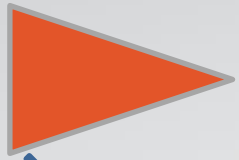


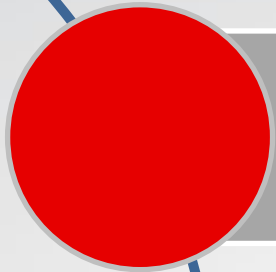


# Création et validation de formulaires dans Angular

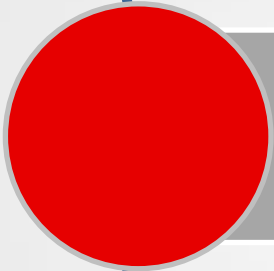
**ÉCOLE SUPÉRIEURE PRIVÉE D'INGÉNIERIE ET DE TECHNOLOGIES**



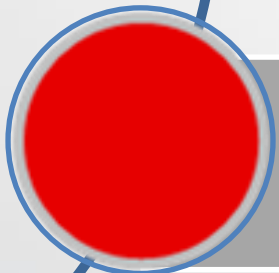
# Plan



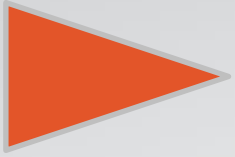
Introduction



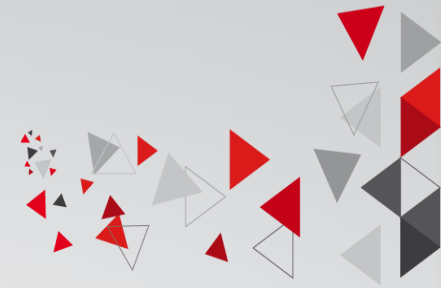
Les fondamentaux des formulaires  
Angular



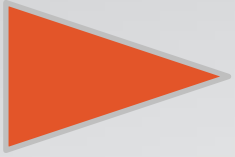
Template – driven



# Introduction (1/2)



- Les formulaires sont presque toujours présents sur tous les sites Web et applications.
- Ils peuvent être utilisés pour effectuer: l'authentification, la création d'un profil, l'envoi d'une demande, contact, etc.
- Un Formulaire, avant sa soumission, doit respecter un ensemble de règles afin de garantir l'intégralité des données soumises => c'est la validation des formulaires



## Introduction (2/2)



- La validation d'un formulaire se fait moyennant des validateurs.
  - Exemple: Un champ est obligatoire. Donc si le champ est vide alors il faut afficher un message indiquant que le champ doit être rempli.
- => Le champ est valide si le validateur « required » est respecté



# Les fondamentaux des formulaires Angular (1/3)


Pour la création des formulaires, Angular propose deux approches:

1. **Reactive Forms (ou Model Driven Forms)**: le formulaire ainsi que les validateurs sont créés dans la classe du composant et ensuite liés au template grâce au DataBinding.
2. **Template Driven Forms** : le formulaire ainsi que les validateurs sont créés directement au niveau du template.

# Les fondamentaux des formulaires Angular (2/3)

Pour utiliser l'une de ces approches, il faut déclarer ça au niveau du module racine, en important le module adéquat.

```
import { ReactiveFormsModule } from '@angular/forms';
import { FormsModule } from '@angular/forms';
@NgModule({
  ....
  imports: [ // other imports ...,
    ReactiveFormsModule,
    FormsModule ],
  ....})
export class AppModule { }
```






# Les fondamentaux de formulaires Angular (3/3)




La différence majeure entre les deux approches est que les **Reactive forms** sont synchrones tandis que les **Template-driven forms** sont asynchrones



# Template-driven – FormControl

## (1/3)



- Chaque élément du formulaire possède une valeur, peut avoir plusieurs contraintes et peut passer par plusieurs états.
- **FormControl** est une classe prédéfinie qui permet de suivre la valeur et l'état de validation d'un élément du formulaire.
- Un élément du formulaire est une instance de la classe **FormControl**



# Template-driven – FormControl

## (2/3)

- La directive **NgModel** assure le two-way DataBinding. En l'utilisant dans un formulaire, cette directive permet, de plus, de suivre l'état de chaque FormControl auquel elle est associée.
- Pour accéder alors aux propriétés d'un FormControl, il faut exporter la directive NgModel dans une variable locale.

```
<input type="text" [(ngModel)]="model.cin" name="cin"  
#cin="ngModel">
```

# Template-driven – FormControl

## (3/3)

La classe FormControl offre plusieurs propriétés qui permettent de suivre l'état d'un élément. [1]

Propriété	Rôle
<b>Valid/invalid</b>	Si valid = true alors invalid=false => détermine si un élément est valid ou non en fonction des validateurs déclarés dessus
<b>errors</b>	Si null => pas d'erreur Si non null (objet) => indique les erreurs commise
<b>Pristine /dirty</b>	Si pristine = true alors dirty = false => l'élément sa valeur initial n'a pas changé et vis versa.
<b>Touched/untouched</b>	Si touched = true alors untouched = false => l'élément a été visité
<b>Status = VALID ou INVALID</b>	Détermine si un élément est valid ou non en fonction des validateurs déclarés dessus

# Template-driven - NgModel

La directive NgModel suit l'état d'un FormControl et le met à jour en lui ajoutant des classes CSS Angular. Exemples:

état	Si état est true	Si état est false
Control visité	ng-touched	ng-untouched
Valeur initiale modifiée	ng-dirty	ng-pristine
Valeur valide	ng-valid	ng-invalid

# Template-driven – NgForm

## (1/3)

- La directive **NgForm** complète le formulaire avec des fonctionnalités supplémentaires.
- En effet, cette directive surveille les propriétés de tout le formulaire.
- Tout élément du formulaire attaché à la directive NgModel, peut être suivi par la directive NgForm.

# Template-driven – NgForm

## (2/3)

- Angular crée et associe automatiquement pour chaque balise `<form>`, la directive NgForm.
- Pour accéder aux propriétés du formulaire, il faut alors exporter la directive NgForm dans une variable référence de template

```
<form #formulaire="ngForm">
```

```
.....
```

```
</form>
```

# Template-driven – NgForm

## (3/3)

La directive NgForm hérite de la classe AbstractControlDirective. Du coup les propriétés retournées par NgForm sont les mêmes que **FormControl** mais qui réfère sur tout le Formulaire avec tous ses éléments. [2]

Propriété	Rôle
<b>Valid/invalid</b>	Si valid = true alors invalid=false => détermine si un formulaire est valide ou non en fonction des validateurs déclarés sur les différents éléments. Un formulaire est valide si tout les éléments du formulaires sont valides.
<b>errors</b>	Si null => pas d'erreur Si non null (objet) => indique les erreurs commises dans les différents éléments du formulaire

# Template-driven – Validation

## (1/4)



- Plusieurs contraintes peuvent être attachés à un élément dans le formulaire:
  1. Les attributs HTML de validation : required, minlength, maxlength, min, max, pattern, ...
  2. Des validateurs personnalisés
- Un élément est valide si la contrainte attachée est respectée.

# Template-driven – Validation (2/4)



Les propriétés **valid**, **invalid** et **errors** permettent de détecter si un élément est valide ou non.

Exemple1: Un seul validateur sur un élément

```
<input name="name" required  
[(ngModel)]="model.name" #name="ngModel" >
```

```
<div *ngIf="name.valid"> Name is required. </div>
```



# Template-driven – Validation

## (3/4)

Exemple 2: Plusieurs validateurs sur un élément

```
<input name="name" required minlength="4"  
appForbiddenName="bob"  
[(ngModel)]="model.name" #name="ngModel" >
```

← Validateur personnalisé

```
<div *ngIf="name.errors.required"> Name is  
required. </div>  
<div *ngIf="name.errors.minlength"> Name must  
be at least 4 characters long. </div>  
<div *ngIf="name.errors.forbiddenName"> Name  
cannot be Bob. </div>
```