231)Angular 5 and Http

**Angular 5** is currently the latest version of Angular and it **deprecates** the Http-access method taught in this module.

**What does this mean?**

It means that the method **still works**, still is secure - you can use it! But there is a better Http module to use now: **HttpClient**.

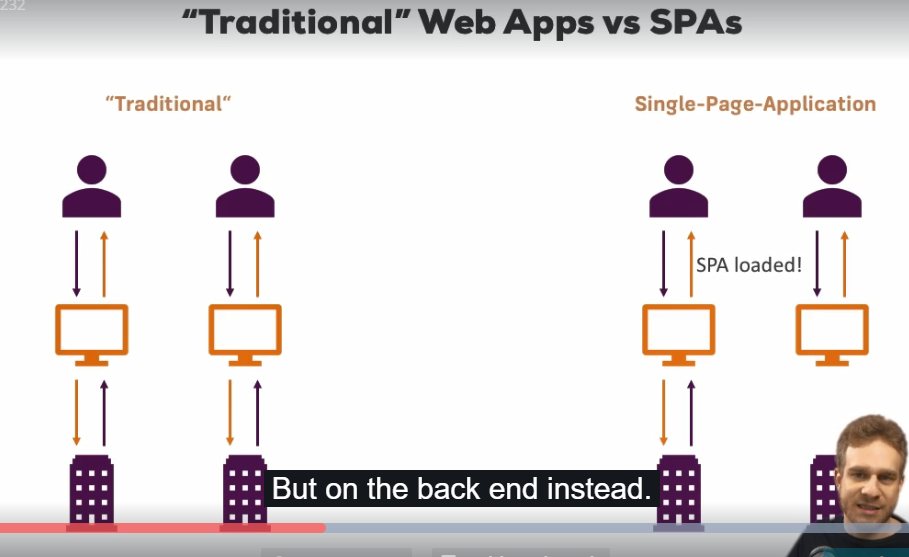
I added a **module (section 23)** on that new client months ago, even before Angular 5 was released. You'll meet it later in the course and we'll **easily update all our Http calls** with the new client there.

So for now, **follow along with this module here** - the core concepts taught here will still apply (i.e. how it works etc).

And later in the course, we'll revisit this solution and update it to HttpClient.

232)Introduction and how http request work in SPA

In SPA we do not reload the page upon every request.



Whenever we send request to server we dnt get new page,because that would mean that our angular app gets restarted because it consist of only one page. and if we kind of over write this page and send new page then our application is dead. This is not what we do.

Instead we send request from ur angular app but response will be sent back to that very same app, very same angular app. This request is sent via AJAX. It’s HTTP request sent in background through XMlHttpRequest object that javascript provides us. Thanfully we dnt have to write all the logic to send that request from our own. Instead angular ships with a convenience service we can use and we will use it in this module where you will learn how to use it.

233)Example app & backend Setup

We will use firebase a service by google as a backend. It gives us ready to use backend out of the box. Setup firebase-

Create a project in firebase. Then go to databse in this project, there go to rules, then set and read and write to ” true”. Then click on write. Now in database, go to data tab, there you can see your prject url, copy it.

234)Sending Request (Example: POST Request)

We create a service for sending request . in that service we inject built in Http service.as we are injecting service into service we use @Injectble. Now this Http service has many methods like – get,post,put,delete. These methods reprsent the kind of request being sent. Here we will first use post request. Now we can configure our backend how it handles post request. Generally post request is used to send some data.But in firbase case it is already configured.

With post request firebase appends it to any existing element, while a put request would override it. So we use post method, it takes 3 argumnets- url, body and options (this is optional and of type RequestOptionsArgs).

this.http.post('https://udemy-ng-http2.firebaseio.com/', servers);

behind the scenes angular is using observables. So we dnt send request like this. So post method will only create an observable. this observable kind of wraps our configured request but very important it is not sending it yet. Our request is not sent yet.

The reason for this is since we use observable to also then subscribe to it and react to any response we get back, if we have such an observable and we don’nt subscribe to it as we do now. There is no need to send it. There is no one to listen for your response so why would you send that. This is what is happening there as long as we don’t subscribe, no request will get sent.

I can subscribe to observable here in our service but I want to subscribe to it in component where we call storeServers method of this service. So in serverService we simply return the observable that is created.

storeServers (servers: any[]) {

return this.http.post('https://udemy-ng-http2.firebaseio.com/', servers);

}

Again request is not sent yet.

Now I provide this ServerService in provider array of app.component.

Then I inject my service in app.component. then I call method storeServers and servers array to it, still request is not sent. This method returns us a observable. so we subscribe to it, now our request is sent.

Now there is no need to unsubscribe from this service , angular will do that for us once response is send back.

Now you have to add **/data.json** at the end of your project url. This is firebase specific thing. It means we want to work with database. Body of request can have any object. This object will be stored in databse.

Now most important thing- In app.module, you have to import built in **HttpModule.**  If we dnt import this module then we cnt use this http service.

App.component.ts-

onSave() {

this.serverService.storeServers(this.servers)

.subscribe((response) => {

console.log(response);

},

(error) => {

console.log(error);

}

);

}

If we run our code and click on save. W see response on console. we open our project and go to database then data we can see our object there.

235)Adjusting request headers

Lets say we want to send some headers with our request. In last lecture pre-configured headers were enough but lets say your backend need some speicifc headers. we can do it

We create variable which is of type Headers(imported from @angulatr/http), we call constructor and to this constructor we pass javascript object. In this object we configure our headers. Now we have our headers, then we pass them to get method like this-

storeServers (servers: any[]) {

const headers = new Headers({'Content-Type': 'application/json'});

return this.http.post('https://udemy-ng-http2.firebaseio.com/data.json', servers, {headers: headers});

}

}

We pass third options argument to get method. This third argument is optional.

Here we do not need to set up this headers because default is this. But in this way we can add more headers.

Now to check which headers were passed in request- go to chrome dev tools, click on network tab. Then click on request, hen go to headers section. Here we can see request and response headers.

236)Sending GET Request

Here we use get method. Get has only 2 arguments, first is url and second one is optional options parameter, we can add headers if we want to, here we don’t need them. So we send request in serversService like this-

getServers() {

return this.http.get('https://udemy-ng-http2.firebaseio.com/data.json');

}

We subscribe to observable returned in app.component-

onGet() {

this.serverService.getServers().subscribe(

(response) => {

console.log(response);

},

(error) => {

console.log(error);

});

}

Response is a complex object. Now we want to extract data which is sent in json format in response body. When we were sending data in post request, there angular automatically converted it into json format. But when data comes back in json format in response body, angular does’nt unwrap it automatically.

Now we have to unwrap our data and convert it into js object so that we can work with it. We want to get data that is in body property of response object.data that we get from that observable is of type Response. Now on this object we get a method named, json. This method do both of these tasks that we want to do.

onGet() {

this.serverService.getServers().subscribe(

(response: Response) => {

const data = response.json();

console.log(data);

},

(error) => {

console.log(error);

});

}

237)Sending a PUT Request

Put request is just like post request. We send body with it. Now it depends upon your backend how it handles it. Generally put request is used to update records and post is used to create new record.

In case of fireBase, same thing happens. When we send post request it updates our existing array. when we send post request, it adds new array to our database.

Now we comment out post method and use put method. Arguments and all other things remain same.

238)Transforming Responses easily with Observable Operator(map())

Here we are extracting the data from response in function that pass to subscribe function. now may be we are calling getServers method in ServerService from many components then we have to duplicate code where we extract body from response or do some transformations in all components.

So better approach is to do all these things in SeverService. As methods of Http service return us observables we can use , operators on them. Observables provide us many operators. But use these operators we need to add this import in our file-

import 'rxjs/Rx';

here we use map operator. To this operator we pass a function as with one argument and this functions returns something. Now data emitted by observable is received by this function as a argument , it modifies this data in function then it retruns that data. But before returning it wraps it up in observable, so in the end we gain get a observable to which we can subscribe but data is transformed.

serverService.ts –

getServers() {

return this.http.get('https://udemy-ng-http2.firebaseio.com/data.json').

map((response: Response) => {

const data = response.json();

return data;

});

}

Now in next function that we pass to subscribe we are not getting response we are getting extracted data, so we have to change code there.-

onGet() {

this.serverService.getServers().subscribe((data) => {

console.log(data);

});

}

So now we are doing the transformation in central place i.e in our service.

239)Using the Returned Data

Here we saw that how we can use the data that we fetched from backend in our component.

onGet() {

this.serverService.getServers().subscribe((servers) => {

this.servers = servers;

});

}

240)Catching HTTP Errors

Remove **.data** from url to which you are sending the request. Now this url will be invalid for firebase. If we send request, then we get a error on browser console, this is generated by browser, we also get response from firebase which is not that useful. Lets say we want to catch errors like this. We can do this by going to ServerService and adding catch operator to our observable. now this error operator needs to return us an observable just like map operator because we subscribing to this observable in our app.component. but for some reasons this error operator does’nt wrap data that we return in an observable as map operator do. We do create our observable manually.

getServers() {

return this.http.get('https://udemy-ng-http2.firebaseio.com/data.json').

map((response: Response) => {

const data = response.json();

return data;

}).

catch((error: Response) => {

console.log(error);

return Observable.throw(error);

});

}

But here we are just logging the error into console. infact error is logged 2 times. It is bacause we also log error in function that we pass to subscribe method. But it shows that catch method is getting called. More useful useage of catch will be to throw some error message but error.

getServers() {

return this.http.get('https://udemy-ng-http2.firebaseio.com/data.json').

map((response: Response) => {

const data = response.json();

return data;

}).

catch((error: Response) => {

return Observable.throw('Something went Wrong');

});

}

Now in app.component second function(error) to subscribe will be called. Here we can display a modal with error message or something like this. Here we simply log it.

App.component.ts-

onGet() {

this.serverService.getServers().subscribe((servers) => {

this.servers = servers;

},

(error: Response) => {

console.log(error);

});

}

Now if we run our code. We see a error message followed by our message. First error message is by browser we cnt hide it. But below you can control what you want to display to user.ypu can log this error someplace else, whatever you want to do. It’s important to know that catch method is there and you can do some centralize error transformation, if you need it your app, in this operator.