ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Services & Dependency Injection

Alex Thalhammer

What are Services?

Reusable

Replaceable

Testable

Classes

Example: FlightService

What type of Services?

Infrastructure (API/Backend)

Application (Functions)

Communication bw. Comp.

State Management

Service

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

    […]

}
```

Service

global scope

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

    […]

}
```

services are singletons

(in their “scope")

Consumer gets injected service in constructor

```
@Component({
    selector: 'app-flight-search',
    templateUrl: 'flight-search.component.html'
})
export class FlightSearchComponent {
    from = '';
    to = '';
    flights: Flight[] = [];

    constructor(private flightService: FlightService) { … }

    search(): void { [...] }
    select(flight): void { [...] }
}
```

Token

Token vs. Service

- Token: What the consumer requests
  (e. g. flightService)

- Service: What the consumer receives
  (e. g. advancedFlightService)

Token

- Almost everything can be a token
- In most cases: default implementation of service
- Abstract (Base)-Class
- Constant
- But no interface

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 (environment.production) {
            return new DefaultFlightService(http);
        } else {
            return new DummyFlightService(http);
        }
    },
    deps: [HttpClient]
})
export abstract class FlightService {
    abstract find(from: string, to: string): Observable<Flight[]>;
}
```

DEMO

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

LAB

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT