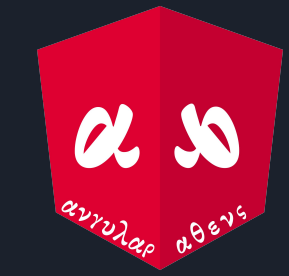
# Reactive Forms with Smart Dumb Components

by Fanis Prodromou



# Agenda

- Reactive vs Template Driven Forms
- Reactive Forms - Basics
- Binding and Validation
- Interaction with Template
- Nested Form in Smart Dumb Components

# Template driven vs Reactive forms

## Template Driven

- Unit test against **DOM**
- Validation in **HTML**

## Reactive Forms

- Unit test against **Form Model**
- Validation in **component**
- Form Model in component
- Complex validations
- Conditional validations
- Immutable data model
- Add dynamically HTML elements

# Shift of responsibility

## Template Driven

Template

- Form elements
- Binding to a **data** model
- **Validation conditions**
- **Validation messages**

Component

- **Data model**
- Form methods

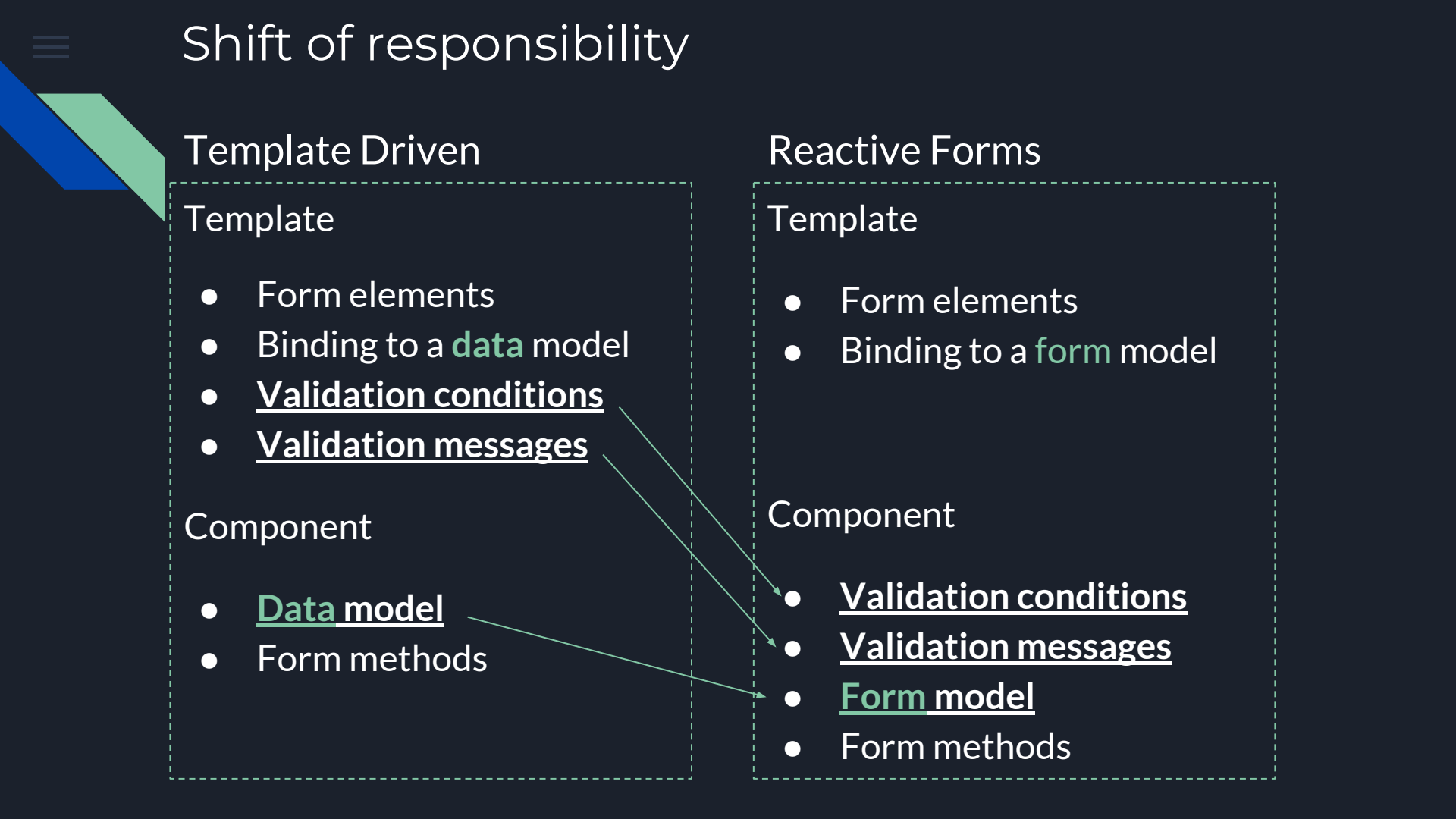
## Reactive Forms

Template

- Form elements
- Binding to a form model

Component

- **Validation conditions**
- **Validation messages**
- **Form model**
- Form methods



# Forms are not always that easy...

# The basics of Reactive Forms

Handles the value and validity state of:

**Form Control**

- the form elements

**Form Group**

- a set of Form Controls

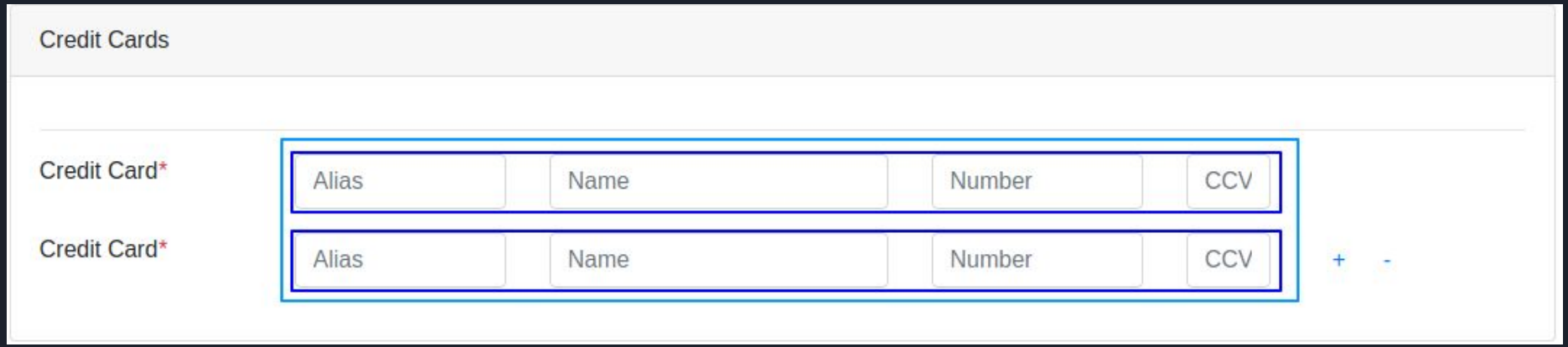
**Form Array**

- an array of FormControl, FormGroup

# Form Control on the UI

# Form Group on the UI

# Form Array on the UI



# Registering the reactive forms module

```
import { NgModule } from '@angular/core';
import { ReactiveFormsModule } from '@angular/forms';

@NgModule({
  imports: [ReactiveFormsModule],
})
export class MyModule {}
```

# Initialize the FormGroup

```
@Component({
  selector: 'app-customer-credit-cards',
  templateUrl: './customer-credit-cards.component.html'
})
export class CustomerCreditCardsComponent implements OnInit {

  creditCardsForm: FormGroup;

  ngOnInit() {
    this.creditCardsForm = new FormGroup({
    });
  }

}
```

# Initialize the FormGroup

```
@Component({
  selector: 'app-customer-credit-cards',
  templateUrl: './customer-credit-cards.component.html'
})
export class CustomerCreditCardsComponent implements OnInit {

  creditCardsForm: FormGroup;

  ngOnInit() {
    this.creditCardsForm = new FormGroup({
        cardAlias: new FormControl(),
        cardHolderName: new FormControl(),
        cardNumber: new FormControl(),
        ccv: new FormControl()
    });
  }

}
```

# Initialize the FormGroup

```
import { Validators } from '@angular/forms';
@Component({})
export class CustomerCreditCardsComponent implements OnInit {

  creditCardsForm: FormGroup;

  ngOnInit() {
    this.creditCardsForm = new FormGroup({
        cardAlias: new FormControl(null, Validators.required),
        cardHolderName: new FormControl(null, Validators.required),
        cardNumber: new FormControl(null, [Validators.required,
                                           Validators.min(0)]),
        ccv: new FormControl(null, Validators.required)
    });
  }
}
```

# Initialize the FormGroup with FormBuilder

```
import { Validators } from '@angular/forms';
@Component({})
export class CustomerCreditCardsComponent implements OnInit {

  creditCardsForm: FormGroup;

  constructor(private fb: FormBuilder) {}

  ngOnInit() {
    this.creditCardsForm = this.fb.group({
      cardAlias: [null, Validators.required],
      cardHolderName: [null, Validators.required],
      cardNumber: [null, [Validators.required, Validators.min(0)]],
      ccv: [null, Validators.required]
    });
  }
}
```

# Associating the FormGroup model and view

```
<div  [formGroup]="creditCardsForm">

    <input type="text" formControlName="cardAlias" placeholder="Alias" />
    <input type="text" formControlName="cardHolderName" placeholder="Name" />
    <input type="text" formControlName="cardNumber" placeholder="Number" />
    <input type="text" formControlName="ccv" placeholder="CCV" />

</div>
```

# Form Model Properties

## Validity

- valid
  - The form is valid
- invalid
  - The form is invalid

## Visited

- untouched
  - The default property
- touched
  - If the user focus on any form field

## Value Changed

- pristine
  - No values has been entered in the form
- dirty
  - The form has been altered

# Access Form Model Properties

01

```
formGroup.controls.formControlName.property
```

02

```
formGroup.get('formControlName').property
```

# Utilize the Form Model Properties 1/2

```
<div  [formGroup]="creditCardsForm">


    <input  type="text"
            formControlName="cardAlias"
            placeholder="Alias"
            [ngClass]="{'redBorder': !creditCardsForm.get('cardAlias').valid}"/>

</div>
```

# Utilize the Form Model Properties 2/2

```
<div  [formGroup]="creditCardsForm">


    <input  type="text"
            formControlName="cardAlias"
            placeholder="Alias"
            [ngClass]="{'redBorder': !creditCardsForm.controls.cardAlias.valid}"/>

</div>
```

# Initialize the FormArray

```
@Component({
  selector: 'app-customer-credit-cards',
  templateUrl: './customer-credit-cards.component.html'
})
export class CustomerCreditCardsComponent implements OnInit {

  creditCardsFormArray: FormArray;

  ngOnInit() {
    this.creditCardsFormArray = new FormArray([
    ]);
  }
}
```

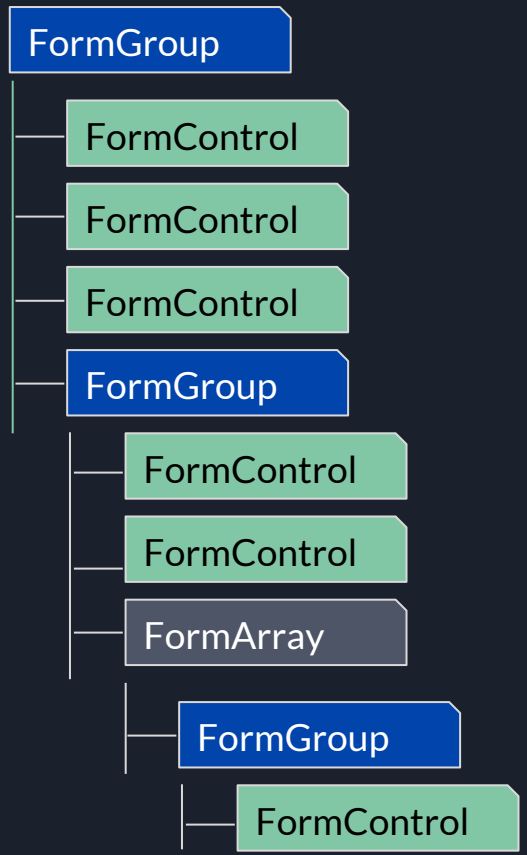
# Initialize the FormArray with FormGroup

```
@Component({
  selector: 'app-customer-credit-cards',
  templateUrl: './customer-credit-cards.component.html'
})
export class CustomerCreditCardsComponent implements OnInit {

  creditCardsFormArray: FormArray;

  ngOnInit() {
    this.creditCardsFormArray = new FormArray([
       new FormGroup({
        cardAlias: new FormControl(),
        cardHolderName: new FormControl(),
        cardNumber: new FormControl(),
        ccv: new FormControl()
      })
    ]);
  }
}
```

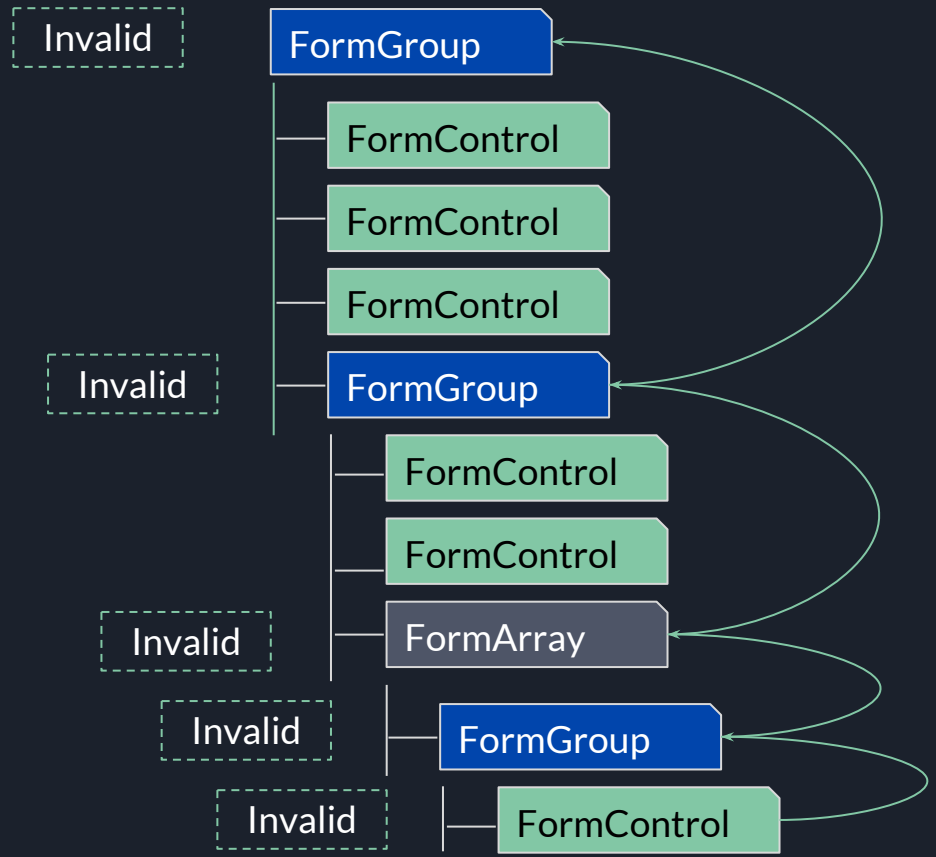
# Associating the FormArray and view

```
<div  *ngFor="let card of creditCardsFormArray.controls;"
      [formGroup]="card">

    <input type="text" formControlName="cardAlias" placeholder="Alias" />
    <input type="text" formControlName="cardHolderName" placeholder="Name" />
    <input type="text" formControlName="cardNumber" placeholder="Number" />
    <input type="text" formControlName="ccv" placeholder="CCV" />

</div>
```

# Form Tree Hierarchy



# Form Status Traverse The Tree



# Let’s see an example

## Basic Info

Name*
First Name
Last Name

Email*

Gender

Age

Phone
Area Code
Number

## Address

Address*
Street
Number

Postal Code*

Country

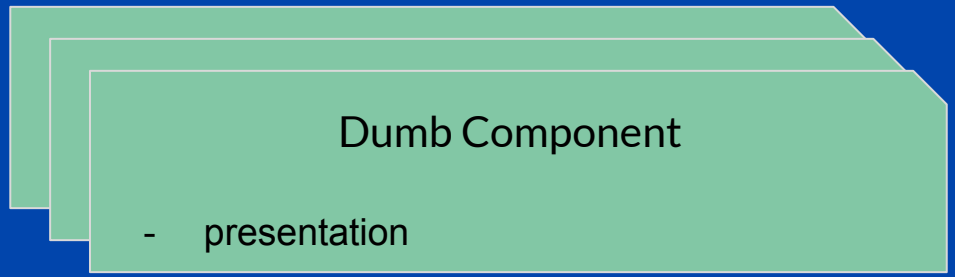
## Credit Cards

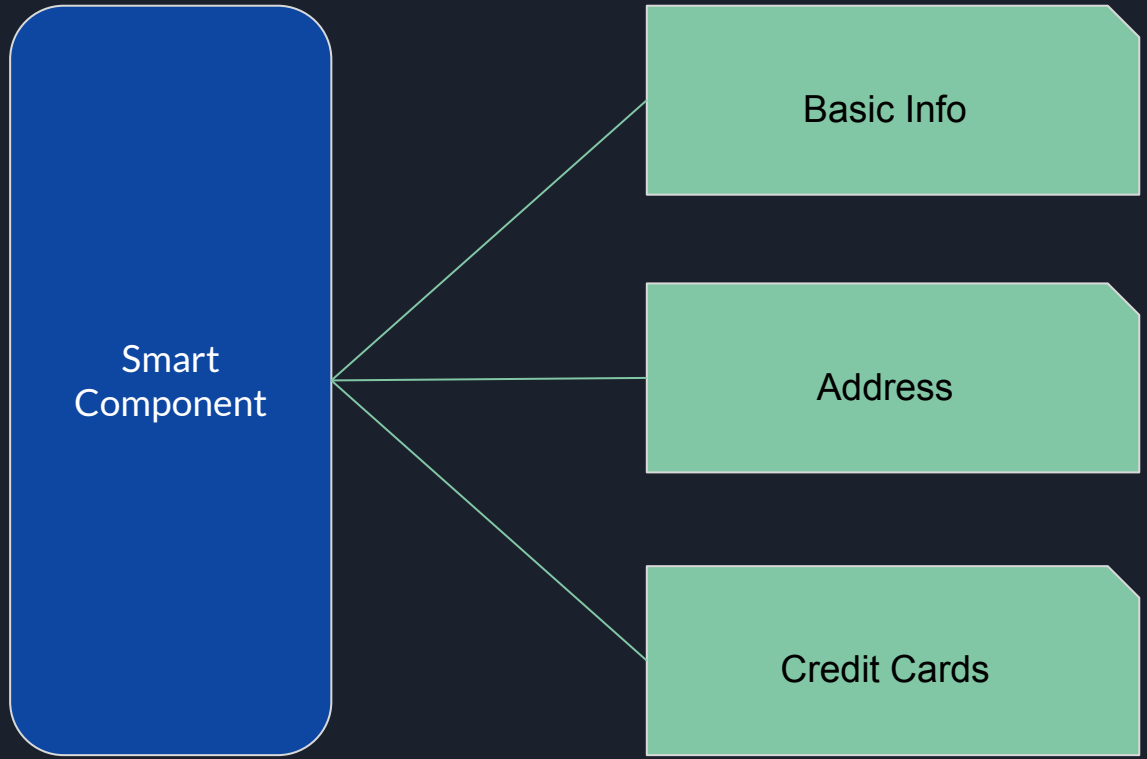
Credit Card*
Alias
Name
Number
CCV
+

Submit

## Smart Component

- (how things works)
- Component data orchestration
- Collect data from services
- Initiate the communication with server API




Smart
Component
Basic Info
Address
Credit Cards

# The problem

keep track of form validity

unit tests

# Approach #1

all in one

# Smart Component

```
this.theForm = this.fb.group({
  basic: this.fb.group({
    firstName: ['', Validators.required],
    lastName: ['', Validators.required],
    age: ['', Validators.min(0)],
    gender: [],
    email: ['', [Validators.required, Validators.email]],
    phone: this.fb.group({
      areaCode: [],
      phoneNumber: []
    })
  }),
  address: this.fb.group({
    street: ['', Validators.required],
    number: ['', [Validators.required, Validators.min(0)]],
    postal: ['', Validators.required],
    country: []
  }),
  creditCards: this.fb.array([
    this.fb.group({
      cardAlias: [],
      cardHolderName: [],
      cardNumber: [],
      ccv: []
    })
  ])
});
```

# Smart Component - HTML

```
<form [formGroup]="theForm" novalidate autocomplete="off">

  <app-customer-basic [basicFormGroup]="theForm.get('basic')">
  </app-customer-basic>

  <app-customer-address [addressFormGroup]="theForm.get('address')">
  </app-customer-address>

  <app-customer-credit-cards [creditCardsFormArray]="theForm.get('creditCards')">
  </app-customer-credit-cards>

  <button type="submit" class="btn btn-primary" [disabled]="!theForm.valid">Submit</button>

</form>
```

# Approach #1

## Pros

- The smart component is automatically updated of the form status
- Lean dumb components and easy to maintain

## Cons

- Hard to identify what is the model of each component
- Re-create the form model:
  - unit testing
  - addCreditCard and removeCreditCard

# Approach #2

improve credit cards

# Smart Component

```
this.theForm = this.fb.group({
  basic: this.fb.group({
    firstName: [, Validators.required],
    lastName: [, Validators.required],
    age: [, Validators.min(0)],
    gender: [],
    email: [, [Validators.required, Validators.email]],
    phone: this.fb.group({
      areaCode: [],
      phoneNumber: []
    })
  }),
  address: this.fb.group({
    street: [, Validators.required],
    number: [, [Validators.required, Validators.min(0)]],
    postal: [, Validators.required],
    country: []
  }),
  creditCards: this.fb.array([])
});
```

# Credit Cards Component

```
ngOnInit() {
  this.creditCardsFormArray.push(this.initCreditCard
}));

addCreditCard() {
  const control = this.creditCardsFormArray;
  control.push(this.initCreditCard());
}

removeCreditCard(index: number) {
  const control = this.creditCardsFormArray;
  control.removeAt(index);
}

initCreditCard() {
  return this.fb.group({
    cardAlias: ['', Validators.required],
    cardHolderName: ['', Validators.required],
    cardNumber: ['', Validators.required],
    ccv: ['', Validators.required]
  });
}
```

# Approach #2 pros - cons

## Pros

- The smart component is automatically updated of the form status
- The dumb components are lean and easy to maintain
- Responsible for the form model is only the credit cards component

## Cons

- Hard to identify what is the model of each component
- Re-create the form model:
  - unit testing
- Inconsistency of the initialization of the form model in components

# Approach #3

combine statuses

# Smart Component - HTML

```
<form novalidate autocomplete="off">

  <app-customer-basic>
  </app-customer-basic>

  <app-customer-address>
  </app-customer-address>

  <app-customer-credit-cards>
  </app-customer-credit-cards>

  <button type="submit" [disabled]="!(formIsValid$ | async)">Submit</button>

</form>
```

# Smart Component

```
formIsValid$: Observable<boolean>;

@ViewChild(CustomerBasicComponent) customerBasicComponent;
@ViewChild(CustomerCreditCardsComponent) customerCreditCardsComponent;
@ViewChild(CustomerDetailComponent) customerDetailComponent;

ngAfterViewInit(): void {
  const statusIsTrue = map(status => status === 'VALID');

  const basicFormStatus =
    this.customerBasicComponent.basicFormGroup.statusChanges.pipe(statusIsTrue);

  const creditCardFormStatus =
    this.customerCreditCardsComponent.creditCardsFormArray.statusChanges.pipe(statusIsTrue);

  const addressFormStatus =
    this.customerDetailComponent.addressFormGroup.statusChanges.pipe(statusIsTrue);

  this.formIsValid$ = combineLatest(basicFormStatus,
                                    addressFormStatus,
                                    creditCardFormStatus).pipe(
                                        map((statuses: boolean[]) => statuses.every(Boolean))
                                    );

}
```

# Approach #3 pros - cons

## Pros

- The dumb components are lean and easy to maintain
- Easy to identify the model of each dumb component as we decreased the distance
- We shouldn't re-create the form of each dumb component for the unit testing
- Consistent form initialization
- The form status of smart component is automatically updated

## Cons

- Requires orchestration on the smart component

# Q & A

Thank you!!


/prodromouf


/prodromouf

**Blog** https://profanis.weebly.com/