Angular Pipes (17), Http(18)

https://ide.c9.io/laczor/angular
More information about the pipes
https://angular.io/api

session_17_Pipes
222. Using Pipes
223 Parametirizing Pipes
224 Chaining Pipes
226 Creating a Custom Pipe + parametarize
228 Creating FIlter Pipe, *pure* property
230 Using Async Pipe
231 CHallange Creating a String reverse + sorting pipe -->> (no need for extra variable + short string comparison)
session_18_Http (http-start)
233 Example App with Backend (FireBase)
234 Sending POST request
235 Adjusting Header Request
236. GET Method
237 PUT Method
238 Transform response with Observables
239. Using Returned Data
240 Catching Http Errors
241 Using Asycn pipe with Http Request
246. Project Work , using Http Service + centralized data service (Avoiding circular reference, + tranforming the recieved data)
247. Transforming the retrieved Data , empty arrays [] included

222. Using Pipes

Pipes are actually built in tools that allow us to modify the output of the data. (We use them in the HTML)

Built in Pipes:

```
// These will transform it  <div><strong>{{ server.name }}</strong>|  {{ server.instanceType |
 uppercase}} |    {{ server.started | date}} <     From User Database| large | Mon Aug 09 1920
 00:00:00 GMT+0100 (GMT nyári idő) to User Database| LARGE | Aug 9, 1920


{{ server.instanceType | uppercase}} |

{{ server.started | date}}
```

223 Parametirizing Pipes

```
// From User Database| LARGE | Aug 9, 1920  to  User Database| LARGE | Monday, August 9, 1920

{{ server.started | date:'fullDate'}}
```

224 Chaining Pipes

It is working from left to right, and the pipes argument will be the information/data, next to it to the left.

```
{{ server.started | date:'fullDate' | uppercase}}                // It will work

{{ server.started |  uppercase | date:'fullDate' |}}             // It will not work, since the

 uppercase, can'T work with date, only with string
```

226 Creating a Custom Pipe + parametarize

Will shorten our arguments with a custom pipe
1. Create a custom file with a exported class which implements `PipeTransform`
2. You can use the angular CLi for this purpose **n g p pipeName**

shorten.pipe.ts

```
import { Pipe,                            //THis special decorator will define our pipe's name

 PipeTransform                            // It is a good practice to implement PipeTransform t

o tell angular that this is a Pipe

  } from '@angular/core';

@Pipe({

  name: 'shorten'

})

export class ShortenPipe implements PipeTransform {

  transform(value: any, limit: number) {        //This is where the parameter is inserted with

 the type declaration

    if (value.length > limit) {

      return value.substr(0, limit) + ' ...';

    }

    return value;

  }
```

```
}
```

2. Include it in the **app.module.ts**

```
import { ShortenPipe } from './shorten.pipe';


@NgModule({
  declarations: [
    ShortenPipe,
  ],
```

3. Use it on the **app.component.html**

```
<strong>{{ server.name | shorten : 15}}</strong>
```

228 Creating FIlter Pipe, *pure* property

- With the following CLI line we can create the pipe atomatically: **ng g p filterPipe**
- It will create a pipe file + register it in the app.module.ts
1. Create the pipe
filter.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';


@Pipe({
  name: 'filter',
  pure: false                    //It means if any data is changing on the site, the pipes run a
utomatically, can cause performance issue.
})
export class FilterPipe implements PipeTransform {
//So the pipe result type can be anything
//value recieved is a string,
//filterString is the value to which we would like to filter
//propName is the property to which we would like to filter on the looping server object server
[propname] = server[status]
```

```
  transform(value: any, filterString: string, propName: string): any {
    if (value.length === 0 || filterString === '') {
      return value;
    }
    const resultArray = [];
//If we recieved multiple values due to the *ngFor collection, we loop through the recieved arr
ay, and create a new array with the filtered data, which will be returned
    for (const item of value) {
      if (item[propName] === filterString) {
        resultArray.push(item);
      }
    }
    return resultArray;
  }
}
```

2. Apply to the HTML
- Be sure to implement a variable to store the **filteredStatus** variable value
app.component.html

```
<!--
1. First at filteredStatus we create a component's variable with to way databinding
2. Then at the *ngFor loop we apply the the pipes inserting the following parameters
filteredStatus is the componenet's variable
'status' is a passed string property, which is a property of the the looped server object. serv
er[status]
-->
<div class="container">
  <div class="row">
    <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
      <input type="text" [(ngModel)]="filteredStatus">                    <!-- Two way databind
ing-->
      <br>
      <button class="btn btn-primary" (click)="onAddServer()">Add Server</button>
```

```
      <br><br>
      <h2>App Status: {{ appStatus | async}}</h2>                        <!--Async pipe appliad
 to the string interpolation-->
      <hr>
      <ul class="list-group">
        <li
          class="list-group-item"
          *ngFor="let server of servers | filter:filteredStatus:'status'"                <!-- A
pplying the filter on a collection of servers-->
          [ngClass]="getStatusClasses(server)">
          <span
            class="badge">
            {{ server.status }}
          </span>
          <strong>{{ server.name | shorten:15 }}</strong> |
          {{ server.instanceType | uppercase }} |
          {{ server.started | date:'fullDate' | uppercase }}
        </li>
      </ul>
    </div>
  </div>
</div>
```

230 Using Async Pipe

1. Wrapping an async operation in a promise, so we will be able to return data if it was successfull.
(Note this promise is a component's variable called **appStatus**)
app.component.ts

```
  appStatus = new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve('stable');
    }, 2000);
  });
```

2. Then apply the async pipe to the string interpolation
app.component.html

```
<h2>App Status: {{ appStatus | async}}</h2>
```

231 CHallange Creating a String reverse + sorting pipe

(pipes-assignment-start)

My solution

```
  transform(value: any): any {
    if (value.length === 0 ) {
      return value;
    }
      const sortArray =   value.sort(function(a, b) {
                            var nameA = a.name.toUpperCase(); // nagybetűk és kisbetűk elhagyás
a
                            var nameB = b.name.toUpperCase(); // nagybetűk és kisbetűk elhagyás
a
                            if (nameA < nameB) {
                              return -1;
                            }
                            if (nameA > nameB) {
                              return 1;
                            }

                            // a neveknek egyeznie kell
                            return 0;
                          });
    console.log(sortArray)
    return sortArray;
  }
```

Course solution
- SInce we got an array, we modify it and returning it, it is not necessary to create a sample array for it, we just have to sort it and immediately return it
- als with just 1 line we can sort the name of the object with the **<** or **>** comparisons. **(a[propName] > b[propName])**

```
transform(value: any, propName: string): any {
  return value.sort((a, b) => {
    if (a[propName] > b[propName]) {
      return 1;
    } else {
      return -1;
    }
  });
}
```

233 Example App with Backend (FireBase)

https://firebase.google.com/

THis is the link for the created project:
https://udemy-ng-http-dded5.firebaseio.com/

1. Login, create a sample project
2. Change the Database/Rules

```
{
  "rules": {
    ".read": "true",  // from  auth != null"
    ".write": "true"
  }
}
```

234 Sending POST request

0. IMport **HttpModule** to app.module.ts

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


imports: [   HttpModule ],
```

1. Create a **server-service.service.ts** file

```
import { Injectable } from '@angular/core';            //It is needed since we are injecting a
 http service into this service
import { Http }from '@angular/http';                  //Built in http service


@Injectable()
export class ServerService {
    constructor(private http: Http){}               //Inject http


    storeServers(servers:any[]){                    //Waiting an array of element of type (any)
        //(Url, data)
      //This http.post is actually an observable, which is a data package, everytime you have t
o subscribe to it, so it can send and be recieved.
       //Without subscription it is not sending the request, so where we are using this method,
 we have to subsribe for the returning observable
       //With firebase we have to specify our endpoint .json is important since this tells fire
base that we will work with the database
        return this.http.post('https://udemy-ng-http-dded5.firebaseio.com/data.json', servers);
    }
}
```

2. Import it in the **app.module.ts**

```
import { ServerService } from './server-service';
providers: [ServerService],
```

3.Add a button which will trigger **onSave()** in the **app.component.html**

```
<button class="btn btn-primary" (click)="onAddServer(serverName.value)">Add Server</button>
```

3. Inject the ServiceServer + subscribing to the emitted observable by the http.post method

```
import { ServerService } from './server-service';

export class AppComponent {

 constructor( private serverService:ServerService){}

 //It is extremely important to subscribe to the obserable, since otherwise it will not send th
e request,

 //Angular will automatically delete the subscriptions

  onSave(){

    this.serverService.storeServers(this.servers).subscribe(

      (response)=>{console.log(response);

      (error)=>{console.log(error)}

    });

}
```

235 Adjusting Header Request

1. Simply just import Headers from http module
2. Define it as the custom object
3. Parse it in as a 3rd argument for the **http.post (url,data,{headers:'xxxx'})**

server-service.service.ts

```
import { Injectable } from '@angular/core';            //It is needed since we are injecting a
 http service into this service

import { Http, Headers }from '@angular/http';                  //Built in http service


@Injectable()

export class ServerService {

    constructor(private http: Http){}               //Inject http


    storeServers(servers:any[]){                    //Waiting an array of element of type (any)

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

        return this.http.post(

        'https://udemy-ng-http-dded5.firebaseio.com/data.json',
```

```
        servers,

        {headers:headers});

    };

}
```

236. GET Method

Good practice is to return an observable and where we are actually using the get request, we should subscribe to that observable there

1. Create the getServers() in the **server-service.ts**
get(url) --> Has only just 1 parameter the attribute which is pointing to the database, or the url

```
  getServers(){

        return this.http.get('https://udemy-ng-http-dded5.firebaseio.com/data.json');

    }
```

2. Use this method, subscribe to the returning observable
3. Use **respone.json()** to extract the json object from the body of the response
app.component.ts

```
    onGet(){

    this.serverService.getServers().subscribe(                    //Subscription is needed to t
he observable to send the request

      (response)=>{

       const data = response.json();                                //if it comes back we can e
xtract the json form the body

       console.log(data);

      },

      (error)=>{console.log(error)});

  }
```

237 PUT Method

Will overWrite all of the existing data on the URL

```
      storeServers(servers:any[]){                    //Waiting an array of element of type (an
y)
       const headers = new Headers({'Content-Type':'application/json'});
       return this.http.put(
        'https://udemy-ng-http-dded5.firebaseio.com/data.json',
        servers,
        {headers:headers});
    };
```

238 Transform response with Observables

1. Import (**'rxjs/Rx';**) to use the observable operators , like map()
2. Import (**Response**) since the get method will return a response object
3. GET method will return a Response observable, and with the osbervable methods (map) we loop over the respone, and returns the individual json files.

```
import { Injectable } from '@angular/core';            //It is needed since we are injecting a
 http service into this service
import { Http, Headers, Response }from '@angular/http';                  //Built in http service
import ' rxjs/Rx ';
@Injectable()
export class ServerService {
    constructor(private http: Http){}               //Inject http


    storeServers(servers:any[]){                    //Waiting an array of element of type (any)
         const headers = new Headers({'Content-Type':'application/json'});
       return this.http.put(
        'https://udemy-ng-http-dded5.firebaseio.com/data.json',
        servers,
        {headers:headers});
    };
    getServers(){
    //GET method will return a Response observable, and with the osbervable methods we loop ove
r the respone, and returns the individual json files.
```

```
        return this.http.get('https://udemy-ng-http-dded5.firebaseio.com/data.json')
        .map( (response:Response)=>{                                                    //Callback
 function of the returned Response observable
        const data = response.json();                                                   //Looping
 over the response body and return json objects
        return data;
        });
    }
}
```

4. Update the onGet method at **app.component.ts**

```
    onGet(){
    this.serverService.getServers().subscribe(
      (servers: any[])=>{console.log(servers);                    //We are expecting an array o
f type any, which we will log out.
      },
      (error)=>{console.log(error)});
  }
```

239. Using Returned Data

1. Access the returned data and modify it's content (with a for each loop)

```
    getServers(){
        return this.http.get('https://udemy-ng-http-dded5.firebaseio.com/data.json')
        .map( (response:Response)=>{
            const data = response.json();
            for(const server of data){
                server.name = 'Fetched  ' + server.name;
            }
            return data;
        });
```

```
    }
```

2.Instead of console.log, we can assign the respsonse, to the component's variable

```
      onGet(){

    this.serverService.getServers().subscribe(

      (servers: any[])=>{this.servers = servers},

      (error)=>{console.log(error)});

  }
```

240 Catching Http Errors

1. Since getServers returning an observable, we can use it's **catch()** method to catch the errors if sth went wrong
2. The catch method does not return an observable, which is a problem, since at the **app.component.ts** we are subscribing to it,
we can use the **Observable** module and + **Observable.throw()** method to create a returning value wrapped in an Observable

```
import {Observable} from 'rxjs/Observable';


    getServers(){

        return this.http.get('https://udemy-ng-http-dded5.firebaseio.com/data.json')

        .map( (response:Response)=>{

        const data = response.json();

        for(const server of data){

            server.name = 'Fetched  ' + server.name;

        }

        return data;

        }).catch(

  //Using the catch method for every observable(9

            (error:Response)=> {

 //Expecting the Response type

             console.log(error);

             return Observable.throw(error);                                //Will throw back a

n observable, to which we can subscribe
```

```
            // or we can have a more meaningful message
            // return Observable.throw('Something went wrong);
        });
    }
```

241 Using Asycn pipe with Http Request

- So first of all, Async pipe is waiting for a promise or for an observable!

1. Create a new get request on the **server-service.ts** file
   (Important! you can point directly to the a**ppropriate json file** if you know it's location and key!)
2. Subscribe to it in the **app.component.tS**

https://udemy-ng-http-dded5.firebaseio.com/

udemy-ng-http-dded5
- **appName:** "Http_Request
- **data**

1. My solution is to like this
server-service.ts

```
getAppName(){
        return this.http.get('https://udemy-ng-http-dded5.firebaseio.com/appName.json');
    }
```

app.component.ts
- Wrapping the response in a promise, since we got a Response type.

```
    constructor( private serverService:ServerService){}
   appName = new Promise((resolve, reject) => {
     this.serverService.getAppName().subscribe(
       (response :Response)=>  {
         console.log(response);
        resolve(response.json());
```

```
    });
  });
```

2. Course Solution
server-service.ts
- We are using the built in observable operator map, to loop over the recieved data, and return it as a **json() object wrapped in an observable** which can be **expected by Async** operator.

```
  getAppName() {
    return this.http.get('https://udemy-ng-http.firebaseio.com/appName.json')
      .map(
        (response: Response) => {
          return response.json();
        }
      );
  }
```

app.component.ts

```
  appName = this.serverService.getAppName();
```

246. Project Work , using Http Service + centralized data service (Avoiding circular reference, + tranforming the recieved data)

(Session_18_Http/http-project-start)
0. It is a wrong practice to have circular references to the services
0.1 When **storing recipes** it was enough to s**ubscribe when the put method was called**
0.2 When we are **retrieving data** it is necessary to **subscribe immediately** on the dataStoragesService, get the data and pass it to the other functions

1. **Save the recipes to the server**
   1.1 Create the put method at **data-storage.service.ts** and call the **getRecipes()** service from RecipeService.ts
   1.2 The recipe service will just get a json data, which will be put to the datastorage
   1.3. Since http requests are observable, and you have to subscribe to them to make the request at **header.component.ts** activated method, you have to listen to it

```
import { Injectable } from '@angular/core';

import { Http }  from '@angular/http';

import {RecipeService} from '../recipes/recipe.service';

import {Response} from '@angular/http';

import {Recipe} from '../recipes/recipe.model';


@Injectable()

export class DataStorageService {

constructor(private http:Http, private recipeService:RecipeService) { }


  storeRecipes(){

        return this.http.put('https://udemy-ng-http-dded5.firebaseio.com/recipes.json',this.rec
ipeService.getRecipes());

    }

}
```

recipe-service.ts

```
  getRecipes() {

    return this.recipes.slice();

  }
```

header.component.ts

```
export class HeaderComponent {

  constructor(private dataStorageService:DataStorageService){}

  onSave(){

    this.dataStorageService.storeRecipes().subscribe(

          (response)=>{console.log(response);

          (error)=>{console.log(error)}

        });

  }

  onFetch(){

    this.dataStorageService.fetchRecipes()
```

```
  }

}
```

2. Get a copy from the server

2.1. Create the put method at d**ata-storage.service.ts** and call the **UpdateRecipes()** service from **RecipeService.ts**

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

-Really important that the response is in the same format as the Recipe object model, and we are expeting an array of recipes so that is why

(data: Recipe[]) are defined an array o recipes.

- Then we are passing the data to the **RecipeService.ts**

2.2 Assign the recieved data to the service's varibale of recipes, and notifying other observables.

2.3 From the **header.component.ts** we are callong the dataStorageService' get method

```
import { Http }  from '@angular/http';

import {RecipeService} from '../recipes/recipe.service';

import {Response} from '@angular/http';

import {Recipe} from '../recipes/recipe.model';


@Injectable()

export class DataStorageService {


  constructor(private http:Http, private recipeService:RecipeService) { }


  storeRecipes(){

        return this.http.put('https://udemy-ng-http-dded5.firebaseio.com/recipes.json',this.rec
ipeService.getRecipes());

    }

  fetchRecipes(){

        this.http.get('https://udemy-ng-http-dded5.firebaseio.com/recipes.json')

        .subscribe(                    //Subscription is needed to the observable to send the r
equest

        (response:Response)=>{

          const data: Recipe[] = response.json();                              //if it comes
 back we can extract the json form the body

         this.recipeService.updateRecipes(data);
```

```
        },
        (error)=>{console.log(error)});
    }
}
```

RecipeService.ts

```
  updateRecipes(recipes:Recipe[]) {
    this.recipes = recipes;                                          //Assiging the recieved
 data to the component's variable
    this.recipesChanged.next(this.recipes.slice());                  //Notifying other observ
ables
  }
```

header.component.ts

```
export class HeaderComponent {
  constructor(private dataStorageService:DataStorageService){}
  onFetch(){
    this.dataStorageService.fetchRecipes()
  }
}
```

247. Transforming the retrieved Data , empty arrays [] included exactly the same as we are storing in the app

(Session_18_Http/http-project-Final)

We are checking if the response data ingredients property is empty, if yes we create it an assign an empty array value it

data-storage.service.ts

```
  getRecipes() {
    this.http.get('https://ng-recipe-book.firebaseio.com/recipes.json')
      .map(
        (response: Response) => {                            //Since if the sent recipe didn't
 have any data, recipe.ingredients was not included
```

```
          const recipes: Recipe[] = response.json();         //But we would like to create one
 with empty array if there is no ingredients
          for (let recipe of recipes) {
            if (!recipe['ingredients']) {                   // recipe['ingredients'] = []; shou
ld be there in order to store it as an empty array
              recipe['ingredients'] = [];
            }
          }
          return recipes;
        }
      )
      .subscribe(
        (recipes: Recipe[]) => {
          this.recipeService.setRecipes(recipes);
        }
      );
  }
```