About HTTP Service

## About Http Services

As the standards are going into next levels these days we are making use of HTTP calls to external API's. To do this angular has built-in HTTP library.

There are two ways for processing these asynchronous operations

- **Using Promises**
- **Using Observables**

It's recommended to use **http** calls within wrapped services instead of components which gives us more flexibility to our application. To use this service, we need to just import **(Http)** it and use it.

**HTTP** is used to communicate with the server, where most of the browsers supports two Http based requests (**XHR and JSONP**), and few browsers also supports **Fetch**

- **XML Http Request (XHR):** This request fetches data from server without refreshing page again.
- **JSONP:** This request is mainly used for cross domain requests i.e., calling a service of one domain from another domain.
- **Fetch:** This request fetches with promises.

- Modern browsers support two HTTP-based APIs: XMLHttpRequest (XHR) and JSONP. The Angular HTTP library simplifies application programming with the **XHR** and **JSONP** APIs.
- The Angular Http client communicates with the server using a familiar HTTP request/response protocol. The Http client is one of a family of services in the Angular HTTP library

**Walkthrough**

## Step 1: Setup Web Application project

1. Create a New Project: New →Project → ASP.NET Web Application → Web API → OK
2. Go to Tools → NuGet Package Manager → Manage Nuget Packages for Solution → Add reference to **EntityFramework.Dll**
3. Under Models folder add the following Employee and Context classes

```
public class Employee
{
   [Key]
```

```
    public int EmpId { get; set; }
    public string Name { get; set; }
    public decimal Salary { get; set; }
}
public class DemoContext : DbContext
{
    public DbSet<Employee> Employees { get; set; }
}
```

4. In Web.Config add the following

```
<connectionStrings>
    <add name="DemoContext" connectionString="Data Source=.\sqlexpress;Integrated
Security=True;database=DemoOrganization" providerName="System.Data.SqlClient"/>
  </connectionStrings>
```

5. Build the project
6. Right click on Controller Folder → Add → Controller. In the scaffolding option, select Web API 2 Controller with actions, using Entity Framework.
7. Model Class = Employee, Data Context Class = DemoContext → Add

   Note: EmployeeController would be added to the project. Essentially, we have created **Web API** to perform CRUD operations on the Employee table.

1. Right Click on Controller Folder → Add → Controller… → Select "Web API 2 Controller with actions, using Entity Framework" → Add
2. Select Model = Employee, DataContext Class = "DemoOrganziation" → Add

   Note: This generates Controller along with Views for Add/Edit/Delete operations.
3. Edit Global.asax: Application_Start method

```
GlobalConfiguration.Configure(WebApiConfig.Register);


//Following code is required to convert C# object to Camel Case Notation to be used in JavaScript / Angular
var formatters = GlobalConfiguration.Configuration.Formatters;
var jsonFormatter = formatters.JsonFormatter;
var settings = jsonFormatter.SerializerSettings;
settings.ContractResolver = new CamelCasePropertyNamesContractResolver();
```

## Step 2: Configuring Angular Module to use HTTP and Json Services.

Register providers by importing HttpModule and JsonModule to the root NgModule.

App.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms'

import { HttpModule, JsonpModule } from '@angular/http';

import { AppComponent } from './app.component';
import { EmployeesComponent } from './employees.component'

@NgModule({
  imports: [BrowserModule, FormsModule, HttpModule, JsonpModule],
  declarations: [AppComponent, EmployeesComponent],
  bootstrap: [AppComponent],
})
export class AppModule {
}
```

## Step 3: Fetch data with http.get

- In order to use the new Http module in our components we have to import **Http, Response, Headers, RequestOptions**.
- After that, we can just inject it via Dependency Injection in the constructor.
- ***http.get*** returns an ***Observable*** emitting ***Response*** objects.

```
http.get(url: string, options?: RequestOptionsArgs)
 : Observable<Response>
```

```
interface RequestOptionsArgs {
 url?: string;
 method?: string | RequestMethods;
 search?: string | URLSearchParams;
 headers?: Headers;
 body?: string;
}
```

**About RxJS (Reactive Extensions for JavaScript) library**

- Reactive Extensions for JavaScript (RxJS) is a reactive streams library that allows you to work with Observables.
- Using RxJS we represent asynchronous data streams with **Observables**, query asynchronous data streams using **LINQ operators** and parameterize the concurrency in the asynchronous data streams using **Schedulers**. **Rx = Observables + LINQ + Schedulers.**
- Angular 4 uses **Reactive Programming** as its core building block.

**Enable RxJS operators**

The RxJS library is large. Size matters when building a production application and deploying it to mobile devices. You should include only necessary features.

Each code file should add the operators it needs by importing from an RxJS library. The method needs the **map** and **catch** operators so it imports them like this.

```
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/catch';
import 'rxjs/add/operator/map';
```

We used ***map*** to parse the result into a JSON object. The result of ***map*** is also an ***Observable*** that emits a JSON object containing an Array of Employee.

The **catch** operator passes the error object from http to the handleError method.

**1. Add the following employee.service.ts to the project**

```
import { Injectable } from '@angular/core';
import { Http, Response, Headers, RequestOptions } from '@angular/http';

import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/catch';
import 'rxjs/add/operator/map';

import { Employee } from "./employee"

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

  getEmployees(): Observable<Employee[]> {
    return this.http.get("api/Employees")
```

```
        .map(this.extractData)
        .catch(this.handleError);
  }

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

  private handleError(error: Response | any) {
    let errMsg: string;
    if (error instanceof Response) {
      const body = error.json() || '';
      const err = body.error || JSON.stringify(body);
      errMsg = `${error.status} - ${error.statusText || ''} ${err}`;
    }
    else {
      errMsg = error.message ? error.message : error.toString();
    }
    console.error(errMsg);
    return Observable.throw(errMsg);
  }
}
```

2. Edit employee.component.cs as below

```
import { Component, OnInit } from '@angular/core';
import { Employee } from './employee';
import { EmployeeService } from "./employee.service"

@Component({
  moduleId: module.id,
  selector: 'employees',
  providers: [EmployeeService],
  template: `
    <style>
    table, th, td {
      border: 1px solid black;
    }
    </style>
    <table>
      <tr *ngFor="let emp of employees">
        <td>{{emp.id}}</td>
        <td>{{emp.empName }}</td>
        <td>{{emp.salary }}</td>
      <tr>
    </table>
  <hr>
<pre>{{newEmployee | json}}</pre>
`
})
export class EmployeesComponent {
  employees: Employee[]
  newEmployee: Employee = new Employee(0, "", 0);;
  errorMessage: string;
  constructor(private empService: EmployeeService) {
    //this.employees = empService.getEmployees();
  }
  ngOnInit() {
    this.getEmployees();
  }
  getEmployees() {
    this.empService.getEmployees().subscribe(
        emps => this.employees = emps,
        error => this.errorMessage = <any>error)
  }
}
```

Let's take a look at how we would use the new ***EmployeeService*** we just created to display a list of employees.

- We imported ***EmployeeService*** and in the constructor, we injected the service using the same syntax we saw before.
- During *App* instantiation, ***empService.getEmployees()*** will return an **Observable**. We want to display a list of Employees so we can use ***subscribe*** to set our local ***employees*** variable when the data is emitted. In this case, we just set the resulting Array into *this.employees.*

3. Run and test the application.

## **Step 4:** Sending data to Server / Adding Employee details using http.post

```
http.post(url: string, body: string, options?: RequestOptionsArgs)
 : Observable<Response>
```

1. Add the following method to employees.service.ts

```
  addEmployee(emp: Employee): Observable<Employee> {
     var strEmp = JSON.stringify(emp);
     let headers = new Headers({ 'Content-Type': 'application/json' });
     let options = new RequestOptions({ headers: headers });
     return this.http.post("api/Employees", strEmp, options)
         .map(this.extractData)
         .catch(this.handleError);
  }
```

2. Edit Employees.component.ts

**Insert the following HTML in template**

```
<form>
     Name: <input type="text" [(ngModel)]="newEmployee.empName" name="name" /> <br>
     Salary: <input type="text" [(ngModel)]="newEmployee.salary" name="salary" /> <br>
     <input type="button" value="Add Employee" name="addEmployee" (click)="addEmployee()"/>
</form>
```

**Add the following to the class**

```
 addEmployee() {
     alert("in component")
     this.empService.addEmployee(this.newEmployee).subscribe(
         emp => this.employees.push(emp),
         error => this.errorMessage = <any>error);
  }
```

3. Run and test the application by adding a new employee.