**Step 2.1: Add HTTP Module**

First we need to tell Angular that we will use the HttpModule. For that, we edit src/app/app.module.ts (added parts in red)

import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

**import { HttpModule } from '@angular/http';**

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

@NgModule({

declarations: [

AppComponent

],

imports: [

**HttpModule,**

BrowserModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

**Step 2.2: Configure Component to use HTTP**

In a second step, we configure Edit src/app/app.component.ts:

import { Component } from '@angular/core';

import { Http, Response, Headers } from '@angular/http';

import 'rxjs/add/operator/map'

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

**data: any = null;**

**constructor(private \_http: Http) {**

**this.getMyBlog();**

**}**

**private getMyBlog() {**

**return this.\_http.get('https://public-api.wordpress.com/rest/v1.1/sites/oliverveits.wordpress.com/posts/3078')**

**.map((res: Response) => res.json())**

**.subscribe(data => {**

**this.data = data;**

**console.log(this.data);**

**});**

**}**

}

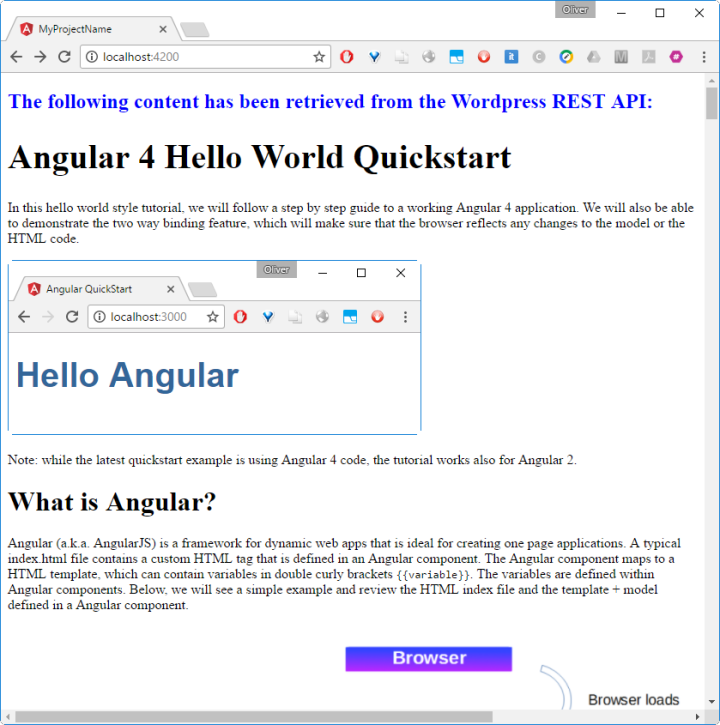
**Step 2.3: Adapt HTML Template**

Remove all content of src/app/app.component.html and replace it by the following content:

https://oliverveits.files.wordpress.com/2017/06/2017-06-24-16_49_57-eingabeaufforderung-vagrant-ssh.png?w=720

(WordPress seems to have a problem displaying the content correctly, so I had to post it as a screenshot this time)

After that, we will see that the browser is displaying following content:

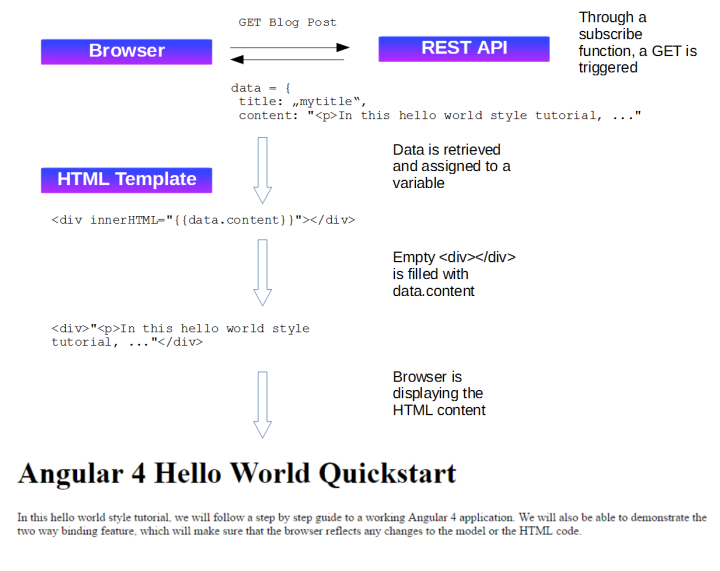


With that, we have created a little proof of concept how to retrieve data from a REST API and display the data dynamically.

**Summary**

In this hello world style step-by-step guide, we have learned

* how to install Angular 4 using the Angular CLI and
* how to consume the WordPress API for retrieving and displaying a blog post title and HTML content.



### Making HttpClient Available In The Project

To be able to use the HttpClient service within your components we first need to include the HttpClientModule in the Angular application. First we need to import HttpClient module in the application’s root module in file app.module.ts:

import { BrowserModule } from '@angular/platform-browser';  
import { NgModule } from '@angular/core';  
import { HttpClientModule } from '@angular/common/http';

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

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

Once imported you can make use of HttpClient in your components. To make HttpClient available in the component class you need to inject it into the class constructor like you can see in the following:

import { Component, OnInit } from '@angular/core';  
import { HttpClient } from '@angular/common/http';

@Component({  
 selector: 'app-root',  
 templateUrl: './app.component.html',  
 styleUrls: ['./app.component.css']  
})  
export class AppComponent {  
 title = 'app';

constructor(private http: HttpClient){  
 }

}

HttpClient will use the XMLHttpRequest browser API to execute HTTP request. In order to execute HTTP request of a specific type you can use the following methods which corresponds to HTTP verbs:

* get
* post
* put
* delete
* patch
* head
* jsonp

### Using HttpClient To Request Data

Let’s implement a simple example which uses GitHub’s REST API to request user data. Insert the following code in file app.component.ts:

import { Component, OnInit } from '@angular/core';  
import { HttpClient } from '@angular/common/http';

@Component({  
 selector: 'app-root',  
 templateUrl: './app.component.html',  
 styleUrls: ['./app.component.css']  
})  
export class AppComponent implements OnInit {  
 title = 'app';  
 results = '';

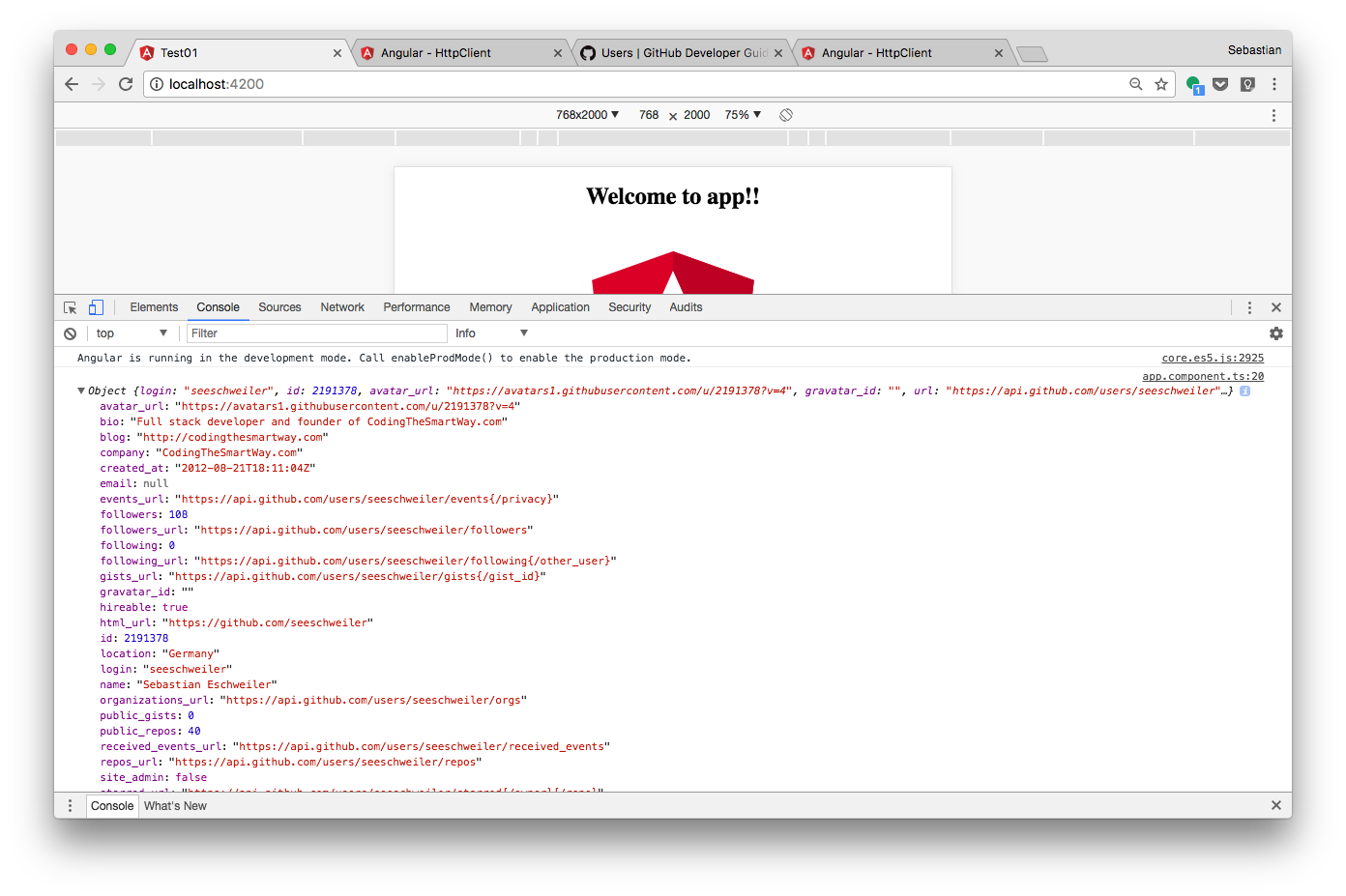
constructor(private http: HttpClient){

}

ngOnInit(): void {  
 this.http.get('https://api.github.com/users/seeschweiler').subscribe(data => {  
 console.log(data);  
 });  
 }

}

The output which is displayed in the browser console should look like the following:



The results shows that it’s possible to directly access the JSON response by subscribing to the Observable which is returned from the get method.

### Typed Response

From the console output you can see that the returned JSON object has a lot of properties. If you now try to access one of those properties by using the dot notation you’ll get back an error:  
   
 console.log(data.login);  
   
 The error message is saying: “Property 'login' does not exist on type 'Object'”. As data is of type Object you can not access properties directly. However we're able to cast the response Object to a type which is containing the corresponding properties. Let's define an interface type which is containing some if the properties which are part of the response:

interface UserResponse {  
 login: string;  
 bio: string;  
 company: string;  
}

Next, let’s make use of UserResponse to cast the return type of the get call:

this.http.get<UserResponse>('https://api.github.com/users/seeschweiler').subscribe(data => {  
 console.log("User Login: " + data.login);  
 console.log("Bio: " + data.bio);  
 console.log("Company: " + data.company);  
 });

Accessing the values by using data.login, data.bio and data.company is possible now. The output in the browser console should no deliver the following result:

https://cdn-images-1.medium.com/max/1600/1*TvlBI2n9q2m7e8iByGVZWA.png

### Error Handling

A HTTP request can fail. Because of a poor network connection or other circumstances which can not be foreseen. Therefore you should always include code which handlers an error situation. Adding a second callback method to the subscribe method is the way this is done:

this.http.get<UserResponse>('https://api.github.com/users/seeschweiler').subscribe(  
 data => {  
 console.log("User Login: " + data.login);  
 console.log("Bio: " + data.bio);  
 console.log("Company: " + data.company);  
 },  
 err => {  
 console.log("Error occured.")  
 }  
 );

You can also get more specific information about the error by defining a parameter of type HttpErrorResponse for the error handler function. HttpErrorResponse needs to be imported from @angular/common/http:

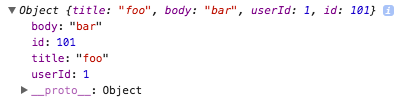
this.http.get<UserResponse>('https://api.github.com/users/seeschweiler').subscribe(  
 data => {  
 console.log("User Login: " + data.login);  
 console.log("Bio: " + data.bio);  
 console.log("Company: " + data.company);  
 },  
 (err: HttpErrorResponse) => {  
 if (err.error instanceof Error) {  
 console.log("Client-side error occured.");  
 } else {  
 console.log("Server-side error occured.");  
 }  
 }  
 );

### Using HttpClient To Send Data

Next, let’s see how data can be send via HttpClient. For sending data we’ll be using the post method of the HttpClient object. Of course we need a backend which offers a REST API which accepts POST HTTP requests. To avoid setting up our own backend API we can instead make use of JSONPlaceholder which is a fake online REST API for testing and prototyping (<https://jsonplaceholder.typicode.com/>).  
   
 The endpoint <http://jsonplaceholder.typicode.com/posts> is support POST HTTP request. Let’s use that endpoint to create a new post record:

const req = this.http.post('http://jsonplaceholder.typicode.com/posts', {  
 title: 'foo',  
 body: 'bar',  
 userId: 1  
 })  
 .subscribe(  
 res => {  
 console.log(res);  
 },  
 err => {  
 console.log("Error occured");  
 }  
 );

Here you can see that the data of the post request are passed to the postmethod as a second parameter in JSON format. We’re getting back a response which is confirming that the object has been created succesfully:



### Interceptors

One of the new features of the new HttpClient module is the availability of interceptors. Interceptors are sitting in between your application and the backend. By using interceptors you can transform a request coming from the application before it is actually submitted to the backend. The same is possible for responses. If a response arrivers from the backend a interceptor can transform that response before it arrives in your application.   
   
 The best way to discover how interceptors work is to implement a simple example. Create a new file githubauth.interceptor.ts and insert the following code:

import { Injectable } from '@angular/core';  
import { HttpEvent, HttpInterceptor, HttpHandler, HttpRequest } from '@angular/common/http';  
import { Observable } from 'rxjs/observable';

@Injectable()  
export class GithubAuthInterceptor implements HttpInterceptor {  
 intercept (req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {  
 return next.handle(req);  
 }  
}

The interceptor class GithubAuthInterceptor implements the interface HttpInterceptor which is part of the @angular/common/http library. Implementing this interface requires us to add the intercept method to the class. This method is doing the main work of the interceptor. The method is expecting to get two parameters. The first parameter is containing the original request. The second parameter is the next HTTP handler to which the request needs to be passed on for further processing.   
 In this first example the intercepts method is just passing on the original request to the next handler. Having understood the interceptor approach we can now move on and manipulate the request in the intercept method:

import { Injectable } from '@angular/core';  
import { HttpEvent, HttpInterceptor, HttpHandler, HttpRequest } from '@angular/common/http';  
import { Observable } from 'rxjs/observable';

@Injectable()  
export class GithubAuthInterceptor implements HttpInterceptor {  
 intercept (req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {  
 const authReq = req.clone({  
 headers: req.headers.set('Authorization', 'token <your GitHub token>')  
 });  
 return next.handle(authReq);  
 }  
}

First we’re creating a new request by using the clone method. At the same time we’re passing in an JSON object containing the headers property. We’re using the req.headers.set method to create a new header entry for the Authorizationproperty. This property is used to submit the GitHub access token.   
   
 Finally the newly created request object (with the header included) is passed on for further processing by using the next.handle method.

### Providing The Interceptor

In order to activate the interceptor for our application we need to provide it to the main application module AppModule in file app.module.ts:

import { BrowserModule } from '@angular/platform-browser';  
import { NgModule } from '@angular/core';  
import { HttpClientModule } from '@angular/common/http';  
import { HTTP\_INTERCEPTORS } from '@angular/common/http';

import { AppComponent } from './app.component';  
import { GithubAuthInterceptor } from './githubauth.interceptor';

@NgModule({  
 declarations: [  
 AppComponent  
 ],  
 imports: [  
 BrowserModule,  
 HttpClientModule  
 ],  
 providers: [{  
 provide: HTTP\_INTERCEPTORS,  
 useClass: GithubAuthInterceptor,  
 multi: true  
 }],  
 bootstrap: [AppComponent]  
})  
export class AppModule { }