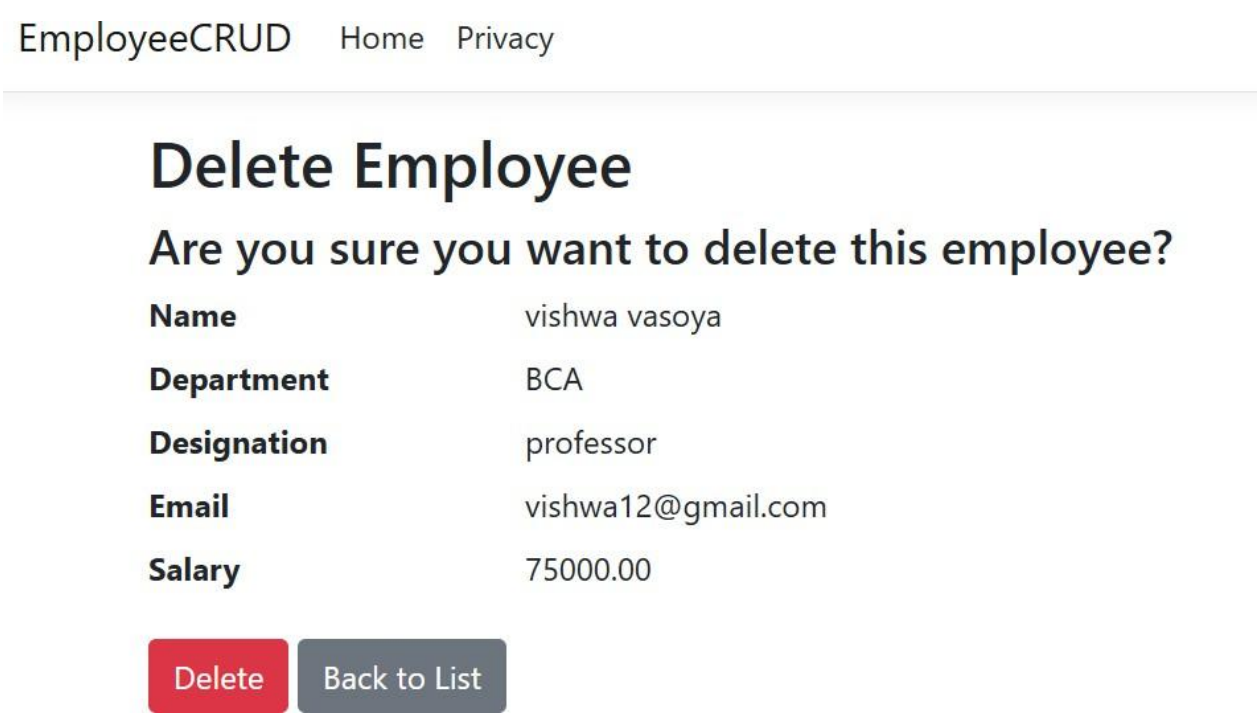
Add New Employee

Name	Department	Designation	Email	Salary	Actions
pranjal	ce	teacer	pranjalpatilk22@gmail.com	12.00	Details | Edit | Delete
Dhruvi	CSE	Teacher	dhruvi@gmail.com	123445	Details | Edit | Delete

Details:



Delete:



Conclusion:

In this practical:

- We built a **web application** using **ASP.NET Core MVC** with an **in-memory list** to manage employee records.
- **Model** defined employee structure and validation rules.
- **Controller** handled CRUD operations on the in-memory list.
- **Views** provided forms and interfaces for creating, reading, updating, and deleting employees.
- This demonstrates how MVC separates concerns and allows managing data without a database.

Practical-7

ASP.NET Core Web API Product Information – CRUD Operations Without Database

Scenario

You are working as a software developer in a retail company's IT department. The inventory team has requested a lightweight RESTful API to manage product data for testing and internal usage. This is an initial prototype and does not require a database at this stage. Instead, all product data will be stored and manipulated in an in-memory list during runtime. Your task is to build a basic ASP.NET Core Web API that allows external applications (e.g., frontend apps, Postman, mobile clients) to create, read, update, and delete product records through HTTP requests. This project will not include views or UI. It will follow REST principles and use the standard HTTP verbs:

● GET to read data,

● POST to create data,

● PUT to update data,

● DELETE to remove data.

Tasks

Task 1: Create a New ASP.NET Core Web API Project

Task 2: Create the Product Model

Define a class Product inside the Models folder with properties:

- Id (int)
- Name (string)
- Category (string)
- Price (decimal)
- Stock (int)

Task 3: Create the ProductController

- Add a controller named ProductController inside the Controllers folder.
- Decorate it with:

[ApiController]

[Route("api/[controller]")]

Task 4: Define In-Memory Data Store

Declare a static or singleton List<Product> in the controller to simulate an in-memory database.

Task 5: Implement CRUD Endpoints Using HTTP Verbs

- GET /api/product – Return the list of products.
- GET /api/product/{id} – Return a specific product by ID.

- POST /api/product – Add a new product to the list.
- PUT /api/product/{id} – Update an existing product by ID.
- DELETE /api/product/{id} – Remove a product from the list byID.

Each action should return appropriate HTTP status codes such as 200 OK, 201 Created, 400 Bad Request, or 404 Not Found.

Task 6: Add Validation with Data Annotations

Apply data validation rules using Data Annotations in the model:

- [Required] for Name and Category
- [Range(0.01, 100000)] for Price
- [Range(0, int.MaxValue)] for Stock

Model validation should be enforced automatically via [ApiController].

Task 7: Test Endpoints Using Swagger or Postman

- Use Swagger UI (auto-generated in ASP.NET Core Web API) or

Postman to:

- Create (POST) new product entries.
- Read (GET) the full list or a single product.
- Update (PUT) a product by ID.
- Delete (DELETE) a product by ID.

Code:

Product.cs

```
using System.ComponentModel.DataAnnotations;

namespace ProductAPI.Model  {

   public class Product    {

      public int Id { get; set; }

      [Required(ErrorMessage = "Name is required")]

      public string Name { get; set; }

      [Required(ErrorMessage = "Category is required")]

      public string Category { get; set; }

      [Range(0.01, 100000, ErrorMessage = "Price must be between 0.01 and 100000")]

      public decimal Price { get; set; }

      [Range(0, int.MaxValue, ErrorMessage = "Stock cannot be negative")]
```

```
        public int Stock { get; set; }
    }
}
```

ProductController.cs

```
using Microsoft.AspNetCore.Mvc;
using ProductAPI.Models;
using System.Collections.Generic;
using System.Linq;
namespace ProductAPI.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class ProductController : ControllerBase
    {
        private static List<Product> products = new List<Product>();

        // GET: api/product
        [HttpGet]
        public IActionResult GetAll()
        {
            return Ok(products);
        }

        // GET: api/product/{id}
        [HttpGet("{id}")]
        public IActionResult GetById(int id)
        {
            var product = products.FirstOrDefault(p => p.Id == id);
```

```
    if (product == null)
        return NotFound(new { message = "Product not found" });
    return Ok(product);
}

// POST: api/product
[HttpPost]
public IActionResult Create(Product product)
{
    if (!ModelState.IsValid)
        return BadRequest(ModelState);
    product.Id = products.Count > 0 ? products.Max(p => p.Id) + 1 : 1;
    products.Add(product);
    return CreatedAtAction(nameof(GetById), new { id = product.Id }, product);
}

// PUT: api/product/{id}
[HttpPut("{id}")]
public IActionResult Update(int id, Product product)
{
    if (!ModelState.IsValid)
        return BadRequest(ModelState);
    var existing = products.FirstOrDefault(p => p.Id == id);
    if (existing == null)
        return NotFound(new { message = "Product not found" });
    existing.Name = product.Name;
    existing.Category = product.Category;
    existing.Price = product.Price;
    existing.Stock = product.Stock;
```

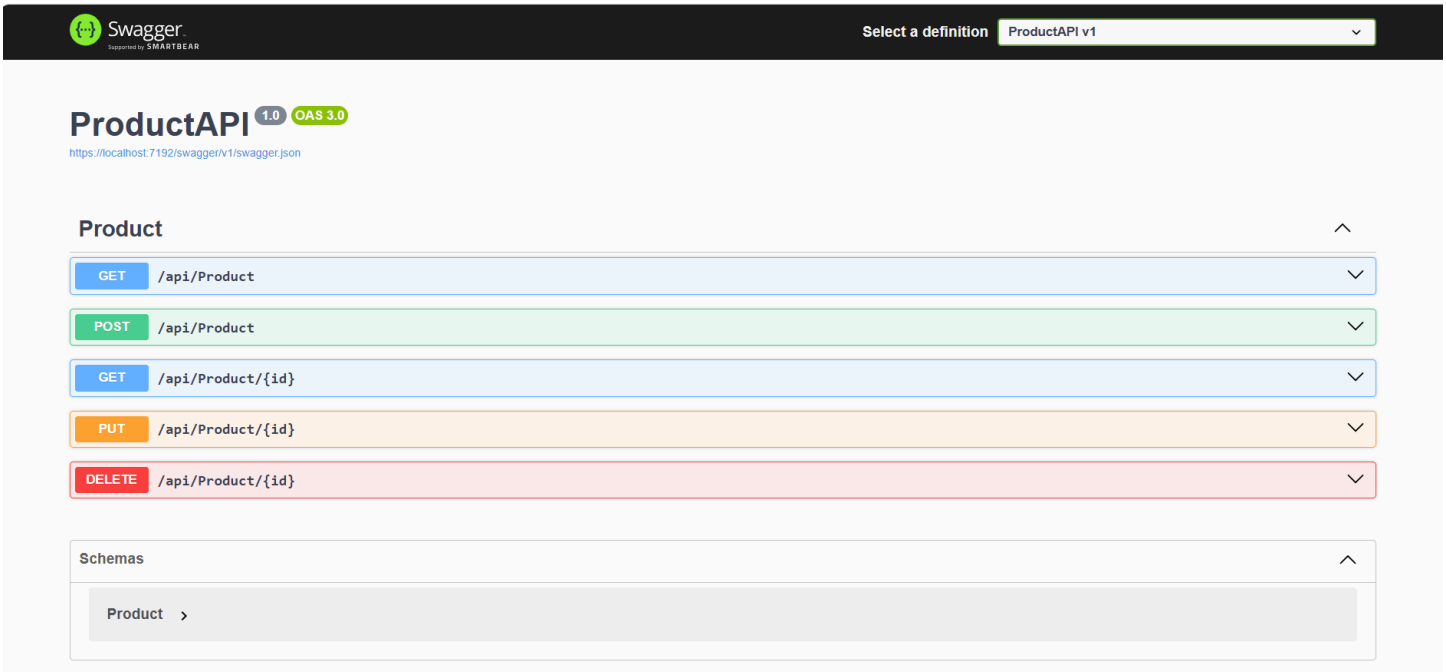
```
            return Ok(existing);
        }

        // DELETE: api/product/{id}
        [HttpDelete("{id}")]
        public IActionResult Delete(int id)
        {
            var product = products.FirstOrDefault(p => p.Id == id);
            if (product == null)
                return NotFound(new { message = "Product not found" });
            products.Remove(product);
            return Ok(new { message = "Product deleted successfully" });
        }
    }
}
```

Program.cs

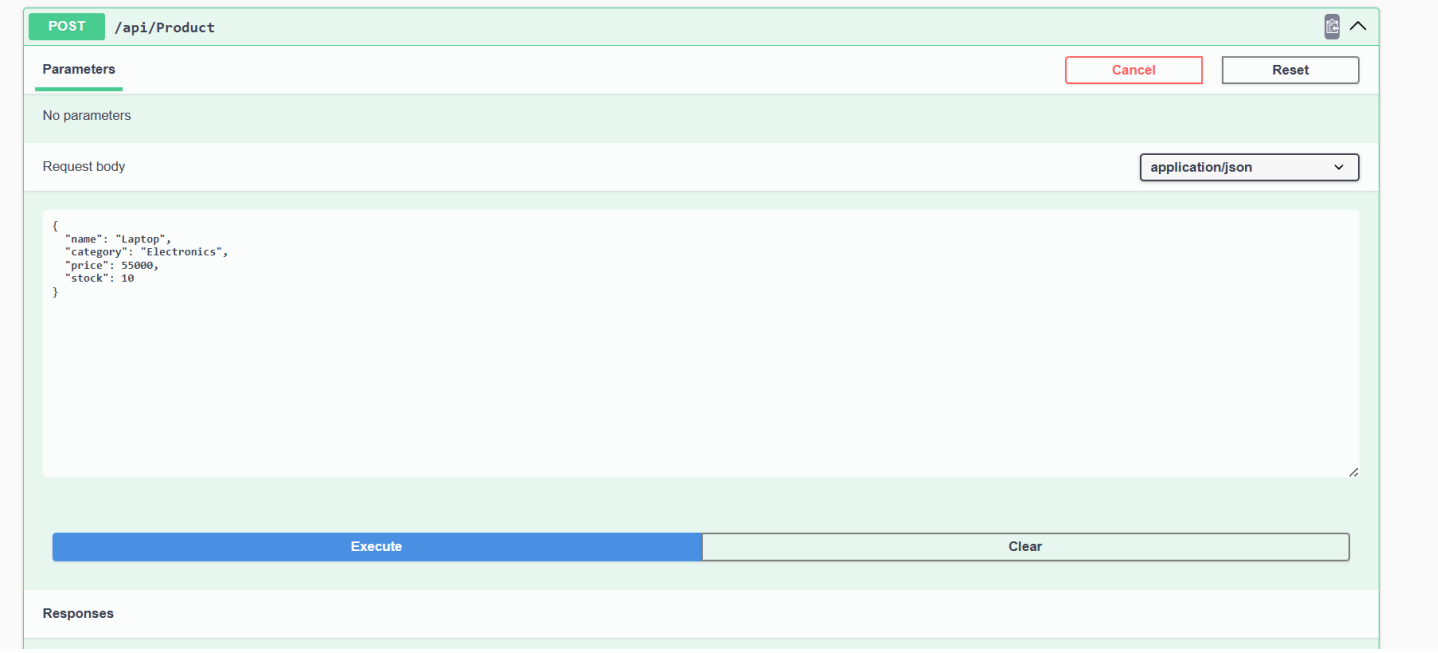
```
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
var builder = WebApplication.CreateBuilder(args);

// Add services
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer(); // For Swagger
builder.Services.AddSwaggerGen();

var app = builder.Build();
// Middleware
```

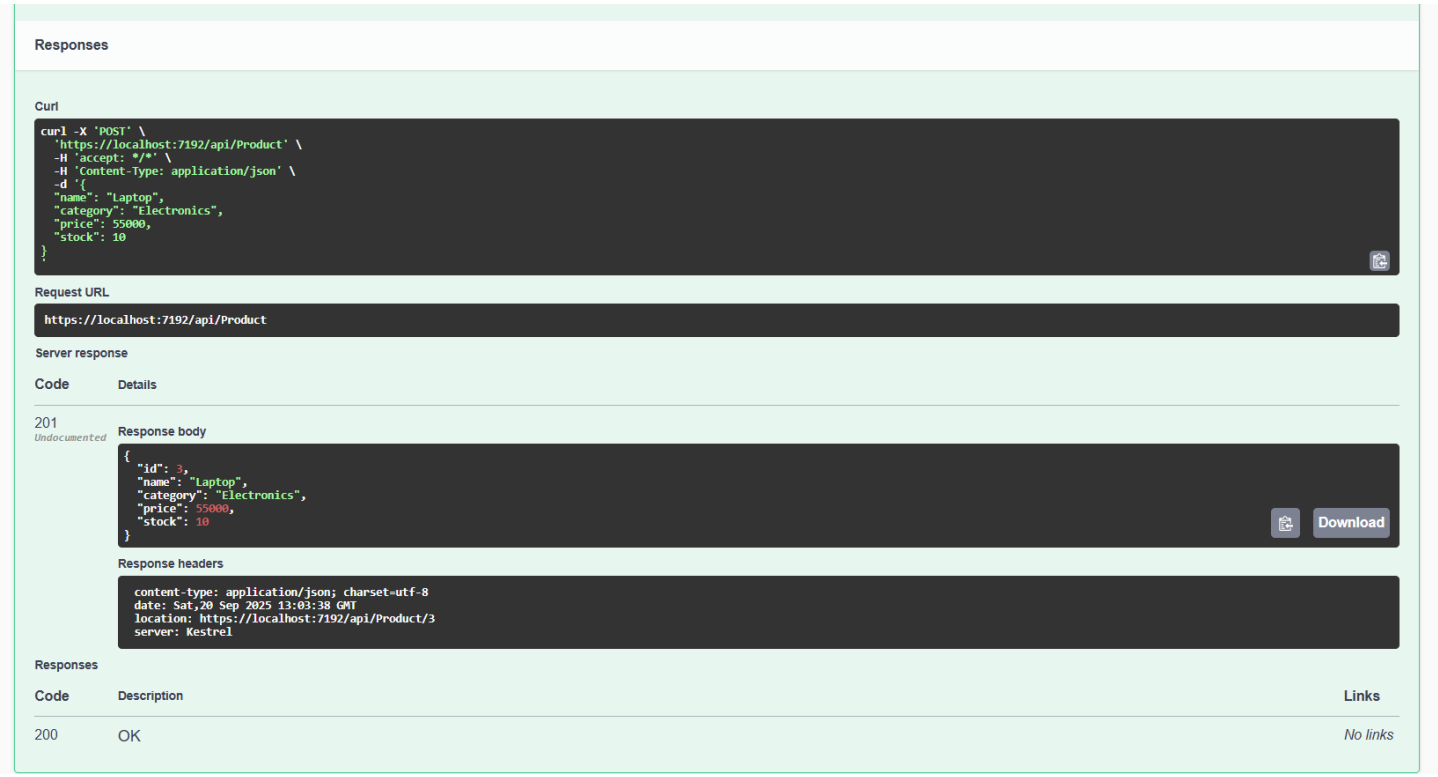
```
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();
app.Run();
```

Output:

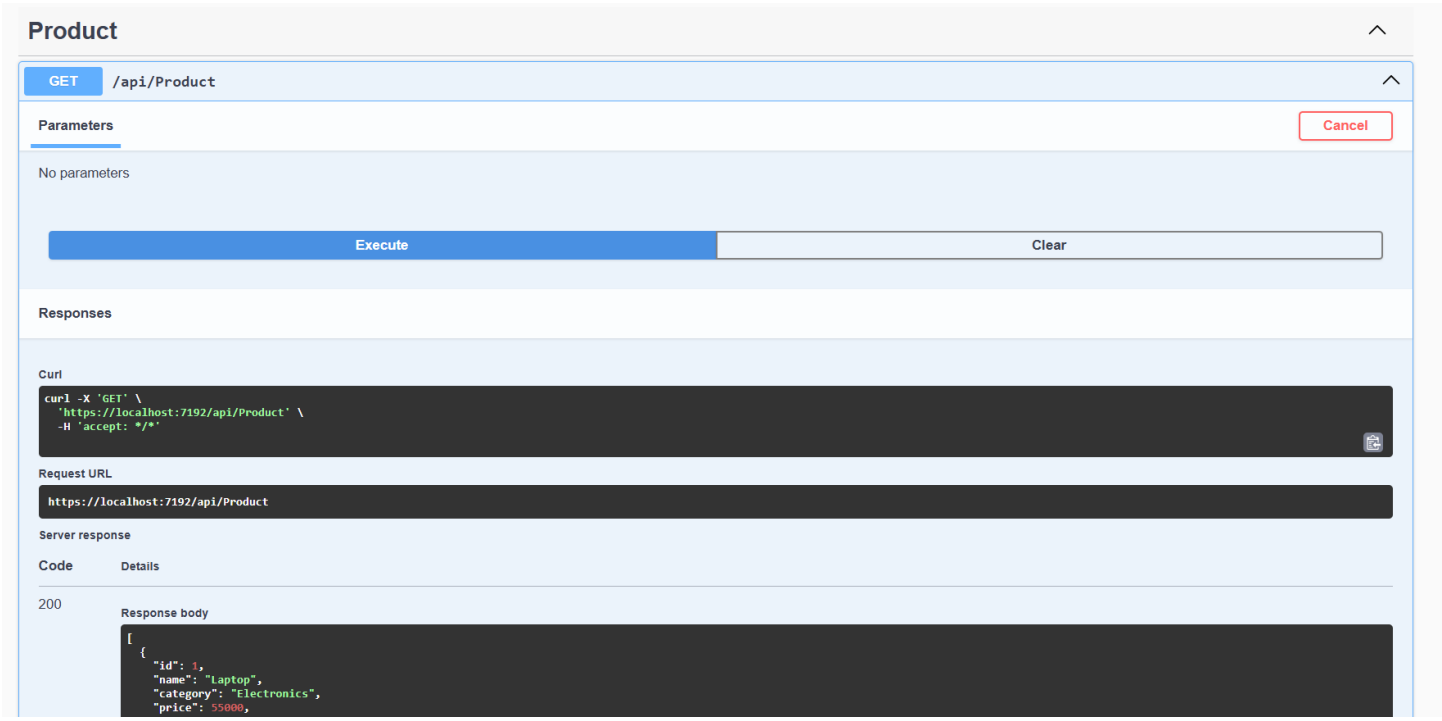


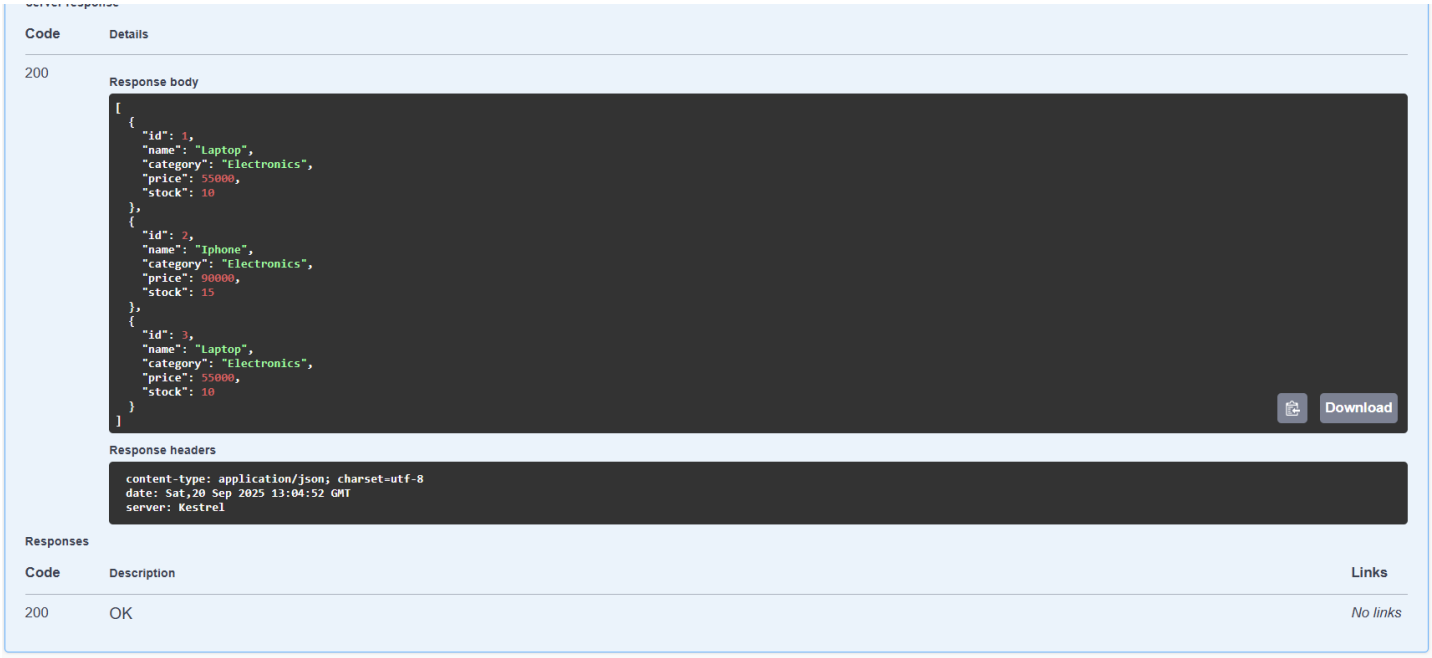
Post Product(create)





Get Product:


Product
GET
/api/Product
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
curl -X 'GET' \
'https://localhost:7192/api/Product' \
-H 'accept: */*'
Request URL
https://localhost:7192/api/Product
Server response
Code
Details
200
Response body
[
{
"id": 1,
"name": "Laptop",
"category": "Electronics",
"price": 55000,

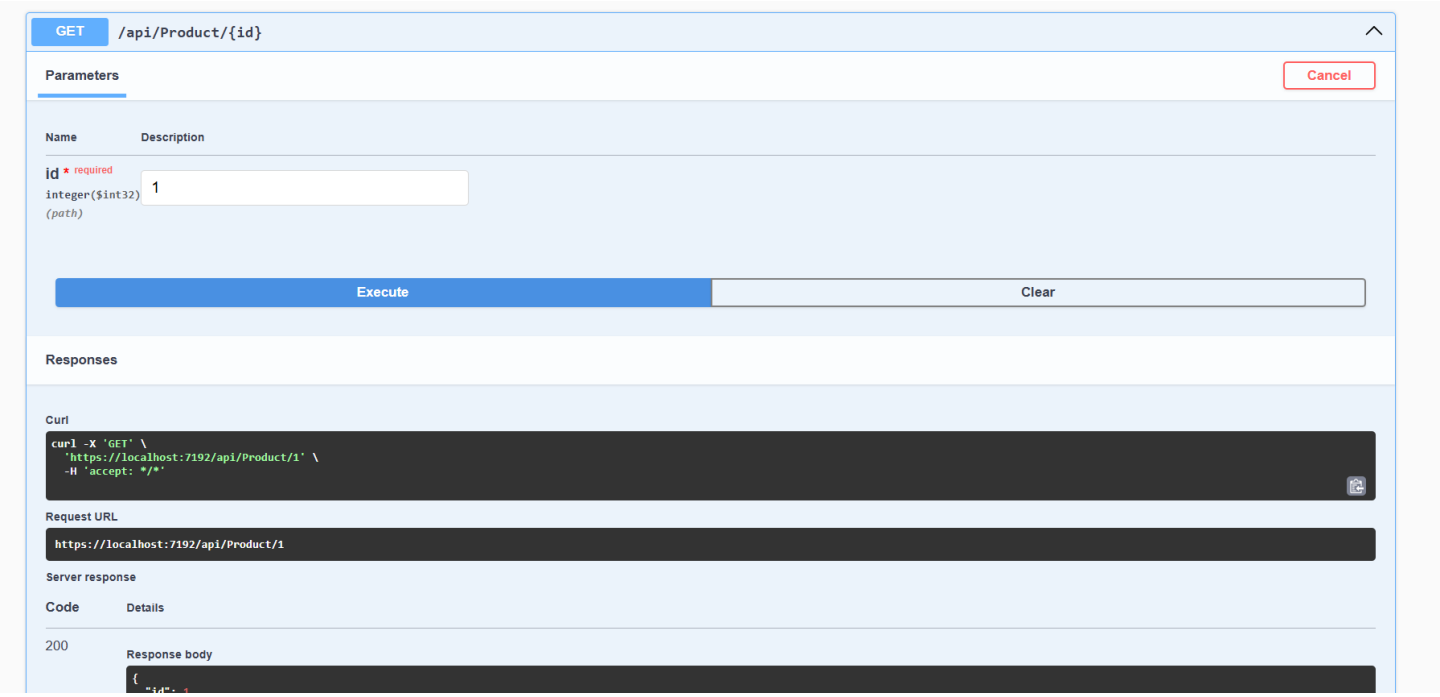

Code
Details
200
Response body
[
{
"id": 1,
"name": "Laptop",
"category": "Electronics",
"price": 55000,
"stock": 10
},
{
"id": 2,
"name": "Iphone",
"category": "Electronics",
"price": 90000,
"stock": 15
},
{
"id": 3,
"name": "Laptop",
"category": "Electronics",
"price": 55000,
"stock": 10
}
]
Download
Response headers
content-type: application/json; charset=utf-8
date: Sat,20 Sep 2025 13:04:52 GMT
server: Kestrel
Responses
Code
Description
Links
200
OK
No links

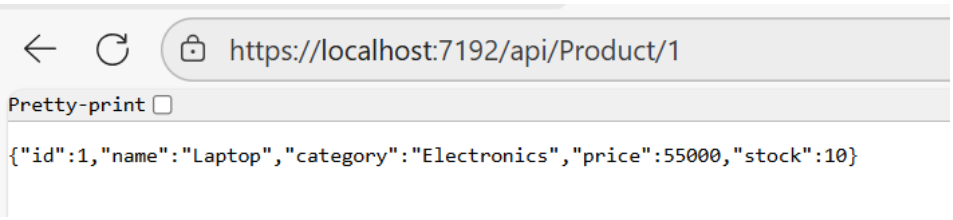
https://localhost:7192/api/Product

Pretty-print

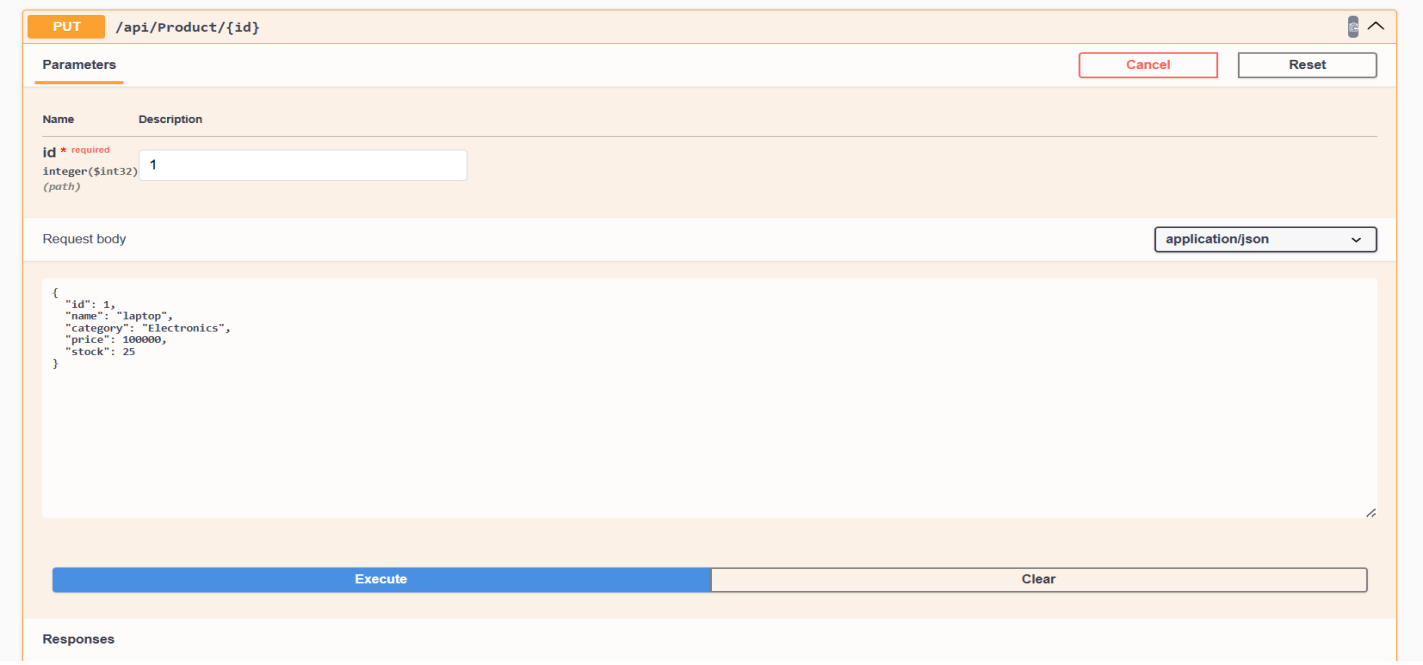
[{"id":1,"name":"Laptop","category":"Electronics","price":55000,"stock":10},
{"id":2,"name":"Iphone","category":"Electronics","price":90000,"stock":15},
{"id":3,"name":"Laptop","category":"Electronics","price":55000,"stock":10},
{"id":4,"name":"string","category":"string","price":100000,"stock":2147483647}]

Product get by id:

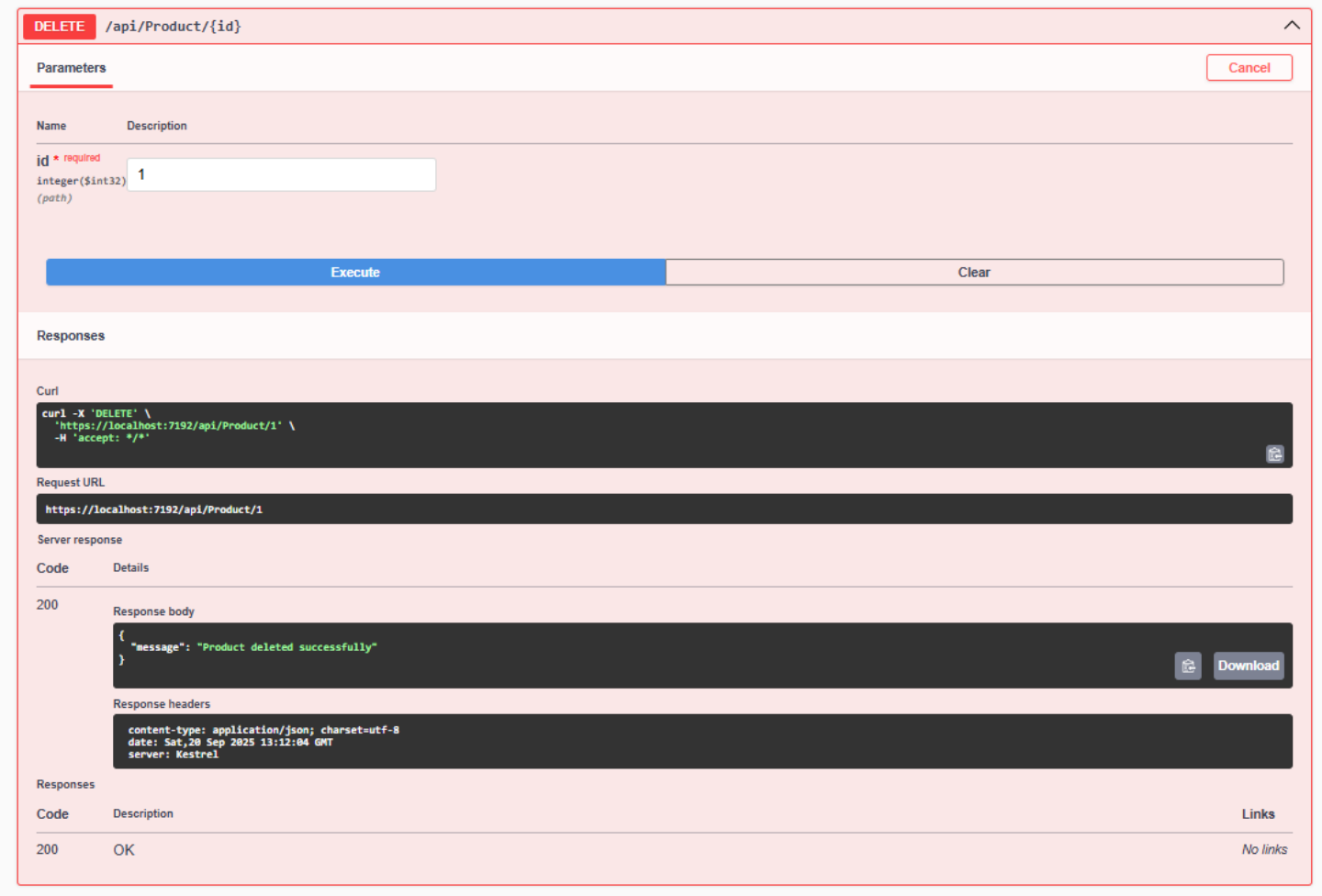


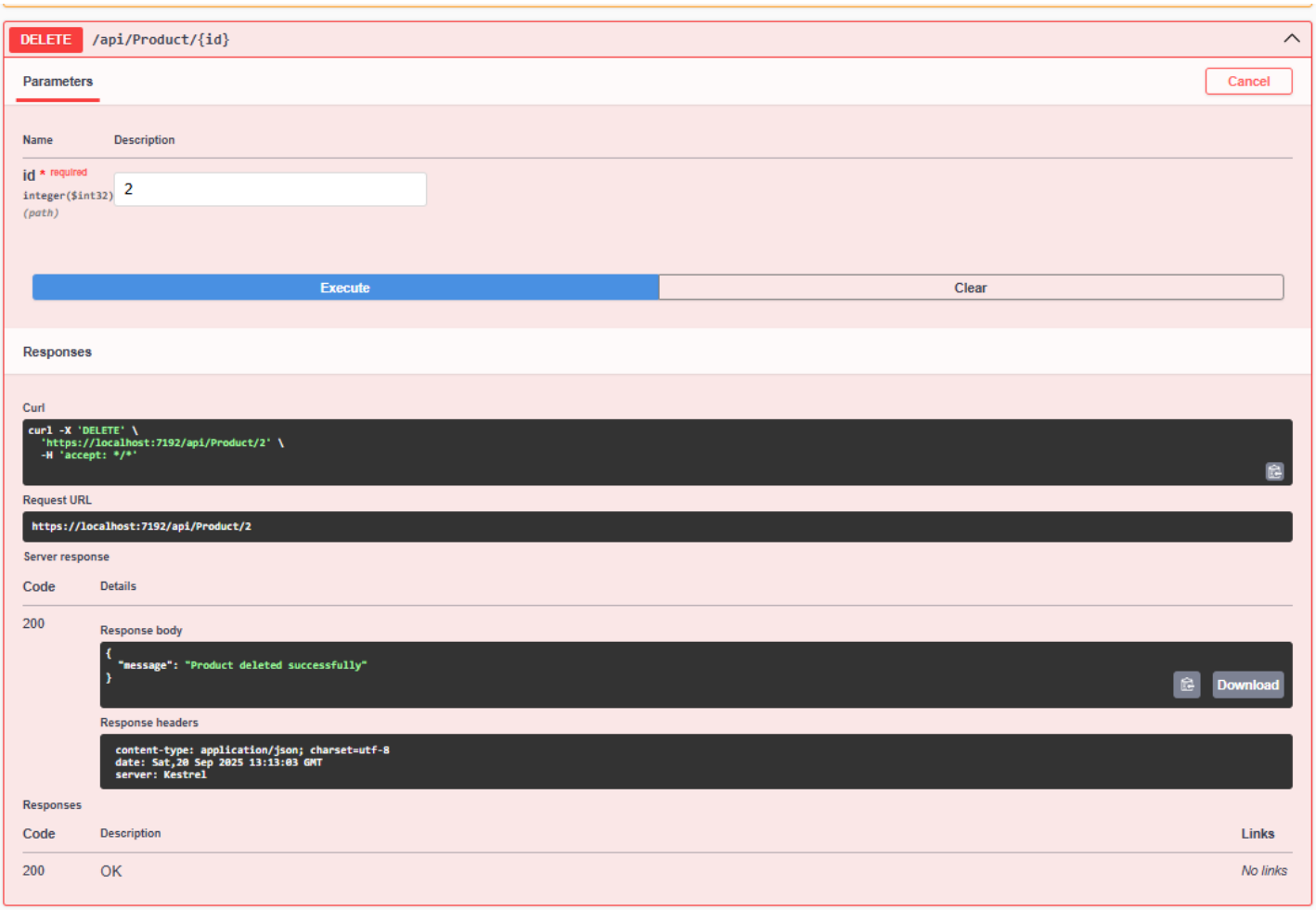


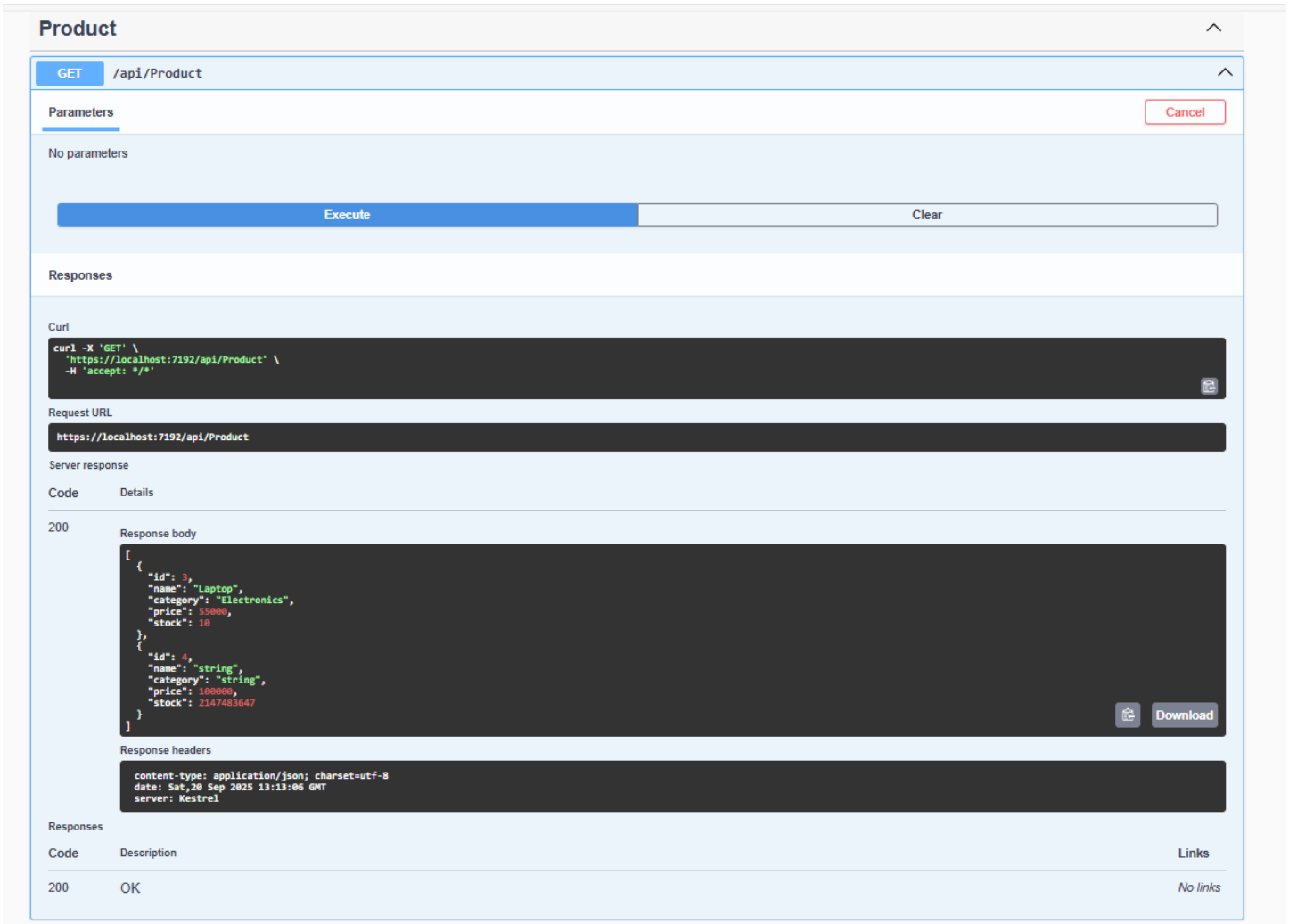
Put by Id(Edit):

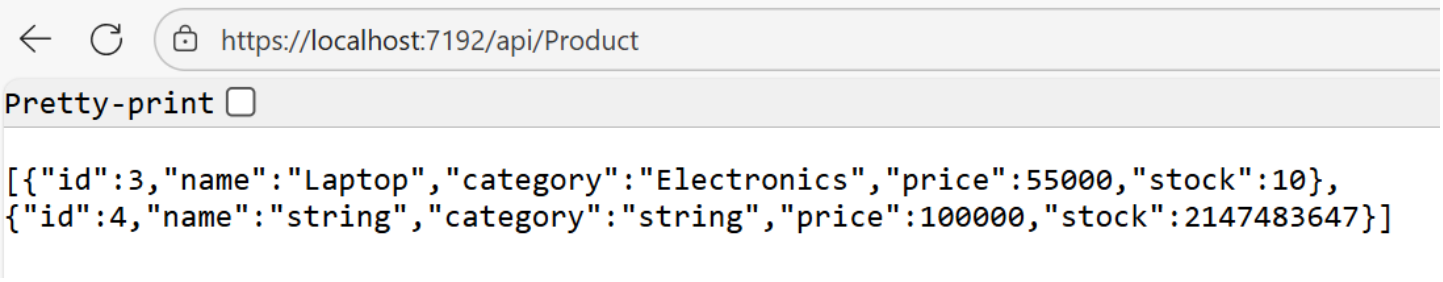


Delete by id:

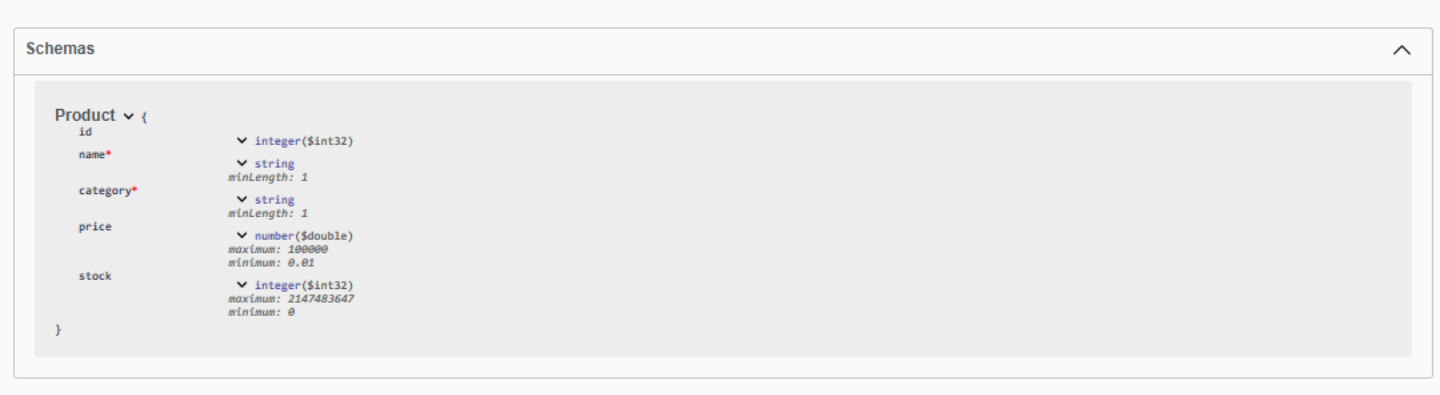

DELETE
/api/Product/{id}
Parameters
Cancel
Name
Description
id * required
integer($int32)
(path)
1
Execute
Clear
Responses
Curl
curl -X 'DELETE' \
'https://localhost:7192/api/Product/1' \
-H 'accept: */*'
Request URL
https://localhost:7192/api/Product/1
Server response
Code
Details
200
Response body
{
"message": "Product deleted successfully"
}
Download
Response headers
content-type: application/json; charset=utf-8
date: Sat,20 Sep 2025 13:12:04 GMT
server: Kestrel
Responses
Code
Description
Links
200
OK
No links


DELETE
/api/Product/{id}
Parameters
Cancel
Name
Description
id * required
integer($int32)
(path)
2
Execute
Clear
Responses
Curl
curl -X 'DELETE' \
'https://localhost:7192/api/Product/2' \
-H 'accept: */*'
Request URL
https://localhost:7192/api/Product/2
Server response
Code
Details
200
Response body
{
"message": "Product deleted successfully"
}
Download
Response headers
content-type: application/json; charset=utf-8
date: Sat,20 Sep 2025 13:13:03 GMT
server: Kestrel
Responses
Code
Description
Links
200
OK
No links


Product
GET
/api/Product
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
curl -X 'GET' \
'https://localhost:7192/api/Product' \
-H 'accept: */*'
Request URL
https://localhost:7192/api/Product
Server response
Code
Details
200
Response body
[
{
"id": 3,
"name": "Laptop",
"category": "Electronics",
"price": 55000,
"stock": 10
},
{
"id": 4,
"name": "string",
"category": "string",
"price": 100000,
"stock": 2147483647
}
]
Download
Response headers
content-type: application/json; charset=utf-8
date: Sat,20 Sep 2025 13:13:06 GMT
server: Kestrel
Responses
Code
Description
Links
200
OK
No links


https://localhost:7192/api/Product
Pretty-print
[{"id":3,"name":"Laptop","category":"Electronics","price":55000,"stock":10},
{"id":4,"name":"string","category":"string","price":100000,"stock":2147483647}]

Product schema:



Conclusion:

In this practical:

- We developed a **RESTful Web API** using **ASP.NET Core** with an **in-memory list** to manage product data.
- **Model** defined the product structure and validation rules.
- **Controller** implemented CRUD endpoints using standard HTTP verbs (GET, POST, PUT, DELETE).
- External clients (Postman, Swagger) could interact with the API, showing how data can be manipulated programmatically.
- This demonstrates the use of **REST principles** and server-side validation without a UI.

Practical-8

ASP.NET Core MVC with Entity Framework Core Product Information Manager – CRUD Operations Using MVC and Database

Scenario

You are working as a software developer in a retail company's IT department. The inventory management team has requested a web-based system to manage product information. The system should allow users to add, view, edit, and delete product records from a database. Your task is to build a basic ASP.NET Core MVC application using Entity Framework Core for database operations. This application will provide a user-friendly interface to manage product information such as Product ID, Name, Category, Price, and Quantity.

The application should follow the MVC architecture:

● The Model will define the product structure.

● The Controller will handle CRUD logic and communicate with the database.

● The Views will provide interfaces for user interaction.

Tasks:

Task 1: Create a New ASP.NET Core MVC Project

Task 2: Configure SQL Server and Entity Framework Core

- Install the necessary NuGet packages:
- Microsoft.EntityFrameworkCore.SqlServer
- Microsoft.EntityFrameworkCore.Tools
- Configure the connection string in appsettings.json.

Task 3: Create the Product Model

Add a Product class inside the Models folder with fields:

- ProductId (int)
- Name (string)
- Category (string)
- Price (decimal)
- Quantity (int)

Apply Data Annotations such as:

- [Required], [StringLength], [Range]

Task 4: Setup the Database Context Class

Create a ProductDbContext class inheriting from DbContext and add DbSet<Product> Products.

Task 5: Apply EF Core Migrations

- Use CLI or Package Manager Console:

Add-Migration InitialCreate

Update-Database

- Ensure the SQL Server database is created with the Products table.

Task 6: Scaffold Controller and Views

- Use Scaffolding to generate:
- ProductController
- Razor Views for Index, Create, Edit, Details, and Delete
- Connect them to EF Core through the context.

Task 7: Configure Default Route

Set the default controller and action in Program.cs:

app.MapControllerRoute(

name: "default",

pattern:

"{controller=Product}/{action=Index}/{id?}");

Task 8: Build and Run the Application

- Launch the application in a browser.
- Verify all CRUD operations (Create, Read, Update, Delete) are working.
- Data should persist in the SQL Server database.

Task 9: Add Validation to Product Model

- Ensure server-side validation rules are enforced.
- Display error messages using Razor helpers like:

@Html.ValidationMessageFor(model => model.Price)

Code:

Models/Product.cs

```
using System.ComponentModel.DataAnnotations;

namespace ProductInfoManager.Models {

   public class Product  {

      public int ProductId { get; set; }
```

```
        [Required]
        [StringLength(100)]
        public string Name { get; set; }
        [Required]
        [StringLength(50)]
        public string Category { get; set; }
        [Range(1, 100000)]
        public decimal Price { get; set; }
        [Range(0, 10000)]
        public int Quantity { get; set; }
    }
}
```

Controllers/ProductController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using ProductInfoManager.Data;
using ProductInfoManager.Models;

namespace ProductInfoManager.Controllers {
    public class ProductController : Controller  {
        private readonly ProductDbContext _context;
        public ProductController(ProductDbContext context)  {
            _context = context;
        }

        // GET: Product
        public IActionResult Index()  {
            var products = _context.Products.ToList();
```

```
    return View(products);
}
// GET: Product/Details/5
public IActionResult Details(int? id)     {
    if (id == null)
        return NotFound();
    var product = _context.Products.FirstOrDefault(p => p.ProductId == id);
    if (product == null)
        return NotFound();
    return View(product);
}

// GET: Product/Create
public IActionResult Create()  {
    return View();
}

// POST: Product/Create
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Create(Product product)  {
    if (ModelState.IsValid)  {
        _context.Products.Add(product);
        _context.SaveChanges();
        return RedirectToAction(nameof(Index));
    }
    return View(product);
}
```

```
// GET: Product/Edit/5
public IActionResult Edit(int? id)  {
   if (id == null)
      return NotFound();
   var product = _context.Products.Find(id);
   if (product == null)
      return NotFound();
   return View(product);
}

// POST: Product/Edit/5
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Edit(int id, Product product)   {
   if (id != product.ProductId)
      return NotFound();
   if (ModelState.IsValid)  {
      _context.Entry(product).State = EntityState.Modified;
      _context.SaveChanges();
      return RedirectToAction(nameof(Index));
   }
   return View(product);
}

// GET: Product/Delete/5
public IActionResult Delete(int? id)   {
   if (id == null)
      return NotFound();
   var product = _context.Products.FirstOrDefault(p => p.ProductId == id);
```

```
        if (product == null)
            return NotFound();
        return View(product);
    }

    // POST: Product/Delete/5
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public IActionResult DeleteConfirmed(int id) {
        var product = _context.Products.Find(id);
        if (product != null) {
            _context.Products.Remove(product);
            _context.SaveChanges();
        }
        return RedirectToAction(nameof(Index));
    }
  }
}
```

Data/ProductDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using ProductInfoManager.Models;
using System.Collections.Generic;
namespace ProductInfoManager.Data {
  public class ProductDbContext : DbContext {
    public ProductDbContext(DbContextOptions<ProductDbContext> options) : base(options) { }
    public DbSet<Product> Products { get; set; }
  }
}
```

Views/Creat.cshtml

```
@model ProductInfoManager.Models.Product
@{
   ViewData["Title"] = "Create Product";
}
<h2>Add New Product</h2>
<form asp-action="Create" method="post">
   <div class="form-group">
      <label asp-for="Name"></label>
      <input asp-for="Name" class="form-control" />
      <span asp-validation-for="Name" class="text-danger"></span>
   </div>
   <div class="form-group">
      <label asp-for="Category"></label>
      <input asp-for="Category" class="form-control" />
      <span asp-validation-for="Category" class="text-danger"></span>
   </div>
   <div class="form-group">
      <label asp-for="Price"></label>
      <input asp-for="Price" class="form-control" />
      <span asp-validation-for="Price" class="text-danger"></span>
   </div>
   <div class="form-group">
      <label asp-for="Quantity"></label>
      <input asp-for="Quantity" class="form-control" />
      <span asp-validation-for="Quantity" class="text-danger"></span>
   </div>
```

```
    <br />
    <button type="submit" class="btn btn-success">Save</button>
    <a asp-action="Index" class="btn btn-secondary">Cancel</a>
</form>
@section Scripts {
    @{
        await Html.RenderPartialAsync("_ValidationScriptsPartial");
    }
}
```

Views/Index.cshtml

```
@model IEnumerable<ProductInfoManager.Models.Product>
@{
    ViewData["Title"] = "Products";
}
<h2>Product List</h2>
<p>
    <a class="btn btn-primary" asp-action="Create">Add New Product</a>
</p>
<table class="table table-striped">
    <thead>
        <tr>
            <th>Name</th>
            <th>Category</th>
            <th>Price</th>
            <th>Quantity</th>
            <th>Actions</th>
        </tr>
    </thead>
```

```
    <tbody>
        @foreach (var item in Model)
        {
            <tr>
                <td>@item.Name</td>
                <td>@item.Category</td>
                <td>@item.Price</td>
                <td>@item.Quantity</td>
                <td>
                    <a class="btn btn-info btn-sm" asp-action="Details" asp-route-id="@item.ProductId">Details</a>
                    <a class="btn btn-warning btn-sm" asp-action="Edit" asp-route-id="@item.ProductId">Edit</a>
                    <a class="btn btn-danger btn-sm" asp-action="Delete" asp-route-id="@item.ProductId">Delete</a>
                </td>
            </tr>
        }
    </tbody>
</table>
```

Views/Edit.cshtml

```
@model ProductInfoManager.Models.Product
@{
    ViewData["Title"] = "Edit Product";
}
<h2>Edit Product</h2>
<form asp-action="Edit" method="post">
    <input type="hidden" asp-for="ProductId" />
    <div class="form-group">
```

```
        <label asp-for="Name"></label>
        <input asp-for="Name" class="form-control" />
        <span asp-validation-for="Name" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Category"></label>
        <input asp-for="Category" class="form-control" />
        <span asp-validation-for="Category" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Price"></label>
        <input asp-for="Price" class="form-control" />
        <span asp-validation-for="Price" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Quantity"></label>
        <input asp-for="Quantity" class="form-control" />
        <span asp-validation-for="Quantity" class="text-danger"></span>
    </div>
    <br />
    <button type="submit" class="btn btn-success">Update</button>
    <a asp-action="Index" class="btn btn-secondary">Cancel</a>
</form>

@section Scripts {
    @{
        await Html.RenderPartialAsync("_ValidationScriptsPartial");
    }
}
```

Views/Details.cshtml

```
@model ProductInfoManager.Models.Product
@{
    ViewData["Title"] = "Product Details";
}
<h2>Product Details</h2>
<div>
    <h4>@Model.Name</h4>
    <hr />
    <dl class="row">
        <dt class="col-sm-2">Category</dt>
        <dd class="col-sm-10">@Model.Category</dd>
        <dt class="col-sm-2">Price</dt>
        <dd class="col-sm-10">@Model.Price</dd>
        <dt class="col-sm-2">Quantity</dt>
        <dd class="col-sm-10">@Model.Quantity</dd>
    </dl>
</div>
<a asp-action="Index" class="btn btn-secondary">Back to List</a>
```

Views/Delete.cshtml

```
@model ProductInfoManager.Models.Product
@{
    ViewData["Title"] = "Delete Product";
}
@section Scripts {
```

```
    @{
        await Html.RenderPartialAsync("_ValidationScriptsPartial");
    }
}
<h2>Delete Product</h2>
<h3>Are you sure you want to delete this product?</h3>
<div>
    <h4>@Model.Name</h4>
    <hr />
    <dl class="row">
        <dt class="col-sm-2">Category</dt>
        <dd class="col-sm-10">@Model.Category</dd>
        <dt class="col-sm-2">Price</dt>
        <dd class="col-sm-10">@Model.Price</dd>
        <dt class="col-sm-2">Quantity</dt>
        <dd class="col-sm-10">@Model.Quantity</dd>
    </dl>
</div>
<form asp-action="Delete" method="post">
    <input type="hidden" asp-for="ProductId" />
    <button type="submit" class="btn btn-danger">Delete</button>
    <a asp-action="Index" class="btn btn-secondary">Cancel</a>
</form>
```

Program.cs

```
using Microsoft.EntityFrameworkCore;
using ProductInfoManager.Data;
var builder = WebApplication.CreateBuilder(args);
// Add services to the container.
```

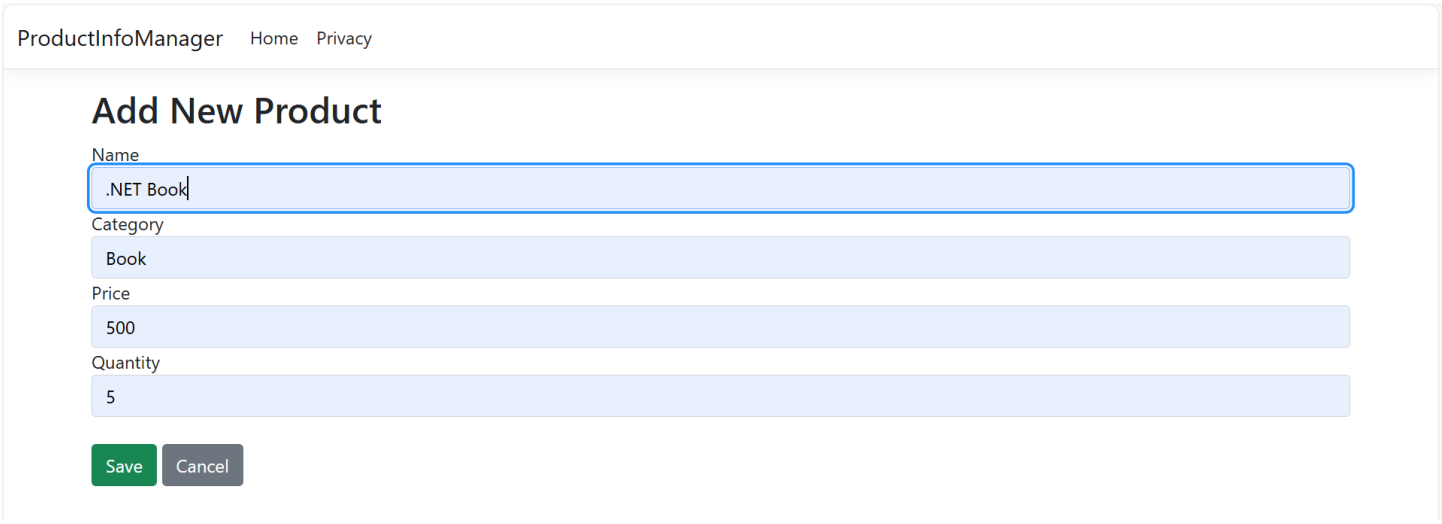
```
builder.Services.AddControllersWithViews();
builder.Services.AddDbContext<ProductDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));
var app = builder.Build();
// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment()) {
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseAuthorization();
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Product}/{action=Index}/{id?}");
app.Run();
```

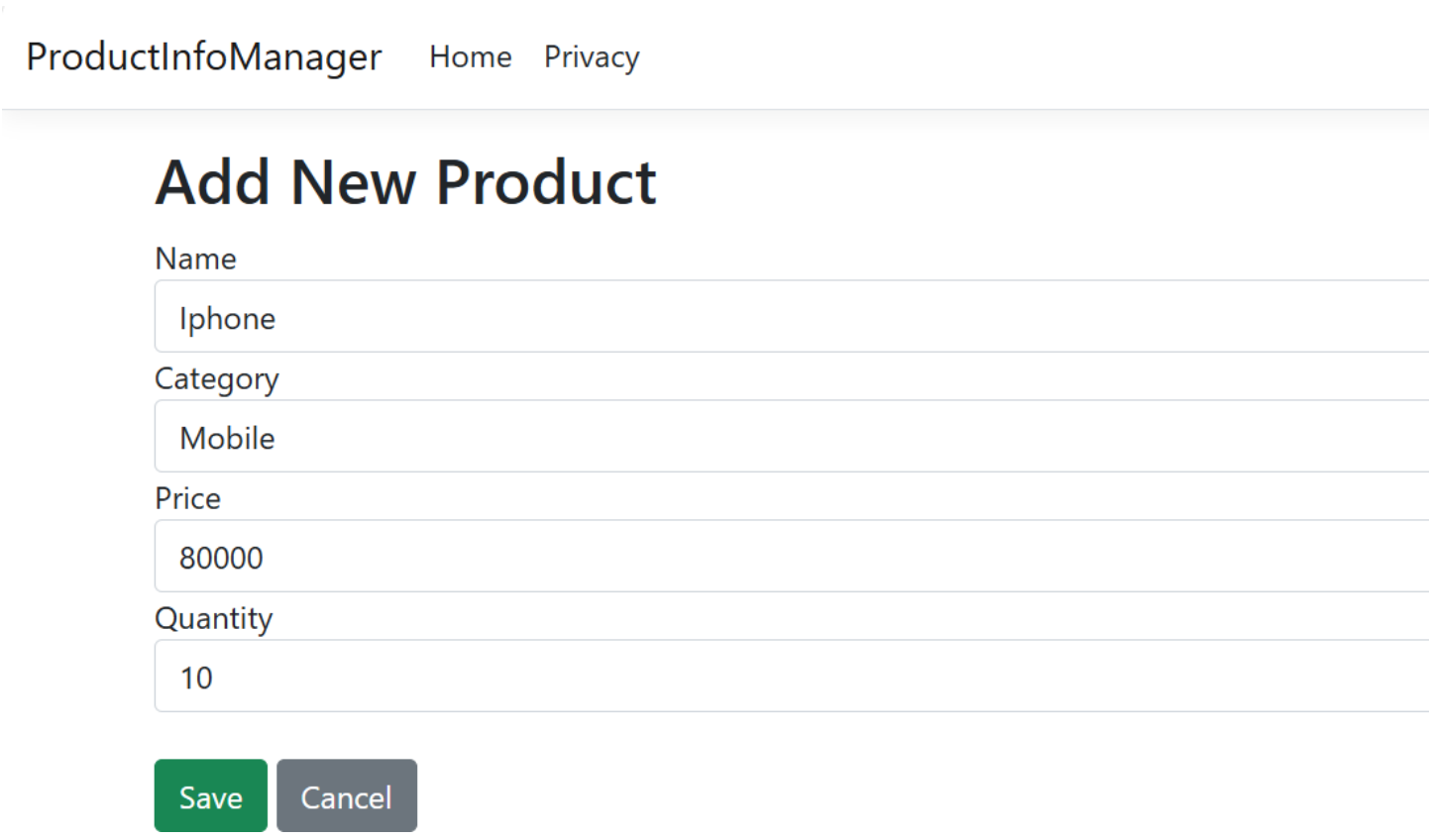
appsettings.json

```
{
  "ConnectionStrings": {
    "DefaultConnection":
"Server=(localdb)\\MSSQLLocalDB;Database=ProductDB;Trusted_Connection=True;MultipleActiveR
esultSets=true"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
```

```
  },
  "AllowedHosts": "*"
}
```

Output:





ProductInfoManager Home Privacy

Product List

Add New Product

Name	Category	Price	Quantity	Actions
.NET Book	Book	500.00	5	Details Edit Delete
Iphone	Mobile	80000.00	10	Details Edit Delete

ProductInfoManager Home Privacy

Product List

Add New Product

Name	Category	Price	Quantity	Actions
.NET Book	Book	500.00	5	Details Edit Delete
Iphone	Mobile	80000.00	10	Details Edit Delete
TV	Home Applications	25000.00	8	Details Edit Delete
Refrigerator	Home Applications	50000.00	15	Details Edit Delete

Details:

ProductInfoManager Home Privacy

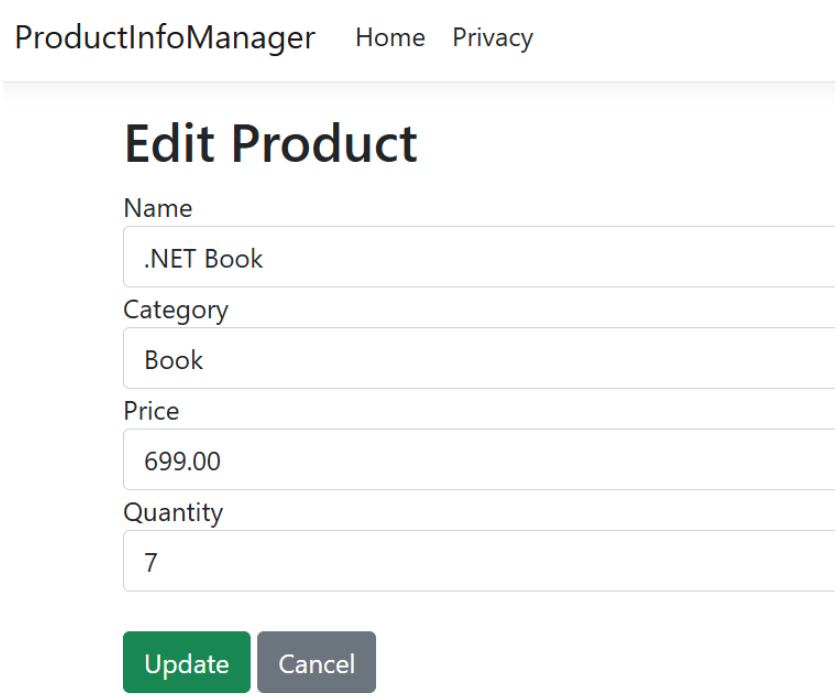
Product Details

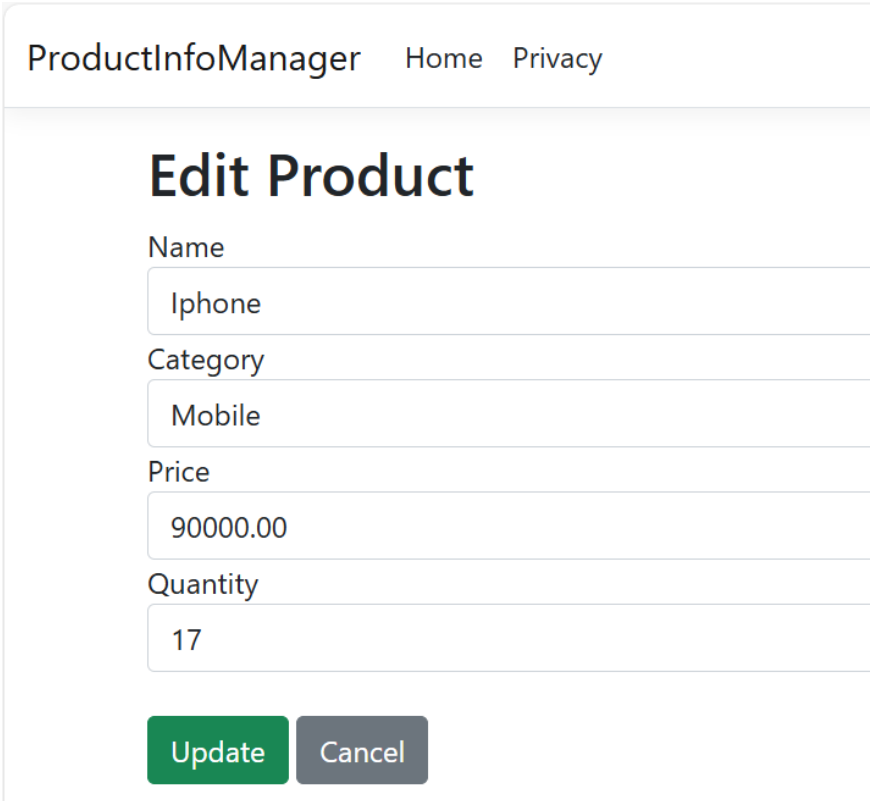
.NET Book

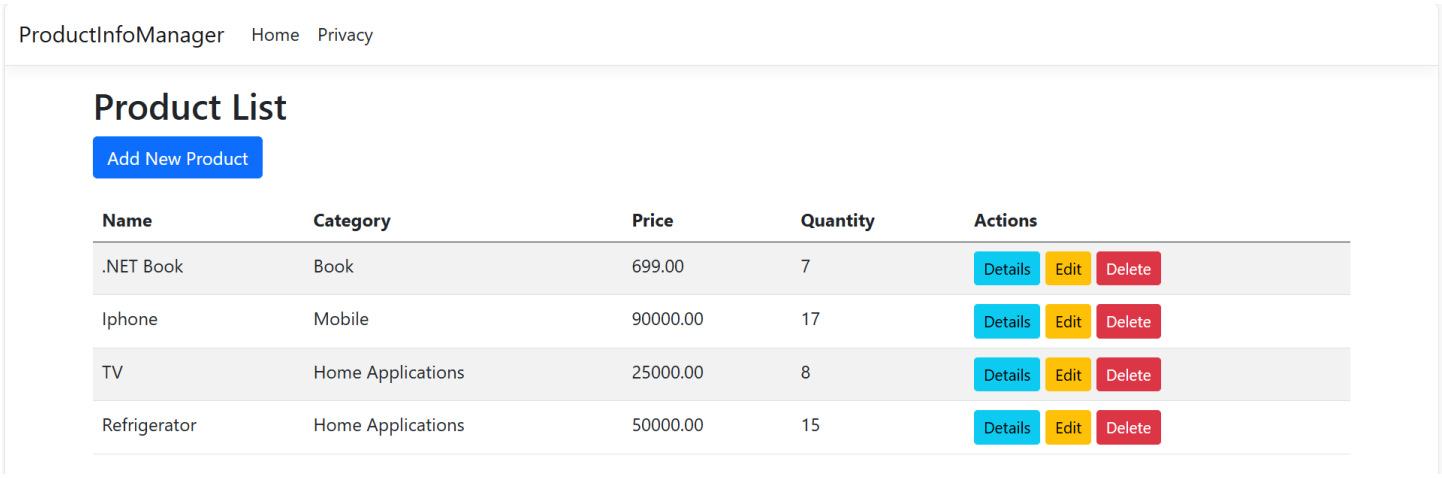
Category	Book
Price	500.00
Quantity	5

Back to List

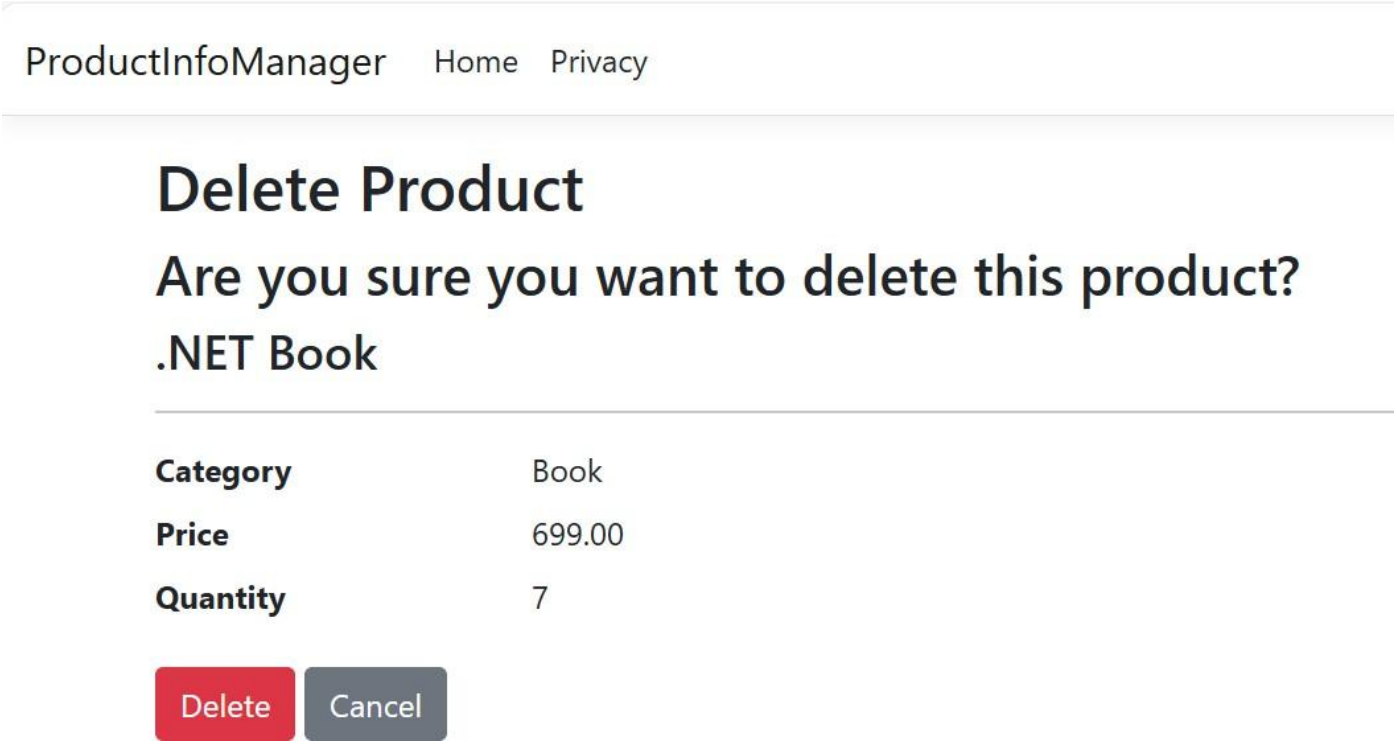
Edit:

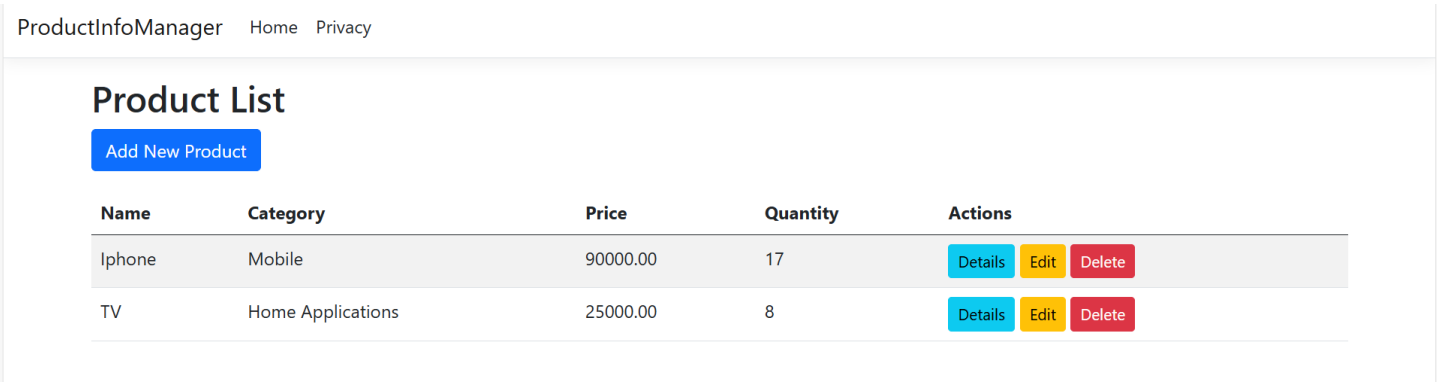

ProductInfoManager
Home
Privacy
Edit Product
Name
.NET Book
Category
Book
Price
699.00
Quantity
7
Update
Cancel


ProductInfoManager
Home
Privacy
Edit Product
Name
Iphone
Category
Mobile
Price
90000.00
Quantity
17
Update
Cancel

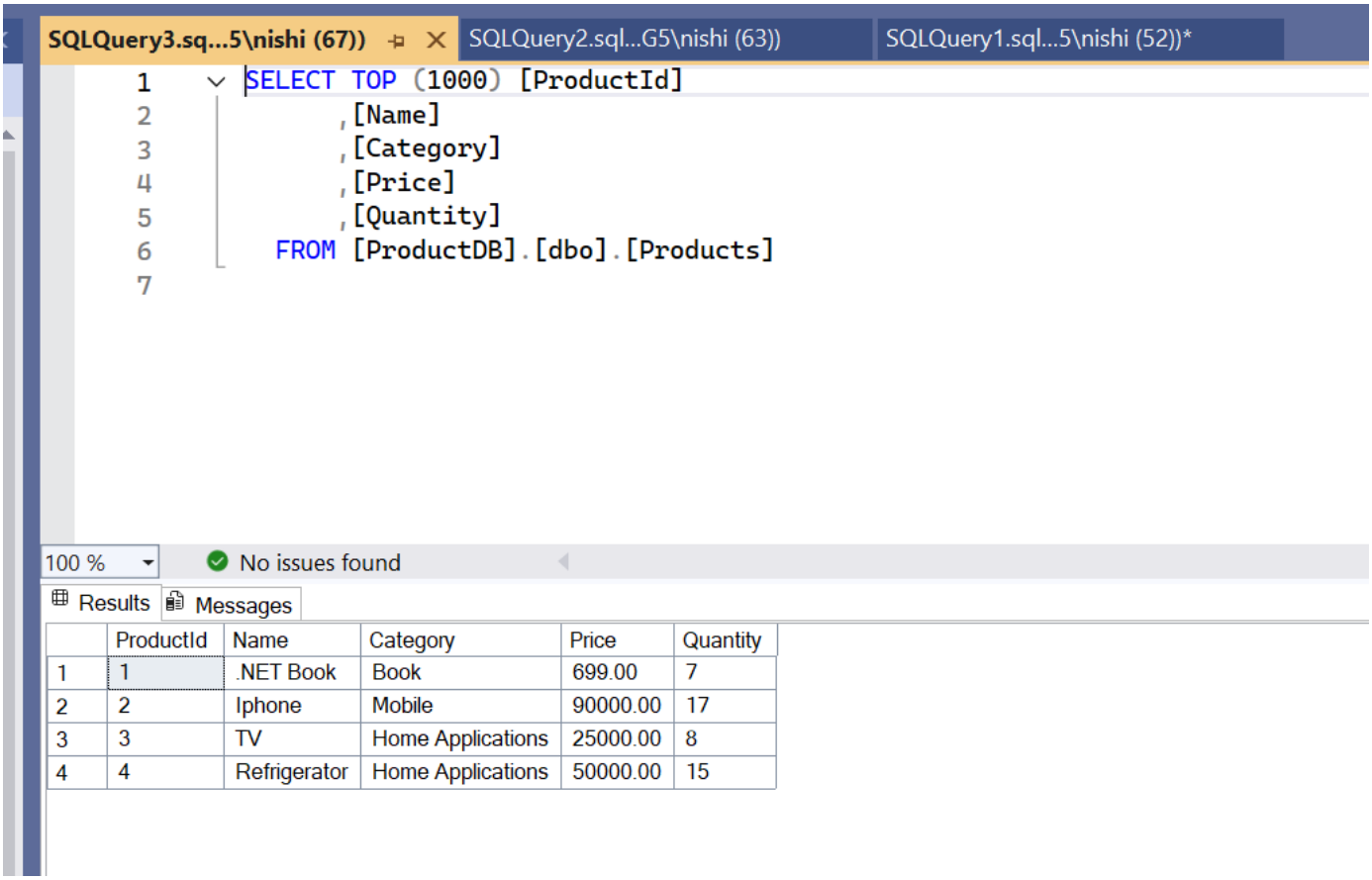

ProductInfoManager
Home
Privacy
Product List
Add New Product
Name
Category
Price
Quantity
Actions
.NET Book
Book
699.00
7
Details
Edit
Delete
Iphone
Mobile
90000.00
17
Details
Edit
Delete
TV
Home Applications
25000.00
8
Details
Edit
Delete
Refrigerator
Home Applications
50000.00
15
Details
Edit
Delete

Delete:


ProductInfoManager
Home
Privacy
Delete Product
Are you sure you want to delete this product?
.NET Book
Category
Book
Price
699.00
Quantity
7
Delete
Cancel



Database:



Conclusion:

In this practical, we successfully built a **Product Information Manager** using **ASP.NET Core MVC** and **Entity Framework Core** with a SQL Server database. The application follows the **MVC architecture**:

- **Model**: Defines the product structure (Product.cs) and validation rules.

- **View**: Provides user interfaces (Razor pages) for creating, reading, updating, and deleting products.
- **Controller**: Handles CRUD operations and communicates with the database through ProductDbContext.

Using **Entity Framework Core**, we were able to:

- Connect to a **SQL Server database**.
- Perform **migrations** to create the Products table.
- Execute **CRUD operations** with data persisting in the database.

This practical demonstrates how MVC architecture separates concerns, and how EF Core simplifies data interaction, making web application development faster and more maintainable.

PRACTICAL: 9

AIM:

LINQ Operations – Querying and Manipulating In-Memory Collections

Scenario :

You are working as a software developer for a data analytics company. Your manager has asked you to create a prototype that can perform various operations on in-memory collections (like lists of products or employees) to filter, sort, group, and summarize the data.

Instead of writing traditional loops and conditionals, you are required to use LINQ (Language Integrated Query) to query and manipulate data using clean, concise, and readable syntax.

This practical will help you understand how LINQ works with in-memory collections like List<T> using both query syntax and method syntax.

Tasks:

● Task 1: Create a New Console Application in C# Named LinqOperationsDemo

● Task 2: Define a Class Product with the appropriate Properties: (Product ID, Name, Category, Price, Stock Quantity)

● Task 3: Create a List of Products and Populate it with at Least 6 Sample Records

● Task 4: Perform the Following LINQ Operations Using Method Syntax:

- Display all products in the list
- Filter products where Price > 500
- Get products that belong to a specific Category (e.g.,
- "Electronics")
- Sort products by Name in ascending order
- Calculate the total stock quantity of all products
- Find the most expensive product

● Task 5: Perform the Following LINQ Operations Using Query Syntax:

- Select only product names from the list
- Group products by Category
- Count the number of products in each category
- Find average price of all products

● Task 6: Display the Output of Each Query in the Console with Proper Labels

CODE:

Product.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LinqOperationsDemo
{
   public class Product
   {
      public int ProductID { get; set; }
      public string Name { get; set; }
      public string Category { get; set; }
      public double Price { get; set; }
      public int StockQuantity { get; set; }
   }
}
```

Program.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace LinqOperationsDemo
{

   class Program
   {
      static void Main(string[] args)
      {
         // Step 3: Create sample product list
         List<Product> products = new List<Product>()
         {
            new Product { ProductID = 1, Name = "Laptop", Category = "Electronics", Price
= 60000, StockQuantity = 10 },
            new Product { ProductID = 2, Name = "Headphones", Category = "Electronics",
Price = 1500, StockQuantity = 50 },
            new Product { ProductID = 3, Name = "Refrigerator", Category = "Appliances",
Price = 25000, StockQuantity = 5 },
            new Product { ProductID = 4, Name = "Shoes", Category = "Fashion", Price =
2000, StockQuantity = 20 },
            new Product { ProductID = 5, Name = "Mobile Phone", Category =
"Electronics", Price = 30000, StockQuantity = 15 },
```

```
            new Product { ProductID = 6, Name = "Washing Machine", Category =
"Appliances", Price = 18000, StockQuantity = 8 }
        };

        // ---------- Step 4: Method Syntax ----------
        Console.WriteLine("=== Method Syntax Queries ===");

        // 1. Display all products in table format
        Console.WriteLine("\nAll Products:");
        Console.WriteLine("----------------------------------------");
        Console.WriteLine($"{"ID",-5} {"Name",-20} {"Category",-15} {"Price",-10}
{"Stock",-6}");
        Console.WriteLine("----------------------------------------");
        products.ForEach(p =>
            Console.WriteLine($"{p.ProductID,-5} {p.Name,-20} {p.Category,-15}
{p.Price,-10} {p.StockQuantity,-6}")
        );

        // 2. Filter products where Price > 500


        var expensiveProducts = products.Where(p => p.Price > 500);
        Console.WriteLine("\nProducts with Price > 500:");
        Console.WriteLine("--------------------");
        Console.WriteLine($"{"Name",-20} {"Price",-10}");
        Console.WriteLine("--------------------");
        foreach (var p in expensiveProducts)
            Console.WriteLine($"{p.Name,-20} {p.Price,-10}");


        // 3. Get products from Electronics category
        var electronics = products.Where(p => p.Category == "Electronics");
        Console.WriteLine("\nElectronics Products:");
        Console.WriteLine("--------------------");
        Console.WriteLine($"{"Name",-20} {"Price",-10}");
        Console.WriteLine("--------------------");
        foreach (var p in electronics)
            Console.WriteLine($"{p.Name,-20} {p.Price,-10}");

        // 4. Sort by name
        var sortedByName = products.OrderBy(p => p.Name);
        Console.WriteLine("\nProducts Sorted by Name:");
        Console.WriteLine("------------------------ ");
        foreach (var p in sortedByName)
            Console.WriteLine(p.Name);

        // 5. Total stock
        int totalStock = products.Sum(p => p.StockQuantity);
        Console.WriteLine($"\nTotal Stock Quantity: {totalStock}");
```

```
        // 6. Most expensive
        var mostExpensive = products.OrderByDescending(p => p.Price).First();
        Console.WriteLine($"\nMost Expensive Product: {mostExpensive.Name} -
{mostExpensive.Price}");

        // ---------- Step 5: Query Syntax ----------
        Console.WriteLine("\n=== Query Syntax Queries ===");

        // 1. Select only product names
        var productNames = from p in products
                           select p.Name;
        Console.WriteLine("\nProduct Names:");
        Console.WriteLine("---------- ");
        foreach (var name in productNames)
            Console.WriteLine(name);

        // 2. Group products by category
        var groupedByCategory = from p in products
                                group p by p.Category;
        Console.WriteLine("\nProducts Grouped by Category:");
        foreach (var group in groupedByCategory)
        {
            Console.WriteLine($"\nCategory: {group.Key}");
            Console.WriteLine("----------");
            Console.WriteLine($"{"Name",-20} {"Price",-10}");
            foreach (var p in group)
                Console.WriteLine($"{p.Name,-20} {p.Price,-10}");
        }

        // 3. Count products in each category
        var categoryCounts = from p in products
                             group p by p.Category into g
                             select new { Category = g.Key, Count = g.Count() };
        Console.WriteLine("\nNumber of Products in Each Category:");
        Console.WriteLine("----------");
        foreach (var c in categoryCounts)
            Console.WriteLine($"{c.Category,-15} {c.Count}");

        // 4. Average price
        var avgPrice = (from p in products select p.Price).Average();
        Console.WriteLine($"\nAverage Price of Products: {avgPrice}");
    }
  }
}
```

OUTPUT:

```
=== LINQ Method Syntax Demo ===

All Products:
Laptop - Electronics - 800 Rupees - Stock: 10
Smartphone - Electronics - 600 Rupees - Stock: 25
Desk Chair - Furniture - 150 Rupees - Stock: 5
Headphones - Electronics - 120 Rupees - Stock: 30
Coffee Table - Furniture - 200 Rupees - Stock: 7
Shoes - Clothing - 90 Rupees - Stock: 50

Products with Price > 500:
Laptop - 800 Rupees
Smartphone - 600 Rupees

Electronics Products:
Laptop
Smartphone
Headphones

Products Sorted by Name:
Coffee Table
Desk Chair
Headphones
Laptop
Shoes
Smartphone

Total Stock Quantity: 127
```

```
Most Expensive Product: Laptop - 800 Rupees

=== LINQ Query Syntax Demo ===

Product Names:
Laptop
Smartphone
Desk Chair
Headphones
Coffee Table
Shoes

Products Grouped by Category:
Electronics
  Laptop
  Smartphone
  Headphones
Furniture
  Desk Chair
  Coffee Table
Clothing
  Shoes

Product Count by Category:
Electronics: 3
Furniture: 2
Clothing: 1

Average Price of Products: 326.6666666666667 Rupees
```

```
Product Count by Category:
Electronics: 3
Furniture: 2
Clothing: 1

Average Price of Products: 326.6666666666667 Rupees

--------------------------------
Program by: Dhruvi
ID: 23DCS029
--------------------------------

C:\Users\DHRUVI\Desktop\.NET\LinqOperationsDemo\bin\Debug\net8.0\LinqOperationsDemo.exe (process 16720) exited with code
 0 (0x0).
Press any key to close this window . . .
```

Conclusion:

The project demonstrated how LINQ simplifies querying and manipulating in-memory collections. Using both method and query syntax, it efficiently filtered, sorted, grouped, and summarized product data with clean and readable code.