9 Angular HttpClient (23)

More info:
https://angular.io/guide/http

289 Unlocking

1. Import it **HttpClientModule** to the **app.module.ts**

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


import { AppComponent } from './app.component';
import { AppRoutingModule } from './app-routing.module';
import { SharedModule } from './shared/shared.module';
import { ShoppingListModule } from './shopping-list/shopping-list.module';
import { AuthModule } from './auth/auth.module'
import { CoreModule } from './core/core.module';


@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    AppRoutingModule,
    SharedModule,
    ShoppingListModule,
    AuthModule,
    CoreModule
  ],
```

```
  bootstrap: [AppComponent]
})
export class AppModule { }
```

2.Using Get/Put method, + using **typed get request**

Data-storage.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders, HttpParams, HttpRequest } from '@angular/common/http';
import 'rxjs/Rx';

import { RecipeService } from '../recipes/recipe.service';
import { Recipe } from '../recipes/recipe.model';
import { AuthService } from '../auth/auth.service';

@Injectable()
export class DataStorageService {
  constructor(private httpClient: HttpClient,
              private recipeService: RecipeService,
              private authService: AuthService) {
  }
//1. The PUt method can be used as the same as with Http
  storeRecipes() {
    const token = this.authService.getToken();
    return this.httpClient.put('https://ng-recipe-book-3adbb.firebaseio.com/recipes.json?auth=
 + token', this.recipeService.getRecipes()
    });
  }
//2. With the httpClient we can get typed response, we can specify the type in which we would l
ike to get the response.
  getRecipes() {
     this.httpClient.get<Recipe[]>('https://ng-recipe-book-3adbb.firebaseio.com/recipes.json?au
th=' + token)
```

```
      .map(
        (recipes) => {                          //Since we got an array of Recipe[] model, we
  just have to loop through, no need to declare it
          console.log(recipes);                  // like(recipes: Recipe[])
          for (let recipe of recipes) {
            if (!recipe['ingredients']) {
              recipe['ingredients'] = [];
            }
          }
          return recipes;
        }
      )
      .subscribe(
        (recipes: Recipe[]) => {
          this.recipeService.setRecipes(recipes);
        }
      );
  }
}
```

290 Requesting Configuration and Response

1. You can define what response would like to recieve with the **GET** method. Has 3 main things, you can define the type to be recieved + the url + an optional object which lets you configure a structure from a response
get<RecievedType>('url', {observe: 'xx', responseType: 'xx' }).

```
this.httpClient.get<Recipe[]>('https://ng-recipe-book-3adbb.firebaseio.com/recipes.json', {
      observe: 'body',
      responseType: 'json'


    })
```

Data-storage.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders, HttpParams, HttpRequest } from '@angular/common/http';
import 'rxjs/Rx';

import { RecipeService } from '../recipes/recipe.service';
import { Recipe } from '../recipes/recipe.model';
import { AuthService } from '../auth/auth.service';

@Injectable()
export class DataStorageService {
  constructor(private httpClient: HttpClient,
              private recipeService: RecipeService,
              private authService: AuthService) {
  }

  storeRecipes() {
    const req = new HttpRequest('PUT', 'https://ng-recipe-book-3adbb.firebaseio.com/recipes.jso
n', this.recipeService.getRecipes(), {reportProgress: true});
    return this.httpClient.request(req);
  }

  getRecipes() {
    // this.httpClient.get<Recipe[]>('https://ng-recipe-book-3adbb.firebaseio.com/recipes.json?
auth=' + token)
    this.httpClient.get<Recipe[]>('https://ng-recipe-book-3adbb.firebaseio.com/recipes.json', {
//Takes the body and converts it into json
      observe: 'body',
      responseType: 'json'
  //Takes a string, and give us the full response observe: 'response' in a text format
 //    observe: 'response',
  //    responseType: 'text'
    })
```

```
      .map(
        (recipes) => {
          console.log(recipes);
          for (let recipe of recipes) {
            if (!recipe['ingredients']) {
              recipe['ingredients'] = [];
            }
          }
          return recipes;
        }
      )
      .subscribe(
        (recipes: Recipe[]) => {
          this.recipeService.setRecipes(recipes);
        }
      );
  }
}
```

290 Requesting Events

1. You can use the **put, observe:'events** to listen for specific events'
Data-storage.service.ts

```
  storeRecipes() {
     return this.httpClient.put('https://ng-recipe-book-3adbb.firebaseio.com/recipes.json', thi
s.recipeService.getRecipes(), {
       observe: 'events'
     });
  }
```

2. WHen you are subscribin, triggering event, you can listen to the events, but import the **HttpEvent** Object, + **HttpEventType** as well to get the event data
header.component.ts

```
import { Component } from '@angular/core';
import { HttpEvent, HttpEventType } from '@angular/common/http';

import { DataStorageService } from '../../shared/data-storage.service';
import { AuthService } from '../../auth/auth.service';

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html'
})
export class HeaderComponent {
  constructor(private dataStorageService: DataStorageService,
              private authService: AuthService) {
  }

  onSaveData() {
    this.dataStorageService.storeRecipes()
      .subscribe(
        (response: HttpEvent<Object>) => {
          console.log(response.type = HttpEventType.sent);
        }
      );
  }
}
```

291 Setting Headers

1. Import **HttpHeaders** + create a `new` **Header** object with **.set** or add **.append** than pass it into the header optional object for the Put request.

data-storage.service.ts

```
  import { HttpClient, HttpHeaders, HttpParams, HttpRequest } from '@angular/common/http';
  storeRecipes() {
```

```
    const headers = new HttpHeaders().set('Authorization', 'Bearer afdklasflaldf');
    const headers = new HttpHeaders().append('Authorization', 'Bearer afdklasflaldf');


     return this.httpClient.put('https://ng-recipe-book-3adbb.firebaseio.com/recipes.json', thi
s.recipeService.getRecipes(), {
       observe: 'body',
       params: new HttpParams().set('auth', token)
       // headers: headers    -> Firebase does not expect header information, but this would be
 the method
     });
  }
```

292. Setting parameters

0. `Important` the url will not have the **?auth= + token !!!!!**
1. Import the **HttpParams**, and create a `New` **HttpParams** object, which can be inserted in the PUt request optional object

data-storage.service.ts

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


  storeRecipes() {
     const headers = new HttpHeaders().set('Authorization', 'Bearer afdklasflaldf');
//Important the url will not have the ?auth= + token !!!!!
     return this.httpClient.put('https://ng-recipe-book-3adbb.firebaseio.com/recipes.json', thi
s.recipeService.getRecipes(), {
       observe: 'body',
       params: new HttpParams().set('auth', token)
       // headers: headers
     });


  }
```

293 Progress

1. Import the **HttpRequest** and you can create a `new` **request object**
This will give you progress data about your request **reportProgress: true.** +adding auth parameters

data-storage.service.ts

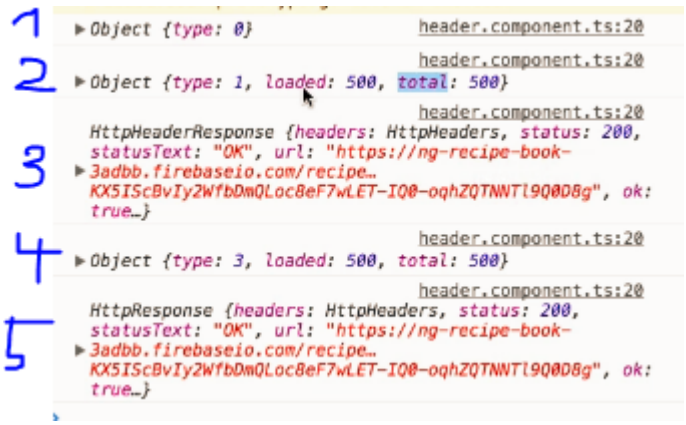
```
 import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders, HttpParams, HttpRequest } from '@angular/common/http';


  storeRecipes() {


    const req = new HttpRequest('PUT', 'https://ng-recipe-book-3adbb.firebaseio.com/recipes.jso
n', this.recipeService.getRecipes(),
    {reportProgress: true,
    params: new HttpParams().set('auth', token)
    });
    return this.httpClient.request(req);
  }
```

The progress data looks the following:

1. The event type - > 0 means it it sent, === HttpEventType.sent
2. The Uploaded content, type 1 means it is uploading, loaded is the loaded files, total is the amount if files which has to be loaded
3. Response recieved for the upload
4. The Donwloading status, type 3 ===HttpEventType.download
5. Response recieved for the download



294. Interceptors, modifying requests

Checking any outgoing requests and manipulates them.
0.Executing code before the requests have been completed

1. With **HttpINterceptor** + **HttpHandler**, intercept all of the requests, and add an auth="paramater " to the request with **httphanlder**

auth.interceptor.ts

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

import { AuthService } from '../auth/auth.service';

@Injectable()
//Implemenets a special angular library. it will require us to implement a intercept() funciton
//This intercept method will return an observable, with an HttpEvent generic type
export class AuthInterceptor implements HttpInterceptor {
  constructor(private authService: AuthService) {}

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    console.log('Intercepted!', req);
    // const copiedReq = req.clone({headers: req.headers.set('', '')});
    const copiedReq = req.clone({params: req.params.set('auth', this.authService.getToken())});
    return next.handle(copiedReq);
    // return null;
  }
}
```

2. YOu have to register it in the **core module** or in the app.module.ts

- provide: HTTP_INTERCEPTORS, --> Angular will know that it has to intercept all of our outgoing request
- useClass: AuthInterceptor, --> Will know which defined intercpetor should be used (has to be imported)
- multi: true --->Means multiple interceptor is defined

Note: The interceptors will be executed in the order they are defined in the below ts

```
import { NgModule } from '@angular/core';
import { HTTP_INTERCEPTORS } from '@angular/common/http';
```

```
import { AuthInterceptor } from '../shared/auth.interceptor';
import { LoggingInterceptor } from '../shared/logging.interceptor';

@NgModule({
  declarations: [
    HeaderComponent,
    HomeComponent
  ],
  imports: [
    SharedModule,
    AppRoutingModule
  ],
  exports: [
    AppRoutingModule,
    HeaderComponent
  ],
  providers: [
    AuthService,
    {provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true},
    {provide: HTTP_INTERCEPTORS, useClass: LoggingInterceptor, multi: true}
  ]
})
export class CoreModule {}
```

296 Multiple Interceptors

Executing code after the requests have been completed

Intercepting incoming requests
Using the `.do()` observable operator
which means we are not consuming it, just do sth between the steps, between emitting it.
logging.interceptor.ts

```
import { HttpEvent, HttpHandler, HttpInterceptor, HttpRequest } from '@angular/common/http';
```

```
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/do';


export class LoggingInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    return next.handle(req).do(
      event => {
        console.log('Logging interceptor', event);
      }
    )
  }
}
```