# Cours Angular

2.3 | TD Login

Directives – Validation – Input - Output

## 1] Intégration d'un début de validation

- ***Vérification des données***

Classiquement pour tout formulaire, la vérification des données entrées par l'utilisateur s'impose. Dans notre formulaire, nous allons voir comment intégrer le test sur l'absence d'entrée et donc l'erreur résultante.

Les constructeurs des contrôles comme `FormControl` ou `FormGroup` acceptent un second paramètre déterminant une liste de fonctions de validation (`validators`). Note : cette façon de faire ne sera pas vu pour l'instant.

Pour l'instant, vous verrez la directive `ngClass` (https://angular.io/api/common/NgClass), permettant l'**ajustement conditionnel** d'un style CSS à un élément HTML.

**T1.1** Ajoutez au template du composant `ident`, au niveau du login le code (en gras) suivant :

```
…
  <div class="control">
    <p [ngClass]="{ 'has-error': isSubmitted && badLogin }">
      <input class="input is-success" formControlName="login" type="text"
placeholder="">
    </p>
    <div *ngIf="isSubmitted && badLogin" class="help-block">
      <div *ngIf="badLogin">Un login est demandé !</div>
    </div>
  </div>
…
```

Explications :

→ la directive `ngClass` s'active en cas « d'erreur » c'est-à-dire si le formulaire est soumis et qu'il y a un problème, matérialisée par le positionnement de deux valeurs booléennes à `true` ;
→ l'affichage de la phrase d'erreur est conditionnée par `ngIf` de façon identique.

**T1.2** Ajoutez ensuite le code nécessaire ; le mieux ici est de vous redonner tout le code au niveau de l'export du composant `ident.component.ts` :

```
…
export class IdentComponent implements OnInit {

  identForm: FormGroup;
  isSubmitted = false;
  badLogin = false;

  constructor() { }

  ngOnInit(): void {
    this.identForm = new FormGroup({
      login: new FormControl(''),
      passe: new FormControl('')
    });
```

```
  }

  get formControls() { return this.identForm.controls; }

  authentification() {
    this.isSubmitted = true;
    console.log("ident :" + this.identForm.value.login);

    if (this.identForm.value.login == '') {
      this.badLogin = true;
      return;
    }
  }
}
```

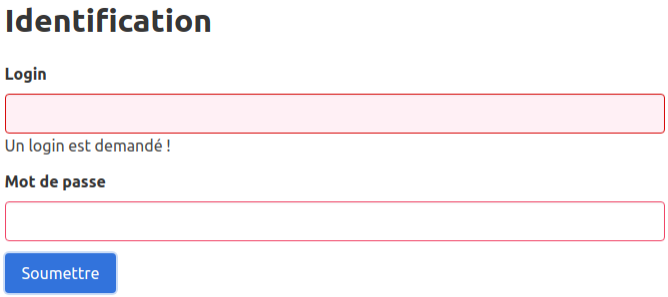
Explications :

→ la démarche consiste à déclarer deux variables booléennes et à les initialiser à `false` ;
→ `isSubmitted` porte sur le formulaire, ici nous n'avons qu'un seul test, mais nous pourrions tester un autre ou alors mettre une nouvelle condition (la longueur par exemple) ;
→ au déclenchement du `submit`, les variables sont positionnées en conséquence et on revient sur le formulaire en cas d'erreur.

**T1.3** Pour faire « style », mettez dans fichier de styles du composant `ident` :

```
.has-error input[type="text"],
.has-error input[type="password"] {
  border-color: rgb(216, 12, 12);
  background-color:lavenderblush;
  color: rgb(230, 14, 14);
}
```

Retour sur le formulaire, un envoi en ne mettant rien aboutit à l'écran :




**Exercice** Ajoutez le test d'un minimum de trois caractères pour le login…

## 2] Transmission des données entre composants

- ***Principes***

Comme avec Angular tout est composant, l'échange des données entre ceux-ci constitue une des bases d'une application Web. Un composant est dit **enfant d'un autre** – son parent – lorsque celui est dans le « sous-arbre » du second.

Nous souhaitons ici que le composant principal (`app.component`) puisse être informé lorsque un utilisateur est connecté. Pour cela on doit mettre en place une communication entre un de ses composants enfant

(identification) et lui-même. Le composant principal devra également informer le composant header (un de ses enfants) d'un changement d'état éventuel.

Concrètement on aura :

→ la réception de données du composant parent à l'enfant, ce sera le rôle du « **décorateur** » `@Input`
→ la transmission de données du composant enfant au parent, ce sera le rôle du « **décorateur** » `@Output`

Plus tard, nous verrons qu'il sera plus « professionnel » de passer par la **création d'un service**.

- ***L'émission de données de l'enfant vers le parent***

Il faut traiter l'évènement au niveau du formulaire. Cela implique :

- les importations nécessaires
- l'initialisation de la variable suivant un type spécial : l'**émission d'un évènement**
- l'émission proprement dite vers le parent

**T2.1** Modifications du code `ident.component.ts` :

Pour les importations, cela concerne la propriété et « décorateur » (terme Angular) **Output** et la méthode `EventEmitter` pour la gestion de l'événement :

```
import { Component, Output, EventEmitter, OnInit } from '@angular/core';
```

Pour l'initialisation dans l'export de la variable `leLogin` (à la suite des autres) :

```
  @Output() leLogin = new EventEmitter<string>();
```

Pour l'émission dans la méthode `authentification()` :

```
  authentification() {
    this.isSubmitted = true;
    console.log("ident :" + this.identForm.value.login);

    if (this.identForm.value.login == '') {
      this.badLogin = true;
      return;
    } else {
      this.leLogin.emit(this.identForm.value.login);
    }
  }
```

Notez qu'en cas d'erreur dans l'émission, Angular ne montrera aucune erreur. L'émission ne se fera pas, c'est tout…

- ***La réception de la valeur par le parent***

Cette fois, le déroulement des modifications se passe dans `app.component.ts` et concernera :

- la déclaration/initialisation de la variable pour la réception
- la méthode récupérant la valeur de l'émission par l'enfant comme paramètre
- une méthode testant le changement d'état de la variable pour voir si une connexion est faite

**T2.2** Modifications du code dans la partie export d'`app.component.ts` car nous sommes cette fois au niveau du parent:

```
…
  login:string = '';

  onLoginAdded(event : string) {
    this.login = event;
  }

  getConnect() {
    return this.login != '';
  }
```

- ***L'émission de données du parent vers l'enfant***

La correspondance entre l'émission chez le composant enfant et le parent s'inscrit dans le template du composant parent au moment de l'inclusion avec la mention de la méthode utilisée dans le code du parent.

**T2.3** Modifiez le template `app.component.html` pour cela :

```
<ng-template #noconnect>
  <app-ident (leLogin)="onLoginAdded($event)"></app-ident>
</ng-template>
```

Récupérer le login de l'utilisateur, c'est bien, mais il faudra l'afficher dans la page d'accueil. Notez d'ailleurs que l'on ne s'occupe pas pour l'instant de la validation du mot de passe…

**T2.4** Modifiez à nouveau le template `app.component.html` mais cette fois au niveau du composant `header` car c'est là que nous allons utiliser le nom de l'utilisateur :

```
<app-header [login]="login"></app-header>
```

Le décorateur `Input` va être utilisé chez le composant enfant, ici **dans le code ci-dessous** (à vous de trouver le bon fichier…) du `header`.

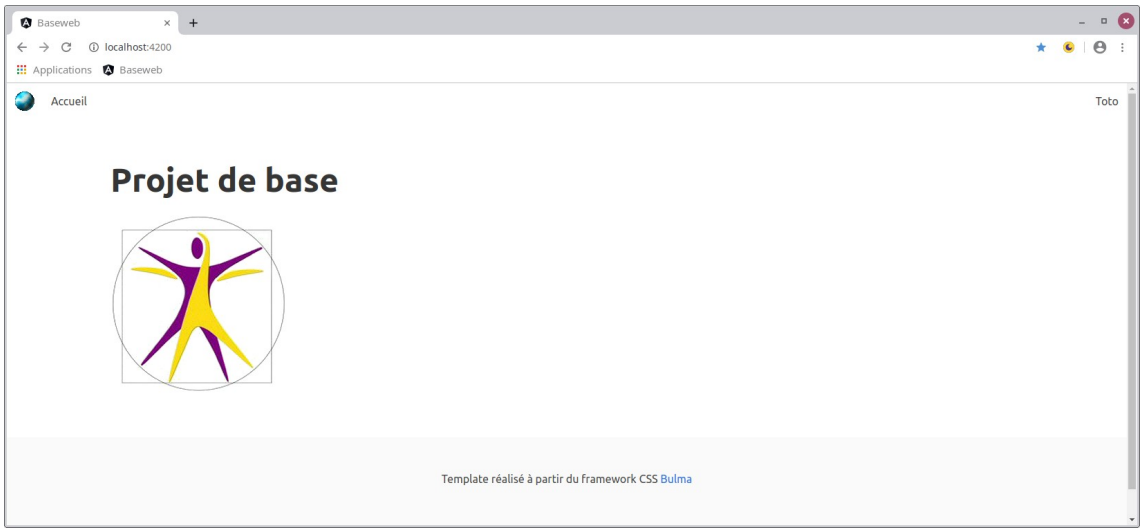
**T2.5** Ajoutez l'importation du décorateur et sa déclaration dans l'export (l'initiation sera effectué au moment de l'inclusion) :

```
import { Component, Input, OnInit } from '@angular/core';
…

export class HeaderComponent implements OnInit {

@Input() login!: string;
```

**T2.6** La suite se passe dans le template du header ou dans premier temps nous allons remplacer le texte par la variable `login` suivant la méthode « *string interpolation* » avec la classique double accolades ou notation moteur de template « Mustache » :

```
…
  <div class="navbar-end">
    <div class="navbar-item">
      <div class="buttons">
        <p>{{login}}</p>
      </div>
    </div>
  </div>
</nav>
```

Cela doit aboutir à ceci si l'on rentre `Toto` comme nom d'utilisateur :




**Exercice** Faites en sorte que l'intitulé « non connecté » apparaisse tant que l'utilisateur ne s'est pas correctement logué (la solution est très simple…).