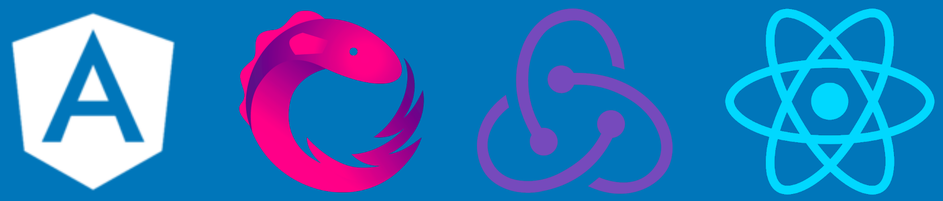
DI tricks

w/ decorators

Michele Stieven

Sviluppatore web e consulente lato front-end



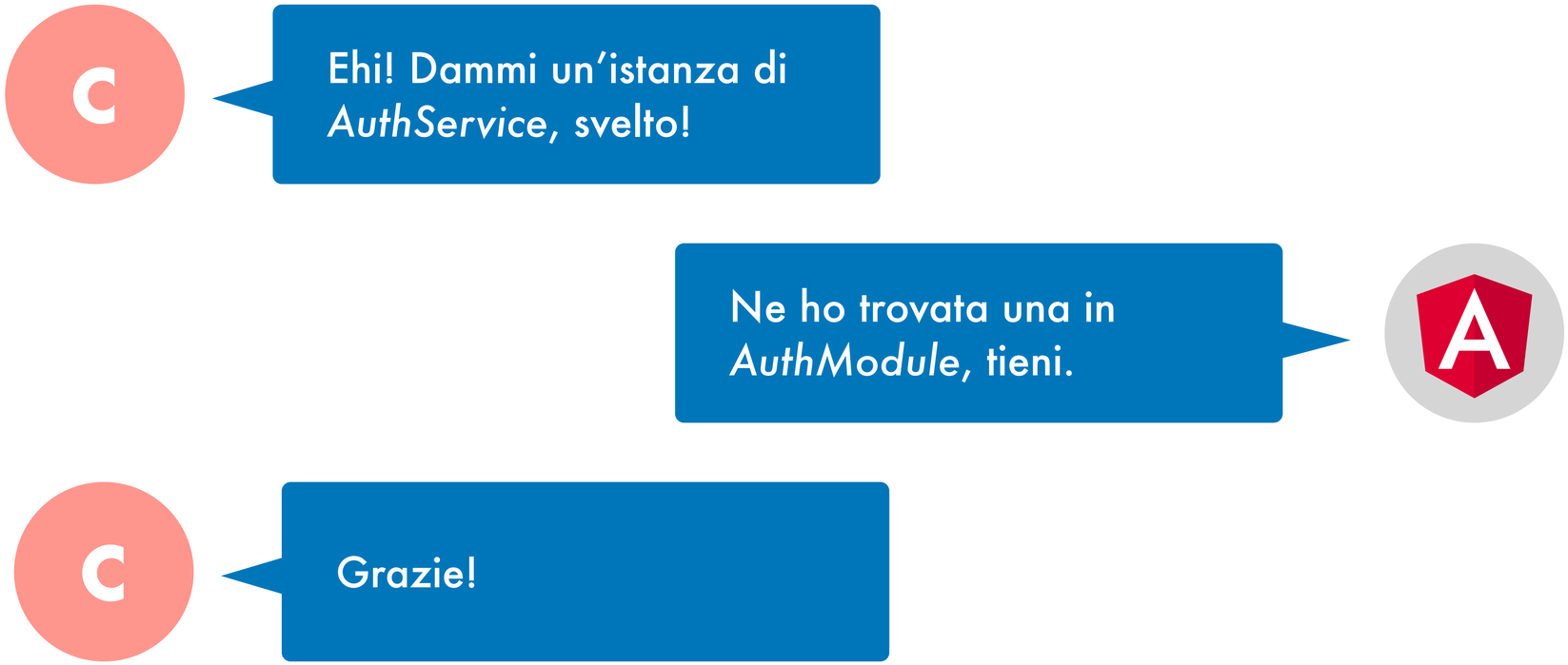
twitter.com/michelestieven

facebook.com/michelestieven

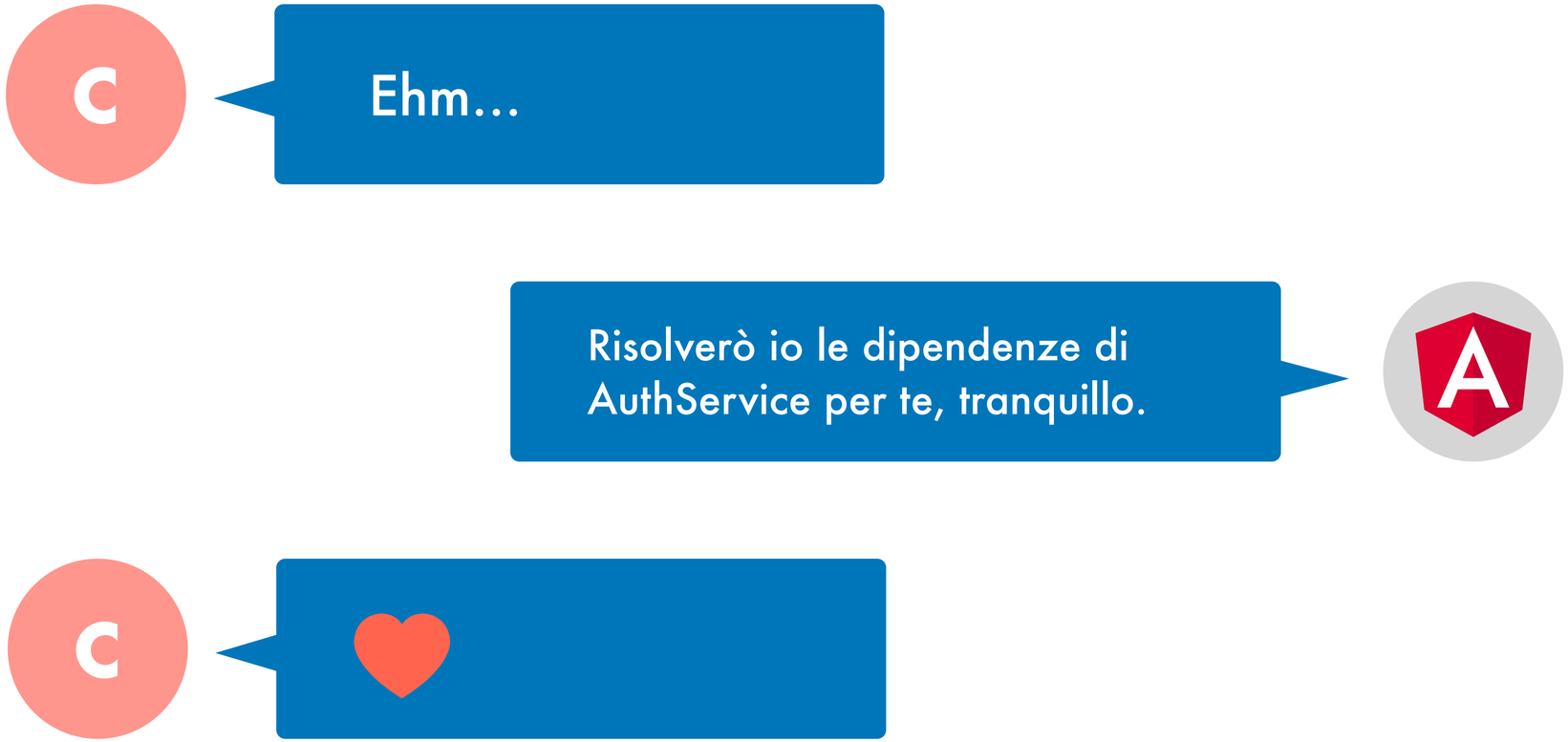
linkedin.com/in/michelestieven

medium.com/@michelestieven

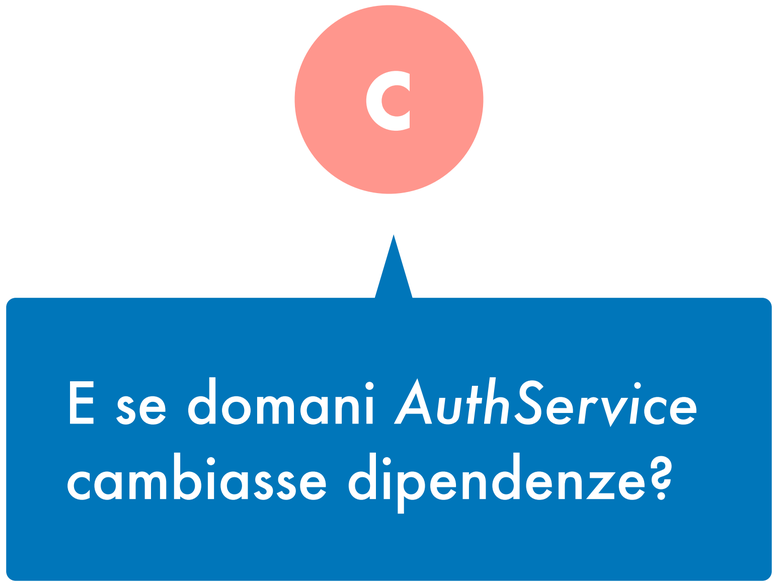
constructor(auth: AuthService)



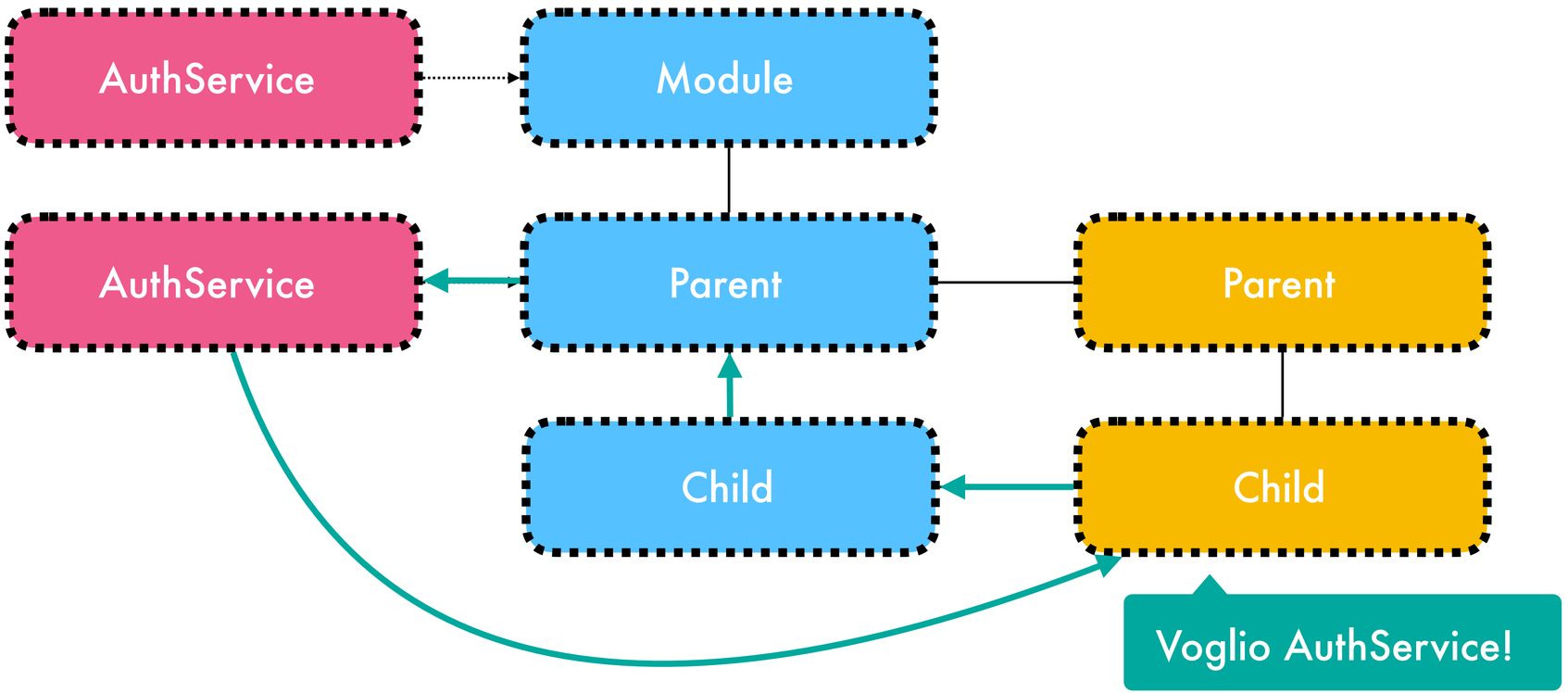
Il componente è ignorante



…Ma ha i suoi buoni motivi



Providers Injectors Components



Il *DI Framework* di Angular decide
quale istanza passare al componente.

Rendiamo il componente più intelligente!

@Decoratori

I decoratori ci permettono di scrivere codice **espressivo**,
manipolando e aggiungendo metadati alle nostre strutture.

Class decorators

```
@Directive({ ...metadata })
export class MyDirective { ... }
```

Property decorators

```
export class MyDirective {
 @Input() myInput;
}
```

Parameter Decorators

- Optional
- Self
- SkipSelf
- Host
- Inject

"They alter the way the DI framework provides a dependency, by annotating the dependency parameter on the constructor of the class that requires the dependency."

@Self

constructor(@Self() auth: AuthService) {}

Utile per:

- Assicurarsi di iniettare il servizio specificato nello stesso componente
- In una direttiva, iniettare un servizio specificato nell'elemento sul quale è usata

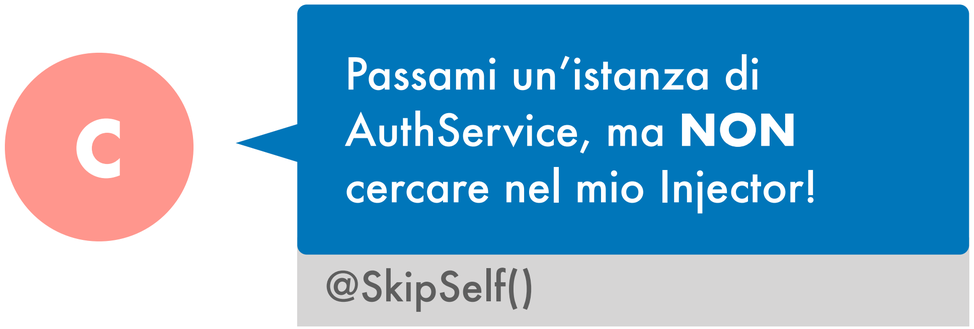
@Self

```
<my-component A>
```

```
@Directive({
  selector: '[A]'
})
export class ADirective {
  constructor(@Self() s: MyService) {}
}

@Component({
  selector: 'my-component',
   ... ,
  providers: [ MyService ]
})
export class MyComponent { }
```

@SkipSelf

constructor(@SkipSelf() auth: AuthService) {}

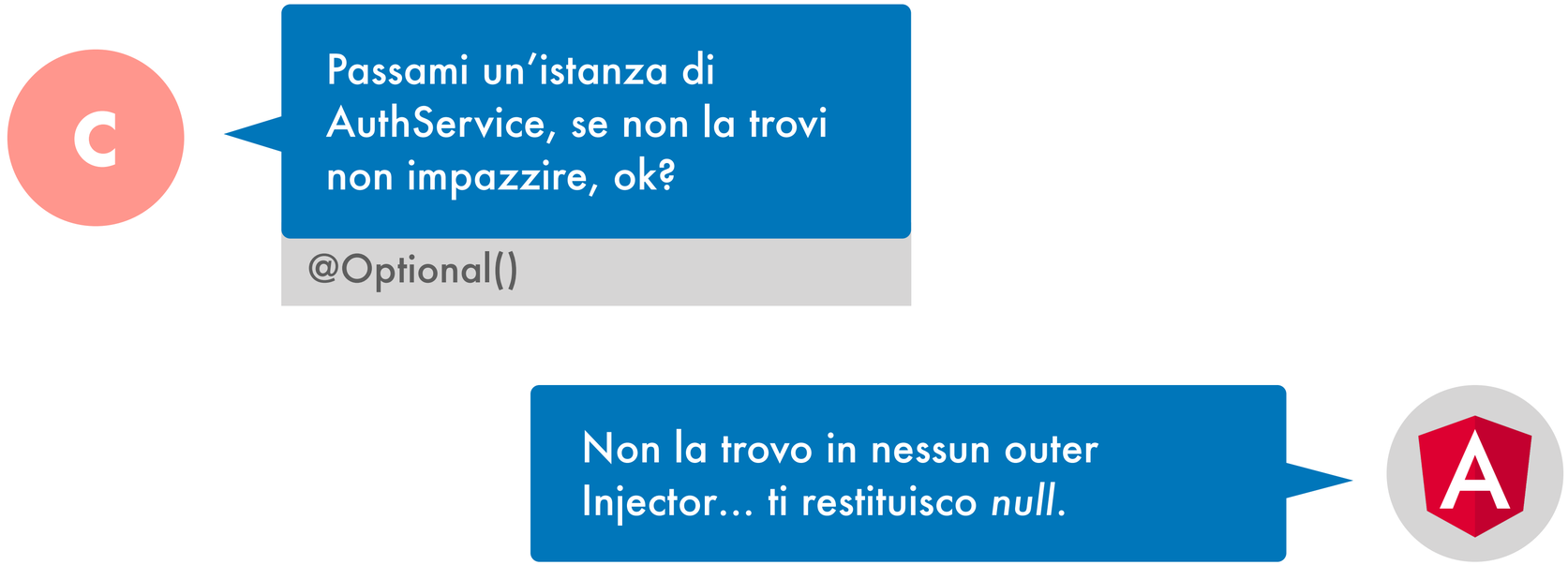


Utile per:

- Elementi o moduli che ri-dichiarano dei provider esistenti
- Auto-iniettare un modulo (all'interno di se stesso)

@Optional

```
constructor(@Optional() auth: AuthService) {}
```



Utile per:

- Verificare l'esistenza di moduli/componenti
- Rendere un componente riutilizzabile (es. Content Projection)

@Optional @SkipSelf

Verificare che un modulo sia Singleton verificando nell'outer Injector

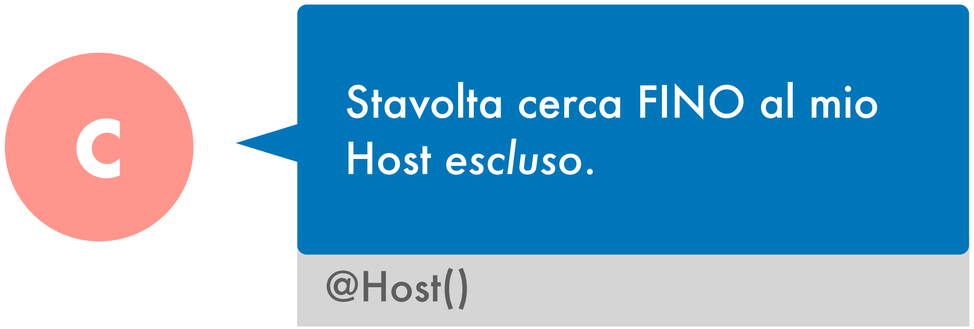
```
@NgModule({ ... })
export class CoreModule {

  constructor(@Optional() @SkipSelf() me: CoreModule) {
    if (me) throw Error('Non importarmi più di una volta!');
  }
}
```

Senza *SkipSelf* il modulo proverebbe ad importarsi dal suo stesso injector, impossibile quando il modulo viene creato la prima volta.

Usare un modulo Core è considerato una best practice dal team di Angular, utile per ripulire AppModule!

@Host

constructor(@Host() auth: AuthService) {}



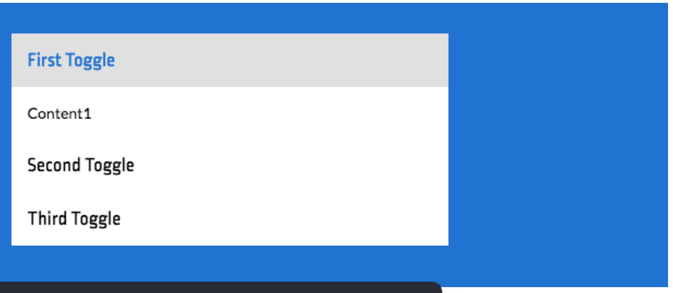
Utile per:

- Componenti o direttive che sappiamo essere dichiarati come figli di altri componenti/direttive **nello stesso template**

@Host

Due componenti/direttive padre-figlio nello stesso template

```
<accordion>
  <toggle title="First Toggle">Content1</toggle>
  <toggle title="Second Toggle">Content2</toggle>
  <toggle title="Third Toggle">Content3</toggle>
</accordion>
```




```
@Component({
  selector: 'toggle',
  ...
})
export class ToggleComponent {

  constructor(@Optional() @Host() accordion: AccordionComponent) {
    /**
    * Do something with the accordion
    */
  }
}
```

Questo decoratore rende il componente utilizzabile anche da solo!

NgModel

Dietro le quinte...

Questa direttiva viene attaccata a tutti i form da Angular, automaticamente!

```
<form #f="NgForm">
  <input ngModel  ... />
</form>
```

```
 */
@Directive({
  selector: '[ngModel]',
  ...
})
export class NgModel {
  constructor(
    @Optional() @Host() parent: ControlContainer,
    @Optional() @Self() @Inject(NG_VALIDATORS) validators,
    @Optional() @Self() @Inject(NG_ASYNC_VALIDATORS) asyncValidators,
    @Optional() @Self() @Inject(NG_VALUE_ACCESSOR) valueAccessors
  ) {
    ...
  }
}
```

Questo è il form (NgForm)

Queste sono le direttive usate con l'elemento input

```
@Directive({
  selector: '[A]',
})
export class ADirective {

  constructor(@Host() s: MyService) {}
}
```

Errore!

```
@Component({
  selector: 'outer',
  template: `
    <div>
        <p A> ... </p>
    </div>
  `,
  providers: [ MyService ]
})
export class OuterComponent {}
```

MyService è nell'injector dell'host, Host() non arriva qui!

```
@Directive({
  selector: '[A]',
})
export class ADirective {

  constructor(@Host() s: MyService) {}
}
```

Ok!

```
@Component({
  selector: 'outer',
  template: `
    <div>
        <p A> ... </p>
    </div>
  `,
  viewProviders: [ MyService ]
})
export class OuterComponent {}
```

viewProviders

Possiamo definire dei **viewProvider** per rendere accessibili i nostri servizi a livello di *View* (quindi accessibili anche da @*Host*)

viewProviders

I viewProvider rendono **inaccessibili** i servizi a tutti i figli proiettati con *ng-content*!

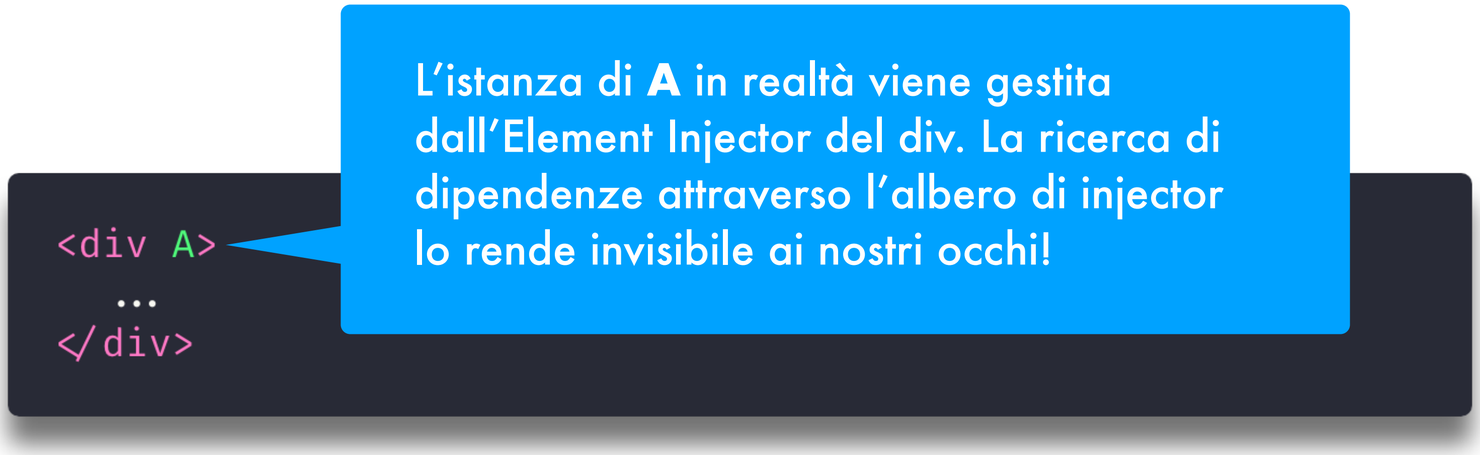
Utile ad esempio nello sviluppo di una libreria di componenti per evitare che gli sviluppatori vadano a toccare servizi ad uso interno.

```
@Component({
  selector: 'outer',
  template: `
    <div>
        <!-- Projected Content can NOT reach MyService -->
        <ng-content></ng-content>
    </div>
  `,
  viewProviders: [ MyService ]
})
export class OuterComponent {}
```

Element Injectors

In un template, ogni elemento ha il suo “mini” Injector, chiamato Element Injector!

Le istanze di direttive applicate ad elementi/componenti vengono gestite dal loro Element Injector.



@Inject

constructor(@Inject(AuthService) auth) {}

Utile per:

- Quando la dipendenza da iniettare non è rappresentata da una classe (es. una stringa o un InjectionToken)

InjectionToken

Quando una dipendenza non è rappresentabile da una classe

```
/**
* Injection Token
*/
const APP_CONFIG = new InjectionToken<AppConfig>('This is the configuration!');
const appConfig: AppConfig = { apiUrl: 'api.domain.com',  ...  };

/**
* Declaring the provider
*/
@NgModule({
   ...
  providers: [{ provide: APP_CONFIG, useValue: appConfig }]
})
export class ConfigModule {}

/**
* Declaring the dependency
*/
@Component({ ... })
export class MyComponent {

  constructor(@Inject(APP_CONFIG) config: AppConfig) {}
}
```

Utilissimo per rendere dei moduli configurabili dall'esterno (vedi *RouterModule*, *StoreModule* di ngrx e tanti altri. La configurazione viene dichiarata nei provider con un InjectionToken!


GRAZIE!