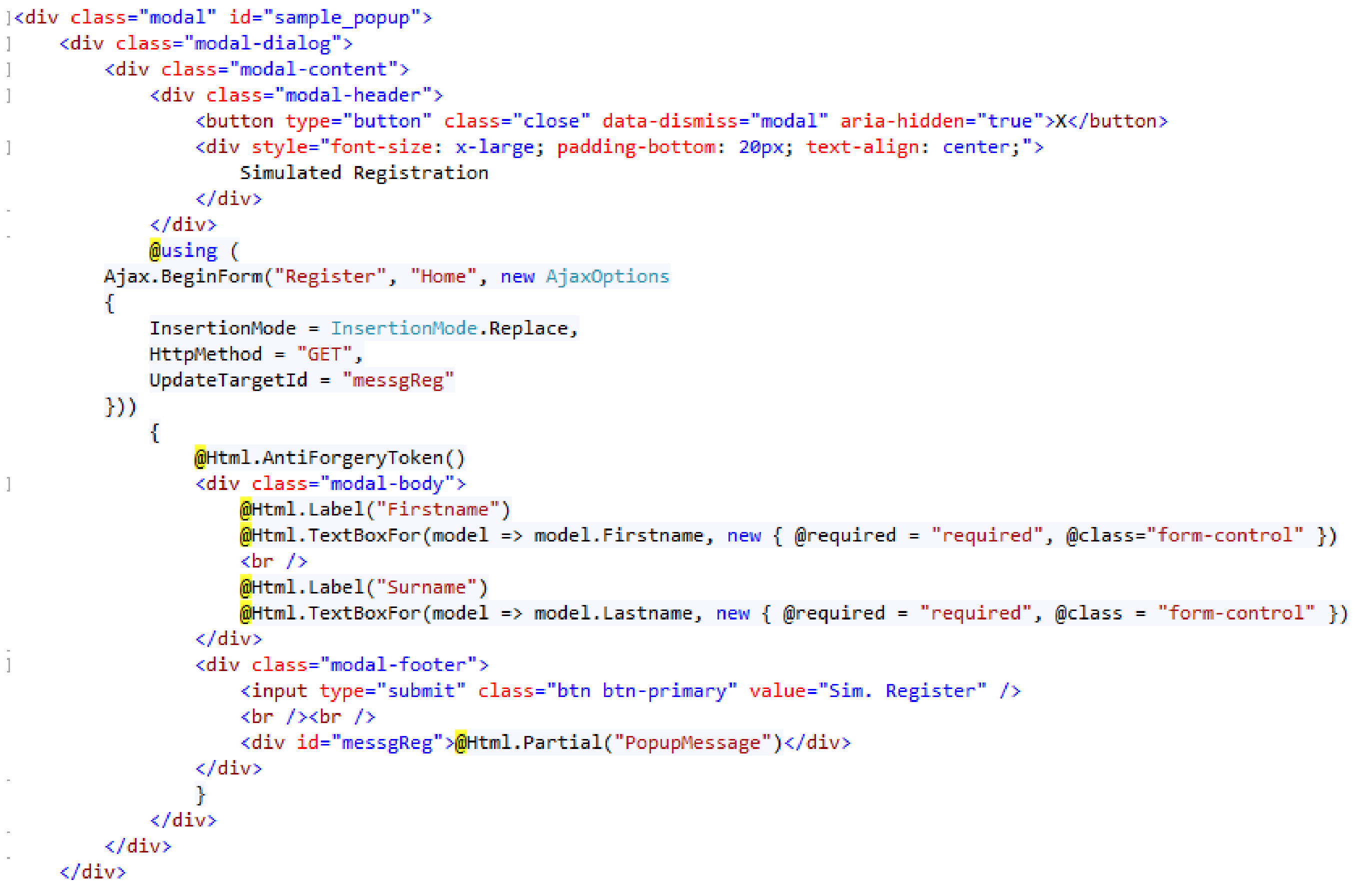
INFO3067 Week 2 Class 1

Review

- Bootstrap footer/modal
- Case1 framework

Exercise - Using HTML Helpers

You use HTML helpers in a view to render HTML content. An HTML helper, in most cases, is just a method that returns a string. You can build an entire ASP.NET MVC application without using a single HTML helper. However, HTML helpers make your life as a developer easier. By taking advantage of helpers, you can build your views with far less work. In addition an HTML helper can work directly with models in a view. So let's look at a simple example. Return to your **MVCExercises project**. Let's use some html helpers to build an Ajax enabled form (AJAX enables a Web application to retrieve data from the server asynchronously and to update parts of the existing page. This improves the user experience by making the Web application more responsive.). Copy the modal code from the Name2 page over to your Index.cshtml (Home view) page and change it to the following markup:



```
<div class="modal" id="sample_popup">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal" aria-hidden="true">X</button>
                <div style="font-size: x-large; padding-bottom: 20px; text-align: center;">
                    Simulated Registration
                </div>
            </div>
            @using (
        Ajax.BeginForm("Register", "Home", new AjaxOptions
        {
            InsertionMode = InsertionMode.Replace,
            HttpMethod = "GET",
            UpdateTargetId = "messgReg"
        }))
            {
                @Html.AntiForgeryToken()
                <div class="modal-body">
                    @Html.Label("Firstname")
                    @Html.TextBoxFor(model => model.Firstname, new { @required = "required", @class="form-control" })
                    <br />
                    @Html.Label("Surname")
                    @Html.TextBoxFor(model => model.Lastname, new { @required = "required", @class = "form-control" })
                </div>
                <div class="modal-footer">
                    <input type="submit" class="btn btn-primary" value="Sim. Register" />
                    <br /><br />
                    <div id="messgReg">@Html.Partial("PopupMessage")</div>
                </div>
            }
        </div>
    </div>
</div>
```

Any of the code that starts with **@Html**… is known as an HTML Helper, in this case we're defining some labels and textboxes. Also in the code we are using an Ajax enabled form starting with the code **Ajax.BeginForm**…The parameters for this call are as follows:

- "Register" – is the method that is called
- "Home" – is the controller that the method is housed in
- AjaxOptions – tell the form to clear the content, use the HTTP Get method, and update a form element called **msg** when the call is complete.
    - InsertionMode - replaces the inner html (or the content) not the outer html (which is <div>..</div>
    - The msg element is a div that simply displays the contents of PopupMessage partial view.

@Html.Partial is typically used in conjunction with a variable (in our case PopupMessage). To facilitate this line of code you will need to create a partial view in the Shared folder called **PopupMessage.cshtml**. It only contains 1 line of code to dump out the ViewBag.Message contents:

- @ViewBag.Message

Next add the following line of code to the top of the page:

- `@model MVCExercises.Models.SampleCustomerModel`

Add the following auto implemented properties to a class called **SampleCustomerModel** in the **Models** folder (note most of this code is already in in the case study's CustomerViewModel class):

```
// Auto-implemented properties
public int CustomerID { get; set; }
public string Username { get; set; }
public string Firstname { get; set; }
public string Lastname { get; set; }
public string Password { get; set; }
public string RepeatPassword { get; set; }
public int Age { get; set; }
public string Email { get; set; }
public string Address1 { get; set; }
public string City { get; set; }
public string Mailcode { get; set; }
public string Country { get; set; }
public string CreditcardType { get; set; }
public string Region { get; set; }
public string Message { get; set; }
```

And then a method called Register:

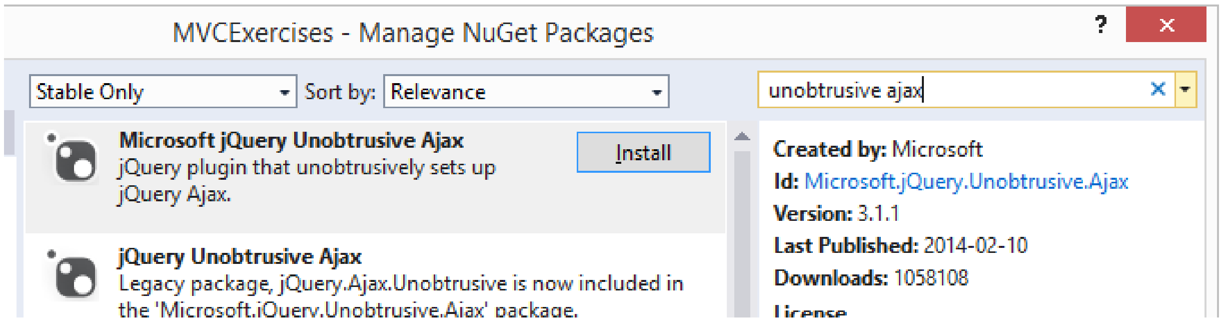
```
public void Register()
{
    Message = "Simulated registration worked for " + Firstname + ", " + Lastname;
    CustomerID = 99;
}
```

And finally update the HomeController to include a **Register** Method that accepts an instance of SampleCustomer as an input parameter:

```
public ActionResult Register(MVCExercises.Models.SampleCustomerModel cust)
{
    cust.Register();
    ViewBag.Message = cust.Message;
    return PartialView("PopupMessage");
}
```

What we have done here is added a Model for our form to use (in our case only two of the properties will be exercised) and populate. This is done through the linq call from the form (eg. **model => model.Lastname**). The Register method is just a simulation that we'll eventually flesh out to pass data over to a database.

Next we have to get the Ajax to work, so we need to add the JQuery Unobtrusive library to the mix using Nuget (version 3.1.1 at the time of this writing)



Microsoft jQuery Unobtrusive Ajax
jQuery plugin that unobtrusively sets up jQuery Ajax.

jQuery Unobtrusive Ajax
Legacy package, jQuery.Ajax.Unobtrusive is now included in the 'Microsoft.jQuery.Unobtrusive.Ajax' package.

Created by: Microsoft
Id: Microsoft.jQuery.Unobtrusive.Ajax
Version: 3.1.1
Last Published: 2014-02-10
Downloads: 1058108
License

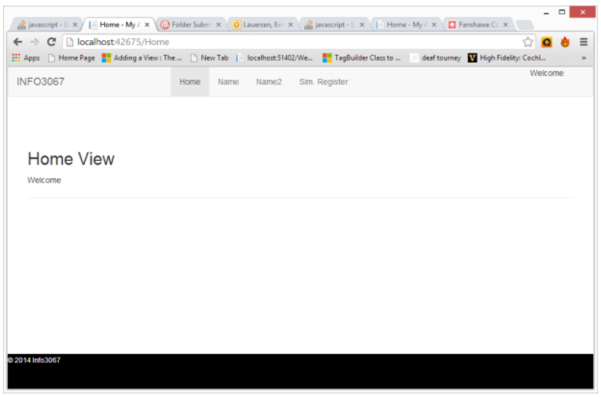
Then add the unobtrusive library to the Layout page:

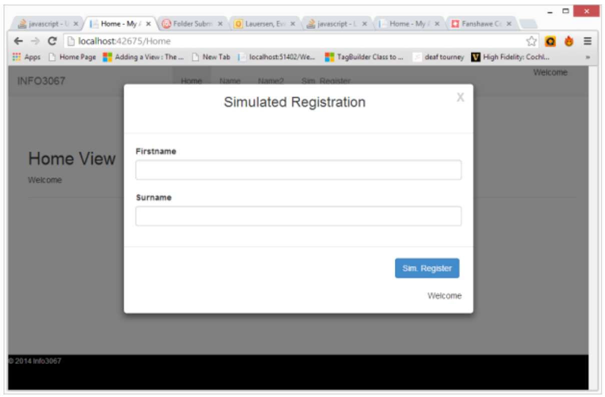
```
    </div>
    <script src="~/Scripts/jquery-1.10.2.min.js"></script>
    <script src="~/Scripts/jquery.unobtrusive-ajax.js"></script>
    <script src="~/Scripts/bootstrap.min.js"></script>
    <script src="~/Scripts/exerciseJS.js"></script>
</body>
</html>
```

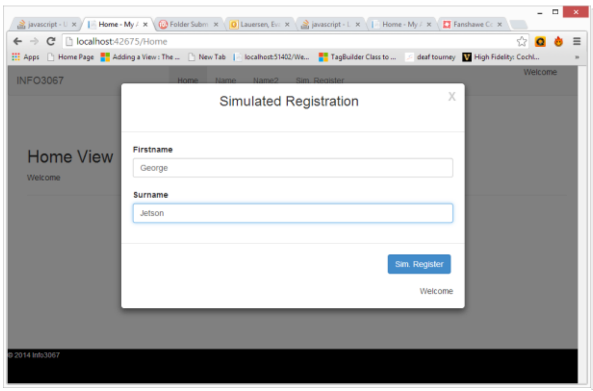
Next add a new line item in the main menu with an entry of **Sim. Register** and point it to our new popup:

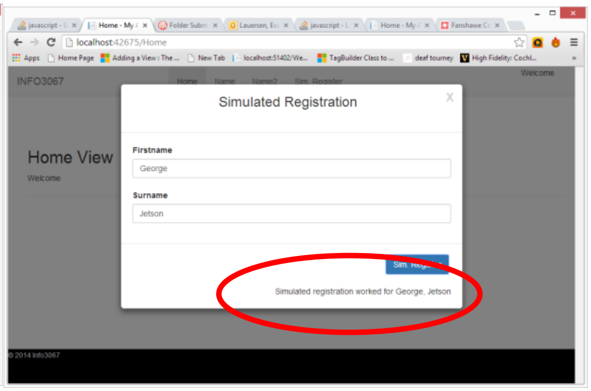
```
<li ><a href="/Home/Name">Name</a></li>
<li><a href="/Home/Name2#sample_popup" data-toggle="modal">Name2</a></li>
<li><a href="/Home#sample_popup" data-toggle="modal">Sim. Register</a></li>
```

Now you should be able to test this out, run the application and click on Sim. Register, fill in the fields and see if the message we set in the registration method appears:









Simulated registration worked for George, Jetson

Warning about using modals and partial views – the modals won't fire unless you're first on the page that contains the modal eg. To see the Sim Register modal you need to be on the Home page first. We'll add a workaround to the case study so that this won't be a problem for an unaware user.

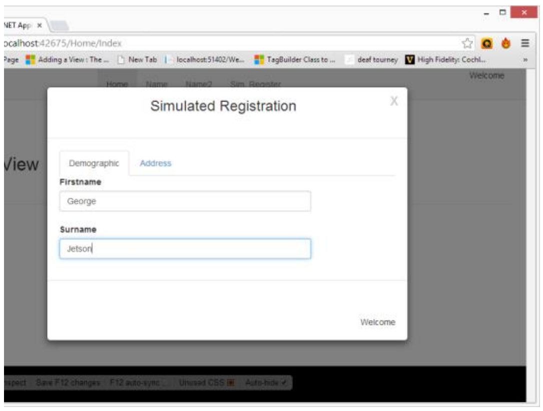
Exercise – Bootstrap Tabs

During our registration process we will be adding a number of fields and we'll break these up into various categories using the **Bootstrap Tab** control. To use the tabs we must do the following:

- Add some tabs markup to the popup area's <div class="**modal-body**"> of Index.cshtml. Replace the current div with a class=modal body with the following:

```
<div class="modal-body">
    <div class="tabbable" id="myTabs">
        <ul class="nav nav-tabs">
            <li class="active"><a href="#personal" data-toggle="tab">Demographic</a></li>
            <li><a href="#address" data-toggle="tab">Address</a></li>
        </ul>
        <div class="tab-content">
            <div class="tab-pane active" id="personal">
                @Html.Label("Firstname")
                @Html.TextBoxFor(model => model.Firstname, new { @class = "form-control", @style = "width:75%;" })
                <br />
                @Html.Label("Surname")
                @Html.TextBoxFor(model => model.Lastname, new { @class = "form-control", @style = "width:75%;" })
            </div>
            <div class="tab-pane" id="address">
                @Html.Label("# and Street:")
                @Html.TextBoxFor(model => model.Address1, new { @class = "form-control", @style = "width:75%;" })
                <br />
                @Html.Label("City:")
                @Html.TextBoxFor(model => model.City, new { @class = "form-control", @style = "width:75%;" })
                <br />
                <input type="submit" class="btn btn-primary pull-right" value="Sim. Register" />
                <br />
            </div>

        </div>
    </div>
</div>
```

Notice how the anchors map to the div id's. Next, test out the modal again and exercise both tabs:

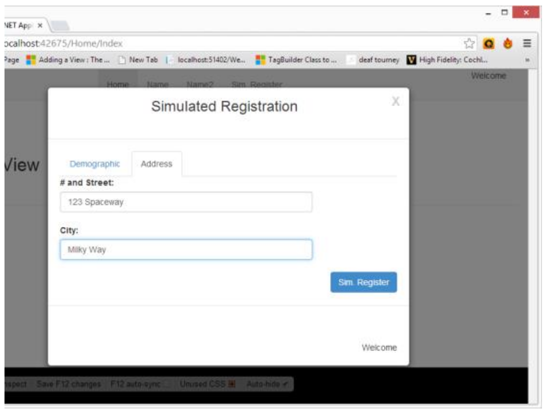

Simulated Registration

Demographic | Address

Firstname: George

Surname: Jetson

Welcome


Simulated Registration

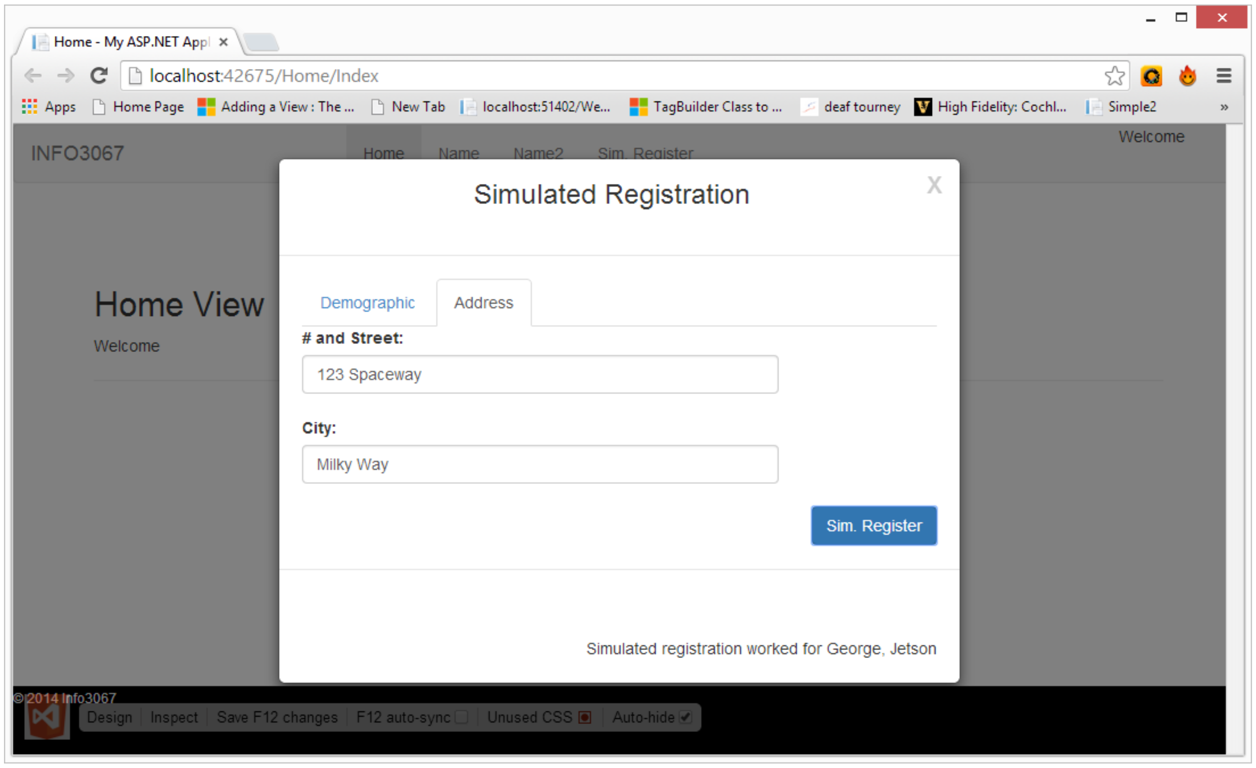
Demographic | Address

and Street: 123 Spaceway

City: Milky Way

Sim. Register

Welcome



and Street:

123 Spaceway

City:

Milky Way

Simulated registration worked for George, Jetson

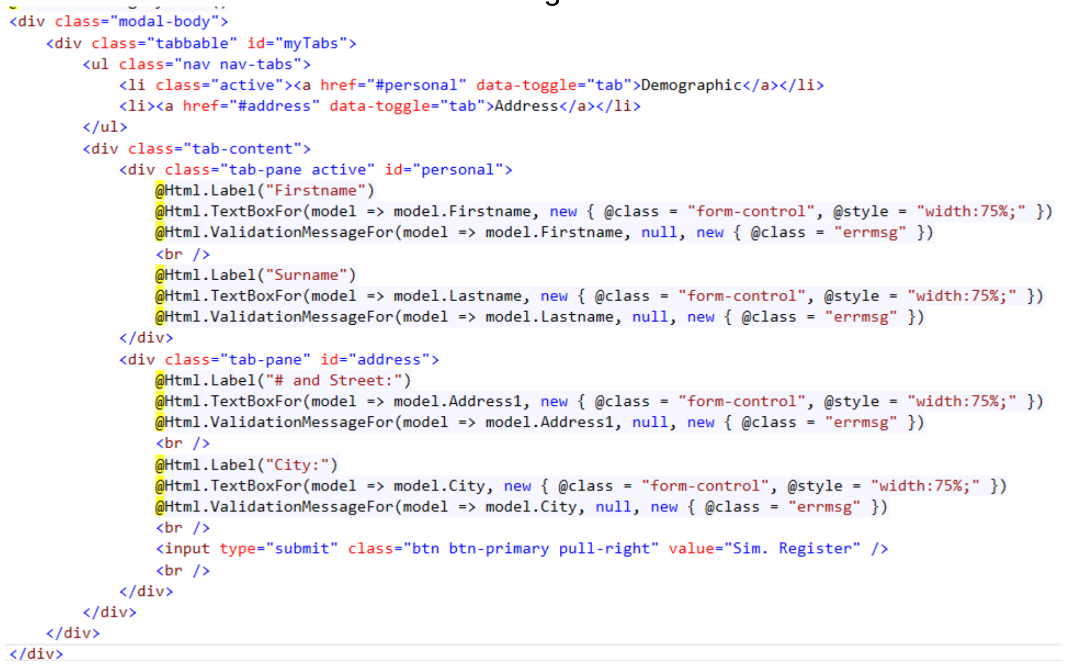
Exercise - Using Validation

The job of validation is to force the user into entering data that makes sense for the application and/or prevent invalid data from entering the system. We have a myriad of choices when doing validation:

- Client side:
  - HTML5 validation
  - Traditional Javascript validation
  - JQuery validators
- Server side:
  - Validation Attributes in code
  - Logic
  - Database constraints

What we'll end up using is a combination of some of these methodologies, for instance we are using both html5 validation and server side validation as seen in the code below. The tab presents a bit of a problem for us as we'll see shortly but we can overcome this with a little bit of additional JQuery coding.

Change the Modal body one more time with the following code (same as above, except each field now has a validation message:

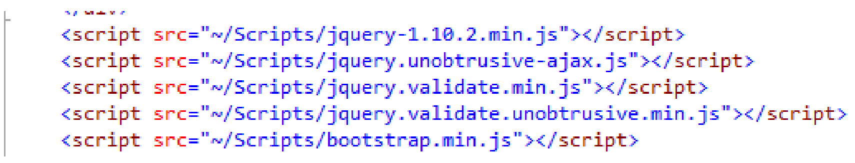


```
<div class="modal-body">
    <div class="tabbable" id="myTabs">
        <ul class="nav nav-tabs">
            <li class="active"><a href="#personal" data-toggle="tab">Demographic</a></li>
            <li><a href="#address" data-toggle="tab">Address</a></li>
        </ul>
        <div class="tab-content">
            <div class="tab-pane active" id="personal">
                @Html.Label("Firstname")
                @Html.TextBoxFor(model => model.Firstname, new { @class = "form-control", @style = "width:75%;" })
                @Html.ValidationMessageFor(model => model.Firstname, null, new { @class = "errmsg" })
                <br />
                @Html.Label("Surname")
                @Html.TextBoxFor(model => model.Lastname, new { @class = "form-control", @style = "width:75%;" })
                @Html.ValidationMessageFor(model => model.Lastname, null, new { @class = "errmsg" })
            </div>
            <div class="tab-pane" id="address">
                @Html.Label("# and Street:")
                @Html.TextBoxFor(model => model.Address1, new { @class = "form-control", @style = "width:75%;" })
                @Html.ValidationMessageFor(model => model.Address1, null, new { @class = "errmsg" })
                <br />
                @Html.Label("City:")
                @Html.TextBoxFor(model => model.City, new { @class = "form-control", @style = "width:75%;" })
                @Html.ValidationMessageFor(model => model.City, null, new { @class = "errmsg" })
                <br />
                <input type="submit" class="btn btn-primary pull-right" value="Sim. Register" />
                <br />
            </div>
        </div>
    </div>
</div>
```

Include some styling in the mvc1.css file for the **errmsg** class:

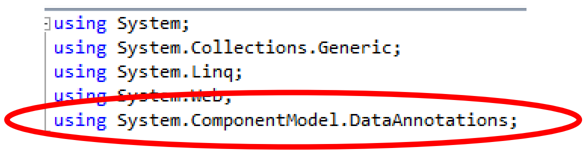
- `.errmsg {color:red;}`

We have all of the markup in place to handle validation but we need to add a couple of more jquery libraries to make it work, return to the _Layout.cshtml and change the scripts to include the following:



```
    <script src="~/Scripts/jquery-1.10.2.min.js"></script>
    <script src="~/Scripts/jquery.unobtrusive-ajax.js"></script>
    <script src="~/Scripts/jquery.validate.min.js"></script>
    <script src="~/Scripts/jquery.validate.unobtrusive.min.js"></script>
    <script src="~/Scripts/bootstrap.min.js"></script>
```

Then make the following changes to your SampleCustomerModel class:



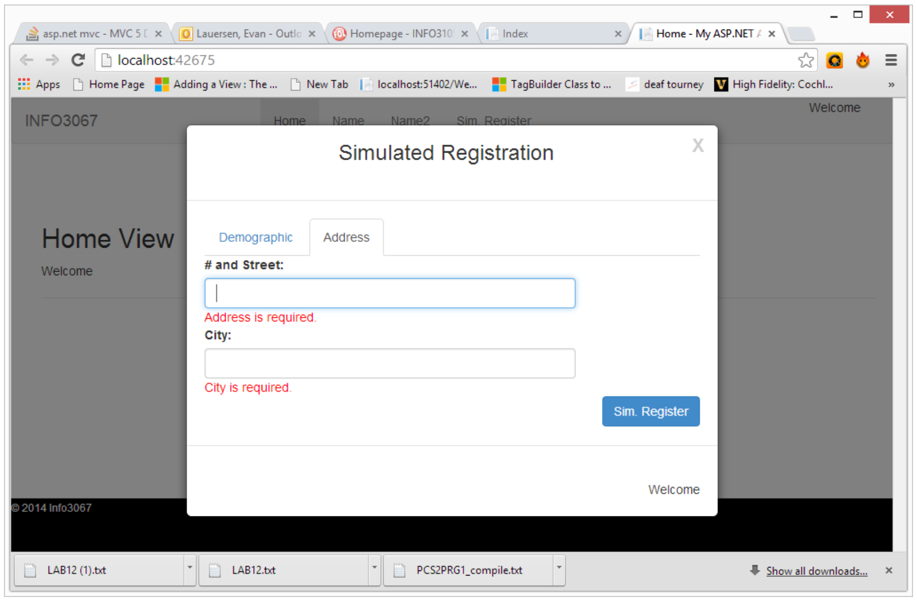
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;
```

Then add the following Required attribute on the Firstname, Lastname, Address1, and City properties

```
[Required(ErrorMessage = "Firstname is required.")]
1 reference
public string Firstname { get; set; }
[Required(ErrorMessage = "Lastname is required.")]
1 reference
public string Lastname { get; set; }
```

```
[Required(ErrorMessage = "Address is required.")]
0 references
public string Address1 { get; set; }
[Required(ErrorMessage = "City is required.")]
0 references
public string City { get; set; }
```

Fyi the ErrorMessage is used in the Html helper. Now try your popup once more and go to the 2nd tab and press the submit button:



Simulated Registration

Demographic | Address

and Street:

Address is required.

City:

City is required.

Sim. Register

Welcome

This is an example of server side validation. The process uses the attribute's error message as the source for the html helper. We have a problem though the validation is only good for the tab that the button is on. We need to come up with a way to force the user to stay on a particular tab until it has been entirely validated, and pass the server messages back to that tab, fortunately a bit of JQuery coding can do this.

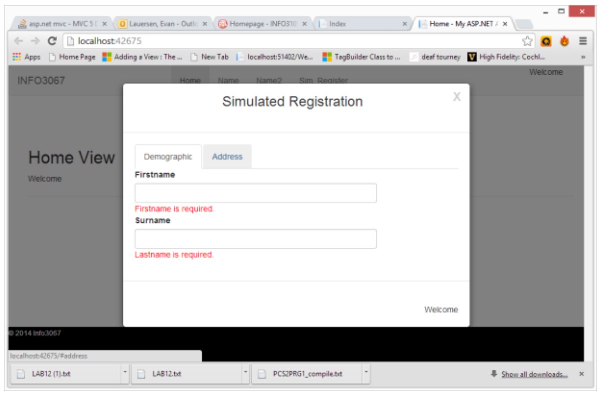
Add a new file to your Scripts folder called **exerciseJS.js** with the following contents:

```
// standalone function
$(function () {

    // click on tab disable click if errors - register popup
    $("#myTabs .nav, nav-tabs").click(function (e) {
        $("#form0").validate();
        if ($("#form0").valid()) {
            return true;
        }
        else {
            return false;
        }
    }); //tabs click

});
```

Don’t forget to update the _Layout.cshtml file to include this new javascript file.

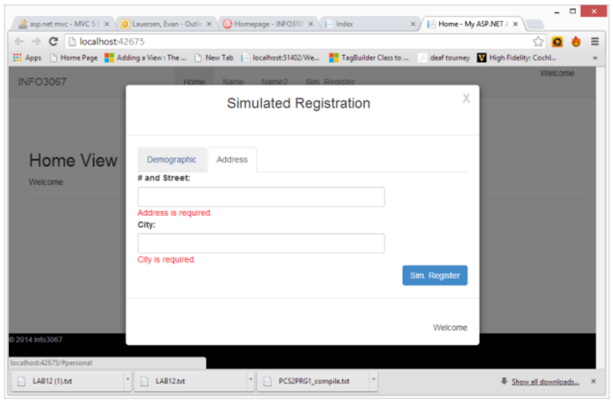
The code is checking the form’s **valid** property (boolean). If false, trap the event and if true, allow the event to proceed. In this case the event would be the click event.

Now test out both tabs for validation:



Firstname is required

Lastname is required



Address is required

City is required

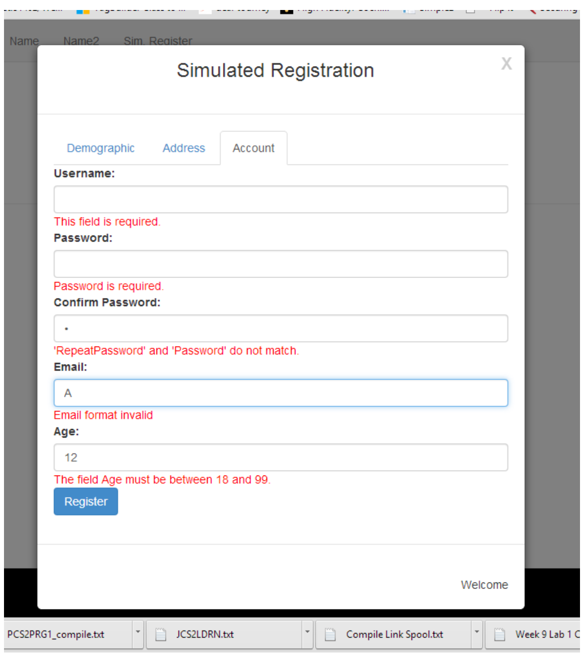
Additional Validation Attributes

In this class’s lab, we’ll test 3 additional validation attributes that we’re going to use in the case study:

- **Range** – the Range attribute is to ensure that value falls into a specific range. For the case study we’ll test and see if age is between 18-99 the syntax for this attribute would be: **[Range(18, 99)]**
- **RegularExpression** – this attribute applies a regular expression against a property. Candidates for using this would be an email address, postal code, phone number…The syntax is as follows (example shows email reg ex): **[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")]**
- **CompareAttribute** – used to make sure two fields have the same data, typically used for password/confirm password combinations. Syntax is as follows: **[CompareAttribute("Password", ErrorMessage = "Passwords don't match.")]**
- **Note –** you can have more than one attribute on a property

LAB 3

- Add a 3rd tab to the exercise to include the additional attributes, specifically have 5 fields on the tab:
    1. Username
    2. Password
    3. Confirm Password
    4. Email
    5. Age
- Move the submit button from tab 2, to the new tab 3
- Submit 5 screen shots in one Word doc
    1. SampleCustomerModel class, updated to include new attributes
    2. Tab 1 all required fields (Firstname, Lastname) in error (see above)
    3. Tab 2 all required fields (Address1 and City) in error (see above)
    4. Tab 3 (see below)
        - Leave Username Empty
        - Leave Password Empty
        - Confirm PW – letter A
        - Email – letter A
        - Age - 12
    5. Tab 3 after fields are fixed and the button is clicked:


Simulated Registration

Username:

This field is required.

Password:

Password is required.

Confirm Password:

•

'RepeatPassword' and 'Password' do not match.

Email:

A

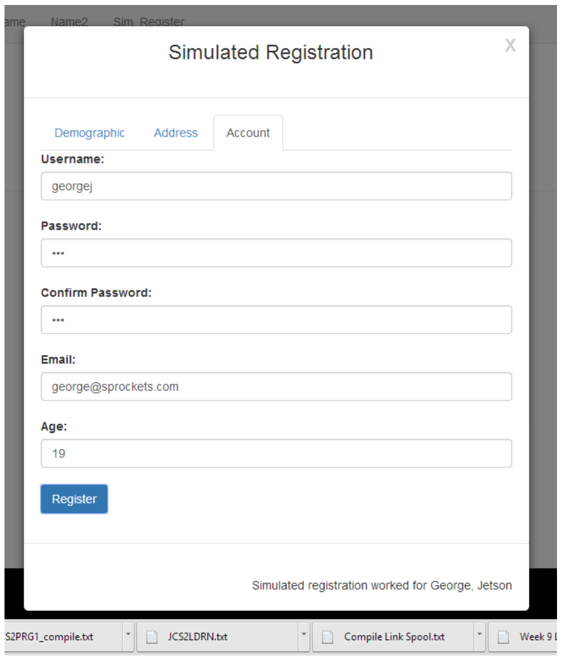
Email format invalid

Age:

12

The field Age must be between 18 and 99.

Welcome


Simulated Registration

Username:

georgej

Password:

•••

Confirm Password:

•••

Email:

george@sprockets.com

Age:

19

Simulated registration worked for George, Jetson

Summary of Key Terms

- Html Helper
- AJAX forms
    - Ajax options
- Linking Model fields to html helpers with LINQ
- @Styles.Render
- @Scripts.Render
- Bootstrap Tabs
- JQuery libraries
    - jquery.unobtrusive-ajax.min.js
    - jquery.validate.min.js
    - jquery.validate.unobtrusive.min.js
- Validation
    - Client Side
        - Html5, JQuery/Javascript
    - Server Side
        - Attributes – Compare, Range, Required, Regular Expression