Configuration Settings for Angular Applications

Just like in .NET applications, you might want to have configuration settings in your Angular applications that you can access from any component or service class. There are many approaches you can take for global settings, however, I am going to use a service that can be injected into any class. I think the flexibility of using a service is an ideal method for providing application-wide settings to any class that needs them. This blog post will describe the process of creating this service.

An AppSettings Class

Create a class with properties to hold the values you wish to use in your Angular application. If you have a Product form that the user will be using to add new products to a database, you might want to provide some default values when they are adding a new product. Below is a class that I named AppSettings. This class has two properties; defaultUrl and defaultPrice. These values will be set into a Product object in a product component class a little later.

```
export class AppSettings {
  defaultUrl: string = "http://www.fairwaytech.com"
  defaultPrice: number = 1
}
```

To create this class, add a \shared folder under the \src\app folder and add a new Typescript file named AppSettings.ts. Enter the code shown in the snippet above.

An AppSettingsService Class

It is now time to create a service class to return an instance of the AppSettings class. Add a new Typescript file under the \shared folder named AppSettingsService.ts. Add the code shown below.

```
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/observable/of';

import { AppSettings } from "./appsettings";

@Injectable()
export class AppSettingsService {
  getSettings(): Observable<AppSettings> {
    let settings = new AppSettings();

    return Observable.of<AppSettings>(settings);
  }
}
```

This service class is fairly standard as far as service classes go. In the getSettings() method you create a new instance of the AppSettings class and return that object from this service. The main reason I create a service here is to provide the flexibility to change the implementation of how I retrieve the settings later. For example, I might choose to read the settings from a JSON file, or I might even make a Web API call to get the settings. Any method that calls this service will always make the same call regardless of where those settings are stored. The calling methods don't know if the implementation changes, they still receive the same settings class.

Using the AppSettingsService Class

To retrieve the settings from this service class, import the AppSettings and AppSettingsService classes in your component.

```
import { AppSettings }
  from '../shared/appsettings';
import { AppSettingsService }
  from '../shared/appsettings.service';
```

Add the AppSettingsService to the constructor to tell Angular to inject the service into this class. Create a private property in your class to hold the settings retrieved from the getSettings() method. In the ngOnInit() method make the call to the getSettings() method using the subscribe method. The subscribe method has three parameters you can pass; a success function, an error function, and a completed function. In the success function set the result returned to the settings property. In this sample, I am ignoring the error. In the completed function, I create a new instance of the Product object and assign the price and url properties to the defaults returned in the settings object. The Product object is bound to field on my detail page as shown in Figure 1.

```
export class ProductDetailComponent implements OnInit {
  constructor(
    private appSettingsService: AppSettingsService) {
  }

  product: Product;
  private settings: AppSettings;

  ngOnInit(): void {
    this.appSettingsService.getSettings()
      .subscribe(settings => this.settings = settings,
        () => null,
        () => {
          this.product = new Product();
          this.product.price = this.settings.defaultPrice;
          this.product.url = this.settings.defaultUrl;
        });
  }
}
```

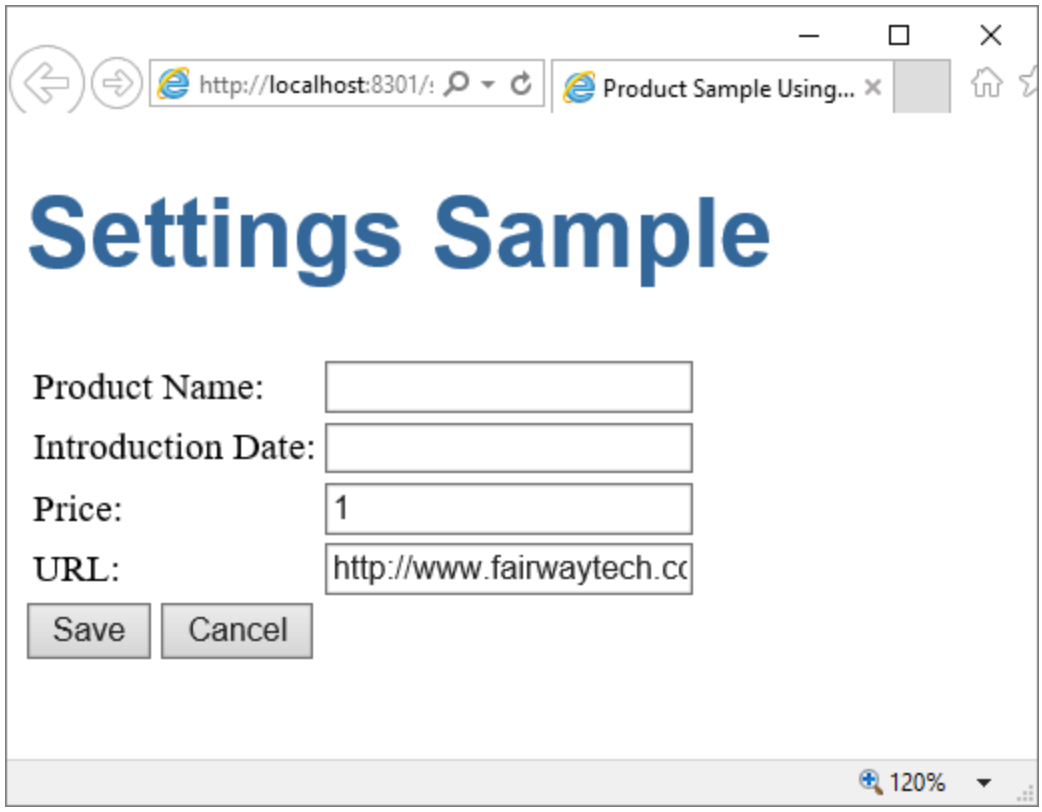

Figure 1: The Price and URL fields are filled in with defaults from the AppSettings class.

Get Settings from a JSON File

Instead of hard-coding the settings values, let's put those settings into a JSON file. Create a folder called \assets under the \src\app folder. Add a JSON file named appsettings.json. Add the following into this file.

```
{
  "defaultUrl": "http://angular.io",
  "defaultPrice": 2
}
```

Change the AppSettingsService class to read from this file. Import the Http and Response classes from @angular/http. Import the ReactiveJS operators map and catch, and the observable, throw. Modify the getSettings() method to call the http.get() method, passing in the path to the JSON file you created. I use a function named extractData() to extract the response returned from

calling the map() function. I also created a handleErrors() method to handle any exceptions.

```
import { Injectable } from '@angular/core';
import { Http, Response } from '@angular/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/catch';
import 'rxjs/add/observable/throw';

import { AppSettings } from "./appsettings";

@Injectable()
export class AppSettingsService {
  constructor(private http: Http) {
  }

  getSettings(): Observable<AppSettings> {
    return this.http.get("/src/app/assets/appsettings.json")
      .map(this.extractData)
      .catch(this.handleErrors);
  }

  private extractData(res: Response) {
    let body = res.json();
    return body || {};
  }

  private handleErrors(error: any): Observable<any> {
    console.error('An error occurred', error);

    return Observable.throw(error.message || error);
  }
}
```

If you don't already have it, you need to import the HttpModule from @angular/http in your app.module.ts file. Add the HttpModule to the imports property in the @NgModule() decorator function.

Summary

In this blog post you learned an approach for handling application-wide settings for Angular applications. A service approach is the most flexible approach to providing settings to any other class in your application. You can choose to store your settings in a class, in an external JSON file, or you can even make a call to a Web API to retrieve the values. The consumers of your

service don't care where they values come from, nor do they have to change the call to the service.

You can get the samples at www.pdsa.com/downloads. Choose "PDSA Blogs" from the Category, then select "Configuration Settings for Angular Applications".