Add Angular to MVC – Part 1

Many of us have MVC applications currently running. You would like to start using Angular 2 or 4 in your web applications, but don't have the time to completely rewrite. It would be nice if you could just re-write one or two pages in Angular and keep the rest of your MVC project in place. Turns out you can. In this blog post, you will learn how to add Angular 2 or 4 to your MVC applications. For this post, I am assuming you are a Microsoft Visual Studio developer and are familiar with MVC, Angular, C#, and the Web API.

Setup Your Machine

Before you begin to use Angular, you must prepare your development machine. There are two tools required, as well as you must configure Visual Studio 2015 to use TypeScript. You should also get the Angular quick start project.

Install Node

If you have not done so already, download and install Node.js and NodeJS Package Manager (npm). You can get these two packages at https://nodejs.org/en/download/. Follow the instructions for downloading and installing these tools on nodejs.org.

Configure Visual Studio 2015

Most developers are using TypeScript for Angular development. Ensure you have downloaded and installed TypeScript 2.x for Visual Studio 2015. Select the Tools | Extensions and Updates menu to bring up the Extensions and Updates window (Figure 1) for Visual Studio. Click on the Installed node in the tree view and look through your list and see if you have installed TypeScript 2.x. If you have an older version of TypeScript you need to update it.

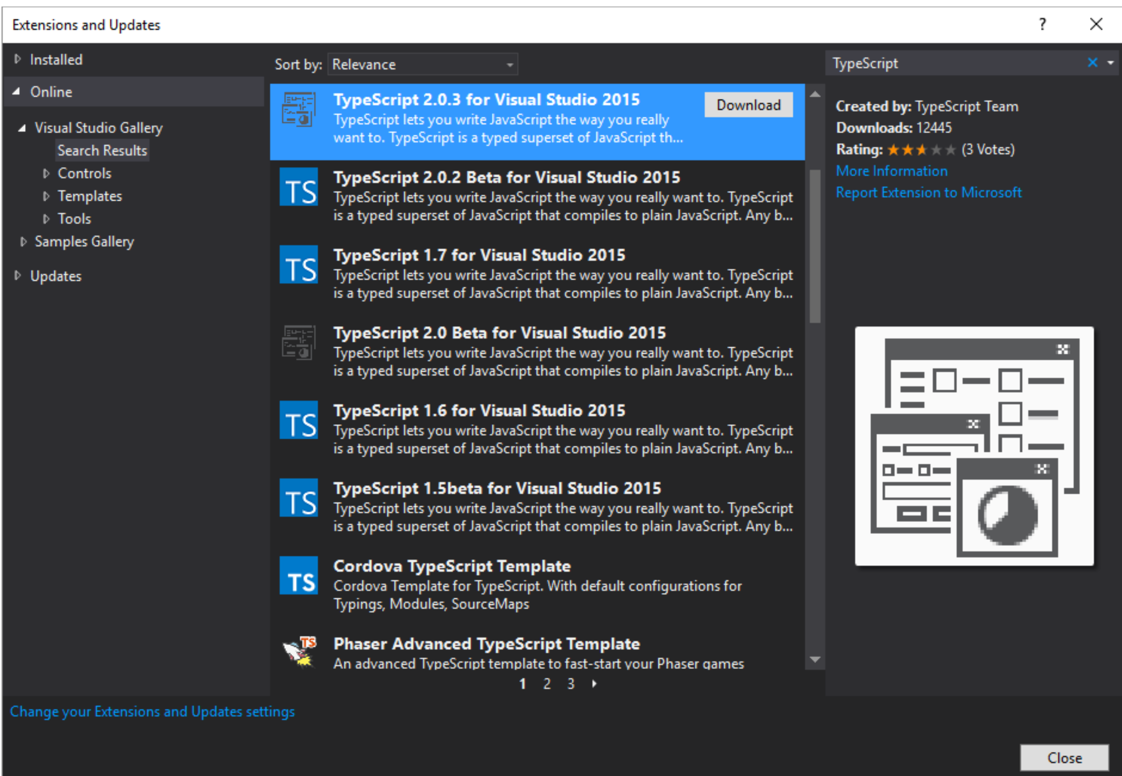

Figure 1: Use the Extensions and Updates window to check for TypeScript.

To update to TypeScript 2.x, click on the Online node in the tree view and perform a search for TypeScript. Locate the latest version of TypeScript and download and install it into Visual Studio.

One last configuration item for Visual Studio is to select the **Tools | Options** menu and expand the Projects and Solutions node. Click on the External Web Tools (Figure 2). Ensure that the $(PATH) variable is located above any $(DevEnvDir) variables if they exist in your environment. In my installation of Visual Studio, these variables did not exist.

Copyright © 2017 by Fairway Technologies, Inc.
All rights reserved worldwide. Reproduction is strictly prohibited.

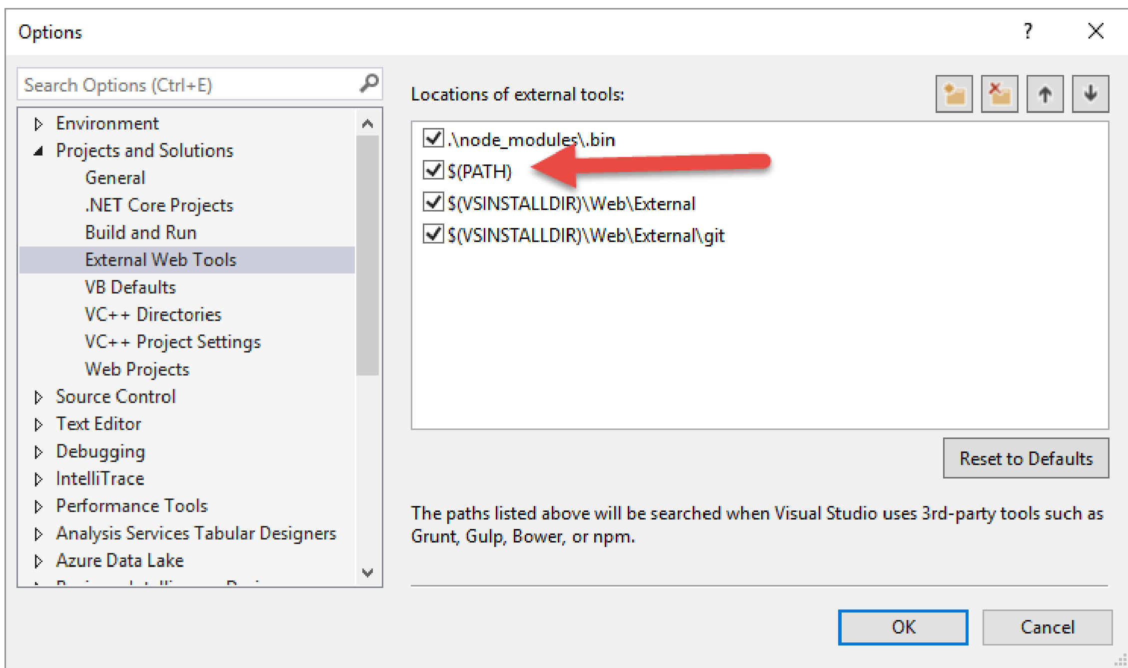

Figure 2: Make sure the $(PATH) variables is high in your location of external tools.

Download Angular Quick Start

Now that you have configured Visual Studio, you are going to need some configuration files and some TypeScript files to make it easier to get started with Angular. The Angular team has created a sample project that contains these files. This sample project is not a Visual Studio project, so you are only going to use some of the files from this project and not bring all of them into your project. Download the quick start zip file from https://github.com/angular/quickstart. After the zip file has been downloaded, unzip the files into a folder on your hard drive.

Sample MVC Application

Let's look at a sample MVC application, named PTC, that I am going to use as a demonstration (Figure 3). You can download this sample, and the final sample by following the instructions at the end of this blog post.

Copyright © 2017 by Fairway Technologies, Inc.
All rights reserved. Reproduction is strictly prohibited.

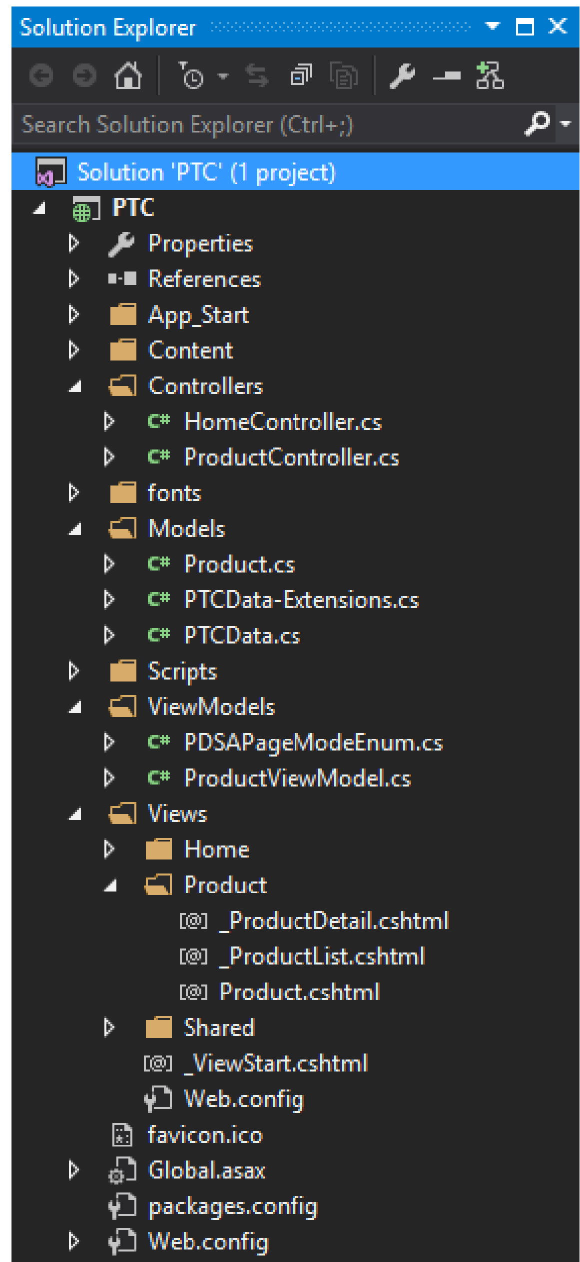

Figure 3: A sample MVC application to display a product list

This is a vanilla MVC application that does not have the Web API in it. It has the following classes that are of interest.

File	Description
ProductController	The MVC controller to display the \Views\Product\Product.cshtml page.
Product	An entity class to represent a product object
PTCData-Extensions	Additional validation for Product data

Copyright © 2017 by Fairway Technologies, Inc.
All rights reserved worldwide. Reproduction is strictly prohibited.

PTCData	Entity Framework generated data layer.
PDSAPageModeEnum	Enumeration for the mode that page is currently in
ProductViewModel	A view model that is called from the ProductController to retrieve data
_ProductDetail.cshtml	A partial page to display, add, and edit a single product record.
_ProductList.cshtml	A partial page that displays a list of product data
Product.cshtml	A MVC page to list or edit product(s) retrieved from the view model.

Table 1: A list of the classes in the PTC MVC application

Install Product Table

Before you can run this sample, you need to download the sample project, locate the \SqlScripts folder and install the Product.sql into a SQL Server database. After you have created the Product table, open the Web.config file and adjust the connection string to point to your server and your database.

Test the Page

You can now run the Product.cshtml page and ensure you get something that looks like Figure 4.

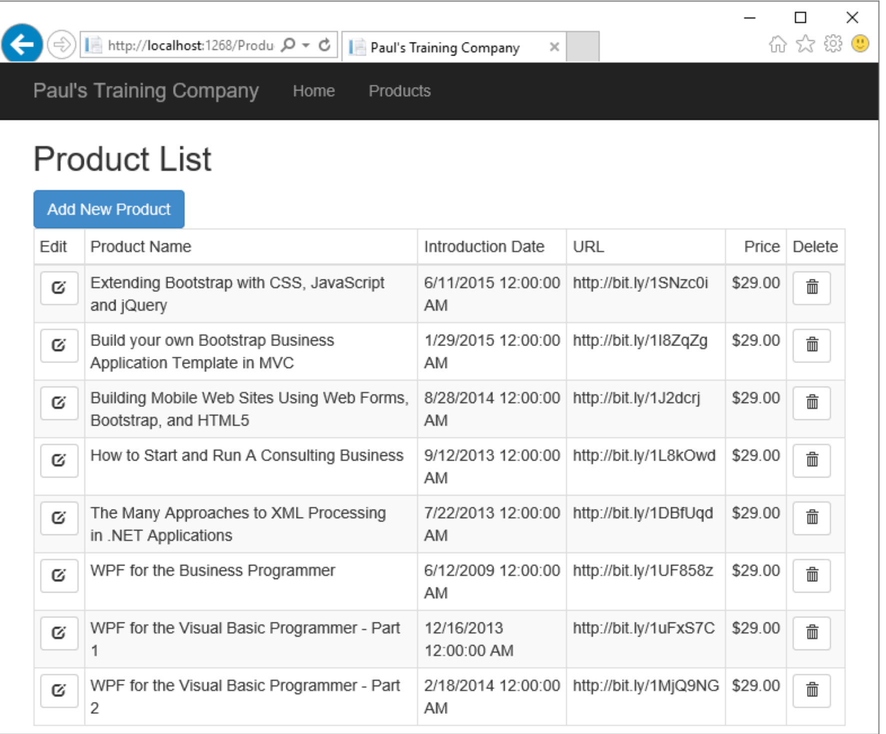

Figure 4: The product list page

Copyright © 2017 by Fairway Technologies, Inc.
All rights reserved. Reproduction is strictly prohibited.

Add Angular Files

You need to add a few Angular files from the quick start project you downloaded. Prior to doing this, add a new folder named \app to your MVC project. Using Windows Explorer, navigate to the folder where you extracted the quick start files. Do NOT copy all these files into your Visual Studio project, you do not need all of them. Locate and copy the following files into the root of your new Visual Studio project.

- \package.json
- \tslint.json
- \src\tsconfig.json

Copy the file \src\systemjs.config.js into the \scripts folder of your VS project.

Copy the file \src\main.ts into the \app folder of your VS project. You will most likely receive a message that your project has been configured to support TypeScript. It will ask you if you want to search for TypeScript typings. Just answer No as you are not going to need these right now.

Expand the \src\app folder in the quick start folder and copy the following TypeScript files located in this folder into the \app folder you created in your VS project.

- app.component.ts
- app.module.ts

Restore Packages

Even though you added some TypeScript files, nothing is going to work yet. You need to download a new folder using the NodeJS Package Manager (npm). Visual Studio can download all of these files using npm, but you must **first close and reopen Visual Studio**. If you don't close and reopen Visual Studio, the appropriate menu item needed will not show up. Go ahead and close your instance of Visual Studio now, then, reopen the ProductApp project. Right mouse click on the package.json and select **Restore Packages** (Figure 5) from the menu. This process takes a little while, so be patient.

Copyright © 2017 by Fairway Technologies, Inc.
All rights reserved worldwide. Reproduction is strictly prohibited.

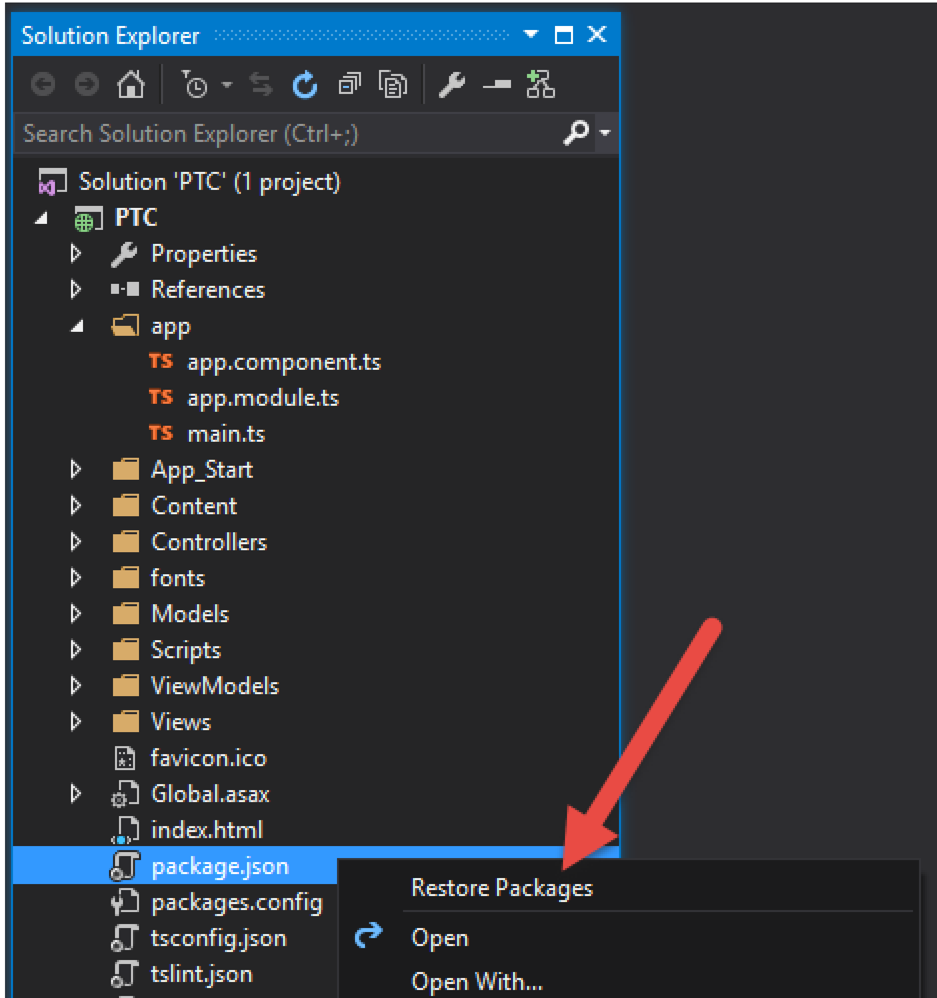

Figure 5: The Restore Packages menu only shows up after closing and reopening Visual Studio.

After the installation of the packages is complete, click on the Show All Files icon in the Solution Explorer window. You should see a new folder has appeared named node_modules (Figure 6). DO NOT include this folder in your project, you do not need all these files in your project. They are just there to support the various libraries you use when developing with Angular.

Copyright © 2017 by Fairway Technologies, Inc.
All rights reserved. Reproduction is strictly prohibited.

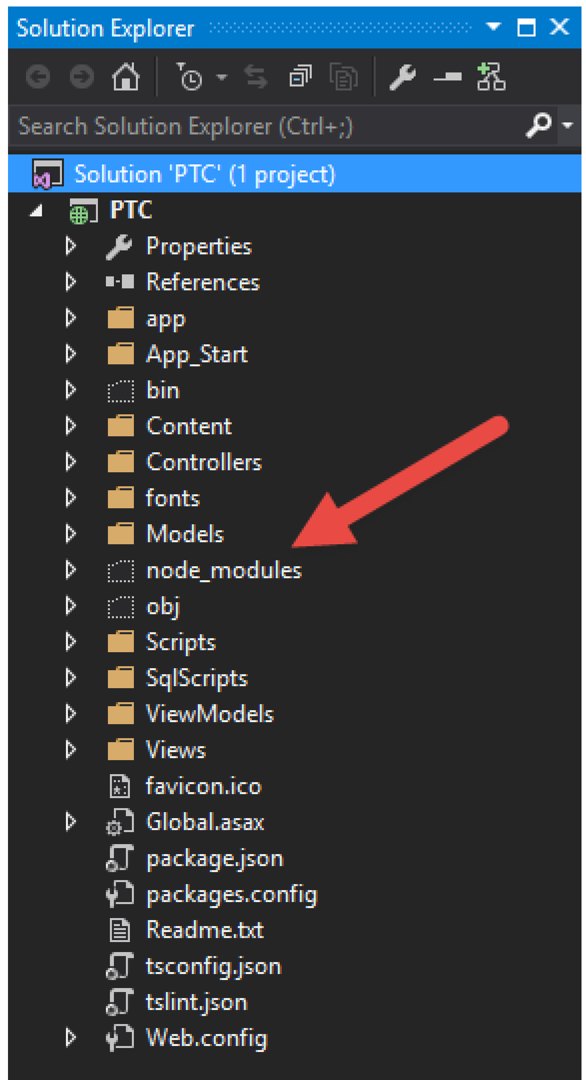

Figure 6: Don't include the node_modules folder in your project.

Add Links to the MVC Shared Layout

Navigate to where you installed the quick start files and open the \src\index.html page. You need to copy a few lines of code from this page into the \Views\Shared_Layout.cshtml file. Open the _Layout.cshtml file in Visual Studio. Locate the following lines of code in the index.html page and paste them into the <head> section of the _Layout.cshtml file.

Copyright © 2017 by Fairway Technologies, Inc.
All rights reserved worldwide. Reproduction is strictly prohibited.

```
<base href="/">
```

I like to the above line of code as the first one after the <head> element. Locate the next lines that start with the comment shown below and copy them all after the @Scripts.Render() and before the </head> element.

```
<!-- Polyfill(s) for older browsers -->
<script src="node_modules/core-js/client/shim.min.js">
</script>

<script src="node_modules/zone.js/dist/zone.js"></script>
<script src="node_modules/systemjs/dist/system.src.js">
</script>

<script src="systemjs.config.js"></script>
<script>
  System.import('main.js')
       .catch(function(err){ console.error(err); });
</script>
```

Since the _Layout is in a different location in your project from the index.html in the quick start project, you need to fix up the src attributes. Add a forward slash in front of each of the above references for the node_modules.

```
<script src="/node_modules/core-js/client/shim.min.js">
</script>

<script src="/node_modules/zone.js/dist/zone.js"></script>
<script src="/node_modules/systemjs/dist/system.src.js">
</script>
```

For the systemjs.config.js file, you placed that into the Scripts folder, so you need to add /scripts/ in front of that file reference.

```
<script src="/scripts/systemjs.config.js"></script>
```

Locate the <script> tag that is responsible for importing the main.js file which is transpiled from the main.ts file. This code is shown in the following code snippet.

Copyright © 2017 by Fairway Technologies, Inc.
All rights reserved. Reproduction is strictly prohibited.

```
<script>
  System.import('main.js')
    .catch(function(err){ console.error(err); });
</script>
```

Since you moved the main.ts file into the \app folder, modify the parameter passed to the import function to ‘app/main’ as shown in the following snippet.

```
<script>
  System.import('/app/main')
    .catch(function(err){ console.error(err); });
</script>
```

In the quick start project, the main.ts file was expecting to find the app.module.ts file in the \app folder below the root folder. However, since these two files are now located in the same folder, you need to open the main.ts file located in the \app folder and locate the following line of code.

```
import { AppModule } from './app/app.module';
```

Remove the ‘/app’ portion from this line of code so that the final code looks like the following snippet.

```
import { AppModule } from './app.module';
```

You should run your Product page again just to make sure that your application still compiles and runs.

Eliminate Hard-Coded HTML Template

The app.component.ts file contains a template property to output some HTML. I do not like having HTML written in TypeScript, so create an HTML page to display the HTML for the landing page of your Angular application.

Right mouse click on the \app folder and select **Add | HTML Page** and set the name to app.component.html. Click the OK button. Remove the HTML in the page and replace it with the following:

Copyright © 2017 by Fairway Technologies, Inc.
All rights reserved worldwide. Reproduction is strictly prohibited.

```
<router-outlet></router-outlet>
```

In the HTML above you are adding a location for the Angular routing engine to place HTML that comes from different components. For example, you are going to create a product list component and corresponding HTML page to display product data. That HTML will be placed in between these two tags.

Open the \app\app.component.ts file and you will see the following declaration.

```
@Component({
  selector: 'my-app',
  template: `<h1>Hello {{name}}</h1>`,
})
```

Now that you have created the new HTML page, you are going to remove the hard-coded HTML from the *template* property in the @Component decorator. I have found that if you have any more than a few lines of HTML, this property becomes cumbersome to maintain. Let's change the *template* property to the *templateUrl* property and set that property to the string 'app.component.html'.

```
@Component({
  moduleId: module.id,
 selector: 'my-app',
 templateUrl: './app.component.html'
})
```

Another change you are making to this @Component decorator is adding the property *moduleId* and setting that property to the value module.id. This property helps Angular understand relative paths in relation to the current component. If you did not use the *moduleId* property, then you change the *templateUrl* property to './app/app.component.html'. I prefer to use relative paths as opposed to having to fully quality the path in relation to the root.

The AppComponent class that came with the quick start sample has a *name* property set to the word 'Angular'. Just delete this code and make your AppComponent class look like the following:

```
export class AppComponent { }
```

You will learn more about how the routing works a little later in this article. But first, let's build the product list component and HTML.

Copyright © 2017 by Fairway Technologies, Inc.
All rights reserved. Reproduction is strictly prohibited.

Stub the Angular Product Components

To test that you can route to a Product list page from your MVC page, create a couple of new files.

Right mouse click on the \app folder and add a new folder called \product. Right mouse click on this product folder and add a new HTML page called product-list.component.html. Delete all the HTML in this new page and add the single line below:

```
<h2>Product List</h2>
```

Right mouse click on the product folder and add a new TypeScript file named product-list.component.ts. Add the following code:

```
import { Component } from "@angular/core";

@Component({
  moduleId: module.id,
  templateUrl: "./product-list.component.html"
})
export class ProductListComponent { }
```

Update the app.module.ts File

Add the ProductListComponent class to the app.module.ts. Open the app.module.ts file and add an import near the top of the file that looks like the following.

```
import { ProductListComponent }
from "./product/product-list.component";
```

In the @NgModule decorator, add the ProductListComponent class to the *declarations* property by adding a comma after AppComponent and typing in the name of this class as shown below.

Copyright © 2017 by Fairway Technologies, Inc.
All rights reserved worldwide. Reproduction is strictly prohibited.

```
declarations: [ AppComponent, ProductListComponent ]
```

Angular Routing

No Angular application would be complete without routing. Routing allows us to navigate from one page to another. The app.component.html page you just created is the main page for getting to other pages in our application via the routing engine. Add a new TypeScript file called app-routing.module.ts and add the code shown in Listing 1.

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes }
  from '@angular/router';

import { ProductListComponent }
  from "./product/product-list.component";

const routes: Routes = [
  {
    path: 'productList',
    component: ProductListComponent
  },
  {
    path: 'Product/Product',
    redirectTo: 'productList'
  }
];

@NgModule({
 imports: [RouterModule.forRoot(routes)],
 exports: [RouterModule]
})
export class AppRoutingModule { }
```

Listing 1: The routine class defines one or more routes for your application.

The important part of the app-routing.module.ts file is the constant named *routes*. This constant contains an array of Route objects. Each object has a *path* property which is used in the *routerLink* attribute you created earlier. If the path matches the address in the browser, then the *component* property is used to instantiate an instance of the object listed in this property. Once this class is instantiated, the HTML defined in that classes *templateUrl* property is inserted into the location where the <router-outlet> directive is located.

Copyright © 2017 by Fairway Technologies, Inc.
All rights reserved. Reproduction is strictly prohibited.

The *routes* constant is added to the singleton instance of the Router service using the forRoot function. You can see this in the *imports* property of the @NgModule decorator. You can add as many routes to the *routes* constant as you need for your application.

Overview of Angular Routing

With all that description above, I thought a graphical representation (Figure 7) of the Angular routing you are using in this application might help clear things up a little.

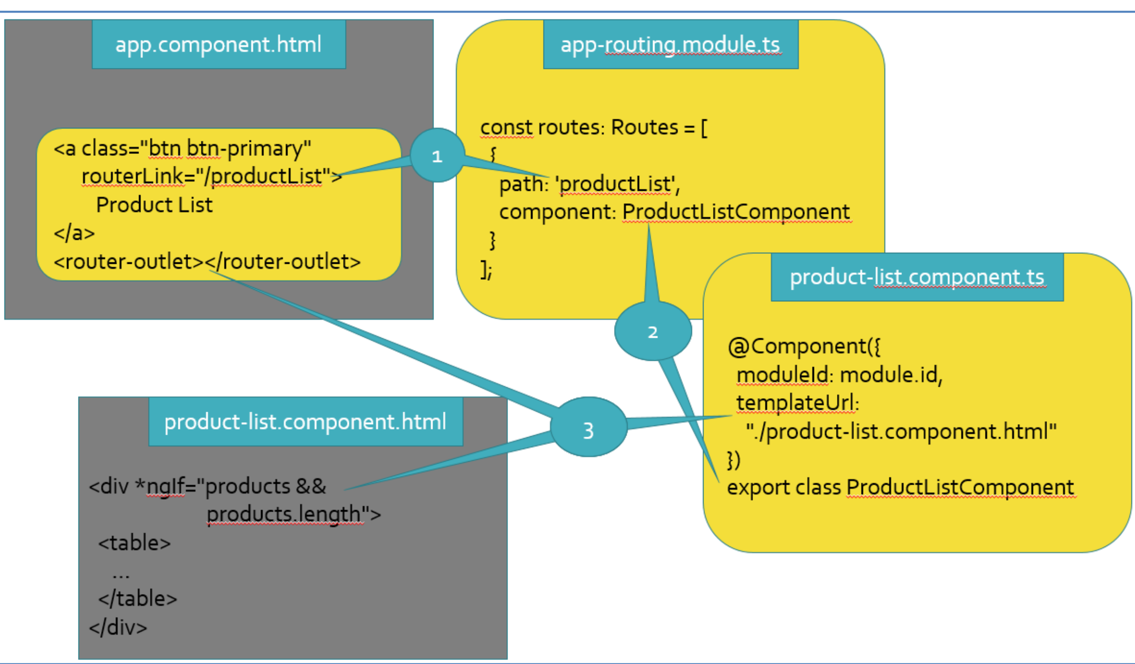

Figure 7: An overview of the Angular routing process.

1. Anytime the browser address bar is updated via an anchor tag, or other mechanism, the last part of the address is matched up to one of the routes added to the singleton instance of the Router service.
2. Once it finds the path in the Router service, it instantiates the class listed in the *component* property.
3. After the class is instantiated, the *templateUrl* property is read from the @Component decorator.
4. The HTML from the file listed in the *templateUrl* property is loaded and inserted into the DOM at the location of the <router-outlet> directive.

Copyright © 2017 by Fairway Technologies, Inc.
All rights reserved worldwide. Reproduction is strictly prohibited.

Update the app.module.ts File

Just as you have done with all your other classes you created, you need to declare your intention to use routing in the app.module.ts file. Open the app.module.ts file and add the following import statement near the top of the file.

```
import { AppRoutingModule }
from './app-routing.module';
```

Add the AppRoutingModule to the *imports* property in the @NgModule decorator. The *imports* property should now look like the following code snippet.

```
imports: [BrowserModule, AppRoutingModule ]
```

Call our Angular page from the Product.cshtml page

Open the \Product\Product.cshtml page and delete all the code. Write the following:

```
<my-app>Loading products...</my-app>
```

Remember the <my-app> is the selector in the app.component.ts file. When this selector is encountered, the Angular system kicks in and starts loading our Angular components to run. You have now successfully removed yourself from the MVC system and are now running all client-side!

Run the Page

At this point you should be able to run the page and see the <h1> you entered. Your page should like Figure 8.

Copyright © 2017 by Fairway Technologies, Inc.
All rights reserved. Reproduction is strictly prohibited.

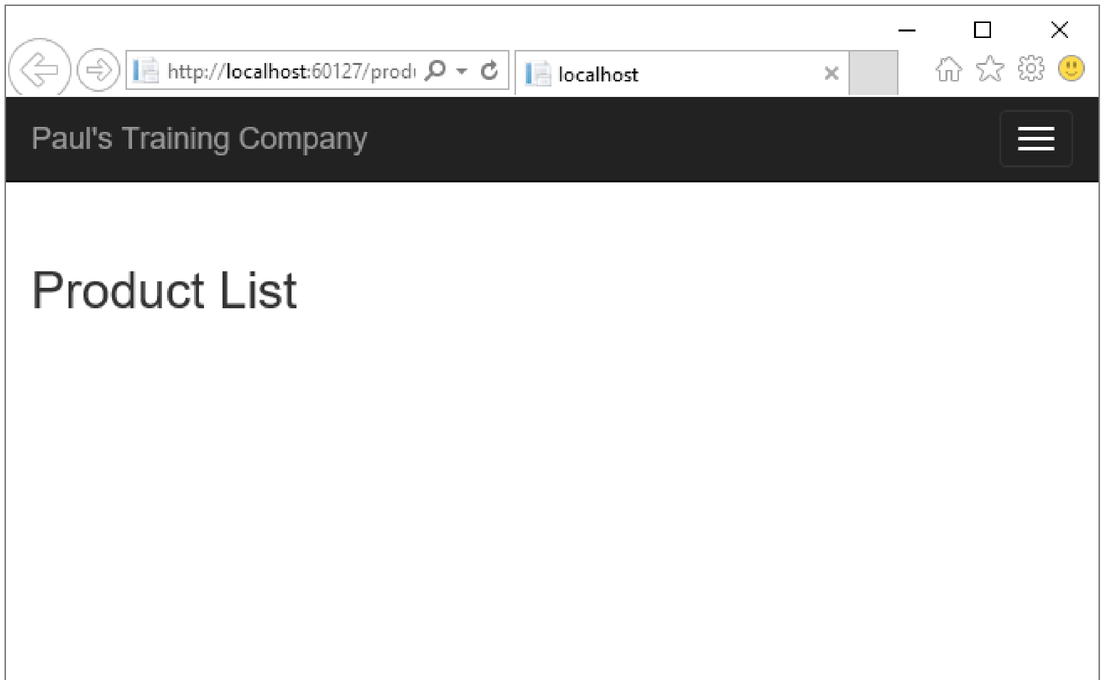

Figure 8: You have rerouted to an angular page from an MVC page.

Summary

Congratulations! You have successfully integrated Angular into a Visual Studio MVC Application. There are a lot of steps to get Angular up and running, however, once you get the basics configured, adding new pages and new components is quite easy because you use the same pattern you learned in this post. In the next blog post, you add and call a Web API from an Angular service, and display the product data in an HTML page using Angular.

Sample Code

You can download the code for this sample at www.pdsa.com/downloads. Choose the category "PDSA Blogs", then locate the sample **Add Angular to MVC - Part 1**. To run the sample, locate the \SqlScripts\Product.sql file and run the script in a SQL Server database. Open the Web.config file in the

Copyright © 2017 by Fairway Technologies, Inc.
All rights reserved worldwide. Reproduction is strictly prohibited.

project and update the connection string to point to your server and database name.

Copyright © 2017 by Fairway Technologies, Inc.
All rights reserved. Reproduction is strictly prohibited.