## Experiment Number: Date of Experiment:

**Aim:**

## Create a Asp.Net Web Pages to mimic the College Website Static Pages and Layout

## Description:

## Steps to Create a Asp.Net Web Pages

## Step 1: Install the ASP.NET Web Application (.NET Framework) Component

* Open Visual Studio.
* If you see "Not finding what you’re looking for?", click on Install more tools and features.
* The Visual Studio Installer window will open. Go to the Individual Components tab.
* In the search bar, type ASP.NET Web Application (.NET Framework).
* Check the box next to ASP.NET and web development (or only ASP.NET Web Application (.NET Framework) if it appears separately).
* Click Modify to install the selected components. Wait for the installation to complete.

Step 2: Create a New ASP.NET Web Application Project

* After installation, open Visual Studio if it's not already open.
* Go to File > New > Project.
* In the Create a new project window:
  1. Type ASP.NET Web Application (.NET Framework) in the search box.
  2. Select ASP.NET Web Application (.NET Framework) from the list.
  3. Click Next.
* Configure your project:
  1. Enter a Project Name (e.g., "CollegeWebsite").
  2. Choose the Location where you want to save the project.
  3. Click Create.
* In the New ASP.NET Web Application window:
  1. Choose Web Forms template (choose Empty for a static HTML layout).
  2. Click Create.

Step 3: Open Site.Master

* Find Site.Master in your project files.
* Update the <title> to "VVIT College Website."
* Modify the navigation bar to include links to Default.aspx, About.aspx, Contact.aspx, Admissions.aspx, and Departments.aspx.
* Ensure there's a <ContentPlaceHolder> with ID="MainContent" for page-specific content.

Step 4: Edit Default.aspx

* Locate Default.aspx as your home page file.
* Replace the placeholder text with a welcome message and a college introduction.
* Wrap content in Bootstrap container and card components for a framed look.
* Add any other introductory text or images as desired.

Step 5: Modify About.aspx

* Find About.aspx in the root of your project.
* Add information about the college’s background and vision.
* Use container and card components to match other pages.

Step 6: Modify Contact.aspx

* Find Contact.aspx in the root of your project.
* Add actual contact details like email, phone, and address.
* Use Bootstrap container and card classes to frame the content.

Step 7: Add New Pages (e.g., Admissions.aspx and Departments.aspx)

* Right-click on the project, select Add > New Web Form, choose a name (e.g., Admissions.aspx), and select Site.Master as the master page.
* Repeat for other pages like Departments.aspx.
* For each new page, add relevant content within Bootstrap container and card components to keep a consistent layout across the website.

Step 8: Style Adjustments and Fine-Tuning

* Adjust padding, margins, and font styles if needed to improve the look and feel.
* Add any additional styling to Site.css (if you have one) or within <style> tags in Site.Master.

Step 9: Test the Project

* Run the project by pressing **F5** or clicking **Start** in Visual Studio.

## Program:

## Site.Master

## <%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site.master.cs" Inherits="lab5b.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 %> - VVIT College Website</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>

## <%--To learn more about bundling scripts in ScriptManager see https://go.microsoft.com/fwlink/?LinkID=301884 --%>

## <%--Framework 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" />

## <%--Site Scripts--%>

## </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="~/">VVIT</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="~/">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="~/Admissions">Admissions</a></li>

## <li class="nav-item"><a class="nav-link" runat="server" href="~/Departments">Departments</a></li>

## <li class="nav-item"><a class="nav-link" runat="server" href="~/Contact">Contact</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 %> - VVIT College Website</p>

## </footer>

## </div>

## </form>

## <asp:PlaceHolder runat="server">

## <%: Scripts.Render("~/Scripts/bootstrap.js") %>

## </asp:PlaceHolder>

## </body>

## </html>

**Default.aspx**

<%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="lab5b.\_Default" %>

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

<!-- Content Section -->

<div class="container mt-5">

<div class="card">

<div class="card-body">

<h2 class="display-4">Welcome to VVIT</h2>

<p class="lead">

This is a sample homepage for our college's website. Here, you can find information about our college, admissions process, courses, and more.

</p>

<p>

Feel free to navigate through the different sections to learn more about what we offer and how to get in touch with us.

</p>

</div>

</div>

</div>

</asp:Content>

**About.aspx**

<%@ Page Title="About" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeBehind="About.aspx.cs" Inherits="lab5b.About" %>

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

<!-- About Page Content -->

<div class="container mt-4">

<div class="card">

<div class="card-body">

<h2 class="display-4">About VVIT</h2>

<p class="lead">

Vasireddy Venkatadri Institute of Technology (VVIT) was established in the year 2007, with an intake of 240 students in four B. Tech programs under Social Educational Trust in Nambur village, Guntur, AP, by Er. Vasireddy Vidya Sagar. It is located strategically between Guntur and Vijayawada in the capital region of Amaravati, AP. In a short span of 15 years, VVIT has grown to accommodate over 6,000 students with a large faculty team striving to fulfill the vision of VVIT.

</p>

</div>

</div>

</div>

</asp:Content>

**Contact.aspx**

<%@ Page Title="Contact" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeBehind="Contact.aspx.cs" Inherits="lab5b.Contact" %>

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

<!-- Contact Page Content -->

<div class="container mt-4">

<div class="card">

<div class="card-body">

<h2 class="display-4">Contact Us</h2>

<p><strong>Email:</strong> contact@vvit.edu</p>

<p><strong>Phone:</strong> +1 (123) 456-7890</p>

<address>

<strong>Address:</strong> Vasireddy Venkatadri Institute of Technology<br>

Nambur (V), Peda Kakani (Md), Guntur (Dt), Andhra Pradesh, 522508

</address>

</div>

</div>

</div>

</asp:Content>

**Admissions.aspx**

<%@ Page Title="Admissions" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeBehind="Admissions.aspx.cs" Inherits="lab5b.Admissions" %>

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

<!-- Admissions Page Content -->

<div class="container mt-4">

<div class="card">

<div class="card-body">

<h2 class="display-4">Admissions</h2>

<p class="lead">

Join VVIT and become part of a thriving academic community. For more details on application deadlines, eligibility, and scholarship options, please contact our admissions office.

</p>

</div>

</div>

</div>

</asp:Content>

**Departments.aspx**

<%@ Page Title="Departments" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeBehind="Departments.aspx.cs" Inherits="lab5b.Departments" %>

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

<!-- Departments Page Content -->

<div class="container mt-4">

<div class="card">

<div class="card-body">

<h2 class="display-4">Departments at VVIT</h2>

<p class="lead">

Vasireddy Venkatadri Institute of Technology (VVIT) offers a variety of undergraduate and postgraduate programs. Here are the departments:

</p>

<h3>Undergraduate Programs (B.Tech)</h3>

<ul>

<li><strong>Civil Engineering (CE)</strong> - Focuses on sustainable infrastructure.</li>

<li><strong>Electrical and Electronics Engineering (EEE)</strong> - Covers power generation and distribution.</li>

<li><strong>Mechanical Engineering (ME)</strong> - Involves design and manufacturing.</li>

<li><strong>Electronics and Communication Engineering (ECE)</strong> - Specializes in telecommunications.</li>

<li><strong>Computer Science and Engineering (CSE)</strong> - Emphasizes programming and algorithms.</li>

<li><strong>Information Technology (IT)</strong> - Focuses on data management and cybersecurity.</li>

<!-- Add additional departments here -->

</ul>

<h3>Postgraduate Programs (M.Tech)</h3>

<ul>

<li><strong>Computer Science and Engineering (CSE)</strong></li>

<li><strong>VLSI and Embedded Systems (VLSI&ES)</strong></li>

<li><strong>Power Electronics and Electric Drives (PEED)</strong></li>

<li><strong>Machine Design (MD)</strong></li>

<li><strong>Software Engineering (SE)</strong></li>

<!-- Add additional programs here -->

</ul>

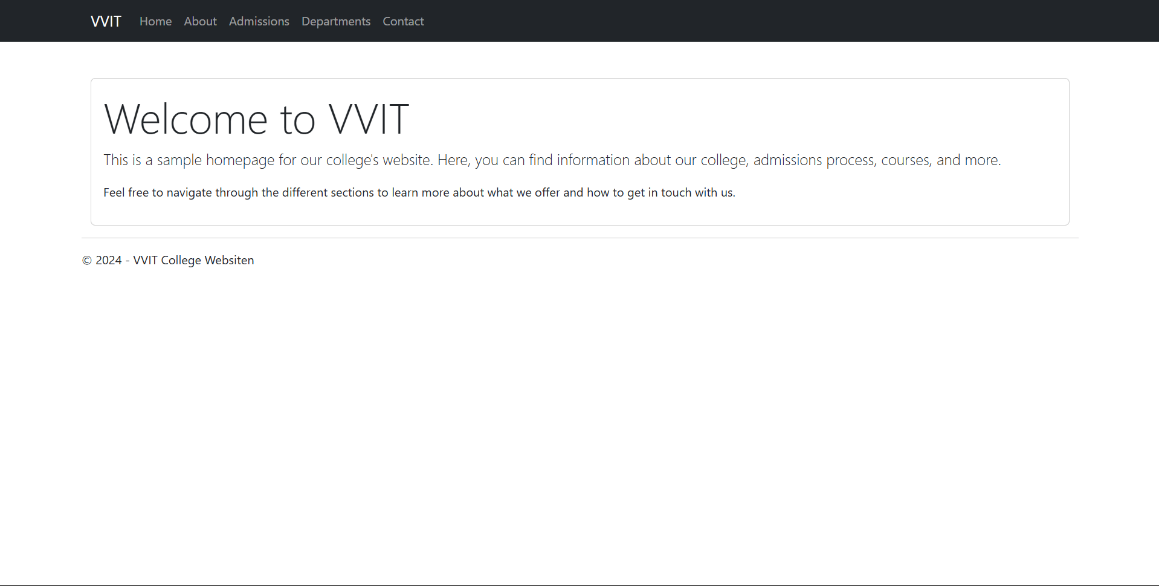
</div>

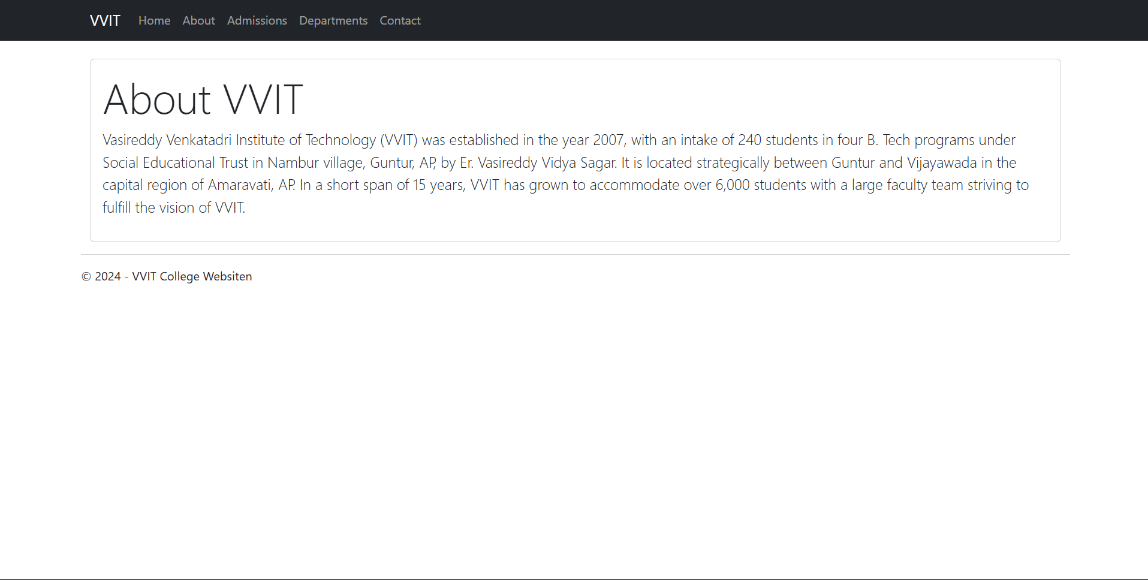
</div>

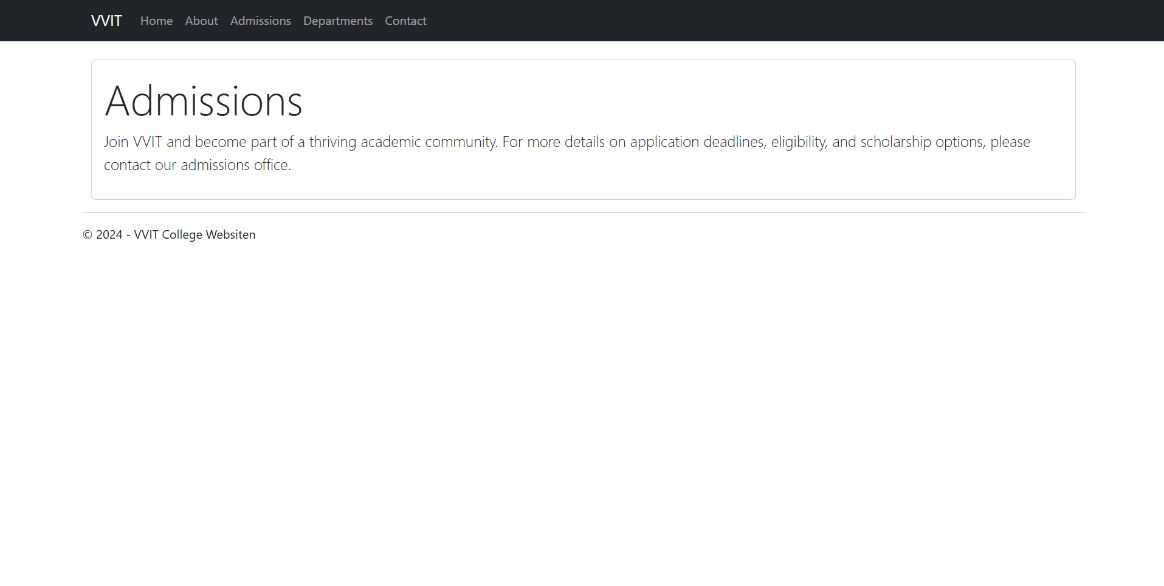
</div>

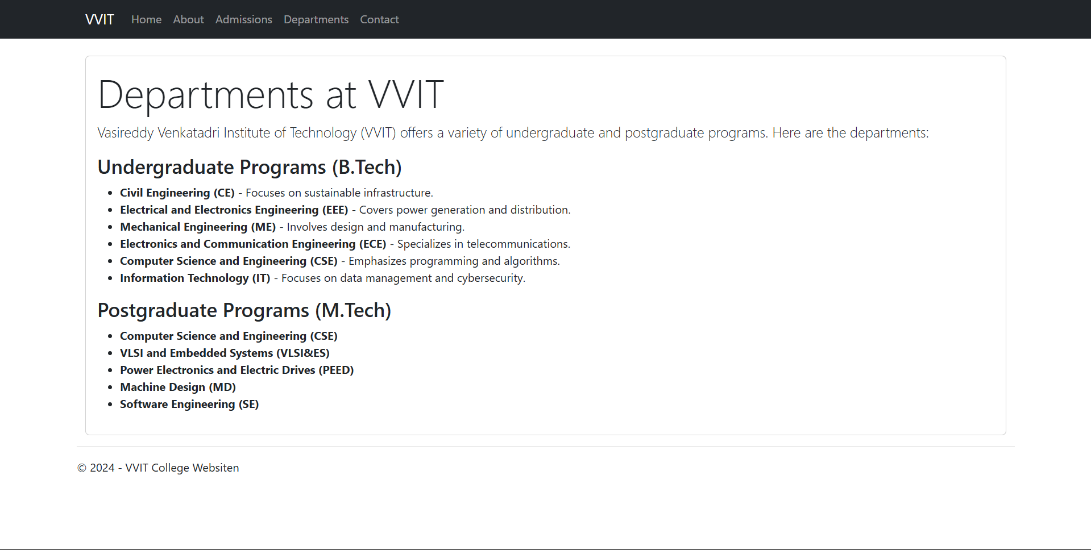
</asp:Content>

## Output:

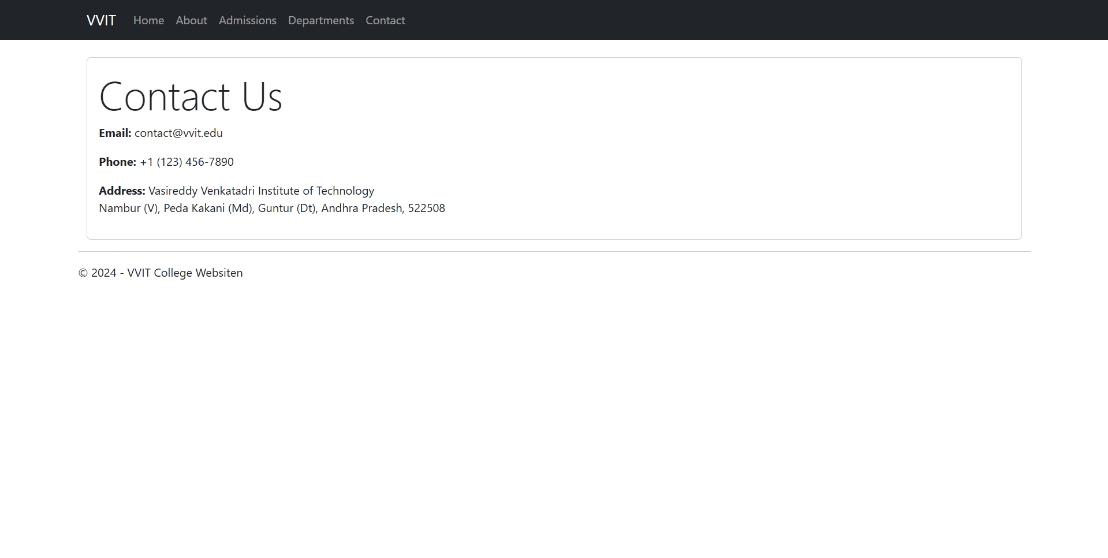
**** **Welcome Page**

**About Page**

**Admissions Page**

**Departments Page**

**Contact Page**



## Experiment Number:

## Date of Experiment:

**Aim:**

Create a Asp.Net Web Application using MVC Architecture with Student, Faculty Registration Forms

**Description:**

The goal of this project is to create a registration web application using ASP.NET MVC, where users can register under different categories (e.g., Student, Faculty, SAC). The application consists of a navigation menu that allows users to switch between these categories and provides a form for each category to capture registration details. This project is built with a structured layout, user-friendly design, and some interactive features for a smooth user experience.

**Step 1: Setting up the ASP.NET MVC Project**

1. Open Visual Studio: Start by opening Visual Studio (preferably Visual Studio 2022 or later).
2. Create a New Project:
   * Select ASP.NET Core Web Application from the project templates.
   * Name the project something like RegistrationApp.
3. Configure the Project:
   * Choose ASP.NET Core 6.0 (or higher) as the framework.
   * Select Web Application (Model-View-Controller) as the project template.
   * Click Create to set up the project.

The project structure will be automatically generated, including folders for Controllers, Views, Models, and wwwroot (for static assets like CSS, JavaScript, and images).

**Step 2: Create Models for Registration Categories**

The **Model** represents the data structure of our application. Here, we’ll define models for each registration category (Student, Faculty, SAC).

**Explanation**:

* The Faculty class represents a model for the faculty registration form.
* DataAnnotations are used for validation. For instance:
  + [Required] ensures the field is not left empty.
  + [EmailAddress] validates the email format.
  + [MinLength(6)] ensures the password is at least 6 characters long.

Repeat a similar process to create models for **Student** and **SAC** registrations

**Step 3: Create Controllers for Each Category**

Controllers handle user requests, process data, and select views to render responses.

**Explanation**:

* The FacultyController handles the faculty registration process.
* The Register() method (GET) loads the registration form.
* The Register(Faculty faculty) method (POST) processes the form data.
* ModelState.IsValid checks if the submitted data meets validation rules (defined in the Faculty model).
* ViewBag.Message is used to display a success message if the form is valid.

**Step 4: Create Views for the Registration Forms**

Each view represents the UI for a specific part of the application. Here, we’ll create a view for the Faculty registration form.

1. In the **Views** folder, create a folder named **Faculty**.
2. Inside the **Faculty** folder, add a new Razor view named Register.cshtml.

**Explanation**:

* @model specifies the data model (Faculty) used in this view.
* The form fields (asp-for) are bound to model properties. Validation messages (asp-validation-for) display error messages if validation fails.
* @ViewBag.Message displays a success message after form submission.
* @section Scripts loads client-side validation scripts.

**Step 5: Styling the Application with CSS**

In the **wwwroot/css** folder, create a CSS file (site.css) and add the following code to style the registration form and layout.

**Step 6: Add JavaScript for Interactive Alerts**

In the **wwwroot/js** folder, create a JavaScript file (site.js) with the following code for navigation link interactions.

**Explanation**:

* The script adds event listeners to each link in the navigation menu.
* When a link is clicked, an alert message is displayed to provide feedback.

**Programs:  
Example Model for Faculty Registration (Faculty.cs in the Models folder)**

using System.ComponentModel.DataAnnotations;

namespace RegistrationApp.Models

{

public class Faculty

{

[Required]

public string Name { get; set; }

[Required]

public string FacultyID { get; set; }

[Required]

public string Department { get; set;

[Required, EmailAddress]

public string Email { get; set; }

[Required, MinLength(6)]

public string Password { get; set; }

}

}

**Example FacultyController:**

using Microsoft.AspNetCore.Mvc;

using RegistrationApp.Models;

namespace RegistrationApp.Controllers

{

public class FacultyController : Controller

{

// GET: Faculty/Register

public IActionResult Register()

{

return View();

}

// POST: Faculty/Register

[HttpPost]

public IActionResult Register(Faculty faculty)

{

if (ModelState.IsValid)

{

ViewBag.Message = "Faculty Registered Successfully!";

// Here, you would typically save data to the database

}

return View();

}

}

}

**Example Faculty Registration View (Register.cshtml)**

@model RegistrationApp.Models.Faculty

<h2>Faculty Registration</h2>

@if (!string.IsNullOrEmpty(ViewBag.Message))

{

<div class="success-message">@ViewBag.Message</div>

}

<form asp-action="Register" method="post">

<div class="form-group">

<label for="faculty-name">Name</label>

<input asp-for="Name" class="form-control" placeholder="Enter name" />

<span asp-validation-for="Name" class="text-danger"></span>

</div>

<div class="form-group">

<label for="faculty-id">Faculty ID</label>

<input asp-for="FacultyID" class="form-control" placeholder="Enter Faculty ID" />

<span asp-validation-for="FacultyID" class="text-danger"></span>

</div>

<div class="form-group">

<label for="department">Department</label>

<input asp-for="Department" class="form-control" placeholder="Enter Department" />

<span asp-validation-for="Department" class="text-danger"></span>

</div>

<div class="form-group">

<label for="email">Email</label>

<input asp-for="Email" class="form-control" placeholder="Enter Email" />

<span asp-validation-for="Email" class="text-danger"></span>

</div>

<div class="form-group">

<label for="password">Password</label>

<input asp-for="Password" class="form-control" placeholder="Enter Password" />

<span asp-validation-for="Password" class="text-danger"></span>

</div>

<button type="submit" class="btn btn-primary">Register</button>

</form>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

**Site.css:**/\* Basic Styling for Body \*/

body {

font-family: Arial, sans-serif;

background-color: #f9fafb;

color: #333;

margin: 0;

padding: 0;

}

/\* Container for the Header \*/

.top-container {

display: flex;

background-color: #3a3f51;

color: #fff;

padding: 20px;

}

/\* Main Content Container \*/

.bottom-container {

padding: 20px;

background-color: #ffffff;

display: flex;

justify-content: center;

align-items: flex-start;

}

/\* Form Styling \*/

.form-control {

width: 100%;

padding: 10px;

margin: 10px 0;

border-radius: 5px;

border: 1px solid #ccc;

}

/\* Button Styling \*/

button {

background-color: #3949ab;

color: #fff;

padding: 10px 20px;

border: none;

border-radius: 5px;

cursor: pointer;

transition: background-color 0.3s;

}

button:hover {

background-color: #283593;

}

/\* Success Message \*/

.success-message {

color: #28a745;

padding: 10px;

border: 1px solid #28a745;

background-color: #e6ffe6;

border-radius: 5px;

}

**Site.js:**

document.addEventListener("DOMContentLoaded", function () {

document.getElementById("home-link").addEventListener("click", function () {

alert("Welcome to the Home page!");

});

document.getElementById("student-link").addEventListener("click", function () {

alert("Register as a Student!");

});

document.getElementById("faculty-link").addEventListener("click", function () {

alert("Register as Faculty!");

});

document.getElementById("sac-link").addEventListener("click", function () {

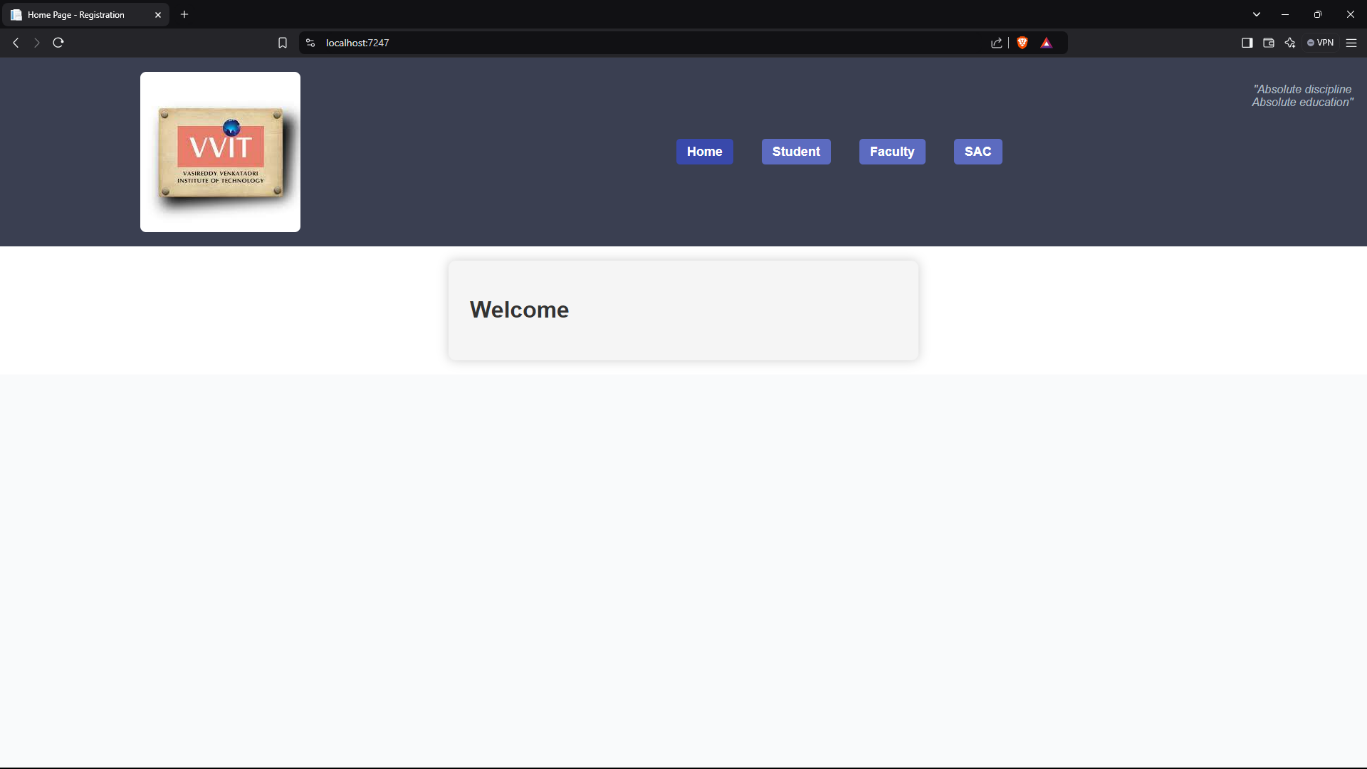
alert("Register as SAC!");

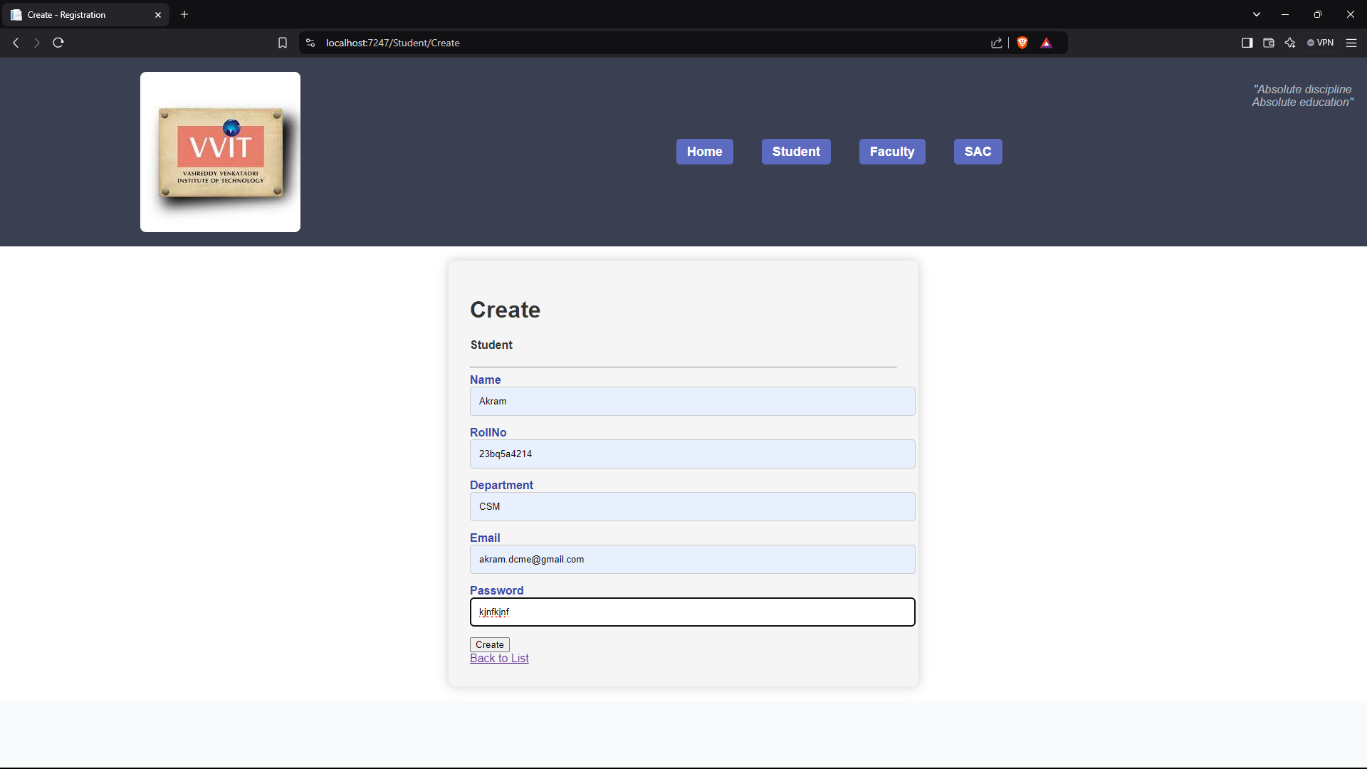
});

});

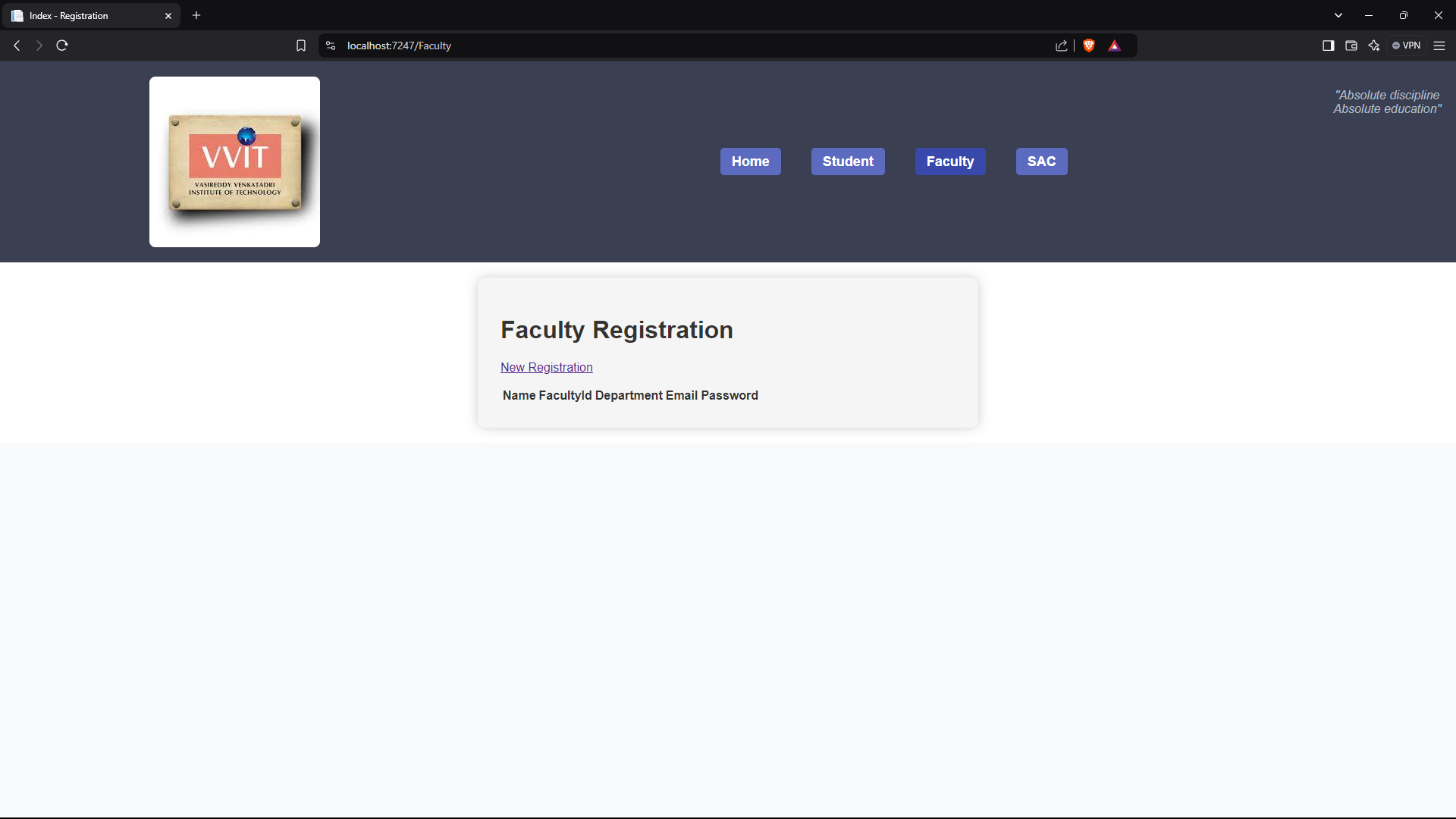
### **Output**

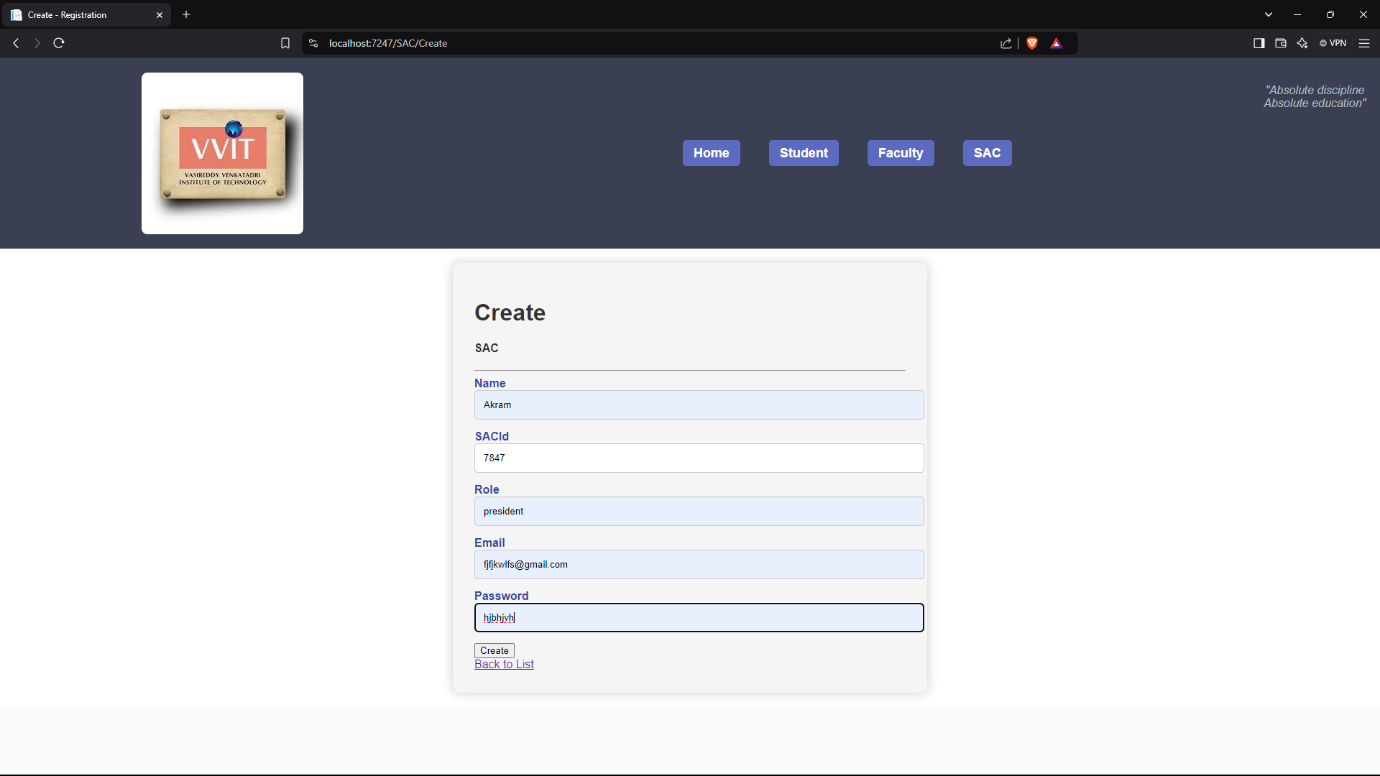
This ASP.NET MVC application demonstrates a structured way to build a registration system with separate forms for different user categories. Each part, from model validation to CSS styling and JavaScript interactivity, contributes to a cohesive and user-friendly web application.

**Home Page :** The home page includes a navigation menu with links to "Home," "Student," "Faculty," and "SAC."

**Student Registration Page**: Displays the top section with the image, menu, and quote. The bottom section includes the "Student Registration" form with fields for student details and a "Register" button.

**Faculty Registration Page**: Similar layout to the student page but tailored for faculty registration, with fields specific to faculty data (like Faculty ID).



**SAC Registration Page**: Displays the top section with the image, menu, and quote. The bottom section includes the "SAC Registration" form with fields for student details and a "Register" button.

## 

## Experiment Number:

## Date of Experiment:

**Aim:**

Create a Asp.Net MVC Web Application Render the Information using POST & GET Method

**Description:**

This experiment demonstrates how to implement a simple ASP.NET MVC web application that utilizes both the GET and POST methods to render and process data. We'll create a registration form where users can submit data, which will then be processed and displayed.

**Step 1: Setting up the ASP.NET MVC Project**

* Open Visual Studio: Start by opening Visual Studio (preferably Visual Studio 2022 or later).
* Create a New Project:
  + Select ASP.NET Core Web Application from the project templates.
  + Name the project something like RegistrationApp.
* Configure the Project:
  + Choose ASP.NET Core 6.0 (or higher) as the framework.
  + Select Web Application (Model-View-Controller) as the project template.
  + Click Create to set up the project.

The project structure will be automatically generated, including folders for Controllers, Views, Models, and wwwroot (for static assets like CSS, JavaScript, and images).

**Step 2: Create Models for Registration Categories**

The **Model** represents the data structure of our application. Here, we’ll define models for each registration category (Student, Faculty, SAC).

**Explanation**:

* The Faculty class represents a model for the faculty registration form.
* DataAnnotations are used for validation. For instance:
  + [Required] ensures the field is not left empty.
  + [EmailAddress] validates the email format.
  + [MinLength(6)] ensures the password is at least 6 characters long.

Repeat a similar process to create models for **Student** and **SAC** registrations

**Step 3: Create Controllers for Each Category**

Controllers handle user requests, process data, and select views to render responses.

**Explanation**:

* The FacultyController handles the faculty registration process.
* The Register() method (GET) loads the registration form.
* The Register(Faculty faculty) method (POST) processes the form data.
* ModelState.IsValid checks if the submitted data meets validation rules (defined in the Faculty model).
* ViewBag.Message is used to display a success message if the form is valid.

**Step 4: Create Views for the Registration Forms**

Each view represents the UI for a specific part of the application. Here, we’ll create a view for the Faculty registration form.

1. In the **Views** folder, create a folder named **Faculty**.
2. Inside the **Faculty** folder, add a new Razor view named Register.cshtml.

**Explanation**:

* @model specifies the data model (Faculty) used in this view.
* The form fields (asp-for) are bound to model properties. Validation messages (asp-validation-for) display error messages if validation fails.
* @ViewBag.Message displays a success message after form submission.
* @section Scripts loads client-side validation scripts.

**Step 5: Styling the Application with CSS**

In the **wwwroot/css** folder, create a CSS file (site.css) and add the following code to style the registration form and layout.

**Step 6: Add JavaScript for Interactive Alerts**

In the **wwwroot/js** folder, create a JavaScript file (site.js) with the following code for navigation link interactions.

**Explanation**:

The script adds event listeners to each link in the navigation menu.

When a link is clicked, an alert message is displayed to provide feedback.

**Programs:**

**Example Model for Faculty Registration (Faculty.cs in the Models folder)**

using System.ComponentModel.DataAnnotations;

namespace RegistrationApp.Models

{

public class Faculty

{

[Required]

public string Name { get; set; }

[Required]

public string FacultyID { get; set; }

[Required]

public string Department { get; set;

[Required, EmailAddress]

public string Email { get; set; }

[Required, MinLength(6)]

public string Password { get; set; }

}

}

**Example FacultyController:**

using Microsoft.AspNetCore.Mvc;

using RegistrationApp.Models;

namespace RegistrationApp.Controllers

{

public class FacultyController : Controller

{

// GET: Faculty/Register

public IActionResult Register()

{

return View();

}

// POST: Faculty/Register

[HttpPost]

public IActionResult Register(Faculty faculty)

{

if (ModelState.IsValid)

{

ViewBag.Message = "Faculty Registered Successfully!";

// Here, you would typically save data to the database

}

return View();

}

}

}

**Example Faculty Registration View (Register.cshtml)**

@model RegistrationApp.Models.Faculty

<h2>Faculty Registration</h2>

@if (!string.IsNullOrEmpty(ViewBag.Message))

{

<div class="success-message">@ViewBag.Message</div>

}

<form asp-action="Register" method="post">

<div class="form-group">

<label for="faculty-name">Name</label>

<input asp-for="Name" class="form-control" placeholder="Enter name" />

<span asp-validation-for="Name" class="text-danger"></span>

</div>

<div class="form-group">

<label for="faculty-id">Faculty ID</label>

<input asp-for="FacultyID" class="form-control" placeholder="Enter Faculty ID" />

<span asp-validation-for="FacultyID" class="text-danger"></span>

</div>

<div class="form-group">

<label for="department">Department</label>

<input asp-for="Department" class="form-control" placeholder="Enter Department" />

<span asp-validation-for="Department" class="text-danger"></span>

</div>

<div class="form-group">

<label for="email">Email</label>

<input asp-for="Email" class="form-control" placeholder="Enter Email" />

<span asp-validation-for="Email" class="text-danger"></span>

</div>

<div class="form-group">

<label for="password">Password</label>

<input asp-for="Password" class="form-control" placeholder="Enter Password" />

<span asp-validation-for="Password" class="text-danger"></span>

</div>

<button type="submit" class="btn btn-primary">Register</button>

</form>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

**Site.css:**/\* Basic Styling for Body \*/

body {

font-family: Arial, sans-serif;

background-color: #f9fafb;

color: #333;

margin: 0;

padding: 0;

}

/\* Container for the Header \*/

.top-container {

display: flex;

background-color: #3a3f51;

color: #fff;

padding: 20px;

}

/\* Main Content Container \*/

.bottom-container {

padding: 20px;

background-color: #ffffff;

display: flex;

justify-content: center;

align-items: flex-start; }

/\* Form Styling \*/

.form-control {

width: 100%;

padding: 10px;

margin: 10px 0;

border-radius: 5px;

border: 1px solid #ccc;

}

/\* Button Styling \*/

button {

background-color: #3949ab;

color: #fff;

padding: 10px 20px;

border: none;

border-radius: 5px;

cursor: pointer;

transition: background-color 0.3s;

}

button:hover {

background-color: #283593;

}

/\* Success Message \*/

.success-message {

color: #28a745;

padding: 10px;

border: 1px solid #28a745;

background-color: #e6ffe6;

border-radius: 5px;

}

**Site.js:**

document.addEventListener("DOMContentLoaded", function () {

document.getElementById("home-link").addEventListener("click", function () {

alert("Welcome to the Home page!");

});

document.getElementById("student-link").addEventListener("click", function () {

alert("Register as a Student!");

});

document.getElementById("faculty-link").addEventListener("click", function () {

alert("Register as Faculty!");

});

document.getElementById("sac-link").addEventListener("click", function () {

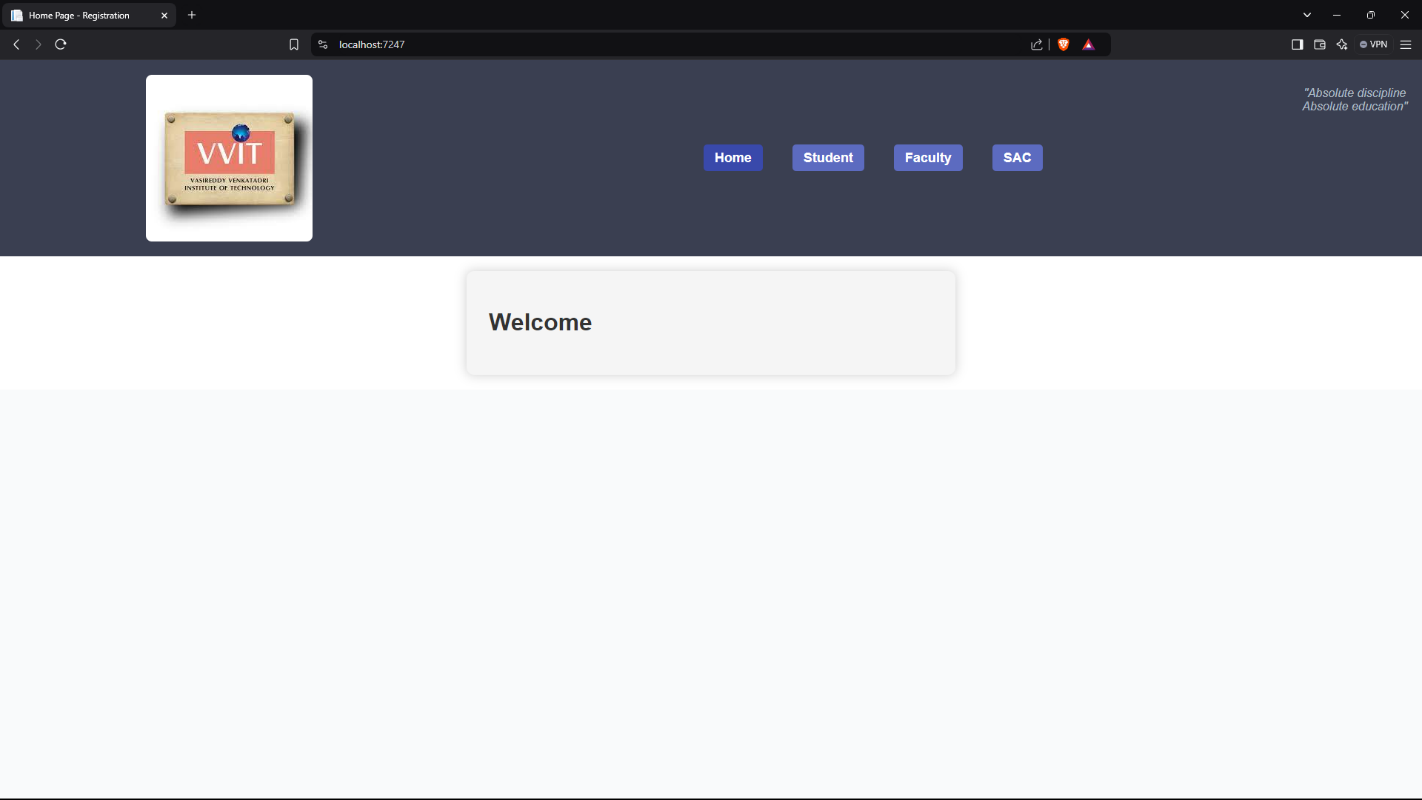
alert("Register as SAC!");

});

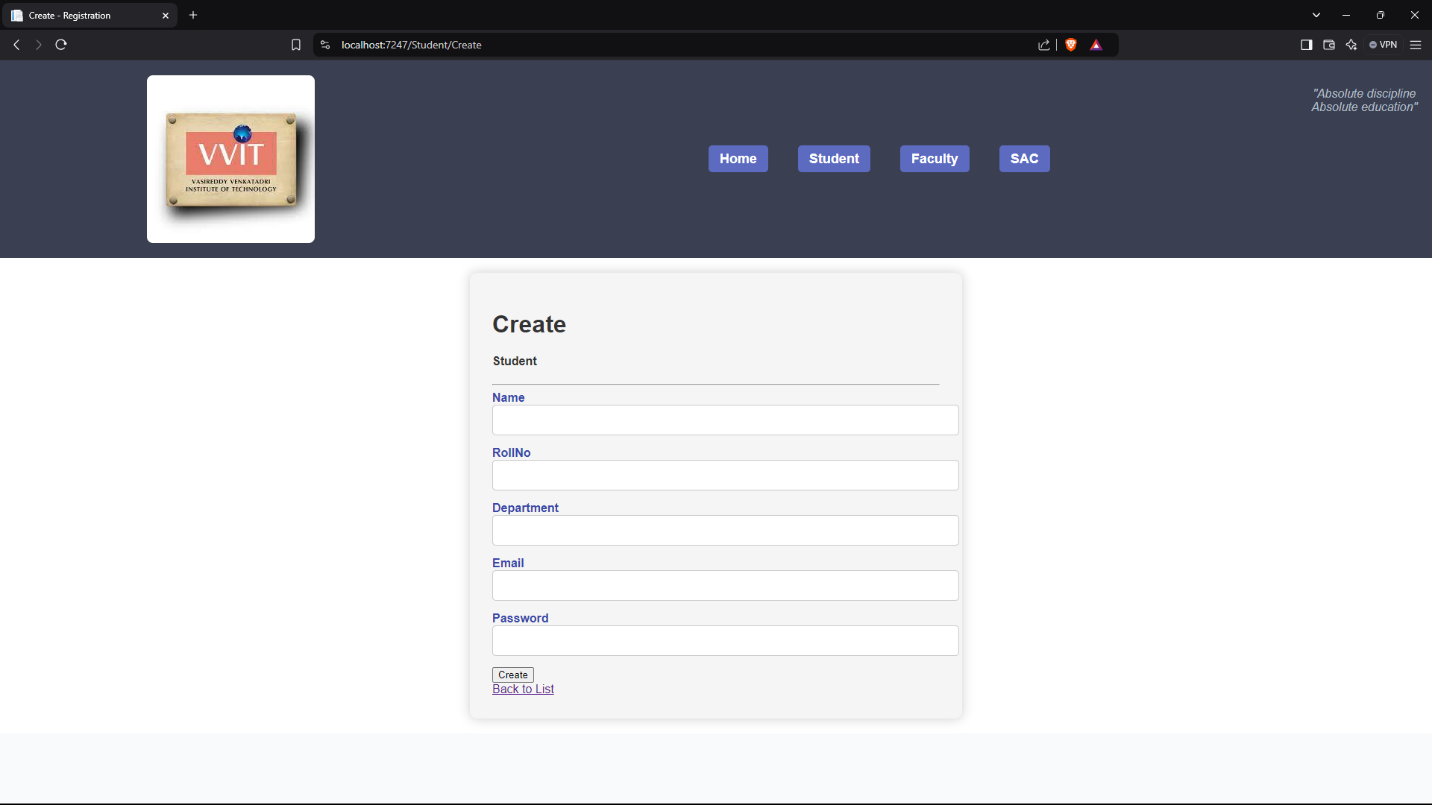
});

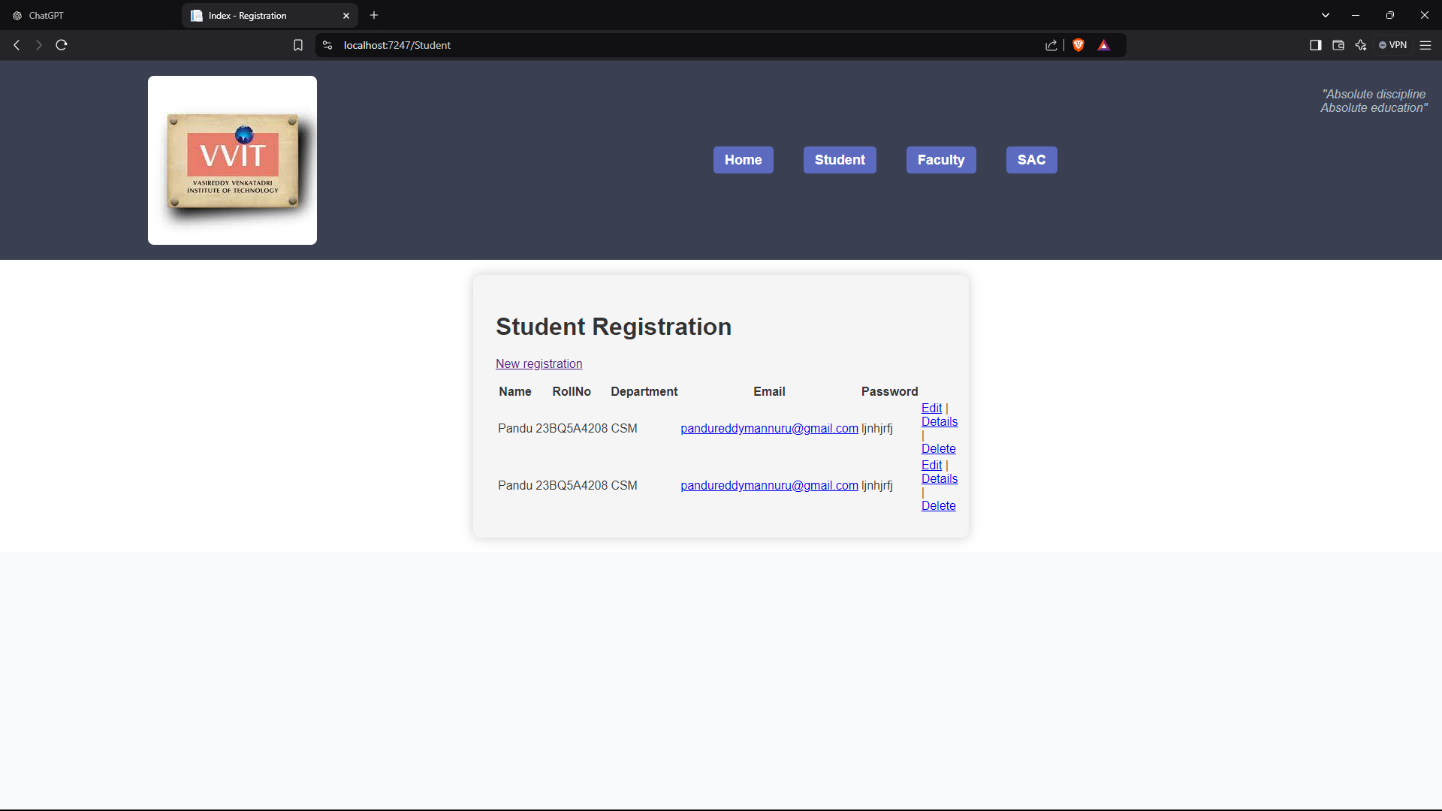
### **Output**

This ASP.NET MVC application demonstrates a structured way to build a registration system with separate forms for different user categories. Each part, from model validation to CSS styling and JavaScript interactivity, contributes to a cohesive and user-friendly web application.

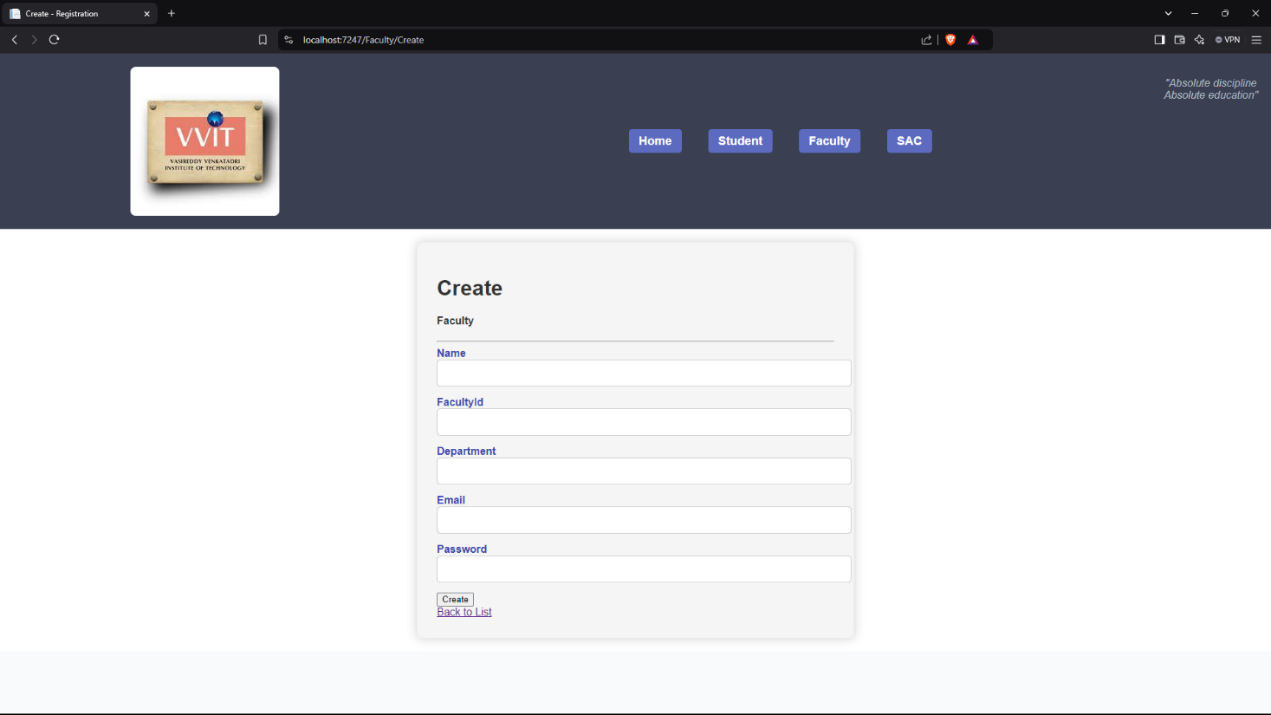
**Home Page :** The home page includes a navigation menu with links to Home, Student, Faculty and SAC

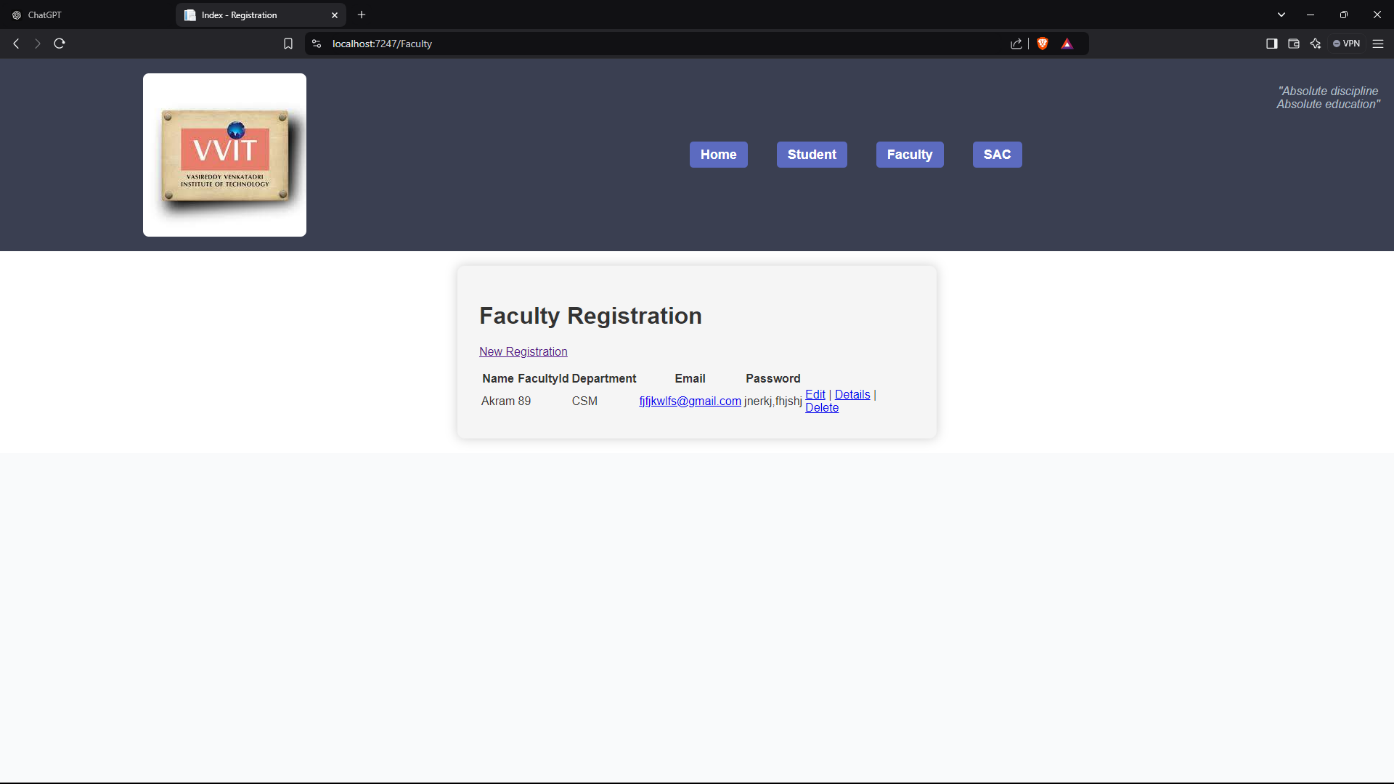
**Student Registration Page**: Displays the top section with the image, menu, and quote. The bottom section includes the "Student Registration" form with fields for student details and a "Register" button.



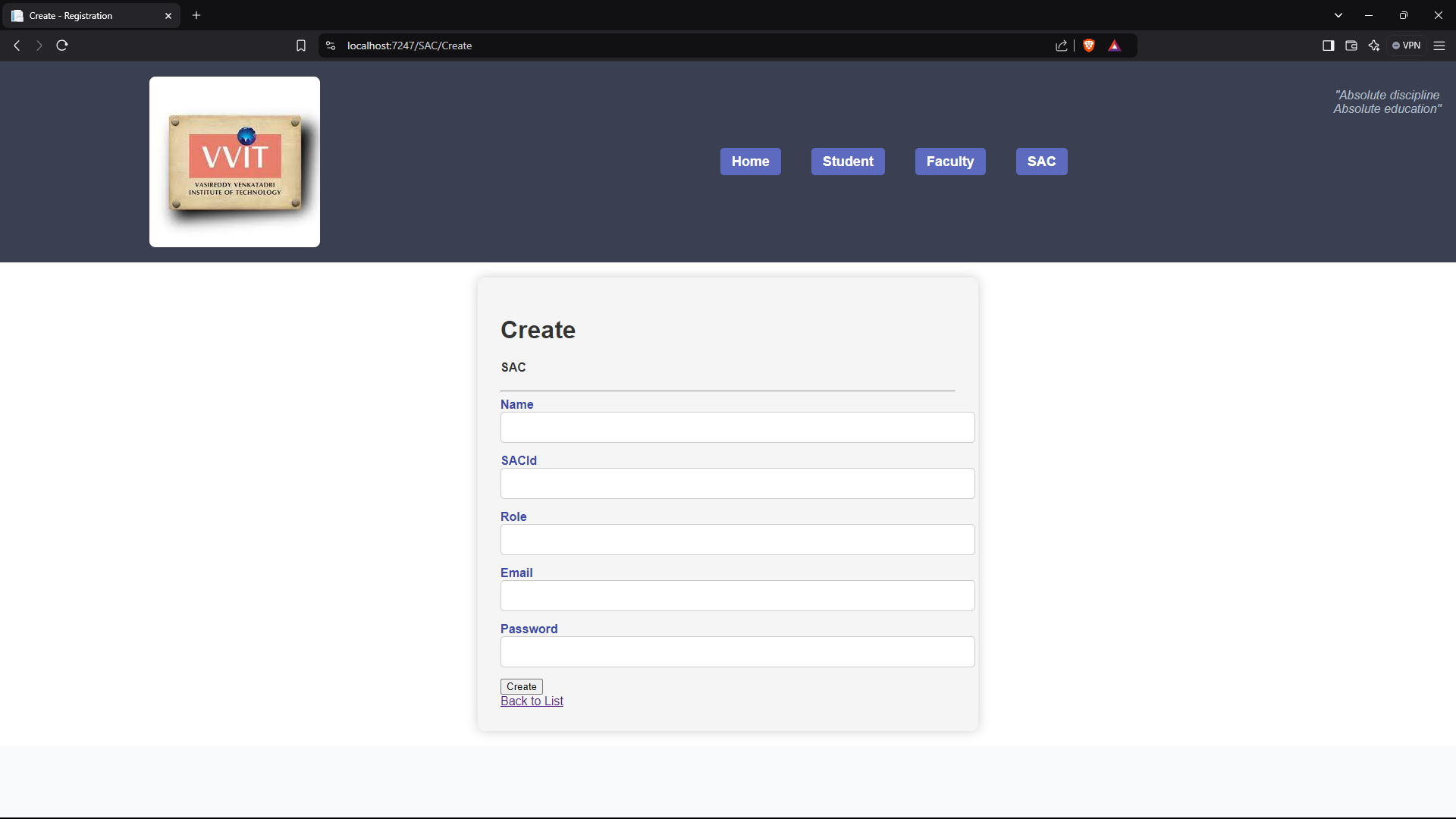
When you create a new registration, the data is automatically rendered and displayed in the list right away. This means that the information gets saved and appears in the table immediately after submitting the form.

**Faculty Registration Page**: Similar layout to the student page but tailored for faculty registration, with fields specific to faculty data (like Faculty ID).

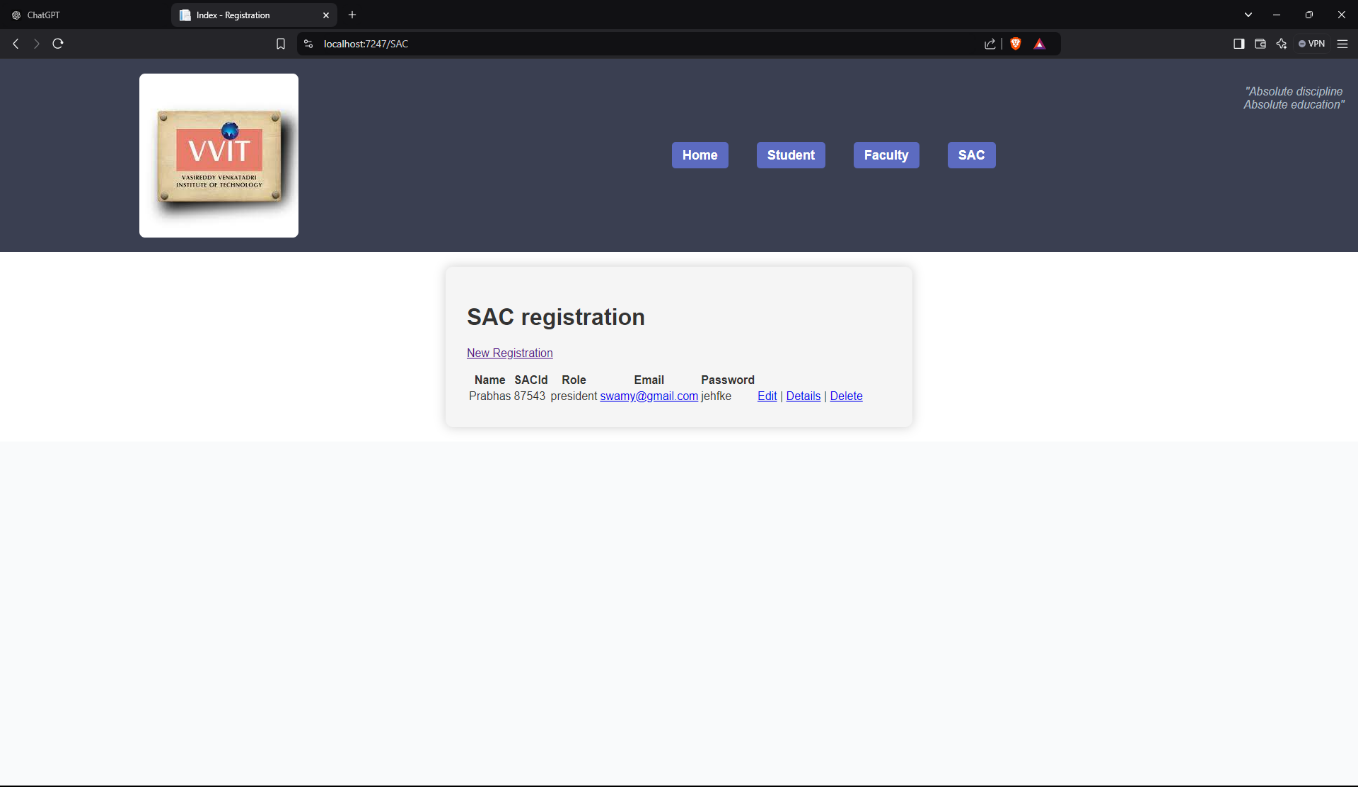


When you create a new registration, the data is automatically rendered and displayed in the list right away. This means that the information gets saved and appears in the table immediately after submitting the form.

**SAC Registration Page**: Displays the top section with the image, menu, and quote. The bottom section includes the "SAC Registration" form with fields for student details and a "Register" button.



When you create a new registration, the data is automatically rendered and displayed in the list right away. This means that the information gets saved and appears in the table immediately after submitting the form.



## Experiment Number:

## Date of Experiment:

**Aim:**

Create a Asp.net MVC Web Application (Front end) Connecting to MySQL Database(BackEnd)

**Description:**

In this experiment, we extend by connecting the ASP.NET MVC application to a MySQL database. By setting up the necessary connection strings, configure Entity Framework Core with MySQL, and save the registration data to the database.

**Step 1: Setting up the ASP.NET MVC Project**

1. Open Visual Studio: Start by opening Visual Studio (preferably Visual Studio 2022 or later).
2. Create a New Project:
   1. Select ASP.NET Core Web Application from the project templates.
   2. Name the project something like RegistrationApp.
3. Configure the Project:
   1. Choose ASP.NET Core 6.0 (or higher) as the framework.
   2. Select Web Application (Model-View-Controller) as the project template.
   3. Click Create to set up the project.

The project structure will be automatically generated, including folders for Controllers, Views, Models, and wwwroot (for static assets like CSS, JavaScript, and images).

**Step 2: Create Models for Registration Categories**

The **Model** represents the data structure of our application. Here, we’ll define models for each registration category (Student, Faculty, SAC).

**Explanation**:

* The Faculty class represents a model for the faculty registration form.
* DataAnnotations are used for validation. For instance:
  + [Required] ensures the field is not left empty.
  + [EmailAddress] validates the email format.
  + [MinLength(6)] ensures the password is at least 6 characters long.

Repeat a similar process to create models for **Student** and **SAC** registrations

**Step 3: Create Controllers for Each Category**

Controllers handle user requests, process data, and select views to render responses.

**Explanation**:

* The FacultyController handles the faculty registration process.
* The Register() method (GET) loads the registration form.
* The Register(Faculty faculty) method (POST) processes the form data.
* ModelState.IsValid checks if the submitted data meets validation rules (defined in the Faculty model).
* ViewBag.Message is used to display a success message if the form is valid.

**Step 4: Create Views for the Registration Forms**

Each view represents the UI for a specific part of the application. Here, we’ll create a view for the Faculty registration form.

1. In the **Views** folder, create a folder named **Faculty**.
2. Inside the **Faculty** folder, add a new Razor view named Register.cshtml.

**Explanation**:

* @model specifies the data model (Faculty) used in this view.
* The form fields (asp-for) are bound to model properties. Validation messages (asp-validation-for) display error messages if validation fails.
* @ViewBag.Message displays a success message after form submission.
* @section Scripts loads client-side validation scripts.

**Step 5: Styling the Application with CSS**

In the **wwwroot/css** folder, create a CSS file (site.css) and add the following code to style the registration form and layout.

**Step 6: Add JavaScript for Interactive Alerts**

In the **wwwroot/js** folder, create a JavaScript file (site.js) with the following code for navigation link interactions.

**Explanation**:

The script adds event listeners to each link in the navigation menu.

When a link is clicked, an alert message is displayed to provide feedback.

**Step 7: Install NuGet Packages**Open the Package Manager Console and run the following commands to install the required packages  
**Step 8: Configure Database Connection**In appsettings.json, add a connection string for MySQL (use your MySQL root password)

**Step 9: Create the DbContext**

* In the **Solution Explorer**, create a folder named Data
* In the Data folder, create a class named ApplicationDbContext.cs with the following code
* Register ApplicationDbContext in Program.cs

**Programs:**

**Example Model for Faculty Registration (Faculty.cs in the Models folder)**

using System.ComponentModel.DataAnnotations;

namespace RegistrationApp.Models

{

public class Faculty

{

public int Id { get; set; }

[Required]

public string Name { get; set; } = string.Empty;

[Required]

public string FacultyId { get; set; } = string.Empty;

[Required]

public string Department { get; set; } = string.Empty;

[Required]

[EmailAddress]

public string Email { get; set; } = string.Empty;

[Required]

public string Password { get; set; } = string.Empty;

}

}

**Example Model for Student Registration (Student.cs in the Models folder)**

using System.ComponentModel.DataAnnotations;

namespace RegistrationApp.Models

{

public class Student

{

public int Id { get; set; }

[Required]

public string Name { get; set; } = string.Empty;

[Required]

public string RollNo { get; set; } = string.Empty;

[Required]

public string Department { get; set; } = string.Empty;

[Required]

[EmailAddress]

public string Email { get; set; } = string.Empty;

[Required]

public string Password { get; set; } = string.Empty;

}

}

**Example Model for SAC Registration (SAC.cs in the Models folder)**

using System.ComponentModel.DataAnnotations;

namespace RegistrationApp.Models

{

public class SAC

{

public int Id { get; set; }

[Required]

public string Name { get; set; } = string.Empty;

[Required]

public string SACId { get; set; } = string.Empty;

[Required]

public string Role { get; set; } = string.Empty;

[Required]

[EmailAddress]

public string Email { get; set; } = string.Empty;

[Required]

public string Password { get; set; } = string.Empty;

}

}

**Example Model for Applicationdatabase context (ApplicationDbContext.cs in the Models folder)**

using Microsoft.EntityFrameworkCore;

using RegistrationApp.Models;

namespace RegistrationApp.Data

{

public class ApplicationDbContext : DbContext

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options) { }

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

public DbSet<Faculty> Faculties { get; set; }

public DbSet<SAC> SACs { get; set; }

}

}

**Example FacultyController:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using RegistrationApp.Data;

using RegistrationApp.Models;

namespace RegistrationApp.Controllers

{

public class FacultyController : Controller

{

private readonly ApplicationDbContext \_context;

public FacultyController(ApplicationDbContext context)

{

\_context = context;

}

// GET: Faculty

public async Task<IActionResult> Index()

{

return View(await \_context.Faculties.ToListAsync());

}

// GET: Faculty/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var faculty = await \_context.Faculties

.FirstOrDefaultAsync(m => m.Id == id);

if (faculty == null)

{

return NotFound();

}

return View(faculty);

}

// GET: Faculty/Create

public IActionResult Create()

{

return View();

}

// POST: Faculty/Create

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,Name,FacultyId,Department,Email,Password")] Faculty faculty)

{

if (ModelState.IsValid)

{

\_context.Add(faculty);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(faculty);

}

// GET: Faculty/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var faculty = await \_context.Faculties.FindAsync(id);

if (faculty == null)

{

return NotFound();

}

return View(faculty);

}

// POST: Faculty/Edit/5

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,Name,FacultyId,Department,Email,Password")] Faculty faculty)

{

if (id != faculty.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(faculty);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!FacultyExists(faculty.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(faculty);

}

// GET: Faculty/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var faculty = await \_context.Faculties

.FirstOrDefaultAsync(m => m.Id == id);

if (faculty == null)

{

return NotFound();

}

return View(faculty);

}

// POST: Faculty/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var faculty = await \_context.Faculties.FindAsync(id);

if (faculty != null)

{

\_context.Faculties.Remove(faculty);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool FacultyExists(int id)

{

return \_context.Faculties.Any(e => e.Id == id);

}

}

}

**Example StudentController:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using RegistrationApp.Data;

using RegistrationApp.Models;

namespace RegistrationApp.Controllers

{

public class StudentController : Controller

{

private readonly ApplicationDbContext \_context;

public StudentController(ApplicationDbContext context)

{

\_context = context;

}

// GET: Student

public async Task<IActionResult> Index()

{

return View(await \_context.Students.ToListAsync());

}

// GET: Student/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var student = await \_context.Students

.FirstOrDefaultAsync(m => m.Id == id);

if (student == null)

{

return NotFound();

}

return View(student);

}

// GET: Student/Create

public IActionResult Create()

{

return View();

}

// POST: Student/Create

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,Name,RollNo,Department,Email,Password")] Student student)

{

if (ModelState.IsValid)

{

\_context.Add(student);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(student);

}

// GET: Student/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var student = await \_context.Students.FindAsync(id);

if (student == null)

{

return NotFound();

}

return View(student);

}

// POST: Student/Edit/5

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,Name,RollNo,Department,Email,Password")] Student student)

{

if (id != student.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(student);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!StudentExists(student.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(student);

}

// GET: Student/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var student = await \_context.Students

.FirstOrDefaultAsync(m => m.Id == id);

if (student == null)

{

return NotFound();

}

return View(student);

}

// POST: Student/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var student = await \_context.Students.FindAsync(id);

if (student != null)

{

\_context.Students.Remove(student);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool StudentExists(int id)

{

return \_context.Students.Any(e => e.Id == id);

}

}

}

**Example SACController:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using RegistrationApp.Data;

using RegistrationApp.Models;

namespace RegistrationApp.Controllers

{

public class SACController : Controller

{

private readonly ApplicationDbContext \_context;

public SACController(ApplicationDbContext context)

{

\_context = context;

}

// GET: SAC

public async Task<IActionResult> Index()

{

return View(await \_context.SACs.ToListAsync());

}

// GET: SAC/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var sAC = await \_context.SACs

.FirstOrDefaultAsync(m => m.Id == id);

if (sAC == null)

{

return NotFound();

}

return View(sAC);

}

// GET: SAC/Create

public IActionResult Create()

{

return View();

}

// POST: SAC/Create

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,Name,SACId,Role,Email,Password")] SAC sAC)

{

if (ModelState.IsValid)

{

\_context.Add(sAC);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(sAC);

}

// GET: SAC/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var sAC = await \_context.SACs.FindAsync(id);

if (sAC == null)

{

return NotFound();

}

return View(sAC);

}

// POST: SAC/Edit/5

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,Name,SACId,Role,Email,Password")] SAC sAC)

{

if (id != sAC.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(sAC);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!SACExists(sAC.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(sAC);

}

// GET: SAC/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var sAC = await \_context.SACs

.FirstOrDefaultAsync(m => m.Id == id);

if (sAC == null)

{

return NotFound();

}

return View(sAC);

}

// POST: SAC/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var sAC = await \_context.SACs.FindAsync(id);

if (sAC != null)

{

\_context.SACs.Remove(sAC);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool SACExists(int id)

{

return \_context.SACs.Any(e => e.Id == id);

}

}

}

**Example Registration View (Create.cshtml,Delete.cshtml.Details.cshtml,Edit.cshtml,Index.cshtml)**

**Create.cshtml:**@model RegistrationApp.Models.Faculty

@{

ViewData["Title"] = "Create";

}

<h1>Create</h1>

<h4>Faculty</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Create">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<div class="form-group">

<label asp-for="Name" class="control-label"></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="FacultyId" class="control-label"></label>

<input asp-for="FacultyId" class="form-control" />

<span asp-validation-for="FacultyId" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Department" class="control-label"></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="Email" class="control-label"></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="Password" class="control-label"></label>

<input asp-for="Password" class="form-control" />

<span asp-validation-for="Password" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

</div>

</form>

</div>

</div>

<div>

<a asp-action="Index">Back to List</a>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

**Delete.cshtml:**@model RegistrationApp.Models.SAC

@{

ViewData["Title"] = "Delete";

}

<h1>Delete</h1>

<h3>Are you sure you want to delete this?</h3>

<div>

<h4>SAC</h4>

<hr />

<dl class="row">

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Name)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Name)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.SACId)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.SACId)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Role)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Role)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Email)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Email)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Password)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Password)

</dd>

</dl>

<form asp-action="Delete">

<input type="hidden" asp-for="Id" />

<input type="submit" value="Delete" class="btn btn-danger" /> |

<a asp-action="Index">Back to List</a>

</form>

</div>

**Details.cshtml:**@model RegistrationApp.Models.SAC

@{

ViewData["Title"] = "Details";

}

<h1>Details</h1>

<div>

<h4>SAC</h4>

<hr />

<dl class="row">

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Name)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Name)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.SACId)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.SACId)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Role)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Role)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Email)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Email)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Password)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Password)

</dd>

</dl>

</div>

<div>

<a asp-action="Edit" asp-route-id="@Model?.Id">Edit</a> |

<a asp-action="Index">Back to List</a>

</div>

**Edit.cshtml:**@model RegistrationApp.Models.SAC

@{

ViewData["Title"] = "Edit";

}

<h1>Edit</h1>

<h4>SAC</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Edit">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="Id" />

<div class="form-group">

<label asp-for="Name" class="control-label"></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="SACId" class="control-label"></label>

<input asp-for="SACId" class="form-control" />

<span asp-validation-for="SACId" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Role" class="control-label"></label>

<input asp-for="Role" class="form-control" />

<span asp-validation-for="Role" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Email" class="control-label"></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="Password" class="control-label"></label>

<input asp-for="Password" class="form-control" />

<span asp-validation-for="Password" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Save" class="btn btn-primary" />

</div>

</form>

</div>

</div>

<div>

<a asp-action="Index">Back to List</a>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

**Index.cshtml:**@model IEnumerable<RegistrationApp.Models.SAC>

@{

ViewData["Title"] = "Index";

}

<h1>SAC registration</h1>

<p>

<a asp-action="Create">New Registration</a>

</p>

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Name)

</th>

<th>

@Html.DisplayNameFor(model => model.SACId)

</th>

<th>

@Html.DisplayNameFor(model => model.Role)

</th>

<th>

@Html.DisplayNameFor(model => model.Email)

</th>

<th>

@Html.DisplayNameFor(model => model.Password)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.Name)

</td>

<td>

@Html.DisplayFor(modelItem => item.SACId)

</td>

<td>

@Html.DisplayFor(modelItem => item.Role)

</td>

<td>

@Html.DisplayFor(modelItem => item.Email)

</td>

<td>

@Html.DisplayFor(modelItem => item.Password)

</td>

<td>

<a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |

<a asp-action="Details" asp-route-id="@item.Id">Details</a> |

<a asp-action="Delete" asp-route-id="@item.Id">Delete</a>

</td>

</tr>

}

</tbody>

</table>

**Site.css:**body {

font-family: Arial, sans-serif;

margin: 0;

padding: 0;

background-color: #f9fafb;

color: #333;

}

.container {

display: flex;

flex-direction: column;

height: 100vh;

}

.top-container {

display: flex;

flex: 0 0 30%;

background-color: #3a3f51;

color: #fff;

padding: 20px;

}

.left-section {

flex: 1;

display: flex;

justify-content: center;

align-items: center;

}

.left-section img {

max-width: 80%;

height: auto;

border-radius: 8px;

}

.menu-section {

flex: 2;

display: flex;

justify-content: center;

align-items: center;

}

.menu-section ul {

list-style-type: none;

padding: 0;

display: flex;

}

.menu-section ul li {

margin: 0 20px;

}

.menu-section ul li a {

text-decoration: none;

font-size: 18px;

color: #fff;

font-weight: bold;

padding: 8px 15px;

background-color: #5c6bc0;

border-radius: 5px;

transition: background-color 0.3s;

}

.menu-section ul li a:hover {

background-color: #3949ab;

}

.quote-section p {

font-size: 16px;

text-align: center;

color: #b0bec5;

font-style: italic;

}

.bottom-container {

flex: 0 0 70%;

background-color: #ffffff;

padding: 20px;

display: flex;

justify-content: center;

align-items: flex-start;

}

#login-content {

width: 100%;

max-width: 600px;

background-color: #f5f5f5;

padding: 30px;

border-radius: 10px;

box-shadow: 0 0 15px rgba(0, 0, 0, 0.2);

}

h2 {

font-size: 26px;

color: #3949ab;

margin-bottom: 20px;

}

.form-group label {

font-weight: bold;

color: #3949ab;

}

.form-control {

width: 100%;

padding: 12px;

margin-bottom: 15px;

border-radius: 5px;

border: 1px solid #ccc;

background-color: #ffffff;

}

button {

padding: 12px 20px;

background-color: #3949ab;

color: white;

border: none;

border-radius: 5px;

cursor: pointer;

font-size: 16px;

transition: background-color 0.3s;

}

button:hover {

background-color: #303f9f;

}

**Site.js:**

document.addEventListener("DOMContentLoaded", function () {

document.getElementById("home-link").addEventListener("click", function () {

alert("Welcome to the Home page!");

});

document.getElementById("student-link").addEventListener("click", function () {

alert("Register as a Student!");

});

document.getElementById("faculty-link").addEventListener("click", function () {

alert("Register as Faculty!");

});

document.getElementById("sac-link").addEventListener("click", function () {

alert("Register as SAC!");

});

});

**\_Layout.cshtml:**<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>@ViewData["Title"] - Registration</title>

<link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />

</head>

<body>

<header class="top-container">

<div class="left-section">

<img src="~/images/vvit.jpeg" alt="Your App Logo" />

</div>

<nav class="menu-section">

<ul>

<li><a asp-controller="Home" asp-action="Index" id="home-link">Home</a></li>

<li><a asp-controller="Student" asp-action="Index" id="student-link">Student</a></li>

<li><a asp-controller="Faculty" asp-action="Index" id="faculty-link">Faculty</a></li>

<li><a asp-controller="SAC" asp-action="Index" id="sac-link">SAC</a></li>

</ul>

</nav>

<div class="quote-section">

<p>"Absolute discipline<br />Absolute education"</p>

</div>

</header>

<main class="bottom-container">

<div id="login-content" role="main">

@RenderBody()

</div>

</main>

<script src="~/js/site.js" asp-append-version="true"></script>

@RenderSection("Scripts", required: false)

</body>

</html>

**Appsettings.json:**{

"ConnectionStrings": {

"DefaultConnection": "Server=localhost;Database=RegistrationDB;User=root;Password=Akram04@vvit;"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

**Program.cs:**using Microsoft.EntityFrameworkCore;

using RegistrationApp.Data; // Ensure this namespace is correct based on your project structure

using RegistrationApp.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews();

// Configure ApplicationDbContext with MySQL connection

builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseMySql(builder.Configuration.GetConnectionString("DefaultConnection"),

new MySqlServerVersion(new Version(8, 0, 21))));

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=Home}/{action=Index}/{id?}");

app.Run();

**SQL Query to create tables in Database:**CREATE DATABASE RegistrationDB;

USE RegistrationDB;

CREATE TABLE Students (

Id INT AUTO\_INCREMENT PRIMARY KEY,

Name VARCHAR(100),

RollNo VARCHAR(50),

Department VARCHAR(100),

Email VARCHAR(100),

Password VARCHAR(100)

);

CREATE TABLE Faculties (

Id INT AUTO\_INCREMENT PRIMARY KEY,

Name VARCHAR(100),

FacultyId VARCHAR(50),

Department VARCHAR(100),

Email VARCHAR(100),

Password VARCHAR(100)

);

CREATE TABLE SACS (

Id INT AUTO\_INCREMENT PRIMARY KEY,

Name VARCHAR(100),

SACId VARCHAR(50),

Role VARCHAR(100),

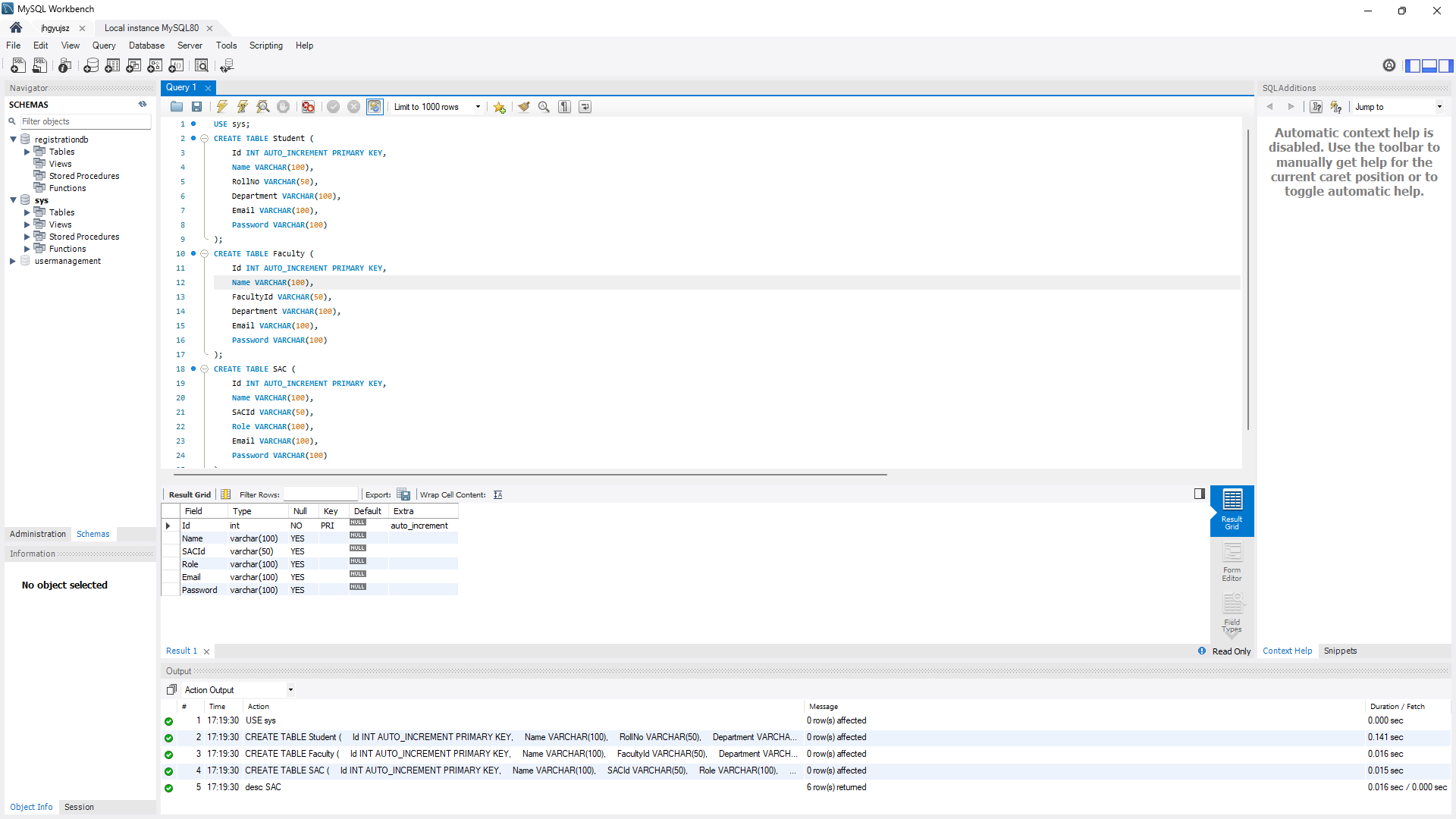
Email VARCHAR(100),

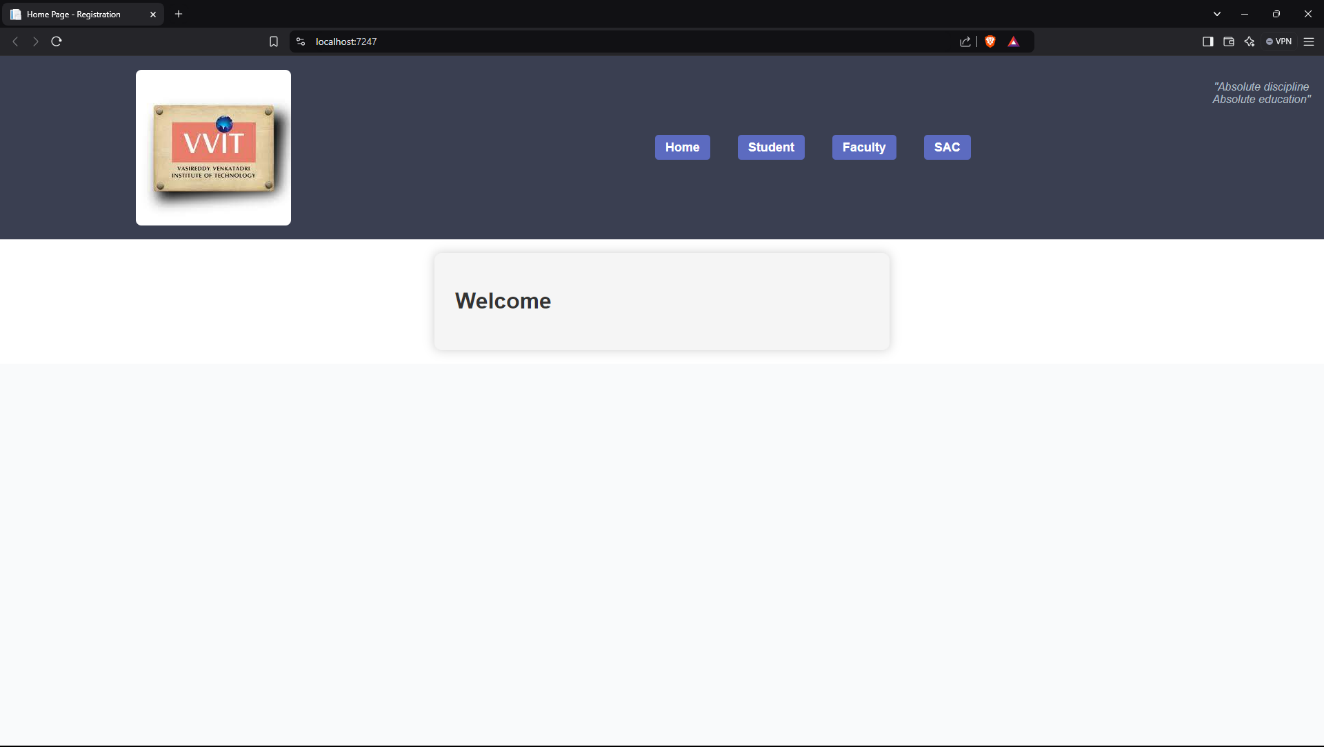
Password VARCHAR(100)

);

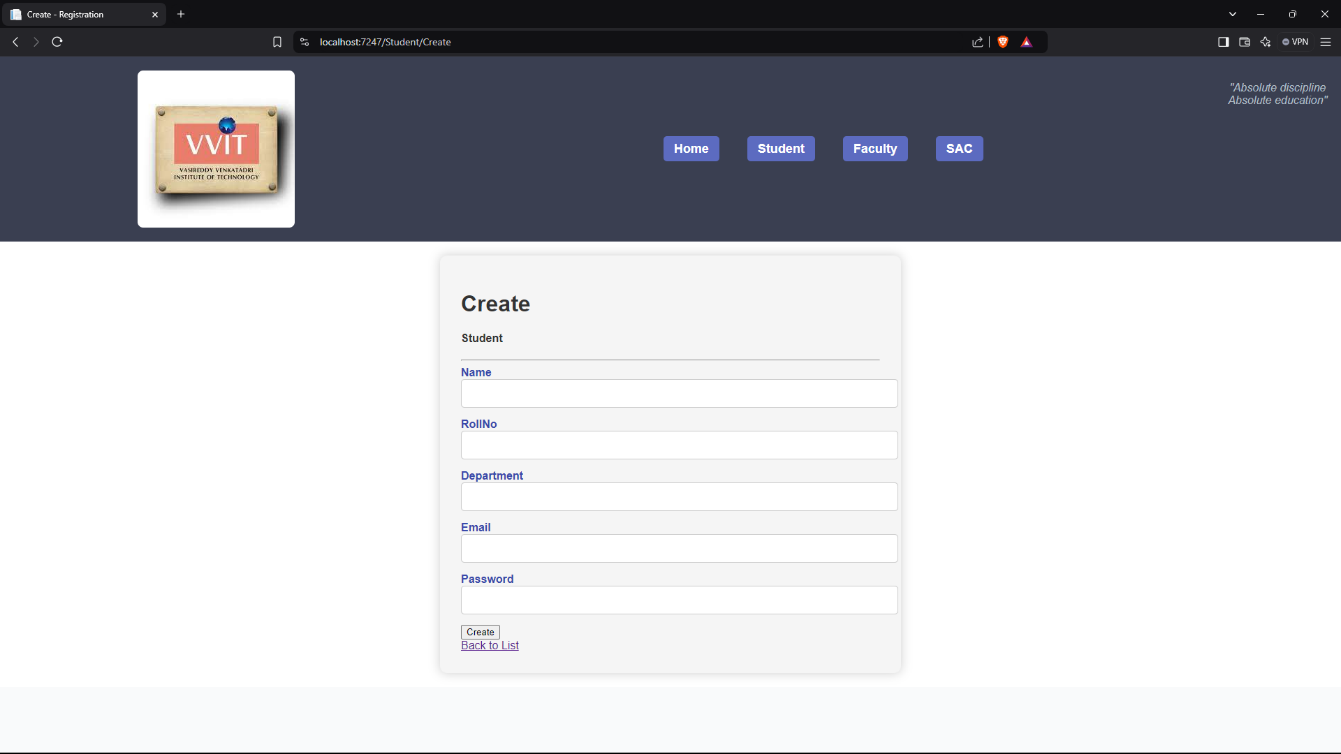
**Output:**

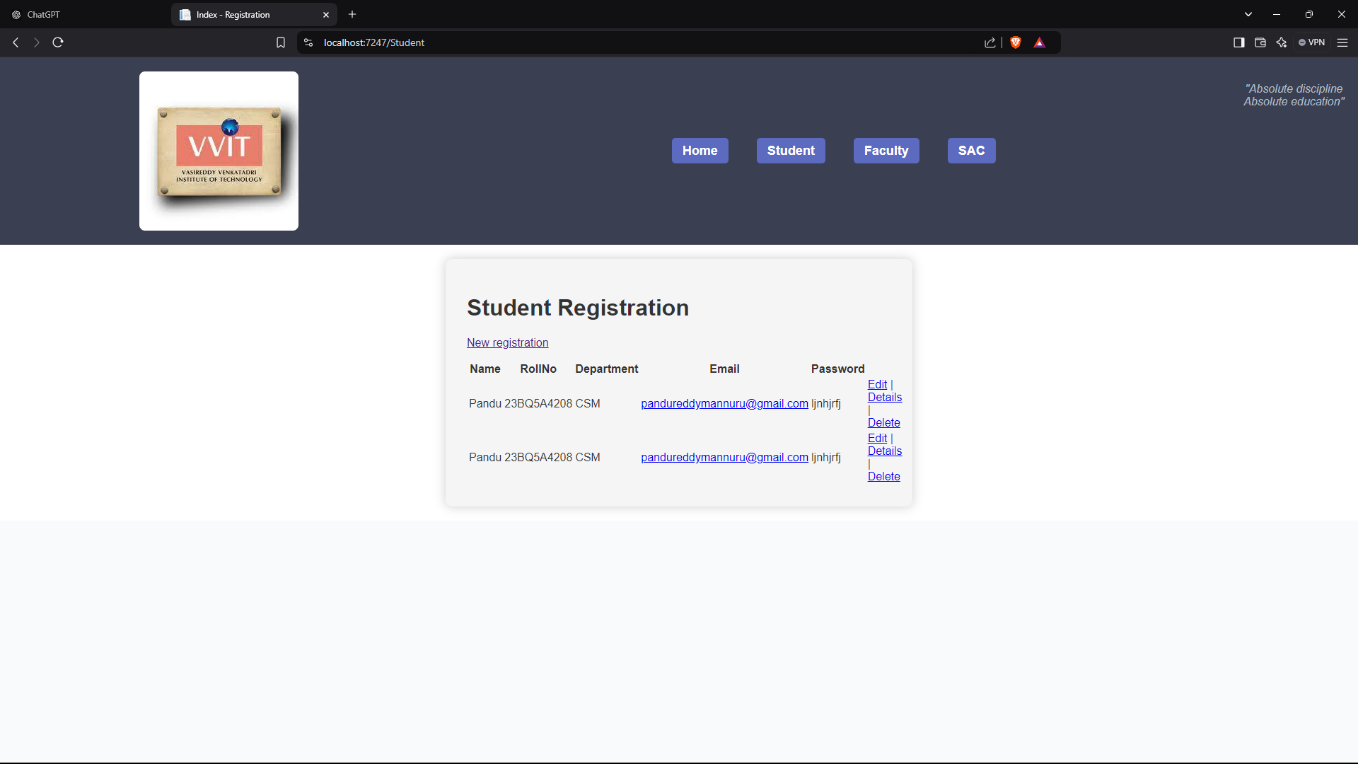
This ASP.NET MVC application demonstrates a structured way to build a registration system with separate forms for different user categories. Each part, from model validation to CSS styling and JavaScript interactivity, contributes to a cohesive and user-friendly web application.

**Tables Creation:**

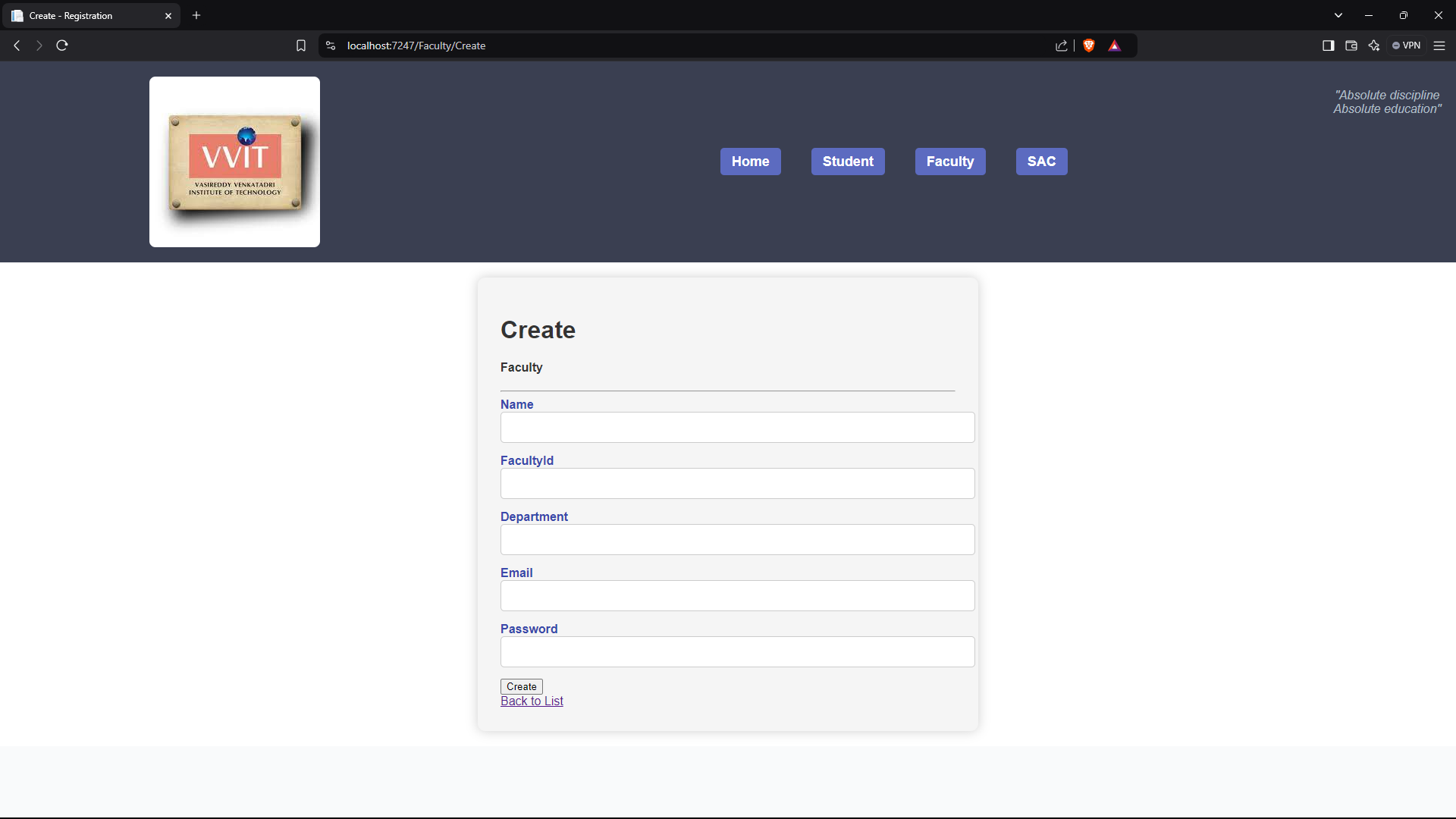
**Home Page :** The home page includes a navigation menu with links to "Home," "Student," "Faculty," and "SAC."

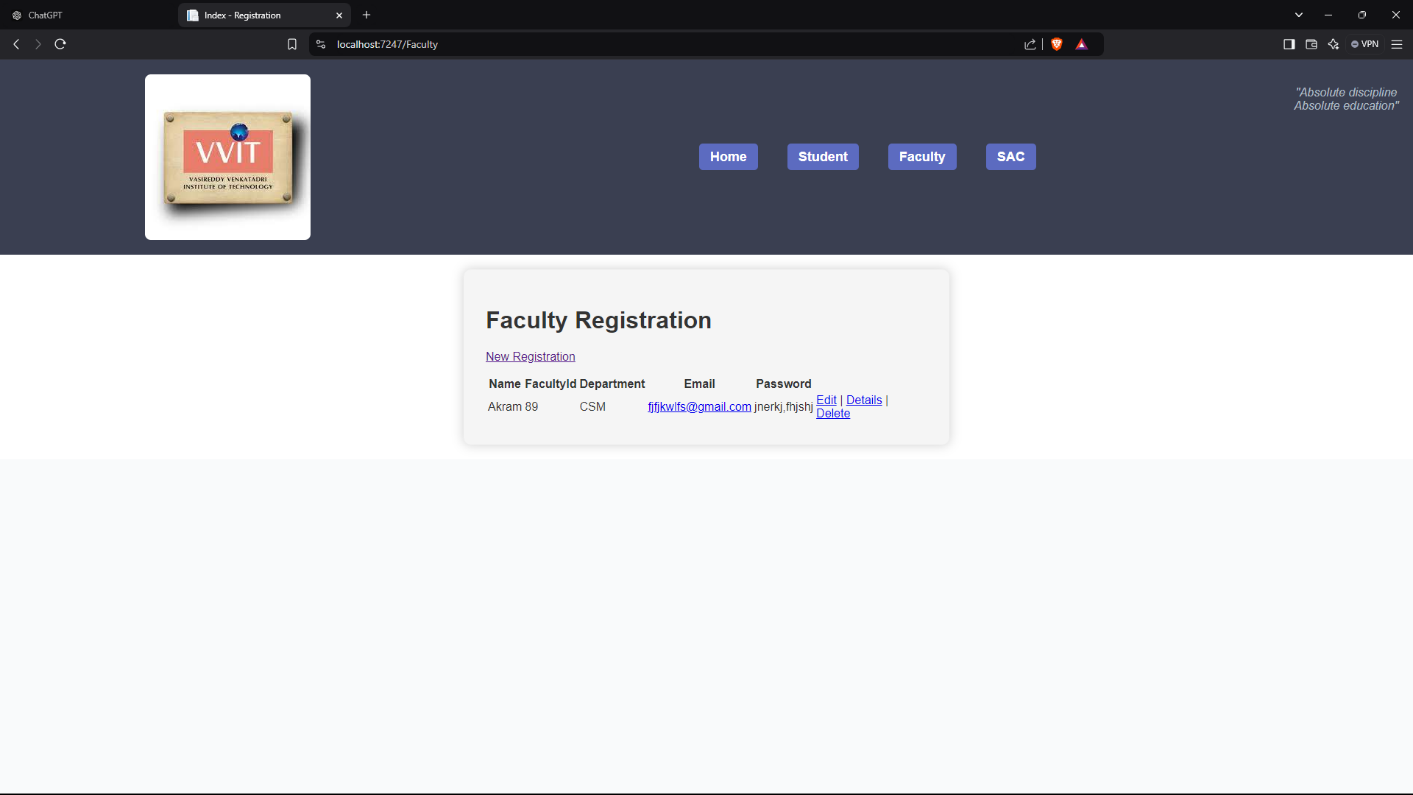
**Student Registration Page**: Displays the top section with the image, menu, and quote. The bottom section includes the "Student Registration" form with fields for student details and a "Register" button.



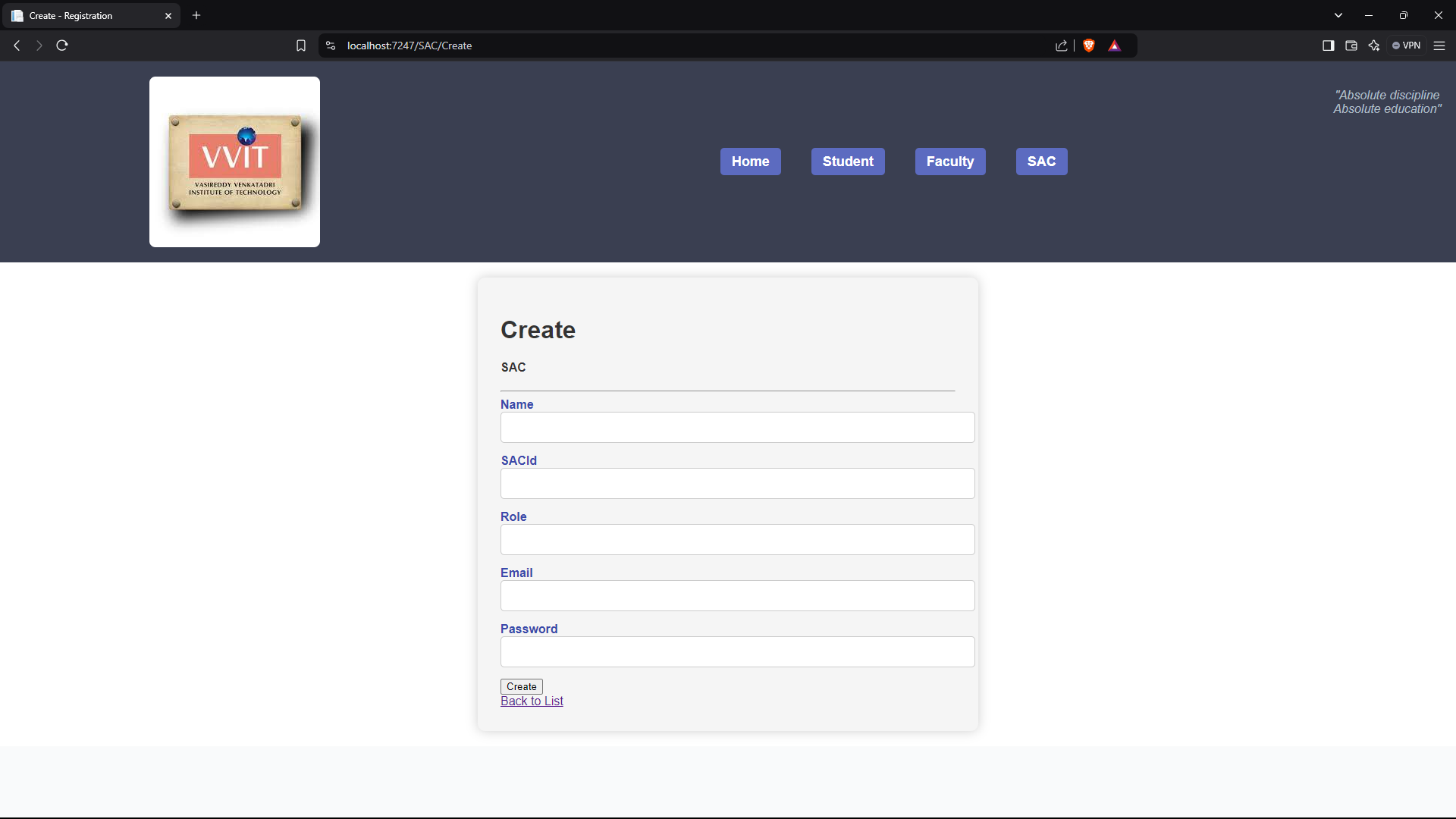
When you create a new registration, the data is automatically rendered and displayed in the list right away. This means that the information gets saved and appears in the table immediately after submitting the form.

**Faculty Registration Page**: Similar layout to the student page but tailored for faculty registration, with fields specific to faculty data (like Faculty ID).

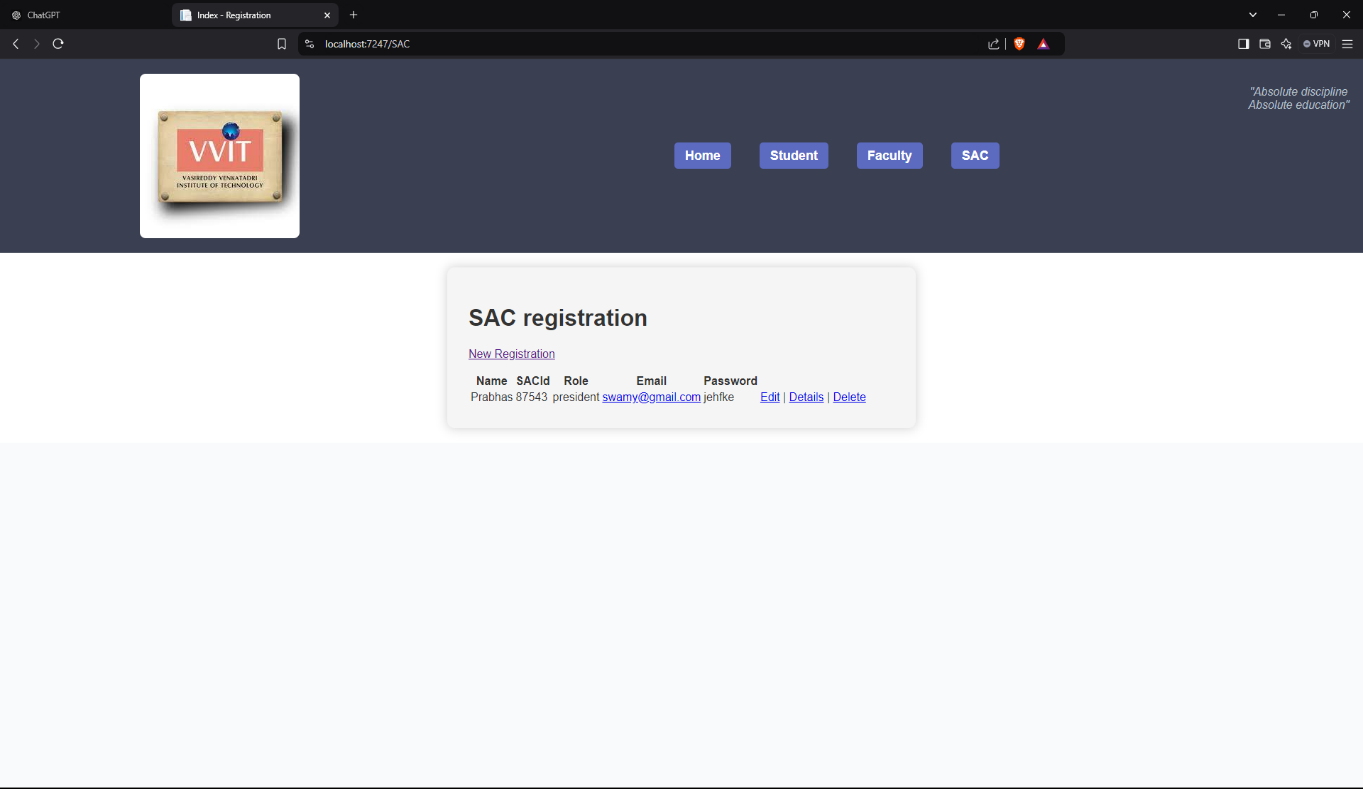


When you create a new registration, the data is automatically rendered and displayed in the list right away. This means that the information gets saved and appears in the table immediately after submitting the form.

**SAC Registration Page**: Displays the top section with the image, menu, and quote. The bottom section includes the "SAC Registration" form with fields for student details and a "Register" button.



When you create a new registration, the data is automatically rendered and displayed in the list right away. This means that the information gets saved and appears in the table immediately after submitting the form.



## Experiment Number: Date of Experiment:

**Aim:**

Create a LINQ Console Application for Data Storing, Select, Filter, Sort, Grouping Operations.

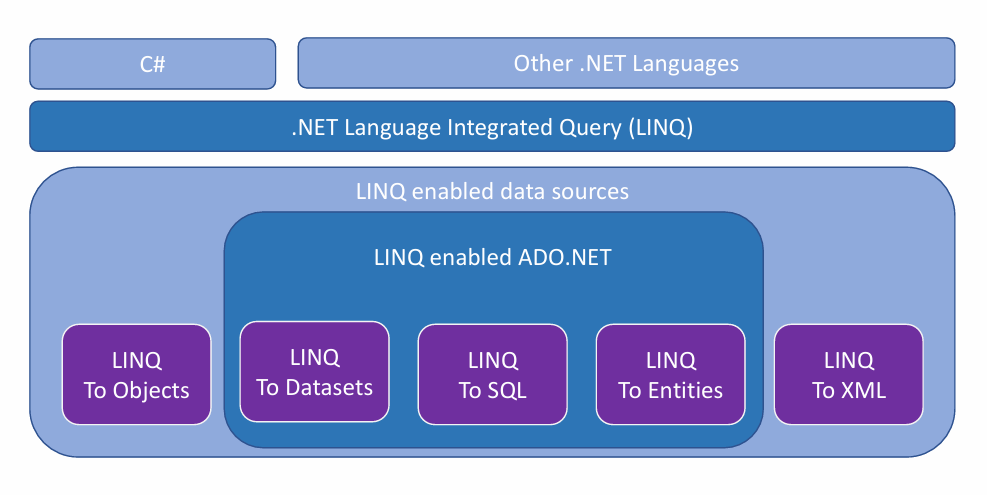
## Description:

**LINQ Overview**

Language-Integrated Query (LINQ) is a powerful tool in .NET that enables querying data in C# in a more readable and efficient way. LINQ provides a consistent model for querying various data sources, such as collections, SQL databases, XML, and more, by integrating query capabilities directly within C#. With LINQ, developers can use SQL-like syntax in C# to filter, sort, group, and retrieve data seamlessly across different data types. Its integration with C# ensures type safety and compile-time checks, reducing runtime errors.

**LINQ Architecture**

The LINQ architecture consists of various components that interact to enable data querying across multiple data sources. The following image illustrates how LINQ integrates with different .NET languages and data sources



* LINQ operates with multiple .NET languages, primarily C#, but can also work with other .NET languages.
* It interfaces with various data sources, including objects, datasets, SQL, entities, and XML, via ADO.NET.
* Each of these data sources has a dedicated LINQ provider, such as LINQ to Objects, LINQ to Datasets, LINQ to SQL, LINQ to Entities, and LINQ to XML, that enables querying on that particular source.

## Step-by-Step Guide: Creating and Running a LINQ Console Application in Visual Studio

### Step 1: Set Up the .NET Project

### Open Visual Studio and select Create a new project.

### Choose Console App (.NET Core) or Console App (.NET Framework) as the project template.

### Click Next, give your project a name (e.g., Lab\_9), and specify a directory to save it.

### Select Create to initialize the project.

### Step 2: Define the Data Model (Student Class)

1. **Add a New Class**:
   * In the **Solution Explorer**, right-click on the project name and select **Add > Class**.
   * Name this class (e.g., Student.cs) and click **Add**.
2. **Define the Class Properties**: In the new class file, define the properties (e.g., Name, Age, Grade, etc.) that will represent each data field in your LINQ queries.

### Step 3: Implement LINQ Queries in the Main Program

1. **Navigate to Program.cs**:
   * Open the Program.cs file from **Solution Explorer**.
2. **Initialize a Data Collection**:
   * Create a list or other data collection of the model class defined in Student.cs.
3. **Write LINQ Queries**:
   * Inside the Main method, apply LINQ queries such as:
     + **Selection**: Retrieve specific attributes or a subset of data.
     + **Filtering**: Extract data based on certain conditions.
     + **Sorting**: Arrange data in a particular order.
     + **Grouping**: Organize data by distinct categories.

### Step 4: Run and Test the Application

1. **Save All Changes**: Save the file to ensure all code changes are updated.
2. **Run the Application**: Click on the **Run** button (or use Ctrl + F5) to execute the console application and see the output of each LINQ operation

## Program:

**Student.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab\_9

{

internal class Student

{

// --------------Data Storing-----------

// Properties of the Student class

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public string Grade { get; set; }

// Optional: Constructor to initialize Student object

public Student(int id, string name, int age, string grade)

{

Id = id;

Name = name;

Age = age;

Grade = grade;

}

}

}

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

namespace Lab\_9

{

internal class Program

{

static void Main(string[] args)

{

// Step 1: Create a list of Student objects

List<Student> students = new List<Student>

{

new Student(1, "Alice", 20, "A"),

new Student(2, "Bob", 22, "B"),

new Student(3, "Charlie", 21, "A"),

new Student(4, "Diana", 23, "C"),

new Student(5, "Eve", 20, "B")

};

// Step 2: Select - Project student names

var studentNames = students.Select(s => s.Name);

Console.WriteLine("Student Names:");

foreach (var name in studentNames)

{

Console.WriteLine(name);

}

// Step 3: Filter - Select students with grade "A"

var gradeAStudents = students.Where(s => s.Grade == "A");

Console.WriteLine("\nStudents with Grade A:");

foreach (var student in gradeAStudents)

{

Console.WriteLine($"{student.Name} (Age: {student.Age})");

}

// Step 4: Sort - Order students by age

var sortedByAge = students.OrderBy(s => s.Age);

Console.WriteLine("\nStudents sorted by Age:");

foreach (var student in sortedByAge)

{

Console.WriteLine($"{student.Name} (Age: {student.Age})");

}

// Step 5: Group - Group students by grade

var groupedByGrade = students.GroupBy(s => s.Grade);

Console.WriteLine("\nStudents grouped by Grade:");

foreach (var group in groupedByGrade)

{

Console.WriteLine($"Grade {group.Key}:");

foreach (var student in group)

{

Console.WriteLine($" - {student.Name}");

}

}

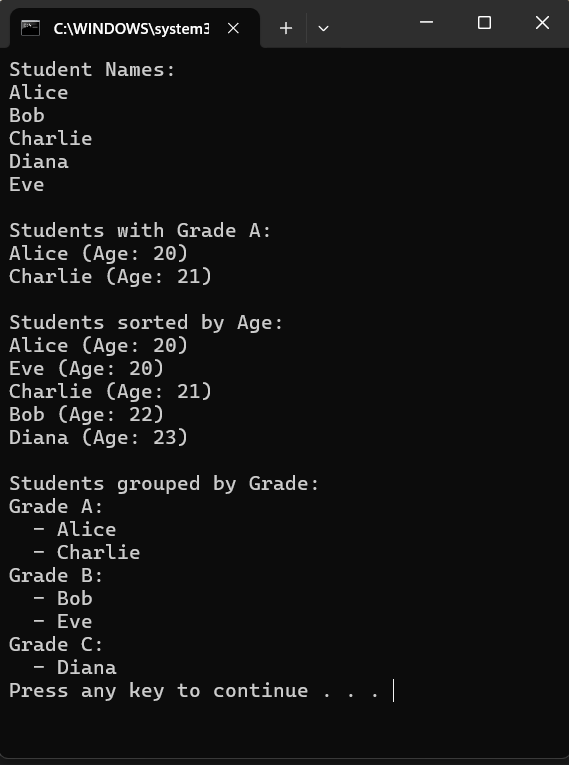
}

}

}

## Output:

The console output should look something like this:



## Experiment Number: Date of Experiment:

**Aim:**

## Create Dynamic LINQ Console Application for implementing Dynamic Expressions

## Description:

**LINQ and Dynamic Expressions**

Language-Integrated Query (LINQ) is a powerful feature in .NET that allows developers to write queries directly in C# (or other .NET languages) against various data sources such as collections, databases, XML, and more. LINQ simplifies data manipulation and retrieval by providing a consistent query syntax across different data sources. It enhances readability and maintainability by allowing developers to work with data in a more natural way.

**Dynamic LINQ** extends the capabilities of LINQ by allowing developers to create and execute queries using strings at runtime. This is particularly useful when the structure of the query is not known until execution, making it flexible for scenarios such as user-driven searches or filtering.

## Steps to Create a Dynamic LINQ Console Application

### Step 1: Set Up the .NET Project

### Open Visual Studio and select Create a new project.

### Choose Console App (.NET Core) or Console App (.NET Framework) as the project template.

### Click Next, give your project a name (e.g., Lab\_10), and specify a directory to save it.

### Select Create to initialize the project.

### Step 2:Install Dynamic LINQ Package

* + 1. Go to Tools > NuGet Package Manager > Package Manager Console.
    2. In the Package Manager Console, type the following command to install the System.Linq.Dynamic.Core package:
* Install-Package System.Linq.Dynamic.Core
  + 1. Press Enter to execute the command. This will download and install the package into your project.

### Step 3: Define the Data Model (Student Class)

1. **Add a New Class**:
   * In the **Solution Explorer**, right-click on the project name and select **Add > Class**.
   * Name this class (e.g., Student.cs) and click **Add**.
2. **Define the Class Properties**: In the new class file, define the properties (e.g., Name, Age, Grade, etc.) that will represent each data field in your LINQ queries.

### Step 4: Implement LINQ Queries in the Main Program

1. **Navigate to Program.cs**:
   * Open the Program.cs file from **Solution Explorer**.
2. Import the necessary namespaces:

* using System.Linq.Dynamic.Core;

1. Implement the Main Program**:**

* Open the Program.cs file.
  + Define the Main method, which will initialize the list of students and prompt the user for filtering conditions.
* Implement dynamic LINQ to filter the student list based on user input.

### Step 5: Run and Test the Application

1. **Save All Changes**: Save the file to ensure all code changes are updated.
2. **Run the Application**: Click on the **Run** button (or use Ctrl + F5) to execute the console application and see the output of each LINQ operation
3. Enter a filtering condition when prompted (e.g., Age > 20).
4. View the filtered results in the console.

## Program:

**Student.cs**

using System;

namespace Lab\_10

{

internal class Student

{

// Properties of the Student class

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public string Grade { get; set; }

// Constructor to initialize Student object

public Student(int id, string name, int age, string grade)

{

Id = id;

Name = name;

Age = age;

Grade = grade;

}

}

}

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Linq.Dynamic.Core;

namespace Lab\_10

{

internal class Program

{

static void Main(string[] args)

{

// Sample data using constructor

List<Student> students = new List<Student>

{

new Student(1, "Alice", 20, "A"),

new Student(2, "Bob", 22, "B"),

new Student(3, "Charlie", 21, "A"),

new Student(4, "David", 23, "C")

};

Console.WriteLine("Enter a condition to filter students (e.g., Age > 20 or Grade == 'A'):");

string condition = Console.ReadLine();

// Dynamic LINQ query

var filteredStudents = students.AsQueryable().Where(condition).ToList();

// Display results

Console.WriteLine("Filtered Students:");

foreach (var student in filteredStudents)

{

Console.WriteLine($"ID: {student.Id}, Name: {student.Name}, Age: {student.Age}, Grade: {student.Grade}");

}

}

}

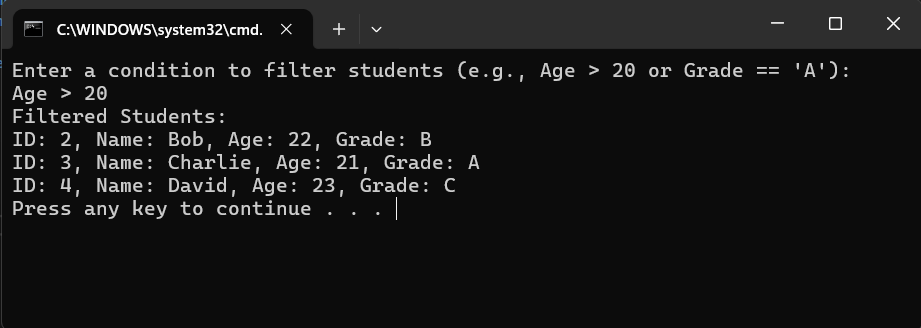
}

## Output:

The console output should look something like this:

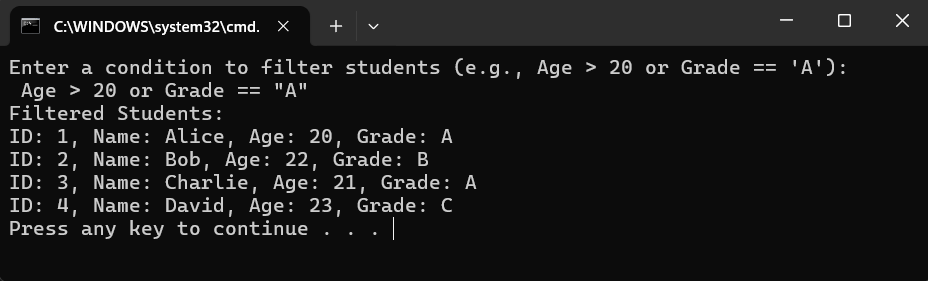
**Input Condition**: Age > 20

Successful Expected Output:



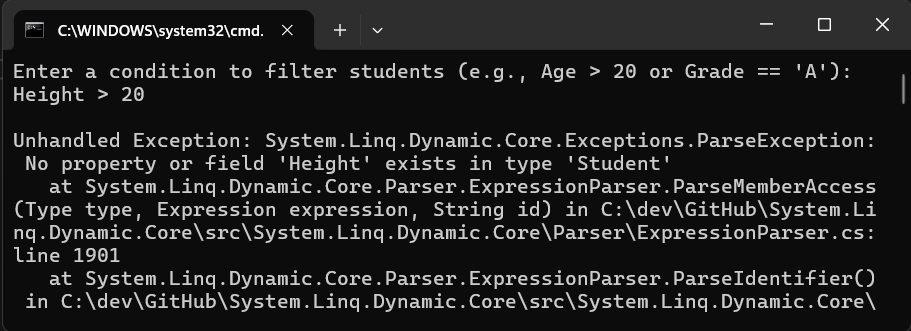
**Input Condition**: Age > 20 or Grade == "A"

Successful Expected Output:



**Input Condition**: Height > 20

Error:The property 'Height' does not exist in the student class



**Input Condition**: Age 20

Error: The input condition has invalid syntax

