

Par NIDHAL JELASSI
nidhal.jelassi@fsegt.utm.tn

PROGRAMMATION WEB AVANCÉE

ANGULAR



Chapitre 7 : Les Forms

OBJECTIFS

- ▶ Créer un formulaire
- ▶ Ajouter des validateurs
- ▶ Appréhender les classes CSS générées par le formulaire
- ▶ Manipuler l'objet ngForm
- ▶ Manipuler les contrôles du formulaire

APPROCHES

- ▶ Etant un framework front-end complet, Angular possède son propre ensemble de bibliothèques pour la création de formulaires complexes.
- ▶ Angular permet de :
 - ▶ Modéliser le formulaire en TypeScript.
 - ▶ Permettre la gestion et la validation du formulaire.
- ▶ La dernière version d'Angular dispose de deux stratégies puissantes de création de formulaires:
 - ▶ **Approche basée Template** (Template-driven forms)
 - ▶ **Approche réactive** (Reactive forms)

2 APPROCHES

Selon la documentation officielle d'Angular :

- ▶ Les **Reactive forms** sont plus robustes: ils sont plus évolutifs, réutilisables et testables. Si les formulaires constituent un élément clé de notre application ou si nous utilisons déjà des modèles réactifs pour générer notre application, c'est le choix idéal.
- ▶ Les **Template-driven forms** sont utiles pour ajouter un formulaire simple à une application, tel qu'un formulaire d'inscription. Ils sont faciles à ajouter à une application, mais ils ne sont pas aussi évolutifs que les formulaires réactifs. C'est un choix idéal si nous avons des exigences de base et une logique pouvant être gérées uniquement dans le template.

APPROCHE BASÉE TEMPLATE

1. Importer le module **FormsModule** dans le AppModule.
2. Angular détecte automatiquement un objet **form** à l'aide de la balise **FORM**. Cependant, il ne détecte aucun des éléments (inputs).
3. Spécifier à Angular quel sont les éléments (contrôles) à gérer. Pour chaque élément (i) ajouter la **directive** angular **ngModel**, (ii) identifier l'élément avec un nom permettant de le détecter et de l'identifier dans le composant.
4. Associer l'objet représentant le formulaire à une variable et la passer à votre fonction en utilisant le référencement interne **#** et la directive **ngForm**

EXEMPLE

```
<form
  (ngSubmit)="onSubmit(formulaire)" #formulaire="ngForm">
```

Template

```
export class
ExampleComponent {
  onSubmit(formulaire:
NgForm) {
  console.log(formulaire);
  }
}
```

Component.ts

PROPRIÉTÉS DES ÉLÉMENTS

- ▶ Afin de valider les propriétés des différents contrôles, Angular utilise des attributs et des directives tel que :
 - ▶ required
 - ▶ email
 - ▶ etc...
- ▶ La propriété **valid** de **ngForm** permet de vérifier si le formulaire est valide ou non en se basant sur les validateurs qu'ils contient.

```
<form #addForm="ngForm">  
  <label for="nom">id</label>  
  <input ngModel required type="text" class="form-control"  
  
  <label for="prenom">adresse mail</label>  
  <input ngModel email type="text" class="form-control" id=
```

CLASSES CSS

- ▶ En détectant le formulaire, Angular décore les différents éléments du formulaire avec des classes (booléennes) qui informe sur leur état :
- ▶ **dirty** : informe sur le fait que l'une des propriétés du formulaire a été modifié ou non
- ▶ **valid-invalid** : informe si le formulaire est valide ou non
- ▶ **Untouched-touched** : informe si le formulaire est touché ou non
- ▶ **pristine** : le formulaire n'a pas été modifié, c'est l'opposé du dirty

Dr IQ

TODO: remove this: form-control ng-untouched ng-pristine ng-valid

EXERCICE

- ▶ Créer un formulaire d'authentification contenant les champs suivants :
 - ▶ Email
 - ▶ Password
 - ▶ Envoyer (Un bouton)
- ▶ Si un champ est invalide alors il devra avoir une bordure rouge.
- ▶ Un champ vide et non encore modifié ne peut avoir de bordure rouge que s'il a été touché. Le bouton « envoyer » ne doit être cliquable que si le formulaire est valide. Utiliser le binding sur la propriété disable.

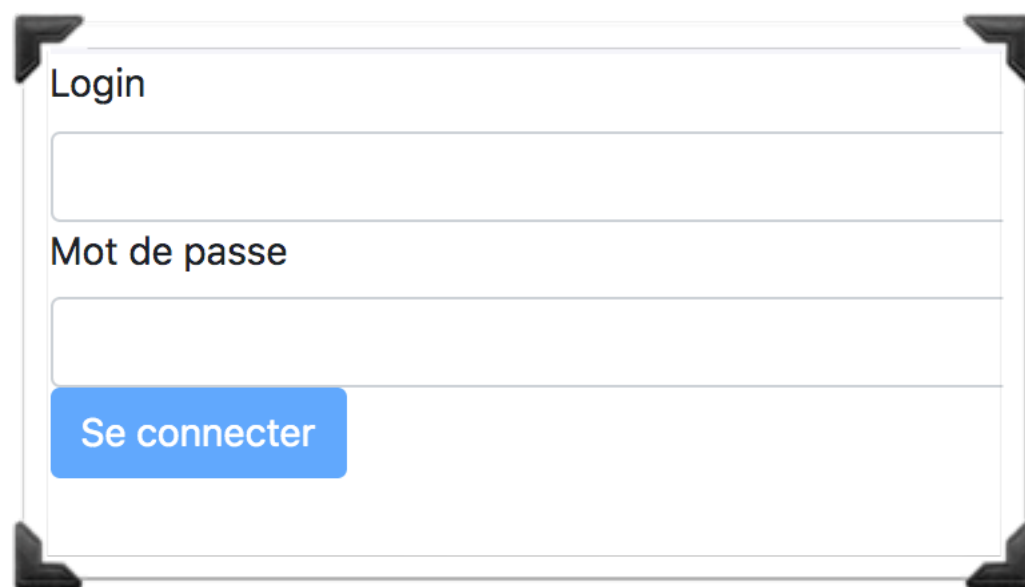
CONTRÔLE DU FORMULAIRE ET SES PROPRIÉTÉS

- ▶ Pour accéder à l'objet form et ces propriétés nous avons utiliser **#notreForm= "ngForm"**
- ▶ Pour les champs du formulaire, pour les récupérer **au sein du formulaire**, c'est la même chose mais au lieu du ngForm c'est un ngModel : **#notreChamp="ngModel"**

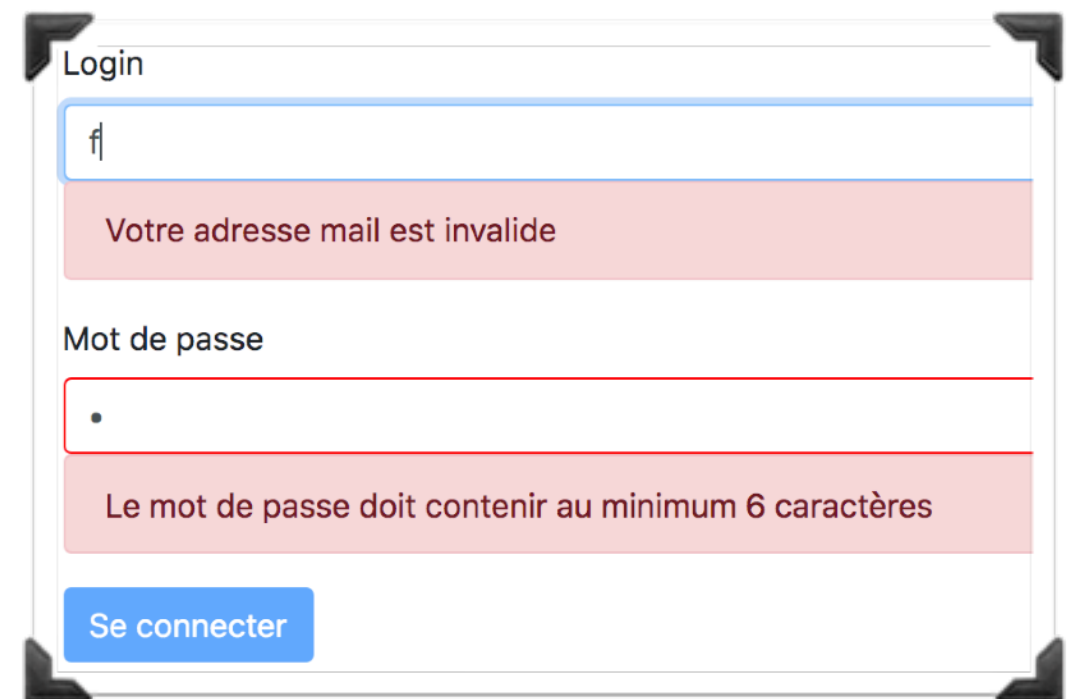
```
<form #addForm="ngForm">  
  <label for="nom">Nom</label>  
  <input ngModel #nom = "ngModel" required type="text" class="form-control">
```

SUITE DE L'EXERCICE PRÉCÉDENT

- ▶ Ajouter un petit message d'erreur qui devra s'afficher sous le champs de l'email s'il est invalide. Ce champ ne devra apparaitre que si l'utilisateur accède ou modifie le champ email.
- ▶ Le password devra avoir au moins 6 caractères. Si l'utilisateur en entre moins, un message d'erreur va s'afficher également.



A login form titled "Login" with two input fields: "Email" and "Mot de passe". Below the "Mot de passe" field is a blue button labeled "Se connecter".



The same login form as before, but with error messages displayed below the input fields. The "Email" field has a red border and a red message box below it saying "Votre adresse mail est invalide". The "Mot de passe" field has a red border and a red message box below it saying "Le mot de passe doit contenir au minimum 6 caractères". The "Se connecter" button remains at the bottom.

ASSOCIER DES VALEURS PAR DÉFAUT AUX CHAMPS

- ▶ Pour associer des valeurs par défaut aux champs d'un formulaire associé à Angular il faut le faire à partir du composant.
- ▶ Afin de gérer les valeurs du formulaire à partir du composant il faut du binding.
- ▶ Au lieu d'avoir juste la primitive `ngModel` associé au contrôle d'un élément on ajoute le property binding avec `[ngModel]`
- ▶ Il est même possible de faire du two-ways binding avec `[(ngModel)]` en cas de besoin.

GROUPING FORM

- ▶ Afin de grouper l'ensemble des contrôles (propriétés/champs) d'un formulaire, on peut utiliser la technique du « grouping form controls ».
- ▶ Il suffit d'ajouter la directive `ngModelGroup` dans la `div` qui englobe les propriétés à grouper.
- ▶ Afin d'accéder à cet objet vous pouvez le référencer localement en utilisant le mot clé `ngModelGroup`

```
<div
  ngModelGroup= "user"
  userData= "ngModelGroup"
>
```

APPLICATION

- ▶ Grouper les données de votre utilisateur dans un `ngModelGroup`
- ▶ Tester l'objet généré
- ▶ Essaye de voir s'il contient les mêmes classes qu'un contrôle simple, telle que `ng-dirty`, `ng-valid`.
- ▶ Ajouter un message d'erreur qui apparaît si votre groupe n'est pas valide.

RETOUR AU PROJET

- ▶ Ajouter dans votre cvTech un composant contenant un formulaire. Ce formulaire devra vous permettre d'ajouter un utilisateur.
- ▶ Après l'ajout, il faut rediriger l'utilisateur vers la liste des cvs.