Experiment Title: Accepting and validating user entered data in registration form using ASP.NET

Objective:

To learn and implement the Accepting and validating user entered data in forms

Theory:

UNDERSTANDING VALIDATION

Validation is a set of rules that you apply to the data you collect. These rules can be many or few and enforced either strictly or in a lax manner: It really depends on your business rules and requirements. No perfect validation process exists because some users may fi nd a way to cheat to some degree, no matter what rules you establish. The trick is to fi nd the right balance of the fewest rules and the proper strictness, without compromising the usability of the application.

CLIENT-SIDE VERSUS SERVER-SIDE VALIDATION

server-side validation

Suppose that the end user clicks the Submit button on a form after filling out some information. What happens in ASP.NET is that this form is packaged in a request and sent to the server where the application resides. At this point in the request/response cycle, you can run validation checks on the information submitted. Doing this is called server-side validation because it occurs on the server.

client-side validation

On the other hand, it's also possible to supply a client-side script (usually in the form of JavaScript) in the page that is posted to the end user's browser to perform validations on the data entered in the form before the form is posted back to the originating server. In this case, client-side validation has occurred.

Client-side validation is quick and responsive for the end user. It is something end users expect of the forms that they work with. If something is wrong with the form, using client-side validation ensures that the end user knows about it as soon as possible. The reason for this is that the client-side validation, if called properly, executes before the form is posted back to the server.

With this said, client-side validation is the more insecure form of validation. When a page is generated in an end user's browser, this end user can look at the code of the page quite easily (simply by right-clicking his mouse in the browser and selecting View Code). When he or she does so, in addition to seeing the HTML code for the page, all of the JavaScript that is associated with the page can be viewed. If you are validating your form client-side, it doesn't take much for the crafty hacker to repost a form (containing the values he wants in it) to your server as valid. Cases also exist in which clients have simply disabled the client-scripting capabilities in their browsers — thereby making your validations useless. Therefore, client-side validation should be considered a convenience and a courtesy to the end user and never a security mechanism.

However, even with the risks, client-side validation is quite popular as it does provide a better user experience. The more secure form of validation is server-side validation. Server-side validation means that the validation checks are performed on the server instead of on the client. It is more secure because these checks cannot be easily bypassed. Instead, the form data values are checked using server code (C# or VB) on the server. If the form data isn't valid, the page that is sent back to the client as invalid. Although it is more secure, server side validation can be slow.

It is sluggish simply because the page has to be posted to a remote location and checked. Your end user might not be the happiest surfer in the world if, after waiting 20 seconds for a form to post, he is told his e-mail address isn't in the correct format.

ASP.NET VALIDATION SERVER CONTROLS

The available validation server controls include:

VALIDATION SERVER CONTROL	DESCRIPTION
RequiredFieldValidator	Ensures that the user does not skip a form entry field.
CompareValidator	Allows for comparisons between the user's input and another item using a comparison operator (equals, greater than, less than, and so on).
RangeValidator	Range Validator Checks the user's input based upon a lower- and upper-level range of numbers or characters.
RegularExpressionValidator	Checks that the user's entry matches a pattern defined by a regular expression. This control is good to use to check e-mail addresses and phone numbers.
CustomValidator	Checks the user's entry using custom-coded validation logic.
DynamicValidator	Dynamic Validator Works with exceptions that are thrown from entity data models and extension methods. This control is part of the ASP.NET Dynamic Data Framework. For more information about this control, be sure to search the Internet for Dynamic Validator
ValidationSummary	Displays all the error messages from the validators in one specific spot on the page.

Validation Causes

Validation doesn't just happen; it occurs in response to an event. In most cases, it is a button click event. The Button, LinkButton, and ImageButton server controls all have the capability to cause a page's form validation to initiate. This is the default behavior. Dragging and dropping a Button server control onto your form gives you the following initial result:

```
<asp:Button ID="Button1" runat="server" Text="Button" />
```

If you look through the properties of the Button control, you can see that the CausesValidation property is set to True. As stated, this behavior is the default — all buttons on the page, no matter how many there are, cause the form validation to fire. If you have multiple buttons on an ASP.NET page, and you don't want every button to initiate the form validation, you can set the CausesValidation property to False for all the buttons you want to ignore in the validation process (for example, a form's Cancel button or pressing enter in another form element such as a search box):

```
<asp:Button ID="Button1" runat="server" Text="Cancel" CausesValidation="false" />
```

Procedure:

1. Create registration form to accept user data and implement the following validations on it:

a.RequiredFieldValidator	d.RegularExpressionValidator
b.CompareValidator	e. CustomValidator
c. RangeValidator	f. ValidationSummary

2. Execute the above validations.

Conclusions: