

# NG FORMULAIRES

INF1013, Hiver 2022

DMI, UQTR

# Plan

- Controller Driven Forms (reactive forms)
- Template Driven Forms
- Validation de formulaire

# Introduction

- Angular utilise 2 approches pour gérer les formulaires.
  - Les Reactive Forms (RF)
    - Plus structurées
    - Mise à l'échelle plus facile
    - Destinés à de gros formulaires
  - Les Templates Driven Forms (TDF)
    - Mise en place plus simple
    - Beaucoup moins structurés
    - Destinés aux petits formulaires.
- La construction des RF débute dans le contrôleur (on les appelle aussi Controller Driven Forms).
- La construction des TDF démarre dans le template.

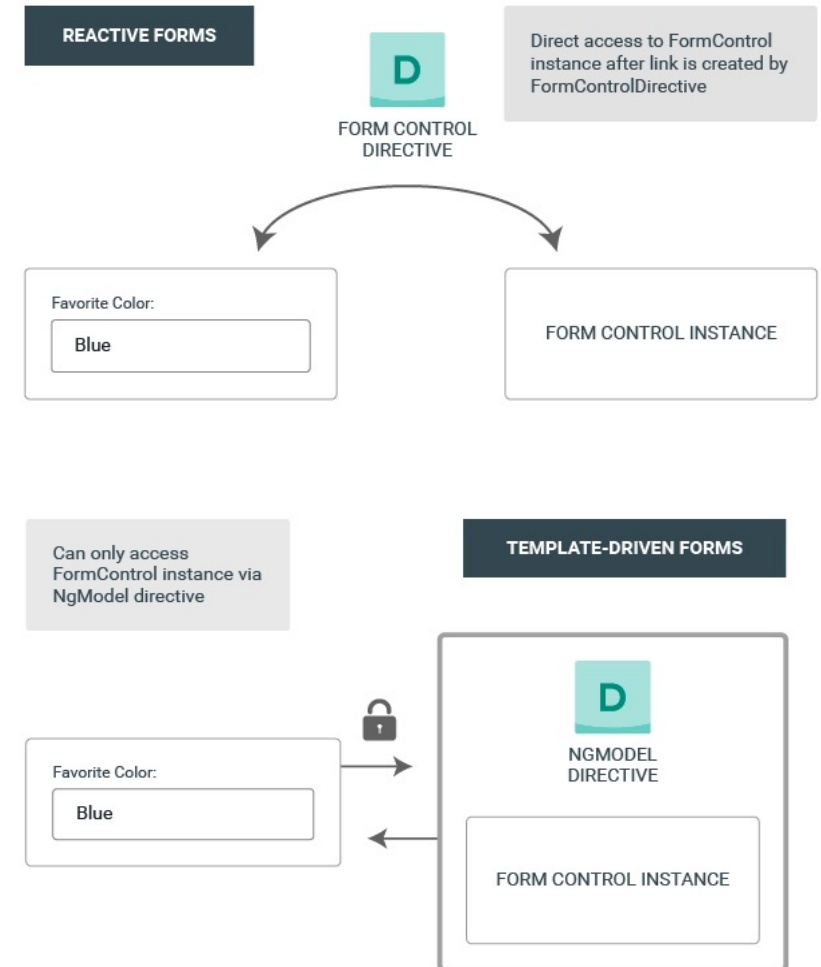
# Caractéristiques des RF et des TDF

- RF:

- Composant central: [FormControl](#)
- Directive: [formControl].

- TDF

- Composant central: var membre du contrôleur
- Directive: [([ngModel](#))]



# Exemple de Template Driven Form

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

```
@Component({  
  selector: 'app-template-favorite-color',  
  template: `  
    Favorite Color: <input type="text" [(ngModel)]="vName">  
  `,  
})  
export class FavoriteColorComponent {  
  vName = '';  
}
```

- Créer une input et l'associer à la directive **ngModel** dans le template.
- Faire la double-liaison (double-binding) avec une variable membre du contrôleur

# Exemple de Reactive Form

```
import { Component } from '@angular/core';
import { FormControl } from '@angular/forms';

@Component({
  selector: 'app-reactive-favorite-color',
  template: `
    Favorite Color: <input type="text" [formControl]="vName">
  `
})
export class FavoriteColorComponent {
  vName = new FormControl("");
}
```

- Créer l'instance de **FormControl** dans le contrôleur
- Créer l'input et l'associer à la directive **formControl** dans le template.
- Cette directive sera liée à l'instance du FormControl déclarée plus tôt

# Les Reactives Forms: Regroupement

- Les RF permettent un regroupement logique des formulaires.
  - [FormGroup](#)
  - L'objectif de ce regroupement est d'obtenir une structure de donnée cohérente (json) qui sera contenue dans l'instance *value* de FormGroup
- La création débute dans le contrôleur.

```
profileForm = new FormGroup({  
  firstname: new FormControl(""),  
  lastname: new FormControl(""),  
  address: new FormGroup({  
    street: new FormControl(""),  
    city: new FormControl(""),  
    state: new FormControl(""),  
    zip: new FormControl("")  
  })  
})
```

# Les Reactive Forms: Regroupement

- Le binding dans le template avec les champs correspondants.

```
<form [formGroup]="profileForm">
  <label>Prénom: <input type="text" FormControlName="firstname"></label>
  <label>Nom: <input type="text" FormControlName="lastname"></label>
  <div FormGroupName="address">
    <h3>Adresse</h3>
    <label>Rue: <input type="text" FormControlName="street"></label>
    <label>Ville: <input type="text" FormControlName="city"></label>
    <label>Province: <input type="text" FormControlName="state"></label>
    <label>CP: <input type="text" FormControlName="zip"></label>
  </div>
</form>
```

- Notez ici que
  - le binding avec le group se fait une seule fois avec la directive **[formGroup]**.
  - La directive **FormGroupName** permet ensuite d'identifier les sous groupes en les bindant avec leur noms.
  - La directive **FormControlName** permet de binder le champ à son nom dans le groupe.



# Les Reactives Forms: Interactions

- On peut modifier le contenu des formulaires de deux façons
  - En interagissant avec le template, ce qui se traduit par l'écriture de données
  - De manière programmatique dans le contrôleur.
- La manière programmatique peut faire appelle aux méthodes
  - setValue() pour des formControls individuels

```
this.fname = this.profileForm.get('firstname') as FormControl;  
this.fname.setValue('Jean');
```

- patchValue() pour les formGroups // projection

```
this.profileForm.patchValue(  
  {  
    firstname: 'Nancy',  
    address: {  
      street: '123 Drew Street'  
    }  
  });
```

# Les Reactives Forms: Pattern Fabrique

- Pour éviter les constructions lourdes des instances de chaque formControl dans les groupes, on utilise un service de fabrique à l'aide du composant [FormBuilder](#) .
  - Le composant est injecté: `constructor(private fb: FormBuilder){}`

```
profileForm = this.fb.group({  
  firstName: [''],  
  lastName: [''],  
  address: this.fb.group({  
    street: [''],  
    city: [''],  
    state: [''],  
    zip: ['']  
  }),  
});
```

# Les Reactive Forms: Formulaire Dynamique

- Dans certaines circonstances, on est emmené à créer des champs à la volée.
  - Exemple ajouter des emails professionnel, secondaire, tertiaire...
  - Dans ces conditions on aimerait que l'utilisateur ajoute des champs (à la volée)
- RF propose le composant [FormArray](#)
  - La création débute dans le contrôleur à l'intérieur de n'importe quel formGroup

```
contacts: this.fb.array([  
  this.fb.control("")  
])
```

# Les Reactive Forms: Formulaire Dynamique

- L'ajout de contrôle dynamiquement se fait ainsi

```
this.contacts = this.profileForm.get('contacts') as FormArray;  
this.contacts.push(this.fb.control(""));
```

- Le lien avec le template se fera alors à l'aide de la directive \*ngFor

```
<div formArrayName="contacts">  
  <h3>Contact</h3>  
  <button (click)="addContact()">Add Contact</button>  
  <div *ngFor="let contact of contacts.controls; let i=index">  
    <label>  
      Contact:{{i}} <input type="text" [formControlName]="i">  
    </label>  
  </div>  
</div>
```

# Exercice Pratique

- Implémentez le formulaire de modification/ajout d'étudiant
  - Utiliser les formGroups
  - Afficher la variable **value** du formGroup avec le pipe json, dans la page
- Implémentez l'ajout des notes à l'aides des formArrays.

# Validation des Formulaires RF

- Le formulaire (ou FormGroup) a une variable d'état **valid**
  - Cette variable détermine l'état de validité du formulaire à chaque modification, cet état est mis à jour.
  - Exemple pour n'activer la soumission que si le formulaire est valide.

```
<button type="submit" [disabled]="!studentForm.valid">Inscrire</button>
```

- La première forme de validation consiste à vérifier que les champs obligatoires sont remplis.
  - Le composant Validators de Angular propose une famille de validateurs prédéfinis.
  - Ex: Validators.*required*

# Validation des Formulaire RF

- Le module RF propose une familles de validateur prédéfinis qui peuvent être étendus par l'utilisateur.
- Il existe deux formes de validateurs:
  - Les validateurs synchrones
    - Valide immédiatement après chaque changement d'état.
    - Ces validateurs sont passés en seconds paramètre à la construction des FormControls
  - La validateurs asynchrones
    - Valide de manière retardée à l'aide des observables qui peuvent signaler des erreurs.
    - Ces validateurs sont passés en 3iem argument à la construction des FormControls

# Validation des Formulaires RF: Exemple

- Si l'on veut rendre les champs nom, prénom, email obligatoires.

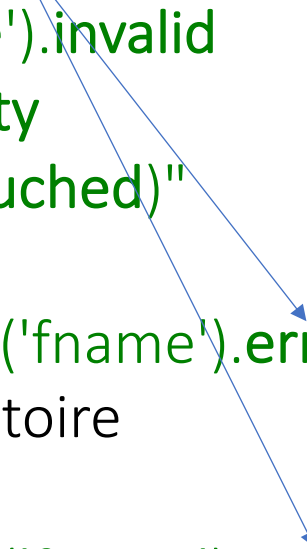
```
this.studentForm = this.fb.group({  
  fname: ['', Validators.required, Validators.minLength(2)],  
  lname: ['', Validators.required],  
  dob: ['', ], pob: ['', ],  
  gender: ['', ], email: ['', Validators.required],  
  tels: this.fb.array([this.createContact() ]),  
  address: this.fb.group({  
    street: ['', ],  
    city: ['', ],  
    state: ['Québec'],  
    zip: ['', ]  
  }  
});
```



# Validation des Formulaires RF: Exemple

- Pour binder les messages d'erreur aux templates:
  - Utiliser le tableau associatif **errors** une v-membre de FormControl

```
<div *ngIf="studentForm.get('fname').invalid
  && (studentForm.get('fname').dirty
    || studentForm.get('fname').touched)"
  class="alert">
  <div *ngIf="studentForm.get('fname').errors.required">
    Le prénom est obligatoire
  </div>
  <div *ngIf="studentForm.get('fname').errors.minlength">
    Le prénom doit contenir au moins 2 caractères
  </div>
</div>
```



# Validation des Formulaires RF: Exemple

- Pour utiliser des validations personnalisées, nous recourrons à des fonctions retournant des lamda que nous définirons dans le contrôleur (à l'extérieur de la classe) ou dans une directive si l'on souhaite l'utiliser plus tard dans les TDF.

```
export function forbiddenName(regParam: RegExp): ValidatorFn {  
    return (control: AbstractControl): {[key: string]: any} | null => {  
        const forbidden = regParam.test(control.value);  
        return forbidden ? {forbiddenName: {value: control.value}} : null;  
    };  
}
```

- L'enregistrement du validateur sur le contrôle se fera comme tous les autres validateurs synchrones

```
lname: ['', [Validators.required, MyValidator.forbiddenName(/ouellet/i)]]
```

# Exercice d'Application

- Reprendre l'application du lab précédent
  - Inclure un menu d'inscription
  - Créer un composant inscription.
  - Inclure à l'aide de *reactive form*, un formulaire complet avec les spécifications suivantes
    - fname, lname, dob, pob, email: obligatoires, email valide
    - Contacts: multiples contacts (mobile, domicile, bureau, skype, autre) valider tel
    - adresses : multiples adresses (correspondance, domiciliaire)
    - Premier choix de program, et Deuxième choix de program (1 choix obligatoire)
  - Lier les messages d'erreur au template.
  - Un bouton **submit** actif uniquement si le formulaire est valide.
- À l'envoi, afficher le formulaire dans la console et sauvegardez-le dans un service de données.