

Data met HTTP



HTTP client

- De Angular HTTP client
 - Wordt gebruikt voor communicatie naar een webservice / API
 - Ondersteuning voor:
 - Maken van HTTP requests (GET, POST, PUT, PATCH, ...)
 - Werken met requests en response headers
 - Asynchroon programmeren
 - Maakt gebruik van [RxJS](#) async library

De HTTP client gebruiken

- De HttpClient is een service die geïnjecteerd wordt in classes
- Voorzien in de constructor van de services:

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

@Injectable()
class myService{
  constructor(private http: HttpClient) {
  }
}
```

De HTTP client gebruiken

- Om gebruik te kunnen maken van HttpClient, moet de HttpClientModule geïmporteerd worden in de app.module.ts:

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

@NgModule({
  declarations: [
    ...
  ],
  imports: [
    BrowserModule,
    HttpClientModule
  ],
})
```

De HTTP client gebruiken

- Bijkomend worden verschillende imports uit de RxJS library geïmporteerd
 - In de service waar de http service geïnjecteerd wordt

```
import { HttpClient, HttpHeadersResponse } from '@angular/common/http';
import { Observable, throwError } from 'rxjs'
import { catchError, tap, map } from 'rxjs/operators'
```

RxJS



RxJS

- Reactive programming in JavaScript
- Asynchrone datastreams aan de hand van observables
 - UI Events
 - **HTTP requests**
 - File systems
 - **Array-achtige objecten**
 - Cache
- Wordt toegepast binnen de HttpClient module van Angular
- Voorkennis niet vereist, maar wel een meerwaarde

RxJS - ReactiveX

- Niet enkel beperkt tot JS:

Java: RxJava

Javascript: RxJS

C#(unity): UniRx

C++: RxCpp

RxJs - Datastreams

- Datastreams zijn lopende events op een tijdlijn met
 - Data
 - Errors
 - Complete signaal (optioneel)



- **Observables** worden gebruikt om deze streams te bekijken en om functies uit te voeren wanneer er een value / error of complete signal binnenkomt

RxJS - Observables

- Bij het uitvoeren van een HTTP request wordt een **Observable** object teruggegeven
- Observables kijken constant naar streams voor nieuwe data
- Een observer kan zich “**subscribe**n” op een Observable object
 - Interactie lijkt op die van een array
 - **Bij het subscribeen op een subscriber wordt de data pas binnengehaald**

RxJS - Observables

- Onderstaande functie returned een Observable
 - De map operator wordt verder uitgelegd

```
getContactList(): Observable<any> {  
    return this.http.get(BASEAPIURL);  
}
```

- De http request wordt pas uitgevoerd bij het subscriben op de observable ergens in onze code:

```
fetchContactList(): void {  
    this.service.getContactList().subscribe(data =>  
    {  
        this.contactList = data;  
    });  
}
```

RxJS – operators

- Operators zijn functies die bovenop observables kunnen draaien om collections te manipuleren
- Verschillende operators kunnen gelinkt worden met de pipe() functie

AREA	OPERATORS
Creation	from, fromPromise, fromEvent, of
Combination	combineLatest, concat, merge, startWith, withLatestFrom, zip
Filtering	debounceTime, distinctUntilChanged, filter, take, takeUntil
Transformation	bufferTime, concatMap, map, mergeMap, scan, switchMap
Utility	tap
Multicasting	share

RxJS – tap operator

- Wordt gebruikt om voor elke value uit de observable bepaalde functies uit te voeren
- Geeft op het einde de observable door naar de volgende functie
- **Manipuleert de data in de observable niet**

```
getMovies(): Observable<Movie[]>{
  return this.http.get<Movie[]>(this.moviesUrl).pipe(
    tap(obj => console.log('fetched movies')),
    catchError(this.handleError)
  );
}
```

RxJS – map operator

- De Map operator voert een functie uit op elk item dat binnenkomt van een observable.
- De Map operator geeft een observable terug met het resultaat van die functie
- **Manipuleert de data in de observable als deze in die functie aangepast wordt**

RxJS – map operator

- http.get geeft altijd een observable terug
- De map operator zet bij elke array die uit de observable komt de title property uit het eerste object om naar ‘test map’
- Geeft vervolgens de aangepaste array terug via een observable

```
getMovies(): Observable<Movie[]>{
    return this.http.get<Movie[]>(this.moviesUrl).pipe(
        tap(movieArr => console.log('fetched movies')),
        map(movieArr => {
            movieArr[0].title = 'test map';
            return movieArr;
        }),
        catchError(this.handleError)
    );
}
```

RxJS – map operator

- Meerdere maps kunnen elkaar opvolgen:

```
getNumbers(): Observable<any> {
    return this.http.get(BASEAPIURL).pipe(
        map(res => res * 2),
        map(res => res - 1)
    );
}
```

- Maps kunnen functies bevatten:

```
getContactList(): Observable<Contact[]> {
    return this.http.get(BASEAPIURL).pipe(
        map(this.parseContactData)
    );
}

parseContactData(rawContacts: any[]): Contact[] {
    return ...
}
```

RxJS – fouten opvangen

- Pipen de observable naar de RxJS catchError() operator
- Deze operator geeft een observable terug
- Voorzien van nodige imports

```
import { HttpClient, HttpErrorResponse } from '@angular/common/http';
import { Observable, throwError } from 'rxjs'
import { catchError } from 'rxjs/operators'
```

RxJS – fouten opvangen

- Bij het uitvoeren van een httpRequest Pipen we de observable door naar de catchError operator:

```
getMovies(): Observable<Movie[]>{
  return this.http.get<Movie[]>(this.moviesUrl).pipe(
    catchError(this.handleError)
  );
}
```

- Deze voert de functie handleError uit die een object van het type HttpErrorResponse meekrijgt
 - Deze functie voorzien we zelf in de service

RxJS – fouten opvangen

```
private handleError(error: HttpErrorResponse) {
  if (error.error instanceof ErrorEvent) {
    // A client-side or network error occurred.
    console.error('An error occurred:', error.error.message);
  } else {
    // The backend returned an unsuccessful response code.
    // The response body may also contain errors.
    console.error(
      `Backend returned code ${error.status}, ` +
      `body was: ${error.error}`);
  }
  return throwError(
    'Something bad happened; please try again later.');
};
```

RxJS – Gimme more please [extra]

- <https://angular.io/guide/rx-library>
- <https://gist.github.com/staltz/868e7e9bc2a7b8c1f754>
- https://www.youtube.com/watch?v=Tux1nhBPl_w
- <https://app.pluralsight.com/library/courses/rxjs-getting-started>

Data ophalen uit een lokaal JSON bestand



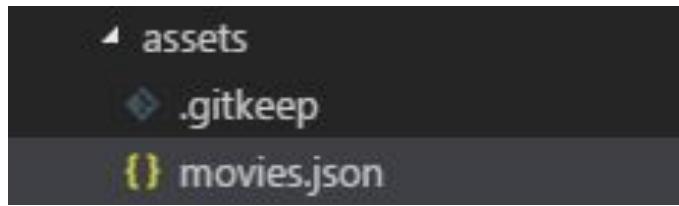
De HTTP client – lokale JSON

- Zie voorbeeld: Resources/Voorbeeld: lokale JSON
- In onze applicatie maken we gebruik van volgende model:

```
export class Movie {  
    title: string;  
    rating: number;  
    isReleased: boolean;  
  
    constructor(t: string, r: number, isR: boolean) {  
        this.title = t;  
        this.rating = r;  
        this.isReleased = isR;  
    }  
}
```

De HTTP client – lokale JSON

- In onze app plaatsen we in de assets folder een JSON file:



```
[  
  {  
    "title": "It",  
    "rating": 5,  
    "isReleased": true  
  },  
  {  
    "title": "Pitch perfect 3",  
    "rating": 7,  
    "isReleased": true  
  },  
  {  
    "title": "Star wars: The last jedi",  
    "rating": 8,  
    "isReleased": true  
  },  
  {  
    "title": "Aquaman",  
    "rating": 0,  
    "isReleased": false  
  },  
  {  
    "title": "Mary Poppins returns",  
    "rating": 0,  
    "isReleased": false  
}
```

De HTTP client – lokale JSON

- Importeren van de HttpClientModule in de app.module.ts file
- Injecteren van de HttpClient in de constructor van de service
- Voorzien van de nodige imports
- Voorzien van een link naar het JSON bestand

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpErrorResponse } from '@angular/common/http';
import { Observable, throwError } from 'rxjs'
import { catchError, tap, map } from 'rxjs/operators'
import { Movie } from '../shared/movie.model';

@Injectable()
export class MovieService {
    moviesUrl = 'assets/movies.json';

    constructor(private http: HttpClient) { }
```

De HTTP client – lokale JSON

- Methode getMovies wordt aangemaakt in de service:

```
getMovies(): Observable<Movie[]>{  
    return this.http.get<Movie[]>(this.moviesUrl);  
}
```

- De methode geeft een Observable terug met een Movie array.
- Omdat ons Model dezelfde structuur & properties heeft als het json object movies, kan er een automatische conversie gebeuren naar onze klasse
 - Indien dit niet is, kan je ook nog manipuleren via de map operator (zie lab)

De HTTP client – lokale JSON

- In de app.component.ts voorzien we een lokale variabele movieList van het type Movie[].
- In de ngOnInit methode subscriben we op de observable die teruggegeven wordt door de getMovies() methode uit de service:

```
export class AppComponent implements OnInit {
  movieList: Movie[];

  constructor(private serv: MovieService) { }

  ngOnInit() {
    this.serv.getMovies().subscribe(data => this.movieList = data);
  }
}
```

De HTTP client – lokale JSON

- In de HTML component gebruiken we een ngFor om te itereren over de lokale array.

```
<h1>Movie table from json file</h1>
<table>
  <tr>
    <th>Movie title</th><th>Rating</th><th>Released</th>
  </tr>
  <tr *ngFor="let movie of movieList">
    <td>{{movie.title}}</td><td>{{movie.rating}}</td><td>{{movie.isReleased}}</td>
  </tr>
</table>
```

Movie title	Rating	Released
It	5	true
Pitch perfect 3	7	false
Star wars: The last jedi	10	true

Data ophalen via een externe API



De HTTP client – Aanspreken van firebase

- Zie voorbeeld: Resources/Voorbeeld 2: Firebase
- Algemene documentatie is beschikbaar via onderstaande link
 - <https://firebase.google.com/docs/database/rest/start>
- API URL is beschikbaar in het databases venster:



De HTTP client – Aanspreken van firebase - GET

- In De service voorzien we volgende API url:

```
const BASE_API_URL = 'https://webexpert-games.firebaseio.com/';
```

- De get Request wordt toegevoegd in de methode getGames

```
getGames(): Observable<Game[]> {
  return this.http.get(BASE_API_URL + 'games.json').pipe(
    map(res => this.parseData(res)),
    catchError(this.handleError)
  );
}
```

- Deze request krijg alle games uit Firebase. De methode parseData zorgt voor de mapping naar het model Game.
 - Firebase geeft een deel JSON objecten terug, geen array

De HTTP client – Aanspreken van firebase - GET

- Object.keys haalt alle keys uit het JSON object in array vorm
- De map maakt voor elke key in de array een Game object aan
- De methode geeft een array van Games terug

```
parseData(json: any): Game[] {  
    return Object.keys(json).map(key => {  
        let game = new Game(json[key].title, json[key].publisher, json[key].rating, key);  
        return game;  
    });  
}
```

De HTTP client – Aanspreken van firebase – GET met parameters

- Het is mogelijk om requests options mee te geven aan requests
- Hiervoor gebruiken we de types Headers en RequestOptions als volgt:

```
getData(){
    let headers = new Headers();
    headers.append("Content-Type", "application/json");
    let reqOpts = new RequestOptions(headers: headers);
    return http.get('app/data.json', reqOpts);
}
```

De HTTP client – Aanspreken van firebase - POST

- Worden gebruikt om data naar de server te sturen (data toevoegen)
- Syntax:

```
post(url: string, body: any, options?: RequestOptionsArgs):  
Observable<Response>
```

- Voorbeeld:

```
addGame(game: Game) {  
    return this.http.post(BASE_API_URL + 'games.json', game).pipe(  
        catchError(this.handleError)  
    );  
}
```

- Game object wordt automatisch omgezet naar JSON

De HTTP client – Aanspreken van firebase - DELETE

- Wordt gebruikt om data van de server te verwijderen
- Syntax:

```
delete(url: string, body: any, options?: RequestOptionsArgs):  
Observable<Response>
```

- Voorbeeld:

```
deleteGame(game: Game) {  
    return this.http.delete(BASE_API_URL + 'games/' + game.id + '.json').pipe(  
        catchError(this.handleError)  
    );  
}
```

De HTTP client – Aanspreken van firebase - PUT

- Put wordt gebruikt om data aan te maken of aan te passen als geheel object
- Syntax:

```
put(url: string, body: any, options?: RequestOptionsArgs):  
Observable<Response>
```

- Voorbeeld:

```
editGamePut(id: string, newGame: Game) {  
    return this.http.patch(BASE_API_URL + 'games/' + id + '.json', newGame).pipe(  
        catchError(this.handleError)  
    );  
}
```

De HTTP client – Aanspreken van firebase - PATCH

- Patch wordt gebruikt om een gedeelte van een bestaand object te updaten
- Syntax:

```
patch(url: string, body: any, options?: RequestOptionsArgs):  
Observable<Response>
```

- Voorbeeld:

```
editGamePatch(id: string, property: any) {  
    return this.http.patch(BASE_API_URL + 'games/' + id + '.json', property).pipe(  
        catchError(this.handleError)  
    );  
}
```

Handige functies



De async pipe

- Om data asynchroon in de view in te laden kan je gebruik maken van de async pipe
- Handig bij grote hoeveelheden data, doorlopende streams
- De async pipe verwacht een Observable, dus je dient hier in de component niet op te subscriben!
 - De async pipe zorgt zelf voor het subscriben

De async pipe

```
@Component({
  selector: 'app-root',
  template: `
    <ul>
      <li *ngFor="let game of gamesObservable | async">{{ game.title }}</li>
    </ul>
  `,
})
export class AppComponent implements OnInit {
  gamesObservable: Observable<Game[]>

  constructor(private gameService: GameService) { }

  ngOnInit() {
    // Returned een observable, GEEN SUBSCRIBE!
    this.gamesObservable = this.gameService.getGames();
  }
}
```

Pipes

- Manier om data te manipuleren
- Pipe symbool: |
- Pipes combineren is mogelijk
- Built in pipes:
 - uppercase
 - lowercase
 - date
 - currency
 - Number
 - ...

```
 {{ name | lowercase}}  
 {{ birthday | date: 'MMMM dd,yyyy' | uppercase }}  
 {{ price | currency:'EUR':true }}  
 {{ 82.328|number: '2.0-2' }}
```

Pipes

- Zelf pipes maken is mogelijk
- Gebruik angularCLI

```
ng generate pipe capitalize
```

- Voorbeeld pipe om alles om te zetten naar hoofdletters

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

@Pipe({
  name: 'capitalize'
})
export class CapitalizePipe implements PipeTransform {

  transform(value: string, args?: any): any {
    return value.toUpperCase();
  }
}
```

Pipes

- Voeg de pipe toe als declaration in de app.module.ts file
 - AngularCLI doet dit automatisch
- Vervolgens kan je deze gebruiken in de template van je components

```
<ul>
| <li *ngFor="let game of gameList">{{game.title | capitalize }}</li>
</ul>
```

Http en filteren van data

- Filtering van data in frontend/backend afhankelijk van case
 - Bij voorkeur grote datasets in backend
- Filtering in de frontend bij kleine datasets / beperkingen van backend
 - **Geen gebruik maken van custom pipes (bad performance!)**
 - Filtering voorzien in service / component

```
getFilteredGames(searchStr: string){
    return this.http.get(BASE_API_URL + 'games.json').pipe(
        map(res => this.parseData(res)),
        map(res => {
            return res.filter(item => item.title.toLowerCase().includes(searchStr.toLowerCase()));
        }),
        catchError(this.handleError)
    );
}
```

Http en filteren van data

```
@Component({
  selector: 'app-root',
  template: `<input type="text" placeholder="search..." #term (keyup)="getGames(term.value)"/>
    <ul>
      <li *ngFor="let game of gameList$ | async">{{ game.title }}</li>
    </ul>`,
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  gameList$: Observable<Game[]>;

  constructor(private gs: GameService) { }

  ngOnInit(): void{
    this.getGames('');
  }

  getGames(term: string){
    this.gameList$ = this.gs.getFilteredGames(term);
  }
}
```

Http en filteren van data

```
export class AppComponent implements OnInit {
  searchString = new Subject<string>();
  gameList$: Observable<Game[]>;
  constructor(private gs: GameService) { }

  ngOnInit(): void{
    this.gameList$ = this.searchString.pipe(
      debounceTime(300),
      distinctUntilChanged(),
      switchMap(term => this.gs.getFilteredGames(term)),
    );
  }

  search(term): void{
    this.searchString.next(term);
  }
}
```

API's om mee te experimenteren

- [Riot Games](#)
- [import.io](#)
- [Spotify](#)
- [Telegram](#)
- [SoundCloud](#)
- [Reddit](#)
- [YouTube](#)
- [Wunderground](#)
- [Firebase](#)
- [Kandy](#)
- [Star Wars](#)
- [Marvel Comics](#)
- [Mashape](#)
- [Foaas](#)
- [BreweryDB](#)
- [Slack](#)
- [Geo Names](#)
- [Common Crawl](#)
- [Programmers API](#)
- [FitBit](#)
- [JawBone](#)
- [Steam](#)
- [Twilio](#)
- [IBM Watson](#)
- [Algolia](#)
- [Battle.net](#)
- [Free Geo IP](#)
- [The Counted](#)
- [Wolfram Alpha](#)
- [Github](#)
- [Twitter](#)
- [Nutritionix](#)
- [Pokémon API](#)
- [Open Weather Map](#)
- [Yodaspeak](#)
- [Recipe API](#)