

Services & Dependency injection (DI)



Service in Angular

- Implementatie gebruik makend van eenvoudige classes

```
Export class LogService{  
  log(message: string){  
    console.log(message);  
  }  
}
```

- Definitie in de module
- @Injectable annotatie voorzien

Services in Angular

- Worden gebruikt door Angular components:
 - Ophalen van data
 - Validatie
 - Logging
 - ...

Services in Angular

- Services zijn eenvoudige classes met een @Injectable() decorator
- Injectable importeren van @angular/core

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

@Injectable()
export class PetService {
  pets: string[] = ['Cat', 'Dog', 'Rabbit', 'Fish']

  getPets() {
    return this.pets;
  }
}
```

Services in Angular

- Central beheer van gegevens in een app
- Meerdere components kunnen met dezelfde services werken
- Singleton (op globaal niveau)
 - Services worden maar één keer geïnitieerd tijdens het opstarten van de app
 - De levensduur van de service is net zo lang als de levensduur van de app

Dependency Injection (DI) – Code Pattern

- Code pattern
- Gebruik van classes zonder hard-coded constructor calls:
 - Flexibele code

```
//Voorbeeld 1: klassieke method
export class Car {
  engine: Engine;
  tire: Tire;

  constructor(){
    this.engine = new Engine();
    this.tire = new Tire();
  }
}
```

```
//Voorbeeld 2: DI
export class Car {
  engine: Engine;
  tire: Tire;

  constructor(engine, tire){
    this.engine = engine;
    this.tire = tire;
  }
}
```

Dependency Injection (DI) – Design Pattern

- Stel dat bij het eerste voorbeeld de Engine constructor aangepast wordt (bv: meegeven parameters), gaat de Car klasse stuk. Bij voorbeeld 2 is dit niet het geval, omdat de Engine buiten de klasse aangemaakt wordt.

```
//Voorbeeld 1: klassieke method
export class Car {
  engine: Engine;
  tire: Tire;

  constructor(){
    this.engine = new Engine();
    this.tire = new Tire();
  }
}
```

```
//Voorbeeld 2: DI
export class Car {
  engine: Engine;
  tire: Tire;

  constructor(engine, tire){
    this.engine = engine;
    this.tire = tire;
  }
}
```

Dependency Injection (DI) – Design Pattern

//Voorbeeld 1: klassieke method

```
export class Car {  
  engine: Engine;  
  tire: Tire;  
  
  constructor(){  
    this.engine = new Engine();  
    this.tire = new Tire();  
  }  
}  
  
// car is afhankelijk van de constructor  
// en opbouw van de classes Engine & Tire  
// er kunnen geen bestaande / oude objects  
// van car & engine gebruikt worden.  
  
car1: Car = new Car();
```

//Voorbeeld 2: DI

```
export class Car {  
  engine: Engine;  
  tire: Tire;  
  
  constructor(engine, tire){  
    this.engine = engine;  
    this.tire = tire;  
  }  
}  
  
// car is onafhankelijk van het aanmaken  
// van engine & tire  
engine1: Engine = new Engine();  
tire1: Tire = new Tire();  
car1: Car = new Car(engine1, tire1);
```


Dependency Injection (DI) - Angular

- Angular heeft ingebouwde dependency Injection
- Services worden via de constructor meegegeven aan components
- Injectors detecteren de services en voorzien de nodige dependencies
- Injectors zorgen voor de initialisatie van de service.

Dependency Injections (DI) - Angular

- Injectors bestaan op globaal niveau
 - Worden voorzien in app.module.ts

```
@NgModule({  
  imports: [...],  
  declarations: [...],  
  providers: [PetService], //Kan ook toegevoegd worden aan @Component  
})
```

- Injectors bestaan op component niveau
 - Worden voorzien in de x.component.ts

```
@Component ({  
  selector: 'app-component',  
  ...  
  providers: [PetService], //Kan ook toegevoegd worden aan @NgModule  
})
```

Dependency Injection (DI) - Angular

- Het initialiseren van de dependencies (Engine & Tire) wordt afgehandeld door het Angular framework. (Injectors)
- Services worden geïnjecteerd via DI in de component constructor

```
import { Component, OnInit } from '@angular/core';
import { PetService } from '../pet.service';
@Component({...})
export class AppComponent implements OnInit {
  pets: string[];
  constructor(private petsrv: PetService) { }
  ngOnInit() {
    this.pets = this.petsrv.getPets();
  }
}
```

Dependency Injection (DI) - Angular

- Indien een component dependency injection toepast en de service niet kan vinden, gaat hij kijken in de injectors van de parent
 - Indien deze de service ook niet kent -> parent
 - Tot aan de root injector (app.module.ts)
- Services globaal declareren: gedeeld over alle componenten
 - In app.module.ts
- Services lokaal declareren: één instantie voor die component
 - In de component

Dependency Injection (DI) - Samengevat

- Services zijn eenvoudige classes
 - met de `@Injectable()` decorator
- Services worden geregistreerd bij een Injector
 - Op `app.module.ts` niveau
 - Op component niveau

Dependency Injection (DI) - Samengevat

- Injectors kunnen enkel services injecteren die ze 'kennen'. Deze worden toegevoegd aan de providers array in de app.module.ts file (globaal) of in de component.

```
@NgModule({  
  imports: [...],  
  declarations: [...],  
  providers: [PetService], //Kan ook toegevoegd worden aan @Component  
  bootstrap: [...]  
})
```

- Injectors werken in een hiërarchie. Als een injector een service niet kent, gaat hij kijken bij de parent.
 - Tot dat hij aan de root injector komt (app.module.ts)

Dependency Injection (DI) - Angular

- Meer info:
 - <https://angular.io/guide/dependency-injection>

(Reusable) forms & services - Voorbeeld

Add event

Name:

Speaker:

Location:

Level:

Name	Speaker	Location	Level	Update
Angular fundamentals	Dries Swinnen	Hasselt	1	<input type="button" value="Update"/>
Angular Advanced	Luc Doumen	Zonhoven	2	<input type="button" value="Update"/>
Angular Expert	Steve Jobs	Neerpelt	3	<input type="button" value="Update"/>

https://github.com/PXL-WebExpert-2018/CH7_Voorbeeld1