Get Started with Angular

Sometimes the best way to learn a new technology is to start using it -- a little at a time. In this series of articles, I will show you how to use Angular in a step-by-step manner. You start with a simple example of adding Angular to a project and creating an Angular module and controller. Once you have the controller, you will see how to expose data so it can be bound to UI elements on your HTML page.

Adding Angular to a Project

Create a new ASP.NET Web Application project, or load one of your existing web projects. Right mouse click on the web project, not the solution, and choose **Manage NuGet Packages…** from the menu. You will then see a screen that looks like Figure 1.

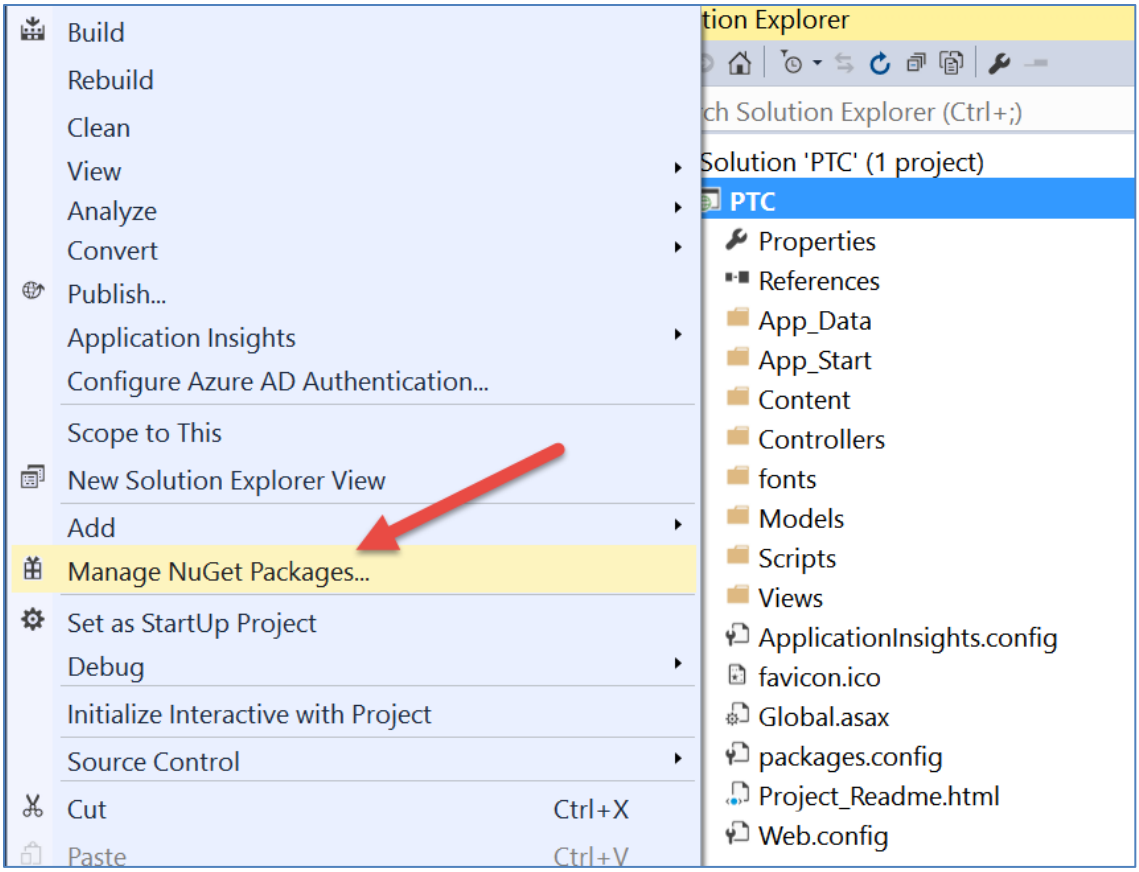

Figure 1: Use the Manage NuGet Packages menu to add Angular to your project.

Click on the Browse tab (Figure 2) and type in AngularJS and press the enter key. Select AngularJS.Core from the list and click on the Install button. Visual Studio will add a few files to your project and you may now use Angular in your project.

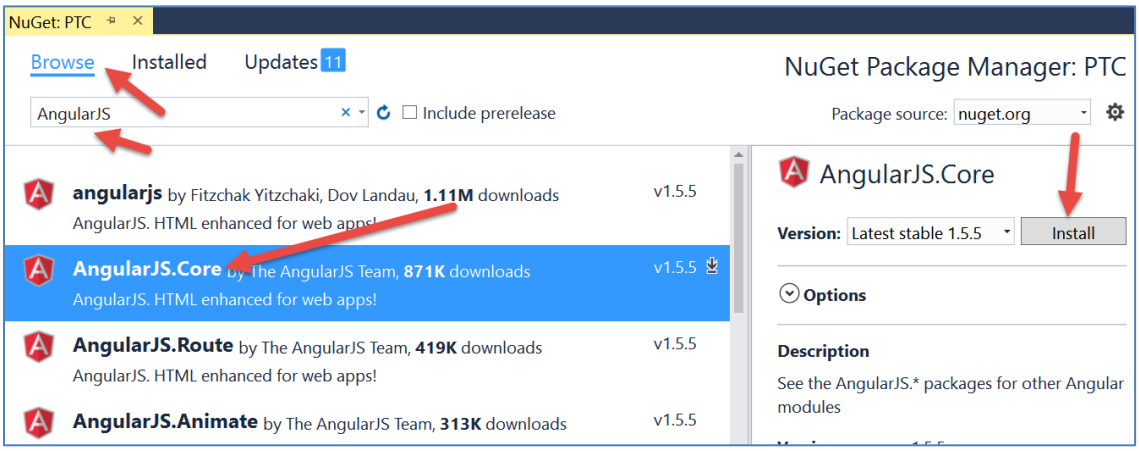

Figure 2: Select AngularJS.Core from the list.

Copyright © 2016 by PDSA, Inc.
All rights reserved worldwide. Reproduction is strictly prohibited.

If you look in the \Scripts folder you should see a few files that start with "angular". The only one you will use on your page for purposes of learning Angular is the angular.min.js.

Create a Test Page

Add a simple HTML page that you can use to test a few concepts of Angular. Right-mouse click on the project and select Add | HTML Page. Type in the name **AngularSample01** and click the OK button. Drag and drop the Content\bootstrap.min.css into the <head> element on the page. Drag and drop the following script files before the </body> tag; Scripts\jquery-VER#.min.js, Scripts\bootstrap.min.js, and Scripts/angular.min.js. Type in the rest of the HTML shown below.

```
<!DOCTYPE html>
<html>
<head>
  <title>Angular Sample</title>
  <meta charset="utf-8" />
  <link href="Content/bootstrap.min.css" rel="stylesheet" />
</head>
<body>
  <div class="container">
    <div class="form-group">
      <label for="ProductName">Product Name</label>
      <input type="text" class="form-control" />
    </div>
    <div class="row">
      <div class="col-xs-12">
        <label for="ProductName">Product Name Entered</label>
        <p class="form-control-static"></p>
      </div>
    </div>
  </div>

  <script src="Scripts/jquery-1.10.2.min.js"></script>
  <script src="Scripts/bootstrap.min.js"></script>
  <script src="Scripts/angular.min.js"></script>
</body>
</html>
```


Copyright © 2016 by PDSA, Inc.
All rights reserved. Reproduction is strictly prohibited.

Create Angular Module

Right mouse click on the Scripts folder and select Add | JavaScript File from the menu. Type in **app** in the text box and click the OK button. In this new JavaScript file, add the following code.

```
(function () {
  'use strict';

  angular.module('ptcApp', []);
})();
```

The above code uses an Immediately Invoked Function Expression (IIFE) to access the angular framework and create a new module. The new module name is passed as a string in the first parameter of the module function. In this case, that module name is 'ptcApp'. Feel free to name this whatever you want, for whatever is appropriate for your application. The second parameter is used to list any dependencies you wish to use in this module. The empty array signifies that there are no dependencies in this case.

Think of an Angular module as a namespace in C#. It is a wrapper, or container, for other things you create within the module. One thing you create within a module is a controller in which you define functions and variables to use from your HTML page.

Create a Controller

In a typical web page you either display data or get data from input fields. With Angular you typically use data binding to accomplish both of these tasks. To use data binding, you first must have some variable names as the source of your bindings. We create these variable names within a "controller" that you define in another JavaScript file. Angular uses the Model View Controller (MVC) design pattern. A controller is responsible for managing the data binding between the model (properties of the $scope variable) and the view (the HTML page).

Right mouse click on the Scripts folder in your project and select Add | JavaScript File from the menu. Type in **productController** in the text box and click the OK button. In this new JavaScript file, add the following code.

Copyright © 2016 by PDSA, Inc.
All rights reserved worldwide. Reproduction is strictly prohibited.

```
1  (function () {
2    'use strict';
3
4    angular.module('ptcApp')
5      .controller('PTCController', PTCController);
6
7    function PTCController($scope) {
8      var vm = $scope;
9
10     // Expose a 'product' object
11     vm.product = {
12      ProductName: 'Pluralsight Subscription'
13     };
14   }
15 })();
```

Line 1 starts an IIFE. On lines 4 and 5 you reference the module named 'ptcApp' you created earlier. Chain the controller() function to declare your intention to use a controller with the name of 'PTCController'. The second parameter to the controller function is a reference to a function callback named PTCController. Lines 7 through 14 is the PTCController function declaration that Angular calls when it needs to reference something from the HTML page.

Angular passes to the controller function a parameter named $scope. The $scope variable is an object created by Angular to provide a mechanism to glue the directives you create on an HTML page with the variables defined within your controller. A good practice is to immediately assign $scope to your own variable name as you see on line 8. Many people use the variable name 'vm' because we think of $scope as a View Model object.

On lines 11 thru 13 you create a variable named 'product' that is an object literal with 1 property named ProductName. You assign a hard-coded value to this property. To display this property's value in and input field on the HTML page, use an Angular directive called 'ng-model' (also written as **ngModel**). The value of this ng-model attribute is the name of the property defined on the $scope.

```
<input type="text"
       ng-model="product.ProductName"
       class="form-control" />
```

The ng-model directive says to look at the current scope of the HTML page and attempt to locate the variable named 'product' within that scope. Then access the ProductName property to display its value within the input field. The question is, how does the ng-model directive know the scope it is contained within, and the controller to use? That is explained in the next section.

Copyright © 2016 by PDSA, Inc.
All rights reserved. Reproduction is strictly prohibited.

Define the Module and Controller for a Page

So far, all you have done is to create an HTML page and a couple of JavaScript files. How do you hook them together? We need to use two more Angular directives; ng-app and ng-controller. The first step is to drag the two .js files onto your HTML page. Be sure to place them below the script tag for angular.

```
<script src="Scripts/angular.min.js"></script>
<script src="Scripts/app.js"></script>
<script src="Scripts/productController.js"></script>
```

Modify the <html> tag of your HTML page so it looks like the following:

```
<html ng-app="ptcApp">
```

Make sure the ng-app (also written as **ngApp**) directive has the exact same name as what you defined in the app.js file. In this case that name is 'ptcApp'. This directive instructs Angular that it should look for a module declared with the name of 'ptcApp' and assign the scope of that module around all of the DOM elements within the <html> tag. The second directive, ng-controller (also written as **ngController**), you place on the <body> tag of your HTML page.

```
<body ng-controller="PTCController">
```

This directive gives scope to everything within the <body> tag on this HTML page to the controller function PTCController. You do not need to place this directive on the <body> tag. You may place this directive on a tag around just one specific area of the page. In this way you could have different controllers for different parts of a page.

Data Binding

Now that you have the scope defined, you bind the 'product' variable defined in your controller function to one or more HTML elements. Open up your HTML page and modify the Product text box to look like the following:

Copyright © 2016 by PDSA, Inc.
All rights reserved worldwide. Reproduction is strictly prohibited.

```
<input type="text"
       ng-model="product.ProductName"
       class="form-control" />
```

When binding to <input> elements in HTML you use the ng-model directive. This directive enables two-way data binding. If you modify the value of the ProductName property in JavaScript code, the UI will be immediately updated with the new value. If the user modifies the input field with a new value, the ProductName property is updated immediately with the new value entered.

To show this in action, let's bind the <p> tag to the product.ProductName property. Since you are not dealing with an <input> element, use the data binding syntax of curly braces as shown below:

```
<label for="ProductName">Product Name Entered</label>
<p class="form-control-static">{{product.ProductName}}</p>
```

Use this data binding syntax when you are simply displaying data on the HTML page. You typically see this kind of data binding when building an HTML table or a <select> list.

Run the Page

If you have followed along with this article, you can now run this HTML page. You should see something that looks similar to Figure 3. If you type into the input field, you will see the value in the <p> tag updated immediately.

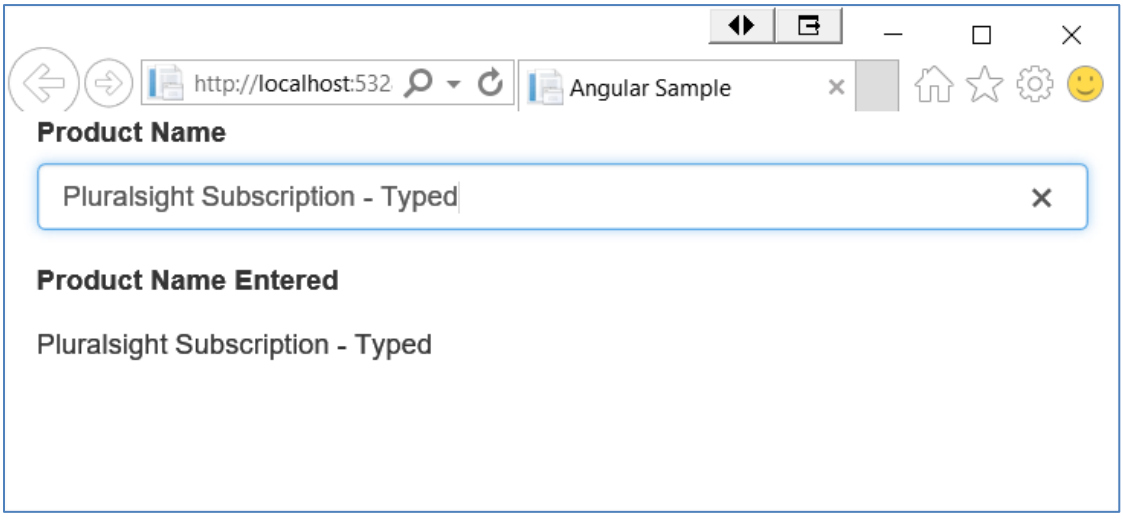

Figure 3: Data binding makes working with data a breeze.

Copyright © 2016 by PDSA, Inc.
All rights reserved. Reproduction is strictly prohibited.

Summary

In this blog post you learned how to add Angular to an MVC project. You learned about modules, controllers and scope. You also created a variable on the scope variable and learned to bind that variable's value to a couple of elements on the HTML page. These are the basics of using Angular. In the next series of blog posts, we will explore a lot of cool things you can do with this powerful framework.

Sample Code

You can download the code for this sample at www.pdsa.com/downloads. Choose the category "PDSA Blog", then locate the sample **PDSA Blog Sample: Get Started with Angular**.

Copyright © 2016 by PDSA, Inc.
All rights reserved worldwide. Reproduction is strictly prohibited.