ISSUE 10 | JAN -FEB 2014

DNCMagazine

www.dotnetcurry.com

Up and Running with AngularJS and ASP.NET MVC

Visual Studio Online 2013 – Developing in the Cloud

ASP.NET MVC 5 Fundamentals

5 New Features of ASP.NET Web API 2.0

SOFTWARE GARDENING - Using the right soil

What's New in .NET Framework 4.5.1

CONTENTS

stay connected

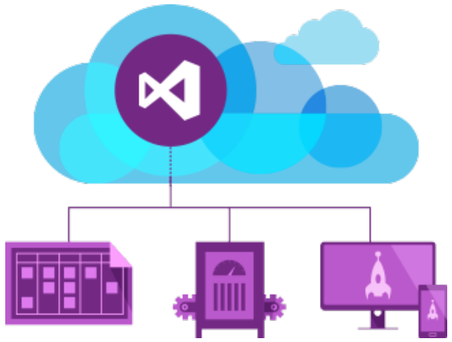
www.Facebook.com/DotNetCurry

@dotnetcurry

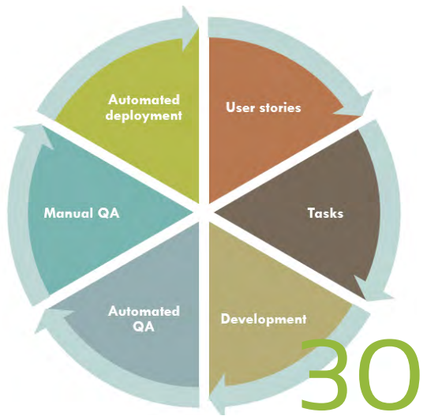
www.DotNetCurry.com/magazine

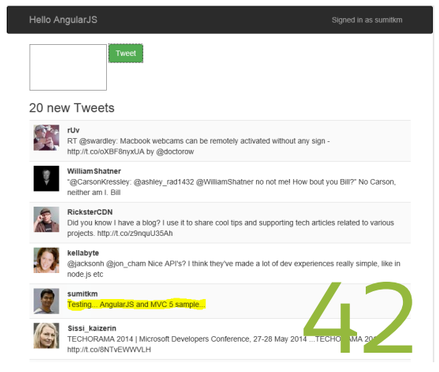
POWERED BY

a2Z Knowledge Visuals



Original image: http://i.visualstudio.com/dynimg/IC691936.png





LETTER FROM THE EDITOR

Happy New Year Readers! New releases in Visual Studio, SQL Server, Web, Cloud, Phone and Windows technologies had kept everyone busy in 2013! As we start a brand New Year here at DNC Magazine, we've got a bumper crop of *What's New* articles this month.

This month's issue of the DNC .NET Magazine leads off with Gouri Sohoni introducing Visual Studio Online, a combination of a powerful IDE with a set of rich developer services running in the Cloud. Mahesh Sabnis highlights the changes in ASP.NET MVC 5.0 with plenty of code walkthroughs of the new features. Pravin Dabade discusses some new features added in the latest version of the .NET Framework - 4.5.1.

Craig Berntson in his Software Gardening column, follows up on the previous edition's column, debunking some myths of comparing software to construction. In this edition, Craig talks about using the right soil, which is choosing Agile methodology over the Waterfall model to manage projects. Highly recommended!

ASP.NET Web API which makes HTTP a first-class citizen of .NET, has a brand new version. Pravin talks about what's new in the latest version of Web API 2.0. Saving the best for the last, Sumit Maitra in his mammoth article introduces some fundamental concepts of AngularJS and gives you a real world development experience by creating a Twitter client in AngularJS and ASP.NET MVC.

With the New Year, we *aren't* making any new resolutions. Instead we plan to continue and strengthen our initiative to bring fresh and cutting edge development technology content for you. We also promise to keep our Novice developer audiences in mind, and you should see some Beginner articles and a new .NET FAQ section in the forthcoming issues.

A thought always lingers our mind: How do we know which topics and technologies are most valuable to you as a reader? Are we over-covering some technologies or are there any technologies been given the miss? In truth, the only way to know is by receiving regular feedback from you. E-mail me at suprotimagarwal@dotnetcurry.com and let us know!

Suprotim Agarwal
Editor in Chief

Editor In Chief • Suprotim Agarwal
suprotimagarwal@dotnetcurry.com

Art Director • Minal Agarwal
minalagarwal@a2zknowledgevisuals.com

Contributing Writers • Craig Berntson, Gouri Sohoni, Mahesh Sabnis, Pravin Dabade, Sumit Maitra

Writing Opportunities • Carol Nadarwalla
writeforus@dotnetcurry.com

Advertising Director • Suprotim Agarwal
suprotimagarwal@dotnetcurry.com

Next Edition • 3rd March 2014

Visual Studio Online 2013 – Developing in the Cloud

Visual Studio Online (VSO) is a cloud based offering from Microsoft for teams that are developing software. It includes foundation services like source repository, tools for project planning and tracking and execution of build for Continuous Integration. Since Team Foundation Service (TFS) is in the cloud, teams can start working on these features without spending any time in installation and configuration. The infrastructure provisioning and maintenance required for these services is taken care of by Microsoft. As soon as the organization subscribes to VSO, it can start using different features from an integrated IDE like Visual Studio or Eclipse.

Visual Studio Online was called as Team Foundation Service until recently. It provides storage and endpoints of the services in the cloud for the team to work.

The VSO features are licensed in 3 major categories as follows:

Visual Studio Online: Basic (5 users free)

Code repositories, Backlog, Track bugs and tasks, Integration with IDEs like VS, Eclipse, Run CI Build, and Includes Visual Studio Express for Web, Windows or Windows Desktop

Visual Studio Online: Professional

All features from basic, up to 10 users per account, subscription to VS Professional IDE

Visual Studio Online: Advanced

All features from basic, break down complex project with Agile Portfolio features, Team Rooms, Integrated feedback request, Integration with major IDEs, Visual Studio Express

Let us first see who can really be benefited from Visual Studio Online.

Any organization which does not want to invest in on-premises installation of Team Foundation Server can use VSO. The team can start working within 5 minutes on a project without getting into any infrastructure specific details. If the team is working in multiple locations, VSO will be ideal to use.

Considering the fact that VSO is a cloud based set of services, some advantages immediately come to mind. Apart from no infrastructure required in-premise, VSO also provides all the new updates for a product, without the organization taking any efforts. These new features will be available even before they are made available to on-premises Team Foundation Server. Being cloud enabled, the team can start working on a project in a very short duration. Time required for installation and configuration of TFS in-premise is saved.

We will be exploring following features of VSO in this article:

1. Code Repository
2. Agile Portfolio Management
3. CI Build
4. Team Room
5. Testing
6. Load Testing

Before we start exploring these features, let us see how to get started with subscribing and using VSO.

1. If you do not have a Visual Studio Online account, you can get one by using Microsoft Account or create an account from www.visualstudio.com. If you have a Hotmail email account then you already have a Microsoft account.
2. You should have Visual Studio 2013 installed on your machine. If you do not have it, you can get one for evaluation from http://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx

3. Now you can create your Team Project by logging on to http://<YourVSOAccount>.visualstudio.com

4. Open Visual Studio from your account after creating a project

5. You can configure workspace and get source.

6. You can check in code, queue build, manage workitems and even execute tests.

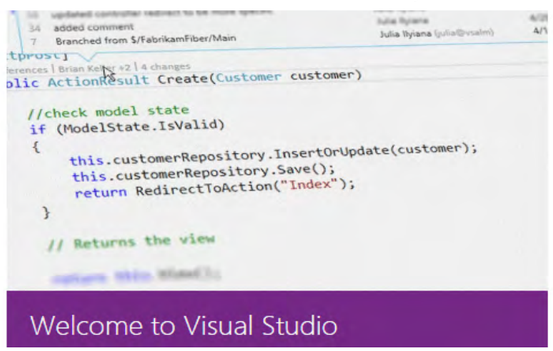
Now that we have seen how to get started with VSO, let's delve into its various features.

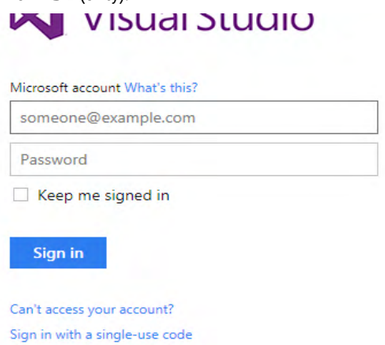
Code Repository

The team members of a project can be at different locations. They require a common repository for storing their work. We have 2 options for source control as Git (Distributed Version Control) or Team Foundation Version Control. Any of these source control mechanisms can integrate with multiple IDEs like Visual Studio or Eclipse. We can provide security of the team by creating groups. You can observe default VSO groups as we have with on premise Team Foundation Server.

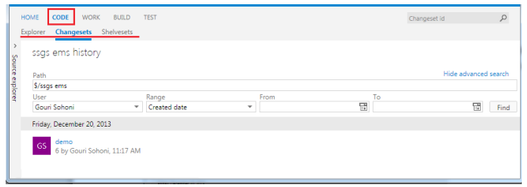
Team members can be added as needed to Team Project. The member can be added to the group as per security requirement. The permissions can be at project level if needed.

The code being developed can be for various Microsoft platforms like Windows, Windows Phones, Desktop or Web. Once the functionality is ready, we can check-in to Visual Studio Online directly from IDE (only).





Now let's shift our focus to the browser based VSO pages. Observe the Code tab which includes Explorer, Changesets and Shelvesets. History can be observed with Changesets and the Shelvesets information can be obtained as well. All changes to the code, the changes made by individual team member or search criteria, can be specified with Changesets.



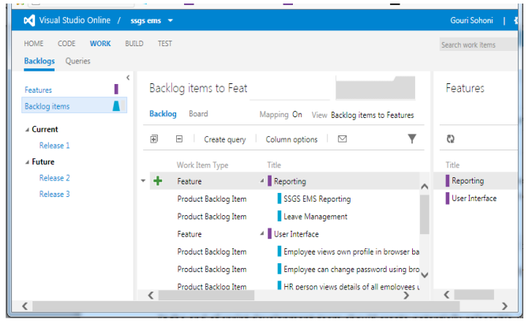
AGILE PORTFOLIO MANAGEMENT

While working in a team, it is required to keep focus on writing right, qualitative code. In order to achieve this goal, we can classify the parts of project into areas or iterations. Project planning tools of VSO will be helpful for this classification. We can just drag and drop the backlog items into required sprint or iteration. We can assign work items to team members by drag-drop mechanism and the team member can concentrate on writing required code. We have already seen how team members can be provided with security by using Groups.

A sprint backlog created by team members helps in understanding work to be done in that sprint. It can be used to manage work. With the work tab, we can also track the changesets to which a particular workitem is associated, as well as other linked work items.

We can observe the current and future releases. A new Product Backlog Item (PBI) or bug can be created. These 2 are represented using 2 colors (blue and yellow).

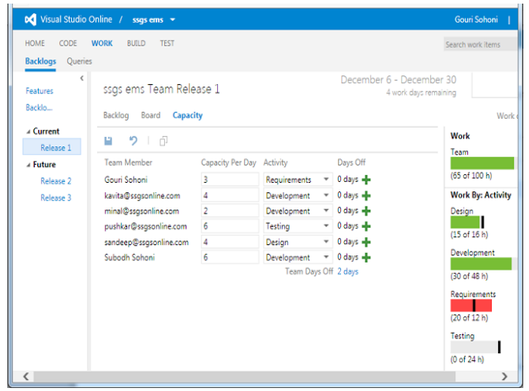
> A *feature* is a logical combination of multiple PBIs that provides a perspective to the product being created. We can create a feature and add PBIs to it. A PBI will be implemented by breaking it into multiple tasks.



Feature wise view of backlog items

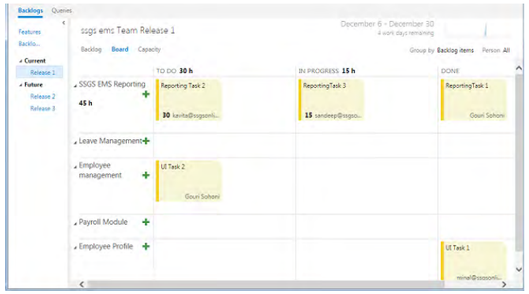
A similar hierarchy of PBIs of tasks can also be viewed.

At the end of sprint development, team should create potentially releasable increments to the product. It can be one or multiple features. Each PBI can be dragged and dropped into a sprint. We can set team's capacity by selecting the *Capacity* tab. Individual capacity can be added. We can even specify if a team member is having day/days off or if it's a holiday for the entire team. Once the capacity is specified and the team velocity (scrum) gives the tasks with remaining work, the overall work separation is represented with a graph.



The green color indicates the work can be completed within the stipulated duration, whereas the red color says that some work needs to be re-allocated.

Once the team starts working on the tasks, the overall view can be seen with the *Board* tab. You can observe the status of various tasks – TO DO, IN PROGRESS and DONE. The workitems can be dragged and dropped to the next state. This view can be either grouped by backlog items or team member wise.

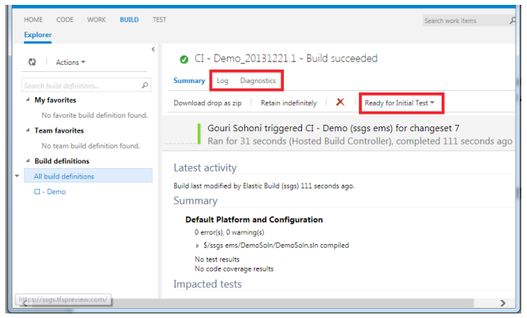


You can view the workitmes with graphs as well. You can add as many as charts as required with different types. Visualizing a graphical view is much easier!

Continuous Integration – Team Build

It is a good practice to capture bugs at an early stage and also maintain the quality of the code. If we specify Continuous Integration of cloud based build services, the quality of the product can be monitored. Every time the code is checked-in, the build will be executed. The build definition needs to be created with Team Explorer from Visual Studio.

The build will be automatically triggered with each check-in. The queued build can be viewed with the *Build* tab. Once the build is completed, its information can be viewed in *Completed* tab. Double clicking the completed build will provide the summary. The log and diagnostics can be viewed as well. The build can be assigned with quality.

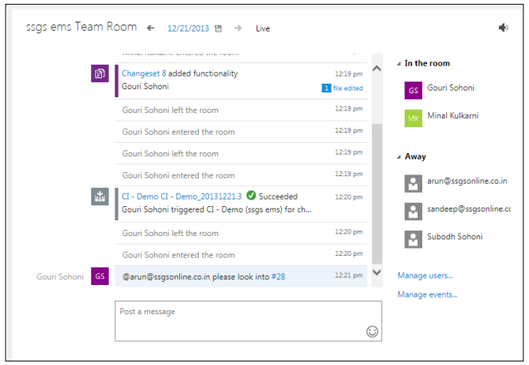


Automated tests can also be run along with build. The build controller used is the *Hosted Build Controller*.

Team Room

The team can collaborate with each other by discussing work in progress and providing information about issues raised. This is taken care by using *Team Room*. This feature is more helpful when team members are in disparate locations.

A team member can enter the room by going to the home page of the team project and selecting Team Room option. A team member can then send a message to another team member. A work item can be included by preceding it with # tag as a part of the message. You can ask a team member to look into a bug by providing the id. A workitem can be opened from a link. An event can be added which can be viewed by all team members eg: completion of build, updating a workitem, changing of code or code review requests. A Room event will appear as links, as shown here:

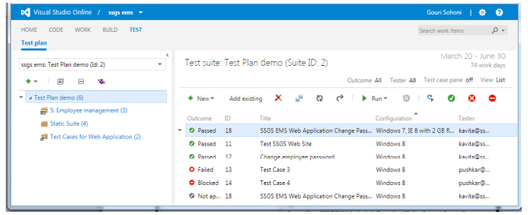


The figure shows the events, the team members currently in Team Room, other members and also how a message can be sent to another team member.

Testing

Visual Studio Online provides 'Test Case Management'. You don't have to switch to Microsoft Test Manager for Test Plans, Test Suites and Shared Steps creation. In order to access the Test tab, we need to provide Full access to the Windows user or Group who is accessing the functionality.

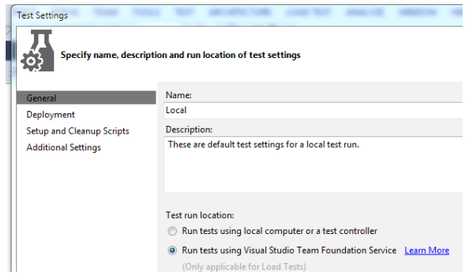
Test Plan, Test Suites, Test Cases or Shared Steps can be seen if they already exist, or new ones can be easily created. There is a link to opening a Test Plan with Microsoft Test Manager. A new Test Case can be created using the usual IDE or multiple Test Cases can be created using a Grid. We can provide all the details to the Test Case like Title, Iteration, Area, and Assigned To (ownership of Test Case). When we have finished creating Test Plans, Test Suites and Test Cases, we can start executing the Test Cases.



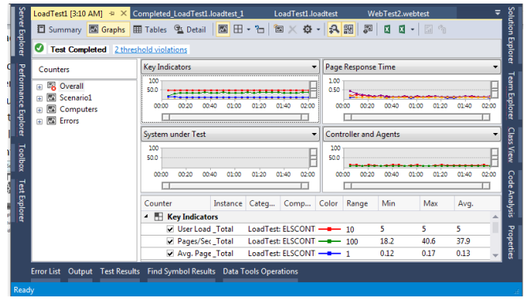
Once the execution starts and we encounter any error, we can create a bug, add a comment to it or add an attachment. Once the execution is completed, we can save and close the runner. We have various options to mark the test case as Pass, Fail, Block or Not applicable. The test case can also be marked as Paused. For Paused Test Case, we later get Resume test as the option. While creating a bug, it is possible to add comments, attachments along with the bug however we cannot create a *Rich* bug. (It requires Microsoft Test Manager installed in order to configure Data Adapters)

Load Testing

Once the functional testing is completed, performance of the project can be tested with increase in users. We do not have to prepare an infrastructure for it, like installing and configuring items. We can use cloud based load testing with ready made virtual machines. The application under test must be available on the Internet. You can use Visual Studio 2013 to create a Load Test. Once the load test is ready with multiple functional tests mix, network mix, concurrent or stepped users, browser mix; we need to configure the load test to run in the cloud.



Before running load tests in Cloud, you need to connect to the Team Project with Visual Studio Online. We can get relevant information about the test, eg: if it is running successfully or not. Once the test is completed, we can download and view the report.



Conclusion

In this article we saw how Visual Studio Online provides an end-to-end, cloud based Application Lifecycle Management solution, which is essentially a collection of developer services that focuses on Agile Team Collaboration, runs on Windows Azure and extends the development experience in the cloud.

It's the beginning of a new era for Visual Studio, so stay tuned for more! ■

Gouri Sohoni, is a Visual Studio ALM MVP and a Microsoft Certified Trainer since 2005. Check out her articles on TFS and Visual Studio ALM at bit.ly/dncm-auth-gso

WHAT'S NEW IN

ASP.NET MVC 5

NEW FEATURES THAT MAKE YOUR WEB APPS SHINE

The ASP.NET MVC 5 Framework is the latest update to Microsoft's popular ASP.NET web platform. It provides an extensible, high-quality programming model that allows you to build dynamic, data-driven websites, focusing on a cleaner architecture and test-driven development.

ASP.NET MVC 5 contains a number of improvements over previous versions, including some new features, improved user experiences; native support for JavaScript libraries to build multi-platform CSS and HTML5 enabled sites and better tooling support.

This article focuses on new features of ASP.NET MVC 5.0. If you are new to ASP.NET MVC, go through this article before moving ahead.

In this article, we will be taking an overview of some of the exciting new fundamental features of ASP.NET MVC 5:

- Scaffolding
- ASP.NET Identity
- One ASP.NET
- Bootstrap
- Attribute Routing
- Filter Overrides

Some of these features like ASP.NET Identity etc are not exactly new features of the core MVC 5 framework, but are worth looking at, as they directly affect/change the way we create ASP.NET applications.

SCAFFOLDING

Visual Studio 2013 includes a new Scaffolding Framework for ASP.NET MVC 5 and the Web API 2. Simply put, Scaffolding is a code generation framework provided for ASP.NET Web Applications. Using this framework, you can quickly generate code that interacts with your data models. This feature reduces the amount of time required to build MVC application with standard data operations. Scaffolding uses code-first approach for data operations.

Note: Visual Studio 2013 does not currently support generating pages for an ASP.NET Web Forms project. The only way out is to add MVC dependencies to your Web Forms project and then use Scaffolding.

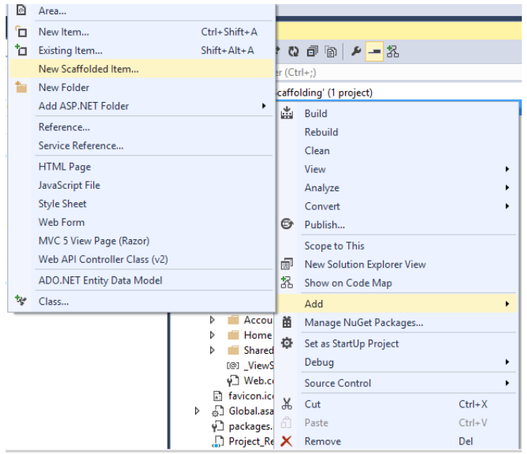
Let's quickly explore Scaffolding in ASP.NET MVC 5.

Step 1: Open VS 2013 and create an ASP.NET MVC application with the name MVC5_Scaffolding. In the model folder, add a new class file with the name Product.cs and add the following class definition in it:

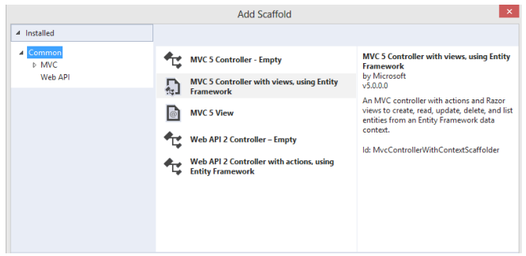
```
using System.ComponentModel.DataAnnotations;
namespace MVC5_Scaffolding.Models
{
public class Product
{
    [Key]
    public int Id { get; set; }
    public string ProdName { get; set; }
    public int ProdPrice { get; set; }
}
}
```

Note that the class Product defines an Id property with the Key attribute representing a unique identification of the Product entity.

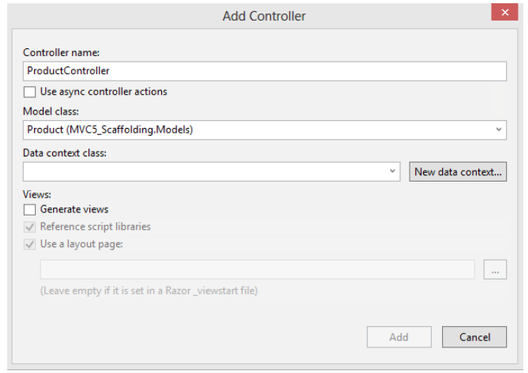
Step 2: In the project, right-click and select Add > New Scaffolded item as shown here:



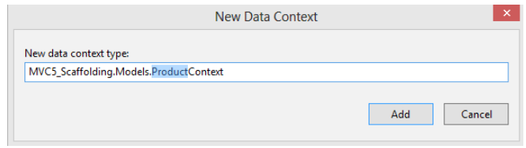
A window for Add Scaffold will be displayed.



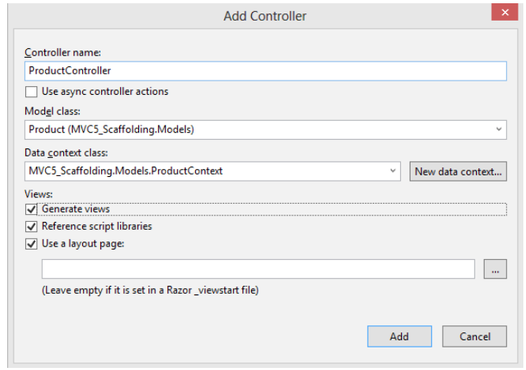
Select the scaffold for MVC 5 Controller with views using Entity Framework, as shown in the screenshot. After clicking the Add button, you should see a window similar to the following:



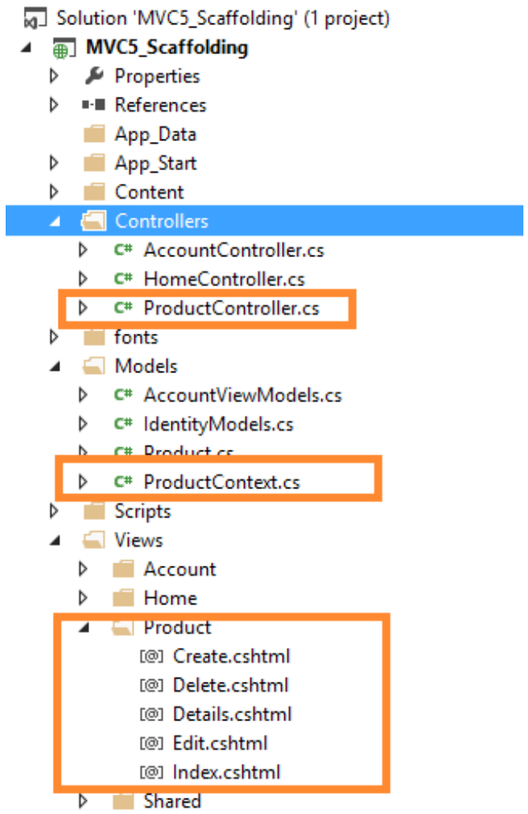
Select the model class from the dropdown. Now to generate the Data Context, click on 'New Data Context'. A window is displayed where the context class name needs to be entered:



Click on *Add* and the control will go back to the *Add Controller window* where you have to check the *Generate Views checkbox* as shown here:



This step will add the ProductController, ProductContext classes and Product folder in Views folder with views for data operations:



This step will also add a connection string of the name *ProductContext* in the web.config file. The code for the Product context class will be as shown here:

```
public class ProductContext : DbContext
{
public ProductContext()
    : base("name=ProductContext")
{
}
public System.Data.Entity.DbSet<MVC5_Scaffolding.
Models.Product> Products { get; set; }
}
```

The context class uses Code-First approach and creates a table of the name Product using DbSet<T> object of the EntityFramework. The Product Controller class interacts with the *ProductContext* class for performing CRUD operations using the views generated. Hence Scaffolding reduces the time for developing MVC data oriented applications.

ASP.NET IDENTITY

In earlier days of ASP.NET 2.0 programming, a membership provider approach was introduced. This allowed the application to store user's data in a SQL Server database. This membership model has changed over the years. The notion that a user can log-in by only using a user-name and password registered in the application, can now be ignored. In today's world, the web has become more social and users connect to each other, and with applications, using social sites like Facebook, Twitter etc. So considering these social integrations, web applications too need to be enhanced to allow users to log-in using their social media credentials.

To get this done, the modern membership framework is now extended to integrate with social credentials and for this purpose, ASP.NET Identity has been introduced. The advantages of the ASP.NET Identity are explained here:

- One ASP.NET Identity System: Can be used across all the ASP.NET Frameworks like WebForms, MVC, Web Pages, Web API, SignalR etc.

- Ease of plugging-in profile data about the user: The user's profile schema information can be integrated with the web application.

- Persistence Control: ASP.NET Identity system stores all user information in the database.

- Social Login Provider: Social log-in providers such as Microsoft Account, Facebook, Twitter, Google, and others can be easily added to the web application.

- Windows Azure Active Directory (WAAD): The Login-in information from WAAD can also be used for authenticating a web application.

- OWIN Integration: ASP.NET Identity is fully compliant with OWIN Framework. OWIN Authentication can be used for login. If you are new to OWIN, read this article.

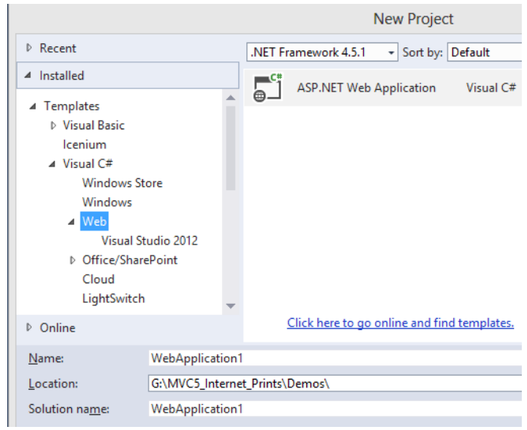
All these features in the ASP.NET Identity membership system are available if you are using Visual Studio 2013. Alternatively, you can also obtain it via NuGet packages Microsoft.Aspnet.Identity.Core and Microsoft.Aspnet.Identity.EntityFramework.

ONE ASP.NET

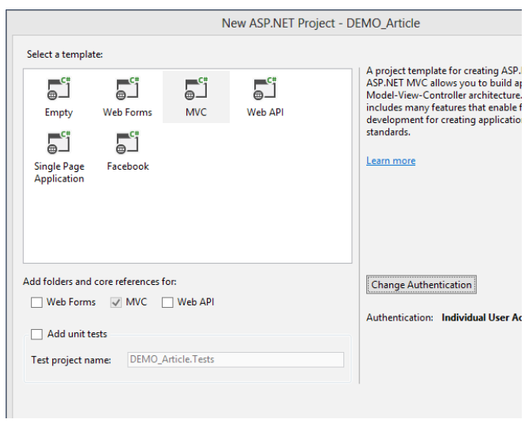
One ASP.NET is a new unified project system for .NET Web Developers. This system makes it easier to work with multiple frameworks like Web Forms, MVC, Web API etc; in a single project. So essentially using the *One ASP.NET* project system, you can use ASP.NET Web Forms and MVC together, and can easily add ASP.NET Web API and SignalR too; in the same Web application.

In Visual Studio 2013, the ASP.NET MVC project template integrates with this new system. One of the useful features while creating a MVC project is that the authentication mechanism can be configured.

Step 1: Open VS 2013 and select File > New > Project, select Web from installed template as below:

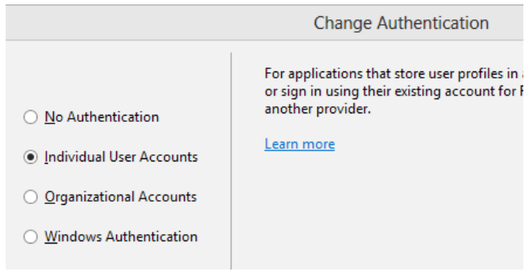


Step 2: Make sure that .NET Framework 4.5.1 is selected; click OK and the following window comes up.



As you can see, a New ASP.NET Project window displays various Web Templates, and based upon the selection of template, the necessary references will get added in the project. Checkboxes indicates the necessary folder structure and core references for the project.

Step 3: Click on *Change Authentication* and the authentication provider windows will be displayed:

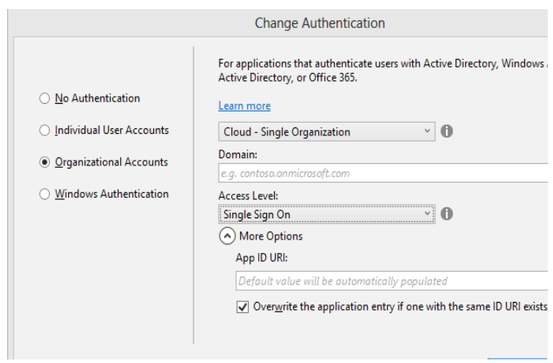


There are four different authentication types that can be set for the application:

1. No Authentication: The application does not require authentication.

2. Individual User Accounts: SQL Server database is used to store user profile information. This authentication can also be extended to provide the end-user with the option to make use of their social profiles like Facebook, Google, Microsoft, Twitter or any other customized provider.

3. Organizational Accounts: The application can authenticate users using the user profiles stored in Active Directory, Windows Azure Active Directory, or Office 365. This provides Single Sign-on access level to the application. The Organizational Accounts require the following details:



Here the application can be configured for:

- Cloud-Single Organization
- Cloud-Multi Organization – This and the previous one can be used when the user authentication information is stored on the Windows Azure Activity Directory (WAAD)
- On-Premises - Used for Active Directory on-premises.
- Domain - The WAAD domain for setting application in it.
- Access Level - The application needs to query or update directory information.
- Application ID URI - Created by appending the project name to the domain.

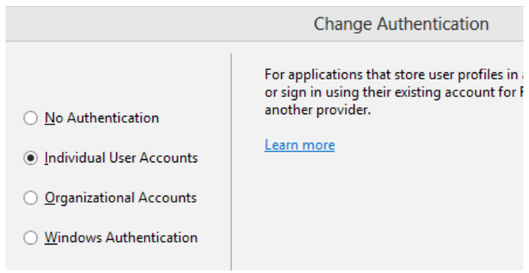
4. Windows Authentication: Used for intranet applications.

Enough theory, let's see an example!

Using Google Authentication for MVC 5 Application

Let's design an ASP.NET MVC 5 application which will enable users to log in using an external authentication provider like Google.

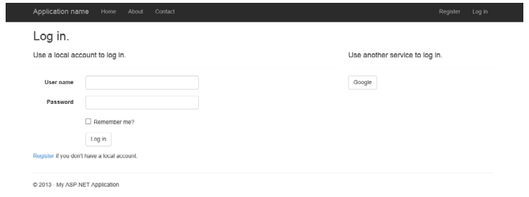
Step 1: Open Visual Studio 2013 and create a MVC application. Select *Individual User Accounts* from the change authentication window as shown here:



Step 2: To enable Google Open ID provider, open the Startup.Auth.cs file from App_Start folder. From the *ConfigureAuth* method of *Startup* class, uncomment the GoogleAuthentication method call as shown here:

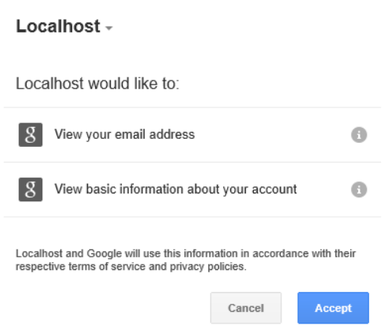
```
public void ConfigureAuth(IAppBuilder app)
{
// Enable the application to use a cookie to store
information for the signed in user
app.UseCookieAuthentication(new
CookieAuthenticationOptions
{
        AuthenticationType =
        DefaultAuthenticationTypes.ApplicationCookie,
        LoginPath = new PathString("/Account/Login")
});
// Use a cookie to temporarily store information about
a user logging in with a third party login provider
app.UseExternalSignInCookie(DefaultAuthenticationTypes.
ExternalCookie);

// Uncomment the following lines to enable logging in
with third party login providers

app.UseGoogleAuthentication();
}
```

Step 3: Run the application; from the Home menu, click on *Log In*. In the Login view, *the Use another service to log in* option will be displayed:

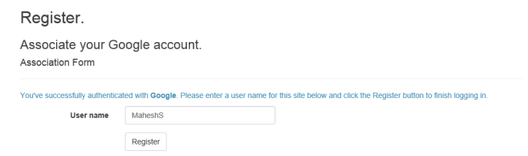


Google provider is now enabled for the application! Once the Google button is clicked, the Google Login page will be displayed where the credential information (Gmail ID/ Password) needs to be entered. The URL on the Google login page contains openid2. This means that Google credentials will be used to login to the site. After entering the credential information, the web application will demand the following permissions:

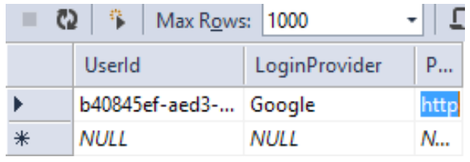
- **View your email address**
- **View basic information about your account.**



Once the Accept button is clicked, the Register view will be displayed as shown below:



This step registers your Gmail credentials with the web site you just created. ASP.NET MVC 5 uses the code-first approach for creating database for storing user's login information. Go to the App_Data folder, a database file of name aspnet-(your application name)-yyyymmdd<applicationid>.mdf will be generated. Open this file in the Server Explorer. In this database, the AspNetUserLogins table will be created. View data from this table and you will find the following information:

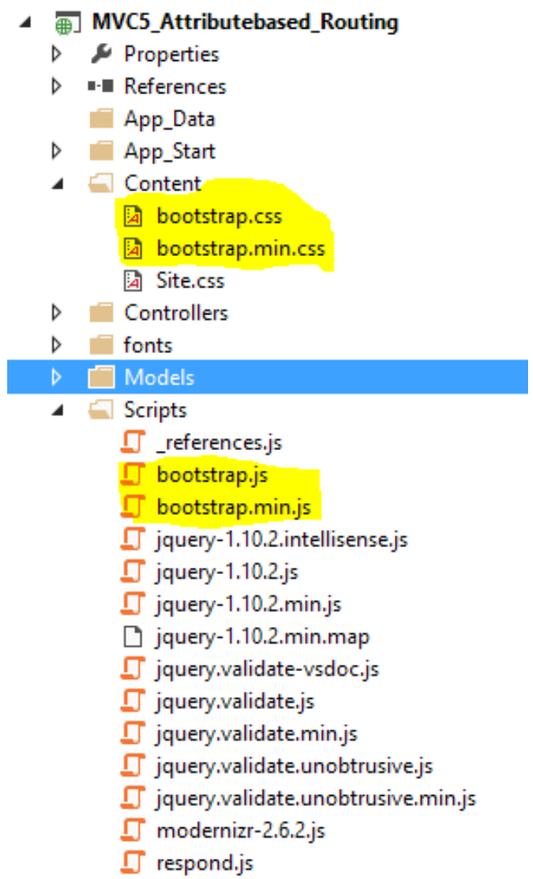


Likewise, other authentication providers like Facebook, Microsoft, Twitter can also be used!

I would strongly recommend you read the ASP.NET MVC 5 Authentication Filters article by Raj Aththanayake where he explores the new IAuthenticationFilter in ASP.NET MVC 5 and explains the CustomAuthentication attribute and how you can use to change the current principal and redirect un-authenticated user to a login page.

BOOTSTRAP

In Visual Studio 2013, Twitter Bootstrap is added as the default user interface framework for an MVC application. Bootstrap is a free collection of HTML and CSS based design templates created at Twitter for designing forms, navigation, buttons, tables etc. Bootstrap can be downloaded from here. The advantage of Bootstrap is that it is used for rapid development of Responsive user interface using basic HTML and CSS based templates. In the MVC 5 project, bootstrap.js and bootstrap.css are already present:



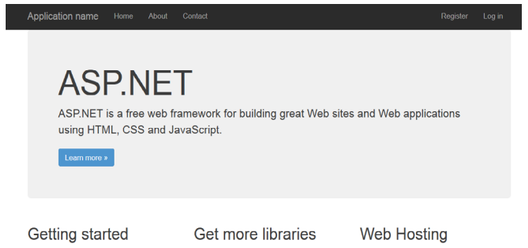
In the MVC 5 project open _Layout.cshtml and you will find a tag under the <body> element as seen here:

```
<div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle" data-toggle="collapse" data-t
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            @Html.ActionLink("Application name", "Index", "Home", null, new { @class
        </div>
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav">
                <li>@Html.ActionLink("Home", "Index", "Home")</li>
                <li>@Html.ActionLink("About", "About", "Home")</li>
                <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
            </ul>
            @Html.Partial("_LoginPartial")
        </div>
    </div>
</div>
```

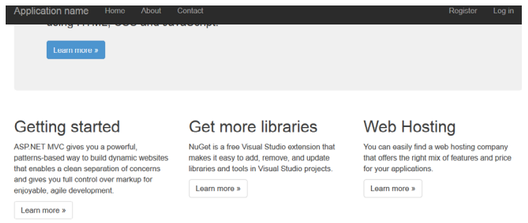
Observe that the tag is decorated with CSS classes like navbar, navbar-inverse, navbar-fixed-top. These classes are declared in the bootstrap.css and it means that the navigation

bar showing *Application name, Home, About* and Contact is displayed on the top with fixed position, so even when the page is scrolled down; the Navigation bar will be fixed on the top as shown here:

Navigation Bar at the beginning of the page:

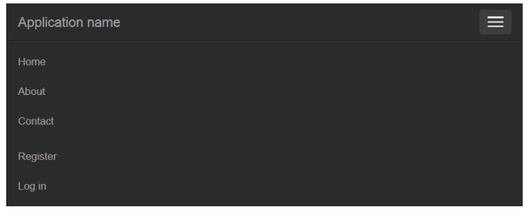


After the page scrolls down, the Navigation bar location is still fixed to the top:



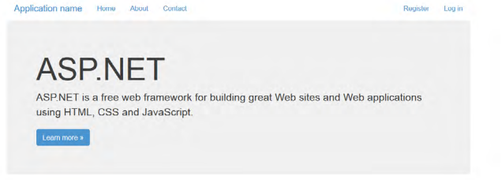
Since Bootstrap promotes Responsive Web Design, if the browser resolution is changed, say to emulate a Phone width, then the page accordingly changes its display to the following:

Observe that there are no Navigation Menu Items you saw earlier. If you click on the browse icon (marked with yellow color) in the right corner, then the Navigation will be displayed:



The nice part here is that the layout is changed automatically; this is the magic and advantage of using Bootstrap.

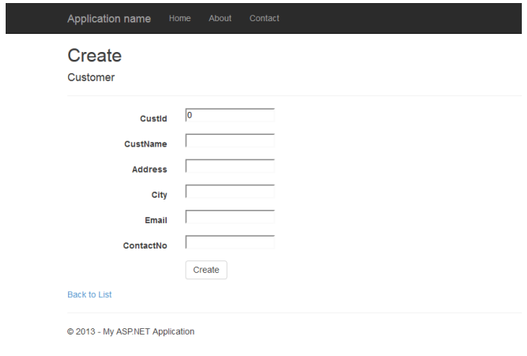
Try by removing the classes applied to the navigation bar for the . You will see that the page appears as shown below:



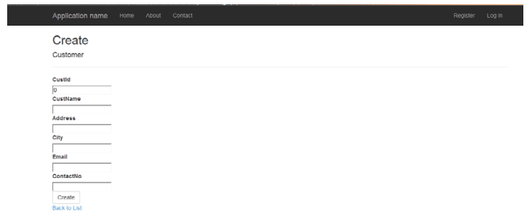
Similarly Bootstrap can also be used for creating nice looking forms using the styles defined in the bootstrap.css. Using Styles like control-label, form-horizontal; rendering can be managed. Consider a *Create Form* for the Customer. The cshtml view markup with style classes generated is as shown here:

```
<div class="form-horizontal">
<h4>Customer</h4>
<hr />
@Html.ValidationSummary(true)
<div class="form-group">
    @Html.LabelFor(model => model.CustId, new { @class
    = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.EditorFor(model => model.CustId)
        @Html.ValidationMessageFor(model => model.
        CustId)
    </div>
</div>

<div class="form-group">
    @Html.LabelFor(model => model.CustName, new { @
    class = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.EditorFor(model => model.CustName)
        @Html.ValidationMessageFor(model => model.
        CustName)
    </div>
</div>

<div class="form-group">
   @Html.LabelFor(model => model.Address, new { @class
   = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.EditorFor(model => model.Address)
        @Html.ValidationMessageFor(model => model.
        Address)
   </div>
</div>

<div class="form-group">
    @Html.LabelFor(model => model.City, new { @class =
    "control-label col-md-2" })
    <div class="col-md-10">
        @Html.EditorFor(model => model.City)
        @Html.ValidationMessageFor(model => model.City)
    </div>
</div>

<div class="form-group">
    @Html.LabelFor(model => model.Email, new { @class =
    "control-label col-md-2" })
    <div class="col-md-10">
        @Html.EditorFor(model => model.Email)
        @Html.ValidationMessageFor(model => model.
        Email)
    </div>
</div>

<div class="form-group">
    @Html.LabelFor(model => model.ContactNo, new { @
    class = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.EditorFor(model => model.ContactNo)
        @Html.ValidationMessageFor(model => model.
        ContactNo)
    </div>
</div>

<div class="form-group">
    <div class="col-md-offset-2 col-md-10">
        <input type="submit" value="Create" class="btn
        btn-default" />
    </div>
</div>
</div>
```

In the mark-up, the style class *control-label* is passed to each label helper with its column location; so if the view is rendered in the browser, it will look like the following:



The label and its corresponding textbox is arranged in a *single horizontal row*. If all styles applied in the mark-up are removed, then the view will be displayed as shown below:



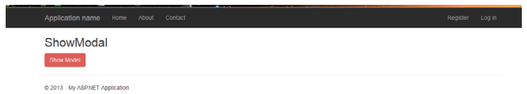
Similarly, the bootstrapper.js file provided in the script folder contains plenty of useful jQuery code which provides interactive components/plugins in a single file. Note that all plugins depend on jQuery (this means jQuery must be included before the plugin files).

One of the nicest feature provided in this file is the support for modal boxes, for which earlier a separate plugin using jQuery UI was needed (jQuery UI dialog). Now components like the Modal boxes can easily be used in our websites using just bootstrap.js and the bootstrap.css. Consider the view below:

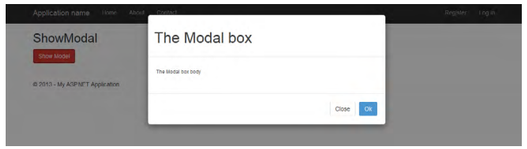
```
<!--The button with the bootstrap class applied on
it-->
<div>
<button class="btn btn-danger" id="showmodalbox">Show
Model</button>
</div>

<!--The Modal box defined using class modal-->
<div id="modalbox" class="modal">
<div class="modal-dialog">
    <div class="modal-content">
        <div class="modal-header">
            <h1> The Modal box</h1>
        </div>
        <div class="modal-body">
            <h6>The Modal box body</h6>
        </div>
        <div class="modal-footer">
            <button class="btn btn-default"
data-dismiss="modal">Close</button>
            <button class="btn btn-primary">Ok</button>
        </div>
    </div>
</div>

</div>
<!--end here-->

@section scripts{
```

```
<script>
    //The script read the modalbox element and define
the
    //modal on it using modal method and calls its
“show state”
    $(function () {
        var modelbox = function () {
            $(“#modalbox”).modal(“show”);
        };
        $(“#showmodalbox”).click(modelbox);
    });
</script>
}
```

In the mark-up and script code, the on the top defines a button which uses the CSS class *btn-danger* defined in the bootstrap.css. The modal box is defined using with id *modalbox*. The with class *modal-dialog* defines a modal using *modal-header, modal-content* and *modal-footer*. The modal footer contains a button which contains the *data-dismiss* attribute. This enables the modal dialog to close when the button is clicked. The script will show the modal dialog when the *Show Modal button* is clicked.



Click on the Show Modal button and a Modal box appears.



The Close button here is used to close the modal dialog. Similarly you can write some code in the OK button too. Hence in MVC 5, with native support of the popular Bootstrap JavaScript library, a developer has a wider range of multi-platform CSS and HTML5 options, than ever before.

ATTRIBUTE BASED ROUTING

The beauty of MVC is in its routing feature. Routing is how ASP.NET MVC matches a URL to an action. In earlier versions of MVC, the routing expressions were provided in the Global.asax class in Application_start event. (Note: In MVC 4, a separate class of name RouteConfig is provided in the App_Start folder.) The route expression set in MVC 4 is similar to the following:

```
public static void RegisterRoutes(RouteCollection
routes)
{
    routes.IgnoreRoute(“{resource}.axd/{*pathInfo}”);

    routes.MapRoute(
        name: “Default”,
        url: “{controller}/{action}/{id}”,
        defaults: new { controller = “Home”, action =
        “Index”, id = UrlParameter.Optional }
    );
}
```

In this code, the defaults are already set to Home controller and its Index action method. This means that when a new URI is requested through the browser addressbar, this expressions has to be re-generated for controller other than Home.

Why to use Attribute Routing?

In MVC 5, to have more control over the URIs of a Web application, we can implement the route definition along with the action methods in the controller class, using attributes. In other words, we use attributes to define routes. To enable attribute based routing, the *RegisterRoutes* method needs to be changed to the following:

```
public static void RegisterRoutes(RouteCollection
routes)
{
    routes.IgnoreRoute(“{resource}.axd/{*pathInfo}”);
    routes.MapMvcAttributeRoutes();
    routes.MapRoute(
        name: “Default”,
        url: “{controller}/{action}/{id}”,
        defaults: new { controller = “Home”, action =
        “Index”, id = UrlParameter.Optional }
    );
}
```

Implementation of Attribute Routing

Step 1: Open Visual Studio 2013 and create a new MVC application, name it as ‘MVC5_Attributebased_Routing’. In the Models folder add a new class file, name it as ‘ModelRepository.cs’ and add the following classes in it:

```
using System.Collections.Generic;

namespace MVC5_Attributebased_Routing.Models
{
public class Customer
{
    public int CustId { get; set; }
    public string CustName { get; set; }
    public string Address { get; set; }
    public string City { get; set; }
    public string Email { get; set; }
    public string ContactNo { get; set; }

}

public class CustomerDatabase : List<Customer>
{
    public CustomerDatabase()
    {
        Add(new Customer() { CustId = 1, CustName =
“Amit”, Address = “AP Road”, City = “Pune”, Email =
“a.b@ab.com”, ContactNo = “1234567” });
        Add(new Customer() { CustId = 2, CustName =
“Ajit”, Address = “BP Road”, City = “Yavatmal”, Email =
“b.b@ab.com”, ContactNo = “2234567” });
        Add(new Customer() { CustId = 3, CustName =
“Abhijit”, Address = “CP Road”, City = “Nagupr”, Email
= “c.b@ab.com”, ContactNo = “3234567” });
        Add(new Customer() { CustId = 4, CustName =
“Sumit”, Address = “DP Road”, City = “Pune”, Email =
“d.b@ab.com”, ContactNo = “4234567” });
        Add(new Customer() { CustId = 5, CustName =
“Sujit”, Address = “EP Road”, City = “Yavatmal”, Email
= “e.b@ab.com”, ContactNo = “5234567” });
        Add(new Customer() { CustId = 6, CustName =
“Rohit”, Address = “FP Road”, City = “Nagpur”, Email =
“f.b@ab.com”, ContactNo = “6234567” });
        Add(new Customer() { CustId = 7, CustName =
“Mohit”, Address = “GP Road”, City = “Pune”, Email =
“g.b@ab.com”, ContactNo = “7234567” });
        Add(new Customer() { CustId = 8, CustName =
“Ranjit”, Address = “HP Road”, City = “Yavatmal”, Email
= “h.b@ab.com”, ContactNo = “8234567” });
        Add(new Customer() { CustId = 9, CustName =
“Kuljit”, Address = “IP Road”, City = “Nagpur”, Email =
“i.b@ab.com”, ContactNo = “9234567” });
    }
}

public class DataAccess
{
    public List<Customer> GetCustomers()
    {
        return new CustomerDatabase();
    }
}

}
```

Step 2: Enable attribute based routing in the RouteConfig.cs class file as explained above.

Step 3: In the Controller folder, add a new controller with the name *CustomerController*. In this controller, add an action method with the name *GetCustomerByCity*:

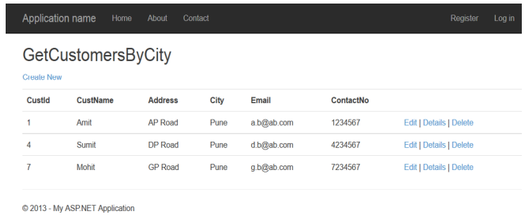
```
public class CustomerController : Controller
{
DataAccess objDs;

public CustomerController()
{
    objDs = new DataAccess();
}

//
// GET: /Customer/
public ActionResult Index()
{
    return View();
}

/// <summary>
/// The method has attribute routing.
/// The parameter is passed as {city}
/// </summary>
/// <param name=”city”></param>
/// <returns></returns>

[Route(“Customers/{city}”)]
public ActionResult GetCustomersByCity(string city)
{
    var customers = from c in objDs.GetCustomers()
                    where c.City == city
                    select c;

    return View(customers);
}
}
```

As you can see, an action method GetCustomersByCity is applied with the Route attribute and a routing expression as *Customers/{city}*

Step 4: Generate View for the above action method and run the application. In the URL, put the following address:

http://MyServer:6764/Customers/Pune

The routing expression Customers/Pune maps with Customers/{city} and the view will be as shown here:



Defining Optional parameter in the URI

The URI parameter can be made optional by adding a *question mark (?)* to the route parameter. An example of the action method with attribute routing and optional parameter is shown here:

```
/// <summary>
/// The action method defines the attribute routing
/// the parameter {city?} is defined as optional
/// </summary>
/// <param name="city"></param>
/// <returns></returns>
[Route("Customers/All/{city?}")]
public ActionResult GetCustomers(string city)
{
    //Check if the city is null or empty
    //If the city value is entered filter customers
    //for that entered city
    if(!string.IsNullOrEmpty(city))
    {
    var customers = from c in objDs.GetCustomers()
                    where c.City == city
                    select c;

    return View(customers);
    }
    //else return all customers
    return View(objDs.GetCustomers());
}
```

In the code we just saw, the *GetCustomers* action method is applied with an attributing routing as shown here: Customers/All/{city?}

Here the {city?} parameter with question mark is optional. After running the application, if the URL is entered as http://localhost:25418/Customers/All then all Customers will be displayed.

However if the URL is entered as http://localhost:25418/Customers/All/Nagpur then all customers only in Nagpur city will be displayed.

Defining the Default value for the URI Parameter

The default value for the URI parameter can be set using *parameter=value*. Here's an example:

```
/// <summary>
/// The action method defines the attribute routing
/// the parameter {city=Nagpur} is defined as default
/// value for the parameter
/// </summary>
/// <param name="city"></param>
/// <returns></returns>
[Route("Customers/City/{city=Nagpur}")]
public ActionResult GetCitywiseCustomers(string city){
        var customers = from c in objDs.GetCustomers()
                        where c.City == city
                        select c;

        return View(customers);
}
```

The action method is applied with the attribute routing: Customers/City/{city=Nagpur}

The city parameter is set to the default value of Nagpur. After running the application, if the URL is specified as: http://localhost:25418/Customers/City, then all customers from Nagpur city will be displayed.

However if the URL is entered as: http://localhost:25418/Customers/City/Yavatmal, then all Customers from the Yavatmal city will be displayed.

Applying with RoutePrefix

Generally if routes in the action methods of the controller start with the same prefix, then we can apply *RoutePrefix* on the controller class as shown here:

```
[RoutePrefix("Customers")]
public class CustomerController : Controller {
[Route("{city}")]
public ActionResult GetCustomersByCity(string city) {
    //Your code here
}
[Route("All/{city?}")]
public ActionResult GetCustomers(string city) {
    //Your Code here
}
[Route("City/{city=Nagpur}")]
public ActionResult GetCitywiseCustomers(string city) {
    //Your code Here
}
```

And with that, we saw how the new Attribute Routing feature in ASP.NET MVC 5 provides more control over a URI in our web application.

FILTER OVERRIDES

In previous versions of MVC, in order to override a filter for a single action or controller, you had to apply a filter for each and every action and controller, one by one. MVC 5 introduces *Filter Overrides which is defined in the documentation as "You can now override which filters apply to a given action method or controller, by specifying an override filter. Override filters specify a set of filter types that should not run for a given scope (action or controller). This allows you to add global filters, but then exclude some from specific actions or controllers".*

Let's understand this better with an example. We know that Action filters in ASP.NET MVC define execution behaviour of the controller and/or action. Typically an Action filter such as Authorize restricts the access of a controller or action method for an unauthenticated user.

Step 1: Open VS 2013 and create an ASP.NET MVC 5 application. To this application, add a class file with the name DataClasses.cs in the Models folder:

```
using System.Collections.Generic;

namespace MVC5_External_Auth.Models {
public class Customer {
    public int CustId { get; set; }
    public string CustName { get; set; }
}

public class CustomerList : List<Customer> {
    public CustomerList()     {
        Add(new Customer() { CustId = 1, CustName =
        "C1" });
        Add(new Customer() { CustId = 2, CustName =
        "C2" });
    }
}

public class DataAccess {
    List<Customer> Customers;

    public List<Customer> GetCustomers()     {
        return new CustomerList();
    }
}
```

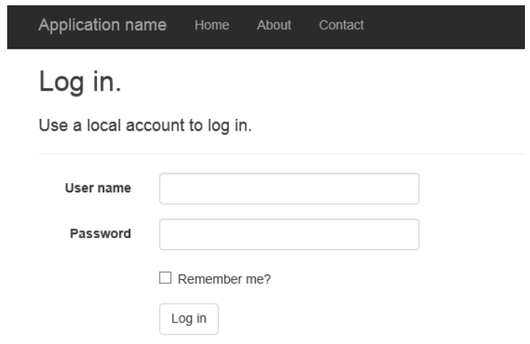
Step 2: Run the application. Create two users of name User1 and User 2 in the application using the Register View.

Step 3: Add a new empty controller in the controller folder with the name CustomerController. Add the following code in it:

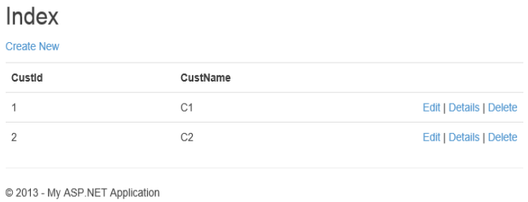
```
[Authorize(Users = "user1")]
public class CustomerController : Controller {
DataAccess objDs;

public CustomerController() {
    objDs = new DataAccess();
}

public ViewResult Details(int id) {
    var Cust = objDs.GetCustomers().Where(c => c.CustId
== id).First();
    return View(Cust);
}

//
// GET: /Customer/

public ActionResult Index() {
    var customers = objDs.GetCustomers();
    return View(customers);
}
}
```

Note that the CustomerController is applied with Authorize filter and it is authorized for user1. This means that the controller can be accessed only by user1.

Step 4: Run the application, and in the URL type http://localhost:4403/Customer/Index. A login page will be displayed:



Here enter the user name as user1 and its associated password and the customer details will be displayed as shown here:



In the above scenario, consider that you require authorization for all action methods in the *CustomerController*, except Details. This means that you have to apply the Authorize action filter on the *CustomerController*, but you want the Details action method to be an exception from using the Authorize filter. So to implement this, the *Details* method must override the filter applied on the Controller class. In ASP.NET MVC5, this is possible using *OverrideAuthorization*.

Step 5: In the application, add a new folder with the name *CustomActionFilterOverride*. Add a new class file of name *CustomActionFilterOverride.cs*. In the class, add the following code:

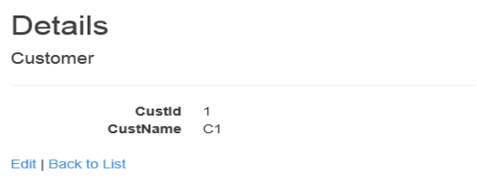
```
using System;
using System.Web.Mvc;
using System.Web.Mvc.Filters;

namespace MVC5_External_Auth.CustomActionFilterOverride
{
public class CustomOverrideAuthorizationAttribute :
FilterAttribute, IOverrideFilter {
    public Type FiltersToOverride {
        get
        {
            return typeof(IAuthorizationFilter);
        }
    }
}
}
```

The class *CustomOverrideAuthorizationAttribute* is inherited from the *FilterAttribute* class and implements *IOverrideFilter*. This interface is used to define the filters applied on the controller. The property *FiltersToOverride* returns the *IAuthorizationFilter* type. This means that Authorize filter applied on the parent (controller or Global application class) will be overridden.

Step 6: Now apply this filter on the Details method of the controller class as shown here:

```
[CustomOverrideAuthorization]
public ViewResult Details(int id) {
    var Cust = objDs.GetCustomers().Where(c => c.CustId
    == id).First();
    return View(Cust);
}
```

Now run the application, and in address bar type URL as http://localhost:4403/Customer/Details/1. The details for the customer with id as 1 will be displayed without redirecting us to the login page.



Please note that there is a bug in the Filter Override feature (https://aspnetwebstack.codeplex.com/workitem/1315) which has been fixed in ASP.NET MVC 5.1.

And with that, we wrap up this journey of some exciting and new features in ASP.NET MVC 5.

Conclusion:

ASP.NET MVC 5 is the latest version of the popular ASP.NET MVC technology that enables you to build dynamic websites using the Model-View-Controller technology, with an emphasis on a clean architecture, test-driven development and extensibility. This article gave you a quick overview of what's new in ASP.NET MVC version 5 and how best to apply these new features in your projects ■

Mahesh Sabnis is a Microsoft Certified Trainer (MCT) since 2005 and has conducted various Corporate Training programs for .NET Technologies (all versions). Follow him on twitter @maheshdotnet. Mahesh blogs regularly on Azure, SharePoint, Metro UI, MVC and other .NET Technologies at http://bit.ly/HsS2on

WHAT'S NEW IN .NET FRAMEWORK 4.5.1

> Microsoft .NET Framework 4.5.1 is an in-place *update* to .NET Framework 4.0 and the .NET Framework 4.5. This version can be run *side-by-side* with .NET 3.5 and earlier versions.

Released in October 2013, a couple of new features were introduced in .NET framework 4.5.1, as well as some improvements were implemented in the areas of Application Performance and Debugging applications.

Some of these features are listed here:

Debugging Feature Improvements

- 64-Bit Edit and Continue support in Visual Studio 2013
- Inspecting Method returns value while debugging using Autos Window as well as a pseudo variable $ReturnValue
- ADO.NET Connection Resiliency
- Async Debugging Enhancements

Application Performance Improvements

- ASP.NET Application Suspension
- On-Demand Large-Object heap compaction
- Multi-core JIT Improvements

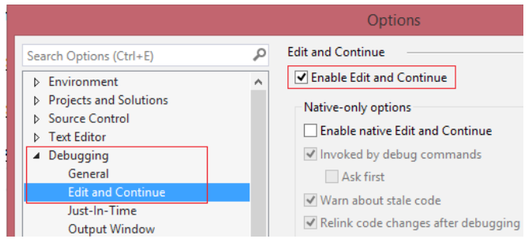
DEBUGGING FEATURE IMPROVEMENTS IN .NET FRAMEWORK 4.5.1

64-Bit Edit and Continue support in Visual Studio 2013

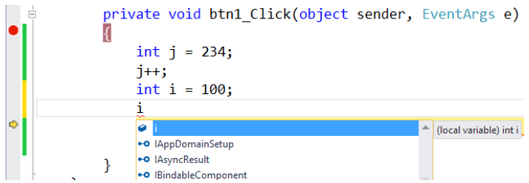
The debugging feature *Edit and Continue*, as our VB 6 developers will recall, was a popular feature which allowed developers to make changes to the code during debugging it. This feature allows user to fix bugs quickly without restarting the debugging session, which in turn expedites testing of various scenarios.

Versions prior to Visual Studio 2013 provided support for Edit and Continue for 32-bit targeted environment. If you tried Edit and Continue for a 64-bit targeted environment in a version prior to VS 2013, you would receive the following error -

In Visual Studio 2013, we now have support for Edit and Continue in 64-bit environments. This feature is by default ON for Windows Application as well as Web Application under Visual Studio 2013. You can cross check the same under debugging section by going to Tools > Options Menu.



When you use the Edit and Continue with 64-bit targeted environments, you can start editing the code while debugging it, as shown here -



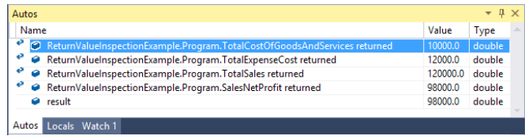
Inspecting Method Return value While Debugging

The next feature improvement which we will talk about is how to inspect a method return value in .NET 4.5.1. In Visual Studio 2013, while debugging, we can now check the return values of a function or of nested functions. The first way of checking return value is to make use of Autos window. Let's see an example of the same -

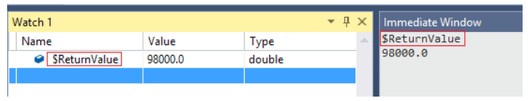
```
class Program
{
    static void Main(string[] args)
    {
        double result =
        SalesNetProfit(TotalCostOfGoodsAndServices(),
        TotalExpenseCost(), TotalSales());
    }

    private static double SalesNetProfit(double COGS,
    double Expense, double ActualSales)
    {
        return ActualSales - (COGS + Expense);
    }
    private static double TotalCostOfGoodsAndServices()
    {
        return 10000;
    }
    private static double TotalExpenseCost()
    {
        return 12000;
    }
```

```
private static double TotalSales()
    {
        return 120000;
    }
}
```

The above console application calls the *SalesNetProfit()* in the *Main* function. The *SalesNetProfit* function calls other functions to fetch parameter values. When you start debugging the application and execute the *SalesNetProfit()* function, open the Autos window and see the output. You will see that the return values of functions/nested functions are listed. This is pretty useful when you are not storing the result of a method call in a variable, e.g: when a method is used as a parameter or return value of another method.



Another way of finding the return values of a function is using a new pseudo variable "$ReturnValue". Just type $ReturnValue in the Immediate window (Ctrl + Alt + I) or a Watch window after you have stepped out of the method call. Shown here is the variable in action:



Entity Framework and ADO.NET Connection Resiliency

The Entity Framework and ADO.NET Connection Resiliency introduced in .NET Framework 4.5.1 recreates a broken connection and automatically tries to establish a connection with the database.

Any data centric application is crucial to a business and should be capable of displaying/fetching data from the remote servers, whenever required. When we design data centric applications, there are a large number of possibilities that needs to be thought of when the connection to the database goes idle/ broken.

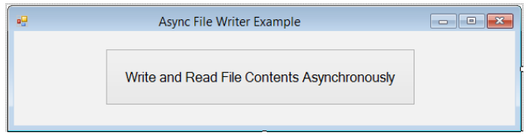
In such situations, you will have to build your own custom Connection Resiliency feature which will retry establishing broken connections. In .NET Framework 4.5.1, Microsoft has provided EF/ADO.NET Connection Resiliency feature out-of-box. What you have to do is target your application to the

use Framework 4.5.1 and this feature will be available to you. No configurations/APIs are needed when you work with this feature. This feature will automatically recreate broken connections and also retry transactions. Pretty cool!

Async Debugging Enhancements

Another feature improvement I want to highlight is Async Debugging Enhancements in Visual Studio 2013. In the .NET world, Async programming has gained momentum when *Task Parallel Library (TPL)* and keywords like *async* and *await* were introduced. For a developer, debugging is an important step during development to check how the code executes, proceeds and if it provides the desired results. We can make use of a Call Stack window for seeing these details.

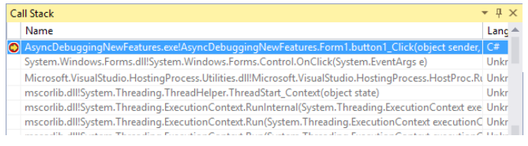
Let's write a program to test the Async Debugging Enhancements feature using Visual Studio 2013. Here's a sample screenshot of our program -



The button click event code looks like the following -

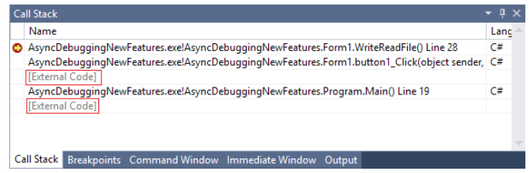
```
private async void button1_Click(object sender,
EventArgs e){
    var result = await WriteReadFile();
    MessageBox.Show(result);
}

private async Task<string> WriteReadFile(){
    StreamWriter SW = new StreamWriter(@"E:\test.txt",
    true);
    await SW.WriteLineAsync("WelCome To India");
    SW.Close();
    StreamReader SR = new StreamReader(@"E:\test.txt");
    var result=await SR.ReadToEndAsync();
    SR.Close();
    return result;
}
```

If you debug the application in Visual Studio 2012, you will see a similar output:



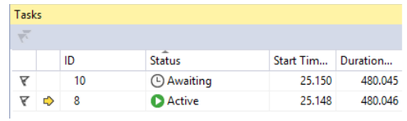
The call stack window shows a lot of code which you don't want see as it does not contain any logical starting point.

However if you debug the same code in Visual Studio 2013, you will observe that the Call Stack is much more cleaner as shown here -



The Call Stack window in Visual Studio 2013 has hidden External Code. Now we have a neat and a clean view of Call Stack window.

Also take a look at the Task window. It shows you all the active tasks, completed tasks, as well as scheduled tasks as shown here -



APPLICATION PERFORMANCE IMPROVEMENTS

ASP.NET App Suspend

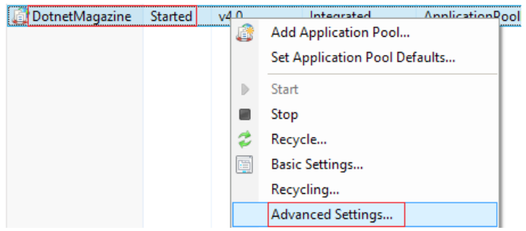
In .NET Framework 4.5.1, a new feature called *ASP.NET App Suspend* has been introduced for hosting an ASP.NET Web site. This feature is introduced in IIS 8.5 and Windows Server 2012 R2.

The basic concept is that all sites are in an *inactive* mode. When a site is requested, it is loaded into memory, the mode becomes active, and the site responds to page requests. When a site becomes idle (depending on the timeout setting), the site is put in a suspended state and the CPU resources and the memory it was using is made available for requests to other sites. As soon as a request for the site is made, it can be resumed very quickly and respond to traffic again.

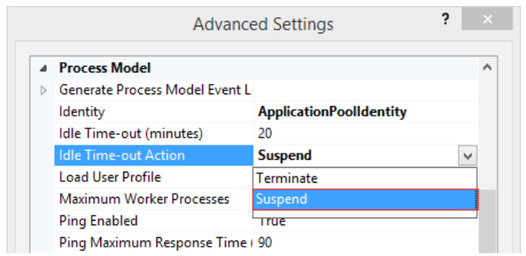
This new feature makes effective utilization of your hardware resources and increases the start-up time.

To use this feature, you don't have to learn any new API. This

feature is a simple configuration of Application Pool. Right click the Application Pool and click on Advance Settings as shown here -



In the Advance Settings window, go to Process Model section and expand the value of "Idle Time-out Action" property value. You will see two values. The default value is set to "Terminate". The other value which is available is "Suspend" as shown here -



By setting the Idle Time-out Action to suspend, when the Idle Time-out occurs, the ASP.NET web site gets suspended from CPU activities. It also gets paged to the disk in a "Ready to Go" state. Hence the start-up time is dramatically increased.

On-Demand Large-Object heap compaction

Another very exciting feature introduced in .NET 4.5.1 is the *On-Demand Large Object Heap Compaction*. Before the release of .NET Framework 4.5.1, the large object heap compaction was not available. Because of this fragmentation occurred over a period of time. When you use 32-bit systems, the processes running under them may throw an *OutOfMemory* exception in case your app is making use of large amount of memory, like large arrays. The On-demand large object heap compaction is now supported in .NET framework 4.5.1 to avoid the OutOfMemory exception. This feature is available as an on-demand feature, as it is expensive. The Garbage Collection pause time is more and hence when to use this feature needs to be decided based on a particular situation. For example if you are experiencing OutOfMemory exceptions.

You can set this feature using GCSettings which comes under System.Runtime namespace. The GCSetting provides a property LargeObjectHeapCompactionMode that has two values - Default and CompactOnce. The default value is *Default* which does not compact the Large Object Heap compaction during Garbage collection. If you assign the property a value of *CompactOnce*, the Large Object Heap is compacted during the next full blocking garbage collection, and the property value is reset to GCLargeObjectHeapCompactionMode.Default

```
GCSettings.LargeObjectHeapCompactionMode = GCLargeObjectHeapCompactionMode.CompactOnce;
GC.Collect();
```

MultiCore JIT Improvements

When you work with large applications, a common concern for developers is how to increase the start-up time of an application. Most of the developers make use of NGen.exe [Native Image Generator] to reduce the start-up time which JIT's your code at installation and not at start-up time. Now this gives us a flexibility to reduce the start-up time of our applications. However what in cases where we don't have installation scenarios? In such situations, you will not be able to take advantage of the NGen.exe tool.

MultiCore JT is a performance feature that can use be used to speed up an application launch time by 40-50%. The benefits can be realized both in a Desktop as well as in an ASP.NET application and server, as well as client apps. Multi-core JIT is automatically enabled for ASP.NET apps and is enabled on Assembly.LoadFrom and Appdomain.AssemblyResolve You can read more about it here.

For additional details on .NET 4.5.1 check the formal MSDN documentation.

So these were some new features and improvements in the latest release of the .NET Framework. I hope you will make use of them in your .NET apps ■

Pravinkumar works as a freelance corporate trainer and consultant on Microsoft Technologies. He is also passionate about technologies like SharePoint, ASP.NET, WCF. Pravin enjoys reading and writing technical articles. You can follow Pravinkumar on Twitter @pravindotnet and read his articles at bit.ly/pravindnc

Using the Right Soil

If there is one thing that's needed for a garden, it's the right soil. If the dirt has too many rocks or too much clay, it's not good for growing much. (Unless of course you're doing hydroponics.) So we begin our discussion of Software Gardening by discussing good soil, which in our case isn't a software development technique at all, but is dependent on project management.

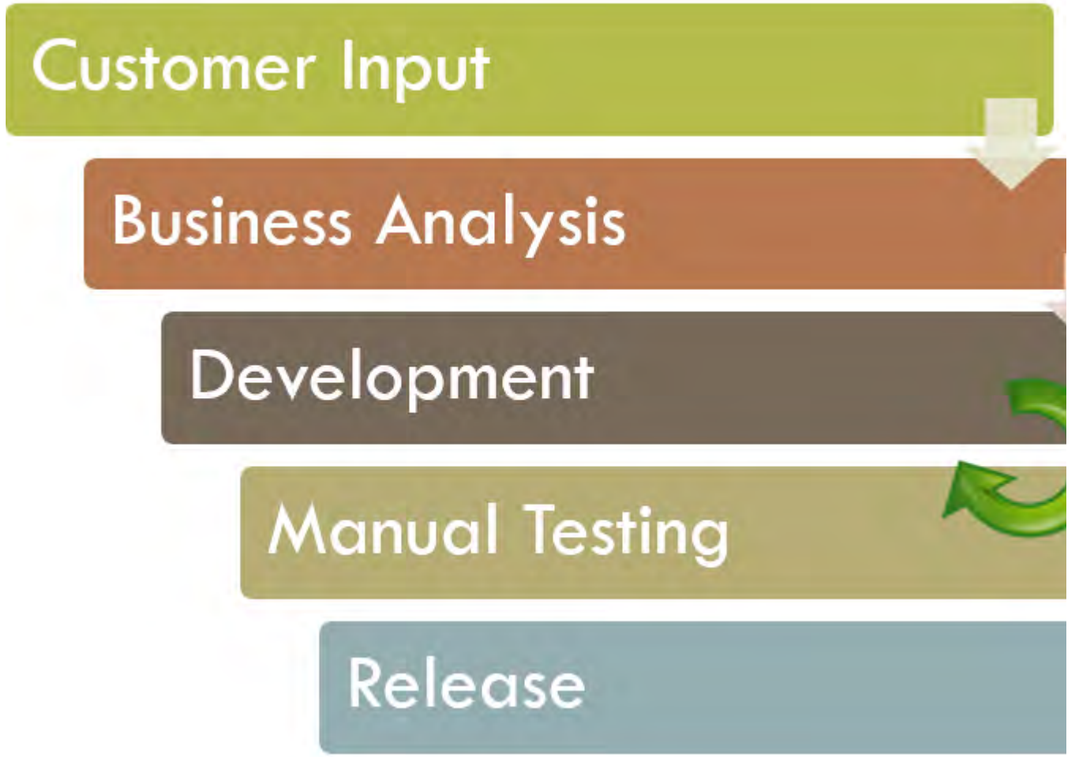
In the previous DNC Magazine issue (Issue 09 Nov 2013), I explained that software development, contrary to how many people portray it, *is not like constructing a building*. Let's continue down that discussion a bit more. When buildings are constructed, an architect or designer meets with the customer to discuss what they need and how they want it to look. They take notes of the meeting, then disappear and start work on blueprints, the plans for the building. Sometimes the land is already purchased so those blue prints include site plans. These are details of how the building will fit into the existing landscape. Other times, the land is purchased afterwards and then the site plans are added. Drawing blueprints could take weeks or months depending on the size of the building.

Once the blueprints are ready and approved by the customer, the project is put out for bid. A General Contractor runs the project and has several sub-contractors, plumbers, electricians, painters, carpet layers, etc. All get a copy of the blueprints and submit their cost and time estimates to the General Contractor, who rolls everything up and submits one overall cost and time estimate.

You might think that once the bid has been awarded that construction will start shortly afterwards. Sometimes it does, but government permits must be secured and the bigger the project, the longer that takes. Actual construction may begin with site work, which prepares the land. You may have to dig a hole for a foundation or in the case of an office tower, underground parking. Concrete is poured. For a small project, subfloors built and walls are framed, the base for the roof is built. On a large office tower, steel girders are erected and eventually each floor starts to take shape. Eventually the electrical, plumbing, and other subcontractors do their job. Along the way, government inspectors may check things over to make sure the work meets legal building codes.

During all this, the General Contractor is managing the project. Each step must be complete before the next begins. For example, the electrician and plumber can't come in before the subfloor is done and the walls studded out. The painter has to wait for sheetrock. The carpet layer has to wait for the painter, etc.

In the end, construction projects are generally managed using waterfall project management. It works for the construction industry, but is *death to a software project*. We can't wait for one step in the process to be completed before moving on to the next.



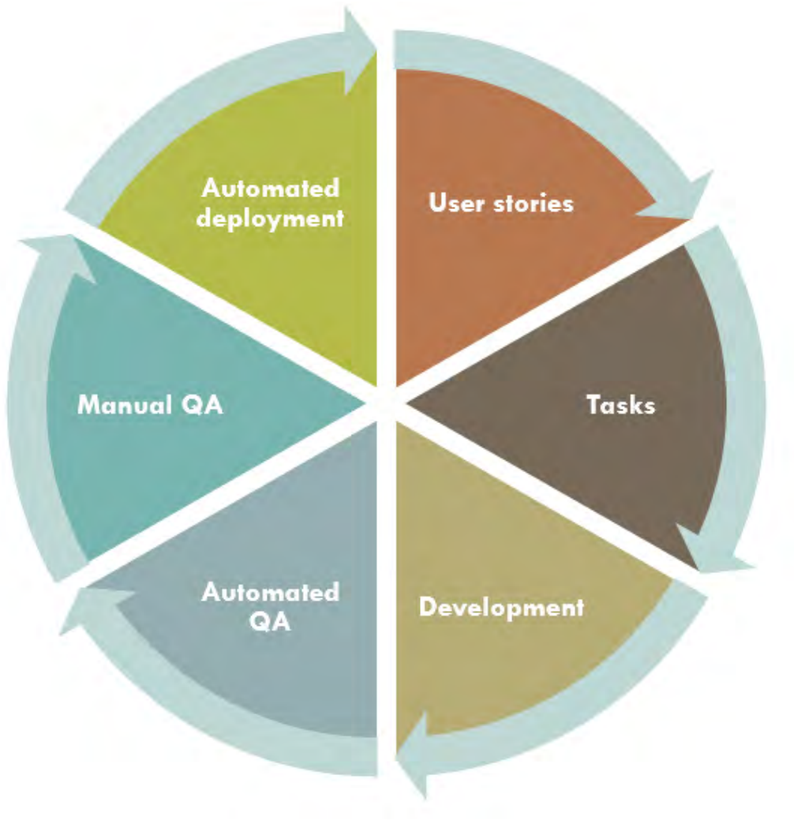
This graphic shows the typical Waterfall process when applied to software. Each step must be completed before you can fall down to the next step. Notice the circular arrow in the graph. Typically when using Waterfall, a programmer compiles the code then throws it over a wall for someone to manually test it. When the code fails (and it often does), it is thrown back to the programmer. This circular process continues for some time. I have seen this happen so often that it seems like an end-less loop of back-and-forth between the programmer and the QA person.

Here in the US, we're transitioning to new government mandated healthcare, commonly called *Obamacare*. Recently, the government run website, healthcare.gov, went online and was a disaster. The site couldn't handle the load put on it and frequently crashed. It couldn't communicate with other government sites. Not all insurance information was available to the user. It didn't have a good user experience at all. It was a disaster. This was a project that took years and hundreds of millions of dollars. The US Congress is holding hearings into the problem, trying to pin the blame on one individual.

But it's not the fault of any one person. It's the fault of the system that used waterfall methodologies to manage the project.

> *"If Waterfall is Bad Soil, Agile is Good Soil"*

By now, you know where I'm going with this. **The answer is to use Agile methodologies to manage the project**. If Waterfall is *bad soil*, Agile is *good soil*. With Agile, you do design, planning, coding, testing, and yes, even deployment, as one process. You don't complete analysis before design. You don't complete design before coding. You don't complete coding before testing. And you don't complete testing before deployment. Everything works together.



Agile has been around a long time, but as an industry, we pinpoint its birth to the signing and publication of the Agile Manifesto. The Agile Manifesto was written in February, 2001 when people that had competing ideas of how to deal with Waterfall, came together at Snowbird, Utah to ski and talk about common ideas. In the end, four main concepts were common with each competing idea.

- **Individuals and Interactions over processes and tools**. This says that people are more important over how we develop and what tools we use to develop the software.

- **Working software over comprehensive documentation.** Don't spend all your time writing documentation that becomes outdated before you need it. Deliver real, working software, and do it often.

- **Customer collaboration over contract negotiation**. Don't waste time trying to define every little thing in the contract. Keep the customer informed on what you're doing. This is generally done daily in a stand-up. Then at the most, every two weeks in a sprint review, where you show off the work you did in the two weeks before and actually deliver it to the user, so they can test it.

- **Responding to change over following a plan**. Business needs change. Prepare for this change. Embrace it. Accept it.

There are many Agile methodologies you can choose from. *Scrum* seems to be the most widely accepted methodology. No matter which methodology you choose, there are some common things that run through most Agile methodologies.

First, Agile teams are *self-organizing*. This means that they take it upon themselves to figure out who will do which tasks. There is often no difference between developer, QA, and other workers. They are all team members and are working to the same goal: release at the end of the Sprint. Typically, work is done in a series of sprints, generally one or two weeks long. At the beginning of the sprint, the business owner of the project presents the team with a list of tasks to complete during the sprint. Each team member picks a task to work on. The ones not picked are returned to the backlog. If someone finishes their task, they pick another task. Tasks are small units of work that make up user stories. User stories describe how a user sees a particular function.

Each morning, the team meets in a stand-up or scrum that shouldn't last longer than about 15 minutes. Each person working on tasks answers three questions. 1) What did you do yesterday? 2) What will you do today? 3) Is there anything stopping you from getting stuff done today? If you're using Scrum, there is a Scrum Master on the team. His job is to remove things blocking work from getting done. He is not a project manager in the traditional sense. He doesn't manage the project. That's done by the team with input from the business owner.

There is no throwing the code over the wall. ***An important part of Agile is good Continuous Delivery techniques where software is automatically compiled and tested***. Integration tests should also be automated as should deployments (you do test deployments, don't you?). I'll have much more to say about Continuous Deployment in future columns.

At the end of the sprint, there is a Retrospective. In this meeting, the new working features are demonstrated to the Business Owner. A discussion about what's right and wrong in the Sprint is also an important part of the meeting. Also, the working software is released, possibly to production.

Now that you've had a very high-level introduction to Agile, you may be wondering if it will really work. Let's circle back to the healthcare.gov web site. After the failed release, three software engineers in California took it upon themselves to do something right. They looked at some of the healthcare.gov code to see how to do access the database, integrate with other sites, etc and over a period of a couple of weeks, working nights and weekends, came up with thehealthcaresherpa.com. While is doesn't have all the functionality of healthcare.gov, but it has much of it and it has a better user experience.

But what about taking things the other way? Can we apply Agile to construction? Turns out, we can. Prior to the 2002 Winter Olympics in Salt Lake City, the primary north-south freeway, Interstate 15 (I-15), through the entire Salt Lake valley was rebuilt. Every bridge, overpass, on-ramp, off-ramp, everything about this 17 mile stretch of freeway was rebuilt. In some cases, the interchange to other freeways needed to be redesigned too. Using standard, waterfall methodologies, the project would have taken over ten years. They had to do it in about four years. The people working on the project designed a small part of the new freeway and started construction on it while the next section was being designed. ***They called this "Design-Build". We call it Agile. And it worked.*** The project came in both under budget and under schedule.

So, you want to bring Agile to your team. How do you do it? First, you need some buy-in from management. This is needed because when the big boss says, "How far along are you", he generally wants to hear "we're 60% done". He won't hear that. He'll hear, "We've completed 60% of our story points". That doesn't mean you're 60% done. But keep in mind that the team is self-organizing.

You need to pick an Agile process. Scrum, and Xtreme Programming (XP) are two favorites but there are others. (Note that Kanban is technically a release management technique, not project management, but many teams find success using it.) Pick one that will work well for your environment. If you frequently need to jump between projects, Scrum may not be for you.

Once you have your system, go all in. Get an expert to guide you. Get the proper training. Far too often I hear of teams that read a couple of books and adopt a couple of Agile ideas, then fail and blame it on Agile, saying that it doesn't work. *If you're going to do it, do it right.*

And finally, keep in mind the importance of Good Soil. By using the right soil and applying software gardening techniques, your applications can be lush, green, and vibrant ■

Craig Berntson is the Chief Software Gardener at Mojo Software Worx, a consultancy that specializes in helping teams get better. He has spoken at developer events across the US, Canada, and Europe for over 20 years. He is the co-author of "Continuous Integration in .NET" available from Manning. Craig has been a Microsoft MVP since 1996. Email: craig@mojosoftwareworx.com, Blog: www.craigberntson.com/blog, Twitter: @craigber. Craig lives in Salt Lake City, Utah.

What's New in ASP.NET Web API 2.0

In its simplest form, a Web API is an API over the web (HTTP). ASP.NET Web API is a framework that allows you to build Web API's, i.e. HTTP-based services on top of the .NET Framework, using a convention based and similar programming model, as that of ASP.NET MVC. These services can then be used in a broad range of clients, browsers and mobile devices.

Two versions of the ASP.NET Web API framework have been released so far, with Web API 2.0 being the latest one. In this article, we will explore some of the new features introduced in ASP.NET Web API 2.0.

If you have never worked with ASP.NET Web API, I recommend you read this article.

When we think about exposing data on the web, we talk about four common operations which we use –

1. CREATE
2. RETRIEVE
3. UPDATE
4. DELETE

We call these operations as CRUD operations. HTTP Services provide 4 basic HTTP verbs which we can map to our CRUD operations, as described here –

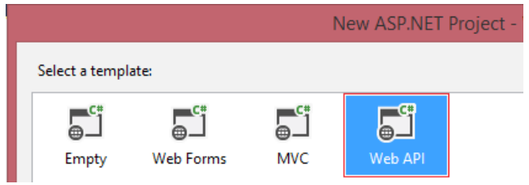
1. POST – CREATE
2. GET – RETRIEVE
3. PUT – UPDATE
4. DELETE – DELETE

The idea is by using HTTP Services if you can connect to the web, then any application can consume your data. When the data is pulled or pushed by using HTTP Service, the data is always serialized using JSON (JavaScript Object Notation) or XML.

Well how does the ASP.NET Web API fit here? As already mentioned, ASP.NET Web API allows you to build HTTP Services on top of the .NET Framework. In version 2.0, the Web API framework has been enhanced to support the following features:

1. IHttpActionResult return type
2. A new Routing Attribute
3. Support for Cross-Origin requests using CORS
4. Securing ASP.NET Web API using OAuth 2.0
5. Support for $expand, $select in OData Service

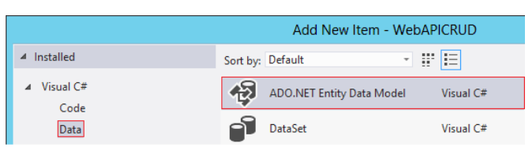
We will shortly start looking at these new features but first, let's create a new Web API application. I am using Visual Studio 2013 and SQL Server 2012 for this demonstration.



Once your project gets created successfully, we will now create a table in our SQL Server database on which we will perform CRUD operations. The table script is as shown below -

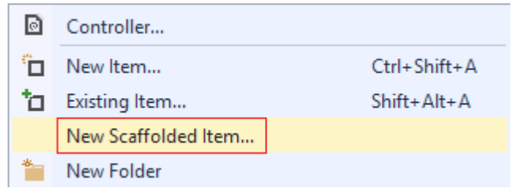
```
USE PurchaseOrderDB
GO
CREATE TABLE Customers
(
    CustomerID NVARCHAR(10) PRIMARY KEY,
    ContactName NVARCHAR(50),
    City NVARCHAR(50)
)
```

Now insert some dummy data into your table which we can fetch later. Once your table is ready, we will add a new item in our Web API Application by right clicking Web Application in Solution Explorer and add a new item which is "ADO.NET Entity Data Model" with the name "PurchaseOrder" as shown here –

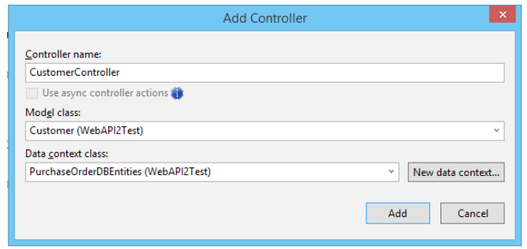


Follow the wizard steps. Choose "Generate from database" > and then choose the connection string for your database. If the connection string is not available, create one and click on the Next button. In the next step, choose the *Customers* table from the Tables section and click on the *Finish* button.

It's time to implement our logic which will perform CRUD operations using ASP.NET Web API. For this implementation, we will start with the new feature of *scaffolding*. Right click the controller and choose "New Scaffolded Item" as shown here -



In the next step, choose the controller template. We will choose "Web API 2 Controller Actions, using Entity Framework" and name our controller as *CustomerController*. Choose a Model class "Customer" and Data Source "PurchaseOrderDBEntities" as shown below -



Click on the Add button. Now observe the code generated by the new scaffolding option. It looks similar to the following -

```
public class CustomerController : ApiController {
private PurchaseOrderDBEntities db = new
PurchaseOrderDBEntities();

// GET api/Customer
public IQueryable<Customer> GetCustomers() {
    return db.Customers;
}

// GET api/Customer/5
[ResponseType(typeof(Customer))]
public IHttpActionResult GetCustomer(string id) {
    Customer customer = db.Customers.Find(id);
    if (customer == null)
    {
        return NotFound();
    }
```

```
    return Ok(customer);
}

// PUT api/Customer/5
public IHttpActionResult PutCustomer(string id,
Customer customer)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    if (id != customer.CustomerID)
    {
        return BadRequest();
    }

    db.Entry(customer).State = EntityState.Modified;

    try
    {
        db.SaveChanges();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!CustomerExists(id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }

    return StatusCode(HttpStatusCode.NoContent);
}

// POST api/Customer
[ResponseType(typeof(Customer))]
public IHttpActionResult PostCustomer(Customer
customer)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    db.Customers.Add(customer);

    try
    {
        db.SaveChanges();
    }
    catch (DbUpdateException)
    {
        if (CustomerExists(customer.CustomerID))
        {
            return Conflict();
        }
        else
        {
            throw;
        }
    }

    return CreatedAtRoute("DefaultApi", new { id =
     customer.CustomerID }, customer);
}

// DELETE api/Customer/5
[ResponseType(typeof(Customer))]
public IHttpActionResult DeleteCustomer(string id)
{
    Customer customer = db.Customers.Find(id);
    if (customer == null)
    {
        return NotFound();
    }

    db.Customers.Remove(customer);
    db.SaveChanges();

    return Ok(customer);
}
}
```

IHTTPACTIONRESULT RETURN TYPE

The first change which you will observe is that the controllers have a return type of *IHttpActionResult*. In the previous version, we would return an object or set the return type to *void* for methods like Post, Put and Delete. In these situations, Web API converted the same to *HttpResponseMessage*. In ASP.NET Web API 2.0, now we have one more option which is implementing *IHttpActionResult*. It helps you to construct and format your response message by giving instructions which you want the Web API to follow. It also provides a couple of helper methods like -

- NotFound() - When you want to return a 404 error
- Ok() - Generates the result with the specified object
- Conflict() - Creates a 409 Conflict result

A NEW ROUTING ATTRIBUTE

ASP.NET Web API 2.0 now supports routing configuration at Web API method level or at the Controller level using a *Route* attribute. The question is why would I use the new [Route] attribute? Earlier, in Web API 1.0, we configured routing at a central location. For example, we would configure routing in the Global.asax Application_Start event as shown below -

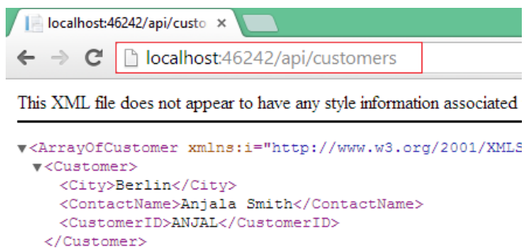
```
protected void Application_Start(object sender, EventArgs e)
        {
            RouteTable.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id =
System.Web.Http.RouteParameter.Optional }
            );
        }
```

This technique of routing was the default option while working with Web API 1.0 and works perfectly fine when you have simple routing rules. But when it comes to complex routing and custom routing, this approach becomes hard to configure. Also when it comes to applying constraints on Routes, you usually end up with writing multiple routes.

Instead of writing routes in the Register method or Application_Start method, we can define the route on a controller's method. Change your GetCustomers() method in the controller using [Route] attribute -

```
[Route("api/customers")]
public IQueryable<Customer> GetCustomers() {
    return db.Customers;
}
```

The output of the above route will look like the following



If you are eager to see how this differs from V 1.0, here's a comparison:

```
config.Routes.MapHttpRoute(          Web API V 1.0
    name: "Customers",
    routeTemplate: "api/customers/{id}",
    defaults: new { id = RouteParameter.Optional });
```

```
                                     Web API V 2.0
[Route("api/customers/{id}",Name="Customers")]
0 references
public IHttpActionResult GetCustomer(string id)
{
```

Microsoft has defined the routing configuration on a Web API method with a single line syntax. It also allows us to configure various options while using [Route] attribute. For example, check the following options.

Routing with Constraints - Earlier we use to implement a separate class for constraints which you would have to then configure with the routes. But now using the [Route] attribute, you can define basic constraints like -

```
[Route("api/customers/{customerid:alpha}")]
                OR
[Route("api/customers/{customerid:int}")]
```

The above route will match the customerid parameter in lower case or upper case alphabets characters.

Default Values with Route - We can also specify default values for the routes as shown below -

```
[Route("api/customers/{contactno=9665622460}")]
```

Route with Optional Values – Similarly we can also configure route with optional values as shown below -

```
[Route("api/customers/{contactno?}")]
```

Route Names - Names can be provided to our routes as shown below -

```
[Route("api/customers",Name="Customers")]
```

As you might have observed, the [Route] attribute has brought a bunch of new flexibilities in Web API 2.0 and is very helpful to ease the Web API development in today's era of HTTP Service programming.

RoutePrefix

When you perform routing, your route prefix will always be common. For example, in our case "api/customers". You can use [RoutePrefix] attribute at a Controller level as shown below -

```
[RoutePrefix("api/customers")]
0 references
public class CustomerController : ApiController
{
    private PurchaseOrderDBEntities db = new PurchaseOrderDBEntities();

    [Route("")]
    0 references
    public IQueryable<Customer> GetCustomers()
    {
```

Test the [RoutePrefix] attribute and you will see the same

output. In some cases, you may want to override the default route prefix, in which case you can make use of [~] operator to do so. For example -

```
[Route("~/myroute/customers")]
0 references
public IQueryable<Customer> GetCustomers()
{
    return db.Customers;
}
```

You can also define optional parameters with route using [?]. But if you are using optional parameters, then you will have to provide the default value to the parameter. There are some other options as well which you can use with [Route] attribute like Route Order and Route Names. Try them out to see what they do.

CROSS ORIGIN RESOURCE SHARING [CORS] -

Another new feature introduced in ASP.NET Web API 2.0 is Cross Origin Resource Sharing [CORS]. Let's talk about why we should think about using this new feature. AJAX request to HTTP services can be made only from URLs of the same Origin. For example if your web services are hosted on http://myservices:8080/api/ then the browser will allow applications hosted on http://myservices:8080/ to make AJAX calls to these services. But services/web applications hosted on http://myservices:8081/ or any other port for that matter, cannot access the HTTP resource/api over AJAX. This limitation is the default browser behaviour for security reasons.

However, W3C has a CORS spec that outlines the 'rules and regulations' to enable Cross Origin Resource Sharing aka CORS. ASP.NET MVP Brock Allen built CORS support for Thinktecture, which was later pulled into ASP.NET Web API core and is now available in Web API 2.0.

We will see Cross Origin Resource Sharing (CORS) feature with a simple example. Let's add a new project using another Visual Studio instance which will be an Empty ASP.NET Application. This application will try to fetch data from our Web API which we just built a short while ago.

Once your Web Application is ready, add an HTML page and design the page as shown here -

```
<!DOCTYPE html>
<html lang=”en”>
<head>
    <title>CORS Example</title>
</head>
<body>
    <div id=”XMLCustomers”></div>
</body>
</html>
```

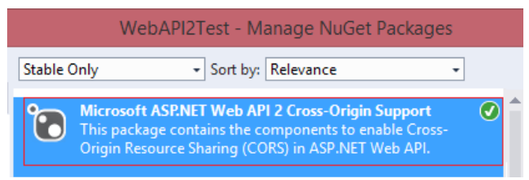
Add the jQuery library using NuGet package and then add a reference to the jQuery library in your HTML page. Write the following code in our HTML page -

```
<script src=”Scripts/jquery-2.0.3.min.js”></script>
<script>
$(function () {
        var webAPIUri = “http://localhost:50008/api/
        customers”;
        $.ajax({
                url: webAPIUri
        }).done(function (data) {

                $(‘#XMLCustomers’).text($.
                parseJSON(data));
        }).error(function (jqXHR, textStatus,
        errorThrown) {
                $(‘#XMLCustomers’).text(jqXHR.response
                Text || textStatus);
        });
});
</script>
```

If your run your application, you will receive an error. You can check this in the Developers tool of a browser of your choice.

```
XMLHttpRequest cannot load http://localhost:46242/api/
customers. No ‘Access-Control-Allow-Origin’ header is
present on the requested resource.
Origin ‘http://localhost:48995’ is therefore not
allowed access.
```

So to access this service, we have to enable CORS in our Web API. We will add the CORS Package using NuGet package as shown below -



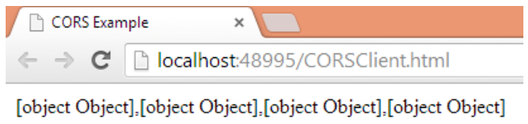
Once you install CORS, we will open the WebApiConfig class which is located in the App_Start folder and enable CORS in the Register method as shown here -

```
public static void Register(HttpConfiguration config)
{
    // Web API configuration and services
    config.EnableCors();
```

We will now add [EnableCors] attribute in our *CustomersController* as shown here -

```
[EnableCors(origins: "http://localhost:48995",headers:"*",methods:"*")]
0 references
public class CustomerController : ApiController
{
    private PurchaseOrderDBEntities db = new PurchaseOrderDBEntities();
```

To test the client, we will run the application and now you should be able to access data as shown here -



You can enable CORS at the Action level or in all Controllers in your Web API Application as well. You can also write a custom CORS policy for better control on your Web APIs.

WEB API AUTHENTICATION USING OAUTH

The next feature we will discuss is authenticating Web API 2.0 using an External Authentication Service like Facebook, Google, Twitter or Microsoft Account.

What is OpenID and OAuth?

We will start our discussion by first understanding what is OpenID and OAuth. I have taken this explanation from DotNetCurry's author Sumit's article on Real World OAuth which I feel is perfect to explain OAuth and OpenID.

At a 100K feet level, both are Authentication protocols that work towards a Single Sign-On model. Basic premise being, instead of having to remember a different username and password for every application we visit on the internet, we have a dedicated Authentication provider with whom we can register our user name and password. Applications (aka Relying Party) redirect us to this provider for authentication and after a successful authentication; the provider passes back a token to the replying party who initiated the request. The target Application then uses the token to verify the identity of the person and provide further access to Application Features.

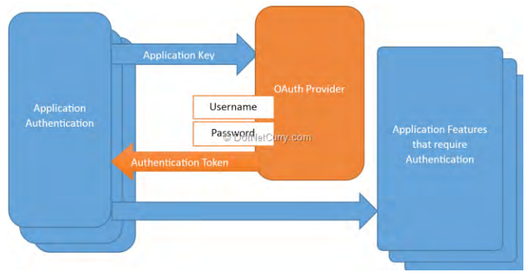
OAuth was initiated by Twitter when they were working on building their OpenID implementation for Twitter API. Their need was more towards controlling what features to make available based on Authentication Types offered at the Provider and those sought by the relying party.

For example when a Relying party signs up for Twitter Authentication access, they can request for a Read-Only connection (this gives them read permissions to a user's stream), a Read/Write connection (this gives them read and write permissions on a user's stream) and a Read/Write with Direct Messages connection (giving them maximum possible access to user's data). Thus, when a user authenticates with Twitter over OAuth, they are told exactly what kind of access they are providing.

On the other hand, OpenID simply ensures that you are who you claim to be by verifying your username and password. It has no way of controlling feature access.

In even simpler terms and for the sake of explanation, OpenID is about Authentication, OAuth is about Authentication and Authorization (for features at the Provider).

Shown here is an architecture of External Authentication using OAuth 2.0 -



The diagram above demonstrates the Authentication process for an OAuth provider. The overall workflow is as follows:

1. Application (Relying Party) registers with OAuth provider and obtains an Application specific key (secret). This is a one-time process done offline (not shown in above image).

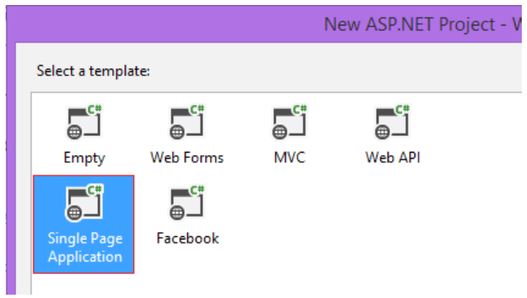
2. Next the Application (Relying Party) has to pass the Application key to the provider every time it intends to authenticate someone.

3. User of the Relying Party needs to be registered with the

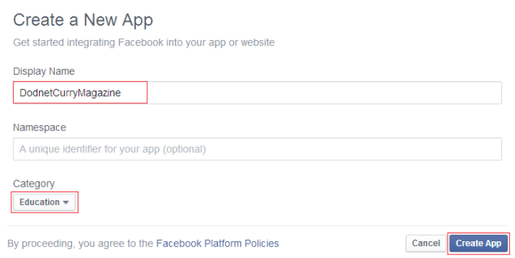
Provider (again a one-time offline process)

4. Every authentication request directs the user to the Auth Provider's site. The user enters the Username and Password that they obtained by creating an account with the provider. The Provider verifies the account credentials, then it matches the Application Key and provides a token back to the Relying Party with which only the Authorized actions can be performed.

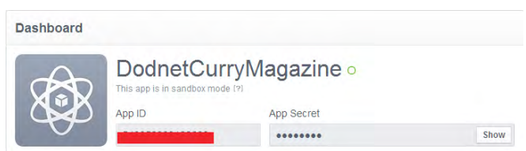
With this OpenID and OAuth basics cleared, let's configure a Facebook account to authenticate our Web API using OAuth. To start with, we will create an ASP.NET SPA [Single Page Application] application as shown below -



To use Facebook authentication, we will have to create a Facebook developer account. Then you will have to grab the Application ID and secret key. So, let's login to " https://developers.facebook.com/". Then create a Facebook app as shown below -

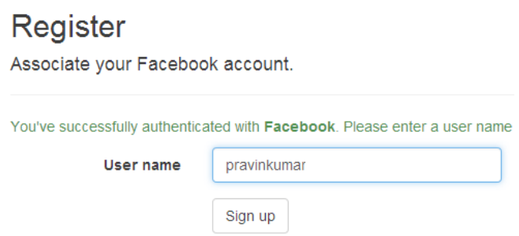


Copy the Application ID and Secret Key in a notepad from the Dashboard as shown below -



Go back to Visual Studio and open Startup.Auth.cs file. Go to *ConfigureAuth* method and uncomment the "UseFacebookAuthentication" method. Copy the App ID and Secret ID from the notepad. After performing this step, go to Settings and click on the Add Platform button. Choose Web Site and paste the URL of your site and Save the changes.

Now run your application. You should see a Local Account to login as well as a Facebook login at right hand side. Click on the Facebook button. It will now ask you to login with your Facebook account. Once you have successfully logged in, you will see a page similar to the following -



Here you can choose your User Name to login and once authenticated, you will see your Home page.

$SELECT AND $EXPAND OPERATOR SUPPORT IN ODATA SERVICES -

ASP.NET Web API 2.0 has expanded the support for OData by introducing $select and $expand query operators for querying the data. Let's talk about how these operators will help us in querying.

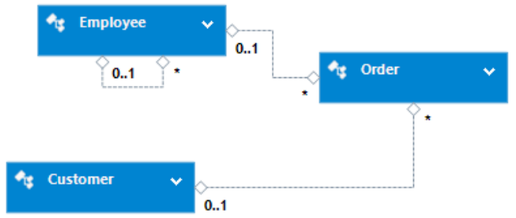
1. $select Operator - This operator is used to select subset of properties. For example if we are querying the Customer Entity, we can use $select operator to fetch only required properties like CustomerID, ContactName and City.

2. $expand Operator - This operator allows us to select the related entities, with the entity being retrieved. For example, while retrieving the Customer entity we can also retrieve the related Orders placed by the Customers. Often, you will use $expand operator with $select operator.

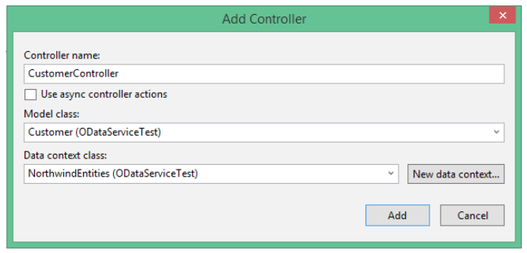
To test the $expand, $select in OData services, we will create a new web application. First we will install the OData package using Package Manager Console. Use the following command to install the OData package -

```
Install-Package Microsoft.AspNet.WebApi.OData
```

Once the package is installed successfully, add a "ADO.NET Entity Data Model" and connect it with Northwind database. Also add three tables, Customers, Employees and Orders. The model should look like the following -



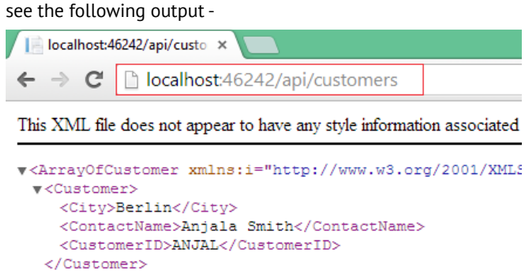
After adding the ADO.NET Data Model, we will add a Scaffold Item by choosing "Web API 2 OData Controller with actions, using Entity Framework" and configure the same as shown here



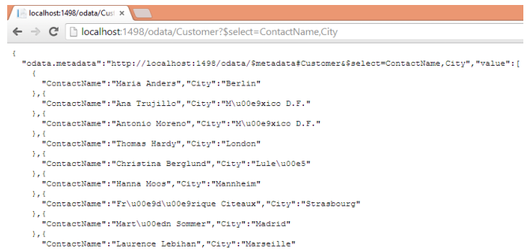
The CustomerController contains the code for the CRUD operations. Similarly create controllers for Employees and Orders as well. Open "WebApiConfig.cs" file from App_Start folder. Import a namespace "using System.Web.Http.OData.Builder" and add the following code in the Register method -

```
1 reference
public static void Register(HttpConfiguration config)
{
    ODataConventionModelBuilder builder = new ODataConventionModelBuilder();
    builder.EntitySet<Customer>("Customer");
    builder.EntitySet<Order>("Order");
    config.Routes.MapODataRoute("odata", "odata", builder.GetEdmModel());
    // Web API configuration and services
    config.EnableQuerySupport();
```

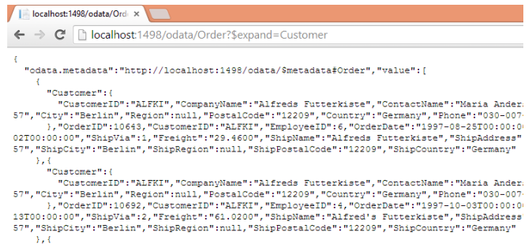
It is time to test our Web API OData Service. Type the url localhost:[portnumber]/odata/Customer in a browser. You will see the following output -



We will now test the new features of Web API 2 OData service. First option is how to use $select. $select allows us to select subset of properties. For example, instead of selecting all the columns of Customers table, let's select some of them as shown here -



$expand allows us to select related entities to be selected as shown below -



And that's it!

Conclusion

In this article, we have seen a couple of new features introduced in ASP.NET Web API 2.0 which includes new Scaffolding options, IHttpActionResult return type, [Route] attribute, CORS support, securing Web API using External Authentication with OAuth service and using the new operators $select and $expand while querying OData services.

Make sure you use these new features to convert your Web API 1.0 code to 2.0 ■

Pravinkumar works as a freelance corporate trainer and consultant on Microsoft Technologies. He is also passionate about technologies like SharePoint, ASP.NET, WCF. Pravin enjoys reading and writing technical articles. You can follow Pravinkumar on Twitter @pravindotnet and read his articles at bit.ly/pravindnc

UP AND RUNNING WITH ANGULARJS & ASP.NET MVC

WHAT IS ANGULAR JS?

Angular JS is (yet another) client side MVC framework in JavaScript that has caught the imagination of the web world in recent times. It was created by a group of developers at Google when they realized that their project (Feedback) had become too unwieldy and needed a cleanup. The three weeks effort at that cleanup lay seed to what is now known as *AngularJS*. Of course AngularJS was thrown open to community and since then has garnered a vibrant community and a boat load of features.

Among things that make AngularJS popular is its focus on testability by having principles of *Dependency Injection* and *Inversion of control(IoC)* built into the framework.

Today we will look at AngularJS in a plain vanilla ASP.NET MVC app. We'll start with an empty project and go ground up. We will start with basic data binding and retrieving of data, then walk through other core Angular features like directives and routing.

BASICS OF DATABINDING

For this article, we could have done the entire code walkthrough without any server side component and some hardcoded client side data, but that's too far away from the real world. So we'll get some data from the ethereal data stream, that is Twitter, and display it in our application demonstrating the concept of *model binding* and *directives* in AngularJS.

To connect to Twitter, we'll use the excellent *Linq2Twitter* library on the server side. Once we have logged in, we'll retrieve and display data using AngularJS. So before we get AngularJS, lets scaffold an ASP.NET MVC app and hook it up with LinqToTwitter. As with all Twitter apps, we have to get a CustomerKey and a CustomerSecret from dev.twitter.com

SCAFFOLDING THE ASP.NET MVC APP

1. We create a MVC 4 project with the Empty template as we want to get started with the bare bones today.

```
PM> install-package linqtotwitter
```

2. We add LinqToTwitter from Nuget

```
PM> install-package linqtotwitter
```

3. We install Twitter Bootstrap CSS and JavaScript files using the following command

```
PM> install-package Twitter.Bootstrap
```

4. We add an Empty HomeController in the Controllers folder. The Index action method calls the Authorization function and if the current user is not Authorized, it redirects to Twitter's Authorization page. Once Authorized, it navigates back to the Index page.

Very briefly, the ConsumerKey and the ConsumerSecret that we got while creating Twitter app is in the AppSettings. These two along with the OAuth token returned by Twitter, complete the Authorization part.

Note: *We have not taken any measure to hide our Auth token once it returns from the Server. In a production application, make sure not to leave it unprotected.*

```
public class HomeController : Controller {
 private IOAuthCredentials credentials = new
SessionStateCredentials();
private MvcAuthorizer auth;
private TwitterContext twitterCtx;
public ActionResult Index()  {
 var unAuthorized = Authorize();
 if (unAuthorized == null)   {
  return View("Index");
 }
 else   {
  return unAuthorized;
 }
}

private ActionResult Authorize()  {
 if (credentials.ConsumerKey == null || credentials.
  ConsumerSecret == null)   {
  credentials.ConsumerKey = ConfigurationManager.
       AppSettings["twitterConsumerKey"];
  credentials.ConsumerSecret = ConfigurationManager.
       AppSettings["twitterConsumerSecret"];
 }
 auth = new MvcAuthorizer   {
  Credentials = credentials
 };
 auth.CompleteAuthorization(Request.Url);
 if (!auth.IsAuthorized)   {
  Uri specialUri = new Uri(Request.Url.ToString());
  return auth.BeginAuthorization(specialUri);
 }
 ViewBag.User = auth.Credentials.ScreenName;
 return null;
```

MVC5 Notes: *Due to changes in ASP.NET Security Policies, DotNetOpenAuth (used internally by Linq2Twitter) breaks, so we have to use a workaround as defined here. The code accompanying this article incorporates this workaround. However since this is not related to the core topic, I have skipped the details.*

5. Finally we add a method that returns a JsonResult containing list of Latest Tweets from the logged in user.

```
[HttpGet]

public JsonResult GetTweets()
{
 Authorize();
 string screenName = ViewBag.User;
 IEnumerable<TweetViewModel> friendTweets = new
 List<TweetViewModel>();
 if (string.IsNullOrEmpty(screenName))
{
     return Json(friendTweets,
     JsonRequestBehavior.AllowGet);
 }
 twitterCtx = new TwitterContext(auth);
 friendTweets =
 (from tweet in twitterCtx.Status
  where tweet.Type == StatusType.Home &&
        tweet.ScreenName == screenName &&
        tweet.IncludeEntities == true
  select new TweetViewModel
{
      ImageUrl = tweet.User.ProfileImageUrl,
      ScreenName = tweet.User.Identifier.
      ScreenName,
      MediaUrl = GetTweetMediaUrl(tweet),
      Tweet = tweet.Text
 })
 .ToList();
 return Json(friendTweets, JsonRequestBehavior.
 AllowGet);
}

private string GetTweetMediaUrl(Status status)
{
 if (status.Entities != null && status.Entities.
       MediaEntities.Count > 0)  {
  return status.Entities.MediaEntities[0].
       MediaUrlHttps;
 }
 return "";
}
```

6. The ViewModel used to encapsulate a Tweet is as follows:

```
public class TweetViewModel
{
   public string ImageUrl { get; set; }
   public string ScreenName { get; set; }
   public string MediaUrl { get; set; }
   public string Tweet { get; set; }
}
```

7. Next we setup the View skeleton using BootStrap (you need to do this if you are using VS2012 or older). VS2013 and MVC5 project template will automatically setup the _Layout to use BootStrap):

a. Add _ViewStart.cshtml in the Views folder. It's content is as follows:

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

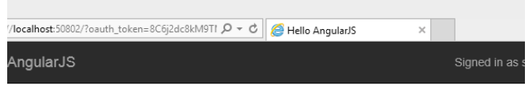
b. Add Views\Shared_Layout.cshtml folder that will serve as our Master Page.

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8" />
 <meta name="viewport" content="width=device-width" />
 <title>Hello AngularJS</title>
 <link href="~/Content/bootstrap.css" rel="stylesheet"
/>
 <link href="~/Content/bootstrap.min.css" rel="
        stylesheet" />
</head>
<body>
 <div class="navbar navbar-inverse navbar-fixed-top">
  <div class="container">
   <div class="navbar-header">
    <button type="button" class="navbar-toggle"
        data-toggle="collapse" data-target=".
        navbar-collapse">
     <span class="icon-bar"></span>
     <span class="icon-bar"></span>
     <span class="icon-bar"></span>
    </button>
    @Html.ActionLink("Hello AngularJS", "Index",
        "Home", null, new { @class = "navbar-brand" })
   </div>
   <div class="navbar-collapse collapse">
    <ul class="nav navbar-nav"></ul>
     @if (ViewBag.User != null)
     {
      <p class="navbar-text navbar-right">Signed in as
        <a href="#" class="navbar-link">@ViewBag.User</
        a></p>
     }
    </div>
   </div>
  </div>
  <div class="container body-content">
   <div class="container-fluid">
    <div class="row-fluid">
     @RenderBody()
    </div>
   </div>
  </div>
</html>
```

c. Finally we add Index.cshtml which only has a Title for now

```
@{
    ViewBag.Title = "Hello AngularJS";
}
```

d. If we run the application at this point, it will ask to Authorize our app with Twitter, and once authorized, we'll be redirected to the home page as follows:



That completes our ground work and we are ready to dive into AngularJS now.

INTRODUCING ANGULARJS

Thanks to the community, AngularJS is easy to get started with, using Nuget.

```
PM> install-package angularjs
```

Once installed, we add a reference to it in _Layout.cshtml

```
<script src="~/Scripts/angular.js"></script>
```


THE ANGULAR APP

First point of difference from libraries like KnockoutJS is that Angular needs an ng-app attribute to be declared, wherever we want the Scope of Angular to start. We will add it in our Index.cshtml by adding a as follows.

```
<div ng-app>
 <!-- More Goodness Coming -->
</div>
```

THE ANGULARJS CLIENT SIDE CONTROLLER AND VIEWMODEL

We will add a hello-angular.js in the Scripts folder. We start off with a Controller that is defined as follows:

```
var indexController = function ($scope)
{

}
```

In Angular, the indexController object is regarded as Controller simply if someone pulls in the ($scope) parameters.

What is $scope?

$scope is the client side version of ViewBag as we know from MVC. It is used for sharing data between the Angular Controller and the View.

How do we use $scope?

1. As we mentioned above, we use Scope as a ViewBag, so to start off with, we'll add a JSON array in it.

```
var indexController = function ($scope)
{
 $scope.tweets = [
  { screenName: "Sumit", tweetText: "Test 1" },
  { screenName: "Sumit", tweetText: "Test 2" }];
}
```

Using Scope and Data Binding in AngularJS

2. Next we go back to Index.cshtml and add the following markup.

```
 <!-- More Goodness Coming -->
  <div ng-controller="indexController">
   <ul>
    <li ng-repeat="item in tweets">
     {{item.screenName}}, {{item.tweetText}}
    </li>
   </ul>
  </div>
```

Introducing Angular Directives

There is a lot of stuff going on here.

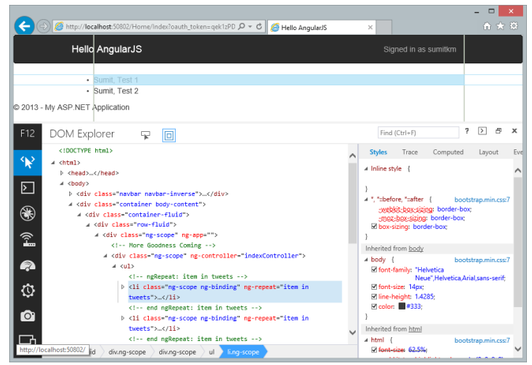
a. In the outermost div, we have specified the directive ng-controller="indexController". This specifies what is the scope of the $scope object.

b. Note the <ul> doesn't have anything special, so data binding to collections is by default container less.

c. The <li> element finally defines the ng-repeat directive and sets it to tweets (as defined in $scope earlier). Note the syntax is like C#'s foreach.

d. Finally we have the {{ ... }} handlebar syntax to define the placeholders.

Those who are familiar with KO's data binding will note how Angular doesn't use data- attributes, rather it uses AngularJS Specific attributes that it modifies at runtime. If we run this app now, we'll see the following:



We can see how the DOM has been manipulated at runtime to get the output.

Just to recap, the $scope that was passed into the controller was actually instantiated and passed by Angular. Those familiar with constructor injection will realize that Angular actually injected the $scope without us having to worry about it.
This is the uniqueness and strength of Angular.

The Pluralizer Directive

We have been using the term *Directives* without explaining it so far. Directives in AngularJS are special keywords that have built in functionality. They can apply to elements attributes, classes or comments. We put in a 'Pluralizer' directive to see the total number of tweets.

In the view, we add the following markup inside the scope of the ng-controller

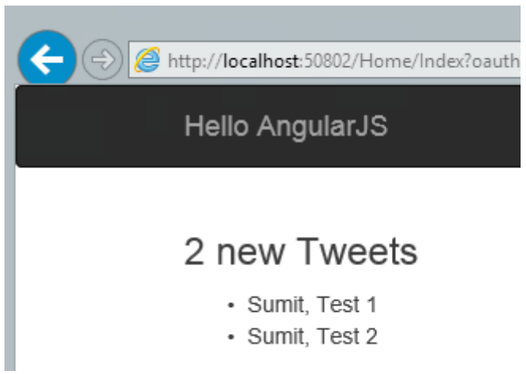
```
<h3>
 <ng-pluralize count="tweets.length"
 when="newTweets"></ng-pluralize>
</h3>
```

The ng-pluralize directive can be read as 'check for the count in tweets.length and based on it, evaluate the switch case in the *newTweets* function.' Actually newTweets is an object in our indexController that we define as follows:

```
$scope.newTweets = {
 0: "No new Tweets",
 other: "{} new Tweets"
}
```

So now our ng-pluralize delegate reads like – "check for the count in tweets.length and if there are no tweets put the text "No new Tweets" else put the text 'n new Tweets' where n is the count".

If we run the application as is, we'll get the following:



Nice. Now let's move on and see how we can get data from the Server.

Promises and Fetching Data from Server

Angular makes it really easy to get data from the server using AJAX. We'll update our client controller (hello-angular.js) as follows:

```
var indexController = function ($scope, $http)
{
    var resultPromise = $http.get("/Home/GetTweets");
    resultPromise.success(function (data)
    {
        $scope.tweets = data;
    });


    $scope.newTweets = {
        0: "No new Tweets",
        other: "{} new Tweets"
    }
}
```

Notice the additional parameter $http, getting injected into the controller. This is a wrapper for http functions as we can guess. We will use this to retrieve the Promise return by the $http.get(...) call. A promise is like a delegate or callback used when doing Async AJAX calls. Note we are calling the GetTweets method on our server controller. Once the Promise (delegate) completes successfully, we get the actual JSON data from the server.

Let's update the Index.cshtml to create a nice list of the Tweets

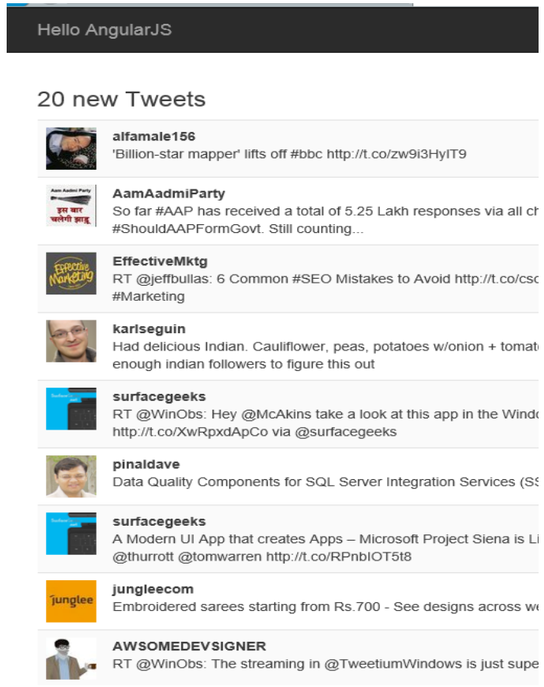
```
<!-- More Goodness Coming -->
<div ng-controller="indexController">
 <h3>
  <ng-pluralize count="tweets.length"
  when="newTweets"></ng-pluralize>
 </h3>
 <table class="table table-striped">
  <tr ng-repeat="item in tweets">
   <td>
    <img src="{{item.ImageUrl}}" />
   </td>
   <td>
    <strong>{{item.ScreenName}}</strong> <br />
    {{item.Tweet}}
   </td>
  </tr>
 </table>
</div>
```

There are a couple of new things to note.

a. We have moved the ng-repeat from the <li> to the <tr> tag.

b. We have added an <img> element and notice how the src is bound in-place to the ImageUrl property in our ViewMode.

Well with that, we are all set. Run the Application and a view like the following comes up!



Pretty neat eh!

For devs familiar with data-binding in any form (no pun intended), be it WebForms, WPF, Silverlight etc. , Angular's binding philosophies match closely except for the fact that Angular's magic happens in the browser.

MODULES AND SERVICES IN ANGULARJS

MODULES IN ANGULARJS

Modules are a way in AngularJS to organize and structure code. It is in a way, similar to Namespaces in C#, but Modules are more than passive naming constructs.

Declaring a Module

The syntax for creating a Module is as follows:

```
// Create ngTwitter Module (roughly Module = namespace
in C#)
var ngTwitter = angular.module("ngTwitter", []);
```

Notice the empty array parameter. This is actually an array of resources (we'll see what are resources shortly), however if you don't have any resources to pass, you need to pass an empty array. Missing out on the empty array may result in 'undesired' results.

As mentioned, Modules are like namespaces, so to add your Controller to the Module instead of having the controller floating around in JavaScript like in our previous example, we update the Controller to be a function in our module as follows:

```
var ngTwitter = angular.module("ngTwitter", []);

ngTwitter.controller("TimelineController", function
($scope, $http) {
 var resultPromise = $http.get("/Home/GetTweets");
 resultPromise.success(function (data) {
  $scope.tweets = data;
 });
 $scope.newTweets = {
  0: "No new Tweets",
  other: "{} new Tweets"
 }
});
```

Note, for better context, I've renamed the controller from IndexController to TimelineController because it is essentially showing Twitter's Timeline.

Using the Modules in your View

Now that the Module has been declared, our Controller is no longer 'visible' to the Angular directly. So if we run the application as is, we'll get errors on our Developer Console saying the Controller was not found.

To hookup the Module, we need to specify it in the ng-app attribute. This tells Angular which module to use for a given section on our page.

Since we changed the name of the Controller, we go ahead and change that as well. Our updated markup is as follows:

```
<div ng-app=”ngTwitter”>
    <!-- More Goodness Coming -->
    <div ng-controller=”TimelineController”>
        <h3>
            <ng-pluralize count=”tweets.length”
                when=”newTweets”></ng-pluralize>
        </h3>
        <table class=”table table-striped”>
            <tr ng-repeat=”item in tweets”>
                <td>
                    <img src=”{{item.ImageUrl}}” />
                </td>
                <td>
                    <strong>{{item.ScreenName}}
                    </strong> <br />
                    {{item.Tweet}}
                </td>
            </tr>
        </table>
    </div>
</div>
```

If we run the app now, things should work fine.

Next we look at defining a service which is again, more of code restructuring than a new framework feature.

YOUR FIRST ANGULARJS SERVICE

Services in AngularJS are simply reusable bits of code aiming to help make your app linear and more testable.

Declaring a Service

You can create Services in Angular using either the factory method or the service method provided by Angular. The Factory method returns a simple object, whereas the Service method returns a function. To start off with, we'll use the Factory method and declare our service as follows:

```
ngTwitter.factory(“TwitterService”, function ($http) {
 return {
  timeline : function()
  {
   return $http.get(“/Home/GetTweets”);
  }
 }
});
```

Notice that we have passed $http service into our TwitterService and Angular injects it for us at runtime.

Using a Service

To use it, we simply 'ask' for it in our Controller function and Angular injects it for us. Now instead of calling the $http service directly, we call our TwitterService and request for the timeline. This still returns a *Promise* so we continue to use the success completion method to bind our tweets.

```
ngTwitter.controller(“TimelineController”, function
($scope, TwitterService) {
 var resultPromise = TwitterService.timeline();
 //var resultPromise = $http.get(“/Home/GetTweets”);
 resultPromise.success(function (data) {
     $scope.tweets = data;
 });

 $scope.newTweets = {
     0: “No new Tweets”,
     other: “{} new Tweets”
 }
});
```

Neat! Another bit of code restructuring making our code more granular as we go along.

THE ANGULAR $RESOURCE PROVIDER

The Angular Resource provider is a plugin that can be used to further abstract and modularize our code. But first up, the Resource provider is in a separate JavaScript file angular-resource.js. So we've to add a reference to it to our markup.

```
45      <script src="~/Scripts/jquery-1.10.2.min.js">
46      <script src="~/Scripts/bootstrap.min.js"></sc
47      <script src="~/Scripts/angular.js"></script>
48      <script src="~/Scripts/angular-resource.js"><
```

Using the $Resource Provider

Next we tell our Module that it needs to use the Resource provider. We do that by passing ngResource, the name of the provider, in the array or resources that we had left empty earlier.

```
var ngTwitter = angular.module(“ngTwitter”,
[‘ngResource’]);
```

Next we update our service to use the resource provider instead of $http directly.

```
ngTwitter.factory(“TwitterService”, function
($resource) {
    return {
        timeline: $resource(“/Home/GetTweets”)
    };
});
```

This change may confuse you. We were using $http.get(…) and now we are directly using $resource(…). How does that work? Well, from the Description provided in the documentation of AngularJS - A factory which creates a resource object that lets you interact with RESTful server-side data sources.The returned resource object has action methods which provide high-level behaviors without the need to interact with the low level $http service.

So basically it is a helper around the $http service and internally provides Get, Save, Query, Delete functions.

We'll use the query method in our controller to get our timeline back. Notice now we no longer have to wait for a *Promise* to return. The trick here is Angular returns an empty *Promise* when the query is fired, and when it returns, it internally updates the original promise with array of objects that were returned. Since we are using data binding, the UI updates itself appropriately.

```
ngTwitter.controller(“TimelineController”, function
($scope, TwitterService) {
    $scope.tweets = TwitterService.timeline.query({},
    isArray = true);

    $scope.newTweets = {
        0: “No new Tweets”,
        other: “{} new Tweets”
    }
});
```

Run the Application and we see a familiar response with a set of 20 latest tweets from our timeline.
That wraps up our code re-structuring. The next step involves using the Resource provider to perform other actions like POSTing data to our server to send new Tweets.

POSTING DATA

Before we get into *Directives*, let's first add another functionality in our little Twitter Reader. The ability to send Tweets. So far we have used the $resource service's query function. As we know, the resource function can also do GET, POST, PUT etc. Following is the map of actions to the HTTP verbs.

```
{ ‘get’:    {method:’GET’},
  ‘save’:   {method:’POST’},
  ‘query’:  {method:’GET’, isArray:true},
  ‘remove’: {method:’DELETE’},
  ‘delete’: {method:’DELETE’} };
```

(Source: Angular JS Documentation)

So to post data, we need to simply call the save function on our resource instance. However there is a slight gotcha. The $resource object expects all the functions to post to the same URL.

In MVC terms, this means posting to the same action method with different HttpGet/HttpPost/HttpPut etc attributes.

In our code, currently the HomeController uses *GetTweets* method to get the Data. Now posting to a URL called GetTweets is semantically icky. So let's update the Controller method to be called just Tweet. So we will do GET, POST, PUT etc. requests to the url /Home/Tweet.

Next we update our *TwitterService* to access the new URL

```
ngTwitter.factory(“TwitterService”, function ($resource)
{
    return {
        timeline: $resource(“/Home/Tweet”)
    }
});
```

Now if we run the Application, we should get the same UI where the latest 20 tweets are shown on screen.

ADDING A TWEET FIELD

To send out Tweets, we need an input box and a Tweet button. To do this, we add the following markup to the Index.cshtml

```
<div>
 <textarea ng-model=”statusText” rows=”5” />
 <button class=”btn btn-success”
  ng-click=”sendStatus()”>Tweet</button>
</div>
```

This adds a multiline textbox and a button to the top of the page. Note that we have added a view model element called *statusText* to contain the text or the status that we'll be sending to Twitter, and a click handler *sendStatus*.

ADDING THE SENDSTATUS METHOD TO THE $SCOPE

With our UI updated, we can add the sendStatus() event handler to our $scope as follows

```
$scope.sendStatus = function ()
{
 var tweetText = $scope.statusText;
 var newTimeLine = new TwitterService.timeline(
  {
   tweet: tweetText
  });
 newTimeLine.$save();
}
```

In the method, we are fetching the text from the *statusText* into *tweetText* variable. Next we are retrieving an instance of the timeline *$resource* and saving it in var *newTimeLine*

Note we are instantiating it with a JSON object with the status text in a property called *tweet*.

Finally we are calling *newTimeLine.$save()* method that will do an HTTP Post to a Tweet method in our Home Controller.

ADDING A TWEET METHOD TO HANDLE POST

Our client is ready to post Data. Let's complete our controller to accept it.

```
[HttpPost]
public JsonResult Tweet(string tweet)
{
 Authorize();
 twitterCtx = new TwitterContext(auth);
 try
 {
  Status stat = twitterCtx.UpdateStatus(tweet);
  if (stat != null)
  {
   return Json(new { success = true });
  }
  else
  {
   return Json(new { success = false, errorMessage =
   "Unknown Error" });
  }
 }
 catch (Exception ex)
 {
  return Json(new { success = false, errorMessage=
ex.Message });
 }
}
```

As seen above, the method accepts a string parameter called *tweet*. Note that it is same as the name of JSON property we initialized our service with.

We check if our AuthToken is still valid via the *Authorize()* helper method. Next we initialize the TwitterContext class provided by Linq2Twitter. Finally we call *UpdateStatus* method on the *TwitterContext* to send the Tweet.

If all goes well, we get a non-null Status and send back a JSON with 'success' property set to true. If there are any exceptions, we handle the exception and send a success = false and an error message.

UPDATING CLIENT TO NOTIFY SAVE STATUS

So far we have not used the return data after posting the Tweet. Let's update our client to do that. You can pass a callback method to *$resource.save(...)* that will retrieve the data returned as well as the HTTP headers. Using the returned data's success property, we can determine if the Tweet was sent successfully or not. We display an alert appropriately and if successful, we update the Text area to empty.

```
$scope.sendStatus = function ()
{
 var tweetText = $scope.statusText;
 var newTimeLine = new TwitterService.timeline(
 {
  Tweet: tweetText
 });
 newTimeLine.$save(function (data, headers)
 {
  if (data.success && data.success == true)
  {
   alert("Tweet Sent Successfully!");
   $scope.statusText = "";
  }
  else
  {
   alert("ERROR: " + data.errorMessage);
  }
 });
}
```

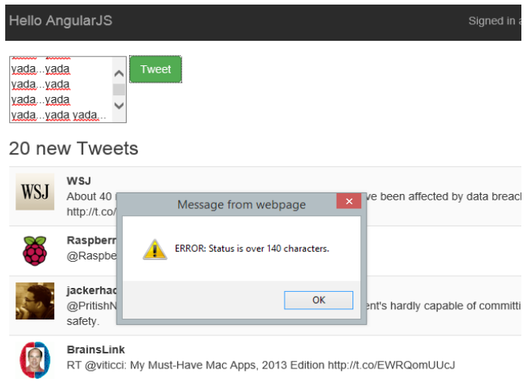

That wraps up the code changes. Time to take it for a spin.

Posting Data using AngularJS (Sending a Tweet)

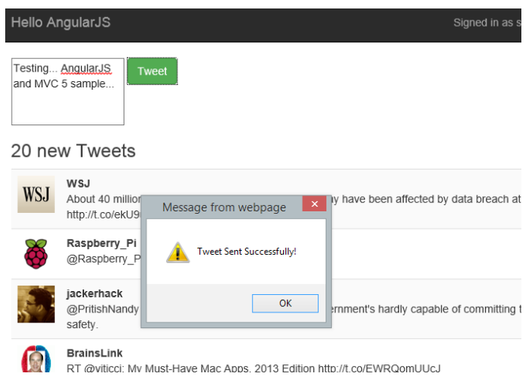
Step 1: On launch we are redirected to the Twitter page to login.

Step 2: Once we authenticate and log in, we'll see the top 20 tweets for the account.

Step 3: First let's try to send a Tweet longer than 140 characters. Boom! Error as expected.

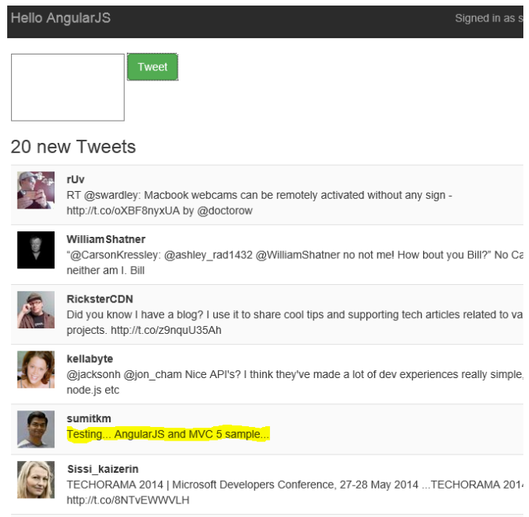


Step 4: Now let's try to send a legit tweet!



Nice! Successfully sent! When you click OK, the text area is cleared out.

Step 5: Hit refresh!



Bingo! There is our Tweet!

With that we complete this small step in Angular. We were able to use the $resource service to post data. In our case, we posted it all the way up to Twitter. But instead of Twitter, we could have easily posted it to a DB if we wanted to.

Customary Note of Caution: The Twitter Authentication method used here is NOT production ready!

CUSTOM ANGULARJS DIRECTIVES

So far, we have seen how we could get started by building a small Twitter Client. We have explored the view model, Modules and Services in Angular JS and how to post data using the $resource Service. We have also used the 'ng-pluralize' directive. The *ng-app* attribute that we used to define the scope of our Angular App is in fact a Directive, because there are no HTML5 attributes by that name! It's Angular who interprets the attribute at runtime. It is time to dive deep into AngularJS Directives now.

Apart from helping add custom attributes, directives can also be used to create the server side equivalent of 'tag-libraries' on the client. Those familiar with ASP.NET WebForms or JSP development will remember you could create server side components with custom Tags like <asp:GridView>... </asp:GridView> where the GridView rendering logic was encapsulated in a server component. Well, Directives allow you build such components, but on the client.

In fact, next, we'll define a 'Retweet' button on our Tweet Reader that will enable us to encapsulate the function in a custom directive.

ADDING THE RETWEET FUNCTIONALITY

Adding a Retweet button is rather simple to do. All you have to do is update the markup and hook it up to a function call JavaScript. We could then use the *$http resource* and Post it to a *Retweet* action method.

The markup for this would be as follows, the new bits are highlighted.

```
<table class="table table-striped">
 <tr ng-repeat="item in tweets">
  <td>
   <img src="{{item.ImageUrl}}" />
  </td>
  <td>
   <div>
    <strong>{{item.ScreenName}}</strong>
    <br />
    {{item.Tweet}}
   </div>
   <div>
    <button class="btn btn-mini" ng-
    click="retweet(item)"><i class="icon-retweet"></i>
    Retweet</button>
   </div>
  </td>
 </tr>
</table>
```

Model and Controller changes

LinqToTwitter's Retweet API uses the Status ID that Twitter generated to do the Retweet. We have not saved StatusId in our TweetViewModel.cs so first we'll update that:

```
public class TweetViewModel
{
 public string ImageUrl { get; set; }
 public string ScreenName { get; set; }
 public string MediaUrl { get; set; }
 public string Tweet { get; set; }
 public string Id { get; set; }
}
```

Next we update the Tweet() HttpGet function in the Controller to save the Id in the Json that will be sent over to the client

```
friendTweets =
 (from tweet in twitterCtx.Status
  where tweet.Type == StatusType.Home &&
   tweet.ScreenName == screenName &&
   tweet.IncludeEntities == true
  select new TweetViewModel
  {
   ImageUrl = tweet.User.ProfileImageUrl,
   ScreenName = tweet.User.Identifier.ScreenName,
   MediaUrl = GetTweetMediaUrl(tweet),
   Tweet = tweet.Text,
   Id = tweet.StatusID
  })
  .ToList();
```

Next we add the Retweet Action method

```
[HttpPost]
public JsonResult Retweet(string id)
{
 Authorize();
 twitterCtx = new TwitterContext(auth);
 try
 {
  Status stat = twitterCtx.Retweet(id);
  if (stat != null)
  {
   return Json(new { success = true });
  }
  else
  {
   return Json(new { success = false, errorMessage =
"Unknown Error" });
  }
 }
 catch (Exception ex)
 {
  return Json(new { success = false, errorMessage =
  ex.Message });
 }
}
```

Finally we add the *retweet* function to the *$scope* as follows.

```
$scope.retweet = function (item) {
 var resultPromise = $http.post("/Home/Retweet/",
item);
 resultPromise.success(function (data) {
  if (data.success) {
   alert("Retweeted successfully");
  }
  else {
   alert("ERROR: Retweeted failed! " + data.
errorMessage);
  }
 });
```


Well if we run the application now, things will just work fine and Retweet will work. This was doing things the HTML/ JavaScript way. Time to get *Directives* into the picture.

CREATING YOUR OWN DIRECTIVE

The idea behind creating a directive is to encapsulate the 'Retweet' functionality on the client side. For this example, this may seem a little counterproductive because the 'Retweet' functionality is rather simple and re-using it, involves only a bit of markup copy-paste.

But imagine for more involved UI Component like say a Tabbed UI functionality. Having a compact directive that renders the tabbed interface would really be a welcome relief.

In our case, we'll simply create a new element called <retweet-button>...</retweet-button>.

Defining the Directive

To do this, we first copy the current markup from the Index.cshtml and replace it with the following:

```
<retweet-button></retweet-button>
```

Next we simply add the following snippet to our hello-angular.js file.

```
ngTwitter.directive("retweetButton", function ()
{
 return {
  restrict : "E",
  template : "<button class='btn btn-mini' ng-
  click='retweet(item)'><i class='icon-retweet'></i>
  Retweet</button>"
 }
});
```

There are a few key things to note in the above snippet and how it 'links' to the custom element we used in the html markup.

1. The *ngTwitter.directive* method's first parameter is the name of your directive. Note the naming convention, the '-' in retweet-button was removed and the next character 'b' was made upper case (camel case) and the markup attribute 'retweet-button' became the directive name 'retweetButton'. Remember this convention else your directive will not be hooked up correctly and you'll end up with invalid HTML.

2. Next, the directive method takes an anonymous function as parameter which returns an object. In our case the object has two properties

a. The 'restrict' property tells Angular what type of directive it is. In our case it's an element, hence the "E". If you omit it, default is attribute and things will break.

b. The 'template' property contains a piece of the DOM (not string). As we can see, we have simply copy pasted the button element from our cshtml file. This works, but it has the action method, the text both hard coded and as we know, hard-coding makes components 'brittle' and less extensible.

Before we go further, you can run the application at this point and things will still work.

'Transclusion' and generalizing the Button's label

First thing we want to do is to be able to have Text that we want, instead of hard-coding it in our Directive. To do this we can add the text in our markup as follows

```
<retweet-button>RT</retweet-button>
```

Then we'll add a 'transclude' property in our directive's return object and set it to true. Finally we'll add the ng-transclude attribute in the template as shown below.

```
ngTwitter.directive("retweetButton", function ()
{
 return {
  restrict: "E",
  transclude: true,
  template : "<button class='btn btn-mini' ng-
  click='retweet(item)' ng-transclude><i class='icon-
  retweet'></i> </button>"
 }
});
```

If we use our browser tools, we'll see that the actual button has been placed inside the <retweet-button> element. This is a little icky and can be easily fixed by using the 'replace' property in our directive's return object.

```
ngTwitter.directive("retweetButton", function ()
{
 return {
  restrict: "E",
  replace: true,
```

```
  transclude: true,
  template : "<button class='btn btn-mini' ng-
  click='retweet(item)' ng-transclude><i class='icon-
  retweet'></i> </button>"
 }
});
```

This tells angular to replace the directive with the Template. Now that we have generalized the text, what if we wanted to pass in the action method name too? *Transcluding* doesn't help there.

Using 'Link Function' in our Directive

As mentioned earlier, hardcoding the click event to an action method reduces the modularity and makes our 'component' brittle. One way to 'not' hardcode is to use the Linking function while creating our Directive. The linking function is defined via the Link attribute.

We update our directive definition as follows:

```
ngTwitter.directive("retweetButton", function ()
{
 return {
  restrict: "E",
  replace: true,
  transclude: true,
  template: "<button class='btn btn-xs'
ng-transclude></button>",
  link: function (scope, el, atts)
  {
   el.on("click", function ()
   {
    scope.retweet(scope.item);
   });
  }
 }
});
```

If we run the app now,we'll still get routed to the *retweet* method. How is this working? Well the scope parameter of the *input* function passes the current *$scope* instance, the *el* parameter passes a jQuery object of the html element and *atts* element contains an array of attributes that we may have added to our directive markup. Currently we haven't used *atts* but we'll see it shortly. However the *el* element is a disaster waiting to happen. Given a jQuery object, we can simply go back to doing things the jQuery way like injecting the template etc. etc. In this case we've attached the click handler.

Well this works, but we can better this too and 'angularize' it further.

The 'scope' Definition for a Directive

We can add another property in the return object for Directive definition. This is the scope property. It is different from $scope as in, scope simply helps configure shortcuts that can be used in our template. For example, remember Angular uses the {{ ... }} syntax for *templating* fields. So instead of using *transclude*, we can use the {{ ... }} expression. But what will be the template property to bind to? We can define it in scope as follows (don't forget to remove the transclude: true property and the ng-transclude attribute).

```
ngTwitter.directive("retweetButton", function ()
{
 return {
  restrict: "E",
  replace: true,
  scope: {
   text: "@"
  },
  template: <button class='btn btn-xs' ng-
click='clickevent()'><span class='glyphicon glyphicon-
retweet'></span> {{text}}</button>"
 }
});
```

So what's happening here? We've defined a scope property that has an object with the property text whose value is set to @. This is actually a 'shorthand' telling Angular to pick up the 'text' attribute from the markup's list of attributes. Next, we have used the {{ text }} templating notation to tell angular that it should replace the value passed in via the scope->text in the template. The corresponding change in the markup is as follows:

```
<retweet-button text="Retweet"></retweet-button>
```

Okay, so this generalized the text in yet another way. How do we generalize the function call? Well we'll use another shortcut in the scope for that. We update the directive script as follows

```
first: ngTwitter.directive("retweetButton", function ()
{
 return {
  restrict: "E",
  replace: true,
  scope: {
   text: "@",
   clickevent: "&"
  },
  template: "<button class='btn btn-xs' ng-
  click='clickevent()'><span class='glyphicon
  glyphicon-retweet'></span>{{text}}</button>"
 };
});
```

Yes, we have used another magic symbol *&* for getting hold of a 'function pointer' called *clickevent*. In our template, we are invoking the function that this pointer is pointing to. But where did we assign the function? In our view of course

```
<retweet-button
text="Retweet"clickevent="retweet(item)">
</retweet-button>
```

Finally we have extracted the action method we are supposed to call on *$scope* out from the directive into the markup making the directive pretty generic.

In hindsight, it seems like a lot of work to reduce.

```
<button class="btn btn-mini" ng-click="retweet(item)"><i
class="icon-retweet"></i> Retweet</button>
```

TO

```
<retweet-button text="Retweet"
clickevent="retweet(item)"></retweet-button>
```

But as I said, we could potentially encapsulate the functionality of say sending a tweet or rendering a timeline etc., using the same directives functionality.

So *Directives* in Angular JS, kind of, teaches HTML to do new tricks. We were able to define a custom tag that did a custom action and ended up with a component that could be reused at other places in the App. Checkout the Tab component on AngularJS documentation for more examples.

ROUTING IN ANGULAR JS

If you are familiar with MVC routing, you are wondering, err... routing is a server side thing, how does it matter to a client side framework. Well you are not alone, even I had the same thoughts when I heard of Routing in JS frameworks. But as you will see, routing is real and it works rather nicely on Angular JS.

BREAKING THE FUNCTIONALITY UP INTO 'VIEWS'

So far the application consists of one view with a single ng-controller. As a next step, we would like to implement a details view of the Tweet that will show us more information and if there are any embedded images, show the image as well.

INTRODUCING THE ROUTE PROVIDER $ROUTEPROVIDER

Before we introduce the status view, let's tell Angular that we want to use the *Route Provider*. To do this we will take the *ngTwitter* application variable (in angular hello-angular.js) and add a delegate to the config function on it as follows:

```
// Create ngTwitter Module (roughly Module = namespace
in C#)
var ngTwitter = angular.module("ngTwitter",
['ngResource','ngRoute']);
ngTwitter.config(function ($routeProvider)
{
 $routeProvider.when(
  "/", {
  templateUrl: "timeline"
 });
});
```

Let's quickly review what we did here:

1. We added a delegate to the ngTwitter app and requested for the $routeProvider in the delegate.

2. Next we told the *routeProvider* that when the URL is '/' that is the root URL, it should use a template called 'timeline'.

3. Also notice we are passing 'ngRoute' parameter to angular. module function.

4. One more thing we have to do, is to add reference to the angular-route.js file in the _Layout.cshtml.

```
<script src="~/Scripts/jquery-1.10.2.min.js"></script>
<script src="~/Scripts/bootstrap.min.js"></script>
<script src="~/Scripts/angular.js"></script>
<script src="~/Scripts/angular-resource.js"></script>
<script src="~/Scripts/angular-route.js"></script>
<script src="~/Scripts/hello-angular.js"></script>
```

DEFINING THE TEMPLATE

Above we have declared a *templateUrl* that's set to the value "timeline" but we haven't defined it yet. To define the template, we'll wrap the existing markup inside our ng-controller with a <script> tag. There are two things special about the script tag.

1. Type is set to type="text/ng-template"

2. The *id* is set to the templateUrl value we set in our route

```
<div ng-app="ngTwitter">
 <div ng-controller="TimelineController">
  <script type="text/ng-template" id="timeline">
<!-- the existing markup -- >
…
  </script>
 </div>
 <ng-view></ng-view>
</div>
```

3. Final thing we added was the <ng-view> element. This is where the Template is placed when the route match happens. If we run the app now, we'll see that our Twitter client brings up the last 20 tweets with the Retweet button and text area to send new Tweets. So we have silently introduced routing in our existing Application.

Note: *I have been holding off on clearing the query string of values returned by Twitter for a while now. Linq2Twitter saves the info in a cookie for us already so we don't need to keep it around. I hacked around it by checking for Request.QueryString contents and re-directing to Index page if Query Strings are present after authentication. So the Index action method looks like the following, the new code is highlighted.*

```
public ActionResult Index()
{
 var unAuthorized = Authorize();
 if (unAuthorized == null)
 {
  if (Request.QueryString.Count > 0)
  {
   return RedirectToAction("Index");
  }
  return View("Index");
 }
 else
 {
  return unAuthorized;
 }
}
```

Removing Controller reference from the View to Route

In the markup above, we have a container that specifies the controller name for the Template. This hard-codes the Controller to the View. We can move it to the Route mechanism by updating the Route configuration as follows:

```
$routeProvider.when(
 "/", {
  templateUrl: "timeline",
  controller: "TimelineController"
 });
```

In the view we simply remove the that had the ng-controller attribute specified.

```
<div ng-app="ngTwitter">
 <script type="text/ng-template" id="timeline">
  <!-- the existing markup -- >
  …
 </script>
 <ng-view></ng-view>
</div>
```

ADDING A NEW ROUTE, NG-CONTROLLER AND A NEW VIEW

Adding a new Route is easy, because the $routeProvider's method returns a $routeProvider so we can simply chain another one like this

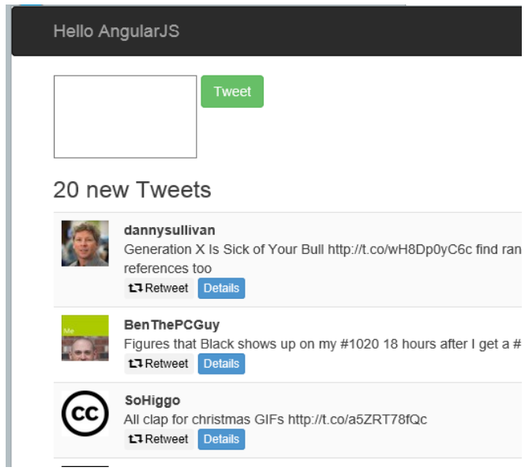
```
$routeProvider.when(
 "/", {
  templateUrl: "timeline",
  controller: "TimelineController"
 }).when(
  "/status/:id", {
  templateUrl: "status",
  controller: "StatusController"
 });
```

What the new route is telling us is, the path /status comes with a parameter called *id* and the template to use is named 'status' and it should use the *StatusController* to get relevant details.

Next in the Index.cshtml, we'll add an anchor tag and style it like a bootstrap button. We'll also set the href to point to the status page and pass the status ID to it. This is accomplished with the following markup:

```
<a class="btn btn-primary btn-mini" href="#/status/
{{item.Id}}">Details</a>
```

Note the #/ notation for URL. This is a part of HTML spec where URL starting with # doesn't cause a postback, instead the browser looks for the anchor in the same page. Running the App will give us a result as shown below.



Setting up the new ng-Controller

Now let's setup the StatusController in the hello-angular.js file.

```
ngTwitter.controller("StatusController", function
($scope, $http, $routeParams,
 TwitterService)
{
 var resultPromise = $http.get("/Home/Status/"+
 $routeParams.id);
 resultPromise.success(function (data)
 {
  $scope.status = data;
 });
});
```

As we can see in the delegate, we have a new parameter getting injected called *$routeParams*. Thanks to our Route definition which specified the parameter as id and the URL in the View that sets up the id value, Angular sets up $routeParams as follows:

```
{ id: 1234567898 }
```

In the controller we setup a *status* object to be used as the view Model in *$scope*. Value of $scope.status is populated by our Server's *Status* action method. We are revisiting $http service here so we have to use the *Promise* to wait for *Success* callback to be called before we can set the value to *$scope.status*.

Adding new action method in HomeController to get Status from Twitter

In the HomeController, we'll add a new *Status* Action method. But before we do that, we'll add a couple of properties to our *TweetViewModel* class. We'll add FavoritedCount, RetweetedCount and HasMedia (a Boolean).

```
public class TweetViewModel{
 public string ImageUrl { get; set; }
 public string ScreenName { get; set; }
 public string MediaUrl { get; set; }
 public string Tweet { get; set; }
 public string Id { get; set; }
 public string FavoriteCount { get; set; }
 public string RetweetCount { get; set; }
 public bool HasMedia { get; set; }
}
```

Next we refactor the translation of Status object into *TweetViewModel* from the Linq query to a helper method *GetTweetViewModel*.

```
private TweetViewModel GetTweetViewModel(Status tweet)
{
 var tvm = new TweetViewModel
 {
  ImageUrl = tweet.User.ProfileImageUrl,
  ScreenName = tweet.User.Identifier.ScreenName,
  MediaUrl = GetTweetMediaUrl(tweet),
  Tweet = tweet.Text,
  Id = tweet.StatusID,
  FavoriteCount = tweet.FavoriteCount.ToString(),
  RetweetCount = tweet.RetweetCount.ToString(),
 };
 tvm.HasMedia = !string.IsNullOrEmpty(tvm.MediaUrl);
 return tvm;
}
```

Finally we add the Status action method to call Twitter using Linq2Twitter.

```
[HttpGet]
public JsonResult Status(string id)
{
 Authorize();
 string screenName = ViewBag.User;
 IEnumerable<TweetViewModel> friendTweets = new
 List<TweetViewModel>();
 if (string.IsNullOrEmpty(screenName))
```

```
 {
  return Json(friendTweets, JsonRequestBehavior.
AllowGet);
 }
 twitterCtx = new TwitterContext(auth);
 friendTweets =
  (from tweet in twitterCtx.Status
   where tweet.Type == StatusType.Show &&
    tweet.ID == id
   select GetTweetViewModel(tweet))
    .ToList();
   if (friendTweets.Count() > 0)
    return Json(friendTweets.ElementAt(0),
    JsonRequestBehavior.AllowGet);
   else
    return Json(new TweetViewModel { Tweet = "Requested
    Status Not Found" },
     JsonRequestBehavior.AllowGet);
}
```

Adding the 'status' View

In the Index.cshtml, we add to following markup that will constitute the Status view.

```
<script type="text/ng-template" id="status">
 <table class="table table-striped">
  <tr>
   <td>
    <img src="{{status.ImageUrl}}" />
   </td>
   <td>
    <div>
     <strong>{{status.ScreenName}}</strong>
     <br />
     {{status.Tweet}}
    </div>
   </td>
  </tr>
  <tr>
   <td>
    <div>
     <retweet-button text="Retweet"
clickevent="retweet(status)"></retweet-button>
    </div>
   </td>
   <td>
    <span class="label label-info" >Retweeted </span>
<span class="badge">{{status.RetweetCount}}</span>
    <span class="label label-info">Favorited </span>
<span class="badge">{{status.FavoriteCount}}</span>
   </td>
  </tr>
 </table>
 <div ng-show="status.HasMedia">
  <img src="{{status.MediaUrl}}" />
 </div>
</script>
```

Most of the markup is easy to understand. We are using the type="text/ng-template" directive to declare this snippet as a template and tying it up with our route using the id=status.

If you remember in the Client Side controller we had added the data to $scope.status hence status is the name of our view model object, so while binding values, we use status.*

Towards the end we have a div with an *ng-show* directive with the value set to 'status.HasMedia'. This is a directive we are using to dynamically show/hide any attached image that a tweet may have. Point to note is that value of *ng-show* is not escaped using {{ ... }} impliying ng-show needs the binding expression from which to get the value and not the value itself.

All done.

Run the application now and click on the details button to navigate to the status page. As we can see below, we have an amusing image shared by the user *Fascinatingpics*.

Broken Retweet functionality

The hawk-eyed will note that the Retweet functionality is broken. This is because, while trying to show how to use a second controller, I left out the retweet method in our TimelineController. Actually the status functionality doesn't need a separate controller of its own. To fix this we do the following:

1. Update *status* route to point to TimelineController

```
"/status/:id", {
    templateUrl: "status",
    controller: "TimelineController"
})
```

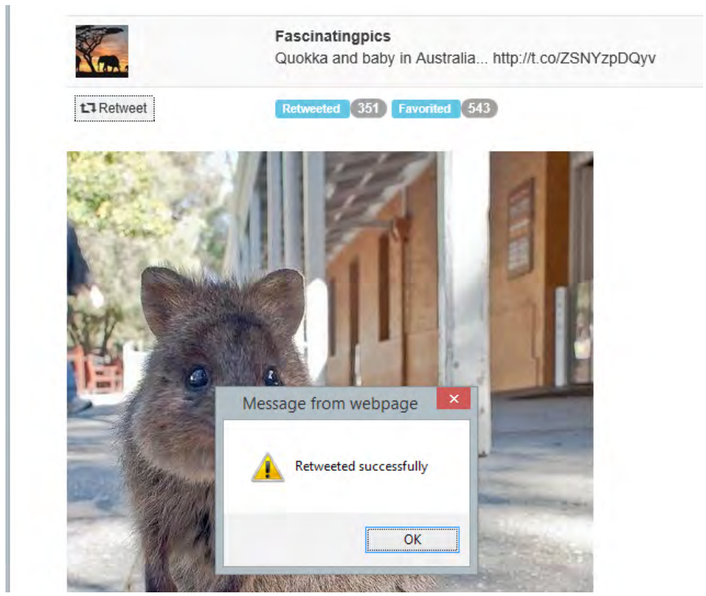
2. Update our *TwitterService* with a new status function, this will also require us to use the *$http* service that we'll simply request Angular to inject. The status function will do the $http.get to retrieve the Status.

```
ngTwitter.factory("TwitterService", function
($resource, $http)
{
 return {
  timeline: $resource("/Home/Tweet"),
  status: function (id)
  {
   return $http.get("/Home/Status/" + id);
  }
 }
});
```

3. Next in the TimelineController, we'll request for the $routeParams service and put an *if* condition to check whether $routeParams has the *id* property and if so, call the status method on the service and wait for the promise to return successfully.

```
ngTwitter.controller("TimelineController", function
($scope, $http, $routeParams, TwitterService)
{
 if ($routeParams.id)
 {
  var statusPromise = TwitterService.
status($routeParams.id);
  statusPromise.success(function (data)
  {
   $scope.status = data;
  });
 }
 else
 {
  $scope.tweets = TwitterService.timeline.query({},
isArray = true);
 }
 // rest of the code remains the same
 …
}
```

That pretty much covers it and now our status view also uses the TimelineController. Thus now the retweet function will work perfectly!



Super sweet, we just verified a use-case for custom directives! We added the Retweet button in a new view and things just worked!

MAKING DIRECTIVES SELF-CONTAINED

In our previous section, we abandoned the StatusController and stuffed everything back into TimelineController. That was an ugly hack. After all we want the Retweet functionality to be self-contained and reusable and Directives are meant to help create reusable components. Let's see how we can do this. Fortunately there is one concept about Directives that we haven't visited yet and that is Directive specific controllers. Yup, directives can have their own controllers as well, so we'll use this feature to further modularize our controller.

Custom Controller Directives

Currently our Retweet Directive is defined as follows:

```
ngTwitter.directive("retweetButton", function ()
{
 return {
  restrict: "E",
  replace: true,
  scope: {
   text: "@",
   clickevent: "&"
  },
 template: "<button class='btn btn-xs' ng-
click='clickevent()'><span class='glyphicon glyphicon-
retweet'></span> {{text}}</button>"
 };
});
```

It uses a local scope that is limited to the properties defined in here (that is text and clickevent properties). This scope overrides the global $scope. This is a key point to keep in mind when using Controllers for Directives.

We can update the above Directive as follows to have its own Controller.

```
ngTwitter.directive("retweetButton", function ($http,
$routeParams)
{
 return {
  restrict: "E",
  replace: true,
  transclude: true,
  controller: function ($scope, $element)
  {
        // do what it takes to retweet
  },
  template: "<button class='btn btn-mini'
ng-transclude><i class='icon-retweet'></i></button>"
 };
});
```

Notice a few significant things:

1. We have let go of the custom scope, because we want access to the $scope.

2. As a result of the above, we have given up on the {{ text }} template in our HTML template for the button.

3. We've also removed the *ng-click* attribute from the button template and put the *ng-transclude* attribute back.

4. In the controller function we have *$element* which is an instance of the button from the template.

Moving the Retweet functionality into the Directive's Controller

Well, we'll need to move the *Retweet* functionality from the *TimelineController* into the *retweetButton* directive's own controller. We add the highlighted code below to our Directive.

```
ngTwitter.directive("retweetButton", function ($http,
$routeParams)
{
 return {
  restrict: "E",
  replace: true,
  transclude: true,
  controller: function ($scope, $element)
  {
   $element.on("click", function ()
   {
    var resultPromise = $http.post("/Home/Retweet/",
    $scope.status);
    resultPromise.success(function (data)
    {
     if (data.success)
     {
       alert("Retweeted successfully");
     }
     else
     {
      alert("ERROR: Retweeted failed! " + data.
      errorMessage);
     }
    });
   });
  },
  template: "<button class='btn btn-mini'
ng-transclude><i class='icon-retweet'></i></button>"
 };
});
```

Let's see what this does line by line:

1. We assign a click event handler to the $element which is essentially the button defined in the template.

2. When the click event fires, we use the $http object to do an http post and pass it the $scope.status object to this.

3. What does this $scope.status contain? Well that depends on the controller. As things are defined now, if we are in the Status Controller it provides us with the status object that is the current tweet as we can see from the controller below.

```
ngTwitter.controller("StatusController", function
($scope, $http, $routeParams,TwitterService){
 var resultPromise = $http.get("/Home/Status/" +
$routeParams.id);
 resultPromise.success(function (data)
 {
  $scope.status = data;
 });
});
```

4. However if we are in the TimelineController, this will return undefined because there is no status object in the TimelineController. Instead we have the tweets object. When looping through the tweet object, we use the following markup and template in the index.cshtml

```
<tr ng-repeat="item in tweets">
   <td>
    <img src="{{item.ImageUrl}}" />
   </td>
   <td>
    <div>
     <strong>{{item.ScreenName}}</strong>
     <br />
     {{item.Tweet}}
    </div>
    <div>
     <retweet-button text="Retweet">Retweet</
      retweet-button>
     <a class="btn btn-primary btn-mini" href="#/
      status/{{status.Id}}">Details</a>
    </div>
   </td>
  </tr>
```

Note we are using the 'item' instance from the tweets collection. If we rename the *item* instance to *status* we are done and the $scope.status in the Directive's controller will be valid.

5. Once we have the $scope.status object sorted out, rest of the code is about handling the return value from the server and displaying a success or failure notice.

We can now remove the $scope.retweet function from the TimelineController.

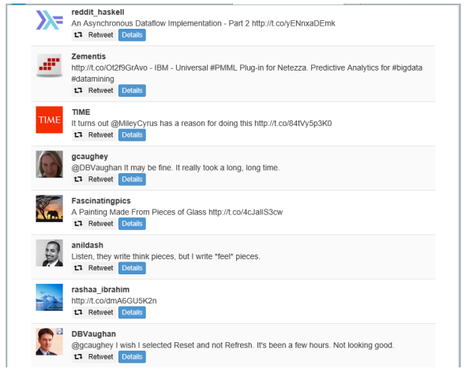
If we run the application now, we'll see that the 'Retweet' text has gone missing. This is because we are now transcluding text from the Directive markup into the Template. So we'll have to update the Directive markup as follows in two places (the timeline and status templates)

```
<retweet-button text="Retweet">Retweet</retweet-button>
```

With this change done we are good to go.

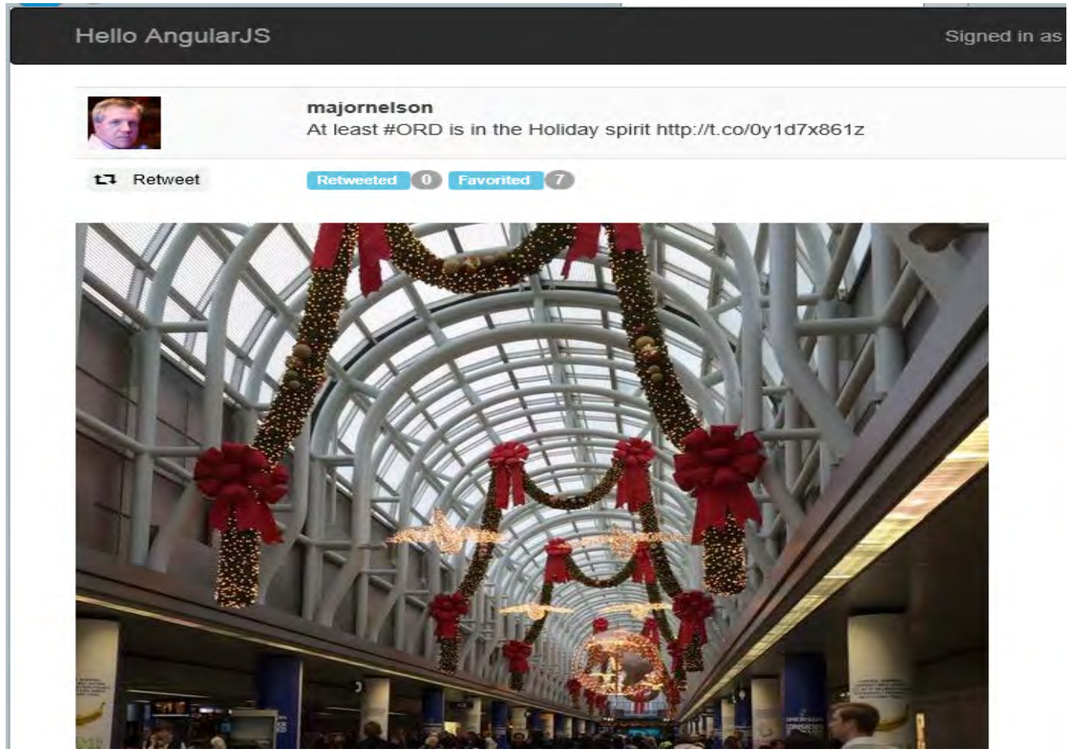
THE FINAL DEMO TIME

Run the application and put a breakpoint in the click event handler of the directive's controller.



Click on Retweet for something you would like to share and it should use the new Directive Controller's click event handler.

Now navigate to the Details page of a Tweet.



Click retweet, you should see the same breakpoint being hit again. Voila!

And that's finally a wrap for now!

CONCLUSION

That was a not-so-quick introduction to some fundamental concepts of Angular JS. We saw how to do Data Binding, fetch and submit Data, create Custom Directives and refactor to some best practices.

We built a Twitter reader over the course of the article, highlighting how well AngularJS was suited to interact with HTTP APIs. Combined with its routing capabilities, AngularJS provides a nice platform for the category of web applications that is now termed as SPAs or Single Page Applications.

It is worth mentioning that Visual Studio 2013 has first class support for Angular JS and it recognizes Angular Directives in your markup and treat them accordingly. ■

Download the entire source code from our GitHub Repository at bit.ly/dncm10-angular

Sumit is a .NET consultant and has been working on Microsoft Technologies for the past 12 years. He edits, he codes and he manages content when at work. C# is his first love, but he is often seen flirting with Java and Objective C. You can Follow him on twitter @sumitkm and read his articles at bit.ly/KZ8Zxb