Custom Controls

In Angular Forms

Michele Stieven

Michele Stieven

Front-end Developer & Consultant

Obiettivi del talk

- Comprendere i vantaggi degli Angular Forms
- Creazione di un controllo custom
- Utilizzo dei controlli custom con Reactive e Template forms
- Capire come scrivere form annidati e riutilizzabili
- Vantaggi e svantaggi dell’approccio

Angular Forms

Reactive Form

- “Vivono” nella classe
- Facili da testare
- Facili da manipolare
- Facili da monitorare (RxJS!)
- Facili da validare
- Ideali per casi complessi
  - Campi dinamici
  - Validatori dinamici
  - Interazioni con servizi
  - ...

Template-driven Form

- Facili da usare (in casi semplici)
- “Vivono” nel template
- Difficili da manipolare
- Difficili da testare
- Ideali per casi semplici

Example

Template-driven Forms

Creazione del Form
delegata ad Angular

```
<form #userForm="NgForm" (ngSubmit)="onSubmit(userForm)">
  <input type="text" [(ngModel)]="user.name" required>
  <input type="text" [(ngModel)]="user.surname" required>
</form>
```

Example

Template-driven Forms (one way binding)

Creazione del Form
delegata ad Angular

```
<form #userForm="NgForm" (ngSubmit)="onSubmit(userForm)">
  <input type="text" [ngModel]="user.name" required>
  <input type="text" [ngModel]="user.surname" required>
</form>
```

Example

Template-driven Forms (no binding)

Creazione del Form
delegata ad Angular

```
<form #userForm="NgForm" (ngSubmit)="onSubmit(userForm)">
  <input type="text" ngModel required>
  <input type="text" ngModel required>
</form>
```

Example

Reactive Forms

```
export class UserComponent {
    public user = new FormGroup({
        name: new FormControl(‘’, Validators.required),
        surname: new FormControl(‘’, Validators.required)
    })
}
```

Example

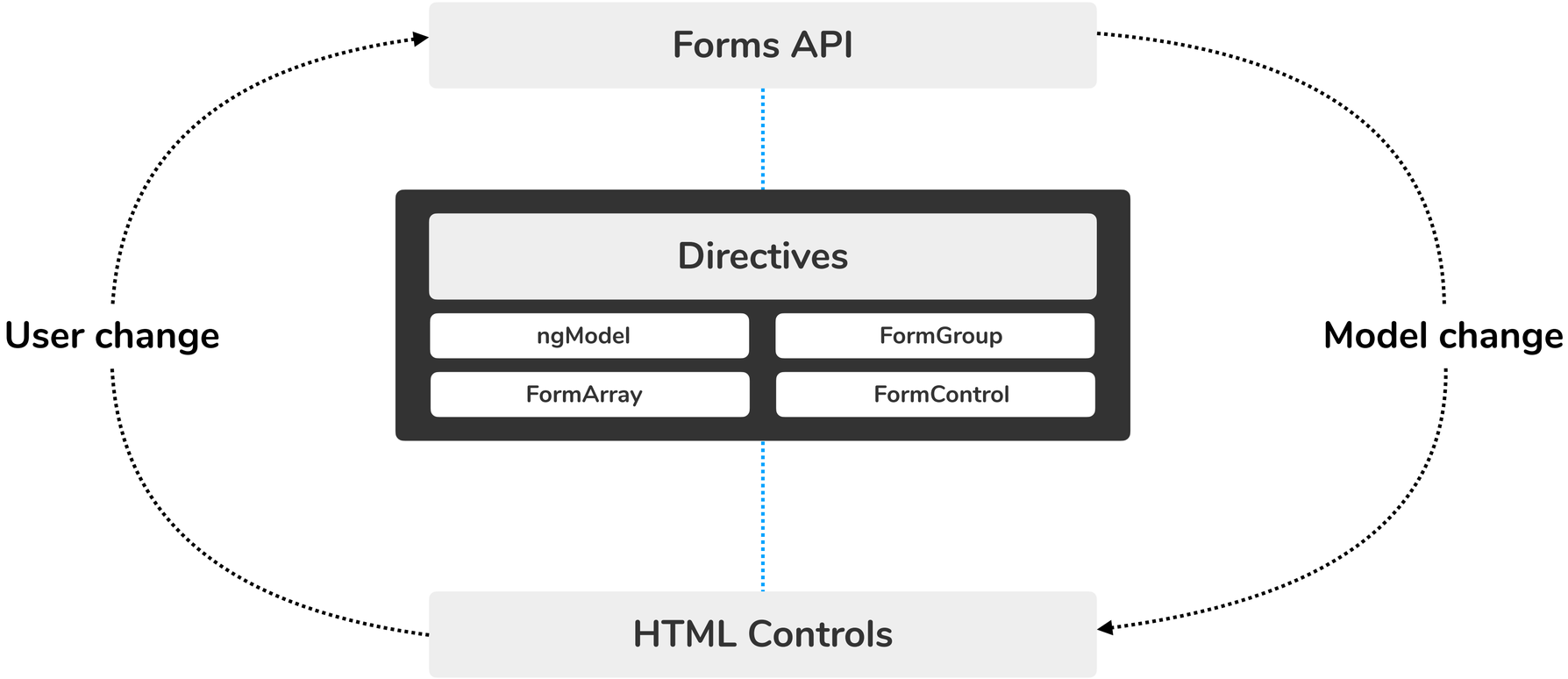
Reactive Forms

Creazione del Form nella classe del componente

```
<form [formGroup]="user" (ngSubmit)="onSubmit()">
  <input type="text" formControlName="name">
  <input type="text" formControlName="surname">
</form>
```

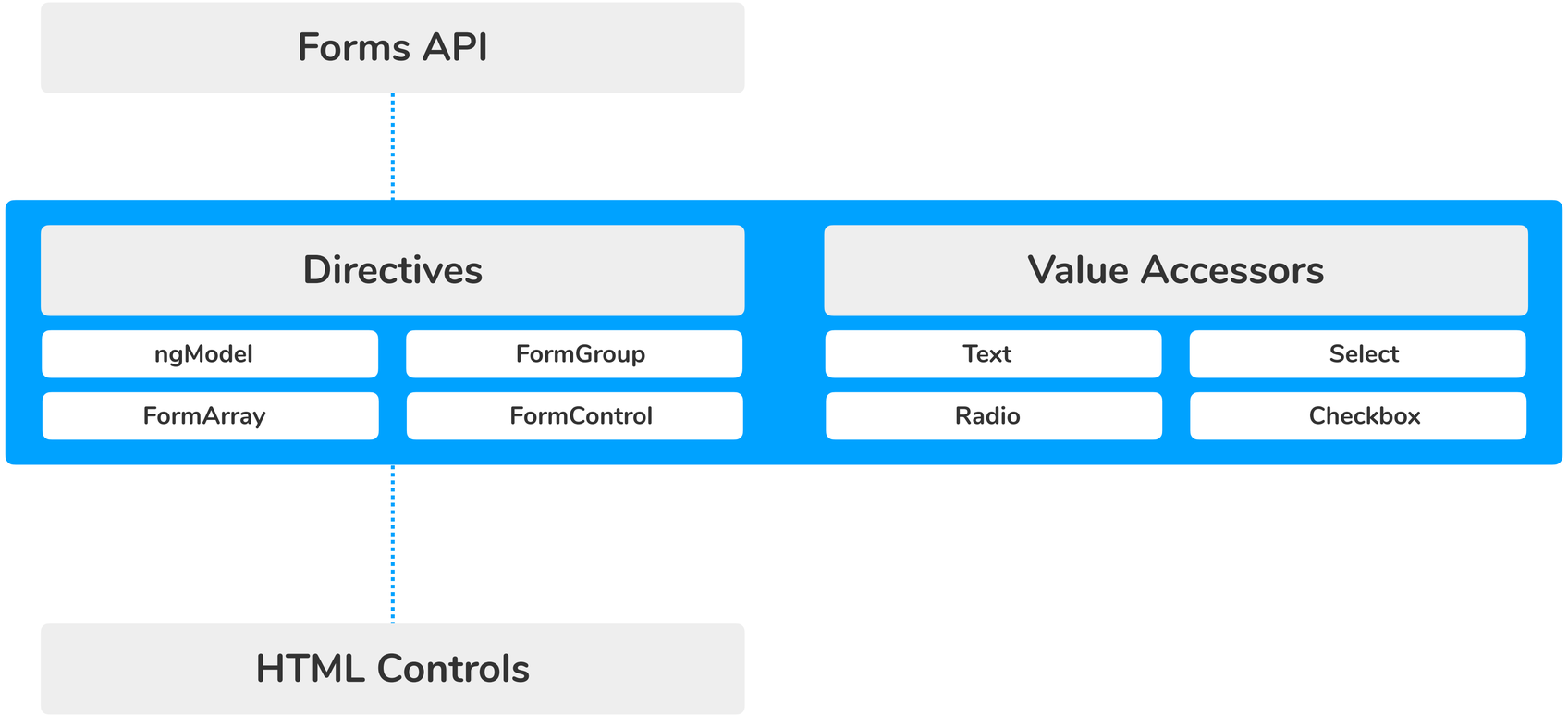
Architecture

Angular Forms



Architecture

Angular Forms

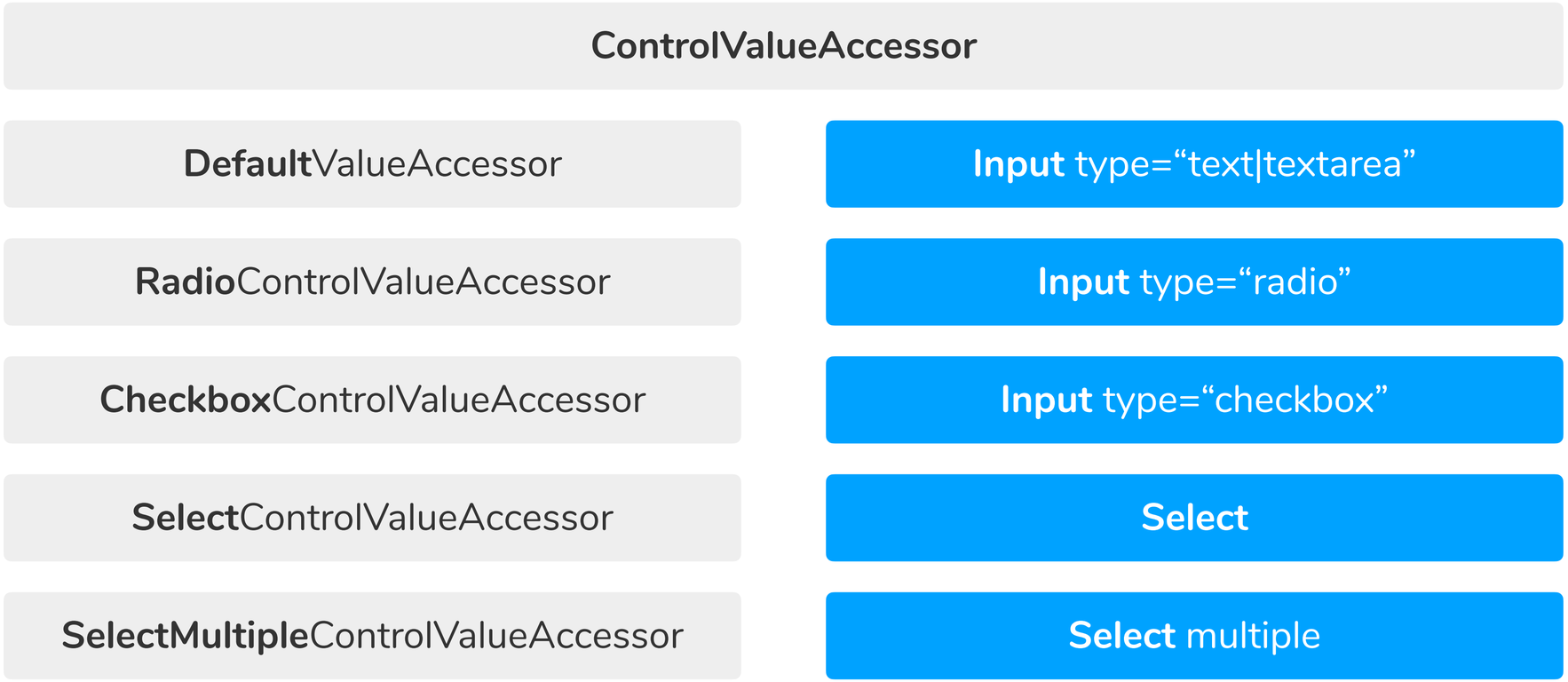


Comunicazione fra @Directive e HTML

Value Accessors

Architecture

ValueAccessor predefiniti



Architecture

ControlValueAccessor Interface

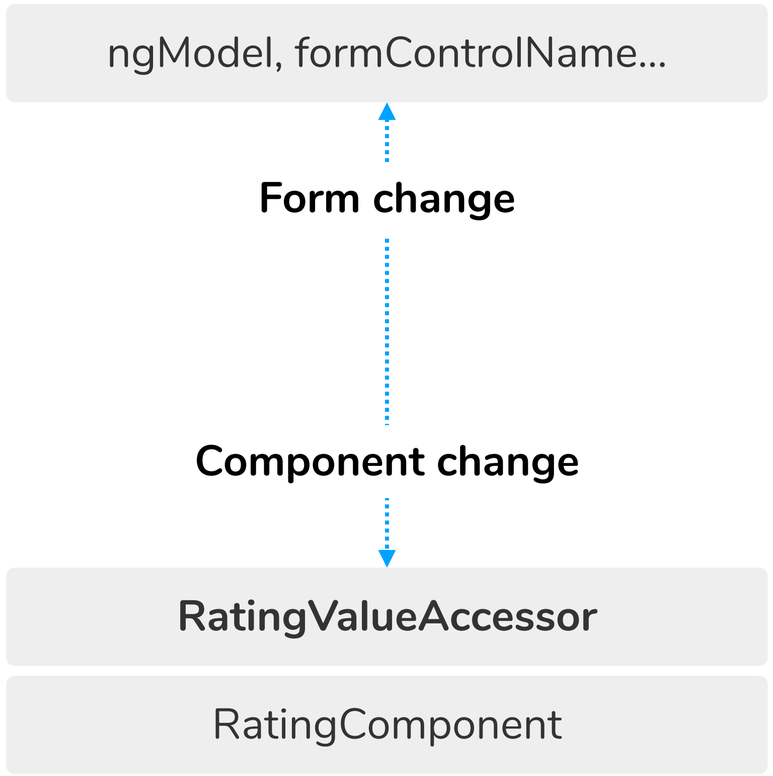
writeValue(value: any): void	**Il modello ha cambiato valore**
registerOnChange(fn: any): void	**Forniscono le funzioni per cambiare il modello (valore, stato)**
registerOnTouched(fn: any): void	
setDisabledState(isDisabled: boolean)?: void	**Il modello è stato disabilitato**

Live code

Introducing **ControlValueAccessor**

Architecture

Cosa abbiamo ottenuto



ControlValueAccessor

Cosa abbiamo ottenuto

- Il componente continua a funzionare anche senza form
- Il componente comunica perfettamente con un form padre
- **Riutilizzabile sia per Reactive che Template-driven forms!**
- Best practice: basso rischio di errori

STYLE	LENSES	FRONT	TEMPLES	SIZE	ENGRAVING	CASE
				55-16		
JUSTIN	BLUE MIRRORED	BLACK RUBBER	BLACK RUBBER	STANDARD SIZE	CUSTOMIZE IT.	BROWN

$163.00

Or 4 payments of $40.75

with afterpay

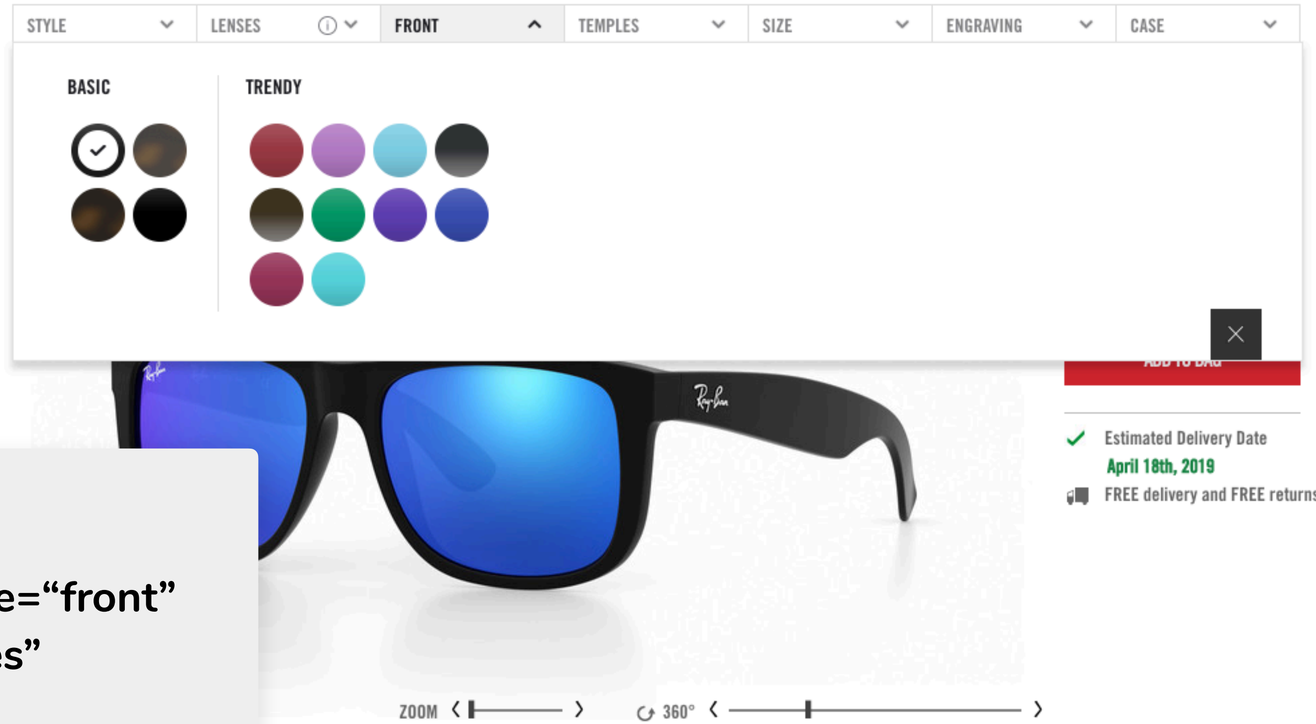
ADD TO BAG

Estimated Delivery Date

April 18th, 2019

FREE delivery and FREE returns

ZOOM 360°




```
<options-images
        formControlName=“front”
        [options]=“images”
        [type]=“‘round’”
        [orientation]=“‘horizontal’”
</options-images>
```

ControlValueAccessor

Cosa possiamo ottenere

- Form nidificati
- Auto-validazione (attenzione!)

Example

Form Nidificati

```
<form>
    <customer></customer>
    <address></address>
    <payment></payment>
</form>
```

nome, cognome, codice fiscale...

indirizzo, città, cap, nazione...

metodo, numero di carta...

Form nidificati

***Bad* Practice #1:** passare il form al componente

```
<form #f="NgForm">
    <customer [form]="f"></customer>
    <address [form]="f"></address>
    <payment [form]="f"></payment>
</form>
```

Form nidificati

***Bad* Practice #1:** passare il form al componente

```
<form [formGroup]="form">
    <customer [form]="form"></customer>
    <address [form]="form"></address>
    <payment [form]="form"></payment>
</form>
```

Form nidificati

***Bad* Practice #1:** passare il form al componente

- Un sotto-componente **manipola una nostra proprietà**
  - Può agganciarsi a controlli esistenti
  - Può crearne di nuovi
  - Può modificare il form e il suo stato
- Il form diventa fuori controllo
- Alto rischio di bug

Bad Practice #2: accedere al form padre

```
constructor(parentForm: FormGroupDirective) {
    parentForm.addControl('customer', new FormGroup({
        name: new FormControl(),
        surname: new FormControl(),
        fiscalCode: new FormControl()
    }));
}
```

Bad Practice #2: accedere al form padre

```
@Component({
    selector: ‘customer’,
    template: ‘
        <div ngModelGroup=“customer”>
            <input ngModel name=“name”>
            <input ngModel name=“surname”>
        </div>
    ’,
    viewProviders: [{ provide: ControlContainer, useExisting: NgForm }]
})
```

Form nidificati

***Bad* Practice #2:** accedere al form padre

- Un sotto-componente **manipola una nostra proprietà**
  - Può agganciarsi a controlli esistenti
  - Può crearne di nuovi
  - Può modificare il form e il suo stato
  - E noi non lo vediamo nemmeno! (no binding)
- Il form diventa fuori controllo
- Alto rischio di bug
- **Non riutilizzabile fra Reactive e Template-driven forms**

Form nidificati

Best Practice: ControlValueAccessor

```
<form [formGroup]="form">
    <customer formControlName="customer"></customer>
    <address formControlName="address"></address>
    <payment formControlName="payment"></payment>
</form>
```

Form nidificati

Best Practice: ControlValueAccessor

- Può contenere un form indipendente dal principale
- Incapsulato: non accede al padre direttamente
- Basso rischio di bug
- **Riutilizzabile sia con Reactive che Template-driven forms**
- Rimane un componente: possiamo passargli **[input]** e ricevere **(output)**
- Dà soddisfazione personale allo sviluppatore!

Form nidificati

Gestione degli errori

- Best practice: il validatore si applica dall'esterno (componente del form principale)
  - Se ci pensate, un input non si valida da solo, è solamente un contenitore
  - Potete usare sia funzioni validatrici (reactive) che direttive validatrici (template-driven)
  - Se il componente deve visualizzare errori internamente, possiamo passarli come [input] oppure AL LIMITE iniettare *NgControl* (attenzione)
- Il controllo può auto-validarsi implementando l'interfaccia **Validator**
  - Massima attenzione: guardando il componente padre, non ve ne accorgerete

Gestione degli errori: dall'esterno

```
<form #f="NgForm">
    <customer ngModel customerValidator />
    <address ngModel addressValidator />
    <payment ngModel paymentValidator />
</form>
```

Gestione degli errori: dall'esterno

```
this.form = new FormGroup({
    customer: new FormControl({}, customerValidator),
    address: new FormControl({}, addressValidator),
    payment: new FormControl({}, paymentValidator)
})
```

ControlValueAccessor

Gestione degli errori: auto-validazione

```
@Component({
    ...
    providers: [
        { provide: NG_VALUE_ACCESSOR, useExisting: MyComponent, multi: true },
        { provide: NG_VALIDATORS, useExisting: MyComponent, multi: true }
    ]
})
export class MyComponent implements ControlValueAccessor, Validator {

    validate(control: AbstractControl) {
        return { ... } || null;
    }
}
```

ControlValueAccessor

Gestione degli errori: auto-validazione **async**

```
@Component({
    ...
    providers: [
        { provide: NG_VALUE_ACCESSOR, useExisting: MyComponent, multi: true },
        { provide: NG_ASYNC_VALIDATORS, useExisting: MyComponent, multi: true }
    ]
})
export class MyComponent implements ControlValueAccessor, AsyncValidator {

    validate(control: AbstractControl) {
        return of({ ... } || null);
    }
}
```

Riepilogo

- Creare controlli custom è facile grazie a **ControlValueAccessor**
- Riutilizzabili fra Reactive e Template-driven forms
- È la soluzione ideale per form annidati complessi
  - Applicate i validatori dal padre, se possibile…
  - …altrimenti utilizzate le interfacce **Validator** / **AsyncValidator**
- Evitate di iniettare controlli padre nei figli
- Evitate di consegnare il form ai componenti figli

Link utili

- Slide del talk:
  - github.com/UserGalileo/talks
- Contatti:
  - Facebook — https://fb.com/michelestieven
  - Twitter — https://twitter.com/MicheleStieven
  - Medium — https://medium.com/@michelestieven
  - LinkedIn — https://linkedin.com/in/michelestieven
  - Website — https://michelestieven.it