Angular HttpClient Tutorial & Example

7 Comments / 9 minutes of reading / January 29, 2022

← SetValidators Http Get Example →

In this Angular HttpClient Tutorial & Examples guide, we show you how to use HttpClient to make HTTP requests like GET & POST, etc. to the back end server. The Angular HTTP client module is introduced in the Angular 4.3. This new API is available in package @angular/common/http. It replaces the older HttpModule . The HTTP Client makes use of the RxJs Observables . The Response from the HttpClient is observable, hence it needs to be Subscribed. We will learn all these in this Tutorial.

Applies to: Angular 5 to the latest edition i.e. Angular 8, Angular 9. Angular 10, Angular 11

Table of Contents

Using Angular HttpClient

The `HttpClient` is a separate model in Angular and is available under the `@angular/common/http` package. The following steps show you how to use the `HttpClient` in an Angular app.

Import HttpClient Module in Root Module

We need to import it into our root module `app.module`. Also, we need to add it to the `imports` metadata array.

```
1
2  import { NgModule } from '@angular/core';
3  import { HttpClientModule } from '@angular/common/http';
4
5  @NgModule({
6     declarations: [
7        AppComponent
8     ],
9     imports: [
10       HttpClientModule
11    ],
12    providers: [],
13    bootstrap: [AppComponent]
14 })
```

```
15 export class AppModule { }
16
```

Import Required Module in Component/Service

Then you should import `HttpClient` the `@angular/common/http` in the component or service.

```
1
2 import { HttpClient } from '@angular/common/http';
3
```

Inject HttpClient service

Inject the HttpClient service in the constructor.

```
1
2 constructor(public http: HttpClient) {
3 }
4
```

Call the HttpClient.Get method

Use `HttpClient.Get` method to send an HTTP Request. The request is sent when we `Subscribe` to the `get()` method. When the response arrives map it the desired object and display the result.

```
1
2 public getData() {
3   this.HttpClient.get<any[]>(this.baseUrl+'users/'+this.userName+'/repos')
4         .subscribe(data => {
5            this.repos= data;
6         },
7         error => {
8         }
9   );
10 }
```

```
11
```

What is Observable?

The Angular `HTTPClient` makes use of observable. Hence it is important to understand the basics of it

Observable help us to manage async data. You can think of Observables as an array of items, which arrive asynchronously over time.

The `observables` implement the observer design pattern, where `observables` maintain a list of dependents. We call these dependents as `observers` . The `observable` notifies them automatically of any state changes, usually by calling one of their methods.

`Observer` subscribes to an `Observable` . The `observer` reacts when the value of the `Observable` changes. An `Observable` can have multiple subscribers and all the subscribers are notified when the state of the `Observable` changes.

When an Observer subscribes to an observable, it needs to pass (optional) the three callbacks. `next()` , `error()` & `complete()` . The observable invokes the `next()` callback, when it receives an value. When the observable completes it invokes the `complete()`

callback. And when the error occurs it invokes the `error()` callback with details of error and subscriber finishes.

The `Observables` are used extensively in Angular. The new `HTTPClient Module` and Event system are all Observable based.

The Observables are proposed feature for the next version of Javascript. The Angular uses a Third-party library called Reactive Extensions or RxJs to implement the Observables. You can learn about RxJs from these RxJx tutorials

Observables Operators

Operators are methods that operate on an Observable and return a new observable. Each Operator modifies the value it receives. These operators are applied one after the other in a chain.

The `RxJs` provides several Operators, which allows you to filter, select, transform, combine and compose Observables. Examples of Operators are `map`, `filter`, `take`, `merge`, etc

How to use RxJs

The `RxJs` is a very large library. Hence Angular exposes a stripped-down version of Observables. You can import it using the following import statement

```
1
2 import { Observable } from 'rxjs';
3
```

The above import imports only the necessary features. It does not include any of the Operators.

To use observables operators, you need to import them. The following code imports the `map` & `catchError` operators.

```
import { map, catchError } from 'rxjs/operators';
```

HTTP GET

The `HttpClient.get` sends the HTTP Get Request to the API endpoint and parses the returned result to the desired type. By default, the body of the response is parsed as JSON. If you want any other type, then you need to specify explicitly using the `observe` & `responseType` options.

You can read more about Angular HTTP Get

Syntax

```
get(url: string,
    options: {
        headers?: HttpHeaders | { [header: string]: string | string[]; };
        params?: HttpParams | { [param: string]: string | string[]; };
        observe?: "body|events|response|";
        responseType: "arraybuffer|json|blob|text";
        reportProgress?: boolean;
```

```
9         withCredentials?: boolean;}
10    ): Observable<>
11
```

Options

under the options, we have several configuration options, which we can use to configure the request.

`headers` It allows you to add HTTP headers to the outgoing requests.

`observe` The `HttpClient.get` method returns the body of the response parsed as JSON (or type specified by the `responseType`). Sometimes you may need to read the entire response along with the headers and status codes. To do this you can set the `observe` property to the **response**.

The allowed options are

- a `response` which returns the entire response
- `body` which returns only the body
- `events` which return the response with events.

`params` Allows us to Add the URL parameters or Get Parameters to the Get Request

`reportProgress` This is a boolean property. Set this to `true`, if you want to get notified of the progress of the Get Request. This is a pretty useful feature when you have a large amount of data to download (or upload) and you want the user to notify of the progress.

`responseType` `Json` is the default response type. In case you want a different type of response, then you need to use this parameter. The Allowed Options are `arraybuffer`, `blob`, `JSON`, and `text`.

`withCredentials` It is of boolean type. If the value is true then `HttpClient.get` will request data with credentials (cookies)

HTTP Post

The `HttpClient.post()` sends the HTTP POST request to the endpoint. Similar to the get(), we need to subscribe to the post() method to send the request. The post method parsed the `body of the response` as `JSON` and returns it. This is the default behavior. If you want any other type, then you need to specify explicitly using the `observe` & `responseType` options.

You can read Angular HTTP Post

The syntax of the HTTP Post is similar to the HTTP Get.

```
1
2 post(url: string,
3     body: any,
4     options: {
```

```
      headers?: HttpHeaders | { [header: string]: string | string[]; };
      observe?: "body|events|response|";
      params?: HttpParams | { [param: string]: string | string[]; };
      reportProgress?: boolean;
      responseType: "arraybuffer|json|blob|text";
      withCredentials?: boolean;
    }
): Observable
```

The following is an example of HTTP Post

```
addPerson(person:Person): Observable<any> {
   const headers = { 'content-type': 'application/json'}
   const body=JSON.stringify(person);
   this.http.post(this.baseURL + 'people', body,{'headers':headers , observe: 'response'})
    .subscribe(
     response=> {
        console.log("POST completed sucessfully. The response received "+response);
     },
     error => {
        console.log("Post failed with the errors");
     },
     () => {
        console.log("Post Completed");
     }
}
```

HTTP PUT

The HttpClient.put() sends the HTTP PUT request to the endpoint. The syntax and usage are very similar to the HTTP POST method.

```
put(url: string,
    body: any,
    options: {
      headers?: HttpHeaders | { [header: string]: string | string[]; };
      observe?: "body|events|response|";
```

```
      params?: HttpParams | { [param: string]: string | string[]; };
      reportProgress?: boolean;
      responseType: "arraybuffer|json|blob|text";
      withCredentials?: boolean;
    }
): Observable
```

HTTP PATCH

The HttpClient.patch() sends the HTTP PATCH request to the endpoint. The syntax and usage are very similar to the HTTP POST method.

```
patch(url: string,
    body: any,
    options: {
      headers?: HttpHeaders | { [header: string]: string | string[]; };
      observe?: "body|events|response|";
      params?: HttpParams | { [param: string]: string | string[]; };
      reportProgress?: boolean;
      responseType: "arraybuffer|json|blob|text";
      withCredentials?: boolean;
    }
): Observable
```

HTTP DELETE

The HttpClient.delete() sends the HTTP DELETE request to the endpoint. The syntax and usage are very similar to the HTTP GET method.

```
delete(url: string,
     options: {
        headers?: HttpHeaders | { [header: string]: string | string[]; };
        params?: HttpParams | { [param: string]: string | string[]; };
        observe?: "body|events|response|";
        responseType: "arraybuffer|json|blob|text";
        reportProgress?: boolean;
        withCredentials?: boolean;}
```

```
    ): Observable<>
```

HttpClient Example

Now, We have a basic understanding of `HttpClient` model & observables, let us build an `HttpClient` example app.

Create a new Angular app

```
ng new httpClient
```

import HttpClientModule

In the `app,module.ts` import the `HttpClientModule` module as shown below. We also add it to the `imports` array

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

```
22
```

Component

Now, open the app.component.ts and copy the following code.

```
1
2  import { Component, OnInit } from '@angular/core';
3  import { HttpClient } from '@angular/common/http';
4
5  export class Repos {
6    id: string;
7    name: string;
8    html_url: string;
9    description: string;
10 }
11
12 @Component({
13   selector: 'app-root',
14   templateUrl: './app.component.html',
15 })
16 export class AppComponent implements OnInit {
17
18   userName: string = "tektutorialshub"
19   baseURL: string = "https://api.github.com/";
20   repos: Repos[];
21
22
23   constructor(private http: HttpClient) {
24   }
25
26   ngOnInit() {
27     this.getRepos()
28   }
29
30
31   public getRepos() {
32
33     return this.http.get<Repos[]>(this.baseURL + 'users/' + this.userName + '/repos')
34       .subscribe(
35         (response) => {                    //Next callback
36           console.log('response received')
37           console.log(response);
38           this.repos = response;
39         },
40         (error) => {                       //Error callback
```

```
41          console.error('Request failed with error')
42          alert(error);
43        },
44        () => {                          //Complete callback
45          console.log('Request completed')
46        })
47    }
48  }
49
```

Import HTTPClient

HTTPClient is a service, which is a major component of the HTTP Module. It contains methods like `GET`, `POST`, `PUT` etc. We need to import it.

```
1
2  import { HttpClient } from '@angular/common/http';
3
```

Repository Model

The model to handle our data.

```
1
2  export class Repos {
3    id: string;
4    name: string;
5    html_url: string;
6    description: string;
```

```
}

```

Inject HttpClient

Inject the `HttpClient` service into the component. You can learn more about Dependency injection in Angular

```

  constructor(private http: HttpClient) {
  }

```

Subscribe to HTTP Get

The `GetRepos` method, we invoke the `get()` method of the `HttpClient` Service.

The `HttpClient.get` method allows us to cast the returned response object to a type we require. We make use of that feature and supply the type for the returned value `http.get<repos[]>`

The `get()` method returns an observable. Hence we `subscribe` to it.

```

public getRepos() {
  return this.http.get<Repos[]>(this.baseURL + 'users/' + this.userName + '/repos')
    .subscribe(

```

When we subscribe to any observable, we optionally pass the three callbacks. `next()`, `error()` & `complete()`. In this example we pass only two callbacks `next()` & `error()`.

Receive the Response

We receive the data in the `next()` callback. By default, the Angular reads the `body of the response` as JSON and casts it to an object and returns it back. Hence we can use directly in our app.

```
      (response) => {                    //Next callback
        console.log('response received')
        console.log(response);
        this.repos = response;
      },
```

Handle the errors

We handle the errors in `error` callback.

```
      (error) => {                       //Error callback
        console.error('Request failed with error')
        alert(error);
      },
```

Template

```
<h1 class="heading"><strong>HTTPClient </strong> Example</h1>


<table class='table'>
  <thead>
    <tr>
      <th>ID</th>
      <th>Name</th>
      <th>HTML Url</th>
      <th>description</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let repo of repos;">
      <td>{{repo.id}}</td>
```