

Angular HTTP GET Example using httpclient

8 Comments / 9 minutes of reading / May 16, 2021

← [HTTP Client Tutorial](#)

[Http Post Example](#) →

This guide explains how to make HTTP GET requests using the `HttpClient` module in Angular. The Angular introduced the `HttpClient` Module in Angular 4.3. It is part of the package `@angular/common/http`. In this tutorial, let us build an HTTP GET example app, which sends the HTTP Get request to GitHub repository using the [GitHub API](#).

Table of Contents

[HTTP Get Example](#)

[Import HttpClientModule](#)

[Model](#)

[HTTP GET Service](#)

[Component](#)

[Template](#)

[HTTP Get in Action](#)

[Get Syntax](#)

[observe](#)

[Complete Response](#)

[events](#)

[Response Type](#)

[Strongly typed response](#)

[String as Response Type](#)[Catching Errors](#)[Transform the Response](#)[URL Parameters](#)[HTTP Headers](#)[Send Cookies](#)[Summary](#)

HTTP Get Example

Create a new Angular App.

```
1  
2 ng new httpGet  
3
```

Import HttpClientModule

To make HTTP Get request, we need to make use of the [HttpClientModule](#), which is part of the package `@angular/common/http`. Open the `app.module.ts` and import it. Also, import the `FormsModule`

You must also include it in the `imports` array as shown below.

```
1  
2 import { BrowserModule } from '@angular/platform-browser';  
3 import { NgModule } from '@angular/core';  
4 import { HttpClientModule } from '@angular/common/http';  
5 import { FormsModule } from '@angular/forms';  
6  
7 import { AppRoutingModule } from './app-routing.module';  
8 import { AppComponent } from './app.component';  
9 import { GitHubService } from './github.service';  
10  
11 @NgModule({  
12   declarations: [  

```

```
13   AppComponent
14 ],
15 imports: [
16   BrowserModule,
17   AppRoutingModule,
18   HttpClientModule,
19   FormsModule
20 ],
21 providers: [GitHubService],
22 bootstrap: [AppComponent]
23 })
24 export class AppModule { }
25
```

Model

Create `repos.ts` file and add the following code. This is a simplified model for the GitHub repository.

```
1
2 export class repos {
3   id: string;
4   name: string;
5   html_url: string;
6   description: string;
7 }
8
```

HTTP GET Service

Let us create a service to handle the HTTP Request. Create a new file `github.service.ts` and copy the following code.

```
1
2 import { Injectable } from '@angular/core';
3
4 import { HttpClient, HttpParams, HttpHeaders } from '@angular/common/http';
5 import { Observable, throwError } from 'rxjs';
6 import { map, catchError } from 'rxjs/operators';
7
8 import { repos } from './repos';
```

```
9
10 @Injectable()
11 export class GitHubService {
12
13     baseUrl: string = "https://api.github.com/";
14
15     constructor(private http: HttpClient) {
16     }
17
18     getRepos(userName: string): Observable<any> {
19         return this.http.get(this.baseUrl + 'users/' + userName + '/repos')
20     }
21 }
22
23
```

First, we import the required libraries. The HttpClient is the main service, which Performs the HTTP requests like GET , PUT , POST , etc. We need to inject this into our GitHubService . Also, import HttpParams which helps us to [add Query Parameters in an HTTP Request](#). Import HTTP Headers using the HttpHeaders which allows us to add HTTP Headers to the request.

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

The HttpClient service makes use of RxJs observable, Here we import Observable , throwError & RxJs Operators like map & catchError

```
1
2 import { Observable, throwError } from 'rxjs';
3 import { map, catchError } from 'rxjs/operators';
4
```

The URL endpoint is hardcoded in our example, But you can make use of a [config file](#) to store the value and read it using the [APP_INITIALIZER](#) token

```
1  
2  baseUrl: string = "https://api.github.com/";  
3
```

We inject the HttpClient using the [Dependency Injection](#)

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

Finally, we use the get method of the httpclient to make an HTTP Get request to GitHub.

The https://api.github.com/users/<username>?repos endpoint returns the list of Repositories belonging to the user <userName>

```
1  
2  //Any Data Type  
3  getRepos(userName: string): Observable<any> {  
4    return this.http.get(this.baseUrl + 'users/' + userName + '/repos')  
5  }  
6
```

Note that httpclient.get method returns the observable . Hence we need to subscribe to it to get the data.

Component

The following is the code from app.component.ts

```
1
2 import { Component } from '@angular/core';
3
4 import { GitHubService } from './github.service';
5 import { repos } from './repos';
6
7 @Component({
8   selector: 'app-root',
9   templateUrl: './app.component.html',
10 })
11 export class AppComponent {
12   userName: string = "tektutorialshub"
13   repos: repos[];
14
15   loading: boolean = false;
16   errorMessage;
17
18   constructor(private githubService: GitHubService) {
19   }
20
21   public getRepos() {
22     this.loading = true;
23     this.errorMessage = "";
24     this.githubService.getRepos(this.userName)
25       .subscribe(
26         (response) => { //next() callback
27           console.log('response received')
28           this.repos = response;
29         },
30         (error) => { //error() callback
31           console.error('Request failed with error')
32           this.errorMessage = error;
33           this.loading = false;
34         },
35         () => { //complete() callback
36           console.error('Request completed') //This is actually not needed
37           this.loading = false;
38         })
39   }
40 }
41
```

We subscribe to the `getRepos()` method in our component class. Only when we subscribe to the observable, the HTTP GET request is sent to the back end server.

```
1  
2 this.githubService.getRepos(this.userName)  
3   .subscribe();  
4
```

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

`Next()` callback is where we get the result of the observable. In this example the list of repositories for the given user.

```
1  
2 (response) => { //next() callback  
3   console.log('response received')  
4   this.repos = response;  
5 }  
6
```

The observable can also result in an error. It will invoke the `error()` callback and pass the error object. The observables stop after emitting the error signal.

```

1
2      (error) => {                                //error() callback
3          console.error('Request failed with error')
4          this.errorMessage = error;
5          this.loading = false;
6      },
7

```

When the observable completes, it will call the `complete()` callback. **There is no need for this call back as the subscription completes when the data is received.**

```

1
2      () => {                                    //complete() callback
3          console.log('Request completed')
4          this.loading = false;
5      })
6

```

Loading Indicator

We create a variable `loading=true` just before subscribing to the GET request. When the observable completes or an error occurs, we make it `false`. This helps us to show some kind of loading indicator to users, while we wait for the response.

Template

```

1
2 <h1 class="heading"><strong>HTTP </strong>Demo</h1>
3
4 <div class="form-group">
5   <label for="userName">GitHub User Name</label>
6   <input type="text" class="form-control" name="userName" [(ngModel)]="userName">
7 </div>
8
9 <div class="form-group">
10  <button type="button" (click)="getRepos()">Get Repos</button>
11 </div>
12
13 <div *ngIf="loading">loading...</div>
14

```



```
15 <div *ngIf="errorMessage" class="alert alert-warning">
16   <strong>Warning!</strong> {{errorMessage | json}}
17 </div>
18
19
20 <table class='table'>
21   <thead>
22     <tr>
23       <th>ID</th>
24       <th>Name</th>
25       <th>HTML Url</th>
26       <th>description</th>
27     </tr>
28   </thead>
29   <tbody>
30     <tr *ngFor="let repo of repos;">
31       <td>{{repo.id}}</td>
32       <td>{{repo.name}}</td>
33       <td>{{repo.html_url}}</td>
34       <td>{{repo.description}}</td>
35     </tr>
36   </tbody>
37 </table>
38
39 <pre>{{repos | json}}</pre>
40
```

The template is very simple

We first ask for the `userName`. We use the two-way data binding to sync `userName` `[(ngModel)]="userName"` with the `userName` property in the component class.

```
1
2 <div class="form-group">
3   <label for="userName">GitHub User Name</label>
4   <input type="text" class="form-control" name="userName" [(ngModel)]="userName">
5 </div>
6
```

`getRepos()` method subscribes to the HTTP get method.

```
1  
2 <div class="form-group">  
3   <button type="button" (click)="getRepos()">Get Repos</button>  
4 </div>  
5
```

We show a loading message until the observable returns response or an error.

```
1  
2 <div *ngIf="loading">loading...</div>  
3
```

Show the error message.

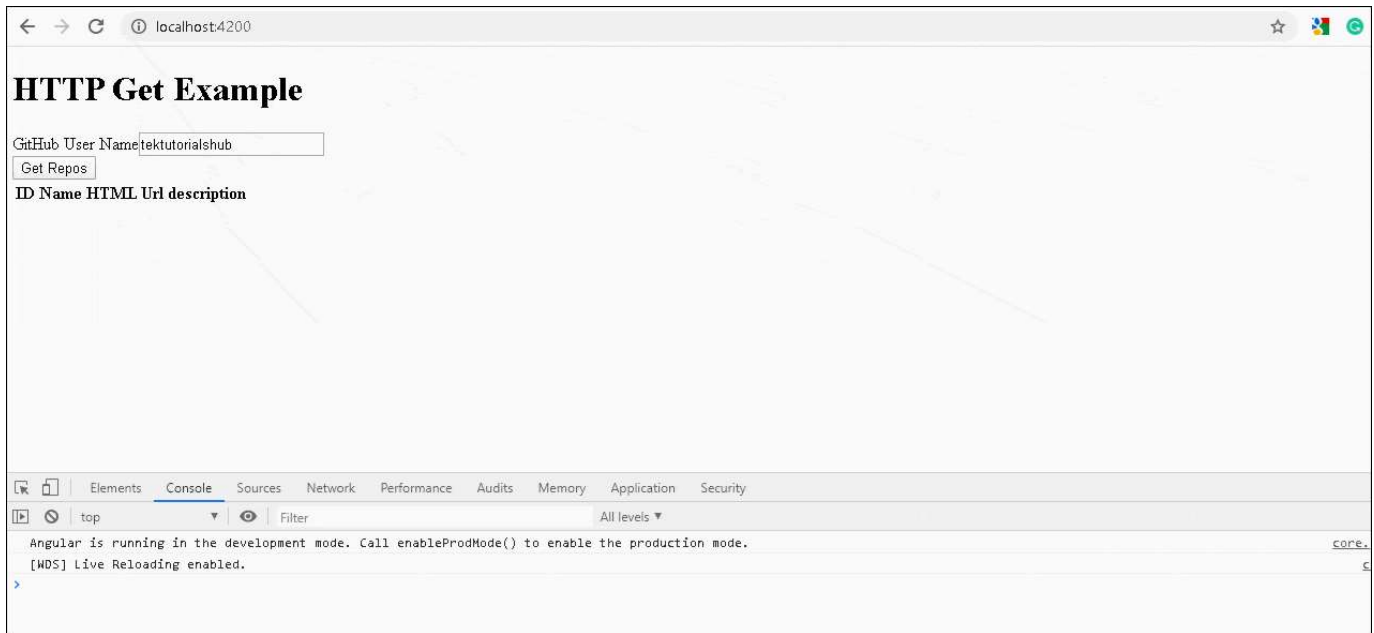
```
1  
2 <div *ngIf="errorMessage" class="alert alert-warning">  
3   <strong>Warning!</strong> {{errorMessage | json}}  
4 </div>  
5
```

The last line shows the response as it is received.

```
1  
2 <pre>{{repos | json}}</pre>  
3
```

HTTP Get in Action

Now, run the app, you should be able to make a successful GET Request.



Get Syntax

The above code is a very simple example of the HTTP `get()` method. The complete syntax of the `get()` method is as shown below. It has second argument `options`, where we can pass the HTTP headers, parameters, and other options to control how the `get()` method behaves.

```
1
2 get(url: string,
3     options: {
4         headers?: HttpHeaders | { [header: string]: string | string[]; };
5         params?: HttpParams | { [param: string]: string | string[]; };
6         observe?: "body|events|response";
7         responseType: "arraybuffer|json|blob|text";
8         reportProgress?: boolean;
9         withCredentials?: boolean; }
10 ): Observable<>
11
```

- `headers` : use this to send the HTTP Headers along with the request
- `params`: set query strings / URL parameters

- observe: This option determines the return type.
- responseType: The value of responseType determines how the response is parsed.
- reportProgress: Whether this request should be made in a way that exposes [progress events](#).
- withCredentials: Whether this request should be sent with outgoing credentials (cookies).

observe

The GET method returns one of the following

1. Complete response
2. body of the response
3. [events](#).

By default, it returns the `body` as shown in our example app.

Complete Response

The following code will return the complete response and not just the body

```
1
2 getReposRawResponse(userName: string): Observable<any> {
3   return this.http.get(this.baseURL + 'users/' + userName + '/repos', { observe: 'response' })
4 }
5
```

The complete response is as follows.

```
1
2 {
3   "headers": {
```

```

4   "normalizedNames": {},
5   "lazyUpdate": null
6 },
7 "status": 200,
8 "statusText": "OK",
9 "url": "https://api.github.com/users/tektutorialshub/repos",
10 "ok": true,
11 "type": 4,
12 "body": [
13   {
14     "id": 102269857,
15
16     ***** Removed for clarity *****
17   }
18 ]
19 }
20

```

events

You can also listen to progress events by using the

{ observe: 'events', reportProgress: true }. You can read about [observe the response](#)

```

1
2 getReposRawResponse(userName: string): Observable<any> {
3   return this.http.get(this.baseURL + 'users/' + userName + '/repos', { observe: 'events',
4 }
5

```

Response Type

The responseType determines how the response is parsed. it can be one of the arraybuffer, json blob or text. The default behavior is to parse the response as JSON.

Strongly typed response

Instead of any, we can also use a type as shown below

```

1

```

```
2  getReposTypedResponse(userName: string): Observable<repos[]> {  
3    return this.http.get<repos[]>(this.baseUrl + 'users/' + userName + '/repos')  
4  }  
5
```

String as Response Type

The API may return a simple text rather than a JSON. Use `responsetype: 'text'` to ensure that the response is parsed as a string.

```
1  
2  getReposTypedResponse(userName: string): Observable<repos[]> {  
3    return this.http.get<repos[]>(this.baseUrl + 'users/' + userName + '/repos', { respon  
4  }  
5
```

Catching Errors

The API might fail with an error. You can catch those errors using `catchError`. You either handle the error or throw it back to the component using the `throw err`

```
1  
2  getReposCatchError(userName: string): Observable<repos[]> {  
3    return this.http.get<repos[]>(this.baseUrl + 'usersY/' + userName + '/repos')  
4      .pipe(  
5        catchError((err) => {  
6          console.error(err);  
7          throw err;  
6
```

```
8      }  
9    )  
10  )  
11  }  
12
```

Read more about error handling from [Angular HTTP interceptor error handling](#)

Transform the Response

You can make use of the `map`, `filter` RxJs Operators to manipulate or transform the response before sending it to the component.

```
1  
2  getReposMap(userName: string): Observable<repos[]> {  
3    return this.http.get<repos[]>(this.baseUrl + 'users/' + userName + '/repos')  
4      .pipe(  
5        map((data) => {  
6          //You can perform some transformation here  
7          return data;  
8        } ),  
9        catchError((err, caught) => {  
10         console.error(err);  
11         throw err;  
12       })  
13      )  
14    )  
15  }  
16  
17
```

URL Parameters

The [URL Parameters or Query strings](#) can be added to the request easily using the [HttpParams](#) option. All you need to do is to create a new `HttpParams` class and add the parameters as shown below.

```
1  
2  //URL Parameter
```

```
3  getRepoUrlParameter(userName: string): Observable<repos[]> {
4
5      const params = new HttpParams()
6          .set('sort', "description")
7          .set('page', "2");
8
9      return this.http.get<repos[]>(this.baseUrl + 'users/' + userName + '/repos', { 'params': params })
10     .pipe(
11         map((response) => {
12             return response;
13         }),
14         catchError((err, caught) => {
15             console.error(err);
16             throw err;
17         })
18     )
19 }
20 }
21 }
```

The above code sends the GET request to the URL

<https://api.github.com/users/tektutorialshub/repos?sort=description&page=2>

The following code also works.

```
1
2  getRepoUrlParameter(userName: string): Observable<repos[]> {
3      return this.http.get<repos[]>(this.baseUrl + 'users/' + userName + '/repos?sort=description&page=2')
4      .pipe(
5          map((response) => {
6              return response;
7          }),
8          catchError((err, caught) => {
9              console.error(err);
10             throw err;
11          })
12      )
13  }
14 }
15 }
```

HTTP Headers

You can also add HTTP Headers using the `HttpHeaders` option as shown below. You can make use of the [Http Interceptor to set the common headers](#).

```
1
2 //HTTP Headers
3 getReposHeaders(userName: string): Observable<repos[]> {
4
5     const params = new HttpParams()
6         .set('sort', "description")
7         .set('page', "2");
8
9     const headers = new HttpHeaders()
10        .set('Content-Type', 'application/json')
11
12
13     return this.http.get<repos[]>(this.baseUrl + 'users/' + userName + '/repos', { 'params': params, 'headers': headers })
14        .pipe(
15            map((response) => {
16                return response;
17            }),
18            catchError((err, caught) => {
19                console.error(err);
20                throw err;
21            })
22        )
23    }
24 }
25
```

Send Cookies

You can send cookies with every request using the `withCredentials=true` as shown below. You can make use of the [Http Interceptor](#) to set the `withCredentials=true` for all requests.

```
1
2 //With Credentials
3 getReposWithCookies(userName: string): Observable<repos[]> {
4
5     const params = new HttpParams()
6         .set('sort', "description")
7
```

```
7      .set('page','2");
8
9      const headers = new HttpHeaders()
10         .set('Content-Type', 'application/json')
11
12
13      return this.http.get<repos[]>(this.baseUrl + 'users/' + userName + '/repos', { 'par
14         .pipe(
15             map((response) => {
16                 return response;
17             })),
18             catchError((err, caught) => {
19                 console.error(err);
20                 throw err;
21             })
22         )
23     )
24 }
25
```

Summary

This guide explains how to make use of HTTP get in Angular using an example app. In the next tutorial, we will look at the HTTP post method.

[← HTTP Client Tutorial](#)[Http Post Example →](#)

Related Posts