Lab 3 – A Look at Custom HTML Helpers and Partial Views

Introduction

The ASP.NET Framework has been designed to help developers be productive while at the same time create advanced web sites. A key feature is to enable fine grained control over content being rendered to the browser. The content is often dynamic depending on criteria defined in the logic of a site.

Custom HTML Helpers

The HtmlHelper class in ASP.NET MVC enables you to programmatically create HTML content. All Html Helpers work by generating content and returning it as a string. In a previous lab we saw how helpers could be used to create forms or menu items based on provided input. It is possible to create custom Html Helpers by creating **Extension Methods** for the HtmlHelper class.

The follow class shows a simple “Copyright Helper” which allows us to pass in text and will render content showing the current year followed by text injected.

```
public static class CopyrightHelper
{
    public static MvcHtmlString Copyright(this HtmlHelper htmlHelper,
        string text)
    {
        MvcHtmlString result = new MvcHtmlString("Copyright " + DateTime.Now.Year + " " + text);
        return result;
    }
}
```

A more complex example could use some sort of conditional logic to potentially alter the out. In the current layout view we are using the ActionLink helper to render the HTML links for our menu navigation. I very commonly requested feature is to have the current page the user is on highlighted in the navigation. Unfortunately the way we currently have the navigation setup in the shared layout there is no easy way to display this. One option would be to move the navigation out of the shared file and into each page, but that creates a lot of unwanted duplication.

```
<ul class="nav navbar-nav">
     <li>@Html.ActionLink("Home", "Index", "Home")</li>
     <li>@Html.ActionLink("Meetings", "Index", "Meetings")</li>
     <li>@Html.ActionLink("About", "Index", "About")</li>
     <li>@Html.ActionLink("Contact", "Index", "Contact")</li>
</ul>
```

An ideal solution would be that if we could somehow make the Html Helper aware of it’s current controller and action and render the content accordingly. The code shown below for the MenuItemHelper lets us in text, a controller, and an action in a similar way, but it will then check in the current ViewContext to see if the current route data matches, and if so will apply an additional “active” css class to that element.

```
public static class MenuItemHelper
{
    public static MvcHtmlString MenuItem(this HtmlHelper htmlHelper,
        string text,string action,string controller)
    {
        var currentAction = (string)htmlHelper.ViewContext.RouteData.Values["action"];
        var currentController = (string)htmlHelper.ViewContext.RouteData.Values["controller"];
        var sb = new StringBuilder();
        string activeClass = "active";

        MvcHtmlString actionLink;
        actionLink = htmlHelper.ActionLink(text, action, controller);
```

```
            if ((string.Equals(currentAction,action,StringComparison.CurrentCultureIgnoreCase)) &
              (string.Equals(currentController,controller,StringComparison.CurrentCultureIgnoreCase)))
            {
                sb.AppendFormat("<li class=\"{0}\">{1}</li>",activeClass,actionLink);
            }
            else
            {
                sb.AppendFormat("<li >{0}</li>",actionLink);
            }
            MvcHtmlString result = new MvcHtmlString(sb.ToString());
            return result;
        }
    }
```

If we replace our old navigation with the following

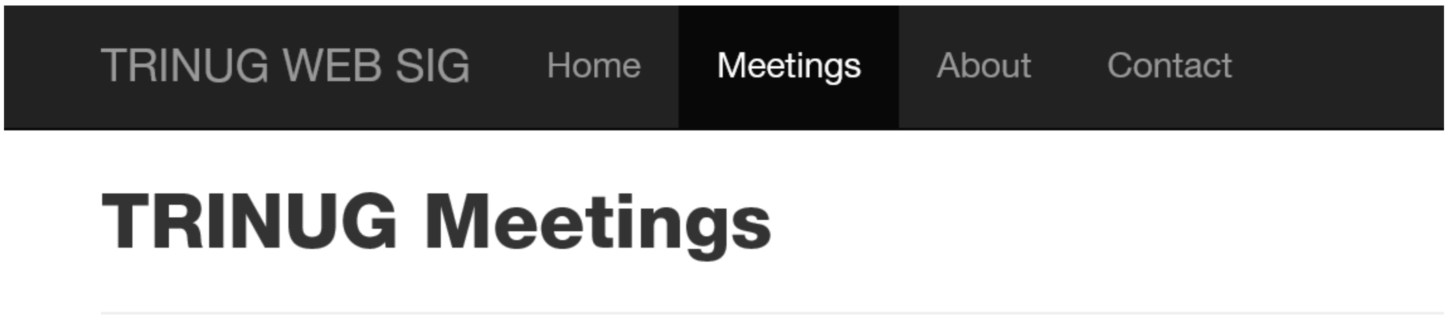
```
                <ul class="nav navbar-nav">
                    @Html.MenuItem("Home", "Index", "Home")
                    @Html.MenuItem("Meetings", "Index", "Meetings")
                    @Html.MenuItem("About", "Index", "About")
                    @Html.MenuItem("Contact", "Index", "Contact")
                </ul>
```

The result is that the "Active" menu item is now automatically highlighted for us.

HTML Rendered:

```
              <ul class="nav navbar-nav">
                     <li ><a href="/">Home</a></li>
                     <li class="active"><a href="/Meetings">Meetings</a></li>
                     <li ><a href="/About">About</a></li>
                     <li ><a href="/Contact">Contact</a></li>
              </ul>
```

Result:



Partial Views

In lab 1 we saw how we could use Visual Studio to create views for us that could be coupled with specific controller actions. Similar to how we could use code to control Html content with code, there are times where it makes sense to break our views down into smaller components know as partial views. In our existing Meetings/Index view we use iterate over the model to create a table. We are going create a similar list on the main Home/Index page and then customize it using partial views.

We need to add some code to our action to load the list of meetings and pass it into the view.

```
        var meetingRepository = new MockMeetingRepository();
        List<Meeting> meetings = meetingRepository.GetAll();

        return View(meetings);
```

We need to add the list of meetings to the Home/Index view by adding the following at the very top of the file

```
@model IEnumerable<MvcDemo.Models.Meeting>
```

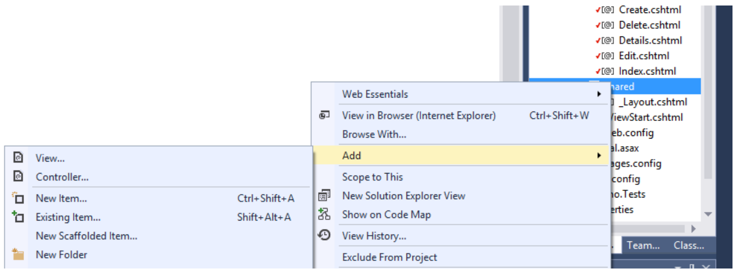
In our Meetings page we use the following code to display iterate through the collection of meetings and display them.

```
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Title)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Description)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.MeetingDate)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.Id }) |
            @Html.ActionLink("Details", "Details", new { id=item.Id }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.Id })
        </td>
    </tr>
}
```

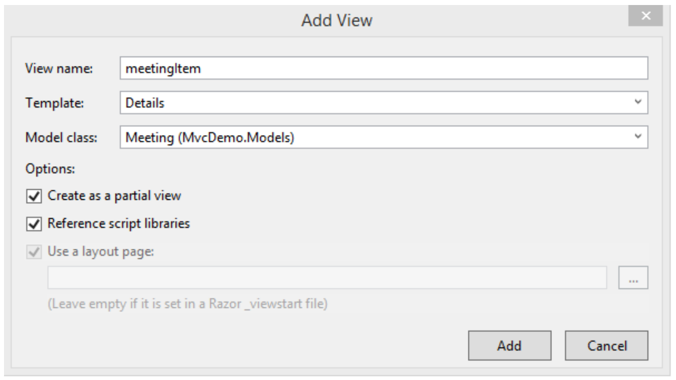
This works, but we are going to do it a little differently this time. Instead let's start by adding the following

```
@foreach (var meeting in Model)
{
   @Html.Partial("meetingItem", meeting)
}
```

This will display a "partial view" called "meetingItem" which we need to create. To do this right click on the Views/Shared folder and select Add->View



We will call the view "meetingItem" and set it to the details template for the Meeting model class. Lastly we will select the "Create as partial view" option.

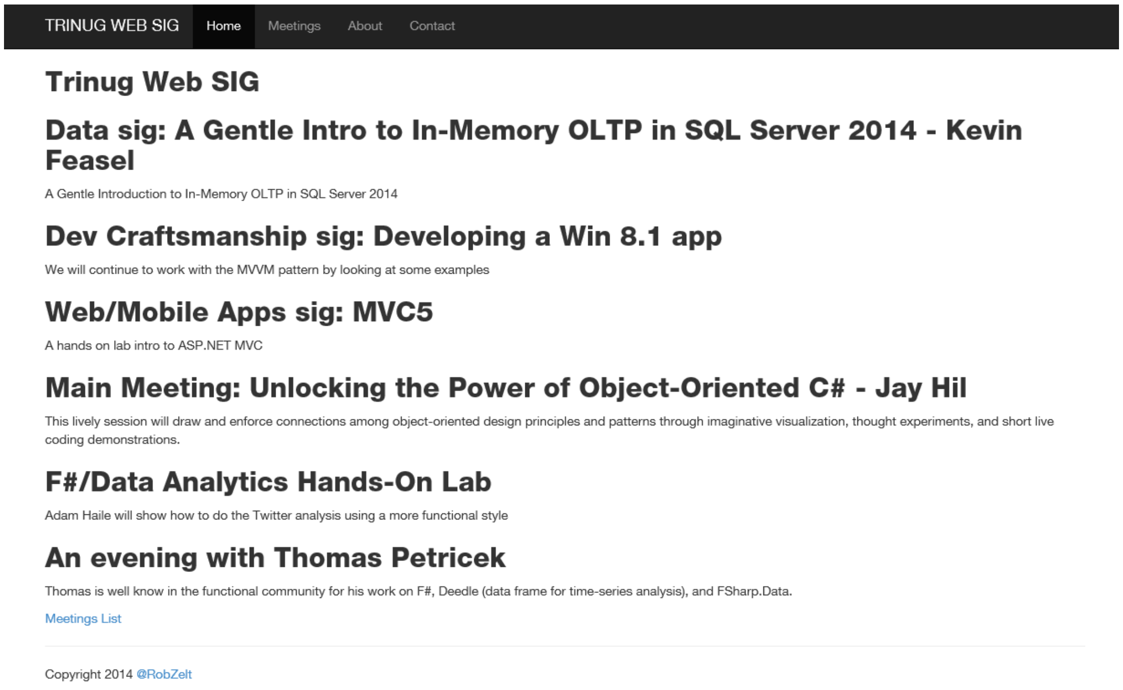


View name: meetingItem
Template: Details
Model class: Meeting (MvcDemo.Models)
Options:
[✓] Create as a partial view
[✓] Reference script libraries
[✓] Use a layout page:
(Leave empty if it is set in a Razor _viewstart file)

This creates a Details form for us for all of the model properties similar to how we created our details, edit, and create pages previousl, but it does not include the layout sections or other html/body elements. In this case we don't want most of the content so let's delete everything within the main <div> and the links at the bottom. We'll also add a couple of lines back in for the meeting title and description like this:

```
@model MvcDemo.Models.Meeting

<div>
    <h2>@Model.Title</h2>
    <p>@Model.Description</p>
</div>
```

Go ahead and run the website and let's look at what is displayed. You should see something similar to this.



Trinug Web SIG

Data sig: A Gentle Intro to In-Memory OLTP in SQL Server 2014 - Kevin Feasel

A Gentle Introduction to In-Memory OLTP in SQL Server 2014

Dev Craftsmanship sig: Developing a Win 8.1 app

We will continue to work with the MVVM pattern by looking at some examples

Web/Mobile Apps sig: MVC5

A hands on lab intro to ASP.NET MVC

Main Meeting: Unlocking the Power of Object-Oriented C# - Jay Hil

This lively session will draw and enforce connections among object-oriented design principles and patterns through imaginative visualization, thought experiments, and short live coding demonstrations.

F#/Data Analytics Hands-On Lab

Adam Haile will show how to do the Twitter analysis using a more functional style

An evening with Thomas Petricek

Thomas is well know in the functional community for his work on F#, Deedle (data frame for time-series analysis), and FSharp.Data.

Meetings List

Copyright 2014 @RobZelt

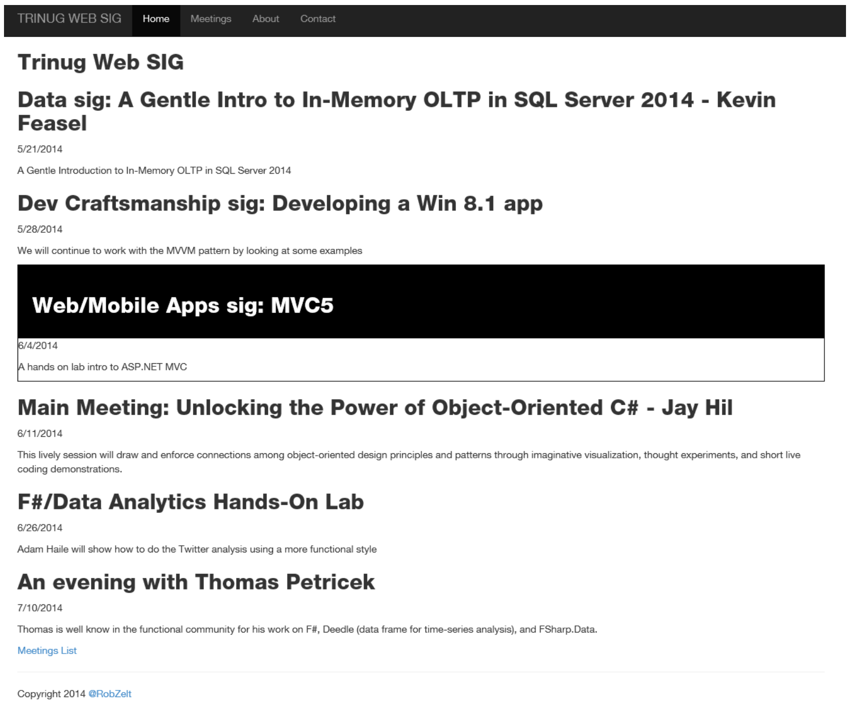
The partial view now gives us a nice modular template to handle our content. We could very easily create a second view (Simply copy the first one and let's call it webMeetingItem) and let's add some customization specific for the web sig.

```
@model MvcDemo.Models.Meeting

<div style="border-color: black; border-style:solid; border-width: thin;">
    <div style="background-color:black; color:white; padding: 20px;">
        <h2>@Model.Title</h2>
    </div>
    <p>@Model.MeetingDate.ToShortDateString()</p>
    <p>@Model.Description</p>
</div>
```

And now back in our main view lets add some logic to check the title for "Web" and render a different view. (Obviously we would want to come up with a more robust way of identifying different meeting types)

```
@foreach (var meeting in Model)
{
    if (meeting.Title.Contains("Web"))
    {
        @Html.Partial("webMeetingItem", meeting)
    }
    else
    {
        @Html.Partial("meetingItem", meeting)
    }
}
```

When we run this we can see we now have a different template being used for the web meeting.



Trinug Web SIG

Data sig: A Gentle Intro to In-Memory OLTP in SQL Server 2014 - Kevin Feasel

5/21/2014

A Gentle Introduction to In-Memory OLTP in SQL Server 2014

Dev Craftsmanship sig: Developing a Win 8.1 app

5/28/2014

We will continue to work with the MVVM pattern by looking at some examples

Web/Mobile Apps sig: MVC5

6/4/2014

A hands on lab intro to ASP.NET MVC

Main Meeting: Unlocking the Power of Object-Oriented C# - Jay Hil

6/11/2014

This lively session will draw and enforce connections among object-oriented design principles and patterns through imaginative visualization, thought experiments, and short live coding demonstrations.

F#/Data Analytics Hands-On Lab

6/26/2014

Adam Haile will show how to do the Twitter analysis using a more functional style

An evening with Thomas Petricek

7/10/2014

Thomas is well know in the functional community for his work on F#, Deedle (data frame for time-series analysis), and FSharp.Data.

Meetings List

Copyright 2014 @RobZelt

This method can be very useful when content needs to be displayed in different ways for different reasons.

Partial Views can also be called through controller actions and used to dynamically render portions of a view. An example of this is to create a model window that appears over the list instead of navigating to specific page. To show this we will alter our Meetings/Index list page.

In addition to the CSS styles that Bootstrap includes it also provides a number of JavaScript functions including the ability to display the contents of a <div> as a dialog. To see the basics of this, please add the following to the bottom of the Meetings/Index view

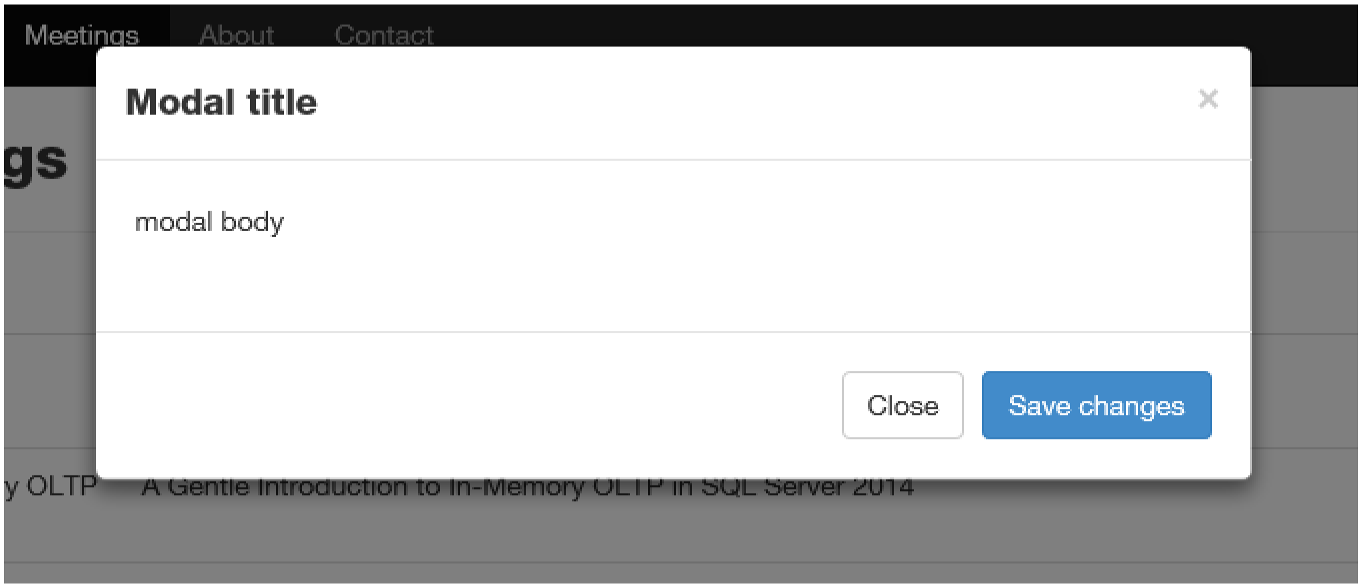
```
<script src="~/Scripts/jquery-1.10.2.js"></script>
<script src="~/Scripts/bootstrap.js"></script>

<script>
    function DisplayModal(meetingId) {
            $("#basicModal").modal(); // show dialog
        }
</script>

<div class="modal fade" id="basicModal" tabindex="-1" role="dialog" aria-labelledby="basicModal" aria-
hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal" aria-hidden="true">×</button>
                <h4 class="modal-title">Modal title</h4>
            </div>
            <div class="modal-body">
                <p>modal body</p>
            </div>
            <div class="modal-footer">
```

```
                <button type="button" class="btn btn-default" data-dismiss="modal">Close</button>
                <button type="button" class="btn btn-primary">Save changes</button>
            </div>
        </div>
    </div>
</div>
```

And also replace the Html.ActionLink for Edit with

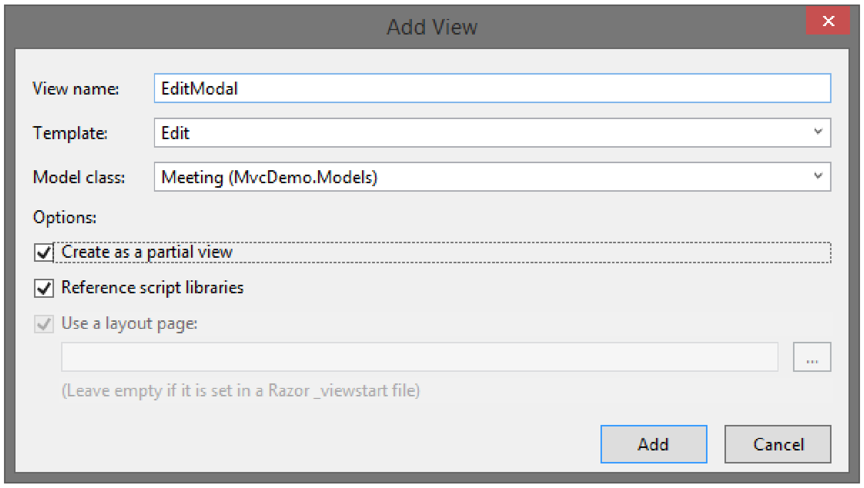
```
<a href="#" onclick="DisplayModal(@item.Id)">Edit</a>
```

Now when we click the Edit link we will see the following:



The next step is for us to replace the contents of the dialog with an html form to allow the editing of the data.

Let’s first create a new view in Meetings called Edit Model


Add View

View name: EditModal

Template: Edit

Model class: Meeting (MvcDemo.Models)

Options:

[✓] Create as a partial view

[✓] Reference script libraries

[✓] Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

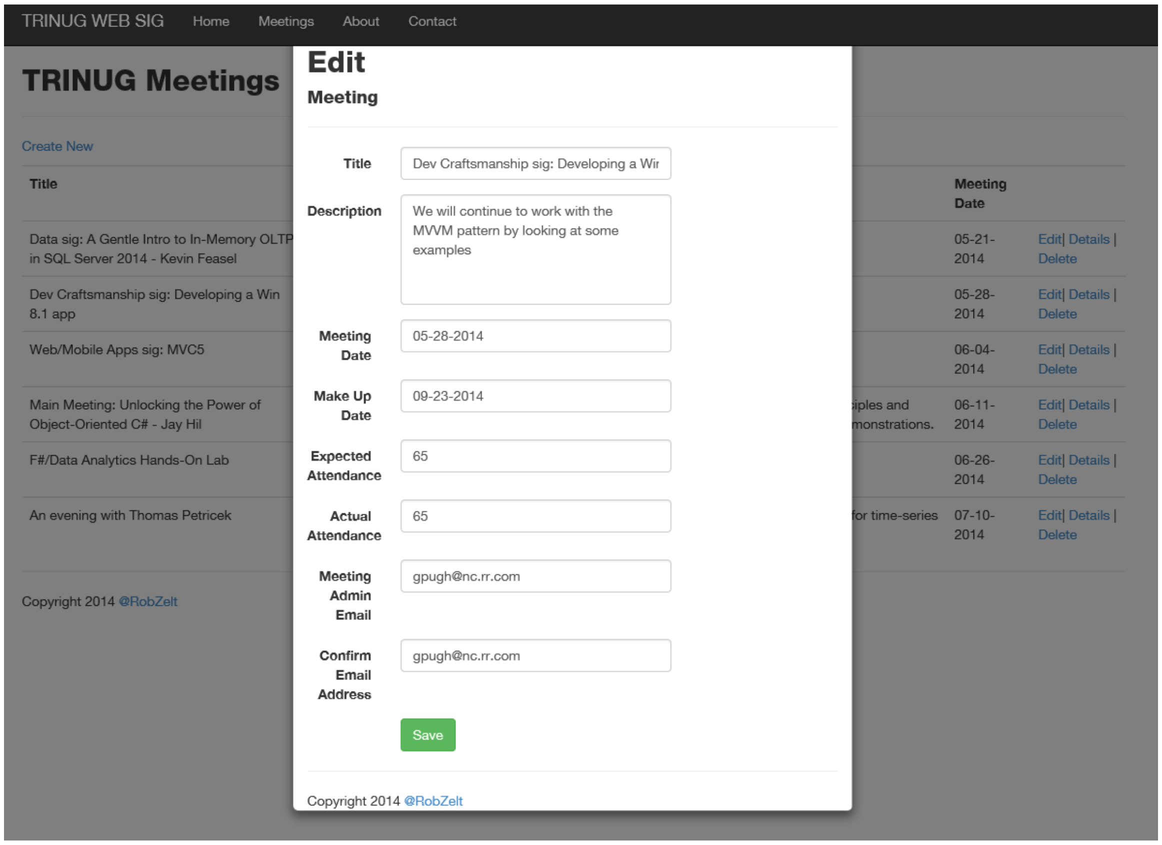
Next in the Meetings controller we need to add an action for EditModal. (One for Get and one for Post)

```
        public ActionResult EditModel(int id)
        {
            //Return a view with a form containing the editable contents of the meeting ID Provided.
            Meeting meeting = meetingRepository.Get(id);
            return View(meeting);
        }

       [HttpPost]
        public ActionResult EditModel(Meeting meeting)
        {
            //Return a view with a form containing the editable contents of the meeting ID Provided.
            meetingRepository.Update(meeting);
            return RedirectToAction("Index");
        }
```

Lastly we need to update the DisplayModel() JavaScript function to use JQuery to populate our content for us.

```
<script>
    function DisplayModal(meetingId) {
            $('.modal-content').load('@Url.Action("EditModal")/' + meetingId);
            $("#basicModal").modal(); // show dialog
        }
</script>
```

We should now be able to click edit and alter the content in a modal window.



TRINUG Meetings

Create New

Title		Meeting Date	
Data sig: A Gentle Intro to In-Memory OLTP in SQL Server 2014 - Kevin Feasel		05-21-2014	Edit| Details | Delete
Dev Craftsmanship sig: Developing a Win 8.1 app		05-28-2014	Edit| Details | Delete
Web/Mobile Apps sig: MVC5		06-04-2014	Edit| Details | Delete
Main Meeting: Unlocking the Power of Object-Oriented C# - Jay Hil	iples and monstrations.	06-11-2014	Edit| Details | Delete
F#/Data Analytics Hands-On Lab		06-26-2014	Edit| Details | Delete
An evening with Thomas Petricek	for time-series	07-10-2014	Edit| Details | Delete

Copyright 2014 @RobZelt

Edit

Meeting

Title	Dev Craftsmanship sig: Developing a Wir
Description	We will continue to work with the MVVM pattern by looking at some examples
Meeting Date	05-28-2014
Make Up Date	09-23-2014
Expected Attendance	65
Actual Attendance	65
Meeting Admin Email	gpugh@nc.rr.com
Confirm Email Address	gpugh@nc.rr.com

Save

Copyright 2014 @RobZelt

Congratulations! You made it through another lab. We will use this base application and extend it in future labs.

The completed files for this lab are available at https://github.com/robzelt/IntroMvcDemo . Please send any comments or feedback to me at robzelt@robzelt.com.