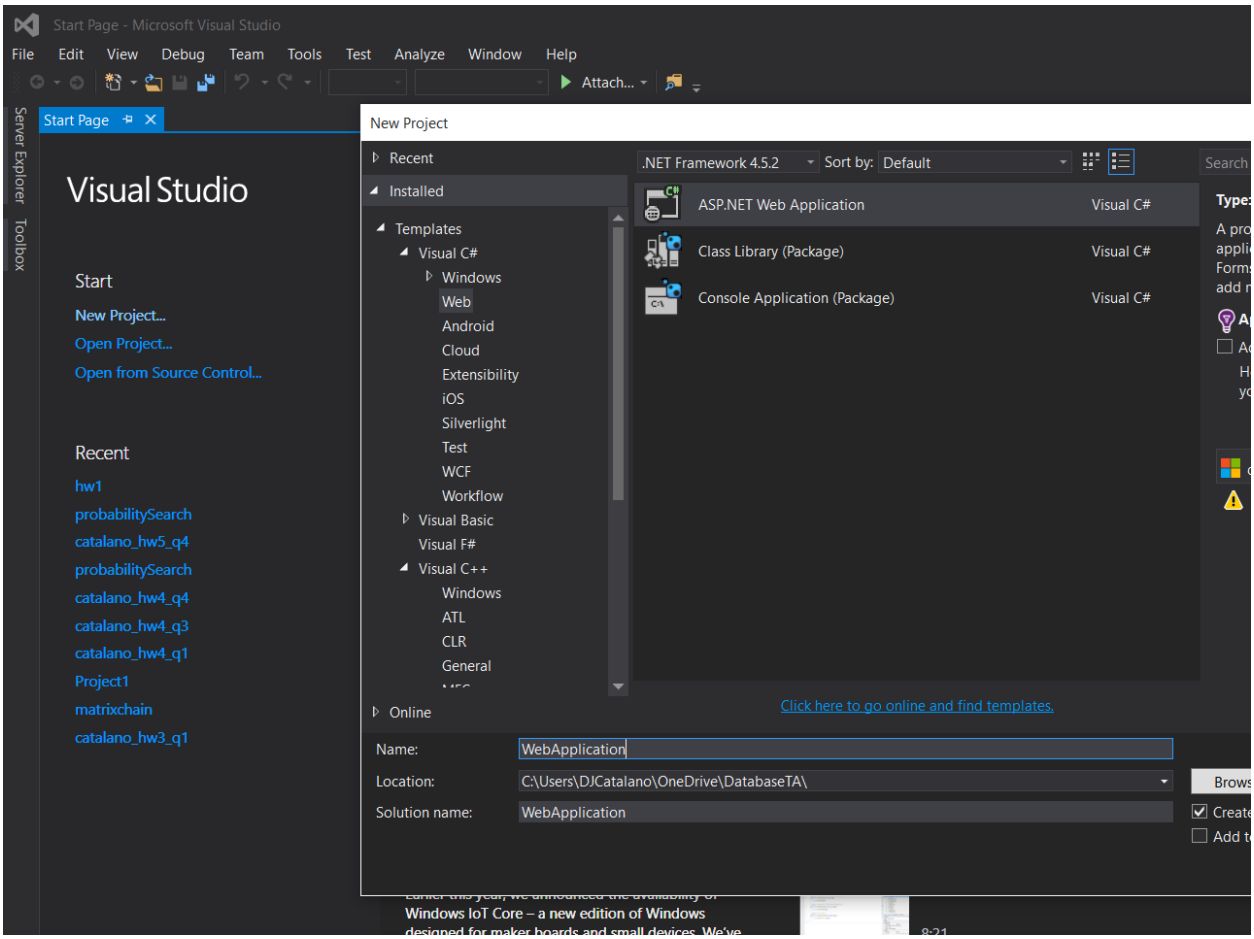
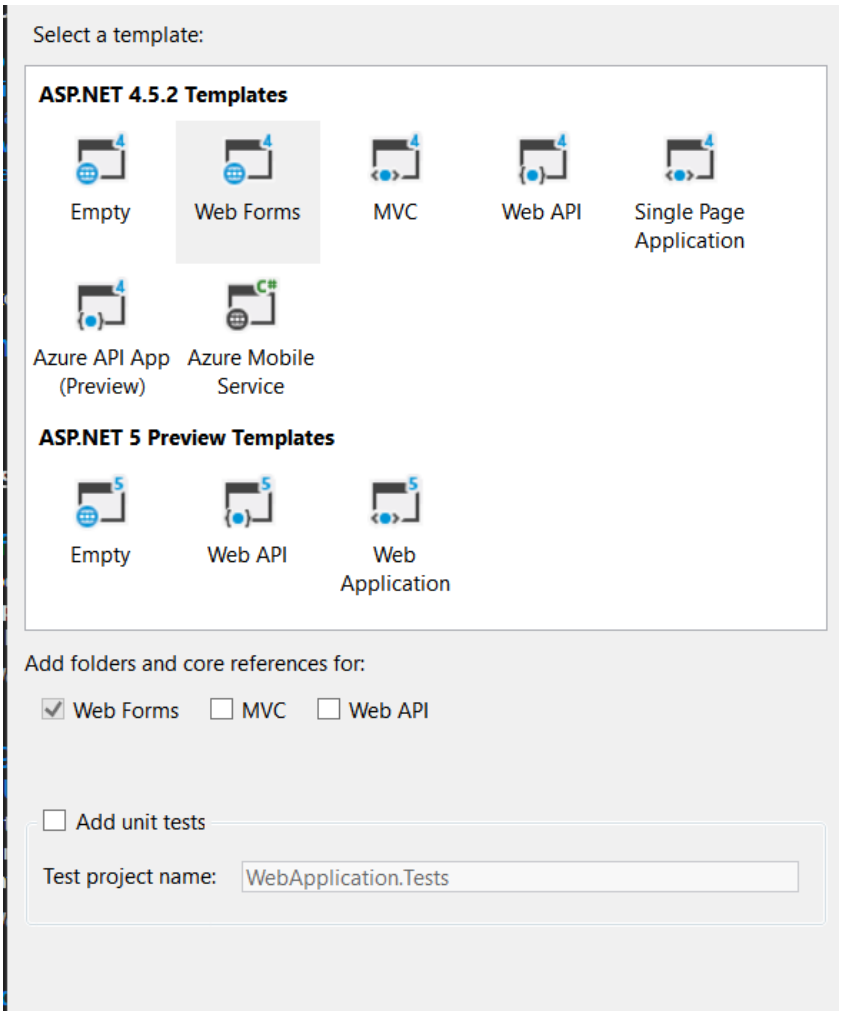
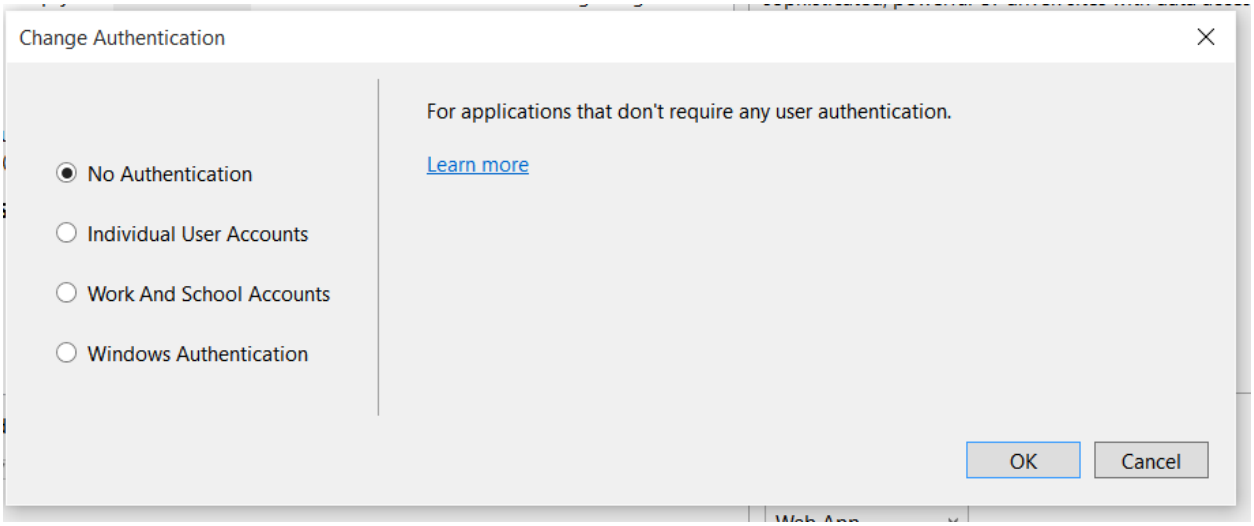
Once VS2015 is configured to build C# projects, create a new Visual C#, ASP.NET Web application



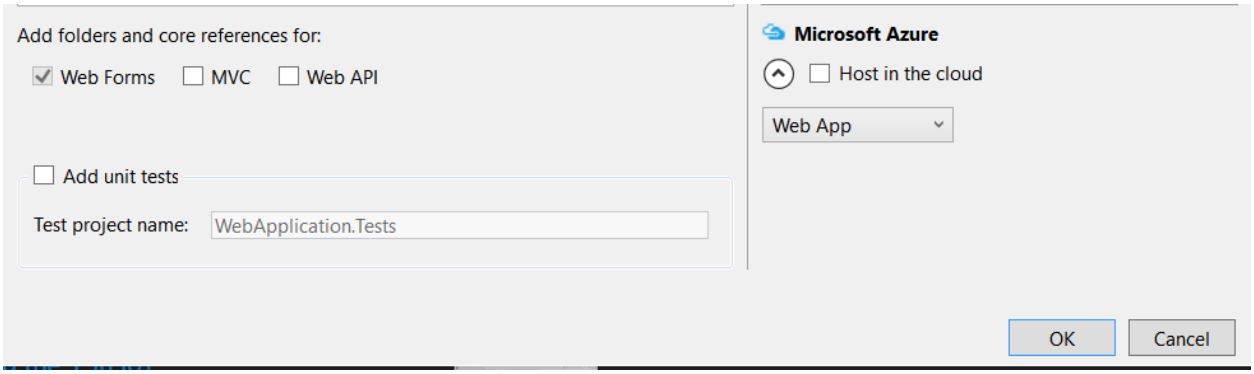
Select Web Forms



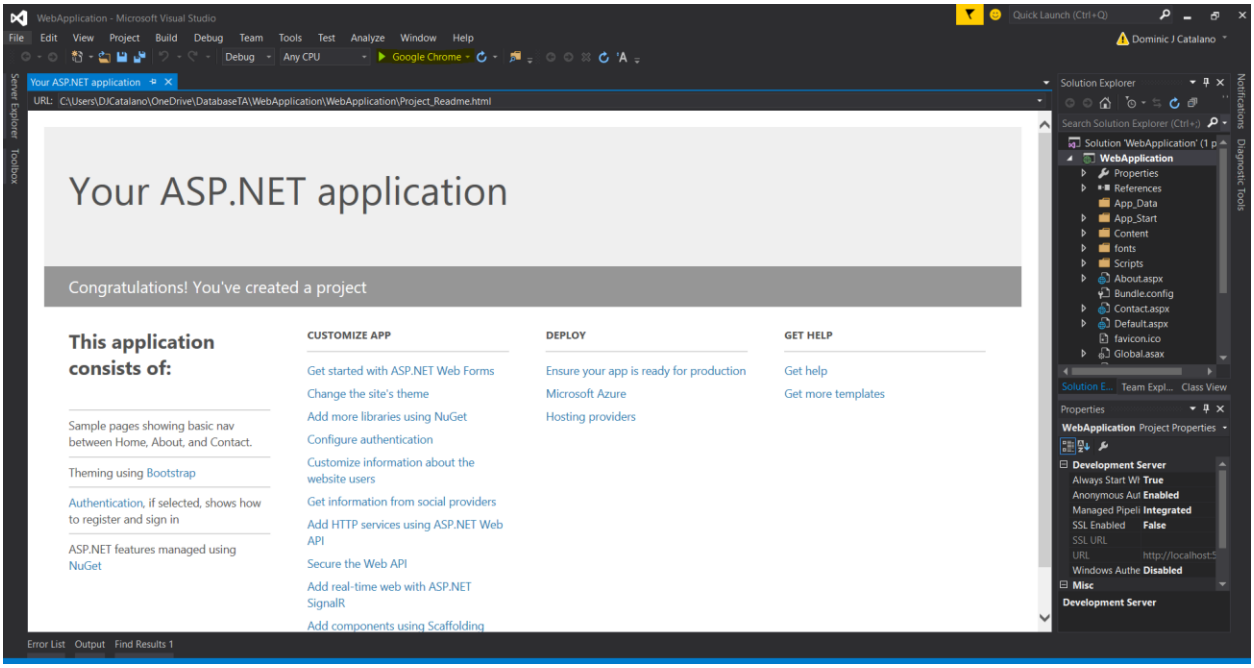
Update the authentication method to No Authentication (if you want, but highly recommended)



Deselect the option to host the application in Azure (if you have an account then by all means host it for real, but for the purposes of this class, everything can be local)

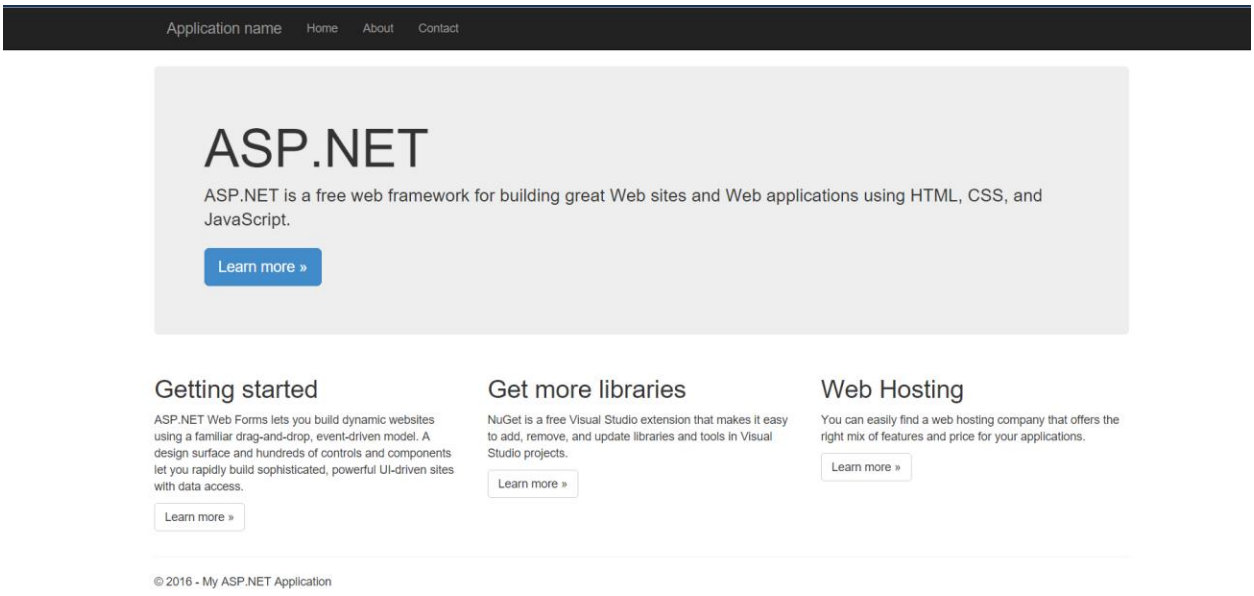


VS will now create a framework for your project. Feel free to browse around what VS created for you by running the application:

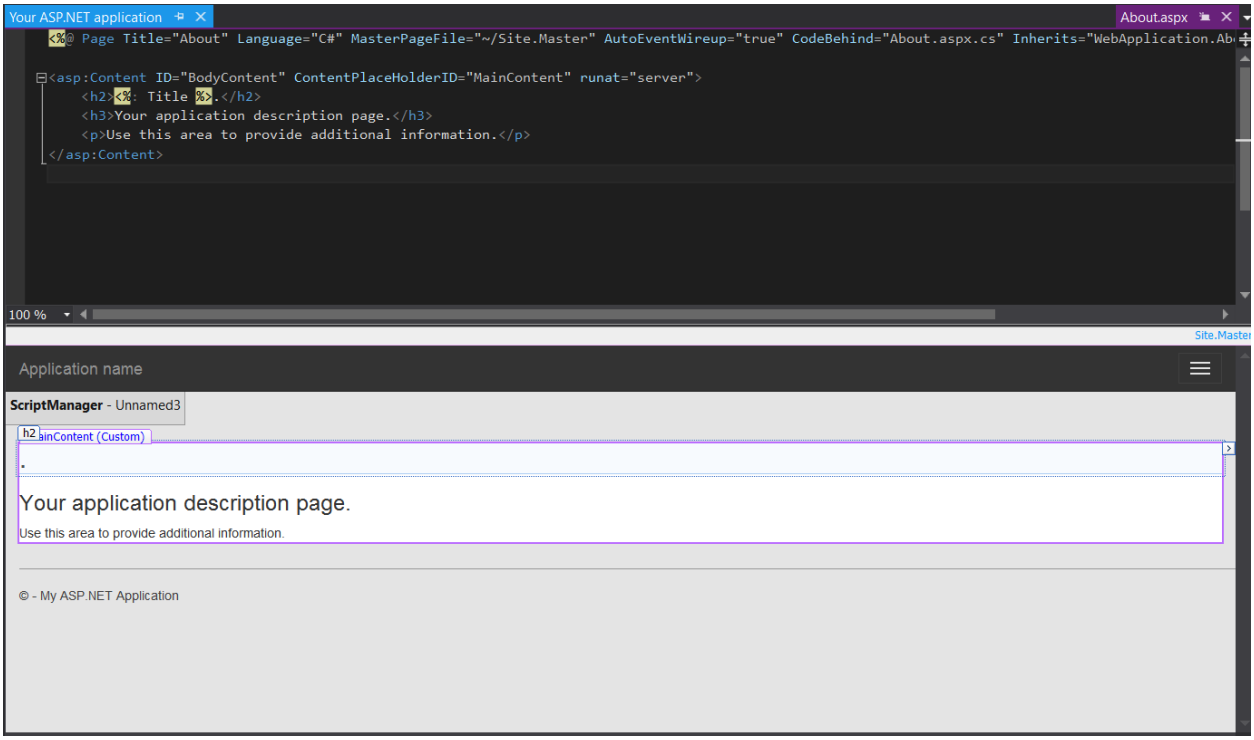


Microsoft does us a large favor in that it builds a cross platform (browser and mobile friendly) application with a built in home page, about page, and Contact page. All of these can be changed and altered, but for the purposes of this project, you can just remove them or use them to add your

information. It also builds out a base CSS package that, while very Microsoft, still looks professional and nice (read, good enough for the purposes of your class project).

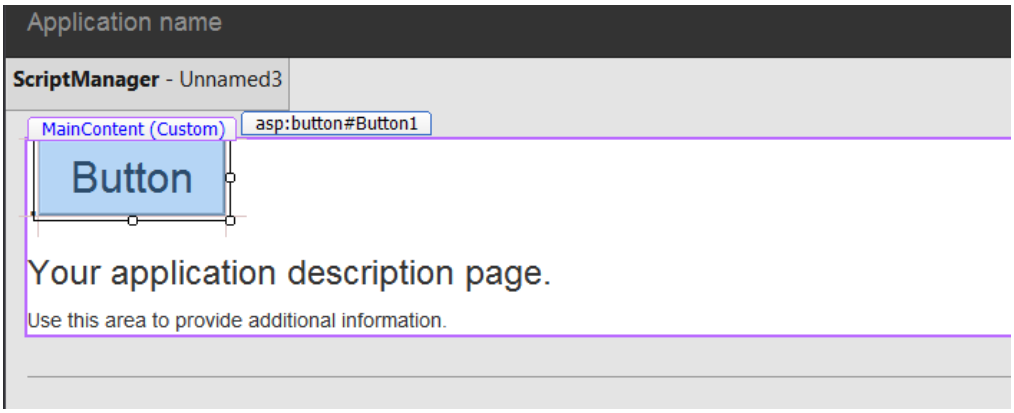


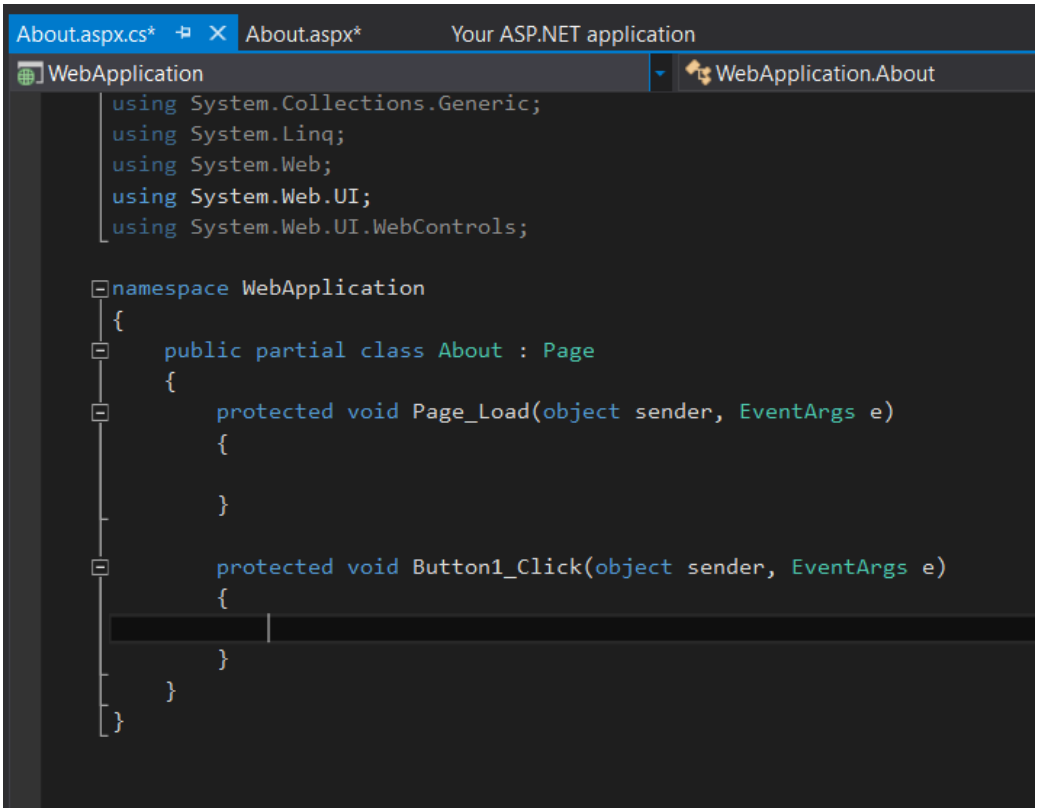
VS offers a drag and drop style of webpage building for aspx type pages as well as the HTML style view.



Every aspx page as CodeBehind file (usually pagename.aspx.cs) in which you can implement cs functionality to interact with the page. To implement interactions (i.e. buttons), simply drag a button

into the working space and double click. VS will build out the methods necessary and direct you to the new event handler.

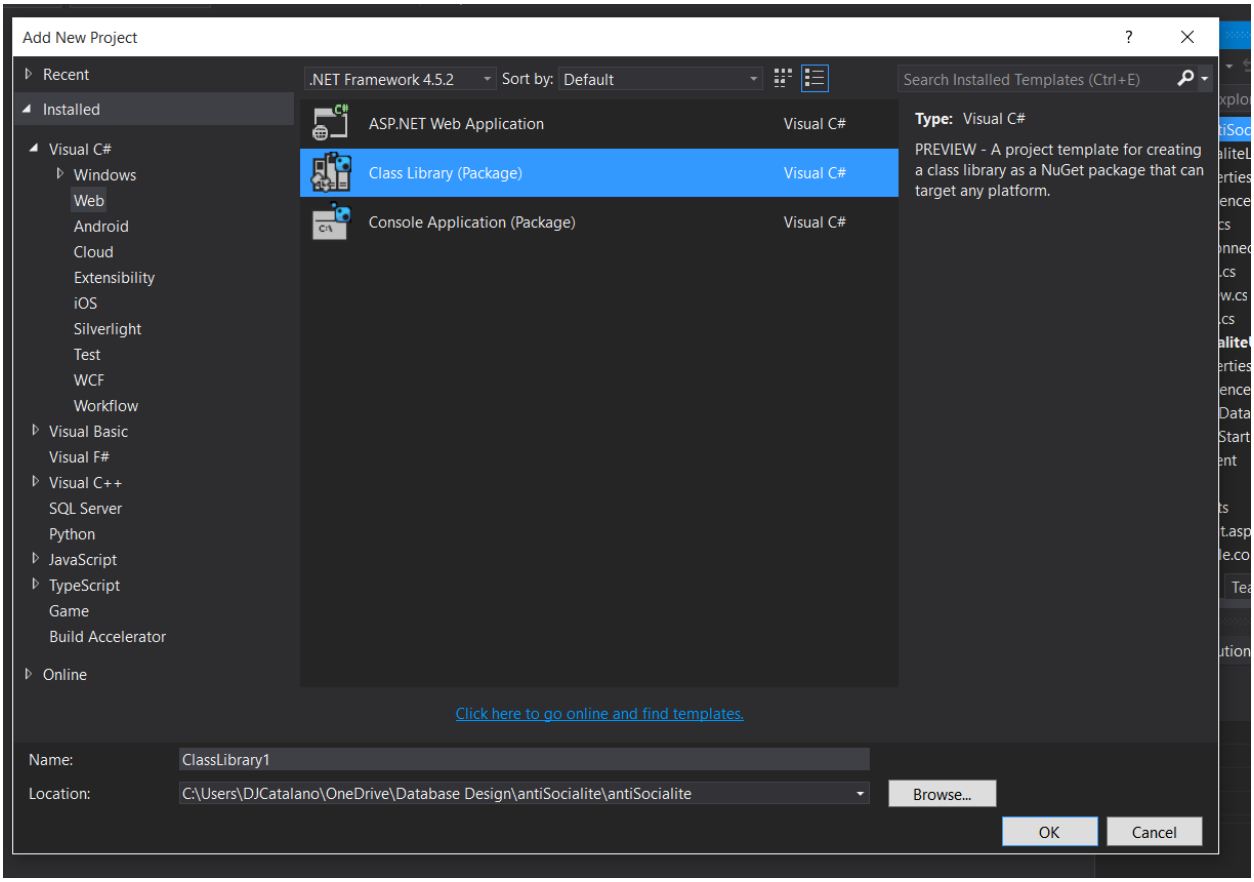




Aspx has a lot of really neat features to implement a fully modern and functional website.

C# is very much like Java in its syntax, capabilities, and methods. For the rest of this rough overview, I will be using the project I created for this class.

In order to achieve a good software architecture, it is important to build a class library outside of the UI implementation. To add a library, right click on the solution name, add, new item. Then select Class Library (naming it SolutionNameLib or whatever naming convention you'd like).



The first class I'd recommend making is a DBConnect Class that can handle the process to connect and disconnect to your database. This will simply your code in other places for a cleaner implementation. When building your classes, think of three main categories: What does your class contain (data relating to the object itself)? Is there related data involved? And What do I want my class to be able to do?

The data in your class should have a direct correlation to the database tables (id, name, etc). These should be private members of your class and accessible through setters and getters

```
private string _name = string.Empty;
public string Name
{
    get { return this._name; }
    set { this._name = value; }
}
```

Fancier implementations will have flags in a base class to represent whether or not the object has been changed or not (I did not do this because I am lazy). The benefit of this being that if the object has not changed since being returned from the database, there is no reason to waste time saving it.

Related objects can be considered to be lists or other objects related to the original, but not necessarily the original object (i.e. A person can write books. Person would be the original, and books would be a list inside of the object). The List object in C# is very flexible and works as an array or vector.

The final section refers primarily to the class's interactions with the database. At a basic level, every class should be able to be Saved, Loaded, and Deleted. I Highly Highly Highly recommend using stored procedures for all of your interactions with the database. They offer much higher levels of security, flexibility, and power when it comes to your application. Let's look at a Save functionality for one of my classes

Once you are ready to use your classes in your UI, simply add using <LibraryName>; to the top of the aspx.cs pages.

```
public void Save()
{
    DBConnect conn = new DBConnect();

    MySqlCommand cmd = new MySqlCommand("saveReview", conn.Connection);
    cmd.CommandType = CommandType.StoredProcedure;

    cmd.Parameters.AddWithValue(CritKeys.Id, _id);
    cmd.Parameters.AddWithValue(CritKeys.DrinkId, _drinkId);
    cmd.Parameters.AddWithValue(CritKeys.Rating, _rating);
    cmd.Parameters.AddWithValue(CritKeys.Review, _review);

    if (conn.OpenConnection())
    {
        _id = (Int32)cmd.ExecuteScalar();
    }

    conn.CloseConnection();
}
```

We start off by creating a new connection. Since we put the implantation for this in the DBConnect class, we don't have to worry about that. Next we create a new command using our stored proc "saveReview" and the connection. We set the command type to stored procedure and add the parameters of the procedure. I store parameter names inside the class itself as a subclass CritKeys so I know the names exist in one and only one place per class.

The power of stored procedures allows the database to handle creating a new object or saving over an existing one through the use of the id. Since my constructor sets the default id to 0, the stored proc knows to create a table entry and return the new id when we execute the stored procedure, otherwise update.

Finally we close the connection while maintaining the object. If the save creates a new entry, the id will be updated, if not the same id will be returned. This could also be used for error checking, but it's not implemented here.

In order to simply code in this area, I'd also recommend creating a hydrate method that can be called to build out objects when selecting data from the database. So the code in the load method looks like this

```
        if(conn.OpenConnection())
        {
            MySqlDataReader data = cmd.ExecuteReader();

            while (data.Read())
            {
                list.Add(Hydrate(data));
            }

        }
```

Which is very clean and simple. The hydrate method takes the datarow from the database and parses the row into an object for return.

```
    public static Drink Hydrate(MySqlDataReader row)
    {
        Drink drink = new Drink();
        drink.Id = Convert.ToInt32(row["id"]);
        drink.Name = row["name"].ToString();
        drink.Description = row["description"].ToString();
        drink.Type = row["type"].ToString();
        drink.Instruction = row["instruction"].ToString();
        drink.Reviews = Convert.ToInt32(row["reviews"]);
        drink.Rating = Convert.ToDecimal(row["rating"]);

        return drink;
    }
```

This parsing can be done via column name or index, but I find name to be more resilient and less dependent on the programmer.