# Angular HttpClient

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.

Steps to include HttpClient:

1)Import HttpClient Module in Root Module

import { HttpClientModule } from '@angular/common/http';

 imports: [

        HttpClientModule

    ]

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

import { HttpClient } from '@angular/common/http'

### 3) Inject HttpClient service inside constructor

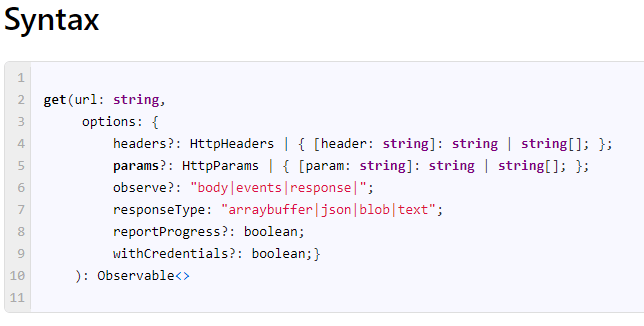
constructor(public http: HttpClient) {}

Methods:

## HTTP GET, HTTP DELETE

## Are similar in syntax.

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.



#### 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](https://www.tektutorialshub.com/angular/angular-pass-url-parameters-query-strings/) to the Get Request

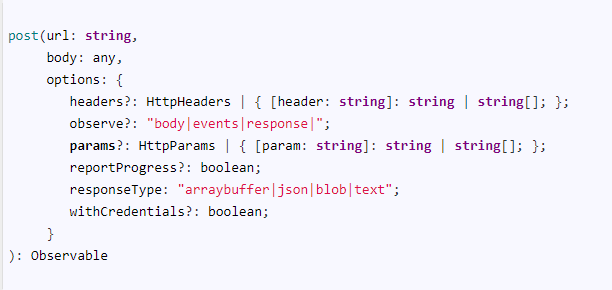
reportProgress This is a boolean property. Set this to true, if you want to get notified of the progress of the Get Request. This is a pretty useful feature when you have a large amount of data to download (or upload) and you want the user to notify of the progress.

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)

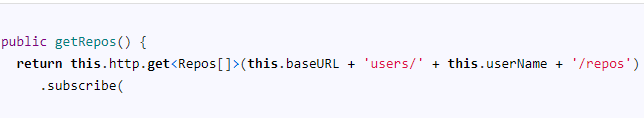
## HTTP Post, HTTP PUT , HTTP PATCH

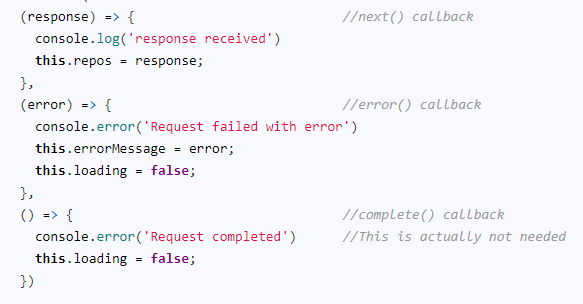
## Three are similar and follows same structure



#### Subscribe to HTTP  Methods:

Example GET method:





The HttpClient.get method allows us to cast the returned response object to a type we require.

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.(as shown as above) By default, the Angular reads the body of the response as JSON and casts it to an object here <Repos[]>and returns it back. Hence we can use directly in our app.

# URL Parameters, Query Parameters, httpparams, name Query strings, Get Params,  in Angular HttpClient

# The HttpParms were known as URLSearchParams

# 

# after the question mark is called **URL Parameter** or **Query strings /Query Parameters**. The Question mark is used as a separator.

## Without params

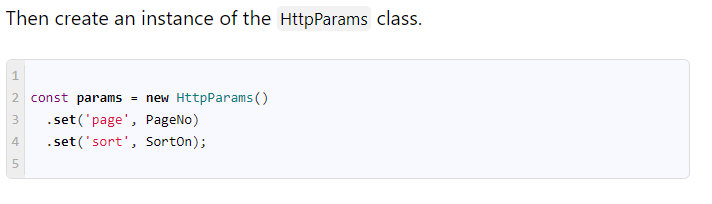
You can also add the parameters directly to the URL, without going through the HttpParams as shown above.

## HttpParams()

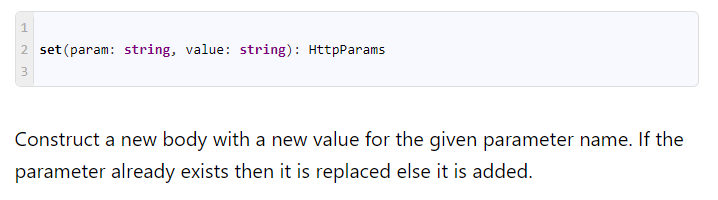
We add the URL parameters using the helper class HttpParams.

Steps to use: (component or service)





### HttpParams.set



Usage:

return this.httpClient.get<repos[]>(this.baseURL + 'users/' + userName + '/repos',{params})

#### HTTPParams is immutable

The HttpParams object is immutable. Every time you call a set method on Params object, it will create and return a new instance of the Params.

For Example

The following code does not work



Each call to set method does not add the options to the existing HttpParams instance, but creates a new instance from the existing instance and returns it.

To work around, you can use the code as follows



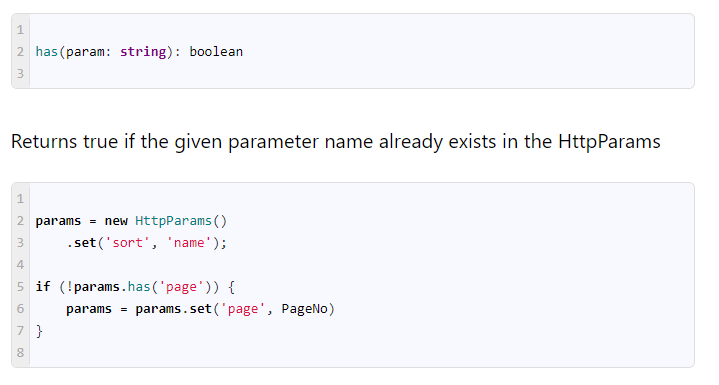
(Or)



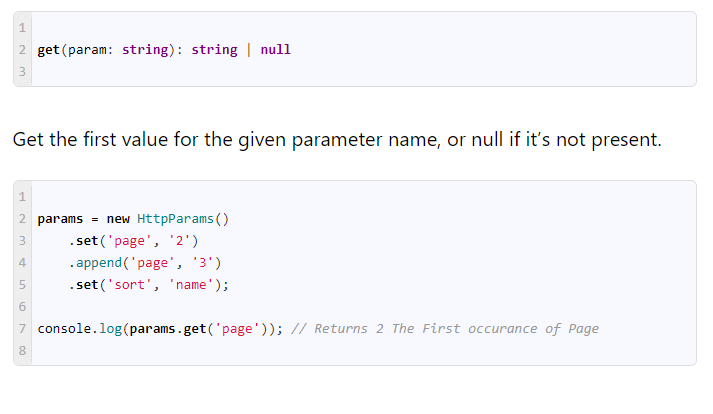
### HttpParams.append

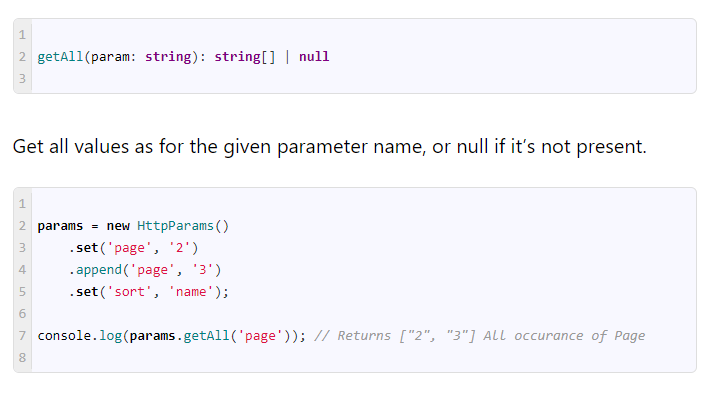
### HttpParams.has



### HttpParams.get



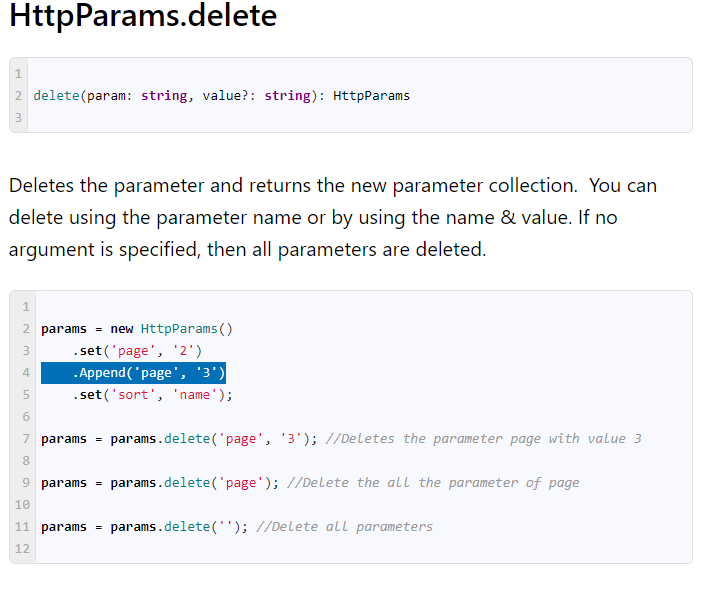
### HttpParams.getAll



### HttpParams.keys



### HttpParams.delete



### HttpParams.toString



# Angular HTTPHeaders

how to add HTTP Headers to an HTTP request in Angular. There are two ways by which we can add the headers. One, we add the HTTP Headers while making a request. The second way is to use the [HTTP interceptor](https://www.tektutorialshub.com/angular/angular-httpclient-http-interceptor/) to intercept all the Requests and add the Headers. In both cases, we use the httpHeaders configuration option provided by angular [HttpClient](https://www.tektutorialshub.com/angular/angular-httpclient/) to add the headers.

[HTTP Headers](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers) let the client and the server share the additional information about the HTTP request or response. For example, we use the [content-type header](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type) to indicate the [media type](https://developer.mozilla.org/en-US/docs/Glossary/MIME_type) of the resource like JSON, text, blob, etc. Another important header is where you send the bearer token using the Authorization header 'Authorization', 'Bearer <yourTokenhere>'

## HttpHeaders

We add HTTP Headers using the [HttpHeaders](https://angular.io/api/common/http/HttpHeaders)helper class. It is passed as one of the arguments to the GET, POST, PUT, DELETE, PATCH & OPTIONS request.



## HttpHeaders from object

The following code shows how you can create a HttpHeaders from an object.

let headers = new HttpHeaders({ 'Access-Control-Allow-Origin': '\*','content-type': 'application/json'}  )

console.log(headers)

### set

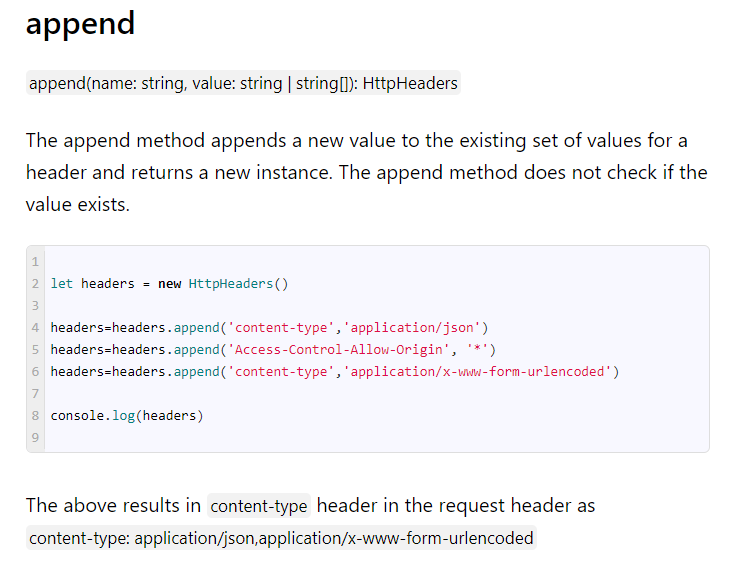
set(name: string, value: string | string[]): HttpHeaders

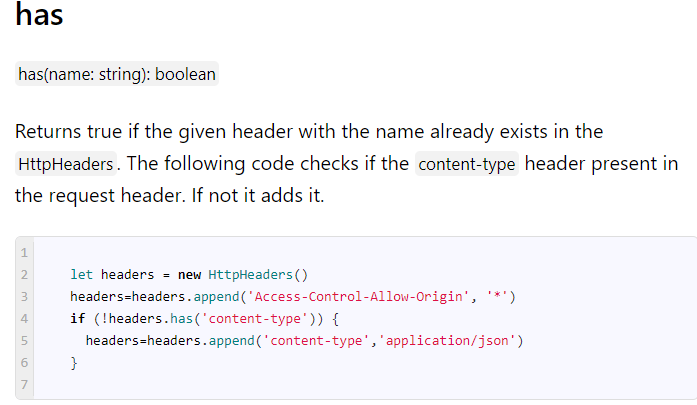
The Sets method returns a new instance after modifying the given header. If the header already exists, its value is replaced with the given value in the returned object.

#### httpHeaders are immutable

The HTTP headers are immutable. The following example does not work as each set method returns a new header and does not update the original header.



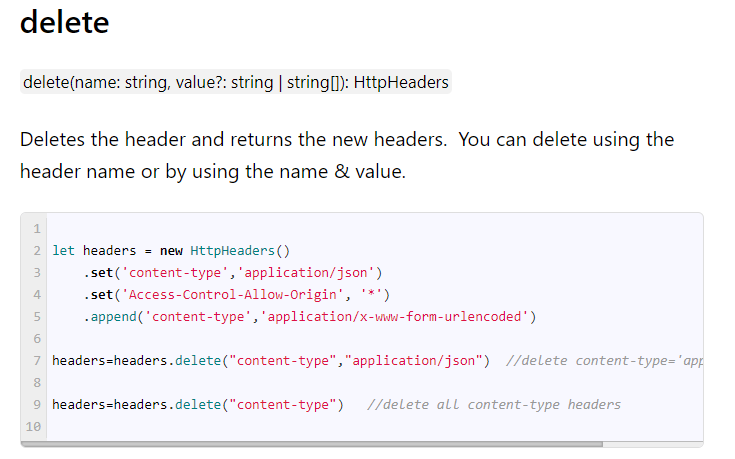












# Understanding HTTP Interceptors in Angular

The Angular HTTP Interceptor is introduced along with the new [HTTPClientModule](https://www.tektutorialshub.com/angular/angular-httpclient/).

## What is angular Http interceptor

The Angular HTTP interceptors sit between our application and the backend. When the application makes a request, the interceptor catches the request before it is sent to the backend. By Intercepting requests, we will get access to request headers and the body. This enables us to transform the request before sending it to the Server.

When the response arrives from the back end the Interceptors can transform it before passing it to our application.

One of the main benefits of the Http Interceptors is to add the Authorization Header to every request. We could do this manually, but that is a lot of work and error-prone. Another benefit is to catch the errors generated by the request and log them.

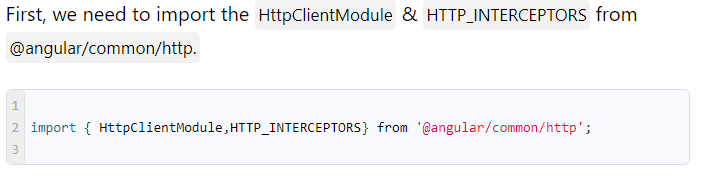
## How to Create Http Interceptor

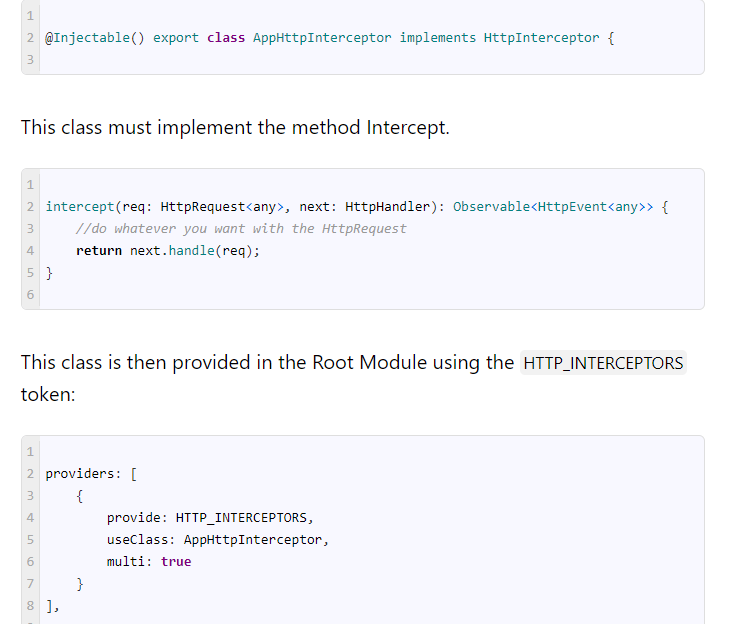
To Implement the Interceptor, you need to create an [injectable service](https://www.tektutorialshub.com/angular/angular-dependency-injection/), which implements the HttpInterceptorinterface.

At the heart of the Interceptor, logic is the [HttpInterceptor Interface](https://angular.io/api/common/http/HttpInterceptor). we must Implement it in our Interceptor Service.

You can define more than one Interceptor. The Interceptors are called in the order they are defined in [provider metadata](https://www.tektutorialshub.com/angular/angular-providers/).

In app module:





### HttpRequest

The first argument is [HttpRequest](https://angular.io/api/common/http/HttpRequest).

The HttpRequest is an outgoing HTTP request which is being intercepted. It contains URL, method, headers, body, and other request configuration.

The HttpRequest is a immutable class. Which means that we can’t modify the original request. To make changes we need to clone the Original request using the HttpRequest.clone method

### HttpHandler

The second argument is [httpHandler](https://angular.io/api/common/http/HttpHandler)

The HttpHandler dispatches the HttpRequest to the next Handler using the method HttpHandler.handle. The next handler could be another Interceptor in the chain or the Http Backend.

## Create the Interceptor



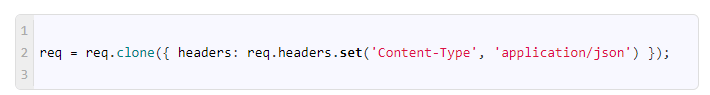
In the method body, you can modify the HttpRequest object. Once done, you can call the HttpHandler.handle method of the HttpHandler with the HttpRequest object. The HttpHandler.handle method invokes the next interceptor or sends the request to the backend server.

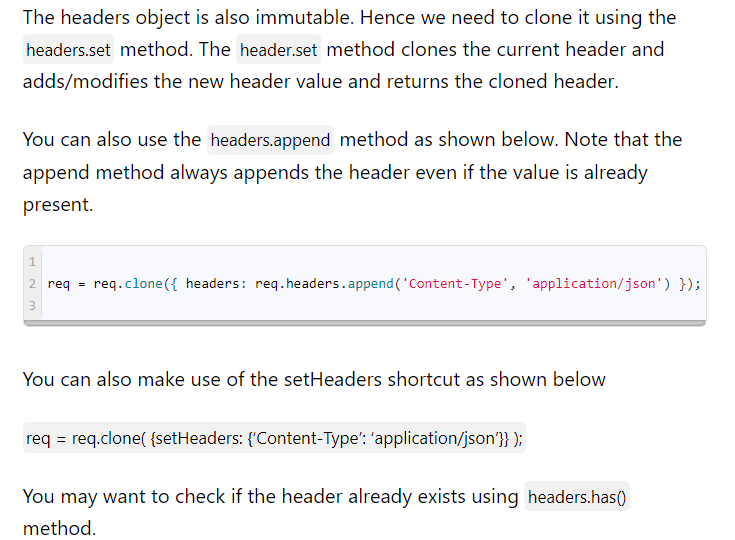
## Setting the new headers

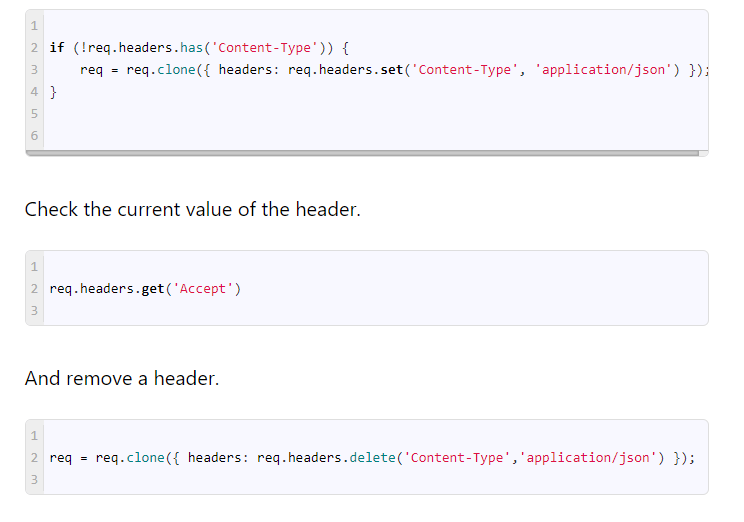
We are able to Intercept the request and log it to the console in the above example. Now we will modify the HTTP Headers and Custom Headers.

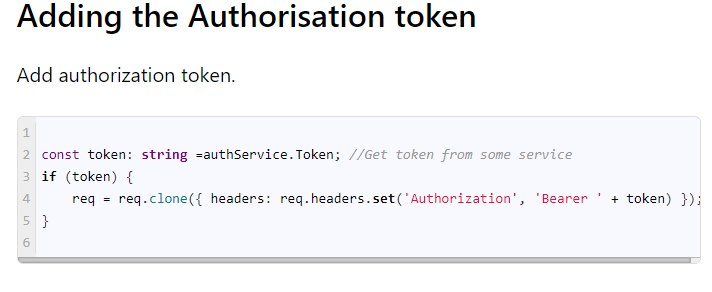
### Adding the Content-Type

To Modify the request we need to clone it. The HttpRequest.clone method allows us to modify the specific properties of the request while copying others. In the following example we are adding the new header content-type to the request.









## Intercepting the Response

The response of the back-end server can be intercepted using the various Rxjs Operators. The [map](https://www.learnrxjs.io/learn-rxjs/operators/transformation/map) can be used to modify the response before sending it to the application. The [do](https://www.learnrxjs.io/learn-rxjs/operators/utility/do) operator is useful for logging the events or time requests. The [catch](https://www.learnrxjs.io/learn-rxjs/operators/error_handling/catch) operator can be used to catch the error. The [retry](https://www.learnrxjs.io/learn-rxjs/operators/error_handling/retry) operator can be used to retry the failed operation.

### Logging

The following example code shows the use of do operator. The do operator is invoked whenever certain events take place on an Observable.



In the above example, do is invoked twice. First time when the request is sent to the server (event={type: 0}). The second time when the response is received (event instanceof HttpResponse).

### Modify Response

The following code shows the use of the map operator, which allows us to transform the response. The response can be modified using the method clone (the response object is immutable). Then return the cloned response. The example below replaces the entire response body with the new body and returns the response.



### Catching the Error

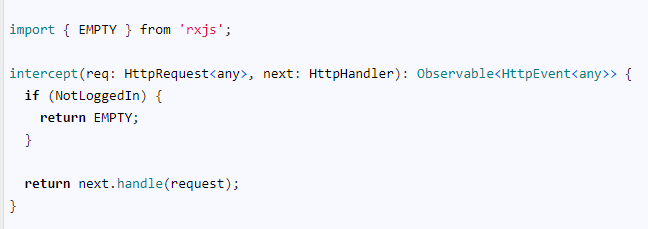
The errors can be caught with the catch operator. The catch callback gets the HttpErrorResponse as its argument, which represents an error object. It contains information about headers, status, statusText  & URL, etc.



## Cancel the current Request

We can also cancel the current request by returning the EMPTY observable.

The following code snippet checks if the user is logged in. If not then it will not send the request to server.



## Change the Requested URL

You can change the requested URL before it sent to the server. The HttpRequest contains the url property, which you can change before sending the request.

This is useful when you want to add the base URL of all the requests, change HTTP to HTTPS etc.

`