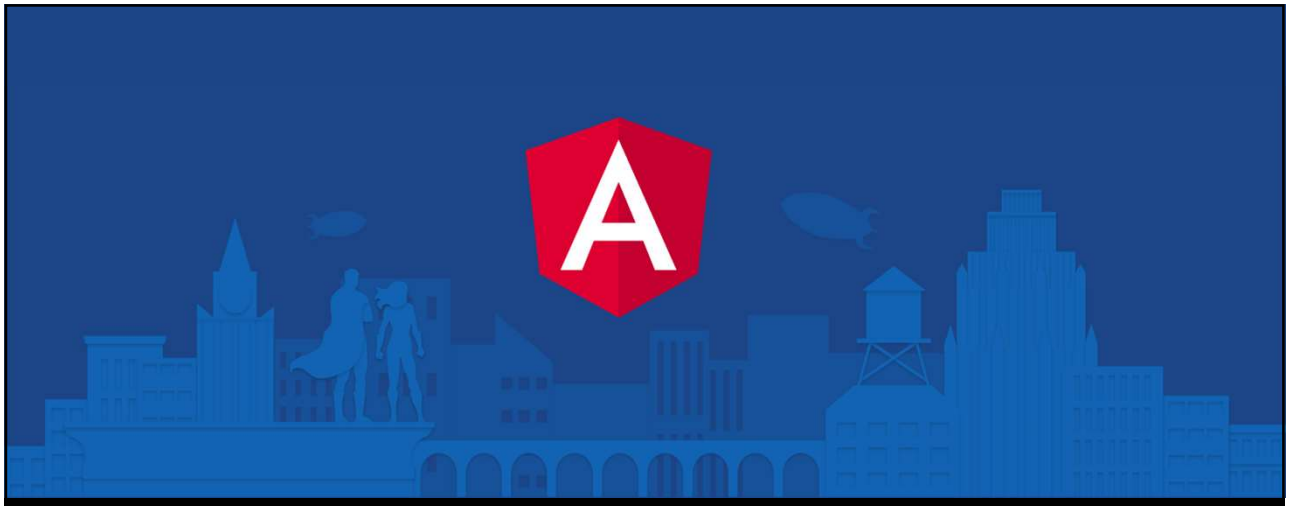


Services und Dependency Injection in Angular

Michael Zikes
SOFTWAREarchitekt.at

Was sind Services?

- Wiederverwendbare Codeeinheiten
- Austauschbar
- Testbar
- Meist Klassen
- Beispiel: FlugService

Service

Globaler Scope

```
@Injectable({ providedIn: 'root' })
export class FlugService {

    […]

}
```

Services sind Singletons

(in ihrem "Scope")

SOFTWARE architekt.at

Alternative

Service wird mit lazy Module mitgeladen

```
@Injectable({ providedIn: LazyApiModule })
export class FlugService {

    […]

}
```

Macht nur mit Lazy Loading sinn!!

Details später in Kapitel "Routing"

SOFTWARE architekt.at

Service

```
@Injectable({ providedIn: 'root' })
export class FlugService {

    […]

}
```

SOFTWARE architekt.at

Konsument bekommt Service injiziert

```
@Component({
    selector: 'flug-suchen',
    templateUrl: 'flug-suchen.html'
})
export class FlugSuchenComponent {

    von: string;
    nach: string;
    fluege: Array<Flug>;

    constructor(flugService: FlugService) { … }

    flugSuchen() { [...] }
    selectFlug(flug) { [...] }
}
```

SOFTWARE architekt.at

DEMO

Konsument bekommt Service injiziert

```
@Component({
    selector: 'flug-suchen',
    templateUrl: 'flug-suchen.html'
})
export class FlugSuchenComponent {

    von: string;
    nach: string;
    fluege: Array<Flug>;

    constructor(flugService: FlugService) { … }

    flugSuchen() { [...] }
    selectFlug(flug) { [...] }
}
```

Token

Token vs. Service

- Token: Das, was Konsument anfordert
  (z. B. FlugService)
- Service: Das, was Konsument wirklich bekommt
  (z. B. AdvancedFlugService)

Weiterleitung

```
@Injectable({
    providedIn: 'root',
    useClass: DefaultFlugService, ←– – Service
    deps: [HttpClient]
})
export class FlugService {

    […]

}
```

Token

Token

- Fast alles kann ein Token sein!
- Häufig: Standard-Implementierung des Services
- Abstrakte (Basis)-Klasse
- Konstante
- NICHT: Interface

Beispiel mit abstrakter Klasse

```
@Injectable({
    providedIn: 'root',
    useClass: DefaultFlugService
})
export abstract class FlugService {

    abstract find(from: string, to: string);

}
```

```
@Injectable()
export class DefaultFlugService implements FlugService {

    find(from: string, to: string) { … }

}
```

DEMO

Factory

```
@Injectable({
    providedIn: 'root',
    useFactory: (http: HttpClient) => {
        return new DefaultFlightService(http);
    },
    deps: [HttpClient]
})
export abstract class FlightService {

    abstract find(from: string, to: string): Observable<Flight[]>;

}
```

Factory

```
@Injectable({
    providedIn: 'root',
    useFactory: (http: HttpClient) => {
        return new DefaultFlightService(http);
    },
    deps: [HttpClient]
})
export abstract class FlightService {

    abstract find(from: string, to: string): Observable<Flight[]>;

}
```


Factory

```
@Injectable({
    providedIn: 'root',
    useFactory: (http: HttpClient) => {
        if (…) {
            return new DefaultFlightService(http);
        }
        else {
            return new DummyFlightService(http);
        }
    },
    deps: [HttpClient]
})
export abstract class FlightService {

    abstract find(from: string, to: string): Observable<Flight[]>;
}
```

DEMO

Alte Schreibweise (Angular <= 5)

```
export abstract class FlugService {

    abstract find(from: string, to: string);

}
```

```
@Injectable()
export class DefaultFlugService implements FlugService {

    find(from: string, to: string) { … }

}
```

Alte Schreibweise (Angular <= 5)

```
@NgModule({
    imports: [
        BrowserModule, HttpModule, FormsModule
    ],
    declarations: [
        AppComponent, FlugSuchenComponent
    ],
    providers: [
        { provide: FlugService, useClass: DefaultFlugService }
    ],
    bootstrap: [
        AppComponent
    ]
})
export class AppModule {
}
```

SOFTWARE architekt.at

Alte Schreibweise (Angular <= 5)

```
@NgModule({
    imports: [
        BrowserModule, HttpModule, FormsModule
    ],
    declarations: [
        AppComponent, FlugSuchenComponent
    ],
    providers: [
        { provide: FlugService, useClass: DefaultFlugService }
    ],
    bootstrap: [
        AppComponent
    ]
})
export class AppModule {
}
```

SOFTWARE architekt.at

Alte Schreibweise (Angular <= 5)

```
@NgModule({
    imports: [
        BrowserModule, HttpModule, FormsModule
    ],
    declarations: [
        AppComponent, FlugSuchenComponent
    ],
    providers: [
        { provide: FlugService, useClass: DefaultFlugService }
    ],
    bootstrap: [
        AppComponent
    ]
})
export class AppModule {
}
```

Token

Service

SOFTWARE architekt.at

Alte Schreibweise (Angular <= 5)

```
@NgModule({
    imports: [
        BrowserModule, HttpModule, FormsModule
    ],
    declarations: [
        AppComponent, FlugSuchenComponent
    ],
    providers: [
        { provide: FlugService, useClass: DefaultFlugService }
    ],
    bootstrap: [
        AppComponent
    ]
})
export class AppModule {
}
```

SOFTWARE architekt.at

Alte Schreibweise (Angular <= 5)

```
@NgModule({
    imports: [
        BrowserModule, HttpModule, FormsModule
    ],
    declarations: [
        AppComponent, FlugSuchenComponent
    ],
    providers: [
        { provide: FlugService, useClass: FlugService }
    ],
    bootstrap: [
        AppComponent
    ]
})
export class AppModule {
}
```


Alte Schreibweise (Angular <= 5)

```
@NgModule({
    imports: [
        BrowserModule, HttpModule, FormsModule
    ],
    declarations: [
        AppComponent, FlugSuchenComponent
    ],
    providers: [
        FlugService
    ],
    bootstrap: [
        AppComponent
    ]
})
export class AppModule {
}
```

DEMO

Service lokal registrieren

```
@Component({
    selector: 'flug-suchen',
    templateUrl: 'app/flug-buchen/flug-buchen.html',
    providers: [{ provide: FlugService, useClass: FlugService}]
})
export class FlugSuchenComponent {

    […]

}
```

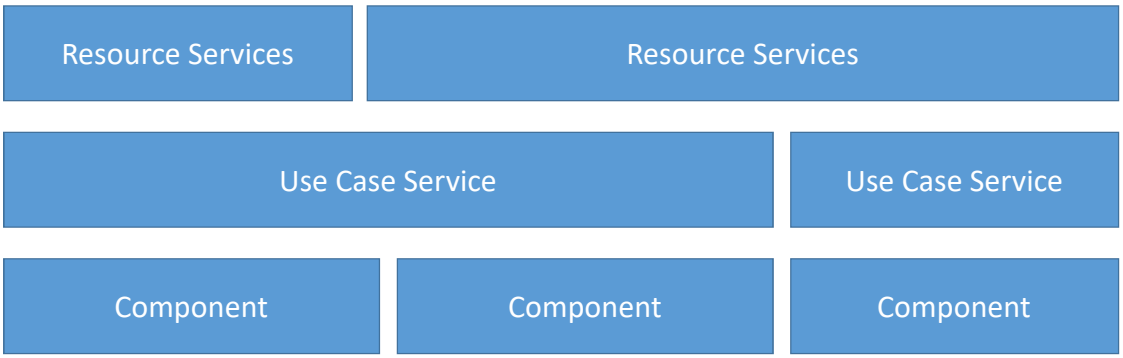
Gilt nur für aktuelle Komponente und darunter!

Service lokal registrieren

```
@Component({
    selector: 'flug-suchen',
    templateUrl: 'app/flug-buchen/flug-buchen.html',
    providers: [FlugService]
})
export class FlugSuchenComponent {

    […]

}
```


DEMO

Vorschlag: Services und Architektur



SOFTWAREarchitekt.at

Konstanten als Token

Warum Konstanten?

- Eventuell gibt es für ein Konzept keinen geeigneten Typ
  - Strings mit Konfigurationsdaten
  - Token, die auf Funktionen verweisen
- Angular nutzt diese Möglichkeit intern

InjectionToken

```
export const BASE_URL =
    new InjectionToken<string>('BASE_URL', […] );
```

InjectionToken

```
export const BASE_URL =
    new InjectionToken<string>('BASE_URL', {
         providedIn: 'root',
         factory: () => 'http://www.angular.at/api' } );
```


InjectionToken

```
export const FLIGHT_SERVICE =
    new InjectionToken<FlightService>('FLIGHT_SERVICE', {
         providedIn: 'root',
         factory: () => new FlightService(inject(HttpClient)) } );
```

InjectionToken

```
export const FLIGHT_SERVICE =
    new InjectionToken<FlightService>('FLIGHT_SERVICE', {
         providedIn: 'root',
         factory: () => new FlightService(inject(HttpClient)) } );
```


Alte Schreibweise (<= Version 5)

```
import { InjectionToken } from "@angular/core";  // Angular 2: OpaqueToken

export const BASE_URL = new InjectionToken<string>("BASE_URL");
```

Alte Schreibweise (<= Version 5)

```
@NgModule({
  […],
  providers: [
    […]
    { provide: BASE_URL, useValue: 'http://…'}
  ]
})
export class AppModule {
}
```


Abhängigkeit injizieren

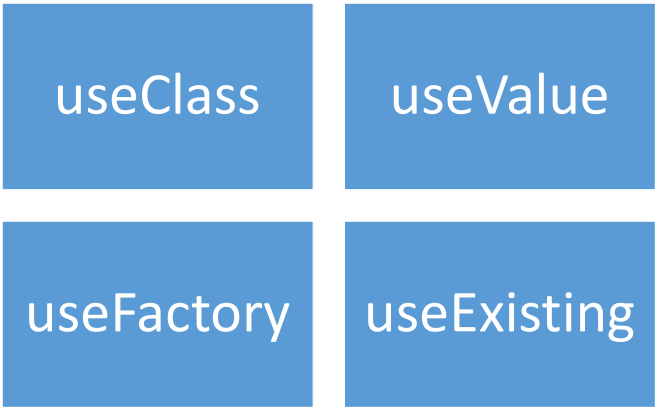
```
@Injectable()
export class FlightService {

  flights: Array<Flight> = [];

  constructor(
    @Inject(BASE_URL) private baseUrl: string,
    private http: Http
  ) {
      […]
  }

  […]
}
```

DEMO

SOFTWARE architekt.at

Arten von Bindungen

Arten von Bindungen



SOFTWAREarchitekt.at