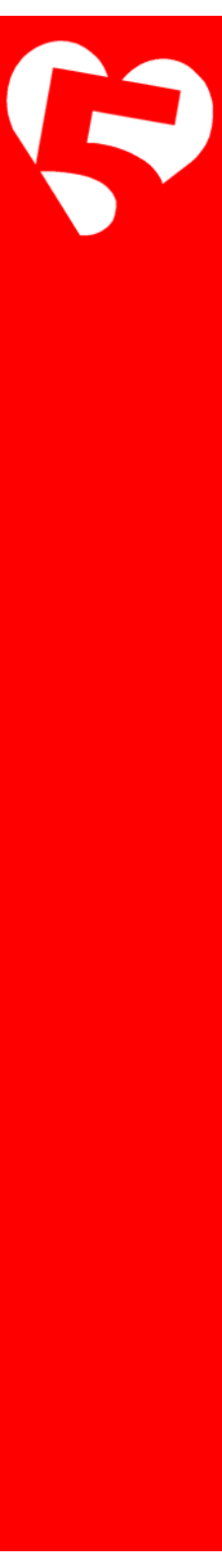


Angular 2

Module 3 - Services



Peter Kassenaar –
info@kassenaar.com



Peter Kassenaar

Angular 2.0

Web Development Library



VAN DUUREN INFORMATICA

Hoofdstuk 5



Services

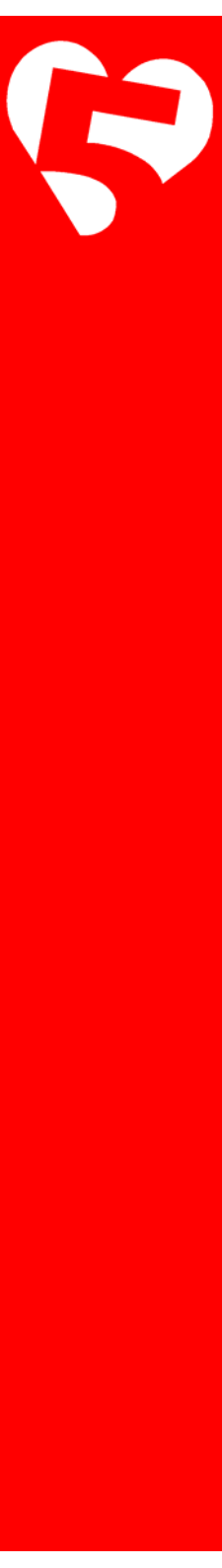
Doel – datafunctionality herbruikbaar maken voor verschillende componenten

- Data retrieval
 - Data caching
 - Data Storage,
 - ...
-
- Angular 2 : één optie
 - `export class myDataService { ... }`

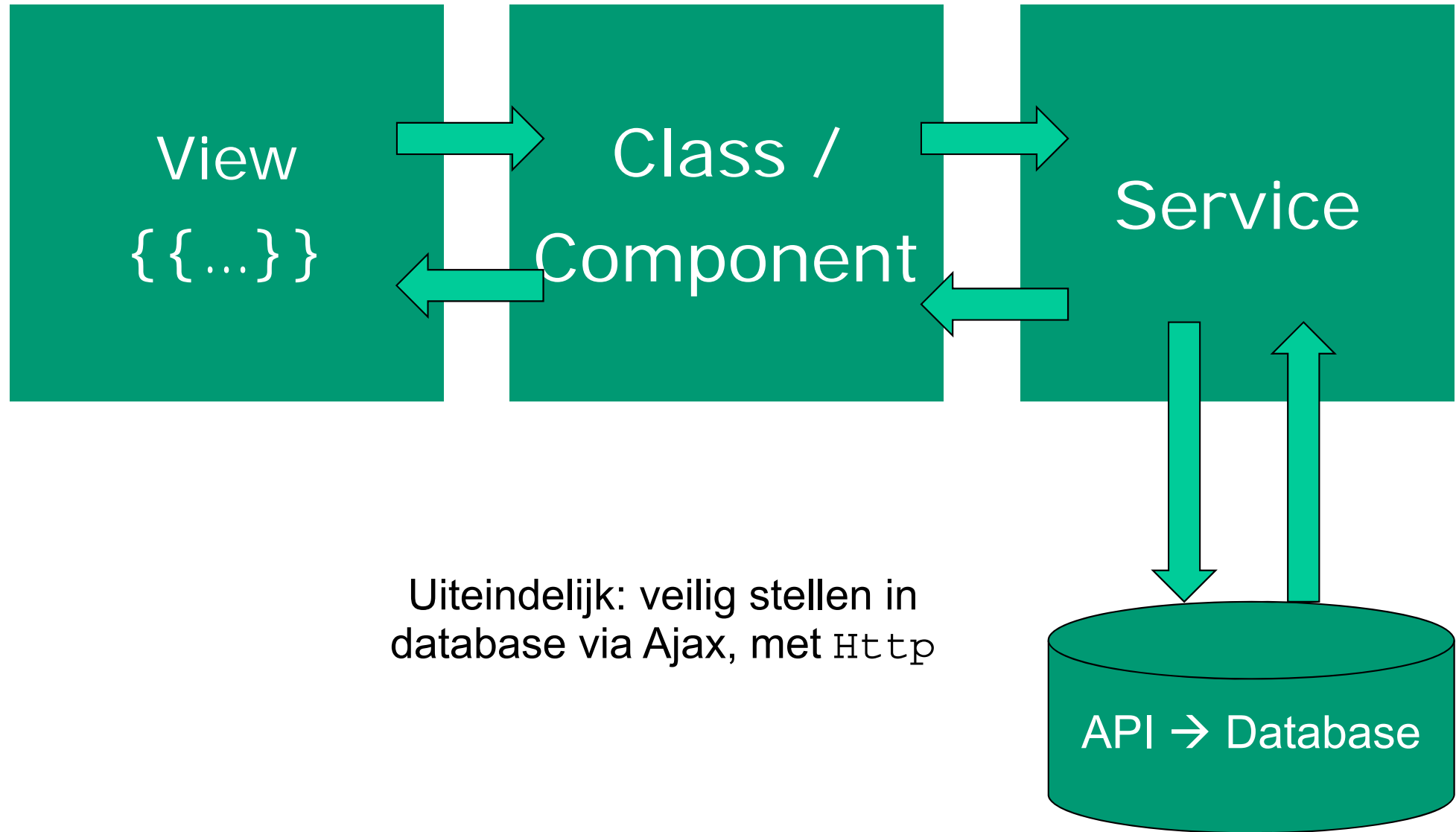


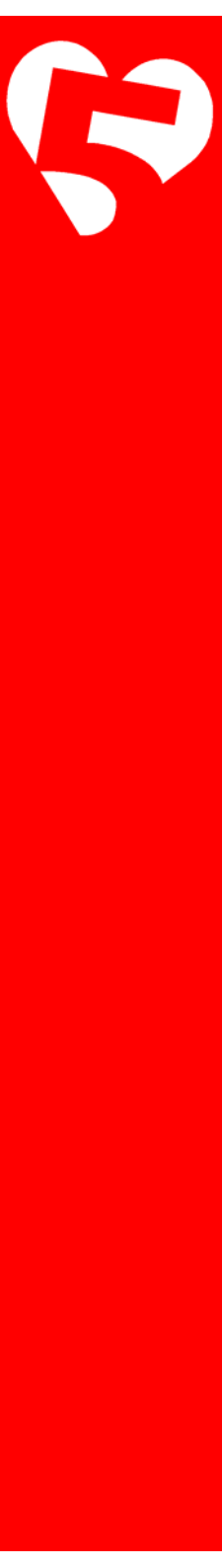
Singleton?

- Services zijn (in principe) singletons
 - Maar: afhankelijk van de plek waar ze geïnstantieerd worden!
 - Ze zijn een singleton voor de Module en alle child components.
 - Module/Site-wide gebruiken? Instantieer service in `app.module.ts`

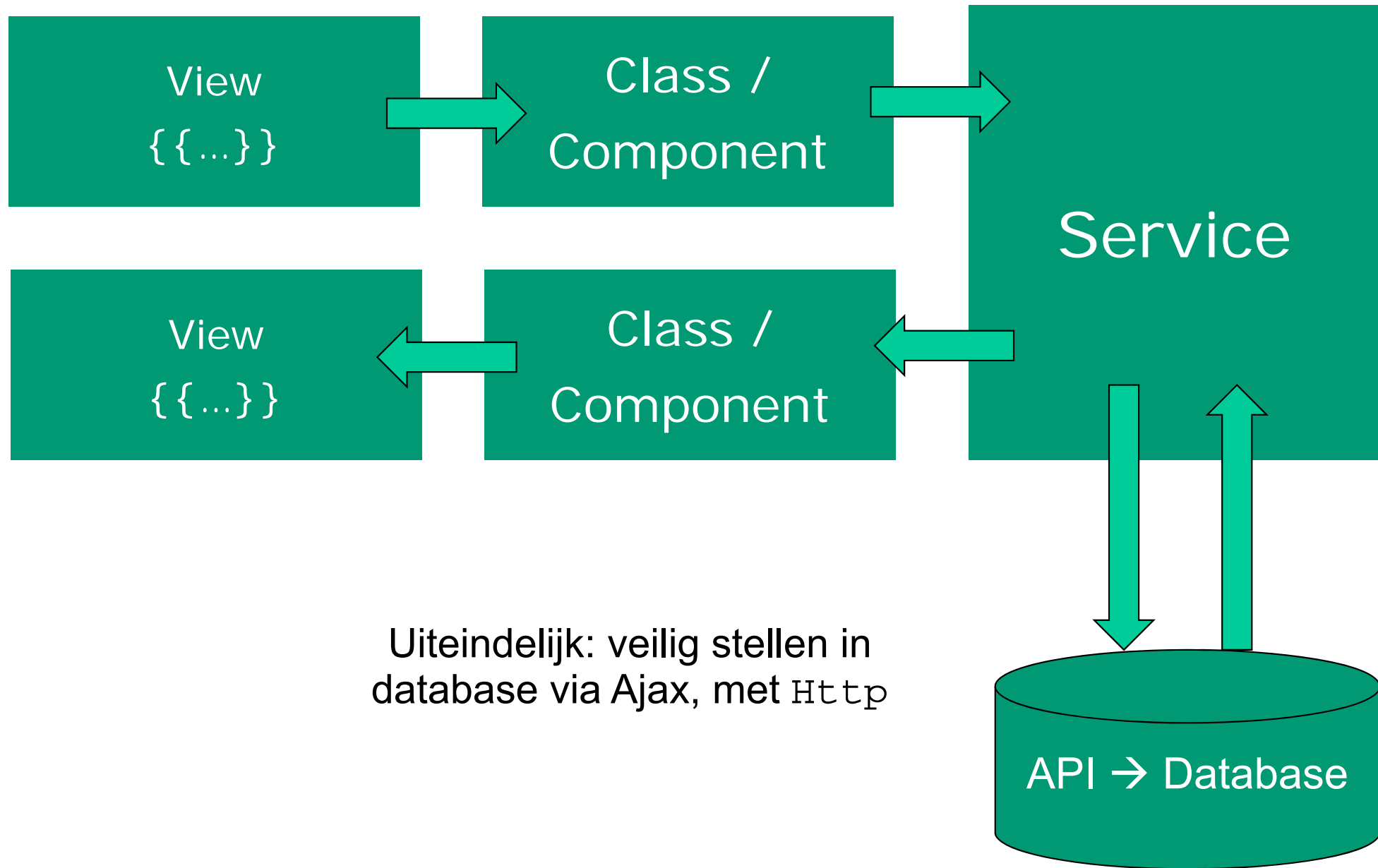


Data flow





Data flow



Uiteindelijk: veilig stellen in
database via Ajax, met `Http`



Services in Angular 2

Data services in Angular 1:

```
angular.module( 'myApp' )  
  .service(...)  
  .factory(...)  
  .provider(...)
```

Data services in Angular 2:

```
import {Injectable} from '@angular/core';  
  
@Injectable()  
export class CityService{  
  //....  
}
```



De rol van @Injectable

Why? – Dependency Injection (DI) en metadata!

“TypeScript sees the @Injectable() decorator and emits metadata about our service, metadata that Angular may need to inject other dependencies into this service.”

<https://angular.io/docs/ts/latest/tutorial/toh-pt4.html>



“Our service doesn't have any dependencies at the moment. Add the decorator anyway.”

*It is a best practice to apply the
@Injectable() decorator from the start both
for consistency and for future-proofing”*



Stap 1 – service maken (static data)

```
import { Injectable } from '@angular/core';
import { City } from './city.model'

@Injectable()
export class CityService {
  cities:City[] = [
    new City(1, 'Groningen', 'Groningen'),
    ...
  ];

  // retourner alle cities
  getCities() {
    return this.cities
  }

  // retourner city op basis van ID
  getCity(id:number) {
    return this.cities.find(c => c.id === id);
  }
}
```



Stap 2 – Service consumeren/injecten

```
...  
import {CityService} from "../city.service";
```

```
@Component({  
  selector    : 'hello-world',  
  templateUrl: 'app/app.html',  
})
```

```
export class AppComponent implements OnInit {  
  // Properties voor de component/class  
  currentCity: City;  
  cities: City[];  
  cityPhoto: string;
```

local
variables

```
  constructor(private cityService: CityService) {  
  
  }
```

```
  ngOnInit() {  
    this.cities = this.cityService.getCities();  
  }
```

```
  getCity(city: City) {  
    this.currentCity = this.cityService.getCity(city.id);  
    this.cityPhoto   = `img/${this.currentCity.name}.jpg`;  
    console.log('City opgehaald:', this.currentCity);  
  }
```

Constructor: shorthand voor
nieuwe private variable +
instantiëring!

Detailgegevens voor
city bij (click) event



Ho, ho, dat gaat snel!

- Let op: geen `new()` instantie van de Service!
 - Services zijn Singletons
 - Worden opgehaald uit de Module en/of geïnstantieerd in een `constructor()`

```
constructor(private cityService:CityService) { ... }
```

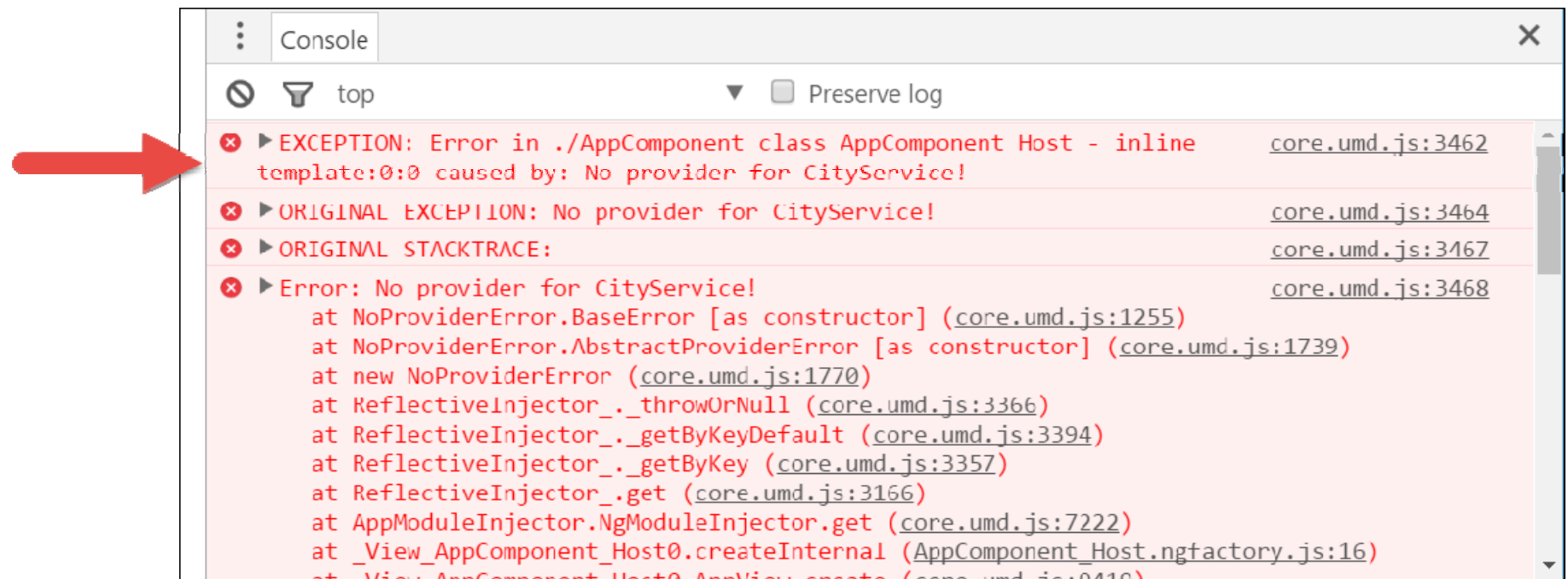
“The constructor itself does nothing.

*The parameter simultaneously defines a
private cityService property and identifies it
as a CityService injection service.”*



“No provider for CityService”

- Solution: inject in `app.module.ts`





Service injecteren in Module

- Alleen de *referentie* naar CityService is niet voldoende.
- Angular moet de service *injecteren* in de module
- Gebruik de annotatie `providers: [...]`

// Module declaration

```
@NgModule({  
  imports      : [BrowserModule],  
  declarations: [AppComponent],  
  bootstrap    : [AppComponent],  
  providers    : [CityService] // DI voor service  
})  
  
export class AppModule {  
}
```



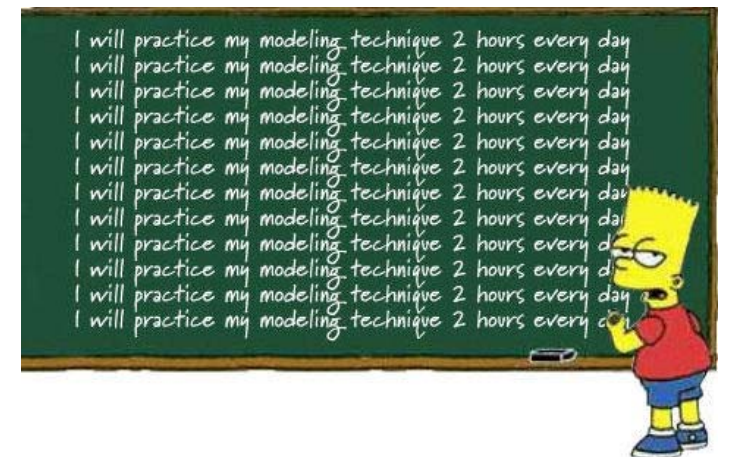
Array met
Service-
dependencies



Checkpoint

- Elke service in Angular 2 is een `class`
- Class importeren in de component die hem gebruikt
- Instantiëren in `constructor()`
- Service invoegen in de Module
- Oefening 5a) + 5b)

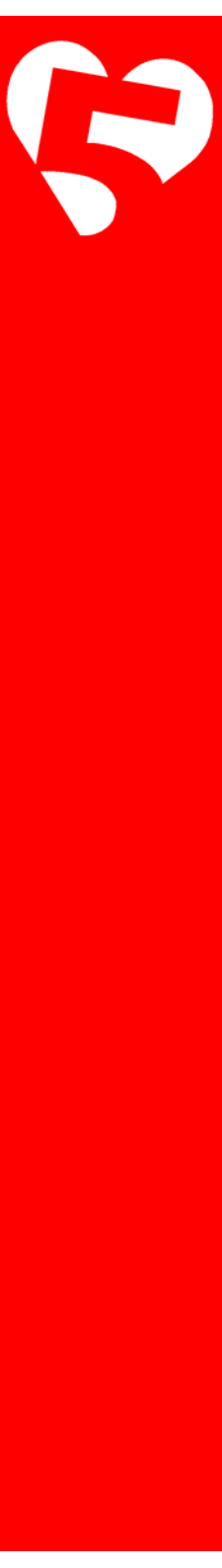
Oefening....





Async services

Werken met Live Data



Hoofdstuk 6



Async Services

- Statische data ophalen: *synchrone* actie
- Werken via `Http`: *asynchrone* actie
- Angular 1: `Promises`
- Angular 2: `Observables`

Bovendien in Angular 2: ReactiveX library `RxJS`



ReactiveX

An API for asynchronous
with observable stream

Choose your platform

Languages

- Java: [RxJava](#)
- JavaScript: [RxJS](#)
- C#: [Rx.NET](#)
- C#(Unity): [UniRx](#)
- Scala: [RxScala](#)
- Clojure: [RxClojure](#)
- C++: [RxCpp](#)
- Ruby: [Rx.rb](#)
- Python: [RxPY](#)
- Groovy: [RxGroovy](#)
- JRuby: [RxJRuby](#)
- Kotlin: [RxKotlin](#)
- Swift: [RxSwift](#)

ReactiveX for platforms and frameworks

- [RxNetty](#)
- [RxAndroid](#)
- [RxCocoa](#)

<http://reactivex.io/>

DOCUMENTATION

[Observable](#)
[Operators](#)
[Single](#)
[Subject](#)

LANGUAGES

[RxJava](#)
[RxJS](#)
[Rx.NET](#)
[RxScala](#)

RESOURCES

[Tutorials](#)

COMMUNITY

[GitHub](#)
[Twitter](#)
[Others](#)

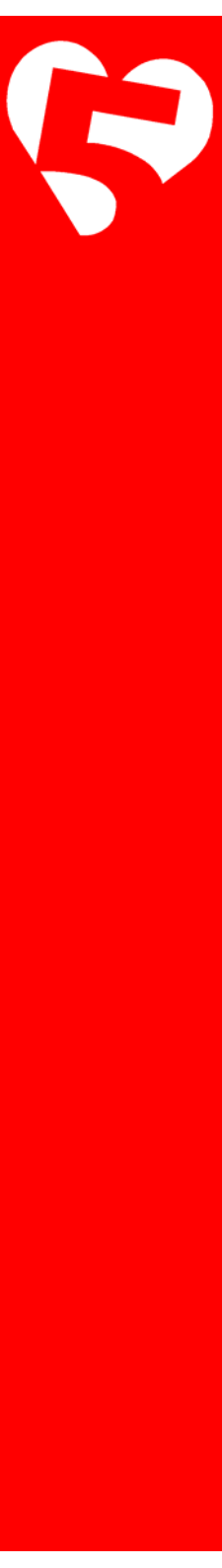


Waarom Observables?

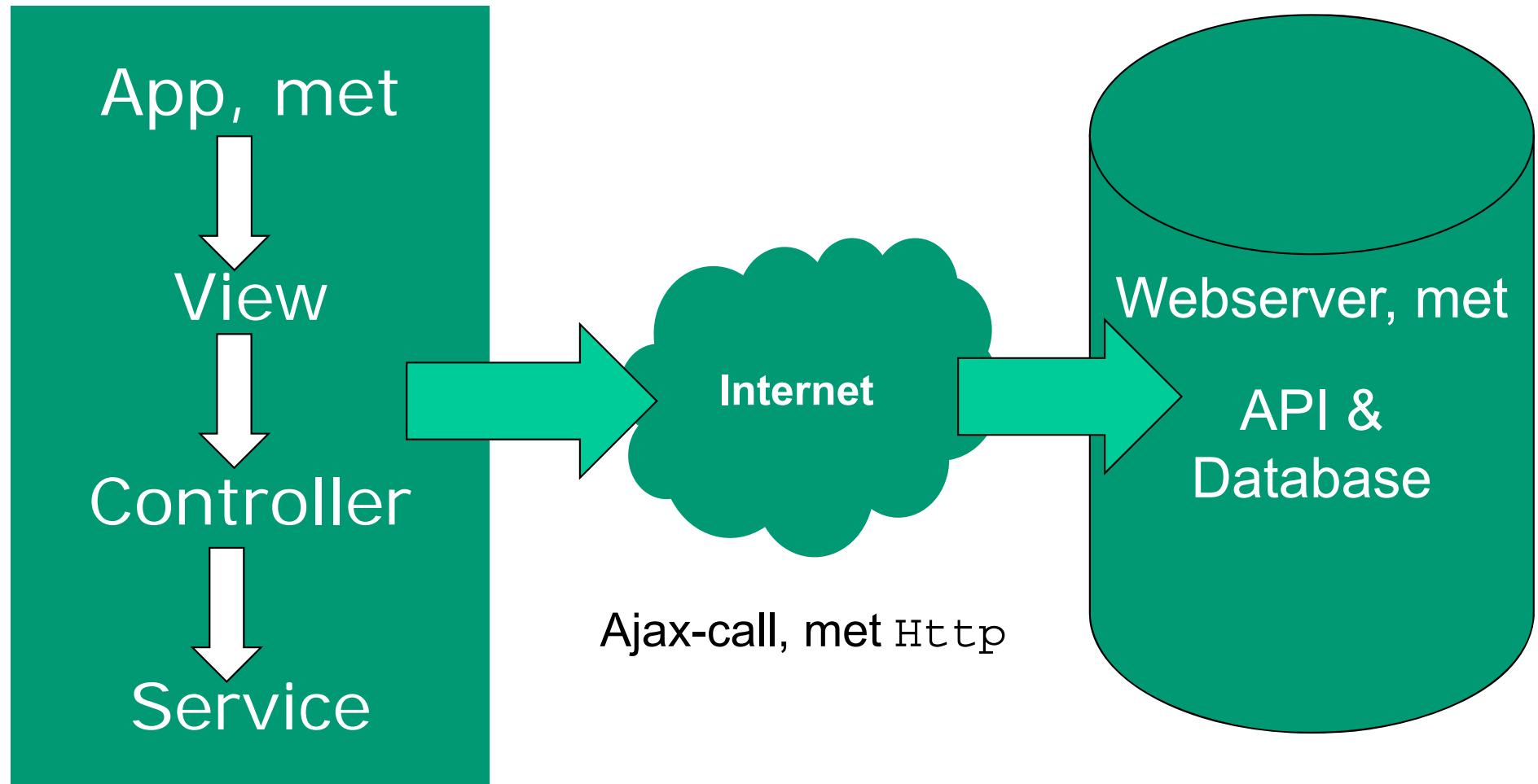
We can do much more with observables than with promises.

With observables, we have a whole bunch of operators to pull from, which let us customize our streams in nearly any way we want.

<https://auth0.com/blog/2015/10/15/angular-2-series-part-3-using-http/>



Async services





Voorbeeld Http / cities.json

Maak een data-bestand; hier: `cities.json`

```
[  
  {  
    "id"      : 1,  
    "name"    : "Groningen",  
    "province": "Groningen",  
    "highlights": "Martinitoren"  
  },  
  ...  
]
```



Stap 1 - Http injecteren in Service

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

@Injectable()
export class CityService {

    constructor(private http:Http) {

    }

    // retourneer alle cities
    getCities(): Observable<Response> {
        return this.http.get('app/cities.json')
    }

    ...

}
```

**Maak lokale
variabele http**

**Retourneer
observable naar de
Component**



Stap 2 – Component aanpassen

```
export class AppComponent implements OnInit {  
  // Properties voor de component/class  
  currentCity: City;  
  cities: City[];  
  cityPhoto: string;  
  
  constructor(private cityService: CityService) {  
  
  }  
  
  ngOnInit() {  
    this.cityService.getCities()  
      .subscribe(cityData => {  
        this.cities = cityData.json()  
      },  
      err => console.log('FOUT: ', err),  
      () => console.log('Getting cities complete'))  
  }  
}
```

Data async ophalen
en .subscribe()
gebruiken



Stap 3 – Module aanpassen

```
// Angular Modules
```

```
...
```

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

```
...
```

```
// Module declaration
```

```
@NgModule({
```

```
  imports      : [BrowserModule, HttpModule],
```

```
  declarations: [AppComponent],
```

```
  bootstrap   : [AppComponent],
```

```
  providers    : [CityService] // DI voor service
```

```
})
```

```
export class AppModule {
```

```
}
```

HttpModule
importeren en
toevoegen



Meer over Observables

- Onderdeel van RxJs
- Drie parameters:
 - success
 - error
 - complete

```
this.cityService.getCities()
```

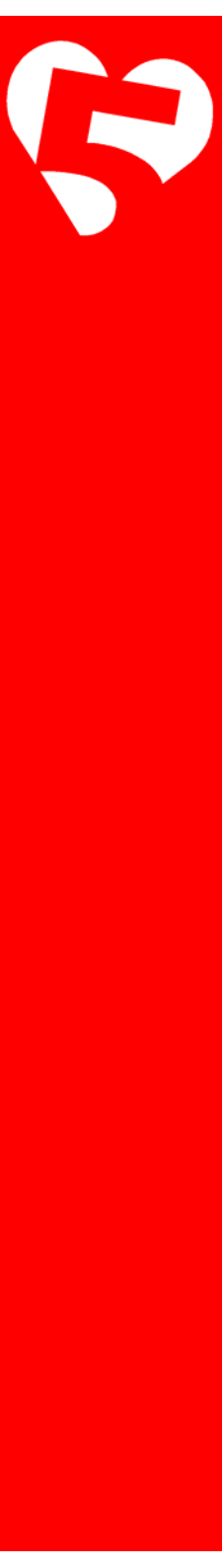
```
    .subscribe(cityData => {  
        this.cities = cityData.json();  
    },  
    err => console.log(err),  
    () => console.log('Getting cities complete...')  
)
```





Observables en RxJs

- “Reactive Programming”
 - *“Reactive programming is programming with asynchronous data streams.”*
 - <https://gist.github.com/staltz/868e7e9bc2a7b8c1f754>
- Observables hebben extra mogelijkheden ten opzichte van Promises
 - Mapping
 - Filtering
 - Combining
 - Cancel
 - Retry
 - ...
- Gevolg: géén `.success()`, `.error()` en `.then()` chaining meer!



AC

ANGULAR CONNECT

RANGLE.IO

REWRITING THE W

Progre

energy

eSynergy

prohire

MONACA

Chakra UI

ANGULAR BOOT CAMP! STL - SF - DC - NYC LONDON - ONLINE

Welcome to go beast mode with realtime interactive interfaces in Angular and Firebase.

0:06 / 25:21

Go beast mode with realtime reactive interfaces in Angular 2 & Firebase | Lukas Ruebbelke

<https://www.youtube.com/watch?v=5CTL7aqSvJU>



Observable Cheat Sheet

genius to understand.

You can download the full-sized infographic at <http://bit.ly/observable-cheat-sheet>.

I really hope that you find the infographic helpful. Be sure to drop me a line below if you have any questions or comments. #highFive

OBSERVABLE CHEAT SHEET

Learning to work with observables is much like learning a new super power in that the entire process can be overwhelming! When you set aside all of the super shiny RxJS operators that you have at your disposal and start with a few key concepts, things suddenly start to come into focus and become fun.

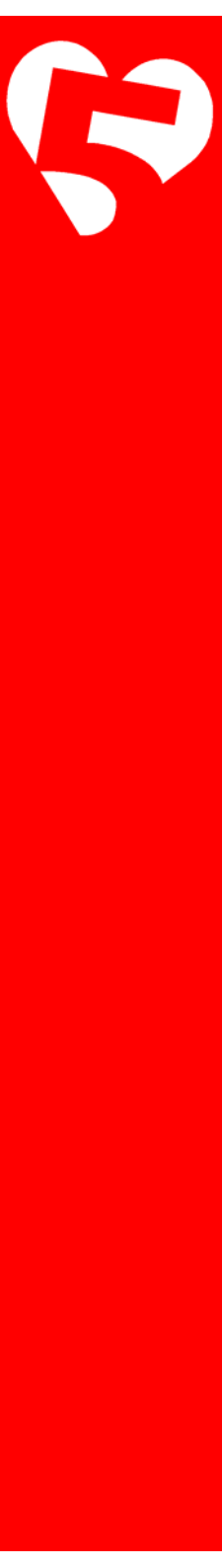
BASIC OBSERVABLE SEQUENCE

The basic observable sequence is the foundation of everything we do with observable streams. In its simplest form, we have an **initial output** of data that we capture and then determine where we will **input** it into the application in its **final** form. We refer to data that arrives in the subscribe block as **final input** because it is no longer under control of the stream as it is being inputted in its final form to the application.

I AM OUTPUTTING DATA!


```
Observable.fromEvent(this.btn, 'click')
```

<http://onehungrymind.com/observable-cheat-sheet/>



Hello RxJS

Gratis online training



Hello RxJS

5% COMPLETE

Class Curriculum

Your Instructor

Class Curriculum

Start next lecture > Presentation: Realtime Observable Streams

Hello RxJS

<input checked="" type="radio"/>	Presentation: Realtime Observable Streams	Start
<input type="radio"/>	Slides: Realtime Observable Streams	Start
<input type="radio"/>	The Basic Observable Sequence (2:09)	Start
<input type="radio"/>	Lab: The Basic Observable Sequence	Start
<input type="radio"/>	Mapping Values (2:22)	Start
<input type="radio"/>	Lab: Mapping Values	Start
<input checked="" type="radio"/>	Maintaining State (3:25)	
<input type="radio"/>	Lab: Maintaining State	Start
<input type="radio"/>	Merging Streams (1:57)	Start
<input type="radio"/>	Lab: Merging Streams	Start
<input type="radio"/>	Mapping to Functions (5:18)	Start
<input type="radio"/>	Lab: Mapping to Functions	Start

<http://courses.ultimateangular.com/>



Observables in een Angular 2-applicatie

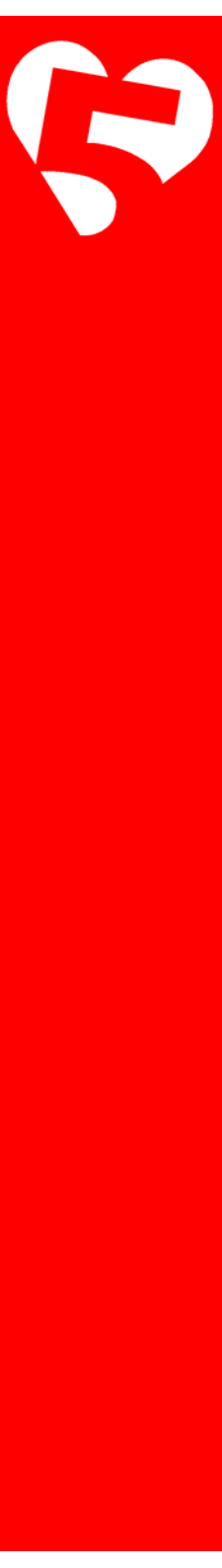
- Importeer Rx in de applicatie
 - Geheel, of alleen de benodigde onderdelen

```
import 'rxjs/Rx';  
import {Observable} from "rxjs";
```
- Gebruik observable functies als `.map()`, `.filter()` etc.
- Piping. Resultaat van de een functie dient als invoer voor de volgende functie.



```
_getCities() {  
  if (!this.cities) {  
    this._cityService.getCities()  
      .map(res => res.json())  
      .delay(3000)  
      .subscribe(cityData => {  
        this.cities = cityData;  
      },  
        err => console.log(err),  
        () => console.log('Getting cities complete...'))  
  }  
}
```

RxJs-functies



THOUGHTRAM

TRAININGCODE REVIEWBLOG

1

2

2

3

TAKING ADVANTAGE OF
OBSERVABLES IN
distinctUntilChanged()
ANGULAR 2

by Christoph Burgdorf on Jan 6, 2016 (updated on May 12, 2016)
12 minute read

Some people seem to be confused why Angular 2 seems to favor the Observable abstraction over the Promise abstraction when it comes to dealing with async behavior.

There are pretty good resources about the difference between Observables and Promises already out there. I especially like to highlight this free 7 minutes video by Ben Lesh on egghead.io. Technically there are a couple of obvious differences like the *disposability* and *lazyness* of Observables. In this article we like to focus on some practical advantages that

<http://blog.thoughttram.io/angular/2016/01/06/taking-advantage-of-observables-in-angular2.html>



RxJs-functies

For example, you can ask the server to retry your call several times:

```
1 ...  
2     return this.http.get('/api/v1/tasks.json')  
3         .retry(5)  
4         .map( res => res.json());  
5     ...
```

In addition you can poll for results - by using `Observable.interval`:

```
1 ...  
2     pollTasks() {  
3         return Observable.interval(10000)  
4             .flatMapLatest(() => http.get('/api/v1/tasks.json'))  
5             .map(res => res.json())  
6     }  
7     // caller can do subscription and store it as a handle:  
8     let tasksSubscription =  
9         pollTasks()  
10        .subscribe( data => this.payload = data);  
11    // turn it off at a later time  
12    tasksSubscription.unsubscribe();
```

RxJs-functies

What the what? Ok, so the `pollTasks()` method emits a call every 10 seconds, which triggers the call inside of `flatMapLatest` - we're basically ignoring the result of that event, and using it to trigger the `http.get` method to fetch our data. We'll map it into JSON each time.

Meer over observables <http://chariotsolutions.com/blog/post/angular2-observables-http-separating-services-components/>



RxJS-operators in de service

```
import {Injectable} from '@angular/core';  
import {Http, Response} from "@angular/http";  
import {Observable} from "rxjs";  
import 'rxjs/add/operator/map';
```

Import operator

```
@Injectable()  
export class CityService {  
  
    constructor(private http: Http) {  
  
    }  
  
    // retourner alle cities  
    getCities(): Observable<Response> {  
        return this.http.get('app/cities.json')  
            .map(cities => cities.json());  
    }  
}
```

Transform stream
in de service



Werken met Live API's

- MovieApp
- Oefeningen\09-services-live





Voorbeeld API's

- <https://pokeapi.co/> - Pokemon API
- <http://openweathermap.org/API> (weerbericht)
- <http://filltext.com/> (random NAW-gegevens)
- <http://ergast.com/mrd/> - Ergast Motor (F1) API
- <http://www.omdbapi.com/> - Open Movie Database
- <http://swapi.co/> - Star Wars API
- Zie ook `JavaScript APIs.txt` met meer voorbeelden



Checkpoint

- Elke service in Angular 2 is een `class`
- Class injecteren in de component die hem gebruikt
- Instantiëren in `constructor()`
- Aanbevolen: leren werken met `RxJs`
- Werken met statische data en werken met live data

Oefening....

