Seneca
SCHOOL OF INFORMATION AND COMMUNICATIONS TECHNOLOGY

BTI425

Web Programming for Apps and Services

Notes | Weekly
Resources | Graded work
Policies | Standards
Professors & addendum | Code examples

Angular services example

In this document, you will learn how to add and use a service in an app. We'll use the Angular app template version 1 as a base or starter app. Then, we will make the changes to support and use services.

Getting started

Start with the template, version 2, from the course's GitHub code repository.

FYI, a solution is published in the Week 9 folder.

Working with a web service

We want to use a web service that's external to (and outside of) this app. In this section, we use a public read-write web API.

When compared to the simple "getting started" example above, there are three new or additional interrelated concepts and techniques that we are interested in:

1. HttpClient

2. RxJS
3. Observable

We blend all three together when working with a web service.

Enable the app to use HTTP

Open the app module (`app.module.ts`) for editing. Near the top, below the BrowserModule import, add this import statement:

```
import { HttpClientModule } from "@angular/common/http";
```

Then, add the `HttpClientModule` to the `imports` collection in the `@NgModule` decorator, below the BrowserModule.

This action will enable all services and components in the app to work with a web API.

Important Note

When using **HttpClient** anywhere else in your application (e.g. a `whatever.service.ts` file), be sure to *import **HttpClient*** (and not HttpClientModule) into that service or component. For example:

```
import { HttpClient } from "@angular/common/http";
```

Visit and browse the web service

For the following, we could use our Teams API. Just to be different and flexible, we'll use a different web service.

A programmer with the moniker *typicode* has published a small-size web service named "JSONPlaceholder". Open Postman (or JSON Formatter), and look at the results from these URLs:

`http://jsonplaceholder.typicode.com/posts`

`http://jsonplaceholder.typicode.com/users`

`http://jsonplaceholder.typicode.com/comments`

All return an array/collection of data.

Write view model classes to match the data

Now, write *view model* classes to match the data returned by the first two URLs above. We must do this when working with web API. Here's how:

Work with the web API. Study the *shapes* of the data in the responses, and the data that must be sent for add and edit requests.

Create a new source code file named `data-classes.ts`. In it, we will write several classes, which will allow us to work with "posts", "users", and "comments":

```
// View models for the typicode.com web service

export class Post {
    id:         number;
    userId:     number;
    title:      string;
    body:       string;
}

export class Comment {
    id:         number;
    postId:     number;
    email:      string;
    name:       string;
    body:       string;
}

export class Geo {
    lat:        number;
    lng:        number;
}

export class Address {
    city:       string;
    geo:        Geo;
    street:     string;
    suite:      string;
    zipcode:    string;
}
```

```
export class Company {
    bs:             string;
    catchPhrase:    string;
    name:           string;
}

export class User {
    address:        Address;
    company:        Company;
    email:          string;
    id:             number;
    name:           string;
    phone:          string;
    username:       string;
    website:        string;
}
```

Prepare the service

Open the service for editing. Add the following near the top:

```
import { Observable } from "rxjs/Observable";

import { Post, Comment, Geo, Address, Company, User } from "./data-cl
```

We need `Observable`, because that's the type of the web API get/fetch result.

We need the view model classes to enable the model binder to correctly create the arrays/collections.

Recommendation - create a "base url" string field

Recommendation - create a "base url" string field, to hold the long and constant part of the URL to the web service. Doing this will make it easy to create a concatenated string that includes the segment we want.

```
  private url = "http://jsonplaceholder.typicode.com";
```

Write a function for each web service resource

Write a function for each web service resource that the app needs. Assume, as noted above, that we want posts, comments, and users.

```
getPosts(): Observable<Post[]> {
  return this.http.get<Post[]>(`${this.url}/posts`)
}

getComments(): Observable<Comment[]> {
  return this.http.get<Comment[]>(`${this.url}/comments`)
}

getUsers(): Observable<User[]> {
  return this.http.get<User[]>(`${this.url}/users`)
}
```

Use the new function(s) in the service

Let's view the web API content in a new component. Generate it, and then configure it to participate in routing and the nav menu. Finally, open it for editing.

Import the service.

Import the view model class. Let's work with posts first, so we need the Post class imported.

Inject the service into the constructor.

Create a field to hold the content we want; it will be a Post array/collection field.

We want to call the service method when the component is loaded/initialized, so do that in the `ngOnInit()` function.

Finally, open the markup template for editing. Add code similar to the following:

```
<h3>Typicode "Post" list</h3>
<table class="table table-striped">
  <tr>
    <th>User ID</th>
    <th>Title</th>
    <th>Body</th>
  </tr>
  <tr *ngFor='let p of posts'>
```

```
        <td>{{ p.userId }}</td>
        <td>{{ p.title }}</td>
        <td>{{ p.body }}</td>
      </tr>
    </table>
    <hr>
```