INFO3067 Week 2 Class 2

Review

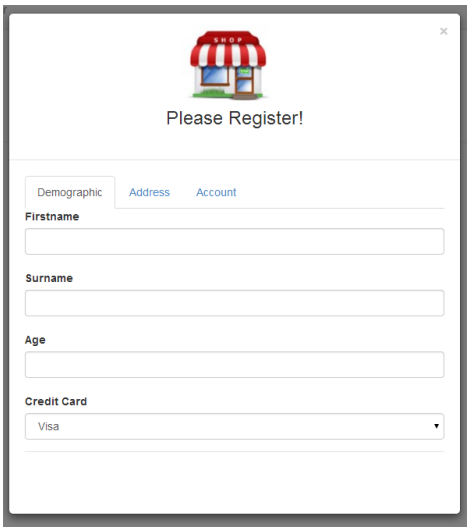
- Html Helpers
- Bootstrap Tab Control

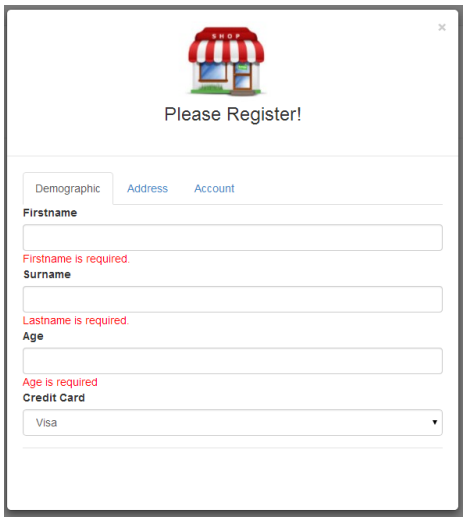
Case Study – Register

We're ready to complete the Register process on our case study site. The register modal is to have similar fields that we did in the last exercise. Note, we have to change the model the view uses, so instead of pointing to our SampleCustomerModel we'll point the page to the **CustomerViewModel** in the eStoreViewModels project. Change the first line of the Index.cshtml page like so:

```
@model eStoreViewModels.CustomerViewModel
```

The Tab contents of the register popup will also have to be fleshed out a little more. We'll have 3 tabs that have the following fields:





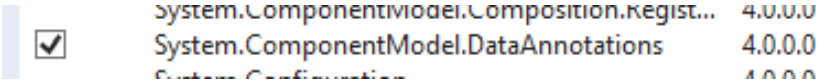
- Firstname is required
- Surname is required
- Age is required, must be a number, must be between 18-99
- To prevent going to a new tab, remember you'll need to create a javascript file (we called this exercisesJS.js in the MVCExercises project) and then point to this file from the _Layout.cshtml view

We haven't looked at how to use an HTML helper to code a drop down, so you can use the following markup to code the credit card type:

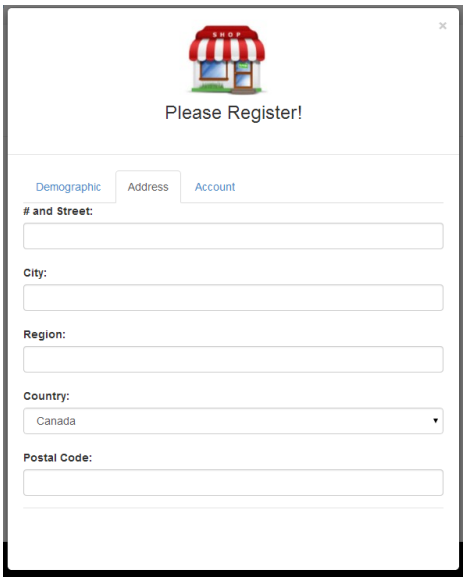
```
<br />
@Html.Label("Credit Card")
@{
    List<SelectListItem> ccs = new List<SelectListItem>();
    ccs.Add(new SelectListItem { Text = "Visa", Value = "V" });
    ccs.Add(new SelectListItem { Text = "Mastercard", Value = "M" });
    ccs.Add(new SelectListItem { Text = "American Express", Value = "A" });
}
@Html.DropDownListFor(model => model.CreditcardType, ccs)
```

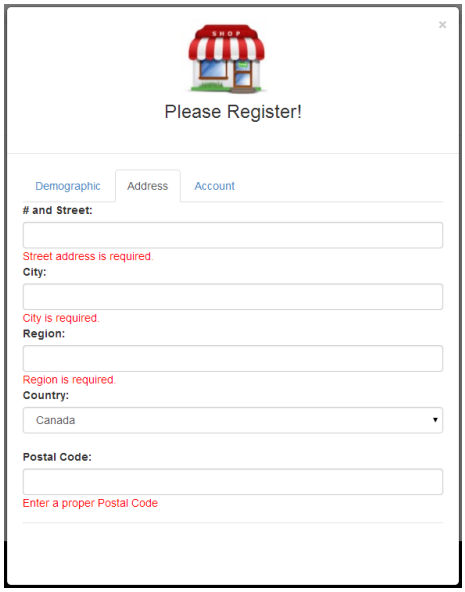
Remember to put the correct attributes on the ViewModel properties to correctly handle the validation, we'll use the following validation attributes for this case:

- [Required(ErrorMessage = "This field is required.")]
- [CompareAttribute("Password", ErrorMessage = "Passwords don't match.")]
- [Range(18, 99)]
- [RegularExpression(@"^([0-9a-zA-Z]([-.\w]*[0-9a-zA-Z])*@([0-9a-zA-Z][-\w]*[0-9a-zA-Z]\.)+[a-zA-Z]{2,9})$", ErrorMessage="Email format invalid")]
- Because your ViewModels are in a separate .dll project now you need to reference the DataAnnotations library:



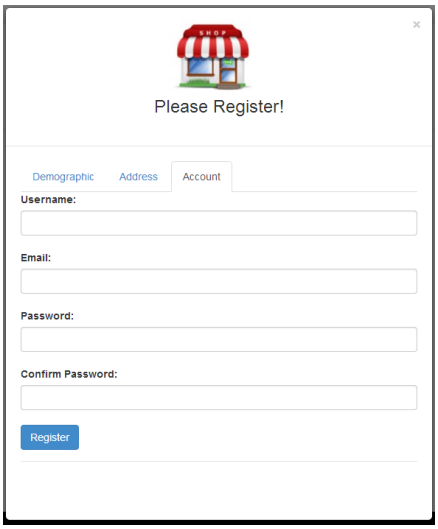
- The next tab will look something like this:

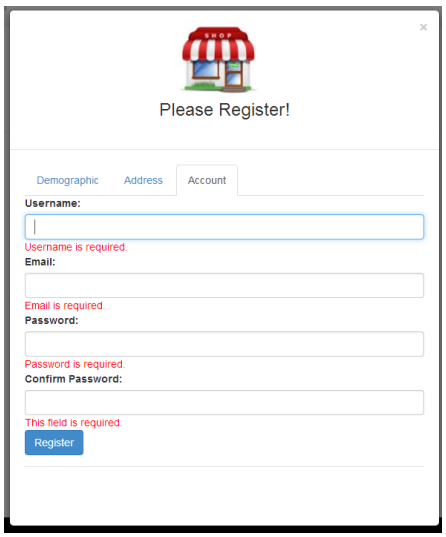




- All fields are required
- You can use the following regular expression for the postal code:
    - [RegularExpression(@"[a-zA-z]\d[a-zA-Z]-\d[a-zA-Z]\d")]

- The 3rd tab would contain

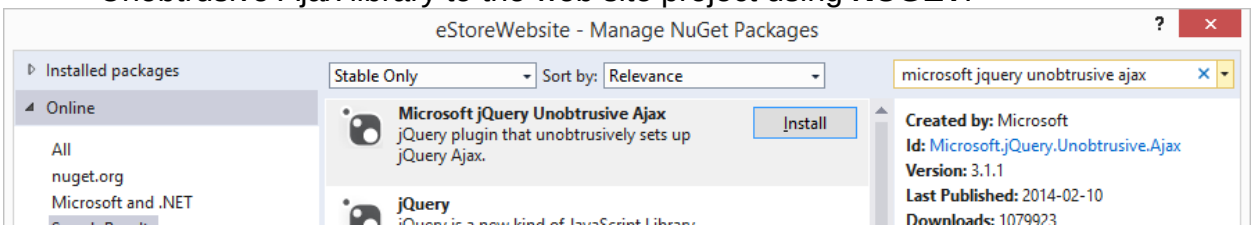




- All fields required
- Email needs regular expression from above
- Password and Confirm password should be masked and identical
  - To mask the password do not use the HTML helper @Html.TextBoxFor, but rather use @Html.PasswordFor:

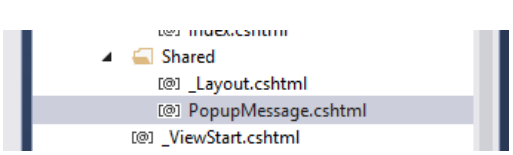
A couple of final things

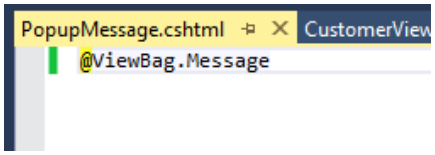
- Like we did in the exercises last class, we need to include the Microsoft jQuery Unobtrusive Ajax library to the web site project using **NUGET**:



- Then add `<script src="~/Scripts/jquery.unobtrusive-ajax.js"></script>` after the jquery entry in the _Layout.cshtml file

- Also we need to add a shared partial view called PopupMessage





Using ASPNet.Identity

Before we test the register button to execute the CustomerViewModel.Register method we will look at what Microsoft calls **ASP.Net Identity**. By installing the default MVC setup we have put in place most of the plumbing required to utilize it. We're going to employ some of their stuff into our setup but still use our own Customer table. It saves us some additional coding to check things like determining if there is a user already registered with that name.

- Add the following usings to facilitate the Register method on the Home controller:

```
// added for Register
using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;
using Microsoft.Owin.Security;
using eStoreViewModels;
using eStoreWebsite.Models;
```

- Download from FOL the code called **Helpers** and place it at the end of the class. This code was originally located in the AccountsController that was installed with the original code

```
        Helpers
    }

}
```

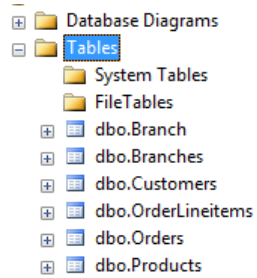
-
- Code the following Register Method:

```
// POST:/Home/Register
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<ActionResult> Register(CustomerViewModel model, string returnUrl)
{
    if (ModelState.IsValid)
    {
        try
        {
            model.Register();
            if (model.CustomerID > 0)
            {
                var user = new ApplicationUser() { UserName = model.Username };
                var result = await UserManager.CreateAsync(user, model.Password);
                if (result.Succeeded)
                {
                    ViewBag.Message = model.Message + ". Please proceed to login";
                }
                else
                {
                    int rowsDelete = model.Delete(); // something went wrong with ASPNet.Identity get rid of our customer
                    ViewBag.Message = "Problem Registering, " + result.Errors.ElementAt(0) + " try again!";
                }
            }
            else
            {
                ViewBag.Message = "Problem Registering, try again later";
            }
        }
        catch (Exception ex)
        {
            ViewBag.Message = "Problem Registering, " + ex.Message + " try again later";
        }
    }
    return PartialView("PopupMessage");
}
```

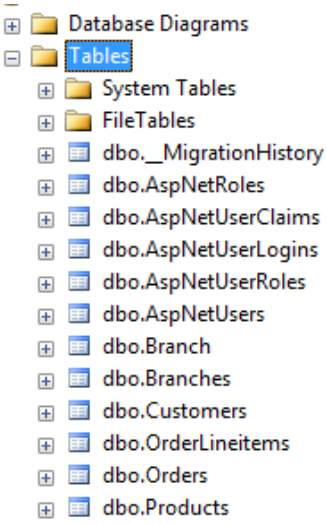
- You'll need to change your View's Ajax form now to use POST (was GET)

```
@using (
Ajax.BeginForm("Register", "Home", new AjaxOptions
{
    InsertionMode = InsertionMode.Replace,
    HttpMethod = "POST",
    LoadingElementId = "ajaxSplashReg",
```

- If we look at our current tables in Mgmt studio they should look like this:



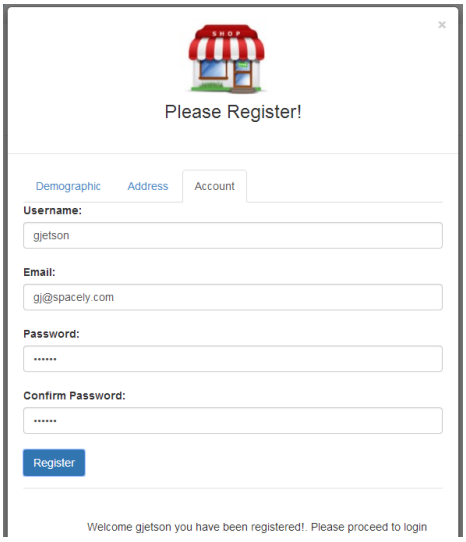
- After trying the register process the ASPNet Identity process will create a number of additional tables for you, so the schema should look like:

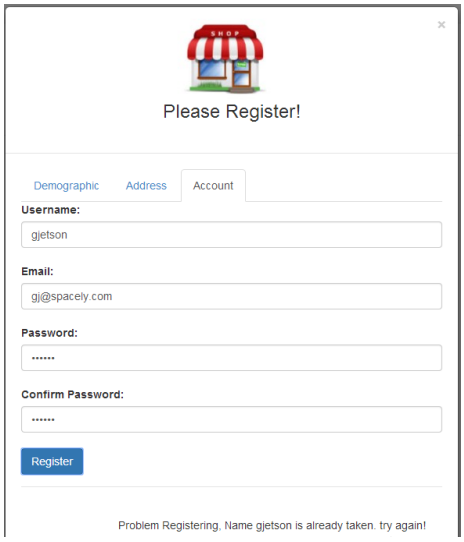


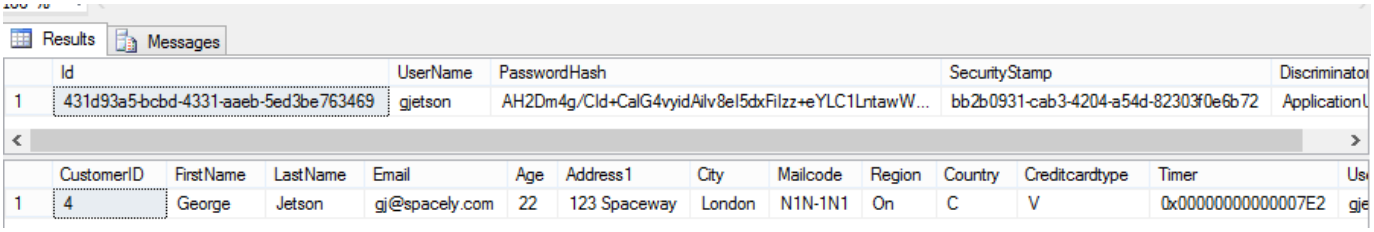
- To get these tables created we need to edit **Web.config** file. The identity process is looking for a connection called **DefaultConnection** we need to change the current connection string to use your database or else the system will create its own. We also need to add a connection string for the db model we are currently using. So copy the eStoreDBEntities connection string from **App.config** in the Models project **immediately after** the new DefaultConnection string:

```
  <add name="DefaultConnection" connectionString="Data Source=localhost;Initial
Catalog=eStoreDB;Integrated Security=True" providerName="System.Data.SqlClient" />
  <add name="eStoreDBEntities"
connectionString="metadata=res://*/eStoreDBModel.csdl|res://*/eStoreDBModel.ssdl|res://*/eStoreDBModel.msl
;provider=System.Data.SqlClient;provider connection string=&quot;data source=.;initial
catalog=eStoreDB;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;"
providerName="System.Data.EntityClient" />
```

- Test out the page now by filling in the fields and having the Ajax form point to the Home Controller's Register method. It may take longer than normal the first time as the VS will be creating your new membership tables (one time only). Then try **hitting the register button again**, if the membership routine is doing its job you should see the User name already exists message:







Results | Messages

	Id	UserName	PasswordHash	SecurityStamp	Discriminator
1	431d93a5-bcbd-4331-aaeb-5ed3be763469	gjetson	AH2Dm4g/Cld+CalG4vyidAilv8el5dxFilzz+eYLC1LntawW...	bb2b0931-cab3-4204-a54d-82303f0e6b72	ApplicationU

	CustomerID	FirstName	LastName	Email	Age	Address1	City	Mailcode	Region	Country	Creditcardtype	Timer	Us
1	4	George	Jetson	gj@spacely.com	22	123 Spaceway	London	N1N-1N1	On	C	V	0x00000000000007E2	gje

LAB 4

- Complete the Register process
- Submit 4 screen shots in one Word doc
    1. Register worked
    2. Register didn't work with Username already existing
    3. Inserted row in our **Customer'**s table (make sure the Userid is the same as the 1st screen shot)
    4. Inserted row in Microsoft's **AspNetUsers** table (Again make sure the UserId field matches)

Summary of Key Terms

- Register Process
- AspNet Identity
    - AspNetUsers table