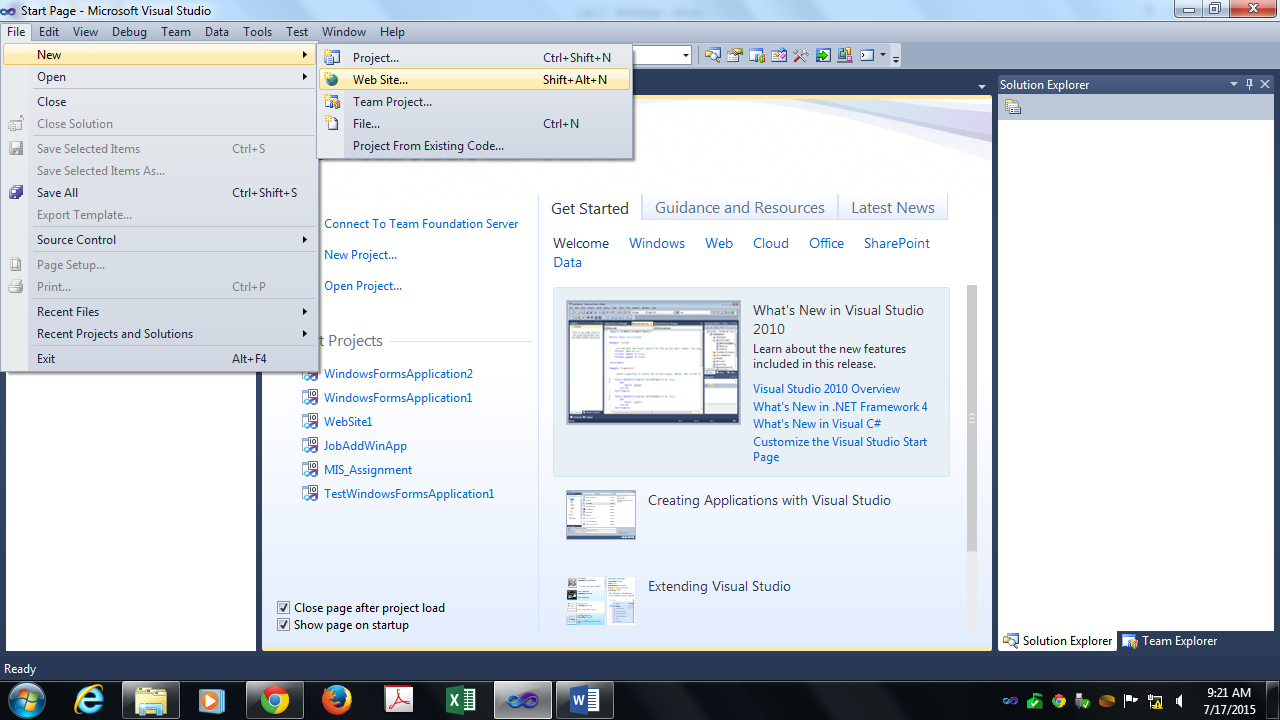
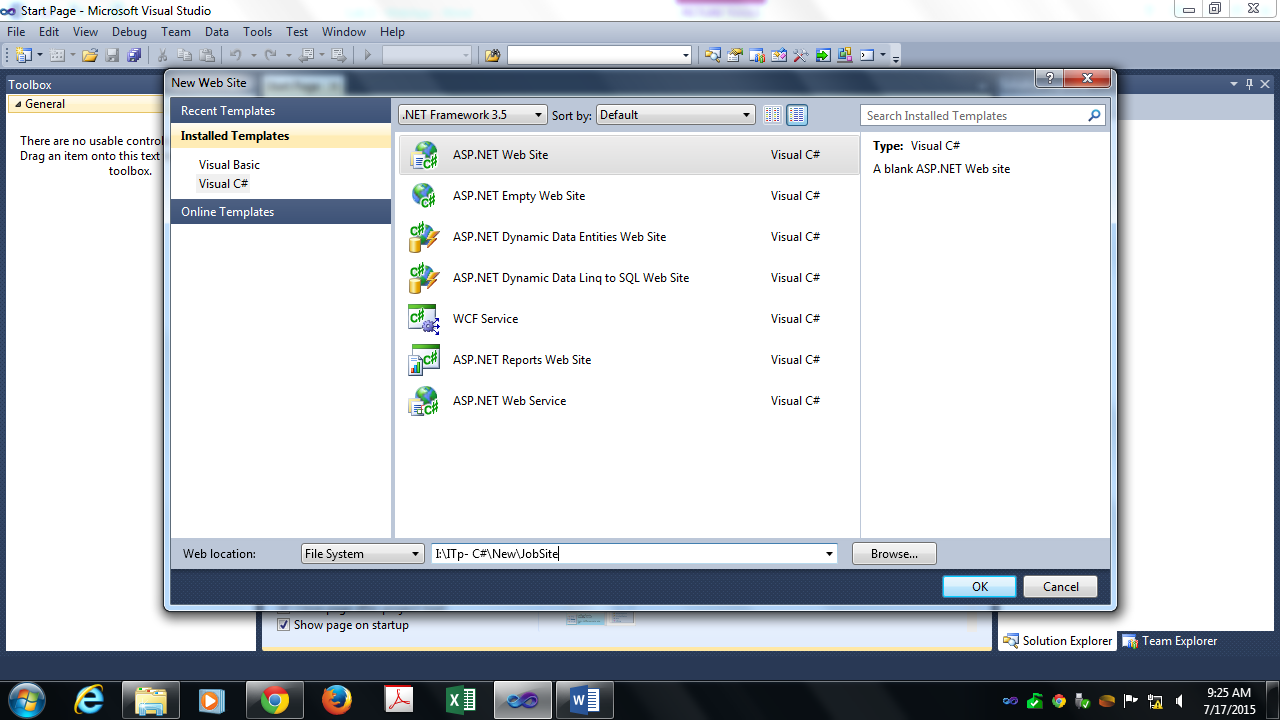
1. **Creating a new Web Site**
2. Run Microsoft Visual Studio 2010, and then select File → New → Web Site

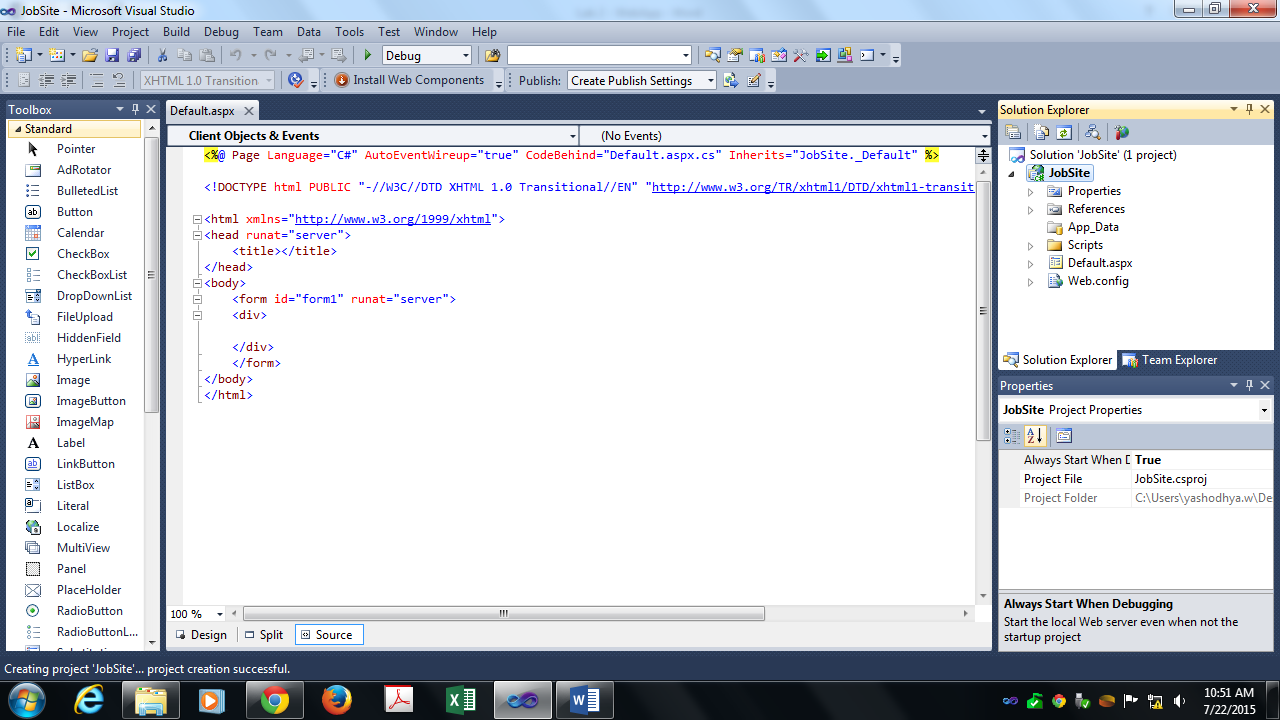


1. Select **Visual C# 🡪** Web from Installed Templates and **ASP.NET Web Application** and make sure the Framework is set to 3.5 or higher and the language is set to Visual C#. Let’s name it **JobSite**



Note the

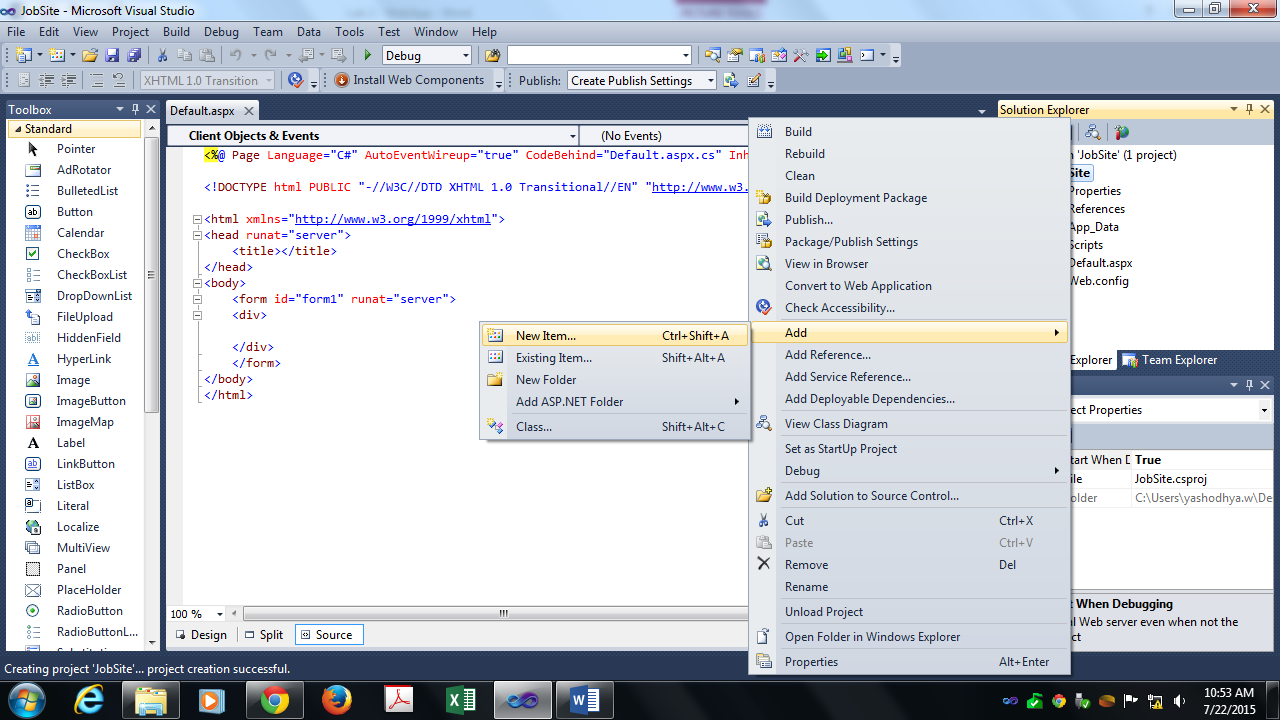
**Default.aspx** page and other initial files are created and displayed in the “**Solution Explorer**” window.



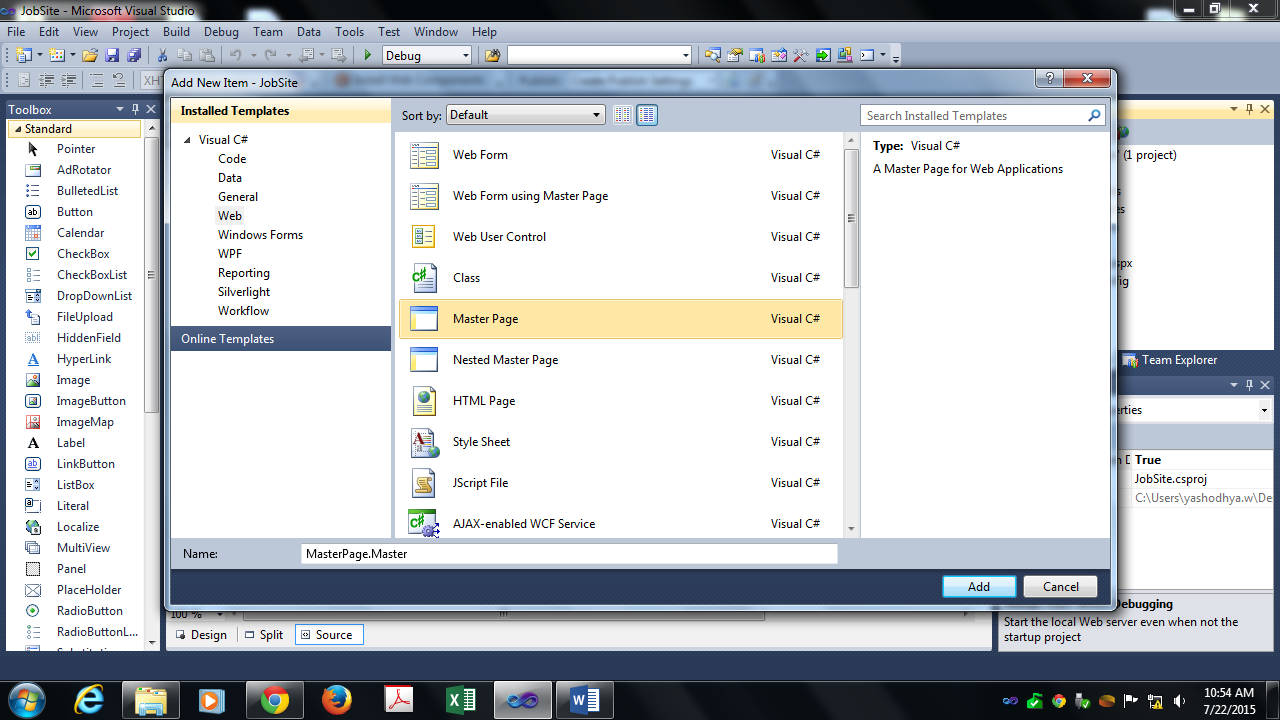
1. **Using Master Pages**

Master Pages allows developers to build a website with a consistent site-wide page layout in which each web page emit common formatting markup in addition to its custom content.

1. Create a Master page by right clicking on the web site and selecting ‘Add New Item’

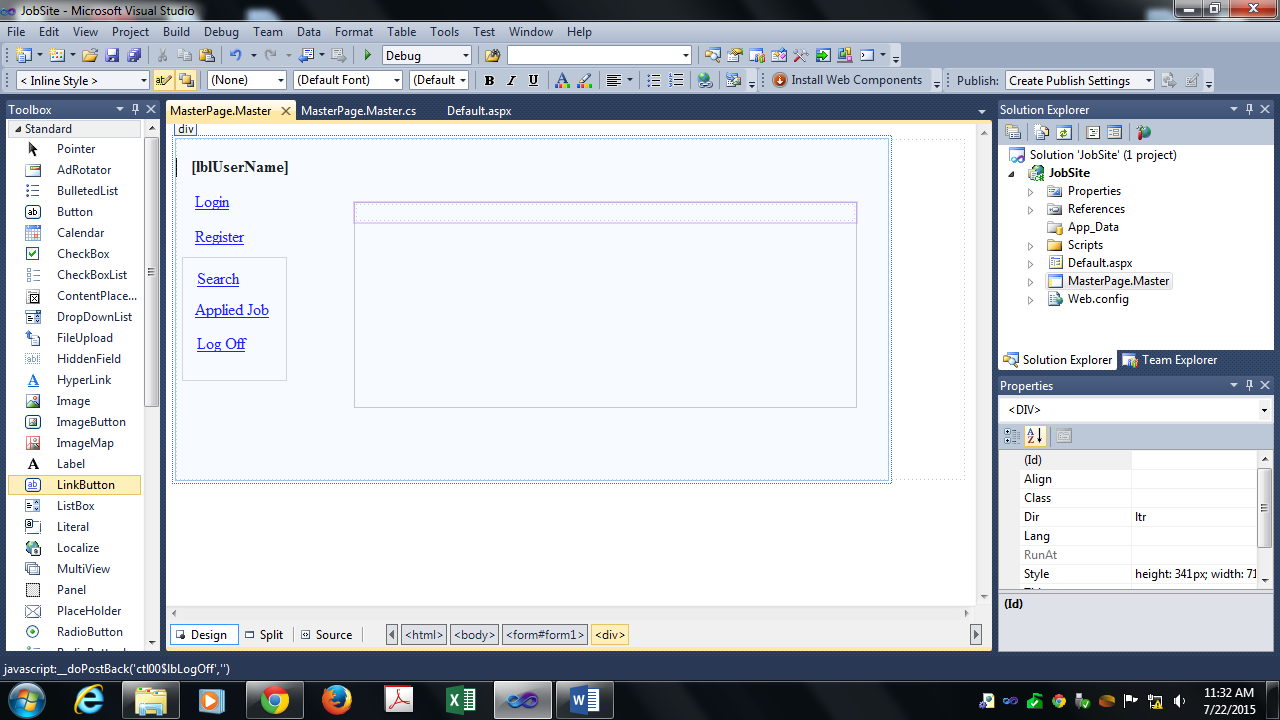


Make sure you select the language to be Visual C#.



1. Design the Master page to appear like below.
2. Click on the **Design** in the left hand bottom
3. Add div. Inside the div add the link buttons.
4. Make the position of all the controls are **‘Absolute’**

Try using below controls



Labels

lblUserName 🡪set **Text** property null

Link Buttons

lbLogOff 🡪Text – Log Off

lbRegister 🡪Text - Register

lbLogin 🡪Text – Login

lbSearch 🡪 Text - Search

lbAppliedJob 🡪Text – Applied Jobs

Panel

pnlLoggedIn 🡪Visible **–** False

Inside this panel add lbLogOff, lbSearch, lbAppliedJob

Use the property **PostBackUrl** for each link button when the respective pages are created (provide the ASPX page name).

HTML Tables

Easier to use compared to ASP Tables. Helps to structure the site content.

Panel

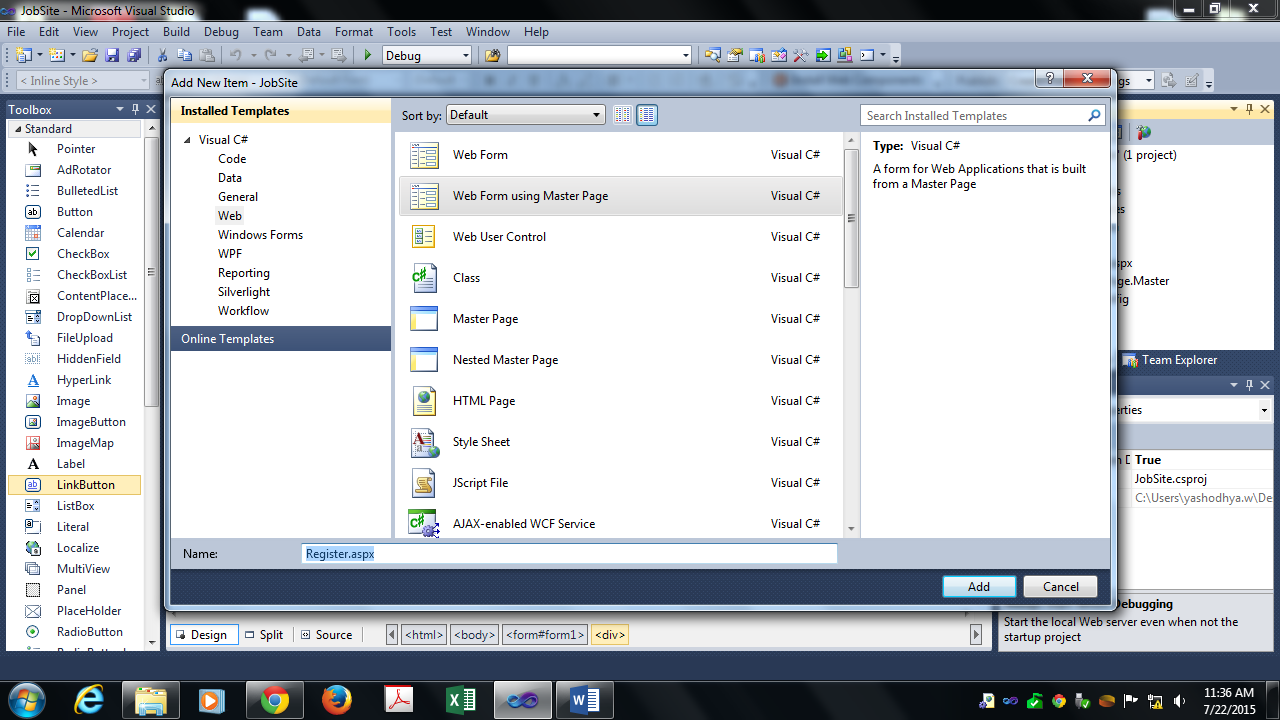
pnlLoggedIn 🡪Visible **–** False

Use **Logoff, Applied Jobs** and **Search** link buttons inside the panel. Why?

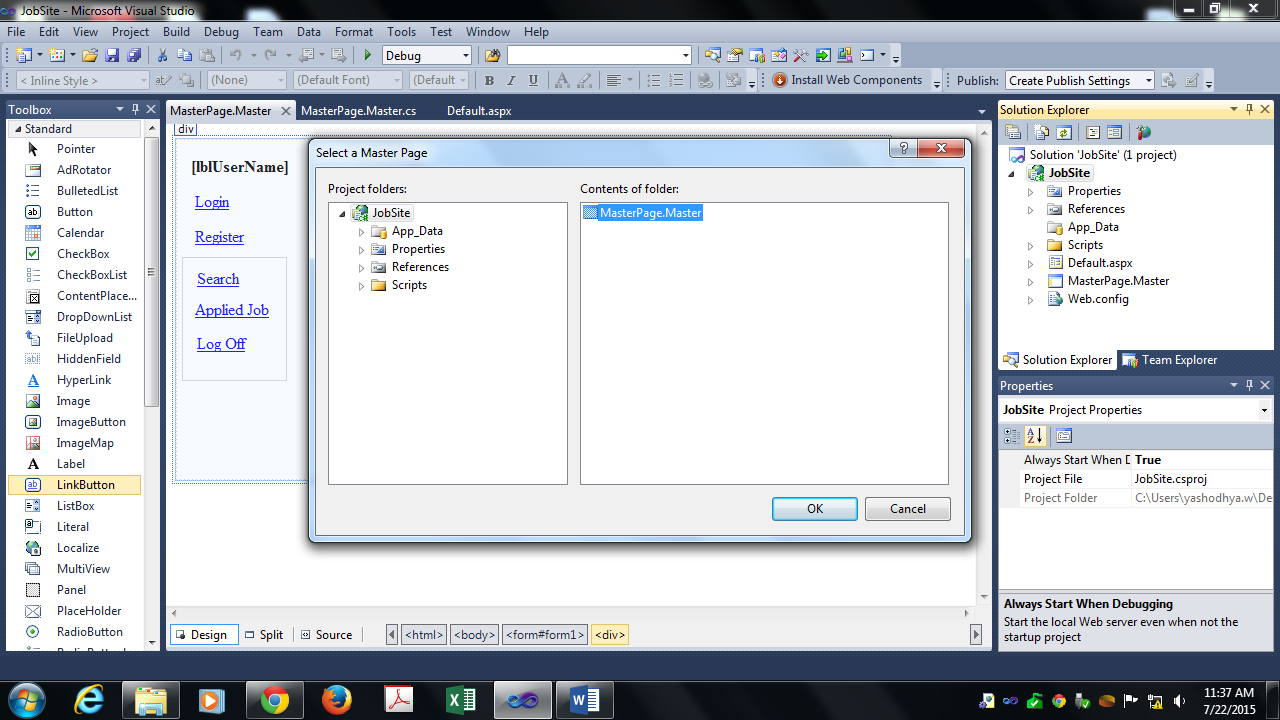
Content Place Holders

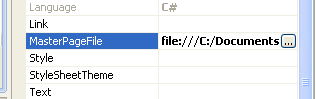
Content placeholder controls indicate regions of replaceable content on the master page.

1. **Creating Registration Page**
2. The first step is to create a Registration page for the users. In order to do that we have to create a new page named Register.aspx. Like in the previous example add a new item and select ‘**Web Form Using Master Page**’ and name it **Register.aspx**. Make sure the language is Visual C# and tick the ‘**Select Master page**’ option as well. And then click Add.



1. Next you will be prompted to select a master page. Select the master page and click OK.



Note: **Default.aspx** page which was created when the web site was created will not inherit this master page automatically. To do that double click on the default.aspx page, in properties find **MasterPageFile** property and set it.

Setting the master page will not correct the problem since there is a HTML tag in Default.aspx page.

Replace the **Default** page source code starting from the second line (starts with <!DOCTYPE) with the source code of **Register.aspx** (without the first line of Register page).

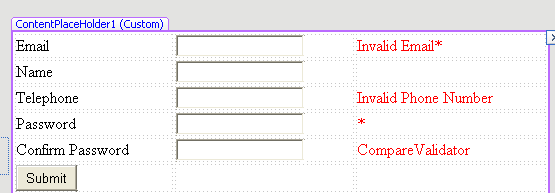
<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">

</asp:Content>

To add anything to the Default.aspx, include them within the Content ID – Content2

1. Design the following User Interface.

HTML Table

Use three columns.

Text Boxes

txtEmail, txtName, txtTelephone, txtPassword, txtConfirmPassword

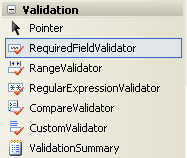
Set property **TextMode** to **Password** for both **txtPassword** and **txtConfirmPassword** controls.

Button

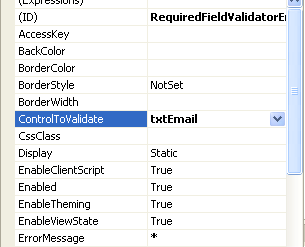
btnSubmit 🡪Text – Submit, ValidationGroup - submit

* 1. **Use of Validation Controls**

ASP.NET validation is used to validate user input at the client side.

Use validation controls to verify the existence and correct format of an Email, correct format of a phone number, existence of a password and equality of both password fields.

* + 1. **RequiredFieldValidator**

This is to indicate a field is essential. Here they are the Email and Password fields.

Important properties

1. ControlToValidate

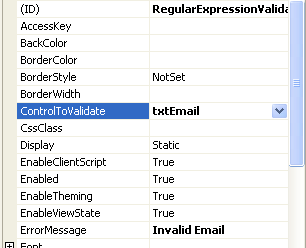
2. ErrorMessage

3. Display (Static/Dynamic)

4. ValidationGroup (explained later)

* + 1. **RegularExpressionValidator**

This is to check whether the input is in defined format. Here it is to check if the Email is in a correct format.

****

Important properties

1. ControlToValidate

2. ErrorMessage

3. Display (Static/Dynamic)

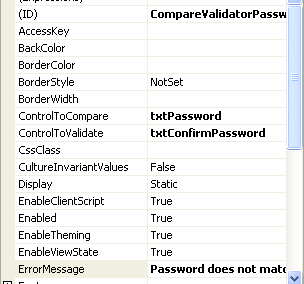
4. ValidationGroup (explained later)

5. ValidationExpression

Use this validation for telephone number as well (Example Expression 🡪 **^\d{10}$** - 10 digits).

* + 1. **CompareValidator**

This is to compare two inputs. In this case its ‘**Password**’ and ‘**Confirm Password**’ fields.



Important properties

1. ControlToValidate

2. ControlToCompare

3. ErrorMessage

4. Display (Static/Dynamic)

5. ValidationGroup (explained later)

6. ValidationExpression

* + 1. **ValidationGroup**

Validation Groups are used to group a set of validation controls together and associate them with another controller. Example – Submit button in this case.

Set **ValidationGroup** as ‘**submit**’ in all validation controls (**ValidationGroup** is set in Submit button when it was designed).

**Note**

The difference between the two can be seen when looking at the source in your browser after the page is loaded at run-time.

When the Display property is set to Dynamic, the CSS display attribute is set to none on the control. When the Display property is set to Static, the CSS visibility attribute is set to hidden and the display property is not used at all as shown in the browser output below:

**Validation Control Display = Dynamic**  
  
<td style=”width: 491px; height: 16px”>  
<span id=”RangeValidator1″  
style=”display: inline-block; color: Red; width: 226px;  
display: none;“>The age must be between 0 and 100 </span>  
</td>  
  
**Validation Control Display = Static**  
  
<td style=”width: 491px; height: 16px”>  
<span id=”Span1″  
style=”display: inline-block; color: Red; width: 226px; visibility: hidden;“>  
The age must be between 0 and 100 </span>  
</td>

The default setting on the control for the Display property is **Static**. By setting this to Static the space for the error message is reserved. The advantages here is that the controls placed positionally to the right of the validation control will not be moved at run-time if the display message of the validation box appears.

* 1. **Setting the Connection String**

Open your web.config and locate the **connectionStrings** bracket. And add a connection string.

<configuration>

.

.

<connectionStrings>

<add name="ConString" connectionString="Data Source=*SERVERNAME*; Initial Catalog=*DBNAME*;User ID=*USERNAME*; Password=*PASSWORD*" providerName="System.Data.SqlClient"/>

</connectionStrings>

.

.

</configuration>

* 1. **Adding required references**

Add ‘System.Configuration’ component to the project.

Also import necessary namespaces to **Register.aspx.cs** file.

using System.Data;

using System.Data.SqlClient;

using System.Configuration;

* 1. **Adding required references**

1. Double click on the Submit button. The following code snippet illustrates the execution of a stored procedure (will discuss in the next step) via C#.NET that will insert user information to the table.

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

{

string ConStr = ConfigurationManager.ConnectionStrings["ConString"].ConnectionString;

SqlConnection NewCon = new SqlConnection(ConStr);

NewCon.Open();

SqlCommand NewCmd = NewCon.CreateCommand();

NewCmd.CommandType = CommandType.StoredProcedure;

NewCmd.CommandText = "AddUser";

NewCmd.Parameters.Add("@email", txtEmail.Text.Trim());

NewCmd.Parameters.Add("@password", txtPassword.Text);

NewCmd.Parameters.Add("@fullName", txtName.Text.Trim());

NewCmd.Parameters.Add("@phoneNumber", txtTelephone.Text.Trim());

//AddWithValue is the most suitable method to use

NewCmd.ExecuteNonQuery();

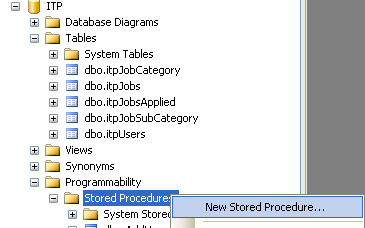
NewCmd.Dispose();

NewCon.Close();

}

* 1. **Stored Procedures**

A stored procedure is a set of SQL commands that has been compiled and stored on the database server*.*

To create Stored Procedures, expand the database, expand the node ‘**Programmability’**, right click ‘**Stored Procedures**’ and select ‘**New Stored Procedure**’.

The below procedure is used to add a user to the database.

CREATE PROCEDURE [dbo].[AddUser]

-- Add the parameters for the stored procedure here

@email varchar(max),

@password varchar(max),

@fullName varchar(max),

@phoneNumber varchar(max)

AS

BEGIN

-- SET NOCOUNT ON added to prevent extra result sets from

-- interfering with SELECT statements.

SET NOCOUNT ON;

-- Insert statements for procedure here

INSERT INTO [itpUsers]

([email]

,[password]

,[fullName]

,[phoneNumber])

VALUES

(@email ,@password ,@fullName ,@phoneNumber)

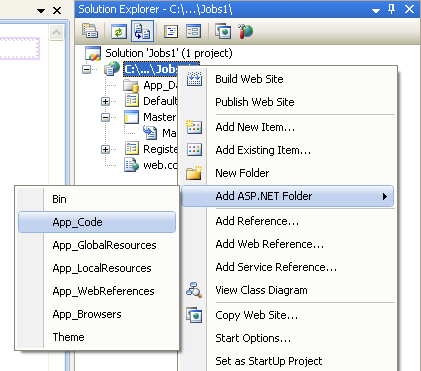
END

* 1. **Adding encryption for passwords**

Since we are storing passwords we need to make sure data entered by the user cannot be seen by anyone. This includes the system administrators as well. In order to do this we can encrypt the password the user provides before storing it in the database.

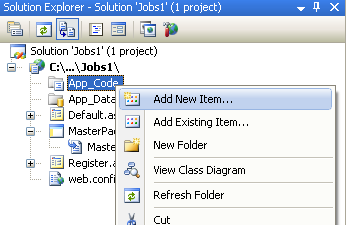
We will use a separate **static** class to enable easy access to the method.

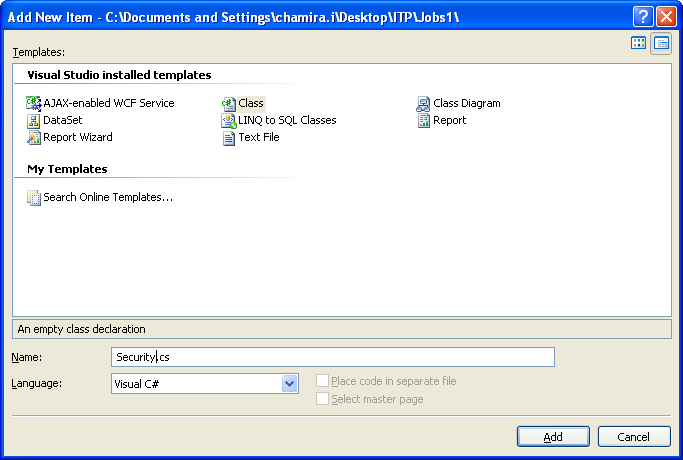
The following steps illustrate how to achieve this feature easily using the MD5 encryption.

1. First add the **App\_Code** folder to the project.

As a best practice in web development all classes are stored inside this folder.

1. Right click App\_Code 🡪 Add New Item 🡪 select Class and name it as **Security.cs**





There will be a constructor when the class is created. Delete it, because no use of a constructor for a static class.

public static class Security

{

public static string Encrypt(string input)

{

//http://blog.brezovsky.net/en-text-2.html

System.Security.Cryptography.MD5CryptoServiceProvider x = new System.Security.Cryptography.MD5CryptoServiceProvider();

byte[] bs = System.Text.Encoding.UTF8.GetBytes(input);

bs = x.ComputeHash(bs);

System.Text.StringBuilder s = new System.Text.StringBuilder();

foreach (byte b in bs)

{

s.Append(b.ToString("x2").ToLower());

}

string password = s.ToString();

return password;

}

}

The reason we are using a static class is because we do not need to create any objects of the class. We can directly call the function from anywhere since we have declared it as public.

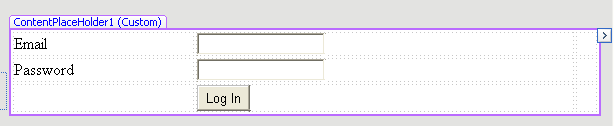
1. In order to make use of this class let’s encrypt the password before it is sent to the stored procedure as a parameter.

NewCmd.Parameters.Add("@password", Security.Encrypt(txtPassword.Text));

Alter this line of code which is inside the **btnSubmit\_Click** event.

* 1. **Link this page to the Master Page**

Finally link this page to the Link Button Register in Master Page (Use the property **PostBackUrl**).

1. **Creating Login Page**
2. The next step is to create a login page that will allow users to login to the system. Create a new page named **Login.aspx**.
3. Be sure to select the master page.
4. Design the page as follows.

Text Boxes

txtEmail, txtPassword (Set property **TextMode** to **Password**)

Button

btnLogIn 🡪Text – Log In, ValidationGroup – login

Use two **RequiredFielValidators** to ensure both **Email** and **Password** Fields are filled by the user. (ValidationGroup – login)

**4.1 Handling User Authentication**

When a user is trying to login to the web site, authentication is essential to verify that the user has necessary permission to use the site contents.

Hence when the ‘**Log In**’ button is clicked first Email and password are verified using the user information table in the database.

**4.1.1 User Information class with Getters & Setters**

User information is very important when it comes to web sites because during the total time period that the user is communicating with the web site (Session), his information should be stored with the session to control his actions according to his permission level.

We create a separate class to keep user information, so that it is easy to keep track of user information.

As done previously, add a new class named **UserInfo.cs** to the **App\_Code** folder. This class is used to maintain user information for a logged user.

public class UserInfo

{

private string \_userId;

private string \_email;

private string \_fullName;

public UserInfo(string userId, string email, string fullName)

{

\_userId = userId;

\_email = email;

\_fullName = fullName;

}

public string UserId

{

get { return \_userId; }

}

public string Email

{

get { return \_email; }

}

public string FullName

{

get { return \_fullName; }

set { \_fullName = value; }

}

}

Usage of this class and storing user information using **UserInfo** objects is illustrated in section 4.2.1, inside **isAuthenticated** method.

**4.2 *Log In* button click event**

When ‘Log In’ button is clicked user should be authenticated and redirected to the first page for logged users if that’s a valid user.

Double Click the Login button and you will be taken to the **Login.aspx.cs**.

Before we implement the button click event let’s implement a method to authenticate the user.

**4.2.1 *isAuthenticated* method and ASP.NET Sessions**

Sessions are used to retain data for unique user sessions. The following code snippet illustrates the **isAuthenticated** method and will introduce sessions (Be sure you implement this method below the **btnLogin\_click** method and **not** inside it). Import these classes as well.

private bool isAuthenticated(string email, string password)

{

string ConStr = ConfigurationManager.ConnectionStrings["ConString"].ConnectionString;

SqlConnection NewCon = new SqlConnection(ConStr);

NewCon.Open();

SqlCommand NewCmd = NewCon.CreateCommand();

NewCmd.CommandType = CommandType.Text;

NewCmd.CommandText = "select \* from dbo.itpUsers where email = '" + email + "' and password = '" + Security.Encrypt(password) + "'";

SqlDataReader dr = NewCmd.ExecuteReader();

if(dr.HasRows)

{

while (dr.Read())

{

**UserInfo ui = new UserInfo(dr["userid"].ToString(), dr["email"].ToString(), dr["fullName"].ToString());**

**Session["loggedUser"] = ui;**

}

NewCmd.Dispose();

NewCon.Close();

return true;

}

NewCmd.Dispose();

NewCon.Close();

return false;

}

using System.Configuration;

using System.Data.SqlClient;

using System.Data;

* + 1. **Button click event**

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

{

if(isAuthenticated(txtEmail.Text.Trim(),txtPassword.Text))

{

Response.Redirect("Default.aspx");

}

else

{

Session.Clear();

}

}

* 1. **Link this page to the Master Page**

Finally link this page to the Link Button Login in Master Page (Use the property **PostBackUrl**).

1. **Master Page code behind**

When designing the master page **Search** and **Applied Jobs** buttons were put into a Panel (pnlLoggedIn) where its property ‘Visible’ was set to false. This was done to ensure that only logged in users have the option to search for posted jobs and apply.

Since we have developed a way to keep track of user information we can develop the code behind to check whether user is logged in and if so, display the panel which includes **Search** and **Applied Jobs** link buttons. This can be done in the page load event of the master page.

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

{

if (!IsPostBack)

{

if (Session["loggedUser"] != null)

{

UserInfo ui = (UserInfo)Session["loggedUser"];

lblUserName.Text = "Welcome" + ui.FullName;

}

}

if (Session["loggedUser"] != null)

{

pnlLoggedIn.Visible = true;

}

else

{

pnlLoggedIn.Visible = false;

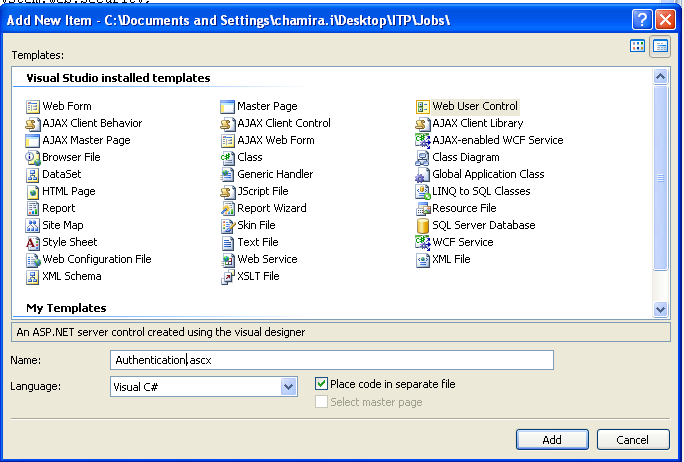
}

}

1. **User Controls**

User controls are reusable user components. Hence similar things that must to be done (like verifying users authentication) in every page can be done using these kind of components rather than developing them everywhere.

1. Next step is to check for authentications on each and every web page. Because every site requires security. Here we discuss on how to enable access for logged in users and anonymous users.

Right click project 🡪 Add new Item 🡪 select **Web User Control** and name it **Authentication.acsx**

1. In the page load of the Authentication.ascx we have to check the status of a user. By using a user control we do not have to check this on every page.

Open **Authentication.acsx.cs** page and develop the **Page\_Load** event as follows.

protected void Page\_Load(object sender, EventArgs e

{

if (Session["loggedUser"] == null)

{

Response.Redirect("Login.aspx");

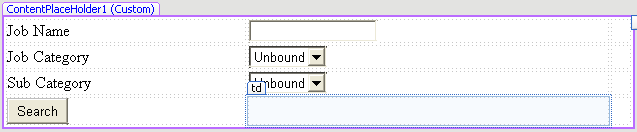
}

}

We check the Session. If the session is null the user will be redirected to the login.aspx page. This is a very basic way of checking for authentication.

Note: Drag and drop this user controls to Search.aspx and AppliedJobs.aspx pages once they are created (because users must be logged in to the system in order work with these pages).

1. **Creating Search Page**

Next step is to create a search page for the users. The logged in users should be able to access the search page. Create **Search.aspx** page and design it as follows.

Text Boxes

txtJobName

Drop Down List

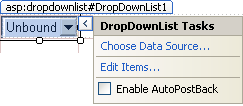
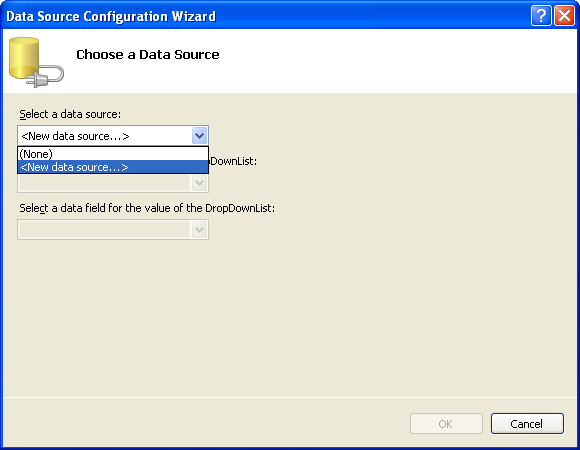
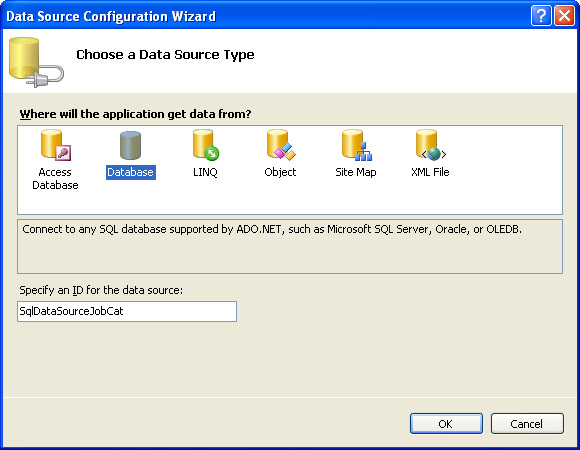
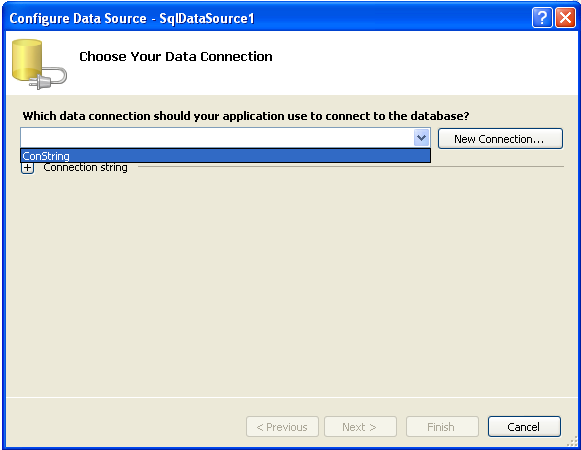
ddlJobCat, ddlSubCat

Button

btnSearch 🡪Text – Search

Afterwards we need to populate the two combo boxes with data. The relationship between Category and Sub Category is **1 to many**. So the sub category combo box data should change upon the selection of the Job Category combo box. In windows form we filled combo boxes using code snippets. For this exercise we will introduce **SQL DataSources**.

**7.1 Database Access via DataSources**

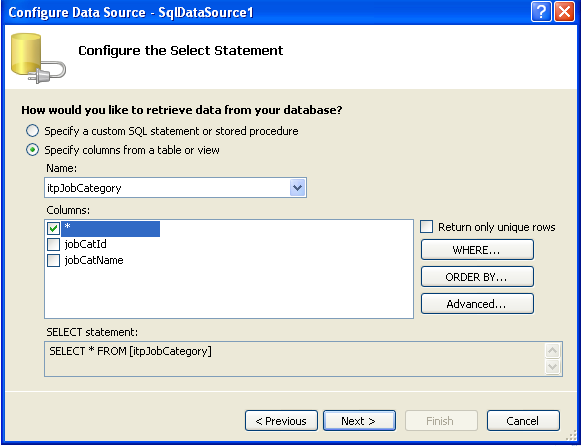
1. Click on the ‘>’ mark next to the ‘**Job Category**’ drop down list and you will be prompted with a context menu. Click ‘Choose Data Source’
2. Chose <New Data Source> from the Data Sources drop down list.
3. Select Database and specify an ID for the data source (SqlDataSourceJobCat).
4. In the next window click on the drop down and select the ‘**ConString**’ connection string. This is derived from the **web.config** file. All the connection strings stored in the **web.config** file will be displayed here. If nothing appears simply click on new connection and add your own connection.

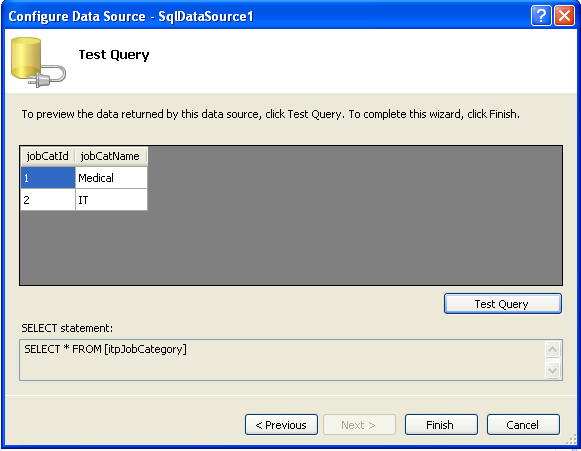
*For this lab we are going to use the* ***ConString*** *connection string. But for your own use, follow below instructions to create a new connection.*

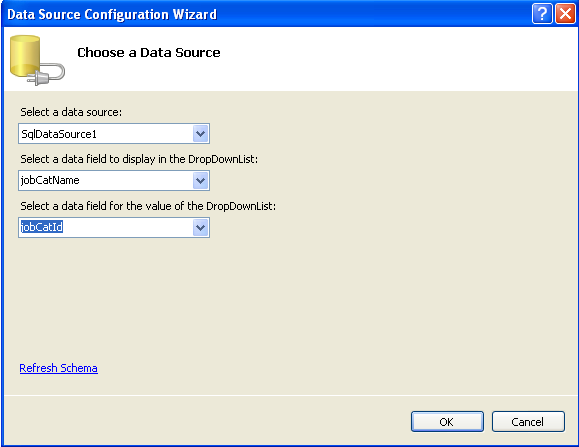
*New Connection -> Select ‘Microsoft SQL Server’ ->Click Continue*

*Add Connection Window*

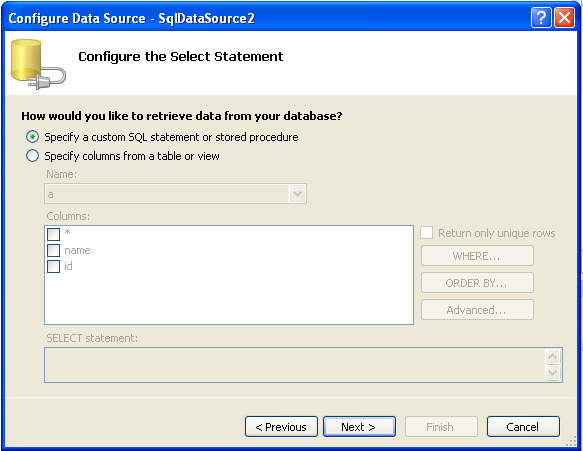
*Select/Add Server Name -> Add the SQL Authentication -> And after selecting the Database click ‘Test Connection’*

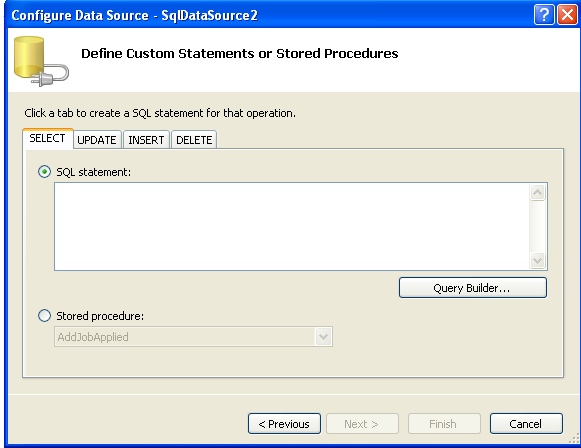
1. Upon selecting the connection string and clicking ‘Next’ you will be prompted with the following screen to select the table and required columns.
2. Click ‘Test Query’ to run your query and then click ‘Finish’.

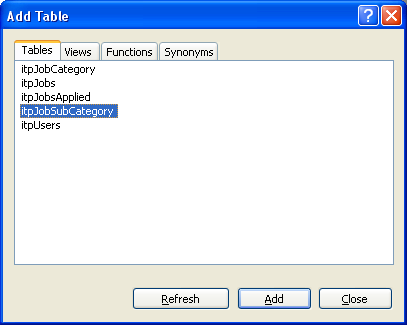
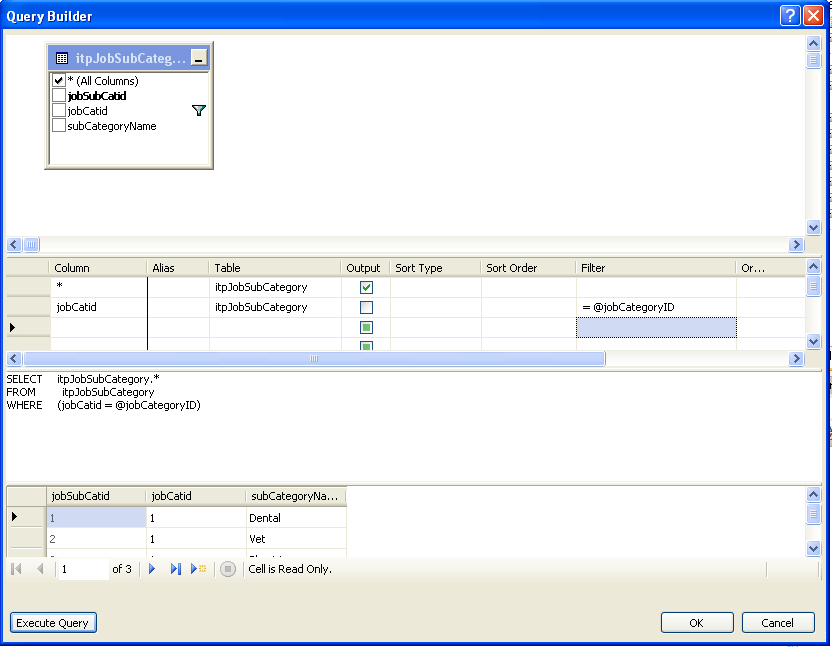


1. In the next window select the created Data Source, **JobCatName** as the ‘field to display’ and **jobCatId** as the ‘field for the value’.

‘Field to display’ is the Attribute that the combo box will display to the users. The ‘field for the value’ is used for data manipulation purposes (Storing as the primary key). Then click OK.

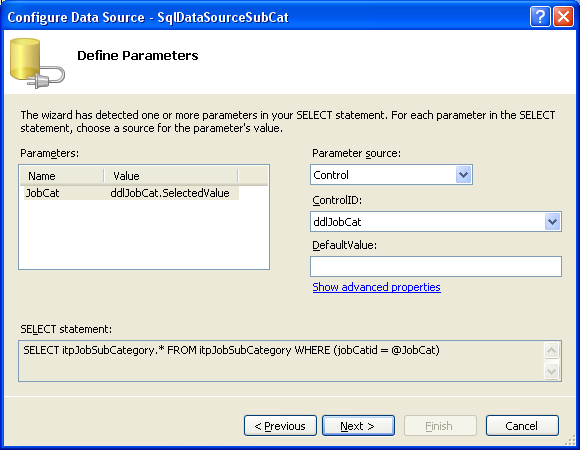
1. Next to display data in the **SubCategories** drop down list. Repeat what you did to add the data source to the **JobCategories** drop down list from steps 1 to 5 (give data source ID as SqlDataSourceSubCat). When you are in screen shows in step 5 it is required to specify a custom SQL statement rather than taking values from a table directly. Because the requirement is it fill sub categories which are under the selected Job Category.
2. When defining a custom query, for this scenario we will focus on a select statement with a parameter because we need to pass the selected ‘Job Category’ in previous drop down list. Click ‘Query Builder’.

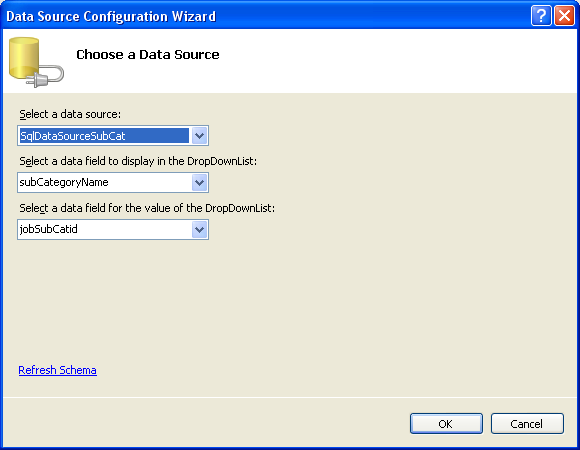


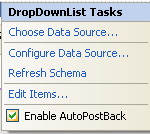
1.  When you are in the window to select tables select and **JobSubCategory** table.
2. Select all columns and filter by jobCatId. Provide the parameter name as **@jobCat** (equal sign is auto generated). Notice the way to provide the parameter name (followed by the ‘@’ sign). Press ‘Execute Query’ and test the query. Click OK.

You will be taken to the window shown in step 9, but Select statement is generated. Then click ‘Next’. You will be taken to the next window.

1. In the next window, the defined parameter in the select query should be specified. It will be passed by the Job Category drop down list which is a **Control**. Hence select the ‘Parameter source’ as ‘Control’. Then specify the ID of the control (ddlJobCat)



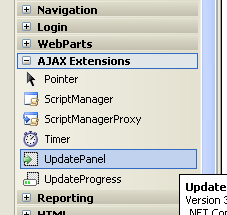
Click Next.

1. **subCategoryName** as the ‘field to display’ and **jobSubCatId** as the ‘field for the value’.
2. Notice that even if you change the drop down entry of the ‘Job Category’ combo box ‘Sub Category’ does not changes according to that. In order to cause a change we must make a server call. In order to do that simply click on the ‘>’ button in the **ddlJobCat** and tick the **EnableAutopostBack** check box. Notice that we should make server call when the job category is changed. Hence do this for the Job Category combo box.

**7.2 Use of AJAX**

Notice that once you change the value of the job combo box a post back occurs. Since you are running locally you wouldn’t notice a lag. But it is similar to clicking a button online. It will take time when it comes to a real Client-Server scenario. To avoid that and give the user a better experience, AJAX was introduced (there are more uses of it than the above mentioned)

1. The simplest way of getting AJAX is to drag and drop the update panel. Surround your other controls that you want to AJAXify with **UpdatePanel** tag.

Important: Add a **ScriptManager** control on top of the page or to the master page first

<asp:UpdatePanel ID="UpdatePanel2" runat="server">

<ContentTemplate>

.

.

.

<asp:DropDownList ID="DropDownList2" runat="server"

DataSourceID="SqlDataSource2" DataTextField="subCategoryName"

DataValueField="jobSubCatid">

</asp:DropDownList>

.

.

.

</ContentTemplate>

</asp:UpdatePanel>

Note: notice that another XML tag is there inside UpdatePanel tag. That is something you need to enter. Whatever content wrapped up with an update panel should be inside ContentTemplate tag.

1. We just covered the basics of AJAX. There is more to it. Do not wrap huge forms with the Update Panel. This will slow down the process. For that we can use triggers. It’s something you should research on your own and use for your 2nd year project.

Instead of adding the **ScriptManager** on this page we can add it to the master page.

Consider: Search button should be outside the Update Panel. Else Data would not be displayed in the grid views that we are about to create in the next step. Else you have the option of keeping the button inside the panel but you need to include the grid views inside the panel also.

**7.3Using ASP.NET Grid Views to display search results**

1. Next focus is on the **GridView** control. We will be discussing about this control in future lab sessions as well. In this part we will focus on displaying the search results and viewing jobs. Drag and drop two **GridViews** and name one **gvSearch** and the other **gvManual**.
2. Double click the Search button and implement the code as follows.

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

{

string ConStr = ConfigurationManager.ConnectionStrings["ConString"].ConnectionString;

SqlConnection NewCon = new SqlConnection(ConStr);

NewCon.Open();

SqlCommand NewCmd = NewCon.CreateCommand();

NewCmd.CommandType = CommandType.Text;

NewCmd.CommandText = "select \* from [itpJobs] where jobName like '%" + txtJobName.Text.Trim() + "%' AND [jobCategory] = " + ddlJobCat.SelectedValue + "AND jobSubCategory = " + ddlSubCat.SelectedValue;

SqlDataAdapter da = new SqlDataAdapter(NewCmd);

DataTable dt = new DataTable("SearchResult");

da.Fill(dt);

gvSearch.DataSource = dt;

gvSearch.DataBind();

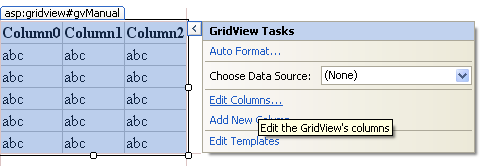
gvManual.DataSource = dt;

gvManual.DataBind();

NewCmd.Dispose();

NewCon.Close();

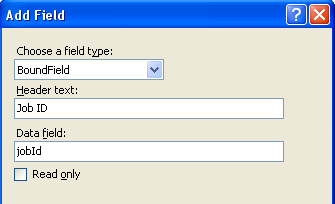
}

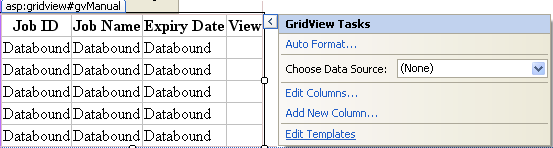
1. Run the Search page and notice that both grids will display result. Both grids generate columns automatically according to the columns that are passed by the select statement.
2. Select **gvManual** grid view and set the property **AutoGenerateColumns** to **False**. Run the program and notice the **gvManual** grid view will not display search results this time.
3. Click on the ‘>’ mark associated to the **gvManual** grid view and select ‘Add New Columns’.
4. In the next window select add fields as follows.

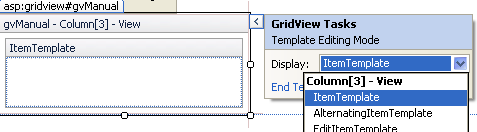
Header Text – Header of the column

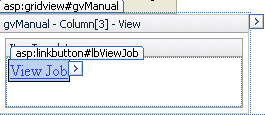
Data field – database column name which is to be bounded to this column

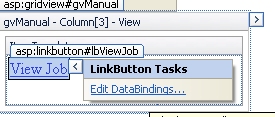
|  |  |  |
| --- | --- | --- |
| Field type | Header text | Data field |
| BoundField | Job ID | jobid |
| BoundField | Job Name | jobName |
| BoundField | Expiry Date | expirationDtae |
| TemplateField | View | - |



1. Click on the ‘>’ mark associated to the **gvManual** grid view again and select ‘Edit Columns’. You will see all the created columns.
2. Click on the ‘>’ mark associated to the **gvManual** grid view again and this time select ‘Edit Templates’.
3. There are many templates that can be modified. In this lab session we will discuss the **ItemTemplate**.



1. We want to give a link to a full descriptive page of a particular job which results in that particular row of the grid view. In order to do so we have to redirect the user to the relevant **Job** page. We can do this by binding the **jobid** in to a query string. First drag and drop a **LinkButton** in to the **ItemTemplate** (button ID – lbViewJob, Text – View Job).
2. Click on the ‘>’ mark associated to the **View Job** LinkButton and select ‘Edit Templates’.



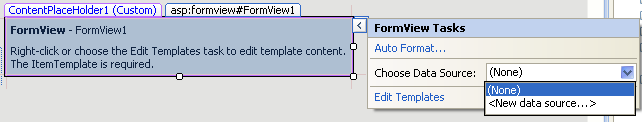
1. Tick the check box ‘Show all properties’ 🡪 select PostBackUrl 🡪in ‘Code expressio’ area paste the following

"~/Job.aspx?jobid=" + Eval("jobId")

Note: **Job.aspx** page is yet to be created

**7.4Creating Job.aspx page**

1. Create a new page named **Job.aspx**
2. Drag and drop a **FormView** control to the page.
3. Click on the ‘>’ mark and in Data Source section select <New data source>.

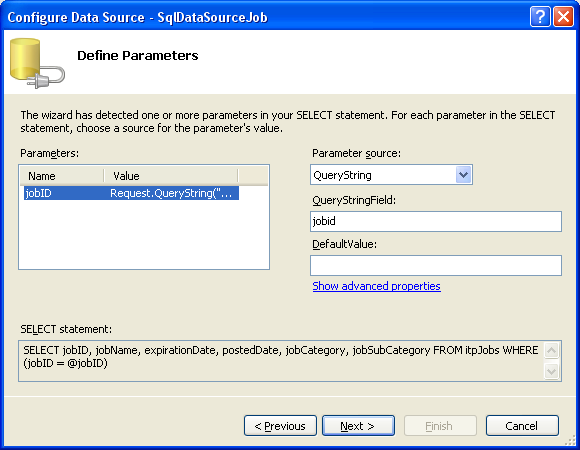


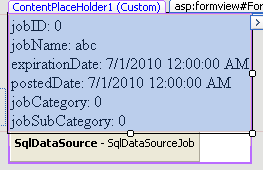
1. Follow the steps in section 7.1 to create the data source. You can select the **ConString** connection string this time also.
2. Specify a custom select statement and build the following statement using **itpJobs** table.

SELECT jobID, jobName, expirationDate, postedDate, jobCategory, jobSubCategory FROM itpJobs WHERE (jobID = @jobID)

1. The next window will let us define parameters. This time the parameters are not passed from a control, instead it is passed from a query string (from the previous page – Search.aspx). Hence select **QueryString** as Parameter Source, **jobid** as QueryStringFoeld

Note: **jobid** is the parameter passed from the previous page in section 7.3: step 12. (Please note that it is **not** the value inside the brackets)

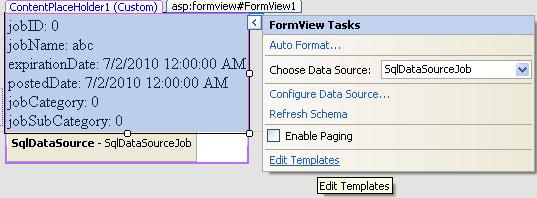
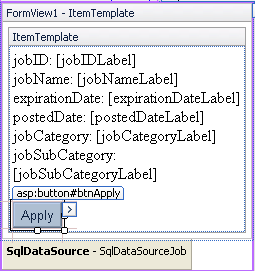
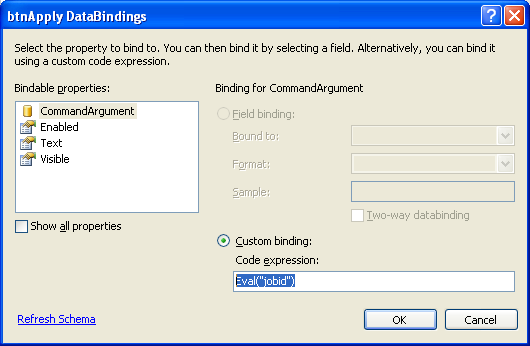
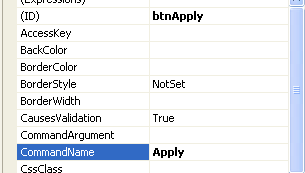


Notice the Parameters box. We are using the QueryString to populate the jobID parameter.

As you can see the **FormView** will be automatically populated. You can edit this accordingly.

Check the Databindings and the EditTemplate option in Form View control.

**7.5 Option to apply for jobs in Job.aspx page (use of ItemCommand)**

1. Next step is to create an applying option to the job.
2. Go to ‘Edit Template’ in the **FormView** and add a button. Name it **Apply** (**btnApply**).
3. Click on the ‘>’ mark associated with Apply button and select ‘Edit DataBindings’.
4. Select Bindable property **CommandArgument** and select radio button Custom binding, set the code expression as **Eval(“jobid’)**.
5. Set the button property **CommandName** 🡪 **Apply**
6. Now click “End Template Editing” and view the events of the **FormView** control. Double Click the **ItemCommand** event. Inside the event method develop the below code (FormView1\_ItemCommand.

protected void FormView1\_ItemCommand(object sender, FormViewCommandEventArgs e)

{

if (e.CommandName == "Apply")

{

Label lbl = FormView1.Row.FindControl("jobNameLabel") as Label;

string jobID = e.CommandArgument.ToString();

UserInfo ui = (UserInfo)Session["loggedUser"];

string ConStr = ConfigurationManager.ConnectionStrings["ConString"].ConnectionString;

SqlConnection NewCon = new SqlConnection(ConStr);

NewCon.Open();

SqlCommand NewCmd = NewCon.CreateCommand();

NewCmd.CommandType = CommandType.StoredProcedure;

NewCmd.CommandText = "AddJobApplied";

NewCmd.Parameters.Add("@userID", ui.UserId);

NewCmd.Parameters.Add("@jobID", jobID);

NewCmd.ExecuteNonQuery();

NewCmd.Dispose();

NewCon.Close();

}

}

Same execution will take place if this code snippet was developed for Apply button click event. But using this example will help you understand the usage of Item Commands.

Stored Procedure **AddJobApplied** is as follows.

CREATE PROCEDURE [dbo].[AddJobApplied]

-- Add the parameters for the stored procedure here

@userID int,

@jobID int

AS

BEGIN

-- SET NOCOUNT ON added to prevent extra result sets from

-- interfering with SELECT statements.

SET NOCOUNT ON;

-- Insert statements for procedure here

INSERT INTO [itpJobsApplied]

([userID]

,[jobID]

,[appliedDate]

)

VALUES

(@userID,@jobID ,GETDATE())

END

But still this web page will not work accurately because we have not checked whether a user is logged in, inside the code of this pageing, and hence there is no **userID**. This is where we use User Control that we created in section 6.

**7.6 Using *Authentication* User Control**

Drag and drop the Authentication user controller from the solution Explorer window to the top of the Search.aspx page.

After adding this control, if you run the web site even from the Search.aspx page it will redirect you to Login.aspx page. That is because in the Authentication control it is coded to do so if the session is null.

* 1. **Link this page to the Master Page**

Finally link this page to the Link Button Search in Master Page (Use the property **PostBackUrl**).

***Eval and Bind***

[*http://forums.asp.net/p/1097781/1661386.aspx*](http://forums.asp.net/p/1097781/1661386.aspx)

.

Create a new page to view your applied jobs. Name it ‘AppliedJobs.aspx’. This page should display all jobs a particular user has applied for.

Exercise 1

1. Add a datagrid and populate it with the applied jobs for the logged in user, using a SQL datasource.
2. Add a Delete column
3. Do a Delete using DataKeyNames and SQL Datasource

*Use authentication user control.*

*Link to master page.*

Exercise 2

1. Display the logged in user’s name on the Master Page using the UserInfo class once he logs into the page.
2. Hide the “Search” and “Applied Jobs” links for anonymous users in the master page.