



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Angular Services & Dependency Injection

Alex Thalhammer

What are Services?

Reusable

Replaceable

Testable

Classes

Example:
FlightService



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: 'flight-search',
  templateUrl: 'flight-search.component.html'
})
export class FlightSearchComponent {

  from: string;
  to: string;
  flights: Flight[];

  constructor(private flightService: FlightService) { ... }

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



Consumer gets injected service in constructor

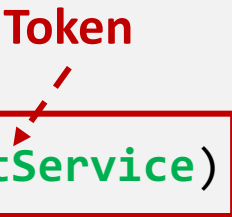
```
@Component({
  selector: 'flight-search',
  templateUrl: 'flight-search.component.html'
})
export class FlightSearchComponent {

  from: string;
  to: string;
  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)

Forwarding

```
@Injectable({  
  providedIn: 'root',  
  useClass: DefaultFlightService, ← -- Service  
  deps: [HttpClient]  
})  
export class FlightService {  
  [...]  
    
  Token  
}
```



Token

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



Example with abstract class

```
@Injectable({  
  providedIn: 'root',  
  useClass: DefaultFlightService  
})  
export abstract class FlightService {  
  
  abstract find(from: string, to: string);  
  
}
```

```
@Injectable()  
export class DefaultFlightService implements FlightService {  
  
  find(from: string, to: string) { ... }  
  
}
```



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[]>;
}
```



Old way of providing services (Angular <= 5)

```
export abstract class FlightService {  
    abstract find(from: string, to: string);  
}
```

```
@Injectable()  
export class DefaultFlightService implements FlightService {  
    find(from: string, to: string) { ... }  
}
```



Old way of providing services (Angular <= 5)

```
@NgModule({  
  imports: [  
    BrowserModule, HttpClientModule, FormsModule  
  ],  
  declarations: [  
    AppComponent, FlightSearchComponent  
  ],  
  providers: [  
    { provide: FlightService, useClass: DefaultFlightService }  
  ],  
  bootstrap: [  
    AppComponent  
  ]  
})  
export class AppModule {}
```



Old way of providing services (Angular <= 5)

```
@NgModule({  
  imports: [  
    BrowserModule, HttpClientModule, FormsModule  
  ],  
  declarations: [  
    AppComponent, FlightSearchComponent  
  ],  
  providers: [  
    FlightService  
  ],  
  bootstrap: [  
    AppComponent  
  ]  
})  
export class AppModule {}
```



Old way of providing services (Angular <= 5)

```
@NgModule({  
  imports: [  
    BrowserModule, HttpClientModule, FormsModule  
  ],  
  declarations: [  
    AppComponent, FlightSearchComponent  
  ],  
  providers: [  
    { provide: FlightService, useClass: FlightService }  
  ],  
  bootstrap: [  
    AppComponent  
  ]  
})  
export class AppModule {}
```



Old way of providing services (Angular <= 5)

```
@NgModule({  
  imports: [  
    BrowserModule, HttpClientModule, FormsModule  
  ],  
  declarations: [  
    AppComponent, FlightSearchComponent  
  ],  
  providers: [  
    { provide: FlightService, useClass: DefaultFlightService }  
  ],  
  bootstrap: [  
    AppComponent  
  ]  
})  
export class AppModule {}
```



Old way of providing services (Angular <= 5)

```
@NgModule({
  imports: [
    BrowserModule, HttpClientModule, FormsModule
  ],
  declarations: [
    AppComponent, FlightSearchComponent
  ],
  providers: [
    { provide: FlightService, useClass: DefaultFlightService }
  ],
  bootstrap: [
    AppComponent
  ]
})
export class AppModule {}
```



Old way of providing services (Angular <= 5)

```
@NgModule({
  imports: [
    BrowserModule, HttpClientModule, FormsModule
  ],
  declarations: [
    AppComponent, FlightSearchComponent
  ],
  providers: [
    { provide: FlightService, useClass: DefaultFlightService }
  ],
  bootstrap: [
    AppComponent
  ]
})
export class AppModule {}
```

Token (points to `FlightService`)

Service (points to `DefaultFlightService`)



Register service locally

```
@Component({  
  selector: 'flight-search',  
  templateUrl: flight-search.html',  
  providers: [FlightService]  
})  
export class FlightSearchComponent {  
  
  [...]  
  
}
```

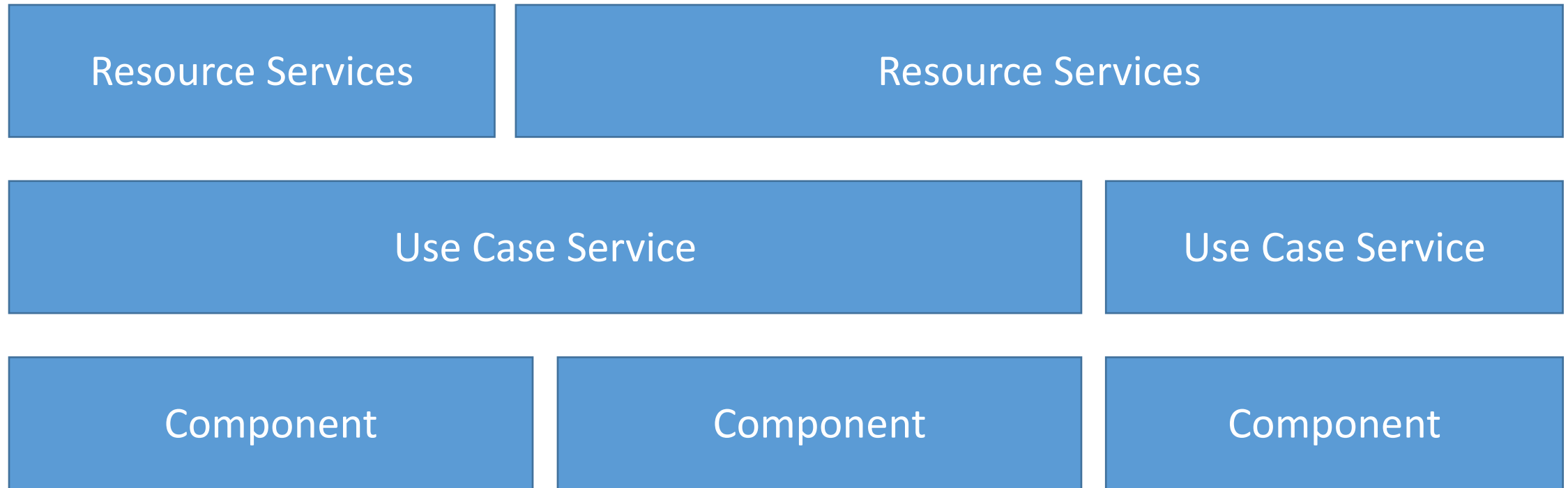


Register service locally w. useClass

```
@Component({  
  selector: 'flight-search',  
  templateUrl: flight-search.html',  
  providers: [{ provide: FlightService, useClass: AdvancedFlightService}]  
})  
export class FlightSearchComponent {  
  
  [...]  
  
}
```

Only used for current component and children

Possible architecture with Services



DEMO



LAB





ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Provider definitions

angular-architects.io

Provider definition keys

useClass

useValue

useFactory

useExisting



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT