

Web Frameworks

Angular – deel 7

Elektronica – ICT

Sven Mariën

(sven.marien01@ap.be)

Inhoud

1. Deel 1

1. Waarom Angular ?
2. Nieuw Angular project
 - Aanmaken
 - Bestandsstructuur
3. Opbouw
 - Van een Applicatie
 - Van een Component
4. Nieuwe Component Aanmaken

2. Deel 2

1. Interpolation
2. Pipes

3. Deel 3

1. Directives

4. Deel 4

1. Property binding
2. Event binding

5. Deel 5

1. Two-way binding
2. Parent-child componenten

6. Deel 6

1. Routing

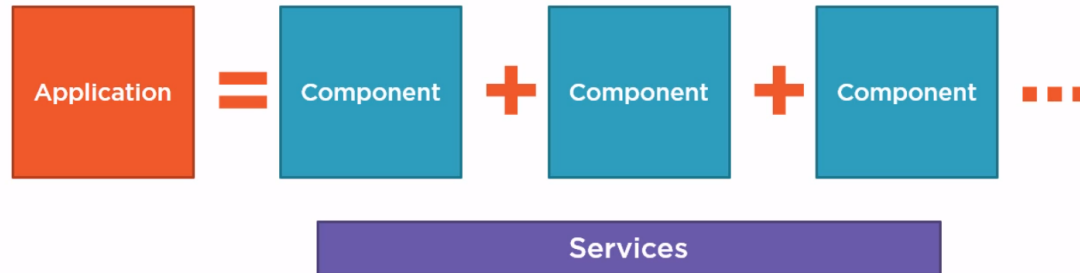
7. Deel 7

1. **Services**

6. Services

- Tot nu zijn we in staat om componenten te bouwen die volledig losstaand van elkaar kunnen werken.
- We hebben nog geen manier om:
 - Tussen componenten onderling gegevens uit te wisselen
 - Vanuit een component een externe gegevensbron aan te spreken (bv. Een webservice)
- Het framework heeft hiervoor zogenaamde '**Services**' voorzien.

Anatomy of an Angular Application



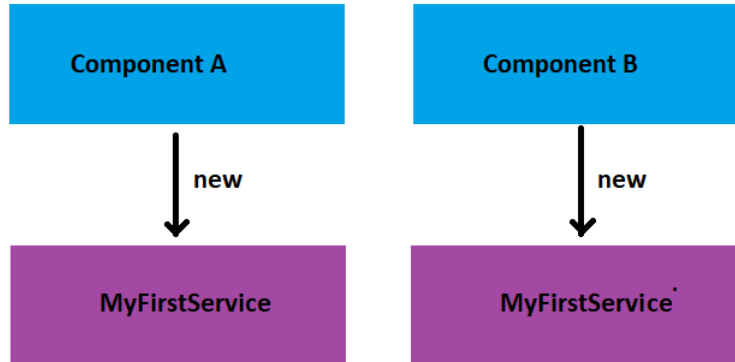
Wat is een Angular service ?

- Een service is een klasse
- De Naam eindigt per afspraak met '**Service**'
- De klasse wordt uiteraard ook gemarkeerd met het '**export**' keyword
- De klasse wordt voorzien van een decorator '@Injectable()'
- Aanmaken van een lege service via de CLI:
 - ng generate service MyFirst
 - **ng g s MyFirst** (verkorte versie)

```
1 import { Injectable } from '@angular/core';
2
3 @Injectable({
4   providedIn: 'root'
5 })
6 export class MyFirstService {
7
8 }
```

Hoe de service instantiëren ?

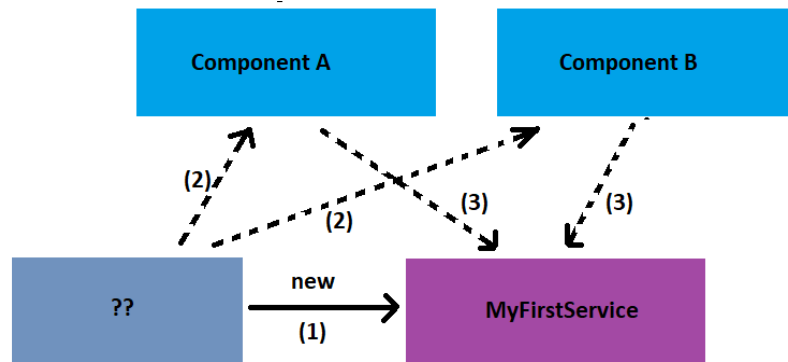
- Als we de service vanuit de component zouden instantiëren via het '**new**' keyword dan bekomen we uiteraard verschillende instanties van service objecten.



- Dit is **niet** de bedoeling aangezien we net gegevens willen 'delen' tussen componenten

Hoe de service instantiëren (2) ?

- We kunnen stellen dat beide componenten **afhankelijk** zijn van de Service. Zij hebben de service immers nodig om te kunnen functioneren maar ze kunnen het service object niet zelf aanmaken.
- We hebben een 3^e partij nodig die de instantie (singleton) aanmaakt en een referentie doorgeeft aan de verschillende componenten.

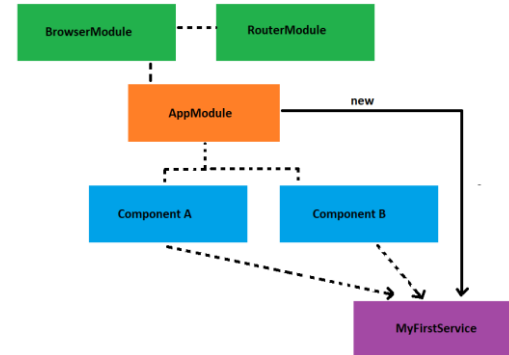


- Hiervoor bestaat een oplossing (die ook in Angular zit ingebakken): “**dependency injection**”

Dependency Injection

- Door het toevoegen van de **@Injectable** decorator bij de service zal het framework ook automatisch instaan voor het instantiëren van het service object.
- We moeten enkel nog een referentie naar de service zien door te geven aan de Componenten die er een afhankelijkheid (=dependency) op hebben.
- Dit gebeurt via de **constructor** van de Componenten in kwestie
- Het framework zal dus de 'dependencies' als het ware '**injecteren**' in de componenten
- De componenten krijgen dezelfde referentie door en werken bijgevolg met hetzelfde service object.

```
@Component({
  selector: 'app-my',
  templateUrl: './my.component.html'
})
export class MyComponent {
  constructor(private service : MyFirstService) {
    //....
  }
}
```



Aanroepen van externe (http) services : **HttpClient**

- We willen uiteraard ook gegevens kunnen opvragen van http services die zich ergens op een server bevinden.
 - Ophalen van “Weerinformatie”
 - Ophalen van informatie van “De Lijn”
 - Ophalen van informatie van “Antwerpen Digitaal”
 - Maar ook het ophalen of wegschrijven van informatie van/naar onze “eigen” server
- Deze “calls” moeten asynchroon gebeuren (ajax) omdat:
 - We niet weten hoe snel de data zal terugkomen (bv. afhankelijk van de belasting van de externe server)
 - We niet (altijd) weten hoeveel data er zal terugkomen
 - We niet weten of de server wel online is of zal reageren
 - We niet willen dat onze web toepassing gedurende langere tijd geblokkeerd wordt (slechte user experience)
 - ...
- We gebruiken hiervoor de Angular **HttpClientModule** en de **HttpClient** klasse

HttpClient

- Kan gebruikt worden om data op een asynchrone wijze:
 - Op te halen => GET
 - Weg te schrijven => POST / PUT
- Aan de hand van de service URL, bv.
 - De vertrekken van een bepaalde halte:
 - <https://www.delijn.be/rise-api-core/haltes/vertrekken/200144/7>
 - Weerinfo van Antwerpen:
 - <http://api.openweathermap.org/data/2.5/weather?q=antwerpen&lang=nl&APPID=c29dbdf3ccc2d57a361ceaeac49d9e53>
 - Lijst van het Leegstandsregister van Open data Antwerpen:
 - <https://datasets7.antwerpen.be/leegstandsregister/2019>
 - Veelal geven de services '**JSON**' data terug, deze zal door de HttpClient worden omgezet naar "objecten", zodat we de data in code kunnen uitlezen en verwerken

Gebruik van de HttpClient vanuit een service

- De **HttpClientModule** moet worden toegevoegd in de AppModule
- De **HttpClient** zal via Dependency Injection (DI) in de service worden geïnjecteerd
- De HttpClient heeft een 'get' methode om een http GET request te doen met de service URL als parameter

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

```
@NgModule({  
  declarations: [  
    AppComponent,  
    HomeComponent,  
    WeatherComponent  
  ],  
  imports: [  
    BrowserModule,  
    FormsModule,  
    HttpClientModule,  
    MDBBootstrapModule.forRoot(),  
    RouterModule.forRoot()
```

```
import { Injectable } from "@angular/core";  
import { HttpClient } from "@angular/common/http";
```

```
@Injectable()  
export class LeegstandService {  
  constructor(private _http: HttpClient) {}  
  
  get Lijst() {  
    return this._http.get("http://datasets.antwerpen.be/v4/gis/lvleegstandsregister.json");  
  }  
}
```

Observables

- De get methode van de HttpClient geeft een **Observable** terug als resultaat.
- Dit heeft te maken de asynchrone werking van de HttpClient.
- Een observable is een object waarop men een '**subscribe**' dient te doen.
- Pas **NA een subscribe** zal de http GET request daadwerkelijk worden uitgevoerd.
- Wanneer het resultaat van de request is aangekomen op de client zal de functie in de subscribe worden uitgevoerd.
- De subscribe wordt typisch aangeroepen vanuit de component(en) die de data wil opvragen aan de service.

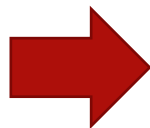
```
export class LeegstandLijstComponent {  
  
  lijst;  
  
  constructor(private service: LeegstandService) {  
    this.service.Lijst.subscribe(d => this.lijst = d);  
  }  
}
```

```
@Injectable()  
export class LeegstandService {  
  constructor(private _http: HttpClient) {}  
  
  get Lijst() : Observable<Object> {  
    return this._http.get(<a href="http://datasets.antwerpen.be">http://datasets.antwerpen.be</a>);  
  }  
}
```

Wat geeft de HttpClient als resultaat ...?

- Standaard “weet” de get methode uiteraard niet welke data er terugkomt en geeft deze een **Observable<Object>** type terug.
- Dat wil zeggen dat de subscriber van de observable een **Object** zal terug krijgen.
- We kunnen hiervoor beter een **interface** voorzien zodat we als ontwikkelaar efficiënter code kunnen schrijven om de data te verwerken.
- Deze interface kan je ofwel:
 - Zelf aanmaken aan de hand van de gegevens van de betreffende service
 - Automatisch laten genereren aan de hand van JSON data via de website <http://json2ts.com/>

```
{
  "id": 10, "objectid": 653046, "geometry": "{ \"type\": \"Po\",
  [4.4196236130751, 51.217934642287], [4.4195756155373, 51.
  [4.4197255200369, 51.218041859954]]]", "shape": null, "p
  tcode": "2018", "reg_entiteit": "Verdieping 1", "reg_aard"
  {
    "id": 13, "objectid": 653047, "geometry": "{ \"type\": \"Po\",
    [4.4196236130751, 51.217934642287], [4.4195756155373, 51.
    [4.4197255200369, 51.218041859954]]]", "shape": null, "p
    tcode": "2018", "reg_entiteit": "Handelsgeleijkvloers", "r
```



```
export interface ILeegstand {
  id: number;
  objectid: number;
  geometry: string;
  geometry2: IGeometry;
  shape?: any;
  pnd_id: string;
  pva_straat: string;
  nva_huisnr1: string;
```

Observable<....>

- De Observable maakt immers gebruik van **generics**.
- Op deze manier kunnen we onze code aanpassen en is het duidelijk wat de service ons aanlevert (in dit voorbeeld een array van Leegstand objecten)

```
export interface ILeegstand {  
  id: number;  
  objectid: number;  
  geometry: string;  
  geometry?: TGeometry;  
}
```

```
export class LeegstandLijstComponent {  
  
  lijst : ILeegstand[];  
  
  constructor(private service: LeegstandService) {  
    this.service.Lijst.subscribe(d => this.lijst = d);  
  }  
}
```

```
@Injectable()  
export class LeegstandService {  
  constructor(private _http: HttpClient) {}  
  
  get Lijst() : Observable<ILeegstand[]> {  
    return this._http.get<ILeegstand[]>("http://datasets.antwerpen.  
  }  
}
```

```
<tbody>  
  <tr *ngFor="let item of lijst" (click)="SelectItem(item)">  
    <th>{{item.pnd_id}}</th>  
    <td>{{item.pva_straat + " " + item.pva_huisnr1}}</td>  
    <td>{{item.reg_opnamedatum | date:'dd-MMM-yyyy'}}</td>  
  </tr>  
</tbody>
```

Component Lifecycle

- Een laatste stap die nodig is om de service te kunnen afwerken is het kennen van de **Component Lifecycle**.
- Elke Angular component heeft immers een **levenscyclus met verschillende “fasen”**.



- Het framework maakt het mogelijk om op elke fase “in te haken”.
- Dit kan door het implementeren van de juiste interface in de component
 - OnInit, OnChanges, OnDestroy,...

```
export class LeegstandLijstComponent implements OnInit, OnChanges, OnDestroy {  
  
  constructor()  
  {}  
  
  ngOnInit() {  
  }  
  
  ngOnChanges(changes: SimpleChanges): void {  
  }  
  
  ngOnDestroy(): void {  
  }  
}
```

Aanroepen van een service: OnInit

- De Angular documentatie raadt immers aan om een http service
 - **Niet** aan te roepen vanuit de **constructor** van een component
 - Maar wel in de '**OnInit**' fase. Deze wordt aangeroepen onmiddellijk nadat een component wordt aangemaakt.

```
//  
export class LeegstandLijstComponent implements OnInit {  
  
    lijst : ILeegstand[];  
  
    constructor(private service: LeegstandService) {  
    }  
  
    ngOnInit() {  
        this.service.Lijst.subscribe(d => this.lijst = d);  
    }  
}
```

Goed alternatief: Promises

- In plaats van met **Observables** kunnen we ook werken met **Promises**
- Een observable kan immers worden omgezet via de **.toPromise()** methode
- Voordeel van promises is dat we gebruik kunnen maken van **async / await**
- Hiermee krijgen we veel beter gestructureerde broncode (zeker indien je meerdere calls na elkaar moet laten uitvoeren).

```
GetWeather()  
{  
  return this.http.get('http://api.openweathermap.org/data/2.5/weather?q=oulu&lang=nl&APPID=c29dt  
  .toPromise();  
}
```

```
async ophalen () {  
  try {  
    console.log("weerinfor ophalen")  
    this.weatherinfo = await this.svc.GetWeather();  
    console.log("de info is binnen")  
  }  
  catch (error) {  
    console.log("Er is een fout opgetreden")  
  }  
}
```


Oefeningen: Services

- Gebruik de Calculator Component uit de vorige oefening (met de button) en geef deze **component** 2x weer op het scherm. Test de knoppen. Je zal merken dat beide componenten onafhankelijk werken en dat de beide componenten (uiteindelijk) elk een eigen counter hebben.
- We willen nu deze beiden laten samenwerken via een service zodat de beiden steeds dezelfde counter gebruiken en dus dezelfde waarde weergeven op de buttons.
- Maak een nieuwe service aan: **MyShareService**
- Gebruik deze in de Calculator Component (via dependency injection)
- Verhuis de 'counter' property vanuit de component naar de service
- Maak in de component hiervoor een 'get'ter => get counter() die de waarde ophaalt uit de service
- Test opnieuw. De componenten gebruiken nu beiden dezelfde counter die zich in de service bevindt. Ze geven beide dus steeds dezelfde waarden aan

Services(2)

- Maak een nieuwe component aan : **WelcomeSelectComponent**
- Geef deze component weer op het scherm net boven de WelcomeComponent
- Deze zal enkel de dropdown (select) lijst bevatten die zich in de WelcomeComponent bevindt (kopieer deze daar uit)
- Zorg er met een service voor dat je ook met deze dropdown de afbeelding in de WelcomeComponent kan aanpassen
- Zorg er bovendien voor dat het in beide richtingen werkt. Dus welke lijst ook wordt aangepast, beiden zullen steeds dezelfde image tonen.

Image 77



Image 77

Services (3)

- Werk de 'De Lijn' Component verder af, zodat deze met 'live' data werkt.
- Service Url: <https://www.delijn.be/rise-api-core/haltes/vertrekken/200144/7>
- Zorg voor de nodige interfaces (<http://www.json2ts.com>) om de JSON data vanuit code te kunnen uitlezen.
- Gebruik de Angular **HttpClientModule** en bijhorende **HttpClient**
- Implementeer volgende de regels van de kunst (Angular Service geeft **Observable** terug aan de component, dewelke een **subscribe** hierop doet)
- Geef het **lijn nummer**, **bestemming** en **vertrek** weer.

Vertrekken		
Lijn	Bestemming	Vertrek
102	Melle Leeuw	2'
104	Muide	4'
102	Melle Leeuw	8:39
104	Muide	11'
102	Melle Leeuw	17'

Services (4)


- Werk de 'Weather' component verder af zodat deze met 'live' data werkt
- Service Url (weerinfo over Antwerpen):
<http://api.openweathermap.org/data/2.5/weather?q=antwerpen&lang=nl&APPID=c29dbdf3ccc2d57a361ceaeac49d9e53>
- Toon eveneens het weer icoontje, als volgt te bekomen:
[http://openweathermap.org/img/w/\[icon\].png](http://openweathermap.org/img/w/[icon].png)
- Waarbij **[icon]** dient te worden vervangen door de naam van het icoon
- Merk op dat de service
 - Default een temperatuur in K teruggeeft
 - De tijden van zonsopkomst/ondergang in unix UTC tijdformaat staan

ANTWERPEN	
Momenteel: lichte regen	
Temperatuur: 5.5	
	08:19
	16:40

Services (5)

- Breid de WeatherComponent uit zodat de gebruiker kan zoeken naar weerinfo van zijn/haar gemeente via een input veld

hasselt|

HASSELT

Momenteel: **lichte regen**

Temperatuur: **4.9**

 08:14

 16:38

dendermonde|

DENDERMONDE

Momenteel: **lichte regen**

Temperatuur: **5.4**

 08:19

 16:42

Services (6)

- Breid de 'De Lijn' component uit zodat de gebruiker kan zoeken op gemeente
- Geef in een dropdown de haltes weer
- Na selectie van een halte krijgt de gebruiker de vertrekken te zien

Verdere Uitbreiding:

- Combineer het Weather en De Lijn zoekveld, zodat de gebruiker met 1 zoekveld ineens zowel:
 - Weerinfo bekommt.
 - De Lijn info bekommt.
- Maak hierbij ook gebruik van promises als alternatieve oplossing en evalueer het verschil in broncode

