# Service

- Serwisy to klasa TypeScript, w której możemy wykorzystać dekorator **@Injectable()**. Serwisy są używane m.in. do obsługi pobierania danych z API / serwera, przechowywania stanu aplikacji, czy obsługi logiki biznesowej, żeby odciążyć komponenty. Dzięki nim można w łatwy sposób komunikować się pomiędzy różnymi komponentami / modułami Angular.
- Dependency injection (wstrzykiwanie zależności) to wzorzec projektowy, który jest zaimplementowany w Angular i to dzięki niemu serwisy są dostępne w komponentach. Angular umożliwia wstrzyknięcie serwisu do komponentu, dzięki czemu komponent może używać metod z serwisu. W Angular dependency injection (w skrócie **DI**) składa się z dwóch składowych: **injector** oraz **provider**.
- Dekorator **@Injectable()** nie jest obowiązkowy – ale dobrą praktyką jest stosowanie go.

```
1  import { Injectable } from '@angular/core';
2
   4 usages
3  @Injectable({
4    providedIn: 'root'
5  })
6  export class TestService {
     no usages
7    constructor() {}
8  }
```

```
9  export class AppComponent {
     no usages
10   constructor(private testService: TestService) {}
11 }
```

# Nowy sposób na wstrzykiwanie zależności (Angular 14)

- Od wersji Angular 14 możemy używać **inject()** w komponentach, dyrektywach i pipe'ach.

```
import {Component, inject, OnInit} from '@angular/core';
```

```
export class AppComponent implements OnInit{
  1 usage
  testService = inject(TestService);
  no usages
  constructor() {}
  no usages
  ngOnInit(): void {
    this.testService.useMe();
  }
```

Pozwala to na nieco większą swobodę – nie musimy koniecznie wstrzykiwać zależności do konstruktora w danym komponencie, za to można m.in. tworzyć re-używalny kod.

```
function fetchGet() {
  return inject(HttpClient).get( url: 'adres');
}
```

```
export class AppComponent {
  no usages
  useMe = fetchGet();
  no usages
  constructor() {}
}
```

# Injector i provider

- **Injector –**Angular tworzy obiekt Injector'a przy uruchamianiu aplikacji (oraz dodatkowe Injector'y, jeśli są wymagane) – z tego powodu nie ma potrzeby własnoręcznego tworzenia Injector'a. Podczas tworzenia komponentu / serwisu / dyrektywy, która potrzebuje zależności (dependency), to Angular najpierw sprawdza, czy injector ma informację na temat tej zależności. Jeśli taka nie istnieje to używa provider, żeby ją stworzyć.
- **Provider** – to obiekt, który informuje Injector, jak ma pobrać lub stworzyć zależność (dependency).
- Dla każdej zależności w naszej aplikacji, musimy zarejestrować provider (możemy dodać wpis do tablicy providers w dekoratorze @NgModule lub w @Component lub użyć metadane dostępne w dekoratorze @Injectable(), czyli providedIn: "root").
- Domyślnie przy korzystaniu z Angular CLI i komendy **ng generate service …** to Angular rejestruje provider za pomocą **providedIn: "root"**.

# Service – rejestracja provider’a

W Angular są trzy sposoby na rejestrację provider’a dla serwisów.

- Zastosowanie metadanej **providedIn: ”root”** w dekoratorze @Injectable. To najbardziej zalecana opcja. Angular tworzy wtedy tylko jedną instancję danego serwisu dla całej aplikacji. Dodatkowo jeśli serwis nie jest używany to serwis jest usuwany z skompilowanej wersji aplikacji (tzw. **tree-shaking**, czyli usuwanie nadmiarowego kodu). Do tego ten sposób powoduje, że serwisy mogą być ładowane **lazily**, czyli wtedy kiedy są wymagane a nie od razu przy starcie aplikacji.
- Dodanie klasy serwisu do tablicy **providers** w dekoratorze @NgModule. Jeśli dodamy ten wpis tylko do AppModule, to Angular utworzy tylko jedną instancje danego serwisu dla całej aplikacji. W przypadku dodania serwisu do **providers** w innych modułach niż AppModule – serwis będzie posiadał po jednej instancji dla każdego modułu oddzielnie.
- Dodanie klasy serwisu do tablicy **providers** w dekoratorze @Component. Każde wykorzystanie tego komponentu powoduje utworzenie nowej instancji danego serwisu, która jest dostępna dla tego jednego wybranego komponentu i jego wszystkich child component. Na przykład, jeśli użyjemy **providers** w AppComponent to dana instancja serwisu będzie taka sama dla wszystkich jego child component, ale w innych modułach już będzie inna instancja serwisu.

# Service jako singleton

W Angular są dwa sposoby, aby upewnić się, że serwis będzie singletonem (będzie dostępna tylko 1 jego instancja na całą aplikację).

- Wykorzystanie metadanej **providedIn: ”root”** (zalecana opcja).

```
@Injectable({
  providedIn: 'root'
})
export class TestService {
  no usages
  constructor() {}
}
```

- Dodanie klasy serwisu **TYLKO** w tablicy providers w AppModule.

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

# Wstrzyknięcie service do service

Aby wstrzyknąć serwis do drugiego serwisu (kiedy zdecydujemy się nie używać **providedIn: ”root**”), to ten serwis, który w swoim konstruktorze wstrzykuje serwis musi mieć wykorzystany dekorator @Injectable(), w przeciwnym wypadku będzie wyrzucany następujący błąd:

```
src/app/app.module.ts:37:5 - error NG2005: The class 'Test2Service' cannot be created via dependency
injection, as it does not have an Angular decorator. This will result in an error at runtime.

Either add the @Injectable() decorator to 'Test2Service', or configure a different provider (such as a
provider with 'useFactory').
```

W takim przypadku zadziała:

```
export class TestService {

  no usages
  constructor() { }
  1 usage
  useMe() {
    console.log('test')
  }
}
```

```
@Injectable()
export class Test2Service {

  no usages
  constructor(private testService: TestService) {
    console.log(testService.useMe())
  }
}
```

```
providers: [
  TestService,
  Test2Service,
```

# Dekorator @Inject

- W Angular oprócz klas serwisów możemy wstrzykiwać także funkcje, obiekty, typy prymitywne, jak string / boolean itp. Wtedy może być konieczne użycie dekoratora **@Inject()**.

myFunction.ts

```ts
4 usages
export function myFunction() {
  console.log('test');
}
```

```ts
@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule
  ],
  providers: [
    { provide: 'myFunction', useFactory: myFunction },
```

```ts
export class AppComponent implements OnInit{
  no usages
  constructor(@Inject('myFunction') private myFunction: Function) {}
  no usages
  ngOnInit(): void {
    this.myFunction;
  }
```

```
test                                    myFunction.ts:2
```

Wynik w przeglądarce

- Oprócz kodu zaprezentowanego na poprzednim slajdzie, funkcja także może mieć zależności:

```
@Injectable({
  providedIn: 'root'
})
export class TestService {
  no usages
  constructor() { }
  1 usage
  useMe() {
    console.log('test')
  }
}
```

```
import {TestService} from "./services/test.service";

4 usages
export function myFunction(
  testService: TestService
) {
  testService.useMe();
}
```

```
@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule
  ],
  providers: [
    { provide: 'myFunction', useFactory: myFunction, deps: [TestService] },
```

```
export class AppComponent implements OnInit{
  no usages
  constructor(@Inject('myFunction') private myFunction: Function) {}
  no usages
  ngOnInit(): void {
    this.myFunction;
  }
```