Spring 2015
.NET Web Application Development
Homework 2

- Due Friday, February 27, 2015 at 8:00 PM Eastern Daylight Time.
- The homework must be submitted entirely through NYU Classes—do not send by email. Late submissions turned in within one (1) day after the deadline will incur a 15 point penalty. Thereafter, late submissions will not be accepted for credit. Submissions using NYU Classes are a multi-step process. Make sure you complete the *entire* submission process in NYU Classes before leaving the page.
- Answers to questions should be submitted to NYU Classes "inline." Programming solutions should be compressed into a .ZIP file and attached to your NYU Classes submission.
- There are 100 possible points. See the brackets for the number of points each question is worth.

Login Controls [30] Logging into a web site is a fundamental operation that most web sites need to support. Although the .NET library provides login controls out of the box, we are going to design our own login functionality.

1. Open Visual Studio and from the file menu, *New*, *Project*.

2. Select *Visual C#* from the list of installed template groups and select *Web* underneath that. Click on the *ASP.NET Web Application*.

3. On the bottom of the dialog, choose a name and location for the project and click OK.

4. On the next screen click the *Web Forms* icon, ensure that *Host in the cloud* is checked. Select *Website* from the dropdown list box. Click *Manage subscriptions* and sign on with your Microsoft ID. If prompted to choose either "Work or School Account" or "Personal Account," choose Personal Account. After you log in, ensure your Azure Subscription ID is checked. (You set this up earlier in the course). Click OK to close the subscription dialog.

5. You will be brought back to the project creation screen. Click OK. You must give your site a unique name. Select the "East US" region from the dropdown list. Under "database server" select *Create New Server* and type in a username and password for your database. (This will be different from your Microsoft ID and password.) **Do not put an @ symbol in the username for your database server, such as an email address. This symbol has a special meaning.** Write down the name of the database server, your username and password. You'll need this later. Click OK and the project will be created. It will take a few moments for Visual Studio to create the project, since the database is created on Azure as part of the project initialization.

6. In Solution Explorer, remove all of the .aspx files (About, Default, Contact). Add a new Web Form by right-clicking on the project, *Add* and then *Add New Item*, and click *Web Form*. Name the page Default.aspx. This will serve as your login page, which will appear automatically when you navigate to the root directory of your web site.

7. Following the same steps as above, create a second screen called Main.aspx. This page is going to hold the actual content of your site (the content that users are coming to your web site to see).

8. Going back to Default.aspx, go to your Toolbox (if not showing, go to the View menu and select it). Under the HTML expander, drag the *Input (Text)* and *Input (Password)* controls onto your form for your username and password, respectively. Label these text boxes "Username" and "Password" for the user. Under the *Standard* expander in the Toolbox, drag a Button control onto the form. A button, labeled "Login," should appear

underneath. Be sure to pick useful ID names for all controls, since each ID will also serve as the name of the object in C#. Style the components using HTML and CSS to look like a typical login screen.

9. Inspect the Default.aspx file under the *Source* view in Visual Studio and you'll notice Visual Studio added opening and closing *form* tags. Make sure all of the login components are enclosed within this tag. Add the attribute `runat="server"` to both text boxes to make them server controls. This will allow you to view and change the properties of the controls programmatically in C# code.

10. In the *Designer* view for Default.aspx, double-click the Button control. This will create and bring you to a button click event handler. (No need to subscribe to the event manually. Because of the code `AutoEventWireup="true"` at the top of the .ASPX file, the .NET Framework will recognize event handlers according to how they are named.) This is where you'll insert your username/password validation logic. If the login is unsuccessful, you should post back to Default.aspx with an error message. If successful, you should redirect the user to Main.aspx. To redirect, issue the command `Response.Redirect("Main.aspx");` in your code-behind file. In the absence of this redirect, clicking the Button control will result in a postback.

11. Create a registration page, Register.aspx, where users can create a new account. Follow steps similar to Default.aspx. I recommend that you borrow the assets from Default.aspx to use on this page.

12. Log into Azure (www.azure.com). Once logged in, go to the management portal and find your previously created database in the list of SQL Databases. Under the "SERVER" column, click on the hyperlink for the server that hosts the database. Click the "CONFIGURE" item and add the IP address of your computer to the firewall so that the database will accept connections from your computer. You can do this by specifying your IP address as both the beginning and end of the IP address range. If you are on a local network, don't confuse your IP address on the internal network with your IP address as far as the Internet is concerned. If you are unsure, you can always discover your public IP address by Googling "What's my IP address?"

13. Now let's log into the database from Visual Studio. Go to Server Explorer (typically located in the upper left corner of Visual Studio, right next to the Toolbox tab). Click on the *Azure (...)* expander if it is not already visible, and under that click on the *SQL Databases* to reveal any databases that exist in your Azure account. You should see the database that was created earlier during the project setup. Right-click on it and select *Open in SQL Server Object Explorer*. You should see a dialog pop up with the credentials of the database server from earlier. Type the password and click *Connect*. The database server should now appear on the list in SQL

Server Object Explorer. Under that should be a folder called "Databases." Find the database you specified earlier and click on "Tables" underneath that.

14. Right click on "Tables" and select *Add New Table....* A new tab should open showing a list of table columns. You'll notice one entry already called *Id* which is a primary key. Change the name to "Username" and change the Data Type to "varchar(50)." In the space below, create a "Password" column of the same type. Make sure *Allow Nulls* is **not** checked for either. In the T-SQL Tab below that, change `[dbo].[Table]` to `[dbo].[Users]`. Click the *Update* button and then click *Update Database*. This action will create a database table called "Users" in Azure.

15. Your web pages can access the database above using the SqlDataSource control. In the Toolbox, drag a SqlDataSource control onto the Default.aspx page. Set the ID property to something meaningful. I will use the ID `UserData` for this example. Click on the small right arrow on the control and select *Configure Data Source*. Click *New Connection*. On the next screen, choose *Microsoft SQL Server (SqlClient)* as the data source. The database server name (something like `<name-you-chose>.database.windows.net`) may be in the next box below by default. If not, enter it. Click "Use SQL Server Authentication" and enter the database server username and password from before. **click Test Connection** before proceeding any further. If this doesn't work, don't continue. If it does work, a list of databases (right now containing only one item) will become available under the section entitled "Connect to a database." Select your database from the list. Note that this screen is finicky and is known to hang up your computer for many seconds if there are any issues with your configuration. In this situation, do not kill Visual Studio–just wait for the hanging to stop. Click *Next*. Here you should see the table columns you created earlier. Select the box next to the star and click *Advanced*. Click on "Generate INSERT, UPDATE, and DELETE statements." (This will generate the SQL commands to perform the basic database operations.) Click OK. On the next screen, click *Test Query*. Since your database table *Users* contains no data, you should see no information in the window, but you should also not receive an error. Click Finish. This will configure the SqlDataSource control and return you to the markup page.

16. Generally speaking, to insert items into the database programmatically in C#, use this code template:

```
UserData.InsertParameters.Clear();
UserData.InsertParameters.Add("Username", UsernameText.Value);
UserData.InsertParameters.Add("Password", PasswordText.Value);
UserData.Insert();
Response.Redirect("Default.aspx");
```

Note that `UserData` is the name of the SqlDataSource control. The quoted words `Username` and `Password` are the names of your table columns as you entered them earlier. Finally, `UsernameText` and `PasswordText` are the IDs of the text boxes we dropped on the screen earlier. The property `Value` gives you whatever is entered in these boxes. The call to `Insert` inserts a row into the table. Assuming this operation is successful, the user will be redirected back to Default.aspx immediately upon entering a new value. Note: you should gracefully handle the situation where the username already exists in the database, since the INSERT statement will fail in that case and throw a `SqlException`. You will have to catch the exception and report the outcome to the user gracefully.

17. To select a specific username from the database, first go into the SqlDataSource control markup and replace the default select statement `SELECT * FROM [Users]` with this: `SELECT * FROM [Users] WHERE ([Username] = @Username)`. In the content portion of the SqlDataSource markup, insert this:

```
<SelectParameters>
   <asp:Parameter Name="Username" Type="String" />
</SelectParameters>
```

Here is an example of code-behind C# code that would execute the select query in order to retrieve the password associated with a specified username:

```
UserData.SelectParameters.Clear();
UserData.SelectParameters.Add("Username", UsernameText.Value);
UserData.Select(DataSourceSelectArguments.Empty);

var myView = (DataView)UserData.Select(DataSourceSelectArguments.Empty);

if (myView.Count == 1)
    {
       var pass = myView[0]["Password"];
    }
```

The possible outcomes of the query are that 0 or 1 rows are returned (`myView.Count`). In this context, 0 means that the user doesn't exist and 1 means that it does. You will notice that `DataView` will be underlined in red initially because it resides in a namespace that hasn't been imported. To import a namespace, we write the `using` statements that you see at the top of the file. See if you can determine which namespace this belongs to.

18. It is now up to you to do the rest. Here are there requirements:

    - Default.aspx should be nicely styled. There should be a link to the registration page so new users can find the registration page easily.

- The Register.aspx page should require that user names must be at least 4 characters and passwords must contain at least 6 characters including a number and a special symbol. If the user tries to register under an existing user name, they should be informed that the username already exists. Otherwise, it should register the user.
- Register.aspx should require that both the username and password be at most 20 characters. Perform this validation prior to any database operations. Either prevent the user from violating this constraint or notify them (gracefully) that the constraint has been violated and ask them to fix the username and/or password and resubmit.
- After successful username and password registration, the user should be redirected to the login screen. It would be helpful if you also informed the user at the same time that registration was successful.
- Once added, username/passwords should not be deleted.
- It is not necessary to encrypt passwords, but feel free to add this in later as an additional exercise if you wish.
- On Default.aspx, 3 failed logins for the same username should result in a lockout for that username only (it should not be possible to log on with that username, even if the password is right). How to enforce this requirement is up to you. After 1 hour, it should be again possible to log in for that username.
- Once logged in successfully, the user should remain logged in until they log out. Add a "Log Out" button to Main.aspx for this purpose.
- Page Main.aspx should only display meaningful content if the user is logged in. Don't worry about encrypting the cookies for this assignment, but feel free to try encrypting cookies later if you have the time and desire. "Meaningful content" could be, for example, one of the web pages you wrote in homework 1. If the user is not logged in, Main.aspx should detect this and mention something to the user about them needing to log in first.
- Once fully debugged and working, publish your site to Azure. To do this, right-click on the project in Visual Studio and select *Publish...*. Check all of the tabs to ensure the information is correct. (If it's not correct and you publish, it can be a hassle to change it after the fact). Under settings, take a moment to read the database connection string. Ordinarily, we would not include the database password in the plain text connection string in real life. For this assignment, however, you may leave it as is.
- Visual Studio will remind you of the URL that your web site has been published to. Check out the web site to ensure it properly functions. The graders will look at this as part of your assignment submission. Submit this URL in the inline text portion of your assignment submission (`<name-you-chose>.azurewebsites.net`). Zip up and submit all project files as an attachment to the assignment.

C# [25]

1. Open Visual Studio and from the file menu, *New*, *Project*.
2. Select *Visual C#* from the list of installed template groups.
3. Select the *Console Application* template. Give the project a meaningful name.
4. Once the solution opens, go to the *Solution Explorer* tab, right-click the project (not the solution). The project has a C# icon next to it. Select *Add* from the context menu, then *New Item*.
5. From the template group entitled *Visual C# Items*, select the template labeled *Class* to add a new class file to the project. Give the file a meaningful name—it should end with the extension `.cs`. The new C# class file should now be open in front of you.
6. Write meaningful triple-slash comments for the classes, methods, and properties discussed below.
7. Right-click on the name of the namespace, which should be the same as your project name. Select *Refactor* and then type "NYU" as the new namespace. Accept the default options and click "Apply." Answer press "Yes" to the confirmation dialogue.
8. Use the same procedure to rename the class itself to `CarMakeModel`, which represents a vehicle of a particular make and model.
9. Ensure that class `CarMakeModel` has public visibility.
10. Add the following properties with public visibility. Use your own discretion in choosing exact types:
    - MakeName
    - ModelName
    - Year
    - Class (Minicompact, Subcompact, Compact, Midsize, Large)
    - Height
    - Length
11. Create a new class per the steps above, and name the class `Tire`. Give it these properties:
    - ManufacturerName
    - ModelName
    - Diameter

12. Create a new class per the steps above, and name the class `CarInstance`. Inherit this class from `CarMakeModel` above. This class represents a physical car of the above model. Provide the following properties with public visibility. Create a backing data member for each property, if needed:

    - OwnerFirstName (maximum 15 characters)
    - OwnerLastName (maximum 20 characters)
    - Color (Red, Blue, Green, White, Black, etc.)
    - IsRegistered
    - Tires (declared as type `IEnumerable<Tire>`, but implemented internally as `List<Tire>`. You must make the tire class public.)
    - Speed (maximum 200 MPH)
    - Heading (ranging from 0 to 359, degrees on a compass)

13. In class `CarInstance`, write the following public methods:

    - `turnLeft(int)` : subtract the specified number of degrees from the Heading. Any formal parameter whose value is ≥ 360 should result in no Heading change. Add 360 to the final value if the subtraction above resulted in a negative value.
    - `turnRight(int)` : add the specified number of degrees to the Heading. Any formal parameter whose value is ≥ 360 should result in no Heading change. Subtract 360 to the final value if the addition above resulted in a value > 360.
    - `accelerate(int)` : add specified value to the speed, maximum 100.
    - `brake(int)` : subtract specified value from the speed, minimum 0.

14. Declare the following events in class `CarInstance`:

    - BrakeLight : raised whenever the brake method is called.
    - NoLongerRegistered: raised whenever `IsRegistered` is set to false.
    - OutOfControl : raised whenever the car changes speed by more than 20 MPH *and* turns by more than 20 degrees (in either order).
    - MaxSpeed : raised whenever the car reaches its maximum speed from any lower speed.

15. The deliverable: proceed to the next step.

Unit Testing [20]

1. Right-click on the solution and go to *Add*, *New Project*. From the template group entitled *Visual C# Items*, select the template labeled *Unit Test Project* to add a project for unit testing.

2. In your new unit test project, you need to add a reference to the code under test. This is because the unit test is in a completely separate project from the code project—the test project won't know about any other assembles unless you specifically create the reference. Heres how to create the reference to a code project in the same solution:

   - Select the project in Solution Explorer.
   - On the Project menu, choose "Add Reference..."
   - In the Reference Manager dialog box, open the Solution node and choose Projects. Check the code project name and close the dialog box.

3. Write unit tests for the four (4) methods you wrote in the previous question. Each method to be tested should correspond to one (1) file in the unit test project. One is provided for you by default. To add the others, right-click on the Unit Test Project, *Add*, *Unit Test*. Rename the class and the methods accordingly. When writing your unit tests, be sure to:

   - Appropriately name your files and the unit test class.
   - Adorn the unit test class and methods with the `TestClass` and `TestMethod` attributes.
   - Write enough tests to ensure that your class works properly, paying particular attention to any special cases (e.g. reaching maximum speed, etc.)

4. Run all of your unit tests by going to the *Test* menu, *Run*, *All Tests*. Confirm that the code works or otherwise fix the code to make the unit tests run successfully. This is how professionals write code.

5. Before submitting your code, first clean your build directory. You may do this by going to the Build menu and selecting "Clean Solution." This will ensure that executables and other products of the build will not be included in the .ZIP file. The graders will inspect and build your full solution on their end.

The deliverable: a .ZIP file containing the entire directory structure in which your solution is contained.

Debugging [25] Attached to the homework assignment, find and unzip the contents of `HW2.zip` into a local directory. Be sure to fully extract the zip file first before loading the solution in Visual Studio. Load the *HW2.zip* solution. If you receive a message about loading a project from a trustworthy source, click the "OK" button.

Task 1:

1. In Solution Explorer, right-click on *HW2* and select *Set as StartUp Project*. In a solution containing multiple projects (as in this case), this determines which project should run first. Here, we are telling Visual Studio to run the *HW2* project.
2. Run the *HW2* program. Notice that the `mySwap` method doesn't work. Using the debugger, determine the issue and fix it.
3. The deliverable: an explanation of your fix. (Leave the solution open for the following questions.)

Task 2:

Now we're going to *remotely* debug a .NET web application. This means that the Visual Studio debugger is attached to a process running on the web server (not your local computer).

1. First, go to `http://buggybits.azurewebsites.net`, click on *Reviews*, and notice the two (2) bogus reviews. Click on the "Refresh" button and notice that the application crashes.
2. Let's investigate. First, deploy the BuggyBits project in the HW2 solution to your Microsoft Azure account by following the steps. Do this by loading the BuggyBits solution into Visual Studio, then right-clicking on the project (not the solution) and selecting "Publish Web Site."
   (a) Click the "Microsoft Azure Websites" button. If prompted, enter your Microsoft ID which is linked to your Azure educator pass. Even if you are not prompted ensure that Visual Studio is logged into your Microsoft account. You should see your initials in a colored box on the far upper right side of the Visual Studio window if you are. Click on the "Profile" tab if it is not already selected.
   (b) If you see a screen with a "New" button, click "New" to create a new Azure web site. Alternatively (if you don't see a "New" button), you may log into `azure.microsoft.com` and create a new Visual Studio Online account.
   (c) In the *Site name* box, enter a site name that includes your name, keeping in mind that the site name is global and everyone in the class must therefore use a unique site name. E.g. plock-nyu-hw2. Your homework submission should include this site name so that the graders may see the site.

(d) In the dropdown list for *Database server*, select "Create new server." Pick a username and password. Write it down. You need not turn this in.

(e) Click OK, which will take you to the previous screen. Then click OK again.

(f) Click *Create* to create the web site on Azure.

(g) This should return you to the previous screen. Click "Publish."

3. Once deployed, click on the file `Reviews.aspx` and check the markup (in Source View) to see what happens when the "Refresh" button is clicked. Notice that the `OnClick` points to a particular method. Right-click anywhere in method name and select *View Code*. Set a breakpoint at the first statement in the method.

4. We are going to remotely debug a web application. To do this, go to Server Explorer on the left, click on the *Windows Azure* expander, then click on the *Web Sites* expander. Find the name of the Azure web site. This is the name you provided to Azure when you deployed the web site. That is, `<name-you-provided>.azurewebsites.net`. Right-click on the name and select "Attach Debugger."'

5. When the web browser window opens, repeat the same sequence of steps you took above to cause the application to crash. This time, the debugger will stop on the breakpoint.

6. Identify which method is causing the issue. It appears to be hanging. Why? (Hint: look up the method using Google).

7. Identify and fix the underlying issue so that the method identified above completes.

8. Why did the page load successfully the first time, but not the second?

9. The deliverable: the answers to the questions above. (Leave the solution open for the following questions.)

Task 3:

This time we're going to debug the same application locally.

1. In Solution Explorer, right-click on the *BuggyBits* project and select *Set as StartUp Project*.

2. Click "Internet Explorer" (or your currently selected browser) on the Visual Studio menu bar to launch the application.

3. Once the application is running, click on the "New account" link.

4. Create a new account by filling in the boxes and click OK. If you encounter any errors, fix them before continuing.

5. Create another new account and pick the *same username* as the first time. Intuitively, you should have received an error message complaining that the account already exists, but you don't. Determine why.

6. The deliverable: provide a detailed description of the problem. (No need to fix it. Leave the solution open for the following questions.)

Task 4:

Now let's investigate a problem with an ASP.NET Label server control.

1. In the solution, find `CreateAccount.aspx` and open the expander to expose the code-behind file, `CreateAccount.aspx.cs`. This file contains the events relating to the *CreateAccount* page. In fact, you were previously working with this file in the previous question.

2. Find the `Page_Render` method. You'll notice that the statement appearing inside appears to be having no effect on the page.

3. First, what do you think the programmer *intended* to happen? Why isn't it happening? Explain. (Hint: make sure you understand the page lifecycle).

4. Fix the problem. Explain what you did.

5. Note that this problem revolves around an ASP.NET server control. Suggest a far more time and memory efficient means of achieving the same effect.

6. The deliverable: the explanations required above. (Leave the solution open for the following questions.)

Task 5:

1. Set a breakpoint on `protected void Page_Load` and run the application.

2. When the breakpoint is reached, find the window that allows you to inspect the variables and click the "Watch 1" tab near the bottom of the screen. We're going to inspect the *Request* object, which you'll recall is the object that encapsulates the details of the HTTP request. Type "Request" (no quotes) and open the expander so you can see all of the fields.

3. Answer these questions: write the the field and the values contained within for the following information:

    - the HTTP method used to request the web page
    - the HTTP headers

- the query string
- the MIME types accepted by the browser.
- the referrer
- the user agent?
- the cookies sent by the browser?

4. The deliverable: the answers required above. (Leave the solution open for the questions on the next page.)