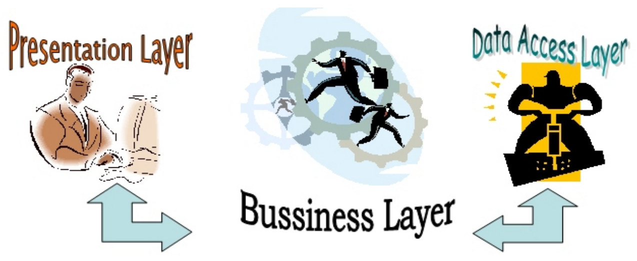
Vol. 1, Issue . 1 / February 2011, pp.173-176

Research Paper - Computer Science

Implementing 3-tier Architecture in C#.net

- Mr. Mahesh Sarvade
M.C.M.
Hitachi Consulting, Pune.



A long ago programmers used to write much lines of code and various logical functions and procedure to support reuse of their code. But in modern world you have to go beyond just reusability and mere programming logic. Today's applications demands vast data support, User Interface Improvisation, data security and dynamic business needs. Here the 3-tier applications came into picture to solve developer's problem. Separating the tiers in applications helps programmer to concentrate on very particular logic and build strong and dynamic programming architecture to support business applications. Theoretically many of you have read about concept and working of 3-tier architecture so will go directly to implementation of 3-tier architecture in c#.net.

As you already know there are 3-tiers in architecture each of which is as follow.

1) Data Access Layer:

The layer which is responsible for accessing the data from databases. You can write your connection string, various data accessing functions like Executing commands and filling of datasets.

2) Business Layer or Logic Layer:

This is layer where you can write the business logic such as calculation of Salary etc. The Layer consists of various properties and stored procedure accessing functions for selection, insertion, updation and deletion of data.

3) Presentation Layer or Application Layer:

The application layer takes care of your User Interface representation, data display and event handling.

Now we will see how to write codes for following layer. We will consider a login form scenario for implementing our 3-tier. User will enter his username and password and then logged onto system. We are using following resources to develop our project.

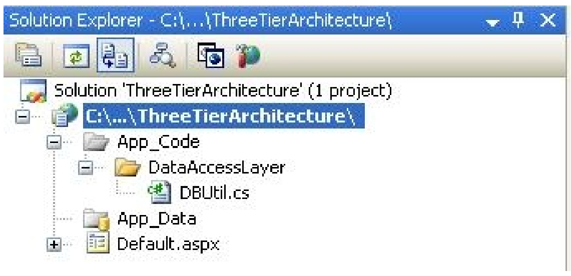
Database Name: dbAuthenticate

Table Name:

Fields : (Uname, pwd, URole, Status)

A) Data Access Layer:

We will create a new website project named as "ThreeTierArchitecure" and add App_Code folder to the project. In App_Code we will create the new folder named as "DataAccessLayer" And there we added a code file named as "DBUtil.cs".


Solution Explorer - C:\...\ThreeTierArchitecture\
Solution 'ThreeTierArchitecture' (1 project)
C:\...\ThreeTierArchitecture\
App_Code
DataAccessLayer
DBUtil.cs
App_Data
Default.aspx

In this dbutility file we can add code for connection string initialization from web.config and various data accessing functions. A typical dbutils file contains following code.

```
public class DBUtils
  {
     SqlConnection con = new SqlConnection();
     SqlCommand cmd = new SqlCommand();
     SqlTransaction sqltrans;

     #region "ConnectionSettings"

      /// <summary>Gets the Sql Connection String by using Web.Config file settings.The name to the connection string must be given as "ProjectConnectionString"</summary>

     public string DBgetConnectionString()
     {
        string ConnectionString = "";
        try
        {
              ConnectionString = System.Configuration.ConfigurationManager.ConnectionStrings["ProjectConnectionString"].ToString();
        }
        catch (Exception ex)
        {
           throw ex;
        }
        return ConnectionString;
     }

           /// The constructor which initializes the connection.
     public DBUtils()
     {
                    con.ConnectionString = DBgetConnectionString();
        cmd = con.CreateCommand();
     }

           /// <summary>"Opens the Database Connection."</summary>
     private void DBOpenConnection()
     {
        try
        {
              if (con.State == ConnectionState.Closed || con.State == ConnectionState.Broken)
           {
             con.Open();
           }
        }
        catch (Exception ex)
        {
           throw ex;
        }
     }

      /// <summary>"Close the Database Connection."</summary>
     public void DBCloseConnection()
     {
        con.Close();
     }
     #endregion

     #region "ExecuteOnlyModules"
              /// <summary>"Executes the Passed Query string."</summary>
     public int DBExecuteNonQuery(string sql)
     {
        int rowAffected = -1;
        DBOpenConnection();
        cmd.Connection = con;
        sqltrans = con.BeginTransaction();
        cmd.Transaction = sqltrans;
        cmd.CommandText = sql;
        try
        {
                                     rowAffected = cmd.ExecuteNonQuery();
           DBCommitTransaction();
        }
        catch (Exception ex)
        {
              throw ex;
        }
        finally
        {
         cmd.Dispose();
      }
      return rowAffected;
   }

            /// <summary>"Committed the transation to server."</summary>
   private void DBCommitTransaction()
   {
      sqltrans.Commit();
      sqltrans = null;
   }

            /// <summary>"Executes the Passed Query string and return the DataReader."</summary>
            public SqlDataReader DBExecuteReader(string sql)
   {
      SqlDataReader reader = null;
      DBOpenConnection();
      cmd.Connection = con;
      cmd.CommandText = sql;
      cmd.CommandType = CommandType.Text;
      try
      {
         reader = cmd.ExecuteReader(CommandBehavior.CloseConnection);
      }
      catch (Exception ex)
      {
         throw ex;
      }
      finally
      {
         cmd.Dispose();
      }
      return reader;
   }
 #endregion

   #region "GetDataModules"
            /// <summary>"Fill the Dataset by passing SQL Command and ref Dataset.Returns Dataset on execution."</summary>
      public void DBGetDataSet(SqlCommand pcmd, ref DataSet ds)
   {
      SqlDataAdapter da = new SqlDataAdapter();
      DBOpenConnection();
      pcmd.Connection = con;
      try
      {
         da.SelectCommand = pcmd;
         da.Fill(ds);
      }
      catch (Exception ex)
      {
         throw ex;
      }
      finally
      {
         cmd.Dispose();
         da.Dispose();
      }
   }
 #endregion
```

For current scenario we only required one function i.e. **"DBGetDataSet".** But other functions are given for practice. Now we moving onto next layer which is business layer.

B) Bussiness Layer or Logic Layer:

As stated earlier the business layer consisting of business logic such as calculations. We can also write properties and stored procedure logic. Same way as we added DataAccessLayer we can add this BusinessLayer in our web project. The code filename is **"CtrlLogin.cs".**

In this file we can declare the two private variables named as strUsername and strPassword. We can encapsulate these fields with get and set properties. So we can use this public property to pass to functions or procedures. The typical code for this is as follows.

```
/* declaration of private variables */
   private string strUsername;
   private string strPassword;

/* encapsulate these variables to set public properties */
```

```
 public string Password
 {
    get {return strPassword; }
   set {strPassword = value; }
 }
public string Username
{
   get {return strUsername; }
   set {strUsername = value; }
}
```

/* accessing the stored procedure to get resultset from DButil file. Here we are creating the object of DButil class and then accessing the method named as "DBGetDataSet" where we pass sqlcommand and reference dataset which gives us resultset contains Roles and status of user.
***/**

```
   public DataSet sp_CheckLogin(CtrlLogin objLog)
   {
      DBUtil Odbutil = null;
       Odbutil = new DBUtil(); // creating object of dbutil class
      DataSet ds = new DataSet();
      try
      {
         SqlCommand cmd = new SqlCommand();
                             cmd.CommandType = CommandType.StoredProcedure;
           cmd.CommandText = "sp_CheckLogin"; // stored proc name

            // passing parameters to stored proc.
 cmd.Parameters.AddWithValue("@Username", objLog.Username);
        cmd.Parameters.AddWithValue("@Pwd", objLog.Password);

         Odbutil.DBGetDataSet(cmd, ref ds);
      }
      catch (Exception ex)
      {
      }
      finally
      {
         if (Odbutil != null)
         {
            Odbutil.DBCloseConnection();
         }
      }
      return ds;
```

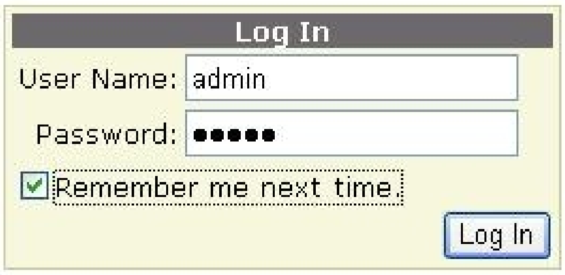
// returing dataset which we use to check role and authenticate in presentation layer

```
   }
```

We can use this procedure in presentation layer for checking authentication.

C) Presentation Layer:

The presentation layer involved in developing more sophisticated and helps ultimate user browsing experience. In our scenario we assume the login control on page which accept username and password and if user is valid then redirect him to profile page.


Log In
User Name: admin
Password: ●●●●●
Remember me next time. ✓

Using this typical login control you can logged onto system. You can write following code to authenticate the user.

```
protected void Login1_Authenticate(object sender, AuthenticateEventArgs e)
   {
      try
      {
         DataSet ds = new DataSet();
         //Creating object of Business layer class
               CtrlLogin objLogin = new CtrlLogin();
                   objLogin.Username = Login1.UserName;
                   objLogin.Password = Login1.Password;
```