2.2 | TD Login

Directives structurelles – DOM – Gestion des données – Formulaire réactif

1] Intégration d'un nouveau composant métier

- ***À nouvelle structure, nouveau composant***

Un changement de structure important est nécessaire. Il faut maintenant créer un nouveau composant, **celui qui sera chargé de l'identification** en n'oubliant pas de gérer le retour à la page d'accueil.

T1.1 Créez le composant `ident` par la méthode vue dans le précédent TD.

T1.2 Remplacez les lignes entre les balises du template dans `app.component.html` la prise en compte de ce nouveau composant en cas de non authentification.

```
…
<ng-template #noconnect>
  <app-ident></app-ident>
</ng-template>
…
```

Ce qui nous donne en retour cette fois :

2] Première approche du formulaire

- ***La gestion dynamique du DOM par Angular***

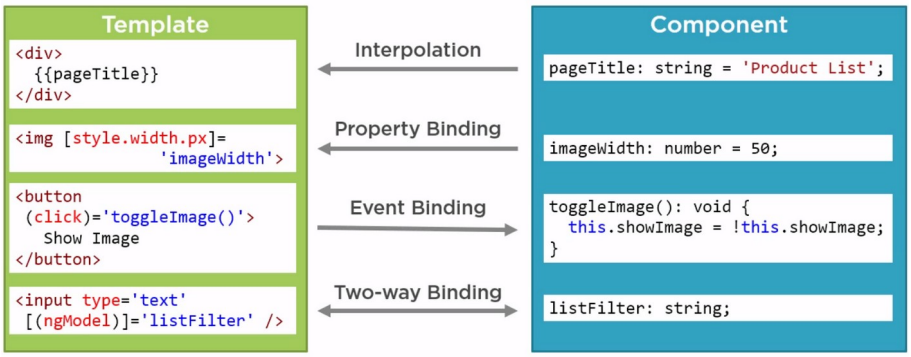
Angular gère les éléments HTML et leurs propriétés de la page dynamiquement par le DOM. Cette façon de faire lie (**to bind** : lier) les données entre le code de l'application (TypeScript) et sa représentation (HTML) suivant le principe du « *data-binding* ».

Concrètement, nous avons deux directions :

→ celle qui **affiche les informations du code vers le navigateur** ; on aura comme méthode « *string interpolation* » et « *property binding* »
→ celle qui **récupère les évènements du template devant être gérés par le code** (typiquement un clic sur un bouton) ; on aura comme méthode « *event binding* »

On parlera aussi de méthode « *two-way binding* » dans le cas d'une communication bilatérale (typiquement soumission et vérification des données d'un formulaire).

La synthèse complète des communications peut se résumer ainsi par un graphique (source : http://www.ai-softwarellc.com/blog/post/angular-communication-part-1/2090) :



- ***Le formulaire d'identification***

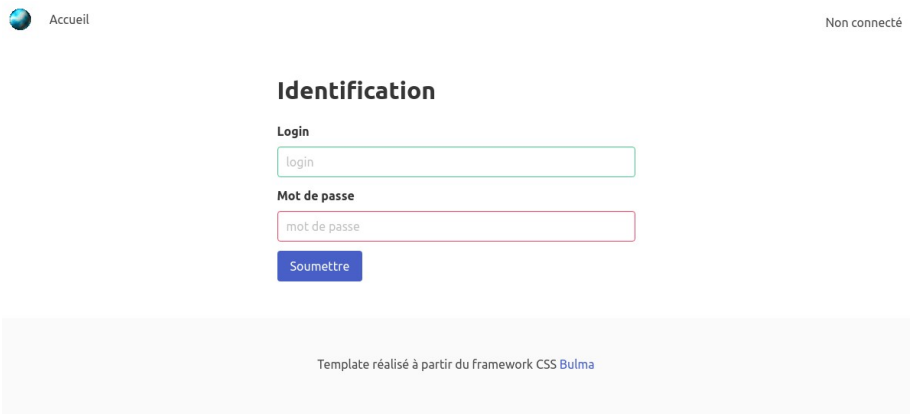
T2.1 Intégrez le code complet du formulaire dans le template `ident.component.html` à la place du code existant :

```
<section class="section">
 <div class="container">
  <div class="columns is-centered">
   <div class="column is-half">
    <h3 class="title is-3">Identification</h3>

    <form>
     <div class="field">
      <label class="label">Login</label>
      <div class="control">
       <input class="input is-success" type="text" placeholder="login" value="">
      </div>
     </div>
     <div class="field">
      <label class="label">Mot de passe</label>
      <div class="control">
       <input class="input is-danger" type="password" placeholder="mot de passe"
value="">
      </div>
     </div>
     <div class="field is-grouped">
      <div class="control">
       <button class="button is-link">Soumettre</button>
      </div>
     </div>
    </form>

   </div>
  </div>
 </div>
</section>
```

Cela devient déjà un plus cohérent dans le navigateur :



- ***Formulaire réactif et modèle de formulaire***

La mise en place de la problématique d'un formulaire sous Angular passe par la notion de « formulaire réactif » grâce au module `ReactiveFormsModule` (méthode « *event binding* »).

Pour activer les syntaxes de formulaire réactives pour Angular, il faut donc importer ce module, elle se fait **manuellement** dans le fichier `app.module.ts`.

T2.2 Importez le module dans le fichier après les deux premiers modules et avant les composants :

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

Et avec la déclaration de cette importation toujours dans le même fichier dans la partie `imports` (n'oubliez pas la virgule) :

```
  imports: [
    …,
    ReactiveFormsModule
```

Pour qu'Angular puisse gérer les champs (et évènements) du formulaire, chaque élément de celui-ci doit avoir une correspondance dans un modèle. En clair : si on met un élément texte, on utilise le modèle de type « text ».

- ***Correspondances***

T2.3 Changez le code du formulaire dans le template en y ajoutant les éléments en gras ; notez aussi l'ajout des identifiants pour les entrées avec la directive `formControlName` :

```
…
  <form [formGroup]="identForm" (ngSubmit)="authentification()">
    <div class="field">
      <label class="label">Login</label>
      <div class="control">
        <input class="input is-success" formControlName="login" type="text"
placeholder="login" value="">
      </div>
    </div>
    <div class="field">
      <label class="label">Mot de passe</label>
      <div class="control">
        <input class="input is-danger" formControlName="passe" type="password" placehol-
der="mot de passe" value="">
      </div>
    </div>
```

```
    <div class="field is-grouped">
    <div class="control">
      <button class="button is-link">Soumettre</button>
    </div>
  </div>
</form>
```

Dans ce formulaire, la **correspondance** sera :

- ✗ balise `<form>` → instance de `FormGroup`
- ✗ balise `<input>` → instance de `FormControl`
- ✗ attribut `submit` du bouton → relié à l'évènement `ngSubmit` dans la balise `<form>` (et non dans le bouton)

La directive `[formGroup]` se note entre crochets car c'est une **expression Angular** (voir plus bas), alors que `formControlName` ne l'est pas, étant une chaîne de caractères, donc sans crochets.

Cet ajout provoque des erreurs, car des choses sont à faire dans le code du composant.

Deux méthodes pour le faire : avec le service `FormBuilder` d'Angular (https://angular.io/guide/reactive-forms) ou manuellement. Cette dernière solution (plus simple) sera utilisée pour l'instant ici.

T2.4 Appliquez les changements dans le code `ident.component.ts` :

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

@Component({
  selector: 'app-ident',
  templateUrl: './ident.component.html',
  styleUrls: ['./ident.component.css']
})

export class IdentComponent implements OnInit {

  identForm!: FormGroup;

  constructor() { }

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

  authentification() {
    console.log('Données du formulaire...', this.identForm.value);
  }
}
```

En résumé :

→ on a lié la balise `<form>` à la variable `identForm` grâce à la directive `formGroup`
→ on a fait de même pour les champs mais cette fois-ci avec la directive `formControlName`
→ à l'initialisation du composant, la variable `identForm` est définie avec ses deux champs
→ on est obligé de mettre `!:` devant `identForm` afin « d'autoriser » une valeur `null` (pas d'initialisation)
→ `ngOnInit` est une instruction Angular appelée lorsque le composant est initié
→ on utilise la méthode `authentification()` pour répondre au `submit` du formulaire par l'évènement `ngSubmit`

Si l'on teste des entrées dans le formulaire, les valeurs apparaissent dans la console comme définies dans la méthode `authentification()` :