DI and Providers

Jan Strapek
Front-end developer


# flow

# DI and Providers

Jan Strapek
Front-end dev @FlowUp

S-Johny

# Without Angular DI

```
function  doStuff() {
  const httpHandler = new MyHttpHandler();
  const httpClient = new HttpClient(httpHandler);
  const myService = new MyService(httpClient);
  /*...*/
}
```

# Dependency Injection

- Help to create more efficient and modular app
- Provides declared dependencies to a class

# Dependencies

➔ Services or object that class need to perform its functions

# Providers

➔ Instruction to DI system on how to obtain a value for dependency

# Injector

- ➔ Uses providers to create new instance of dependency
- ➔ Created for NgModule automatically
- ➔ Provides singleton instances

# DI Token

➔ It’s a key under which provider finds dependency

up

# Old way

```
@NgModule({
  /*...*/
  providers: [MyService]
})
export class MyModule {}
```

# New way

```
@Injectable({
  providedIn: 'root'
})
export class MyService {
  /*...*/
}
```

flow

# Circular Dependencies!

```
@Component({
  templateUrl: './',
  /*....*/
})
export class MyComponent {
  constructor(private readonly MyService) {}
}
// ...
@NgModule({
  declarations: [MyComponent],
})
export class MyModule {}
// ...
@Injectable({
  providedIn: MyModule
})
export class MyService {
  /*...*/
}
```

# Which to use and why?

- ➔ Use ***provideIn: 'root'*** for services which should be available in whole app
- ➔ Use ***provideIn: MyModule*** to prevent service injection in the eagerly imported part of app.
- ➔ Use ***providers: []*** inside of **@Component** or **@Directive** or **@NgModule** to scope service only for the particular component sub-tree

up

```
@NgModule({
 providers: [MyService],
})
export class MyModule {}
```

```
@NgModule({
 providers: [
   {
     provide: MyService,
     useClass: MyService
   }
 ],
})
export class MyModule {}
```

flow

up

```
export const BASE_URL = new InjectionToken<string>( _desc: 'BaseUrl');
```

```
@NgModule({
  declarations: [AppComponent],
  imports: [CommonModule, BrowserModule],
  providers: [{provide: BASE_URL, useValue: 'https://flowup.cz'}],
  bootstrap: [AppComponent]
})
export class AppModule {
  constructor(@Optional() @Inject(BASE_URL) url) {
    console.log('url', url);
  }
}
```

flow

# Resolution modifiers

- **@Optional()**
  - Allows Angular to consider a service you inject to be optional.
  - This way, if it can't be resolved at runtime, Angular simply resolves the service as **null**.
- **@Self()**
  - Angular will only look at the **Element Injector** for the current component or directive.
- **@SkipSelf()**
  - Angular will start its search for service in **parent Element Injector,** rather than current one.
- **@Host()**
  - Lets you designate a component as the **last stop** in the injector tree when searching for providers.

up

# Q&A


Jan Strapek

Front-end dev @FlowUp

S-Johny


flow