

**Công ty Cổ phần Chứng khoán FPT**

Tài liệu Đào tạo ASP.NET

**Hà Nội, <15-03-2018>**

Record of change

\*A - Added M - Modified D - Deleted

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ngày | Hạng mục thay đổi | A\* M, D | Mô tả thay đổi | Version |
| 16-03-2018 |  | A | Tạo tài liệu | 1.0 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

trang ký

**NGƯỜI LẬP:** <Name> <Date>

<Position>

**NGƯỜI XEM XÉT:**

<Name> <Date>

<Position>

<Name, if it’s needed> <Date>

<Position>

**NGƯỜI PHÊ DUYỆT:**

<Name> <Date>

<Position>

TABLE OF CONTENTS

I. HTML 7

1. Keyword 7

2. Theory 7

2.1. What is HTML? 7

2.2. HTML Tags 7

2.3. HTML Table 9

2.4. HTML Form 11

3. Exercise 14

II. CSS 16

4. Keyword 16

5. Theory 16

5.1. What is CSS 16

5.2. CSS Syntax 16

5.3. CSS Selector 17

5.4. How to add CSS 19

5.5. CSS Properties 20

5.6. CSS Design: Form, Layout, Table 21

6. Exercise 24

III. JavaSCRIPT 26

7. Keyword 26

8. Theory 26

8.1. JS: What? Where? 26

8.2. JS Basics 27

8.3. JS Objects 37

8.4. JS BOM 40

8.5. JS Validation 44

8.6. JS Events 45

9. Exercise 46

IV. ASP.NET Webform 48

10. Keyword 48

11. Theory 48

11.1. WF Introduction 48

11.2. ASP.NET App Structure 50

11.3. WF Basic components 50

11.4. Execution Model 51

V. ASP.NET Controls 54

12. Keyword 54

13. Theory 54

13.1. Controls Class 54

13.2. HTML Server Control 56

13.3. Web Server Control 57

13.4. Validation Control 64

13.5. User Controls 69

14. Exercise 73

VI. ASP.NET Master pages, state management 75

15. Keyword 75

16. Theory 75

16.1. Master pages and Content pages 75

16.2. WF Cookie 79

16.3. WF Session 84

17. Exercise 85

VII. ADO.NET, ODP.NET 87

18. Keyword 87

19. Theory 87

19.1. ADO.NET 87

19.2. Using Stored Procedures 90

Introduction 90

Executing a Stored Procedure 90

Sending Parameters to Stored Procedures 91

19.3. OPD.NET 92

20. Exercise 94

VIII. 3 layers architecture 95

21. Keyword 95

22. Theory 95

22.1. 3 layers Architecture 95

23. Exercise 104

# HTML

## Keyword

* What is HTML?
* HTML Tags
* HTML Table
* HTML Form

## Theory

## What is HTML?

HTML is an acronym which stands for Hyper Text Markup Language. Let's see what is Hyper Text and what is Markup Language?

**Hyper Text:** Hyper Text simply means "Text within Text". A text has a link within it, is a hypertext. Every time when you click on a word which brings you to a new webpage, you have clicked on a hypertext.

**Markup language:** A markup language is a programming language that is usedmake text more interactive and dynamic. It can turn a text into images, tables, links etc.

An HTML document is made of many HTML tags and each HTML tag contains different content.

Example:

<!DOCTYPE**>**

**<html>**

**<body>**

**<h1>**Write Your First Heading**</h1>**

**<p>**Write Your First Paragraph.**</p>**

**</body>**

**</html>**

## HTML Tags

HTML tags contain three main parts: opening tag, content and closing tag. But some HTML tags are unclosed tags.

When a web browser reads an HTML document, browser reads it from top to bottom and left to right. HTML tags are used to create HTML documents and render their properties. Each HTML tags have different properties.

Syntax: <tag> content </tag>

Example:

<p> Paragraph Tag </p>

<h2> Heading Tag </h2>

<b> **Bold Tag** </b>

<i> *Italic Tag* </i>

<u> Underline Tag</u>

***Unclosed HTML Tags***

Some HTML tags are not closed, for example br and hr.

**<br> Tag**: br stands for break line, it breaks the line of the code.

**<hr> Tag**: hr stands for Horizontal Rule. This tag is used to put a line across the webpage.

***HTML Meta Tags:***DOCTYPE, title, link, meta and style

***HTML Text Tags:***

<p>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <strong>, <em>, <abbr>, <acronym>, <address>, <bdo>, <blockquote>, <cite>, <q>, <code>, <ins>, <del>, <dfn>, <kbd>, <pre>, <samp>, <var> and <br>

***HTML Link Tags:***<a> and <base>

***HTML Image and Object Tags:***<img>, <area>, <map>, <param> and <object>

***HTML List Tags:***<ul>, <ol>, <li>, <dl>, <dt> and <dd>

***HTML Table Tags:***table, tr, td, th, tbody, thead, tfoot, col, colgroup and caption

***HTML Form Tags:***form, input, textarea, select, option, optgroup, button, label, fieldset and legend

***HTML Scripting Tags:***script and noscript

## HTML Table

**HTML table tag** is used to display data in tabular form (row \* column). There can be many columns in a row.

HTML tables are used to manage the layout of the page e.g. header section, navigation bar, body content, footer section etc. But it is recommended to use div tag over table to manage the layout of the page .

|  |  |
| --- | --- |
| **Tag** | **Description** |
| <table> | It defines a table. |
| <tr> | It defines a row in a table. |
| <th> | It defines a header cell in a table. |
| <td> | It defines a cell in a table. |
| <caption> | It defines the table caption. |
| <colgroup> | It specifies a group of one or more columns in a table for formatting. |
| <col> | It is used with <colgroup> element to specify column properties for each column. |
| <tbody> | It is used to group the body content in a table. |
| <thead> | It is used to group the header content in a table. |
| <tfooter> | It is used to group the footer content in a table. |

Example:

**<table>**

**<tr><th>**First\_Name**</th><th>**Last\_Name**</th><th>**Marks**</th></tr>**

**<tr><td>**Sonoo**</td><td>**Jaiswal**</td><td>**60**</td></tr>**

**<tr><td>**James**</td><td>**William**</td><td>**80**</td></tr>**

**<tr><td>**Swati**</td><td>**Sironi**</td><td>**82**</td></tr>**

**<tr><td>**Chetna**</td><td>**Singh**</td><td>**72**</td></tr>**

**</table>**

Output:

|  |  |  |
| --- | --- | --- |
| **First\_Name** | **Last\_Name** | **Marks** |
| Sonoo | Jaiswal | 60 |
| James | William | 80 |
| Swati | Sironi | 82 |
| Chetna | Singh | 72 |

There are two ways to specify border for HTML tables.

1. By border attribute of table in HTML **<table** border="1"**>**
2. By border property in CSS

**<style>**

table, th, td {

    border: 2px solid black;

    border-collapse: collapse;

}

**</style>**

Output:

|  |  |  |
| --- | --- | --- |
| **Name** | **Last Name** | **Marks** |
| Sonoo | Jaiswal | 60 |
| James | William | 80 |
| Swati | Sironi | 82 |
| Chetna | Singh | 72 |

***colspan***

**<table** style="width:100%"**>**

**<tr>**

**<th>**Name**</th>**

**<th** colspan="2"**>**Mobile No.**</th>**

**</tr>**

**<tr>**

**<td>**Ajeet Maurya**</td>**

**<td>**7503520801**</td>**

**<td>**9555879135**</td>**

**</tr>**

**</table>**

Output:

|  |  |  |
| --- | --- | --- |
| **Name** | **Mobile No.** | |
| Ajeet Maurya | 7503520801 | 9555879135 |

## HTML Form

An **HTML form** is *a section of a document* which contains controls such as text fields, password fields, checkboxes, radio buttons, submit button, menus etc.

An HTML form facilitates the user to enter data that is to be sent to the server for processing.

Syntax:

**<form** action="server url" method="get|post"**>**

  //input controls e.g. textfield, textarea, radiobutton, button

**</form>**

|  |  |
| --- | --- |
| **Tag** | **Description** |
| <form> | It defines an HTML form to enter inputs by the used side. |
| <input> | It defines an input control. |
| <textarea> | It defines a multi-line input control. |
| <label> | It defines a label for an input element. |
| <fieldset> | It groups the related element in a form. |
| <legend> | It defines a caption for a <fieldset> element. |
| <select> | It defines a drop-down list. |
| <optgroup> | It defines a group of related options in a drop-down list. |
| <option> | It defines an option in a drop-down list. |
| <button> | It defines a clickable button. |

***HTML TextField Control***

The type="text" attribute of input tag creates textfield control also known as single line textfield control. The name attribute is optional, but it is required for the server side component such as JSP, ASP, PHP etc.

**<form>**

    First Name: **<input** type="text" name="firstname"**/>** **<br/>**

    Last Name:  **<input** type="text" name="lastname"**/>** **<br/>**

**</form>**

***Label Tag in Form***

It is considered better to have label in form. As it makes the code parser/browser/user friendly.

If you click on the label tag, it will focus on the text control. To do so, you need to have for attribute in label tag that must be same as id attribute of input tag.

**<form>**

**<label** for="firstname"**>**First Name: **</label>**

**<input** type="text" id="firstname" name="firstname"**/>** **<br/>**

**<label** for="lastname"**>**Last Name: **</label>**

**<input** type="text" id="lastname" name="lastname"**/>** **<br/>**

**</form>**

***HTML Password Field Control***

The password is not visible to the user in password field control.

**<form>**

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

**<input** type="password" id="password" name="password"**/>** **<br/>**

**</form>**

***Radio Button Control***

The radio button is used to select one from multiple options. It is used in gender, quiz questions etc.

If you use one name for all the radio buttons, only one radio button can be selected at a time.

**<form>**

**<label** for="gender"**>**Gender: **</label>**

**<input** type="radio" id="gender" name="gender" value="male"**/>**Male

**<input** type="radio" id="gender" name="gender" value="female"**/>**Female **<br/>**

**</form>**

***Checkbox Control***

The checkbox control is used to check multiple options from given checkboxes.

**<form>**

Hobby:**<br>**

**<input** type="checkbox" id="cricket" name="cricket" value="cricket"**/>**

**<label** for="cricket"**>**Cricket**</label>**

**<input** type="checkbox" id="football" name="football" value="football"**/>**

**<label** for="football"**>**Football**</label>**

**<input** type="checkbox" id="hockey" name="hockey" value="hockey"**/>**

**<label** for="hockey"**>**Hockey**</label>**

**</form>**

***HTML Form Example***

Let's see a simple example of creating HTML form.

**<form** action="#"**>**

**<table>**

**<tr>**

**<td** class="tdLabel"**><label** for="register\_name" class="label"**>**Enter name:**</label></td>**

**<td><input** type="text" name="name" value="" id="register\_name" style="width:160px"**/></td>**

**</tr>**

**<tr>**

**<td** class="tdLabel"**><label** for="register\_password" class="label"**>**Enter password:**</label></td>**

**<td><input** type="password" name="password" id="register\_password" style="width:160px"**/></td>**

**</tr>**

**<tr>**

**<td** class="tdLabel"**><label** for="register\_email" class="label"**>**Enter Email:**</label></td>**

**<td**

**><input** type="email" name="email" value="" id="register\_email" style="width:160px"**/></td>**

**</tr>**

**<tr>**

**<td** class="tdLabel"**><label** for="register\_gender" class="label"**>**Enter Gender:**</label></td>**

**<td>**

**<input** type="radio" name="gender" id="register\_gendermale" value="male"**/>**

**<label** for="register\_gendermale"**>**male**</label>**

**<input** type="radio" name="gender" id="register\_genderfemale" value="female"**/>**

**<label** for="register\_genderfemale"**>**female**</label>**

**</td>**

**</tr>**

**<tr>**

**<td** class="tdLabel"**><label** for="register\_country" class="label"**>**Select Country:**</label></td>**

**<td><select** name="country" id="register\_country" style="width:160px"**>**

**<option** value="india"**>**india**</option>**

**<option** value="pakistan"**>**pakistan**</option>**

**<option** value="africa"**>**africa**</option>**

**<option** value="china"**>**china**</option>**

**<option** value="other"**>**other**</option>**

**</select>**

**</td>**

**</tr>**

**<tr>**

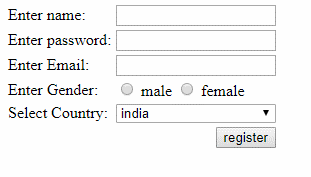
**<td** colspan="2"**><div** align="right"**><input** type="submit" id="register\_0" value="register"**/>**

**</div></td>**

**</tr>**

**</table>**

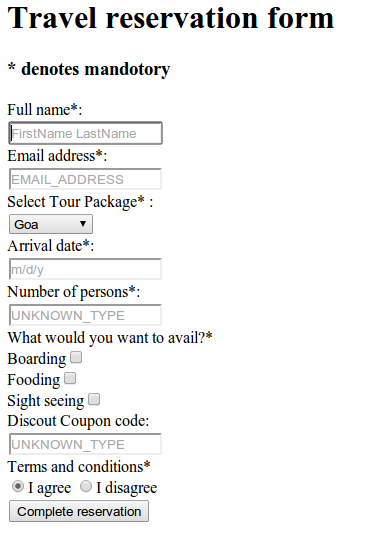
**</form>**



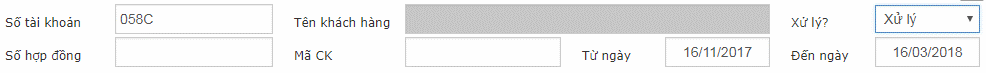
## Exercise

Design HTML pages like that pics:

Ex 1.



Ex 2.



# CSS

## Keyword

* What is CSS
* CSS Syntax
* CSS Selector
* How to add CSS
* CSS Properties
* CSS Design: Form, Layout, Table

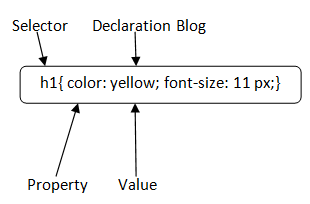
## Theory

## What is CSS

* CSS stands for Cascading Style Sheet.
* CSS is used to design HTML tags.
* CSS is a widely used language on the web.
* HTML, CSS and JavaScript are used for web designing. It helps the web designers to apply style on HTML tags.

## CSS Syntax

A CSS rule set contains a selector and a declaration block.



**Selector:** Selector indicates the HTML element you want to style. It could be any tag like <h1>, <title> etc.

**Declaration Block:** The declaration block can contain one or more declarations separated by a semicolon. For the above example, there are two declarations:

1. color: yellow;
2. font-size: 11 px;

Each declaration contains a property name and value, separated by a colon.

**Property:** A Property is a type of attribute of HTML element. It could be color, border etc.

**Value:** Values are assigned to CSS properties. In the above example, value "yellow" is assigned to color property.

Selector{Property1: value1; Property2: value2; ..........;}

## CSS Selector

**CSS selectors** are used *to select the content you want to style*. Selectors are the part of CSS rule set. CSS selectors select HTML elements according to its id, class, type, attribute etc.

There are several different types of selectors in CSS.

1. CSS Element Selector:

The element selector selects the HTML element by name.

**<style>**

p{

    text-align: center;

    color: blue;

}

**</style>**

1. CSS Id Selector:

The id selector selects the id attribute of an HTML element to select a specific element. An id is always unique within the page so it is chosen to select a single, unique element.

It is written with the hash character (#), followed by the id of the element.

Let?s take an example with the id "para1".

**<style>**

#para1 {

    text-align: center;

    color: blue;

}

**</style>**

1. CSS Class Selector

The class selector selects HTML elements with a specific class attribute. It is used with a period character . (full stop symbol) followed by the class name.

Let's take an example with a class "center".

**<style>**

.center {

    text-align: center;

    color: blue;

}

**</style>**

1. CSS Universal Selector

The universal selector is used as a wildcard character. It selects all the elements on the pages.

**<style>**

\* {

   color: green;

   font-size: 20px;

}

**</style>**

1. CSS Group Selector

The grouping selector is used to select all the elements with the same style definitions.

Grouping selector is used to minimize the code. Commas are used to separate each selector in grouping.

**<style>**

h1, h2, p {

    text-align: center;

    color: blue;

}

**</style>**

## How to add CSS

CSS is added to HTML pages to format the document according to information in the style sheet. There are three ways to insert CSS in HTML documents.

1. Inline CSS:

Inline CSS is used to apply CSS on a single line or element.

For example:

**<p** style="color:blue"**>**Hello CSS**</p>**

1. Internal CSS

Internal CSS is used to apply CSS on a single document or page. It can affect all the elements of the page. It is written inside the style tag within head section of html.

For example:

**<style>**

p{color:blue}

**</style>**

1. External CSS

External CSS is used to apply CSS on multiple pages or all pages. Here, we write all the CSS code in a css file. Its extension must be .css for example style.css.

For example:

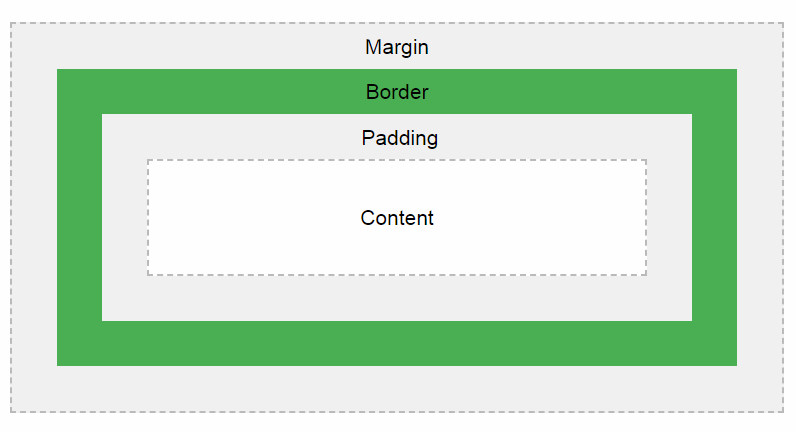
p{color:blue}

You need to link this style.css file to your html pages like this:

**<link** rel="stylesheet" type="text/css" href="style.css"**>**

The link tag must be used inside head section of html.

## CSS Properties



Explanation of the different parts:

* **Content** - The content of the box, where text and images appear
* **Padding** - Clears an area around the content. The padding is transparent
* **Border** - A border that goes around the padding and content
* **Margin** - Clears an area outside the border. The margin is transparent

Ex:

Assume we want to style a <div> element to have a total width of 350px:

div {  
    width: 320px;  
    padding: 10px;  
    border: 5px solid gray;  
    margin: 0;   
}

The total width of an element should be calculated like this:

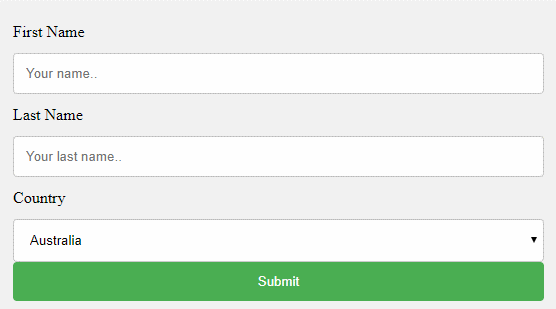
Total element width = width + left padding + right padding + left border + right border + left margin + right margin

The total height of an element should be calculated like this:

Total element height = height + top padding + bottom padding + top border + bottom border + top margin + bottom margin

## CSS Design: Form, Layout, Table

* + 1. **CSS Form**



input[type=text], select, textarea {

width: 100%;

padding: 12px;

border: 1px solid #ccc;

border-radius: 4px;

resize: vertical;

}

label {

padding: 12px 12px 12px 0;

display: inline-block;

}

input[type=submit] {

background-color: #4CAF50;

color: white;

padding: 12px 20px;

border: none;

border-radius: 4px;

cursor: pointer;

float: right;

}

input[type=submit]:hover {

background-color: #45a049;

}

* + 1. **CSS Layout**

**CSS layout** is easy to design. We can use CSS layout to design our web page such as home page, contact us, about us etc.

There are 3 ways to design layout of a web page:

1. **HTML Div with CSS**: fast and widely used now.
2. **HTML Table**: slow and less preferred.
3. **HTML Frameset**: deprecated now.

A CSS layout can have header, footer, left pane, right pane and body part. Let's see a simple example of CSS layout.

**<style>**

.header{margin:-8px -8px 0px;background-image:linear-gradient(145deg,#7379ff,#b524ef);color:white;text-align:center;padding:10px;}

.container{width:100%}

.left{width:15%;float:left;}

.body{width:65%;float:left;background-color:pink;padding:5px;}

.right{width:15%;float:left;}

.footer{margin:-8px;clear:both;background-image:linear-gradient(145deg,#7379ff,#b524ef);color:white;text-align:center;padding:10px;}

**</style>**

* + 1. **CSS Table**

We can apply style on HTML tables for better look and feel. There are some CSS properties that are widely used in designing table using CSS:

* border
* border-collapse
* padding
* width
* height
* text-align
* color
* background-color

We can style even and odd table cells for better look and feel. In this code, we are displaying different background colors on even and odd cells. Moreover, we have changed the background-color and color of <th> tag.

CSS code:

**<style>**

table, th, td {

    border: 1px solid black;

    border-collapse: collapse;

}

th, td {

    padding: 10px;

}

table#alter tr:nth-child(even) {

    background-color: #eee;

}

table#alter tr:nth-child(odd) {

    background-color: #fff;

}

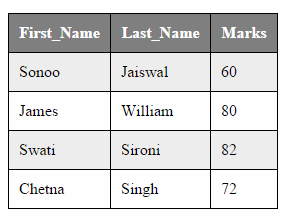
table#alter th {

    color: white;

    background-color: gray;

}

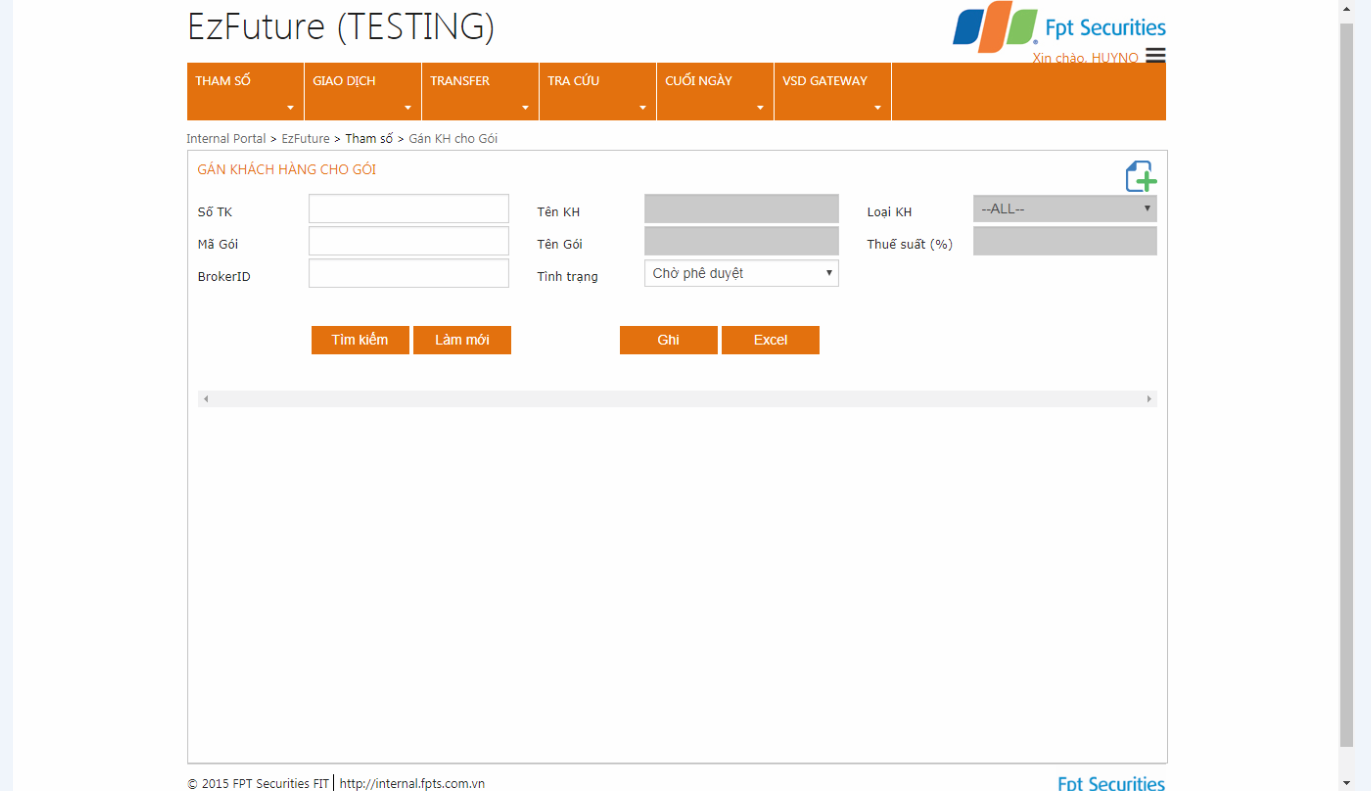
**</style>**



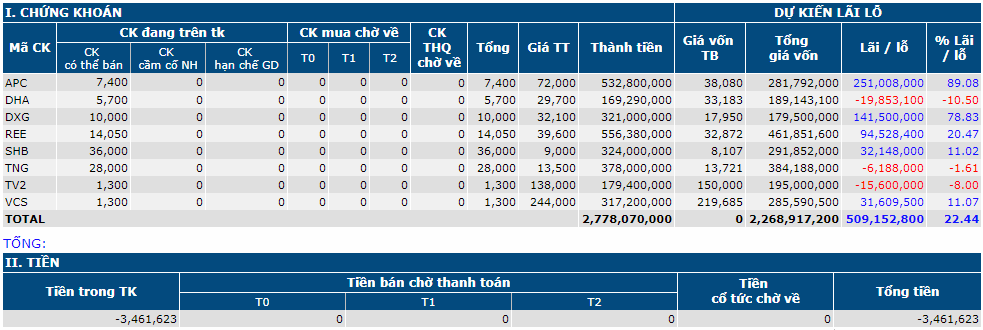
## Exercise

Design HTML and CSS:

Ex1: HTML, CSS form & layout



Ex2: HTML, CSS form Table



# JavaSCRIPT

## Keyword

* JavaScript: what? Where?
* JS Basics: Variable, DataTypes, If, Switch, Loop, Function
* JS Objects: Object, Array, String, Date, Math, Number, Boolean
* JS BOM: Window Object, History Obj, Navigator Obj, Screen Obj
* JS DOM: Document Obj, GetElementById/Name/TagName, innerHTML, innerText
* JS Validation: form validation
* JS Events: HTML/DOM Events

## Theory

## JS: What? Where?

* What: JavaScript is an object-based scripting language that is lightweight and cross-platform.
* Where to put JS code: within body tag, within head tag and external JavaScript file.

Let’s create an external JavaScript file that prints Hello JS in a alert dialog box.

**message.js**

function msg(){

 alert("Hello JS");

}

Let’s include the JavaScript file into html page. It calls the JavaScript function on button click.

**index.html**

**<html>**

**<head>**

**<script** type="text/javascript" src="message.js"**></script>**

**</head>**

**<body>**

**<p>**Welcome to JavaScript**</p>**

**<form>**

**<input** type="button" value="click" onclick="msg()"**/>**

**</form>**

**</body>**

**</html>**

## JS Basics

* + 1. **JS Comments:**

There are two types of comments in JavaScript.

1. Single-line Comment: It is represented by double forward slashes (//). It can be used before and after the statement.
2. Multi-line Comment: It can be used to add single as well as multi line comments. So, it is more convenient.

/\* your code here  \*/

* + 1. **JS Variables:**

A **JavaScript variable** is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.

There are some rules while declaring a JavaScript variable (also known as identifiers).

1. Name must start with a letter (a to z or A to Z), underscore( \_ ), or dollar( $ ) sign.
2. After first letter we can use digits (0 to 9), for example value1.
3. JavaScript variables are case sensitive, for example x and X are different variables.

Ex: var x = 10;

var \_value="sonoo";

1. Local variable

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only.

Ex:

**<script>**

function abc(){

var x=10;//local variable

}

**</script>**

Or,

**<script>**

If(10**<13**){

var y=20;//JavaScript local variable

}

**</script>**

1. Global variable

A **JavaScript global variable** is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable.

Ex:

**<script>**

var value=50;//global variable

function a(){

alert(value);

}

function b(){

alert(value);

}

**</script>**

* + 1. **JS Data Types**

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| String | represents sequence of characters e.g. "hello" |
| Number | represents numeric values e.g. 100 |
| Boolean | represents boolean value either false or true |
| Undefined | represents undefined value |
| Null | represents null i.e. no value at all |

1. Non-primitive (reference) data type

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| Object | represents instance through which we can access members |
| Array | represents group of similar values |
| RegExp | represents regular expression |

* + 1. **JS Operators**

JavaScript operators are symbols that are used to perform operations on operands.

There are following types of operators in JavaScript.

1. Arithmetic Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | Addition | 10+20 = 30 |
| - | Subtraction | 20-10 = 10 |
| \* | Multiplication | 10\*20 = 200 |
| / | Division | 20/10 = 2 |
| % | Modulus (Remainder) | 20%10 = 0 |
| ++ | Increment | var a=10; a++; Now a = 11 |
| -- | Decrement | var a=10; a--; Now a = 9 |

1. Comparison (Relational) Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == | Is equal to | 10==20 = false |
| === | Identical (equal and of same type) | 10==20 = false |
| != | Not equal to | 10!=20 = true |
| !== | Not Identical | 20!==20 = false |
| > | Greater than | 20>10 = true |
| >= | Greater than or equal to | 20>=10 = true |
| < | Less than | 20<10 = false |
| <= | Less than or equal to | 20<=10 = false |

1. Bitwise Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| & | Bitwise AND | (10==20 & 20==33) = false |
| | | Bitwise OR | (10==20 | 20==33) = false |
| ^ | Bitwise XOR | (10==20 ^ 20==33) = false |
| ~ | Bitwise NOT | (~10) = -10 |
| << | Bitwise Left Shift | (10<<2) = 40 |
| >> | Bitwise Right Shift | (10>>2) = 2 |
| >>> | Bitwise Right Shift with Zero | (10>>>2) = 2 |

1. Logical Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| && | Logical AND | (10==20 && 20==33) = false |
| || | Logical OR | (10==20 || 20==33) = false |
| ! | Logical Not | !(10==20) = true |

1. Assignment Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Assign | 10+10 = 20 |
| += | Add and assign | var a=10; a+=20; Now a = 30 |
| -= | Subtract and assign | var a=20; a-=10; Now a = 10 |
| \*= | Multiply and assign | var a=10; a\*=20; Now a = 200 |
| /= | Divide and assign | var a=10; a/=2; Now a = 5 |
| %= | Modulus and assign | var a=10; a%=2; Now a = 0 |

1. Special Operators

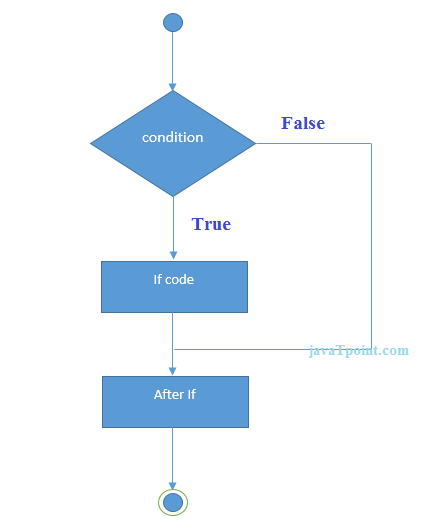
|  |  |
| --- | --- |
| **Operator** | **Description** |
| (?:) | Conditional Operator returns value based on the condition. It is like if-else. |
| , | Comma Operator allows multiple expressions to be evaluated as single statement. |
| delete | Delete Operator deletes a property from the object. |
| in | In Operator checks if object has the given property |
| instanceof | checks if the object is an instance of given type |
| new | creates an instance (object) |
| typeof | checks the type of object. |
| void | it discards the expression's return value. |
| yield | checks what is returned in a generator by the generator's iterator. |

* + 1. **JS If**

The **JavaScript if-else statement** is used *to execute the code whether condition is true or false*. There are three forms of if statement in JavaScript.

1. If Statement

It evaluates the content only if expression is true.



var a=20;

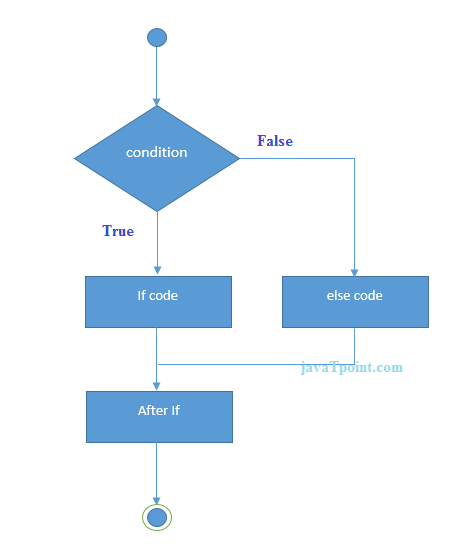
if(a**>**10){

document.write("value of a is greater than 10");

}

1. If else statement

It evaluates the content whether condition is true of false.



var a=20;

if(a%2==0){

document.write("a is even number");

}

else{

document.write("a is odd number");

}

1. if else if statement

It evaluates the content only if expression is true from several expressions.

**<script>**

var a=20;

if(a==10){

document.write("a is equal to 10");

}

else if(a==15){

document.write("a is equal to 15");

}

else if(a==20){

document.write("a is equal to 20");

}

else{

document.write("a is not equal to 10, 15 or 20");

}

**</script>**

* + 1. **JS Switch**

The **JavaScript switch statement** is used to execute one code from multiple expressions. It is just like else if statement that we have learned in previous page. But it is convenient than if..else..if because it can be used with numbers, characters etc.

**<script>**

var grade='B';

var result;

switch(grade){

case 'A':

result="A Grade";

break;

case 'B':

result="B Grade";

break;

case 'C':

result="C Grade";

break;

default:

result="No Grade";

}

document.write(result);

**</script>**

* + 1. **JS Loop**

The **JavaScript loops** are used *to iterate the piece of code* using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.

There are four types of loops in JavaScript.

1. for loop

The **JavaScript for loop** *iterates the elements for the fixed number of times*. It should be used if number of iteration is known. The syntax of for loop is given below.

for (initialization; condition; increment)

{

    code to be executed

}

Let’s see the simple example of for loop in javascript.

**<script>**

for (i=1; i**<**=5; i++)

{

document.write(i + "**<br/>**")

}

**</script>**

1. while loop

The **JavaScript while loop** *iterates the elements for the infinite number of times*. It should be used if number of iteration is not known. The syntax of while loop is given below.

while (condition)

{

    code to be executed

}

Let’s see the simple example of while loop in javascript.

**<script>**

var i=11;

while (i**<**=15)

{

document.write(i + "**<br/>**");

i++;

}

**</script>**

1. do-while loop

The **JavaScript do while loop** *iterates the elements for the infinite number of times* like while loop. But, code is *executed at least* once whether condition is true or false. The syntax of do while loop is given below.

do{

    code to be executed

}while (condition);

Let’s see the simple example of do while loop in javascript.

**<script>**

var i=21;

do{

document.write(i + "**<br/>**");

i++;

}while (i**<**=25);

**</script>**

1. for-in loop
   * 1. **JS Function**

**JavaScript functions** are used to perform operations. We can call JavaScript function many times to reuse the code.

There are mainly two advantages of JavaScript functions.

1. **Code reusability**: We can call a function several times so it save coding.
2. **Less coding**: It makes our program compact. We don’t need to write many lines of code each time to perform a common task.

**The syntax** of declaring function is given below.

function functionName([arg1, arg2, ...argN]){

 //code to be executed

}

JavaScript Functions can have 0 or more arguments.

We can call function that returns a value and use it in our program.

## JS Objects

* + 1. **JS Object**
    2. **JS Array**

**JavaScript array** is an object that represents a collection of similar type of elements.

There are 3 ways to construct array in JavaScript

1. By array literal:

var arrayname=[value1,value2.....valueN];

ex: var emp=["Sonoo","Vimal","Ratan"];

1. By creating instance of Array directly (using new keyword):

var arrayname=new Array();

ex: var emp=new Array();

emp[0] = "Arun";

…

1. By using an Array constructor (using new keyword):

var emp=new Array("Jai","Vijay","Smith");

* + 1. **JS String**

The **JavaScript string** is an object that represents a sequence of characters.

There are 2 ways to create string in JavaScript

1. By string literal:

var stringname="string value";

1. By string object (using new keyword):

var stringname=new String("string literal");

**JS String method:**

* charAt(index): returns the character at the given index
* concat(str): concatenates or joins two strings
* indexOf(str): returns the index position of the given string
* lastIndexOf(str): returns the last index position of the given string
* toLowerCase(): returns the given string in lowercase letters
* toUpperCase(): returns the given string in uppercase letters
* slice(beginIndex, endIndex): returns the parts of string from given beginIndex to endIndex.
* trim(): removes leading and trailing whitespaces from the string
  + 1. **JS Date**

The **JavaScript date** object can be used to get year, month and day. You can display a timer on the webpage by the help of JavaScript date object.

You can use different Date constructors to create date object. It provides methods to get and set day, month, year, hour, minute and seconds.

You can use 4 variant of **Date constructor** to create date object.

1. Date()
2. Date(milliseconds)
3. Date(dateString)
4. Date(year, month, day, hours, minutes, seconds, milliseconds)

The important methods of date object are as follows:

|  |  |
| --- | --- |
| **Method** | **Description** |
| getFullYear() | returns the year in 4 digit e.g. 2015. It is a new method and suggested than getYear() which is now deprecated. |
| getMonth() | returns the month in 2 digit from 0 to 11. So it is better to use getMonth()+1 in your code. |
| getDate() | returns the date in 1 or 2 digit from 1 to 31. |
| getDay() | returns the day of week in 1 digit from 0 to 6. |
| getHours() | returns all the elements having the given name value. |
| getMinutes() | returns all the elements having the given class name. |
| getSeconds() | returns all the elements having the given class name. |
| getMilliseconds() | returns all the elements having the given tag name. |

* + 1. **JS Math**

The **JavaScript math** object provides several constants and methods to perform mathematical operation. Unlike date object, it doesn't have constructors.

The JavaScript **math.sqrt(n)** method returns the square root of the given number.

The JavaScript **math.random()** method returns the random number between 0 to 1.

The JavaScript **math.pow(m,n)** method returns the m to the power of n that is mn.

The JavaScript **math.floor(n)** method returns the lowest integer for the given number. For example 3 for 3.7, 5 for 5.9 etc.

The JavaScript **math.ceil(n)** method returns the largest integer for the given number. For example 4 for 3.7, 6 for 5.9 etc.

The JavaScript **math.round(n)** method returns the rounded integer nearest for the given number. If fractional part is equal or greater than 0.5, it goes to upper value 1 otherwise lower value 0. For example 4 for 3.7, 3 for 3.3, 6 for 5.9 etc.

The JavaScript **math.abs(n)** method returns the absolute value for the given number. For example 4 for -4, 6.6 for -6.6 etc.

* + 1. **JS Number**

The **JavaScript number** object *enables you to represent a numeric value*. It may be integer or floating-point. JavaScript number object follows IEEE standard to represent the floating-point numbers.

By the help of Number() constructor, you can create number object in JavaScript. For example:

var n=new Number(value);

If value can't be converted to number, it returns NaN(Not a Number) that can be checked by isNaN() method.

You can direct assign a number to a variable also. For example:

var x=102;//integer value

var y=102.7;//floating point value

var z=13e4;//exponent value, output: 130000

var n=new Number(16);//integer value by number object

|  |  |
| --- | --- |
| **Methods** | **Description** |
| toExponential(x) | displays exponential value. |
| toFixed(x) | limits the number of digits after decimal value. |
| toPrecision(x) | formats the number with given number of digits. |
| toString() | converts number into string. |
| valueOf() | coverts other type of value into number. |

* + 1. **JS Boolean**

**JavaScript Boolean** is an object that represents value in two states: *true* or *false*. You can create the JavaScript Boolean object by Boolean() constructor as given below.

Boolean b=new Boolean(value);

The default value of JavaScript Boolean object is *false*.

## JS BOM

* + 1. **Browser Objects**

The **Browser Object Model** (BOM) is used to interact with the browser.

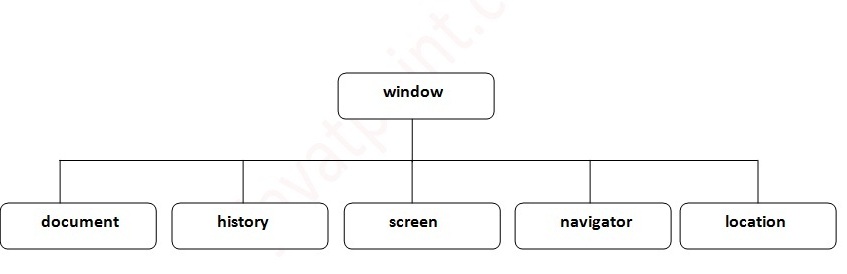
The default object of browser is window means you can call all the functions of window by specifying window or directly. For example:

window.alert("hello javatpoint");

is same as:

alert("hello javatpoint");

You can use a lot of properties (other objects) defined underneath the window object like document, history, screen, navigator, location, innerHeight, innerWidth,



* + 1. **Window Object**

The **window object** represents a window in browser. An object of window is created automatically by the browser.

Window is the object of browser, **it is not the object of javascript**. The javascript objects are string, array, date etc.

The important methods of window object are as follows:

|  |  |
| --- | --- |
| **Method** | **Description** |
| alert() | displays the alert box containing message with ok button.  alert("Hello Alert Box"); |
| confirm() | displays the confirm dialog box containing message with ok and cancel button.   1. var v= confirm("Are u sure?"); 2. if(v==true){ 3. alert("ok"); 4. } 5. else{ 6. alert("cancel"); 7. } |
| prompt() | displays a dialog box to get input from the user.   1. var v= prompt("Who are you?"); 2. alert("I am "+v); |
| open() | opens the new window.   1. open("http://www.javatpoint.com"); |
| close() | closes the current window. |
| setTimeout() | performs action after specified time like calling function, evaluating expressions etc.   1. setTimeout( 2. function(){ 3. alert("Welcome to JS after 2 seconds") 4. },2000); |

* + 1. **History Object**

The **JavaScript history object** represents an array of URLs visited by the user. By using this object, you can load previous, forward or any particular page.

The history object is the window property, so it can be accessed by:

window.history

Or,

history

* + 1. **Navigator Object**

The **JavaScript navigator object** is used for browser detection. It can be used to get browser information such as appName, appCodeName, userAgent etc.

The navigator object is the window property, so it can be accessed by:

window.navigator

Or,

navigator

* + 1. **Screen Object**

he **JavaScript screen object** holds information of browser screen. It can be used to display screen width, height, colorDepth, pixelDepth etc.

The navigator object is the window property, so it can be accessed by:

window.screen

Or,

screen

* + 1. **Document Object**

The **document object** represents the whole html document.

When html document is loaded in the browser, it becomes a document object. It is the **root element** that represents the html document. It has properties and methods. By the help of document object, we can add dynamic content to our web page.

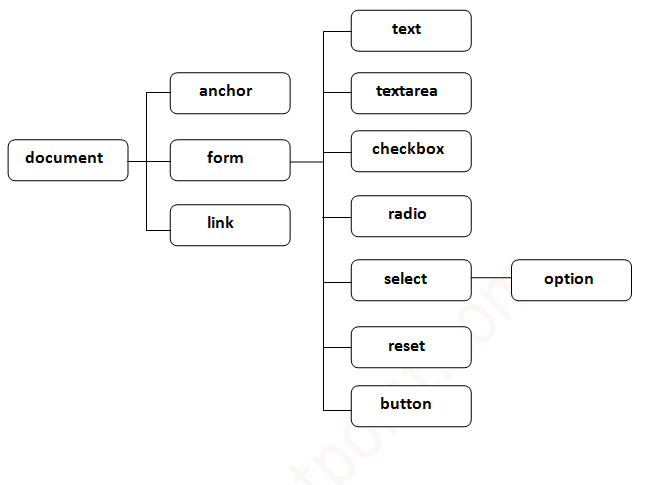
As mentioned earlier, it is the object of window. So

window.document

Is same as

document

Let's see the **properties of document object** that can be accessed and modified by the document object.



We can access and change the contents of document by its methods.

The important **methods of document object** are as follows:

|  |  |
| --- | --- |
| **Method** | **Description** |
| write("string") | writes the given string on the doucment. |
| writeln("string") | writes the given string on the doucment with newline character at the end. |
| getElementById() | returns the element having the given id value. |
| getElementsByName() | returns all the elements having the given name value. |
| getElementsByTagName() | returns all the elements having the given tag name. |
| getElementsByClassName() | returns all the elements having the given class name. |

## JS Validation

It is important to validate the form submitted by the user because it can have inappropriate values. So validation is must.

The JavaScript provides you the facility the validate the form on the client side so processing will be fast than server-side validation. So, most of the web developers prefer JavaScript form validation.

Through JavaScript, we can validate name, password, email, date, mobile number etc fields.

Ex: In this example, we are going to validate the name and password. The name can’t be empty and password can’t be less than 6 characters long.

Here, we are validating the form on form submit. The user will not be forwarded to the next page until given values are correct.

**<script>**

function validateform(){

var name=document.myform.name.value;

var password=document.myform.password.value;

if (name==null || name==""){

  alert("Name can't be blank");

  return false;

}else if(password.length**<6**){

  alert("Password must be at least 6 characters long.");

  return false;

  }

}

**</script>**

**<body>**

**<form** name="myform" method="post" action="abc.jsp" onsubmit="return validateform()" **>**

Name: **<input** type="text" name="name"**><br/>**

Password: **<input** type="password" name="password"**><br/>**

**<input** type="submit" value="register"**>**

**</form>**

Validate the textfield for numeric value only. Here, we are using isNaN() function.

if (isNaN(num)){

    return false;

}else{

  return true;

  }

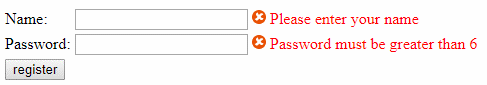
## JS Events

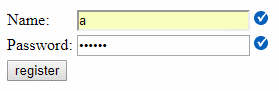
HTML or DOM events are widely used in JavaScript code. JavaScript code is executed with HTML/DOM events. So before learning JavaScript, let’s have some idea about events.

|  |  |
| --- | --- |
| **Events** | **Description** |
| onclick | occurs when element is clicked. |
| ondblclick | occurs when element is double-clicked. |
| onfocus | occurs when an element gets focus such as button, input, textarea etc. |
| onblur | occurs when form looses the focus from an element. |
| onsubmit | occurs when form is submitted. |
| onmouseover | occurs when mouse is moved over an element. |
| onmouseout | occurs when mouse is moved out from an element (after moved over). |
| onmousedown | occurs when mouse button is pressed over an element. |
| onmouseup | occurs when mouse is released from an element (after mouse is pressed). |
| onload | occurs when document, object or frameset is loaded. |
| onunload | occurs when body or frameset is unloaded. |
| onscroll | occurs when document is scrolled. |
| onresized | occurs when document is resized. |
| onreset | occurs when form is reset. |
| onkeydown | occurs when key is being pressed. |
| onkeypress | occurs when user presses the key. |
| onkeyup | occurs when key is released. |

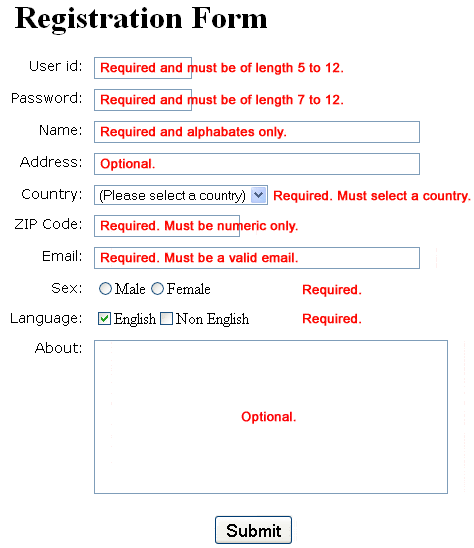
## Exercise

Ex 1: JavaScript form validation example that displays correct and incorrect image if input is correct or incorrect.





Ex 2:



# ASP.NET Webform

## Keyword

* WF Introduction
* App Structure
* Basic components
* Page Execution Model

## Theory

## WF Introduction

ASP.NET is a web framework designed and developed by Microsoft. It is used to develop websites, web applications and web services. It provides fantastic integration of HTML, CSS and JavaScript. It was first released in January 2002. It is built on the Common Language Runtime (CLR) and allows programmers to write code using any supported .NET language.

ASP.NET provides three development styles for creating web applications:

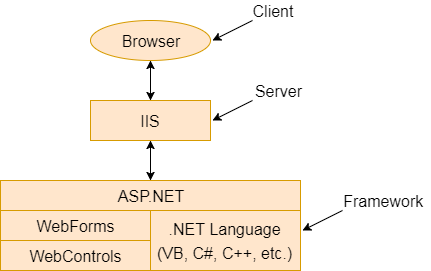
1. Web Forms
2. ASP.NET MVC
3. ASP.NET Web Pages

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Skills** | **Development style** | **Experience** |
| Web Forms | Win Forms, WPF, .NET | Rapid development using a rich library of controls that encapsulate HTML markup | Mid-Level, Advanced RAD |
| MVC | Ruby on Rails, .NET | Full control over HTML markup, code and markup separated, and easy to write tests. The best choice for mobile and single-page applications (SPA). | Mid-Level, Advanced |
| Web Pages | Classic ASP, PHP | HTML markup and your code together in the same file | New, Mid-Level |

Web Forms are web pages built on the ASP.NET Technology. It executes on the server and generates output to the browser. It is compatible to any browser to any language supported by .NET common language runtime. It is flexible and allows us to create and add custom controls.

We can use Visual Studio to create ASP.NET Web Forms. It is an IDE (Integrated Development Environment) that allows us to drag and drop server controls to the web forms. It also allows us to set properties, events and methods for the controls. To write business logic, we can choose any .NET language like: Visual Basic or Visual C#.

Web Forms are made up of two components: the visual portion (the ASPX file), and the code behind the form, which resides in a separate class file.



**Fig:** This diagram shows the components of the ASP.NET

The main purpose of Web Forms is to overcome the limitations of ASP and separate view from the application logic.

## ASP.NET App Structure

**Web.config** is web app's configuration file

* + Holds settings like DB connection strings, HTTP handlers, modules, assembly bindings
  + Can hold custom application settings, e.g. credentials for external services
  + Changes in **Web.config** do not require rebuild
* You may have several **Web.config** files
  + One global for the application
  + Several for different folder in the application
* **Web.config** inherits from the global **Web.config** and from **machine.config**
  + Global settings for all applications on the server
* **Web.Debug.config**
  + Local settings for debugging
  + E.g. local database instance for testing
* **Web.Release.config**
  + Production settings for real world deployment

**Global.asax** defines the HTTP application

* Defines global application events like
  + **Application\_Start**
  + **Application\_BeginRequest**
  + **Application\_EndRequest**
  + **Application\_Error**
  + **…**
* Typically invokes **BundleConfig**, **RouteConfig**, **FilterConfig**, etc.

**Content** and **Content\themes**

**Scripts**, **img**,**fonts**

**Models** / **BLL** / **DALs**

**Site.Master**

## WF Basic components

* **Web Forms**
  + Acts as a Web-based user interface (UI) of the ASP.NET Web Forms applications
  + XML-based language, like XHTML
  + Consists of HTML, C# code and controls
  + Executed at the server-side by ASP.NET
  + Rendered to HTML by the ASP.NET runtime
  + Have complex execution model (many steps)
  + The functionality of the Web form is defined by attributes: **Page**, **Codebehind**, **Inherits**

<%@ Page Language="C#"

Codebehind="TestWebForm.aspx.cs"

Inherits="FirstApp.TestWebForm" %>

<!DOCTYPE html>

<html>

<head><title>First WebForm</title></head>

<form ID="FormTest" runat="server" …>

<asp:Label ID="lbl" runat="server">…</asp:Label>

<asp:TextBox ID="textCustomerName" runat="server"

Text="Customer Name: ">…</asp:TextBox>

<asp:Button ID="btn" runat="server" …></asp:Button>

</form>

</html>

Always put \*\*ID="…"\*\* and \*\*runat="server"\*\* for the ASP.NET controls!

* **Web Control**
  + The smallest part we can use in our Web application (e.g. text box)
  + Deliver fast and easy component-oriented development process
  + HTML abstraction, but has server-side properties and events
  + Rendered as HTML (+ CSS + scripts)

<asp:Button runat="server" ID="btn"

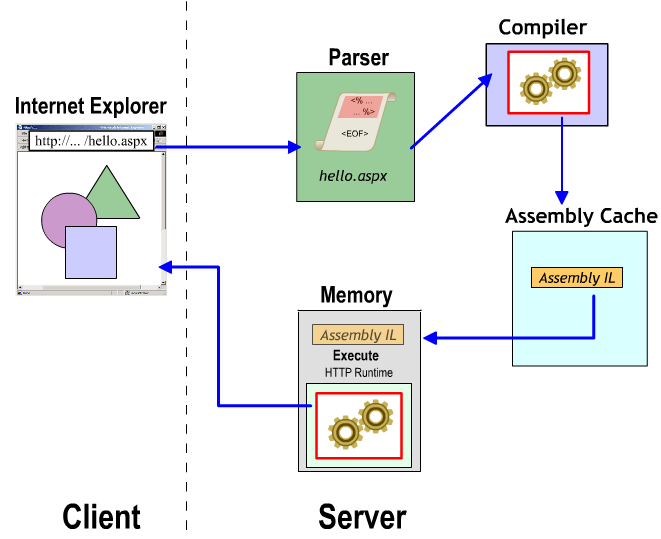
Text="Click me!" OnClick="btn\_Click" />

* **"Code behind"**
  + Contains the server-side C# code behind pages
* **Web.config**

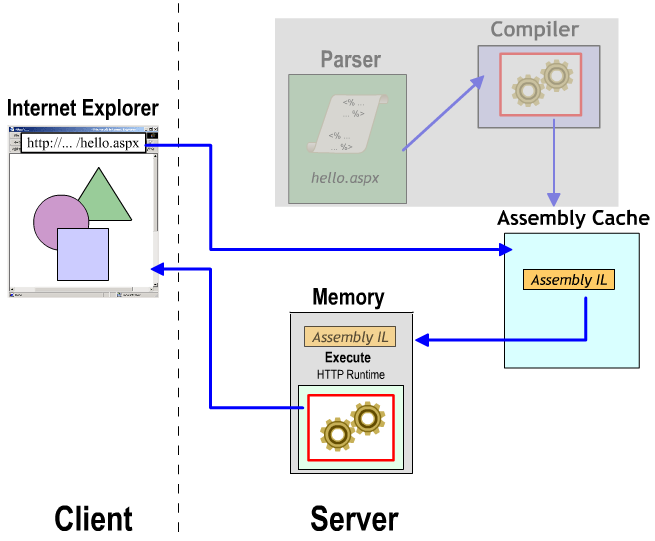
Contains ASP.NET application configuration

## Execution Model

* First call to particular page

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/02.%20ASP.NET-Web-Forms-Intro/imgs/pic21.png)

* Any other call after the first

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/02.%20ASP.NET-Web-Forms-Intro/imgs/pic22.png)

# ASP.NET Controls

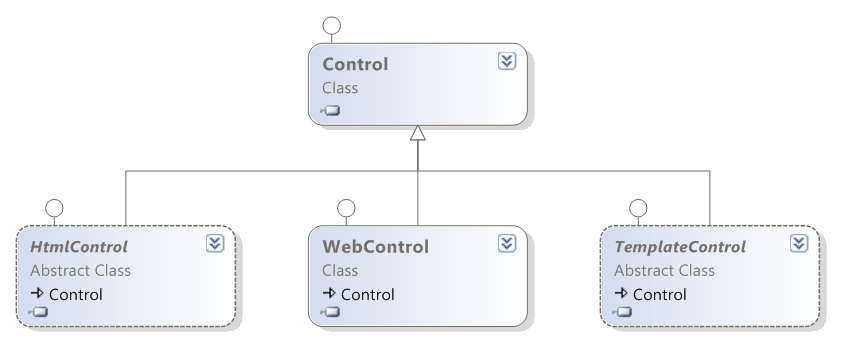
## Keyword

* Controls Class
* HTML Server Control
* Web Server Control
* Validation Control
* Web User Control
* Web Custom Control

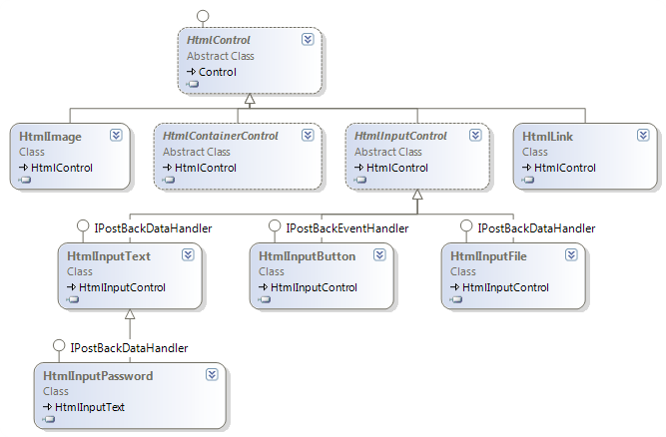
## Theory

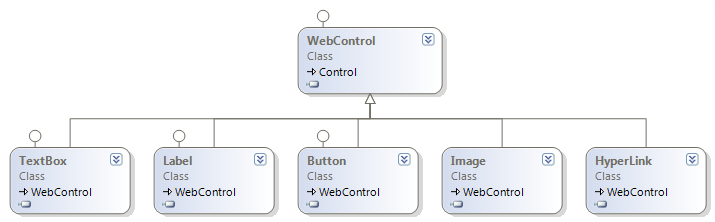
## Controls Class

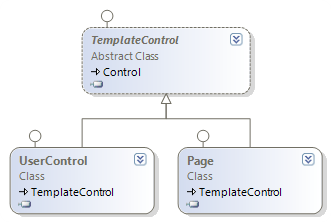
* System.Web.UI.Control
  + Base class for all controls
  + Properties – ID, Page, Context, ViewState, ClientIDMode, Controls , Parent, Visible
  + Methods – Render(HtmlTextWriter writer)

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/03.%20ASP.NET-Web-Controls-and-HTML-Controls/imgs/controls-class-hierarchy.png)

* System.Web.UI.HtmlControls.HtmlControl

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/03.%20ASP.NET-Web-Controls-and-HTML-Controls/imgs/controls-class-hierarchy-2.png)

* System.Web.UI.WebControls.WebControl [](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/03.%20ASP.NET-Web-Controls-and-HTML-Controls/imgs/controls-class-hierarchy-3.png)
* System.Web.UI.TemplateControl

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/03.%20ASP.NET-Web-Controls-and-HTML-Controls/imgs/controls-class-hierarchy-4.png)

## HTML Server Control

* HTML server controls are very simple extension of Control class
* Look like traditional HTML
  + Defined by runat="server"
  + Simple HTML seems like text on the server
  + If an HTML element is converted to HTML server control, a server side object is associated with it
* Valid only inside a Web form tag:
* <form runat="server">…</form>
* HTML Server Control Classes:
  + HtmlForm – <form>…</form>
  + HtmlInputText – <input type="text">
  + HtmlButton – <input type="button" />
    - HtmlAnchor – <a href="…">…</a>
  + HtmlSelect – <input type="select">
  + HtmlTable, HtmlTableCell, HtmlTableRow – <table><tr><td>…</td></tr></table>
  + HtmlImage – <img src="…" />
  + ...
* Example:

<%@ Page Language="C#" %>

<script language="c#" runat="server">

void ButtonSubmit\_Click(Object sender, EventArgs e) {

Response.Write("Value:<b>"+TextField.Value+"</b>");

}

<script>

<html>

<head><title>HTML Server Controls</title></head>

<body>

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

<input id="TextField" type="text" runat="server" />

<input id="ButtonSubmit" type="button"

runat="server" value="Submit"

onserverclick="ButtonSubmit\_Click" />

</form>

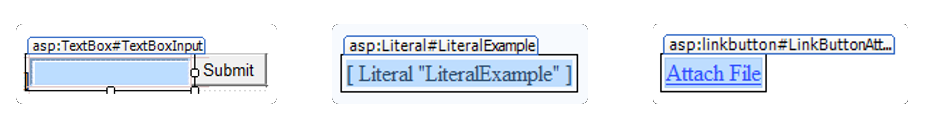
</body>

</html>

## Web Server Control

**13.3.1. Web Server Control**

* Web server controls are server UI controls that abstract the common HTML elements
  + Have own lifecycle and functionality
* Built-in with .NET Framework
  + Located in System.Web.UI.WebControls namespace
  + Inherit from the WebControl class
* The rendered HTML tags are quite different from the design-time markup
* Features:
  + Rich functionality
  + Type-safe programming capabilities
  + Automatic Web browser detection
  + AutoPostBack
    - Submit when the focus is lost
  + Support for themes

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/03.%20ASP.NET-Web-Controls-and-HTML-Controls/imgs/web-server-controls-features.png)

Example:

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

<asp:Label ID="LabelResult" runat="server"

Text="" Visible="false" />

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

<asp:Button ID="ButtonSubmit" runat="server"

Text="Submit" OnClick="ButtonSubmit\_Click" />

</form>

protected void ButtonSubmit\_Click(

object sender, EventArgs e)

{

this.LabelResult.Text =

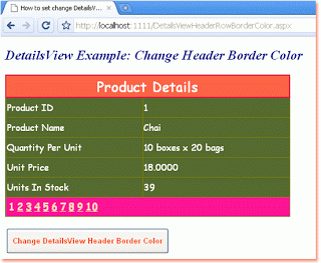
"You entered: " + this.TextBoxInput.Text;

this.LabelResult.Visible = true;

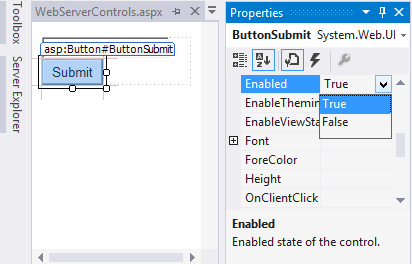
}

**13.3.2. System.Web.UI.WebControls.WebControl**

* The WebControl class defines properties, events and methods for all Web controls
* Control the appearance
  + BackColor
  + ForeColor
  + BorderWidth
  + BorderStyle
  + BorderColor

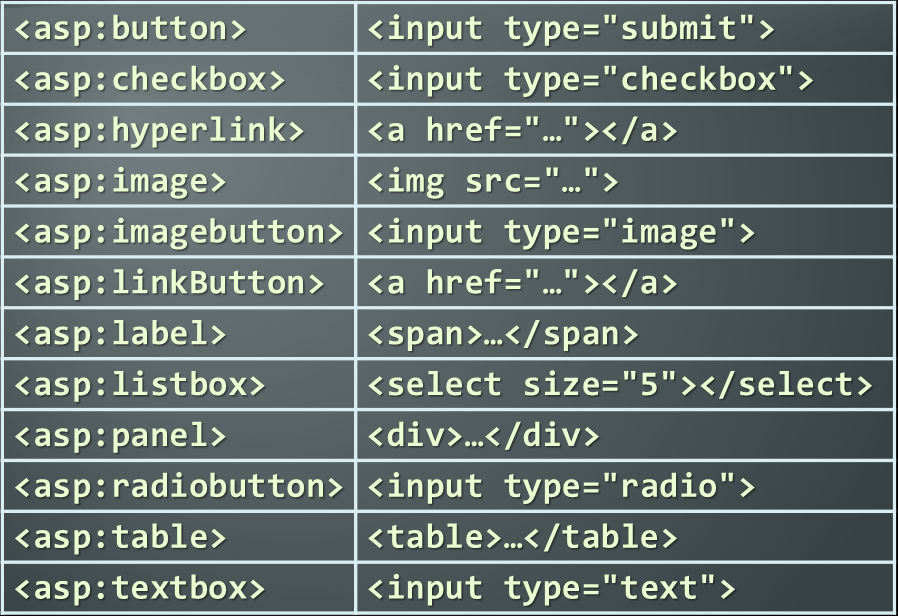
[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/03.%20ASP.NET-Web-Controls-and-HTML-Controls/imgs/web-control-example.png)

* Control the behavior
  + Enabled
  + Visible
  + TabIndex
  + ToolTip
  + …
  + Not all controls support all these properties
    - See the documentation for details

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/03.%20ASP.NET-Web-Controls-and-HTML-Controls/imgs/web-control-example-2.png)

**13.3.3. Basic Web Control**

* Basic Web Controls ↔ HTML

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/03.%20ASP.NET-Web-Controls-and-HTML-Controls/imgs/web-controls-html.png)

* Textbox
  + Creates single-line or multiline text-box
  + Lets the user to enter text
  + Properties
    - Text
    - TextMode – SingleLine, MultiLine, Password
    - MaxLength
    - ReadOnly
    - AutoPostBack
  + Events
    - TextChanged – combined with AutoPostBack
* Label
* Display static text in a <label> input
* Allows programmatically to manipulate it
* Properties
  + Text

[https://github.com/TelerikAcademy/ASP.NET-Web-Forms/raw/master/03.%20ASP.NET-Web-Controls-and-HTML-Controls/imgs/label-text.png](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/03.%20ASP.NET-Web-Controls-and-HTML-Controls/imgs/label-text.png)

* Buttons
* Implement IButtonControl
  + Button, LinkButton, RadioButton, …
* Properties
  + Value – button's title
  + CommandName – pass a command
  + CommandArgument – pass command arguments
  + PostBackUrl – posts back to specified page
  + CausesValidation – perform validation or not
  + ValidationGroup – which validation group to be validated
* Button Events
* Click
* Command
  + CommandName and CommandArgument are passed to the C# code at the server-side
* Different button types
  + Standard button
    - System.Web.UI.WebControls.Button
    - Submits the form by default
    - Has a command name associated(CommandName property)
    - Programmatically determine which button is clicked in the Command event handlers

<asp:Button />

<asp:Button ID="ButtonOK" runat="server" Text="Click here …" />

* + Link button
    - Same functionality as Button
    - Renders as hyperlink
    - Use Hyperlink if you want to link to another page
    - Renders JavaScript on the client browser

<asp:LinkButton ID="ButtonHomePage" runat="server"

Text="Home Page" />

* + ImageButton
    - Display an image that responds on mouse click
    - ImageURL – URL to displayed image
    - Both Click and Command events are raised

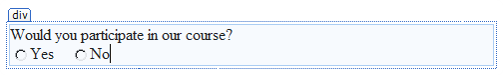
<asp:ImageButton ID="ButtonSubscribe" runat="server"

ImageUrl="~/img/btn\_subscribe.png" />

* CheckBox
* Select between checked / unchecked
* Properties
  + Checked
  + Text – control caption
  + AutoPostBack
    - Automatically posts back the page when control state is changed

<asp:CheckBox ID="CheckBoxAgree" runat="server" />

* CausesValidation – whether validation is performed
* ValidationGroup – which validation group to be validated
* Events
  + CheckChanged
* RadioButton
* Creates a radio button on the Web Forms page
* Properties
  + Text
  + GroupName – allow a mutually exclusive selection from the group
  + AutoPostBack
    - Automatically posts back the page when control state is changed
* See also RadioButtonList

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/03.%20ASP.NET-Web-Controls-and-HTML-Controls/imgs/radiobutton.png)

* PlaceHolder
  + The PlaceHolder control
  + Reserves a space in the page control hierarchy
    - Used to add controls to the page at runtime
    - Does not produce any visible output
  + The Controls properties
    - Use it to add, insert or remove controls from PlaceHolder Control

[https://github.com/TelerikAcademy/ASP.NET-Web-Forms/raw/master/03.%20ASP.NET-Web-Controls-and-HTML-Controls/imgs/placeholder.png](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/03.%20ASP.NET-Web-Controls-and-HTML-Controls/imgs/placeholder.png)

**13.3.4. Validation Control**

* The ASP.NET Web forms validation controls
  + Validate the values that are entered into other controls of the page (e.g. in a TextBox)
* Most important validation controls:
  + RequiredFieldValidator
  + RangeValidator
  + CompareValidator
  + RegularExpressionValidator
  + ValidationSummary

**13.3.5. List Controls**

* List Web controls
  + Display list of items, e.g. table of rows
  + Support binding to a collection
  + Display rows of data in templated format
* Expose data binding properties: DataSourceID, DataSource, DataMember
  + Bind to collection that support IEnumerable, ICollection or IListSource
  + Can bind to databases through Entity Framework
* ListBox
* The ListBox control
  + Holds a list of items
  + Each item has text and value
  + SelectionMode: Single or Multiple
  + Items can be data-bound or provided statically

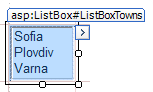
<asp:ListBox ID="ListBoxTowns" runat="server">

<asp:ListItem Value="1">Sofia</asp:ListItem>

<asp:ListItem Value="2">Plovdiv</asp:ListItem>

<asp:ListItem Value="3">Varna</asp:ListItem>

</asp:ListBox>

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/03.%20ASP.NET-Web-Controls-and-HTML-Controls/imgs/listbox.png)

* CheckBoxList
* CheckBoxList displays items as check boxes
  + RepeatColumns
    - The number of columns
  + RepeatDirection
    - Vertical, Horizontal
  + RepeatLayout: Table, Flow

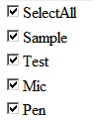
<asp:CheckBoxList ID="Extras" runat="server">

<asp:ListItem Text="Audio System" Value="1">

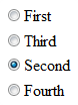
<asp:ListItem Text="Parktronic" Value="2"

Selected="True">

</asp:CheckBoxList>

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/03.%20ASP.NET-Web-Controls-and-HTML-Controls/imgs/checkbox-list.png)

* RadioButtonList
  + RadioButtonList displays data as a list of RadioButton controls
    - RepeatColumns – the number of columns displayed
    - RepeatDirection
      * Vertical, Horizontal
    - RepeatLayout
      * Table, Flow
    - Use the Items property to access its elements

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/03.%20ASP.NET-Web-Controls-and-HTML-Controls/imgs/radiobutton-list.png)

* BulletedList
* Repeater
* DataList
* GridView
* DropDownList
* ListView

## Validation Control

**13.4.1. Data Validation**

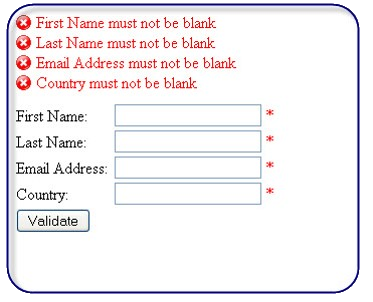
* When a user enters data in a form the programmer expects a certain data type
* Invalid input can break the application logic
* Input data validation means
  + Comparing a field to specific values or a range of acceptable values
  + Checking for required interdependencies between the fields

Predefined input formats usually follows some rules defining the acceptable set of values

* Limitation about the length of a field
  + Allow empty value or not
* Accepting digits only or not
* A range of acceptable values
  + E.g. regular expression
* Mathematical formula
  + E.g. field A > field B
* Where Validation is Performed?
* The validation is always done on the server
  + When the client request is processed
* Additionally some validation rules could be performed at the client side
* In all cases the main part of the validation should be done at the server
  + A mischievous user can alter the validation on the client
* Why Validation is Important?
* Data validation is important!
* Invalid data could cause many problems
  + Crashes of the application
  + Execution of SQL queries (SQL-injection)
  + Revealing protected data
* Example:
  + A user can enter very long text into a Web page
  + This could cause exceptions at the server side

**13.4.2. Validation Control**

* ASP.NET simplifies the process of validation by supplying a set of validation controls
* RequiredFieldValidator
  + Ensures the field is not left empty
* CompareValidator
  + Compares the input data from a control to the data in another control or to a constant value
  + Example:
    - When a password needs to be entered twice
* RangeValidator
  + Checks if the input is in given range of values
    - Between two constant values, e.g. [5..10]
    - Between two values of other controls
* CustomValidator
  + Provides a standard way for writing your own validation logic
  + The programmer needs to implement the logic
* RegularExpressionValidator
* Checks the input against a regular expression pattern, e.g. [1-9][0-9]+
* Examples:
  + Checking SSN, e-mail addresses, phones, etc.
* Visual Studio provides many predefined patterns for the most common validations
* ValidationSummary
* A summary of all errors generated by the validation controls on the page
* Provides an information about all errors in all page fields
* Usually placed near the submit button

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/11.%20Validation-Controls/imgs/validation-controls-form.png)

**13.4.3. Common Properties**

* After the validator is placed on the page you can assign its attributes (properties)
  + ControlToValidate, ErrorMessage, Text, …

<asp:type\_of\_validator

ID="validator\_id"

runat="server"

ControlToValidate="control\_id"

ErrorMessage="error\_message\_for\_summary"

Display="static|dynamic|none"

Text="Text\_to\_display\_by\_input\_control">

</asp:type\_of\_validator>

* ControlToValidate
  + Sets which control to be validated
  + One validator validates one particular field
* IsValid
  + Indicates if the input is valid
* EnableClientScript
  + Sets whether a client side validation is performed
  + Server side validation is always performed
* RangeValidator and CompareValidator have the Type property
  + Specifies the type of the data validated
  + The possible values are String, Integer, Double, Date and Currency
  + The specified range of values belongs to the same type, e.g. [1.1.2009…31.12.2010]
  + Visual Studio automatically sets this property when you specify ControlToValidate
* ErrorMessage
  + A message that is displayed if the input is found to be invalid
  + If the Text property is set it is shown instead
* When a ValidationSummary is defined
  + The value of the Text property is shown at the validation control’s position
  + ErrorMessage is displayed in the ValidationSummary
* The Display property sets the layout of the error message
  + Affects messages which are displayed at the validation control’s position
  + Messages shown in the ValidationSummary control are unaffected
  + Applied when the form use FlowLayout only
  + Static– fixed space is allocated for the error
  + Dynamic– space is allocated only if an error should be displayed, otherwise nothing is shown
* Example

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

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

<asp:CompareValidator ID="CompareValidatorPassword"

runat="server" ControlToCompare="TextBoxPassword"

ControlToValidate="TextBoxRepeatPass"

ValueToCompare="Text" ForeColor="Red"

ErrorMessage="Password doesn't match!" />

<asp:Button ID="BtnSubmit" runat="server"

Text="Submit" />

Note: due to a design flaw in the CompareValidator, when the second control is empty or contains whitespace only, the validator is not executed at all!

* Validation Controls Used in a Combination

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

<asp:RequiredFieldValidator

ID="RequiredFieldValidatorEmail"

runat="server" ForeColor="Red" Display="Dynamic"

ErrorMessage="An email address is required!"

ControlToValidate="TextBoxEmail" />

<asp:RegularExpressionValidator

ID="RegularExpressionValidatorEmail"

runat="server" ForeColor="Red" Display="Dynamic"

ErrorMessage="Email address is incorrect!"

ControlToValidate="TextBoxEmail"

ValidationExpression=

"[a-zA-Z][a-zA-Z0-9\-\.]+[a-zA-Z]@[a-zA-Z][a-zA-Z0-9\-\.]+[a-zA-Z]+\.[a-zA-Z]{2,4}" />

<asp:Button ID="ButtonSubmit" runat="server" />

* A Web Page Validation

The Page.IsValid property is evaluated as a logical "AND" of all validation controls on the page

private void ButtonSubmit\_Click(

object sender, EventArgs e)

{

if (Page.IsValid)

{

LabelMessage.Text = "The page is valid!";

// Perform some logic here

}

// An else clause is unneeded – the page will

// be returned to the user and all error

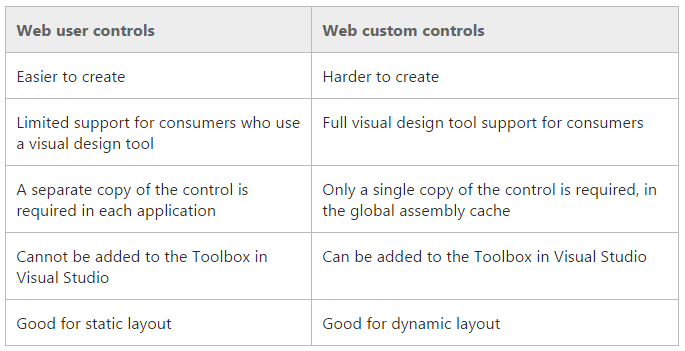
// messages will be displayed

}

## User Controls

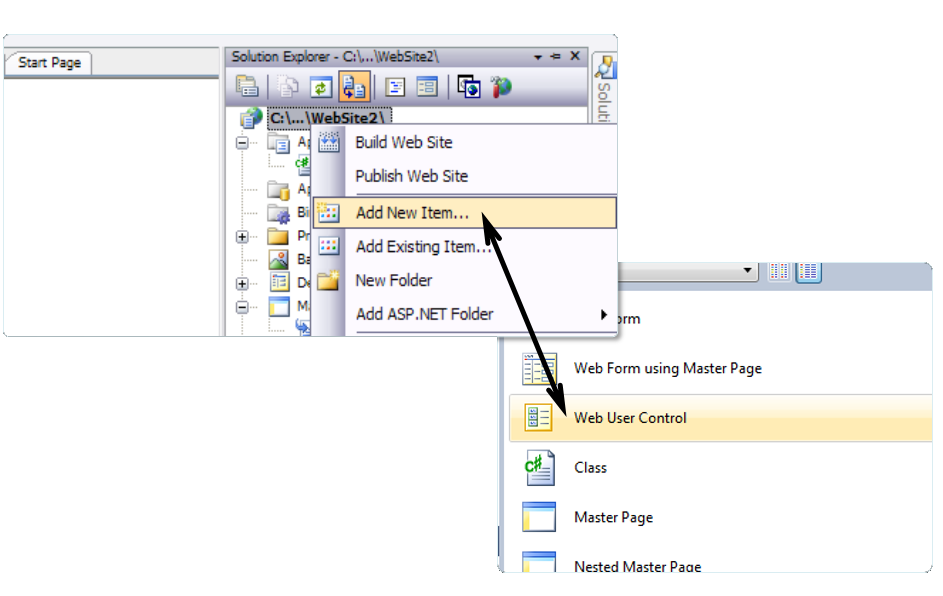
ASP.NET offers two ways of building reusable UI components:

* Web User Controls
  + UI server controls (reusable code snippets), designed in Visual Studio
  + Consist of .ascx and .ascx.cs files, inherit from UserControl
* Web Custom Controls
  + Plain C# code inheriting from WebControl
  + No HTML, rendered in C# code



**13.5.1. User Controls**

* Web user controls are reusable UI components used in ASP.NET Web Forms applications
* User controls derive from UserControl which derive from TemplateControl
  + Similar to a Web form
  + Have HTML code and C# code (code behind)
  + Could have properties and events
  + Allow developers to create their own controls with own UI and custom behavior
* Adding a Web User Control from Visual Studio:

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/13.%20User-Controls/imgs/adding-web-user-control.png)

* A Web user control is:
  + An reusable ASP.NET code snippet that can be nested as part of an ASP.NET page
  + A server component which offers a user interface and attached logic
  + Server side logic and lifecycle events (C# code behind)
  + Client-side logic (JavaScript code)
  + Shared between the pages of the application
  + Cannot be displayed directly in the browser
* Differs from custom server controls
  + Custom controls are advanced and beyond the scope of the course
* Consists of HTML and code
* Doesn’t contain <head>, <body>and <form>HTML tags
* Uses @Control instead of @Page
* Advantages
* Independent
  + Use separate namespaces for the variables
  + Avoid name collisions with the names of methods and properties of the page
* Reusable
  + User controls can be used more than once on a single page
    - No conflicts with properties and methods
* Language neutrality
  + User controls can be written in a language different of the one used in the page
* Using:
* A user control can be added to each ASP.NET Web form
* The form is called "host"
* The form adds the control by using the @Register directive

<%@ Register TagPrefix="demo" TagName="SomeName"

Src="NumberBox.ascx"%>

* TagName defines the name used by tags that will insert an instance of the control
* Src is the path to the user control
* Example Welcome Label:
* We want to create a "Welcome Label" user control
  + Like the <asp:Label> control
  + Has Name and says "Welcome, Name"
  + Has Color and AlternateColor (on mouse over)

WelcomeLabel.ascx

<%@ Control Language="C#" AutoEventWireup="true"

CodeBehind="WelcomeLabel.ascx.cs"

Inherits="Custom\_Controls\_Demo.WelcomeLabel" %>

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

WelcomeLabel.ascx.cs

public partial class WelcomeLabel : System.Web.UI.UserControl

{

…

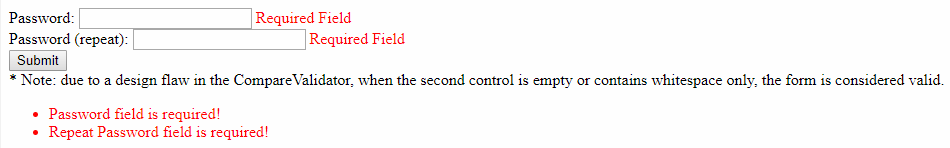
}

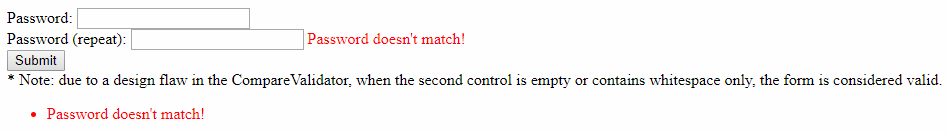
**13.5.2. Custom Controls**

* Web custom controls
* Plain C# code inheriting from WebControl
* No HTML, rendered in C# code
* Attributes [Category("…")] and[Description("…")] serve for interaction with the Visual Studio's Property Designer
* The RenderContents method renders the control as HTML code

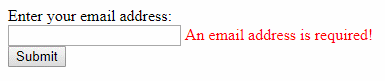
## Exercise

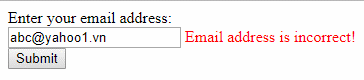
Ex 1: Using CompareValidator

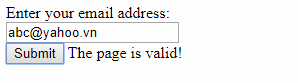




Ex 2: Using Validation Control, Page Validation



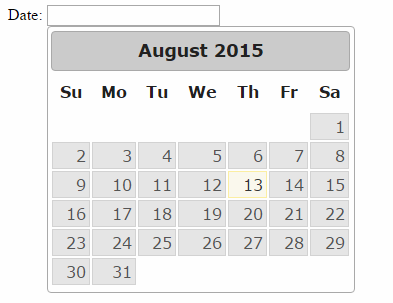




Ex 3: Create Number box User Control



Ex 4: Creating a datepicker user control for textbox



# ASP.NET Master pages, state management

## Keyword

* Master Pages and Content Pages
* WF Cookie
* WF Session

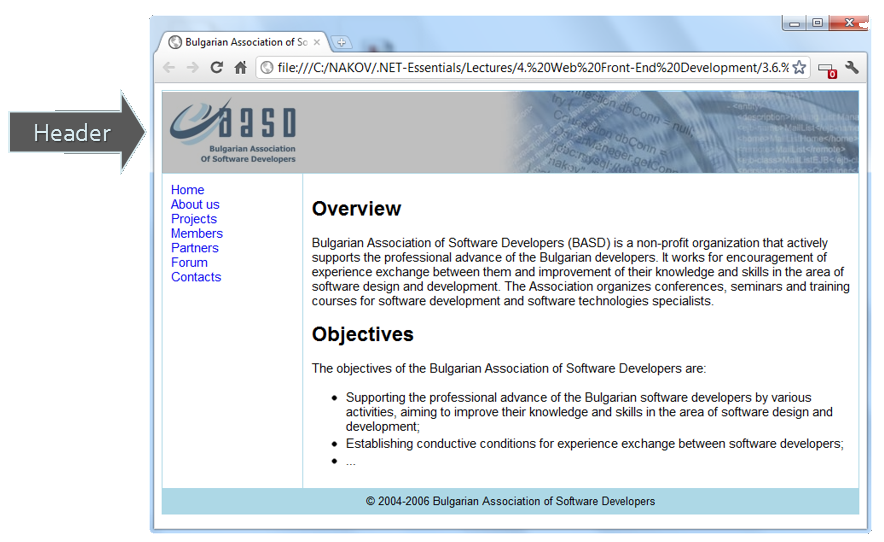
## Theory

## Master pages and Content pages

Assume we have a classical web site like this:

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/04.%20ASP.NET-Master-Pages/imgs/classical-website.png)

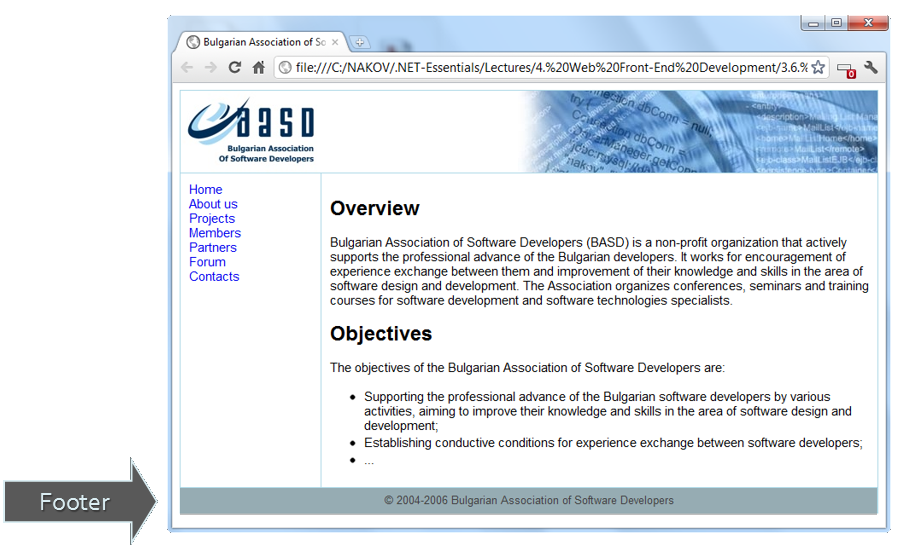
* The header is shared between all pages:

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/04.%20ASP.NET-Master-Pages/imgs/shared-header-snapshot.png)

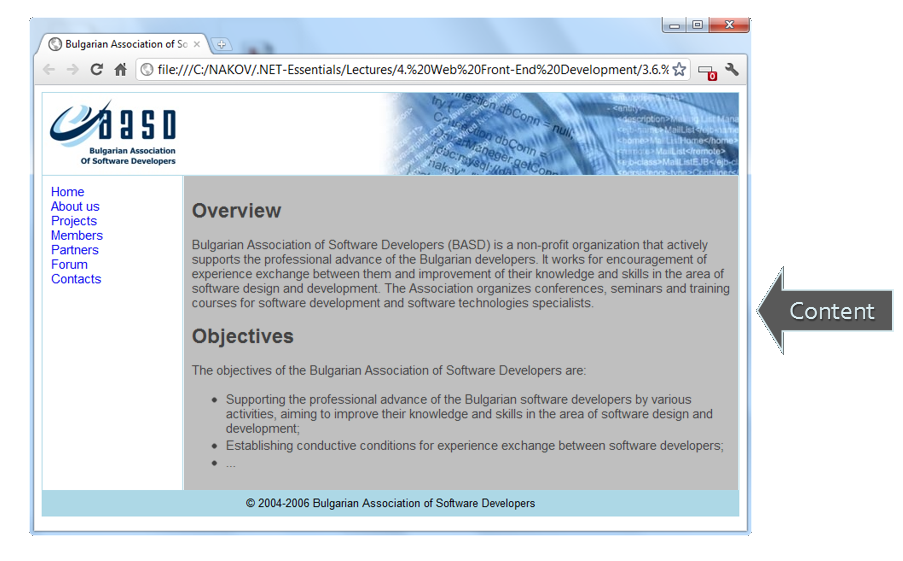
* The navigation is shared between all pages:

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/04.%20ASP.NET-Master-Pages/imgs/shared-nav-snapshot.png)

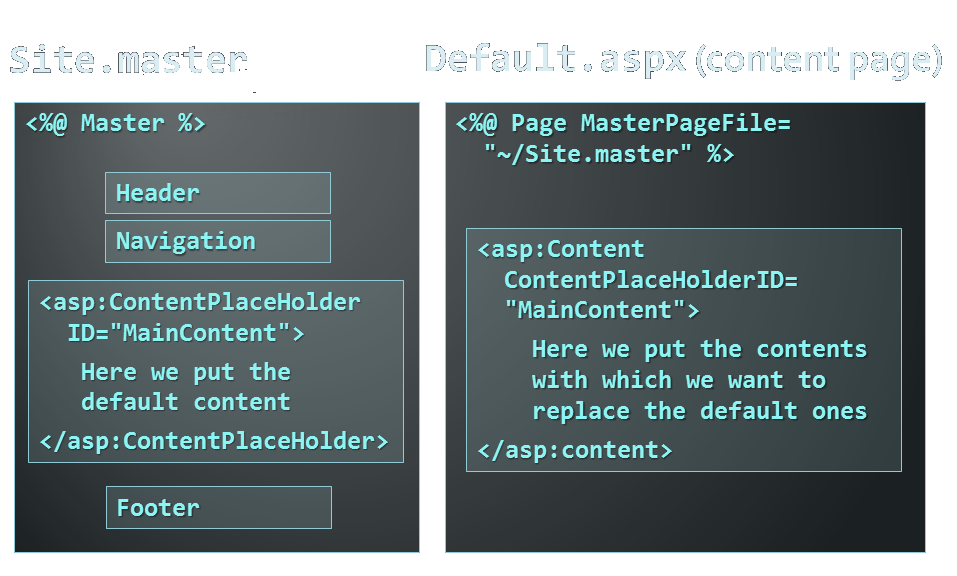
* The footer is also shared between all pages:

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/04.%20ASP.NET-Master-Pages/imgs/shared-footer-snapshot.png)

* The content is different for all pages:

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/04.%20ASP.NET-Master-Pages/imgs/different-content-snapshot.png)

* Why Use Masterand Content Pages?
* Master pages provide reusable UI templates
* The structure of the site is repeatedover most of its pages
  + ASP.NET master pagesallow to share the common HTML between few pages
* Common Look & Feel
* To avoid the repeating (and copying) of HTML code and the logic behind it
* ASP.NET Master Pages
* Master pages in ASP.NET Web Forms start with the @Masterdirective
  + Mostly the same attributes as the @Pagedirective
* Master pages can contain:
  + Markup for the page (<html>,<body>,…)
  + Standard contents (HTML, ASP.NET controls)
  + <asp:ContentPlaceHolder>controls which can be replaced in the content pages
* Content Pages
* Content pagesderive the entire content and logic from their master page
* Use the @Page directive with MasterPageFile attribute pointing to the master page
  + Replace a <asp:ContentPlaceHolder> from the master page by using the <asp:Content> control
  + Set theContentPlaceHolderIDproperty
    - Points to the ContentPlaceHolder from the Master page which content we want to replace
* Master and Content Pages – Mechanics

[](https://github.com/TelerikAcademy/ASP.NET-Web-Forms/blob/master/04.%20ASP.NET-Master-Pages/imgs/master-and-content-pages.png)

* Master and Content Pages – Advanced
* We can change the Master page at runtime in the code-behind

Page.MasterPageFile = "~/SiteLayout.master";

* We can also select the Master page according to the browser type

<%@ Page Language="C#"

ie:MasterPageFile="~/IESiteLayout.master"

mozilla:MasterPageFile="~/FFSiteLayout.master" %>

## WF Cookie

ASP.NET Cookie is a small bit of text that is used to store user-specific information. This information can be read by the web application whenever user visits the site.

When a user requests for a web page, web server sends not just a page, but also a cookie containing the date and time. This cookie stores in a folder on the user's hard disk.

When the user requests for the web page again, browser looks on the hard drive for the cookie associated with the web page. Browser stores separate cookie for each different sites user visited.

Note: The Cookie is limited to small size and can be used to store only 4 KB (4096 Bytes) text.

There are two ways to store cookies in ASP.NET application.

* Cookies collection
* HttpCookie

We can add Cookie either to Cookies collection or by creating instance of HttpCookie class. both work same except that HttpCookie require Cookie name as part of the constructor.

Ex HttpCookie:

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

        {

            //-------------- Creating Cookie --------------------------//

            // Creating HttpCookie instance by specifying name "student"

                HttpCookie cokie = **new** HttpCookie("student");

            // Assigning value to the created cookie

                cokie.Value = "Rahul Kumar";

            // Adding Cookie to the response instance

                Response.Cookies.Add(cokie);

            //--------------- Fetching Cookie -------------------------//

            var co\_val  = Response.Cookies["student"].Value;

            Label1.Text = co\_val;

        }

Ex Cookies collection:

**<**%@ Page Title="Home Page" Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"

Inherits="CoockieExample.\_Default" %**>**

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

**<asp:Label** ID="Label1" runat="server" Text="Select Brand Preferences"**></asp:Label>**

**<br** **/>**

**<br** **/>**

**<asp:CheckBox** ID="apple" runat="server" Text="Apple" **/>**

**<br** **/>**

**<asp:CheckBox** ID="dell" runat="server" Text="Dell" **/>**

**<br** **/>**

**<asp:CheckBox** ID="lenevo" runat="server" Text="Lenevo" **/>**

**<br** **/>**

**<asp:CheckBox** ID="acer" runat="server" Text="Acer" **/>**

**<br** **/>**

**<asp:CheckBox** ID="sony" runat="server" Text="Sony" **/>**

**<br** **/>**

**<asp:CheckBox** ID="wipro" runat="server" Text="Wipro" **/>**

**<br** **/>**

**<br** **/>**

**<asp:Button** ID="Button1" runat="server" OnClick="Button1\_Click" Text="Submit" **/>**

**<p>**

**<asp:Label** ID="Label2" runat="server"**></asp:Label>**

**</p>**

**</form>**

**using** System;

**using** System.Web.UI;

**namespace** CoockieExample

{

**public** partial **class** \_Default : Page

    {

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

        {

            // Setting expiring date and time of the cookies

            Response.Cookies["computer"].Expires = DateTime.Now.AddDays(-1);

        }

**protected** **void** Button1\_Click(**object** sender, EventArgs e)

        {

            Label2.Text = "";

            // --------------- Adding Coockies ---------------------//

**if** (apple.Checked)

                Response.Cookies["computer"]["apple"]  = "apple";

**if** (dell.Checked)

                Response.Cookies["computer"]["dell"]   = "dell";

**if** (lenevo.Checked)

                Response.Cookies["computer"]["lenevo"] = "lenevo";

**if** (acer.Checked)

                Response.Cookies["computer"]["acer"]   = "acer";

**if** (sony.Checked)

                Response.Cookies["computer"]["sony"]   = "sony";

**if** (wipro.Checked)

                Response.Cookies["computer"]["wipro"]  = "wipro";

            // --------------- Fetching Cookies -----------------------//

**if** (Request.Cookies["computer"].Values.ToString() != **null**)

            {

**if** (Request.Cookies["computer"]["apple"] != **null**)

                    Label2.Text += Request.Cookies["computer"]["apple"] + " ";

**if** (Request.Cookies["computer"]["dell"] != **null**)

                    Label2.Text += Request.Cookies["computer"]["dell"] + " ";

**if** (Request.Cookies["computer"]["lenevo"] != **null**)

                    Label2.Text += Request.Cookies["computer"]["lenevo"] + " ";

**if** (Request.Cookies["computer"]["acer"] != **null**)

                    Label2.Text += Request.Cookies["computer"]["acer"] + " ";

**if** (Request.Cookies["computer"]["sony"] != **null**)

                    Label2.Text += Request.Cookies["computer"]["sony"] + " ";

**if** (Request.Cookies["computer"]["wipro"] != **null**)

                    Label2.Text += Request.Cookies["computer"]["wipro"] + " ";

            }**else** Label2.Text = "Please select your choice";

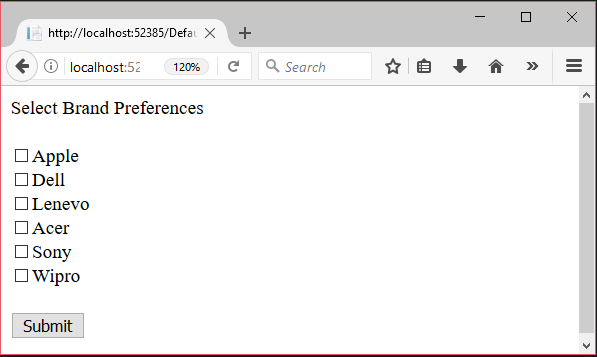
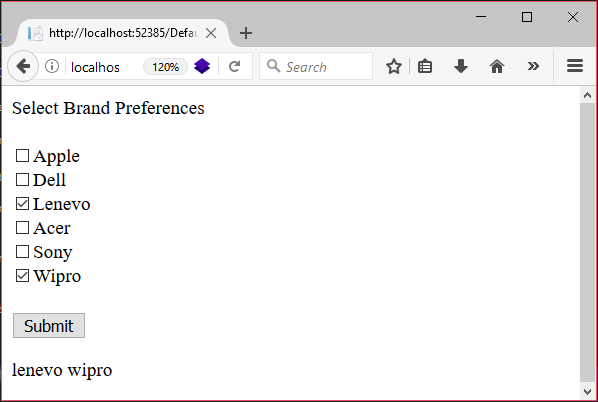
            Response.Cookies["computer"].Expires = DateTime.Now.AddDays(-1);

        }

    }

}

Output:

This example will store selected values as cookie.  


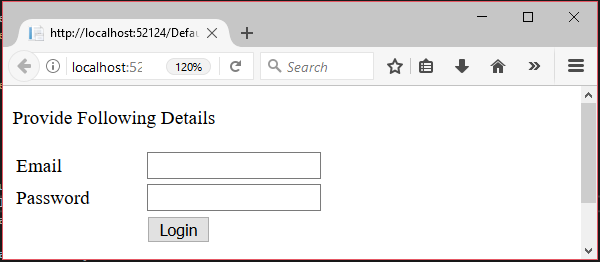
## WF Session

In ASP.NET session is a state that is used to store and retrieve values of a user.

It helps to identify requests from the same browser during a time period (session). It is used to store value for the particular time session. By default, ASP.NET session state is enabled for all ASP.NET applications.

Each created session is stored in **SessionStateItemCollection** object. We can get current session value by using **Session** property of **Page** object. Let's see an example, how to create an access session in asp.net application.

In the following example, we are creating a session and storing user email. This example contains the following files.



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

        {

**if** (password.Text=="qwe123")

            {

                // Storing email to Session variable

                Session["email"] = email.Text;

            }

            // Checking Session variable is not empty

**if** (Session["email"] != **null**)

            {

                // Displaying stored email

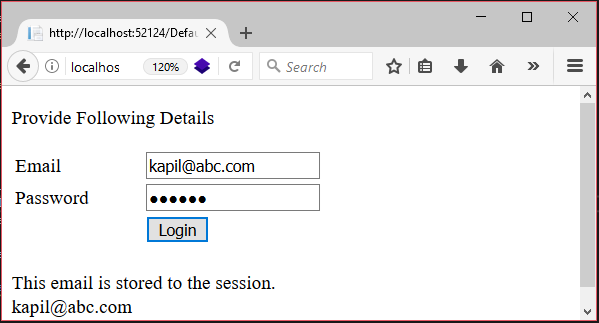
                Label3.Text = "This email is stored to the session.";

                Label4.Text = Session["email"].ToString();

            }

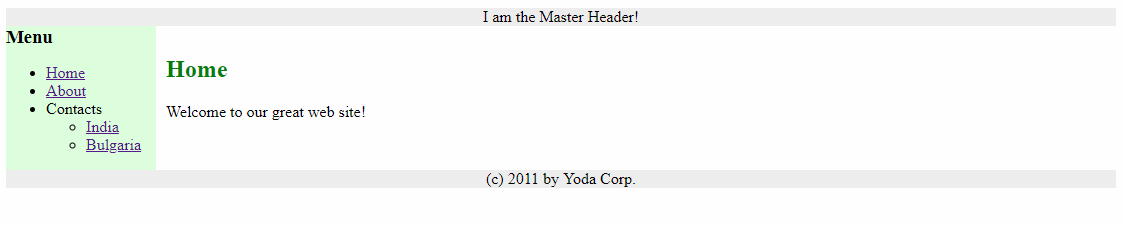
        }

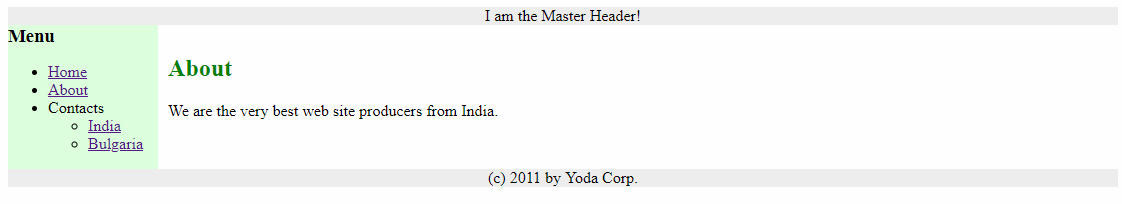
It will show stored session value, user email.

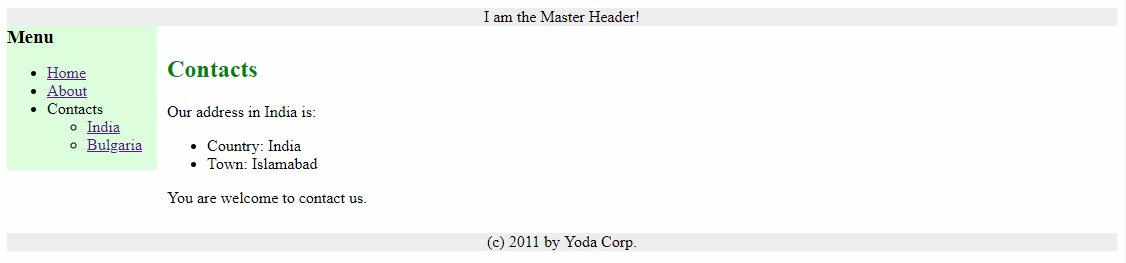


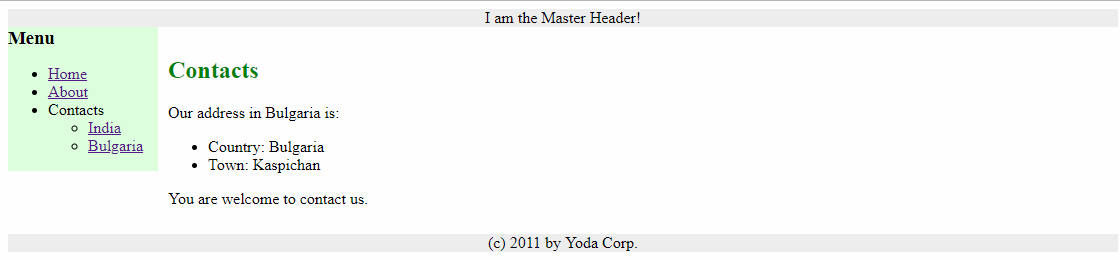
## Exercise

Ex 1: Create master pages and content pages:



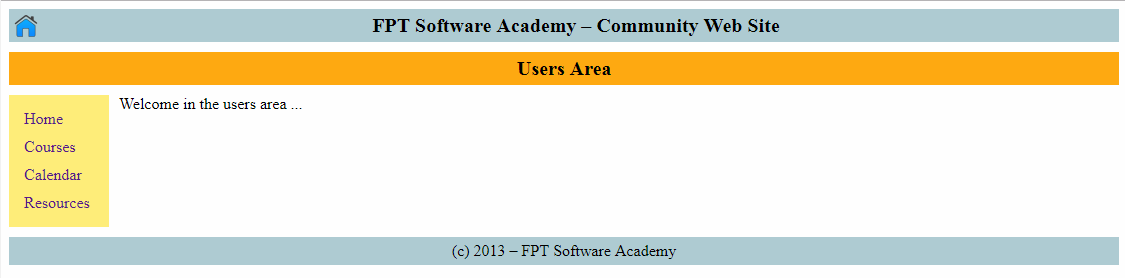


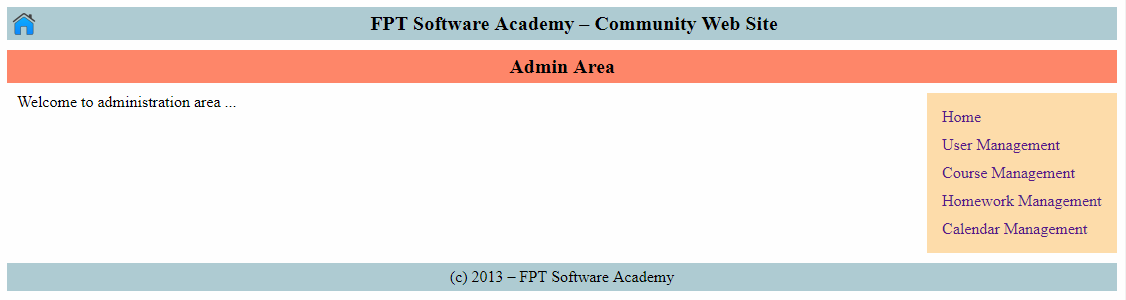




Ex2: Using nested master page:







# ADO.NET, ODP.NET

## Keyword

* ADO.NET
* Using Stored Procedures
* ODP.NET

## Theory

## ADO.NET

* ADO.NET Introduction

It is a module of .Net Framework which is used to establish connection between application and data sources. Data sources can be such as SQL Server and XML. ADO.NET consists of classes that can be used to connect, retrieve, insert and delete data.

All the ADO.NET classes are located into **System.Data.dll** and integrated with XML classes located into **System.Xml.dll.**

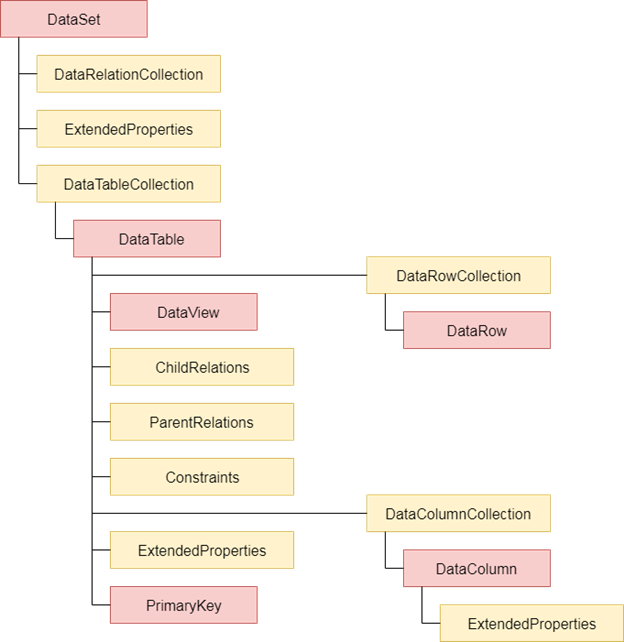
ADO.NET has two main components that are used for accessing and manipulating data are the .NET Framework data provider and the DataSet.

* .NET Framework Data Providers

These are the components that are designed for data manipulation and fast access to data. It provides various objects such as **Connection, Command, DataReader and DataAdapter** that are used to perform database operations. We will have a detailed discussion about **Data Providers** in new topic.

* The DataSet

It is used to access data independently from any data resource. DataSet contains a collection of one or more DataTable objects of data. The following diagram shows the relationship between .NET Framework data provider and DataSet.



**Fig:** ADO.NET Architecture

* Which one should we use DataReader or DataSet?

We should consider the following points to use DataSet.

* It caches data locally at our application, so we can manipulate it.
* It interacts with data dynamically such as binding to windows forms control.
* It allows performing processing on data without an open connection. It means it can work while connection is **disconnected.**

If we required some other functionality mentioned above, we can use **DataReader** to improve performance of our application.

DataReader does not perform in disconnected mode. It requires DataReader object to be **connected.**

* .NET Framework Data Provider for SQL Server

Data provider for SQL Server is a lightweight component. It provides better performance because it directly access SQL Server without any middle connectivity layer. In early versions, it interacts with ODBC layer before connecting to the SQL Server that created performance issues.

The .NET Framework Data Provider for SQL Server classes is located in the **System.Data.SqlClient** namespace. We can include this namespace in our C# application by using the following syntax.

using System.Data.SqlClient;

This namespace contains the following important classes.

|  |  |
| --- | --- |
| **Class** | **Description** |
| SqlConnection | It is used to create SQL Server connection. This class cannot be inherited. |
| SqlCommand | It is used to execute database queries. This class cannot be inherited. |
| SqlDataAdapter | It represents a set of data commands and a database connection that are used to fill the DataSet. This class cannot be inherited. |
| SqlDataReader | It is used to read rows from a SQL Server database. This class cannot be inherited. |
| SqlException | This class is used to throw SQL exceptions. It throws an exception when an error is occurred. This class cannot be inherited. |

## Using Stored Procedures

### Introduction

A stored procedures is a pre-defined, reusable routine that is stored in a database. SQL Server compiles stored procedures, which makes them more efficient to use. Therefore, rather than dynamically building queries in your code, you can take advantage of the reuse and performance benefits of stored procedures. The following sections will show you how to modify the SqlCommand object to use stored procedures. Additionally, you’ll see another reason why parameter support is an important part of the ADO.NET libraries.

### Executing a Stored Procedure

In addition to commands built with strings, the SqlCommand type can be used to execute stored procedures. There are two tasks require to make this happen: let the SqlCommand object know which stored procedure to execute and tell the SqlCommand object that it is executing a stored procedure. These two steps are shown below:

// 1. create a command object identifying

// the stored procedure

SqlCommand cmd = new SqlCommand(

"Ten Most Expensive Products", conn);

// 2. set the command object so it knows

// to execute a stored procedure

cmd.CommandType = CommandType.StoredProcedure;

While declaring the SqlCommand object above, the first parameter is set to “Ten Most Expensive Products”. This is the name of a stored procedure in the Northwind database. The second parameter is the connection object, which is the same as the SqlCommand constructor used for executing query strings.

The second command tells the SqlCommand object what type of command it will execute by setting its CommandTypeproperty to theStoredProcedure value of the CommandType enum. The default interpretation of the first parameter to the SqlCommand constructor is to treat it as a query string. By setting the CommandType to StoredProcedure, the first parameter to the SqlCommand constructor will be interpreted as the name of a stored procedure (instead of interpreting it as a command string). The rest of the code can use the SqlCommand object the same as it is used in previous lessons.

### Sending Parameters to Stored Procedures

Using parameters for stored procedures is the same as using parameters for query string commands. The following code shows this:

// 1. create a command object identifying

// the stored procedure

SqlCommand cmd = new SqlCommand(

"CustOrderHist", conn);

// 2. set the command object so it knows

// to execute a stored procedure

cmd.CommandType = CommandType.StoredProcedure;

// 3. add parameter to command, which

// will be passed to the stored procedure

cmd.Parameters.Add(

new SqlParameter("@CustomerID", custId));

The SqlCommand constructor above specifies the name of a stored procedure, CustOrderHist, as its first parameter. This particular stored procedure takes a single parameter, named @CustomerID. Therefore, we must populate this parameter using a SqlParameter object. The name of the parameter passed as the first parameter to the SqlParameter constructor must be spelled exactly the same as the stored procedure parameter. Then execute the command the same as you would with any other SqlCommand object.

## OPD.NET

ODP.NET is the data provider supported by Oracle and implements several Oracle database's specific features.

The object model of ODP.NET provides a rich collection of classes that assist in easy database interaction and the objects are part of the Oracle.DataAccessClient namespace hosted in the assembly Oracle.DataAccess.dll. When you install ODP.NET, the Oracle Universal Installer registers this assembly with the Global Assembly Cache. You can browse objects any Oracle database through the Oracle Explorer windows available from the View menu in the Visual Studio main menu. You can access any Oracle database's specific features from this window, and allows dragging and dropping schema objects and the generation of automatic code for you.

Now we're going to create the Windows client application performing the following steps.

1. In Visual Studio.NET go to the main menu and select File|New|Project and from the New Project window choose Windows Application template, enter ODPWinClient for the name, and enter a directory to store the project.
2. Add a reference to the assembly Oracle.DataAccessClient.dll selecting Project|Add Reference... from the main menu.
3. Add a DataGridView control from the toolbox to the form. Name it m\_dgvViewer and one Button control.
4. Add the following namespace declaration.

using Oracle.DataAccess.Client;  
using Oracle.DataAccess.Types;

1. The most important object is the connection instance of the class OracleConnection. Prior connecting to an Oracle Database using ODP.NET, you should add Net Service Names. You need to update the file tnsnames.ora in your local ORACLE\_HOME\network\admin directory by adding entries that would be the data source list. You can edit this file manually, or using Oracle Net Configuration tool which walks you through some screens gathering required connection information. The code for creating the connection is show below.

string strConn = "Data Source=ORCL; User Id=scott; Password=tiger";  
OracleConnection objConnection = new OracleConnection();  
objConnection.ConnectionString = strConn;  
  
In this case we have configured the file tnsnames.ora as shown bellow to connect the sample database ORCL using the well known user scott and password tiger.   
   
ORCL =  
 (DESCRIPTION=  
 (ADDRESS\_LIST=  
  (ADDRESS= (PROTOCOL=TCP)(HOST=yourhost)(PORT=1521))  
 )  
 {CONNECT\_DATA=  
  (SERVICE\_NAME=ORCL)  
 }  
 )    
  
There are some connection string properties specific to ODP.NET provider such as DBA Privilege which sets to SYSDBA or SYSOPER to request administrative privileges, ValidateConnection, StatementCachePurge which causes the statement cache to be purged when the connection is closed, StatementCacheSize, ProxyUserId, ProxyPassword, IncrPoolSize and DecrPoolSize.

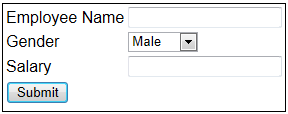
1. You can retrieve information about departments in the ORCL database system (this is an illustrative database system for managing human resources' data) using the following code.

string strConn = "Data Source=ORCL; User Id=scott; Password=tiger";  
using (OracleConnection objConnection = new OracleConnection())  
{  
     objConnection.ConnectionString = strConn;  
     try  
     {  
         objConnection.Open();  
         OracleCommand objCommand = new OracleCommand();  
         objCommand.Connection = objConnection;  
         objCommand.CommandText = "select deptno, dname, loc from dept";  
         objCommand.CommandType = System.Data.CommandType.Text;  
         OracleDataAdapter objAdapter = new OracleDataAdapter(objCommand);  
         DataTable objTable = new DataTable();  
         objAdapter.Fill(objTable);  
         this.m\_dgvViewer.DataSource = objTable;  
         objConnection.Close();  
     }  
     catch (Exception ex)  
     {  
         System.Windows.Forms.MessageBox.Show(ex.ToString());  
     }  
     finally  
     {  
         objConnection.Close();  
     }  
}

## Exercise

Ex1: Using stored procedure in SQL Server, ADO.NET asp.net C#: search, insert data

**The design of the webform, should be as shown below:**



CREATE TABLE tblEmployees

(

EmployeeId int identity primary key,

Name nvarchar(50),

Gender nvarchar(10),

Salary int

)

INSERT INTO tblEmployees values(‘Mason’,’Male’,5000)

INSERT INTO tblEmployees values(‘Priyanka’,’Female’,3500)

INSERT INTO tblEmployees values(‘John’,’Male’,2350)

INSERT INTO tblEmployees values(‘Louna’,’Female’,5700)

INSERT INTO tblEmployees values(‘Jackson’,’Male’,4890)

INSERT INTO tblEmployees values(‘Aulia’,’Female’,4500)

Ex2: Using stored procedure in Oracle, ODP.NET asp.net C# with above ex1.

# 3 layers architecture

## Keyword

* 3 layers Architecture

## Theory

## 3 layers Architecture

Layer indicates logical separation of components, such as having distinct namespaces and classes for the Database Access Layer, Business Logic Layer and User Interface Layer.



Example:

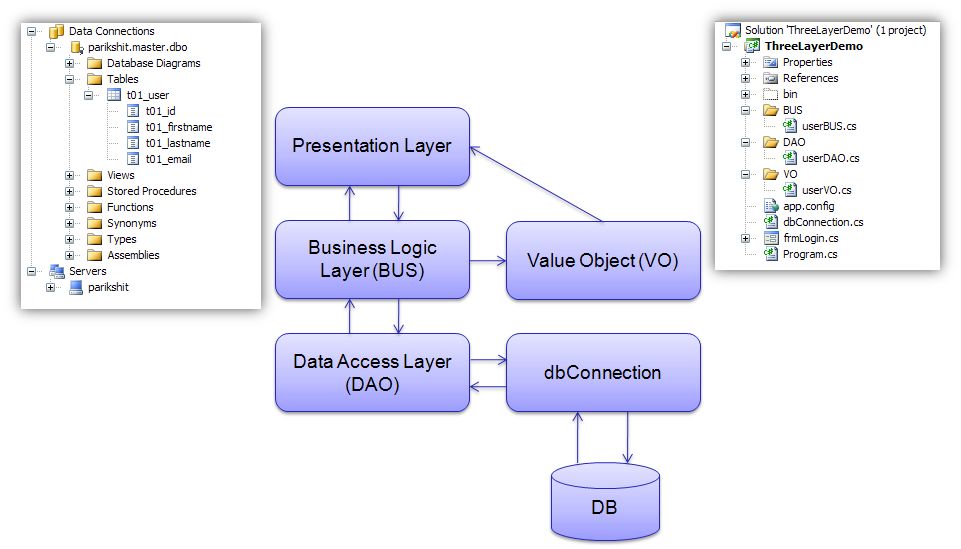
Using table t01\_user

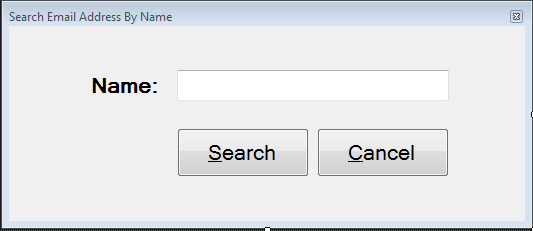
(t01\_id int identity,

t01\_firstname nvarchar(20),

t01\_lastname nvarchar(20),

t01\_email varchar(50)).





#### dbConnection

This class is mainly used to do the database activity like Select, Update and Delete query to database. It also checks if the database connection is open or not. If database connection is not open, then it opens the connection and performs the database query. The database results are to be received and being passing in Data Table in this class.

This class takes the database setting from the app.config file so it’s really flexible to manage the database settings.

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

using System;

using System.Collections.Generic;

using System.Text;

using System.Data;

using System.Data.SqlClient;

using System.Configuration;

namespace ThreeLayerDemo.Core

{

public class dbConnection

{

private SqlDataAdapter myAdapter;

private SqlConnection conn;

/// *<constructor>*

/// *Initialise Connection*

/// *</constructor>*

public dbConnection()

{

myAdapter = new SqlDataAdapter();

conn = new SqlConnection(ConfigurationManager.ConnectionStrings

["dbConnectionString"].ConnectionString);

}

/// *<method>*

/// *Open Database Connection if Closed or Broken*

/// *</method>*

private SqlConnection openConnection()

{

if (conn.State == ConnectionState.Closed || conn.State ==

ConnectionState.Broken)

{

conn.Open();

}

return conn;

}

/// *<method>*

/// *Select Query*

/// *</method>*

public DataTable executeSelectQuery(String \_query, SqlParameter[] sqlParameter)

{

SqlCommand myCommand = new SqlCommand();

DataTable dataTable = new DataTable();

dataTable = null;

DataSet ds = new DataSet();

try

{

myCommand.Connection = openConnection();

myCommand.CommandText = \_query;

myCommand.Parameters.AddRange(sqlParameter);

myCommand.ExecuteNonQuery();

myAdapter.SelectCommand = myCommand;

myAdapter.Fill(ds);

dataTable = ds.Tables[0];

}

catch (SqlException e)

{

Console.Write("Error - Connection.executeSelectQuery - Query:

" + \_query + " \nException: " + e.StackTrace.ToString());

return null;

}

finally

{

}

return dataTable;

}

/// *<method>*

/// *Insert Query*

/// *</method>*

public bool executeInsertQuery(String \_query, SqlParameter[] sqlParameter)

{

SqlCommand myCommand = new SqlCommand();

try

{

myCommand.Connection = openConnection();

myCommand.CommandText = \_query;

myCommand.Parameters.AddRange(sqlParameter);

myAdapter.InsertCommand = myCommand;

myCommand.ExecuteNonQuery();

}

catch (SqlException e)

{

Console.Write("Error - Connection.executeInsertQuery - Query:

" + \_query + " \nException: \n" + e.StackTrace.ToString());

return false;

}

finally

{

}

return true;

}

/// *<method>*

/// *Update Query*

/// *</method>*

public bool executeUpdateQuery(String \_query, SqlParameter[] sqlParameter)

{

SqlCommand myCommand = new SqlCommand();

try

{

myCommand.Connection = openConnection();

myCommand.CommandText = \_query;

myCommand.Parameters.AddRange(sqlParameter);

myAdapter.UpdateCommand = myCommand;

myCommand.ExecuteNonQuery();

}

catch (SqlException e)

{

Console.Write("Error - Connection.executeUpdateQuery - Query:

" + \_query + " \nException: " + e.StackTrace.ToString());

return false;

}

finally

{

}

return true;

}

}

}

#### Database Access Layer

Database Access Layer (DAO) builds the query based on received parameters from the Business Logic Layer and passes it the dbConnection class for execution. And simple return results from the dbConnection class to Business Logic Layer.

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

using System;

using System.Collections.Generic;

using System.Text;

using System.Data;

using System.Data.SqlClient;

namespace ThreeLayerDemo.Core

{

public class UserDAO

{

private dbConnection conn;

/// *<constructor>*

/// *Constructor UserDAO*

/// *</constructor>*

public UserDAO()

{

conn = new dbConnection();

}

/// *<method>*

/// *Get User Email By Firstname or Lastname and return DataTable*

/// *</method>*

public DataTable searchByName(string \_username)

{

string query = string.Format("select \* from [t01\_user]

where t01\_firstname like @t01\_firstname or t01\_lastname

like @t01\_lastname ");

SqlParameter[] sqlParameters = new SqlParameter[2];

sqlParameters[0] = new SqlParameter("@t01\_firstname", SqlDbType.VarChar);

sqlParameters[0].Value = Convert.ToString(\_username);

sqlParameters[1] = new SqlParameter("@t01\_lastname", SqlDbType.VarChar);

sqlParameters[1].Value = Convert.ToString(\_username);

return conn.executeSelectQuery(query, sqlParameters);

}

/// *<method>*

/// *Get User Email By Id and return DataTable*

/// *</method>*

public DataTable searchById(string \_id)

{

string query = "select \* from [t01\_id] where t01\_id = @t01\_id";

SqlParameter[] sqlParameters = new SqlParameter[1];

sqlParameters[0] = new SqlParameter("@t01\_id", SqlDbType.VarChar);

sqlParameters[0].Value = Convert.ToString(\_id);

return conn.executeSelectQuery(query, sqlParameters);

}

}

}

#### Value Object

Value Object is nothing more but a class with the contents GET and SET methods. It’s mainly used to pass Data from one class to another. It’s directly connected with Business Logic Layer and Presentation Layer. As you can see in the diagram object values are being SET in Business Logic Layer and GET from Presentation Layer.

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ThreeLayerDemo.Core

{

public class UserVO

{

private int \_idUser;

private string \_firstname;

private string \_lastname;

private string \_email;

/// *<constructor>*

/// *Constructor UserVO*

/// *</constructor>*

public UserVO()

{

*//*

*// TODO: Add constructor logic here*

*//*

}

public int idUser

{

get

{

return \_idUser;

}

set

{

\_idUser = value;

}

}

public string firstname

{

get

{

return \_firstname;

}

set

{

\_firstname = value;

}

}

public string lastname

{

get

{

return \_lastname;

}

set

{

\_lastname = value;

}

}

public string email

{

get

{

return \_email;

}

set

{

\_email = value;

}

}

}

}

#### Business Logic Layer

Business Logic Layer (BUS) works as a bridge between Presentation Layer and DAO. All the user values received from the presentation layer are being passed to BUS. The results received from the DAO are in row data in Data Table format but in BUS it’s converting into Value Objects (VO). Business Logic Layer (BUS) is the most important class in the whole architecture because it mainly contains all the business logic of the program. Whenever a user wants to update the business logic of the program only need to update this class.

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

using System;

using System.Collections.Generic;

using System.Text;

using System.Data;

namespace ThreeLayerDemo.Core

{

/// *<summary>*

/// *Summary description for UserBUS*

/// *</summary>*

public class UserBUS

{

private UserDAO \_userDAO;

/// *<constructor>*

/// *Constructor UserBUS*

/// *</constructor>*

public UserBUS()

{

\_userDAO = new UserDAO();

}

/// *<method>*

/// *Get User Email By Firstname or Lastname and return VO*

/// *</method>*

public UserVO getUserEmailByName(string name)

{

UserVO userVO = new UserVO();

DataTable dataTable = new DataTable();

dataTable = \_userDAO.searchByName(name);

foreach (DataRow dr in dataTable.Rows)

{

userVO.idUser = Int32.Parse(dr["t01\_id"].ToString());

userVO.firstname = dr["t01\_firstname"].ToString();

userVO.lastname = dr["t01\_lastname"].ToString();

userVO.email = dr["t01\_email"].ToString();

}

return userVO;

}

/// *<method>*

/// *Get User Email By Id and return DataTable*

/// *</method>*

public UserVO getUserById(string \_id)

{

UserVO userVO = new UserVO();

DataTable dataTable = new DataTable();

dataTable = \_userDAO.searchById(\_id);

foreach (DataRow dr in dataTable.Rows)

{

userVO.idUser = Int32.Parse(dr["t01\_id"].ToString());

userVO.firstname = dr["t01\_firstname"].ToString();

userVO.lastname = dr["t01\_lastname"].ToString();

userVO.email = dr["t01\_email"].ToString();

}

return userVO;

}

}

}

#### Presentation Layer

Presentation Layer is the only layer which is directly connected with the user. So in this matter, it’s also a really important layer for marketing purposes. Presentation Layer is mainly used for getting user data and then passing it to Business Logic Layer for further procedure, and when data is received in Value Object then it’s responsible to represent value object in the appropriate form which user can understand.

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using ThreeLayerDemo.Core;

namespace ThreeLayerDemo

{

public partial class frmLogin : Form

{

private UserBUS \_userBUS;

public frmLogin()

{

InitializeComponent();

\_userBUS = new UserBUS();

}

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

{

UserVO \_userVO = new UserVO();

\_userVO = \_userBUS.getUserEmailByName(txtUsername.Text);

if (\_userVO.email == null)

MessageBox.Show("No Match Found!", "Not Found",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

else

MessageBox.Show(\_userVO.email ,"Result",

MessageBoxButtons.OK,MessageBoxIcon.Information);

}

private void btnCancel\_Click(object sender, EventArgs e)

{

Close();

}

}

}

## Exercise

Ex1: Using 3 layers architecture and stored procedure instead of direct command sql for example above.

Ex2: Using 3 layers architecture and stored procedure, ODP.NET with schema hr/hr ORACLE, table DEPARTMENT, EMPLOYEES, asp.net C#.

