| **Name of Student:** Ajay Karthikesan | | | |
| --- | --- | --- | --- |
| **Roll Number:** 57 | | **Assignment Number:** 4 | |
| **Aim of Assignment:**  Design an online registration form using HTML Controls  and validation controls | | | |
| **DOP:** 19.4.23 | | **DOS:** 26.4.23 | |
| **CO Mapped:**  CO1 | **PO Mapped:**  PO3, PO5, PSO1, PSO2 | **Faculty Signature:** | **Marks:** |

## 

## Practical No. 1

**Aim:**

**Theory:**

There are Six Servers as well as client-side validation controls in

ASP.Net

* RequiredFieldValidator control:
  + This control ensures that the control it is used for validation is not empty when the form is submitted. In other words suppose there is one Text Box control and you have used a RequiredFieldValidator to validate that text box; then before submitting the data on the server it checks if the text box is not empty.
* RangeValidator:
  + Checks that the value of the associated control is within a specified range. The value and the range can be numerical, a date or a string. In other words suppose there is one text box and you want to allow only 10 digits or any strings with a specified range using RangeValidator then before submitting the data on the server it ensure that the value is within a specified range.
* CompareValidator:
  + Checks that the value of the associated control matches a specified comparison (less than, greater than, and so on) against another constant value or control.
* RegularExpressionValidator:
  + Checks if the value of the control it has to validate matches the specified regular expression.
* CustomValidator:
  + Allows specification of any client-side JavaScript validation routine and its serverside counterpart to perform your own custom validation logic.
* ValidationSummary:
  + Shows a summary with the error messages for each failed validator on the page (or in a pop-up message box)
* ASP.NET Master Pages: ASP.NET master pages allow you to create a consistent layout for the pages in your application. A single master page defines the look and feel and standard behavior that you want for all of the pages (or a group of pages) in your application. You can then create individual content pages that contain the content you want to display. When users request the content pages, they merge with the master page to produce output that combines the layout of the master page with the content from the content page. Master pages actually consist of two pieces, the master page itself and one or more content pages.
* Master Pages: A master page is an ASP.NET file with the extension .master (for example, MySite.master) with a predefined layout that can include static text, HTML elements, and server controls. The master page is identified by a special @Master directive that replaces the @Page directive that is used for ordinary .aspx pages. Replaceable Content Placeholders In addition to static text and controls that will appear on all pages, the master page also includes one or more ContentPlaceHolder controls. These placeholder controls define regions where replaceable content will appear. In turn, the replaceable content is defined in content pages.
* Content Pages: You define the content for the master pages placeholder controls by creating individual content pages, which are ASP.NET pages (.aspx files and, optionally, code-behind files) that are bound to a specific master page. The binding is established in the content page&#39;s @Page directive by including a MasterPageFile attribute that points to the master page to be used. In the content page, you create the content by adding Content controls and mapping them to ContentPlaceHolder controls on the master page.
* Advantages of Master Pages:
  + Master pages provide functionality that developers have traditionally created by copying existing code, text, and control elements repeatedly; using framesets; using include files for common elements; using ASP.NET user controls; and so on.
* Advantages of master pages include the following:
  + They allow you to centralize the common functionality of your pages so that you can make updates in just one place.
  + They make it easy to create one set of controls and code and apply the results to a set of pages. For example, you can use controls on the master page to create a menu that applies to all pages.
  + They give you fine-grained control over the layout of the final page by allowing you to control how the placeholder controls are rendered.
  + They provide an object model that allows you to customize the master page from individual content pages.

**Code:**

File: success-page.aspx.cs

namespace Practical4

{

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

{

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

{

nameLbl.Text = Request.QueryString["name"];

emailLbl.Text = Request.QueryString["email"];

}

}

}

File: success-page.aspx

﻿<%@ Page Title="" Language="C#" MasterPageFile="~/master.Master" AutoEventWireup="true" CodeBehind="success-page.aspx.cs" Inherits="Practical4.SuccessPage" %>

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

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">

<label>You've Registered Successfully</label><br />

<label>Name: </label>

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

<label>Email ID: </label>

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

</asp:Content>

File: master.Master

﻿<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="master.master.cs" Inherits="Practical4.master" %>

<!DOCTYPE html>

<html>

<head runat="server">

<title></title>

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

</asp:ContentPlaceHolder>

</head>

<body>

<header>

<h1>Nim Inc</h1>

</header>

<main>

<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">

</asp:ContentPlaceHolder>

</main>

<footer>

Website and all its contents belong to Nim Inc.<br />

Author: Ajay Karthikesan, Roll No.: 57

</footer>

</body>

</html>

File: index.aspx.cs

namespace Practical4

{

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

{

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

{

}

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

{

if (IsValid)

{

Response.Redirect("success-page.aspx?name="+nameLbl.Text+"&email="+emailLbl.Text);

}

}

}

}

File: index.aspx

﻿<%@ Page Title="" Language="C#" MasterPageFile="~/master.Master" AutoEventWireup="true" CodeBehind="index.aspx.cs" Inherits="Practical4.index" %>

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

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">

<form runat="server">

<label>Name <asp:TextBox ID="nameLbl" runat="server"></asp:TextBox></label><asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server" ErrorMessage="This is a required field" ControlToValidate="nameLbl"></asp:RequiredFieldValidator><br />

<label>Email ID <asp:TextBox ID="emailLbl" runat="server"></asp:TextBox></label><asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server" ErrorMessage="This a required field" ControlToValidate="emailLbl"></asp:RequiredFieldValidator>

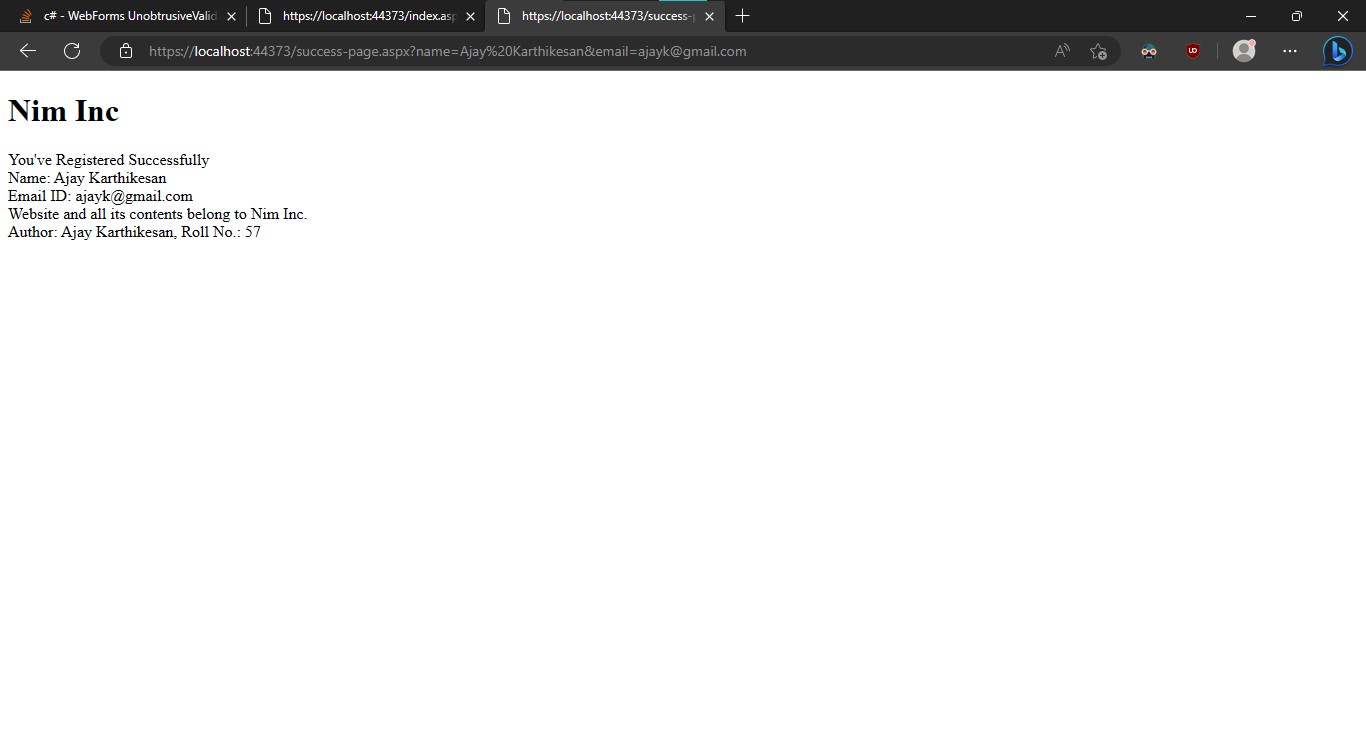
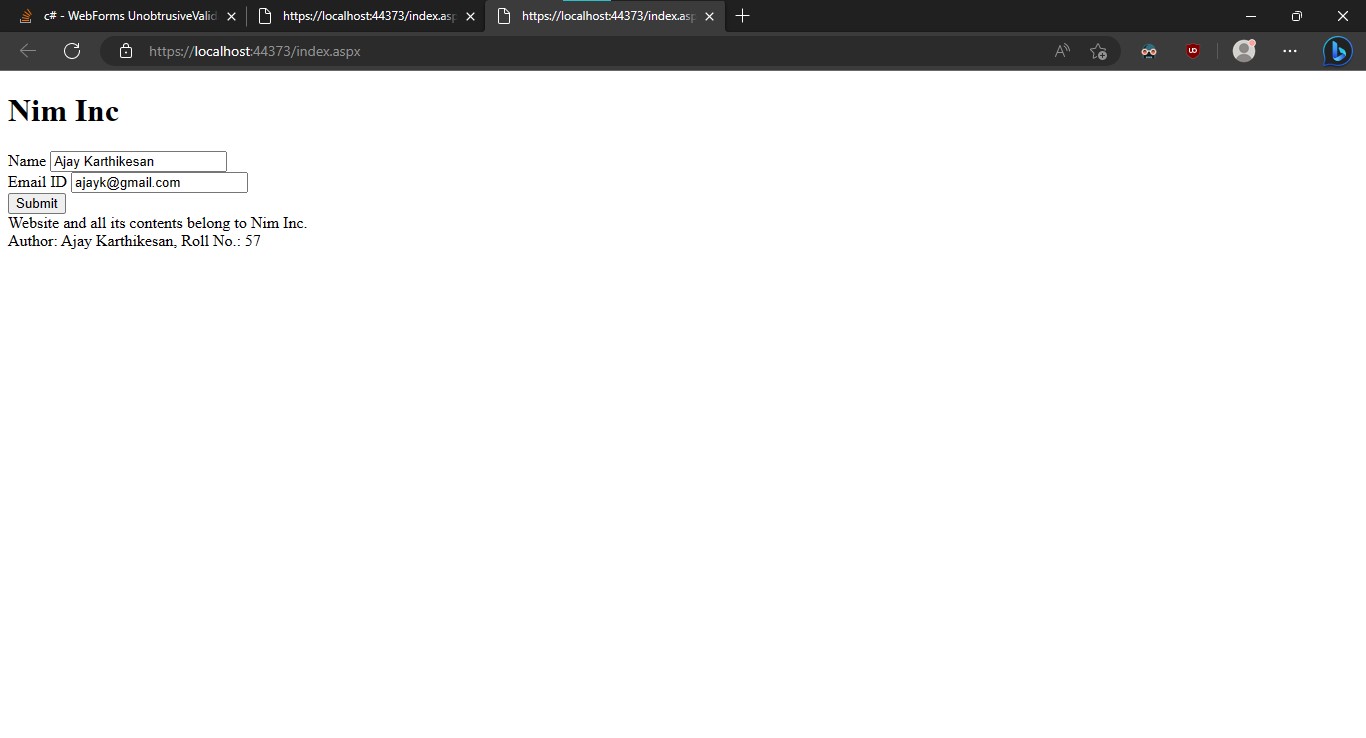
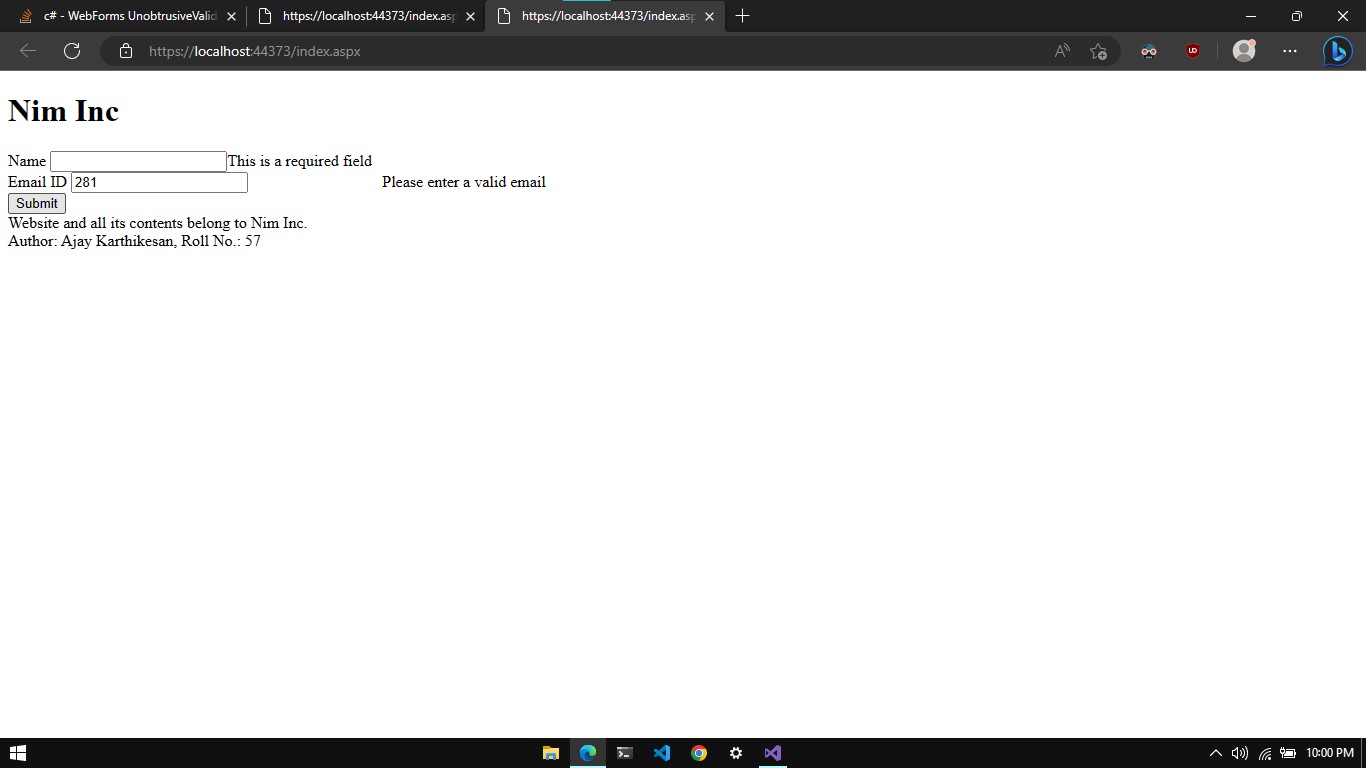
<asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat="server" ErrorMessage="Please enter a valid email" ControlToValidate="emailLbl" ValidationExpression="\w+([-+.']\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*"></asp:RegularExpressionValidator><br />

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

</form>

</asp:Content>

**Output:**



**Conclusion:**

I learnt how to design an online registration form using HTML Controls

and validation controls.