II-DataBinding

LA LIAISON DE DONNÉES OU DATABINDING

Angular permet de gérer le DOM de manière dynamique.

Pour cela, nous utilisons la **liaison de données (ou "databinding")**

Le databinding, c'est la communication entre votre code TypeScript et le template HTML qui est montré à l'utilisateur. Cette communication est divisée en deux directions :

1. Les informations venant de votre code qui doivent être affichées dans le navigateur;
   Les deux principales méthodes pour cela sont le "**string interpolation**" et le "**property binding**" ;
2. Les informations venant du template qui doivent être gérées par le code :
   Exemple: l'utilisateur a rempli un formulaire ou cliqué sur un bouton, et il faut réagir et gérer ces événements. On parlera de "**event binding**" pour cela.

Il existe également des situations comme les formulaires, par exemple, où l'on voudra une *communication à double sens* : on parle donc de "**two-way binding**".

I - String interpolation - {{ ... }}

Ajoute le code suivant dans tes fichiers

> app.component.html

app.component.html

src > app > app.component.html > div.container > div.row > div.col-xs-12 > h2

```
<div class="container">
  <div class="row">
    <div class="col-xs-12">
      <h2>Liste</h2>
      <ul class="list-group">
        <app-mon-premier-component></app-mon-premier-component>
        <app-mon-premier-component></app-mon-premier-component>
        <app-mon-premier-component></app-mon-premier-component>
      </ul>
    </div>
  </div>
</div>
```

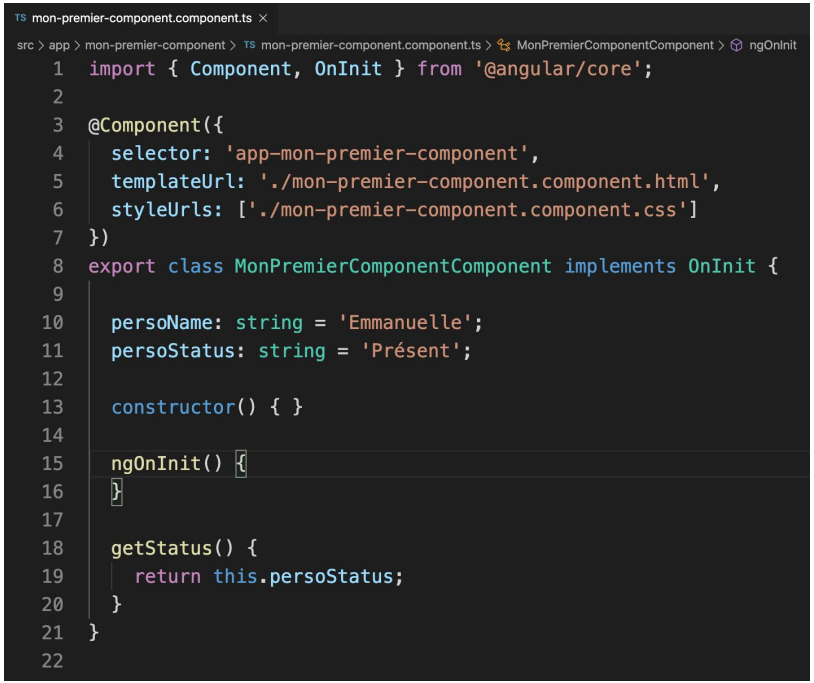
> mon-premier-component.component.html

mon-premier-component.component.html

src > app > mon-premier-component > mon-premier-component.component.html > ...

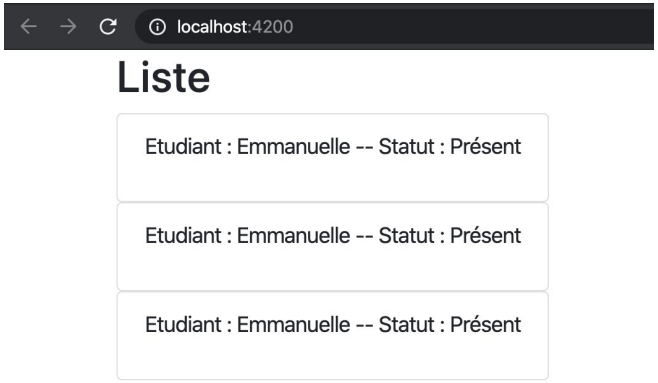
```
<li class="list-group-item">
    <p>Etudiant : {{ persoName }} -- Statut : {{ getStatus() }}</p>
</li>
```

> mon-premier-component.component.ts :



Rque: La déclaration de type ici (les deux-points suivis du type `string` n'est pas obligatoire, car TypeScript déduit automatiquement le type d'une variable lorsque vous l'instanciez avec une valeur. J'ai simplement inclus la déclaration de type pour montrer la syntaxe TypeScript

Résultat :



Dans **`mon-premier-component.component.html`**, nous avons utiliser la syntaxe pour l'interpolation : **les doubles accolades `{{ }}` .**

Exemple : **`{{ persoName }}`**

Ce qui se trouve entre les doubles accolades correspond à l'expression TypeScript que nous voulons afficher.

Cette expression est une variable qui se trouve dans le fichier **`mon-premier-component.component.ts`**

```
persoName: string = 'Emmanuelle';
```

On peut utiliser toute expression TypeScript valable pour l'interpolation. Pour exemple, nous avons utilisé une méthode pour notre second interpolation `{{ getStatus() }}`

II - Property binding

La liaison par propriété ou "property binding" est une autre façon de créer de la communication dynamique entre votre TypeScript et votre template : plutôt qu'afficher simplement le contenu d'une variable, vous pouvez modifier dynamiquement les propriétés d'un élément du DOM en fonction de données dans votre TypeScript.

> app.component.html

src > app > app.component.html > div.container > div.row > div.col-xs-12

```html
<div class="container">
  <div class="row">
    <div class="col-xs-12">
      <h2>Liste</h2>
      <ul class="list-group">
        <app-mon-premier-component></app-mon-premier-component>
        <app-mon-premier-component></app-mon-premier-component>
        <app-mon-premier-component></app-mon-premier-component>
      </ul>
      <input type="checkbox" [checked]="isAuth" >Utilisateur connecté
    </div>
  </div>
</div>
```

> app.component.ts

src > app > app.component.ts > ...

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'mon-premier-projet';
  isAuth = true;
}
```

Résultat:

La propriété `checked` permet de cocher la case d'un input de type checkbox.

Afin de lier cette propriété au TypeScript, il faut le mettre entre crochets `[]` et l'associer à la variable ainsi :

```
<input[checked]="isAuth">Utilisateur connecté
```

III - Event binding - Gestion simple d'événement avec (click)

A présent, nous allons voir comment réagir dans votre code TypeScript aux événements venant du template HTML.

> app.component.html

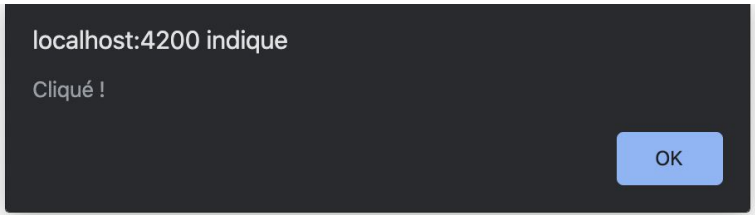
```
<div class="container">
  <div class="row">
    <div class="col-xs-12">
      <h2>Liste</h2>
      <ul class="list-group">
        <app-mon-premier-component></app-mon-premier-component>
        <app-mon-premier-component></app-mon-premier-component>
        <app-mon-premier-component></app-mon-premier-component>
      </ul>
      <input type="checkbox" [checked]="isAuth" (click)="onClique()">Utilisateur connecté
    </div>
  </div>
</div>
```

> app.component.ts

src > app > TS app.component.ts > AppComponent

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'mon-premier-projet';
  isAuth = true;
  onClique() {
    alert('Cliqué !');
  }
}
```

Résultat :



Remarque: nous avons utilisé les parenthèses `()` pour créer une liaison à un événement.
Exemple: (click) = “”;

Pour les méthodes liées aux évènements, il existe une **convention de nomenclature** :
"on" + le nom de l'événement. Cela permet, entre autres, de suivre plus facilement l'exécution des méthodes lorsque l'application devient plus complexe.

III - Two-way binding - [(ngModel)]

La liaison à double sens (ou two-way binding) utilise la liaison par propriété et la liaison par événement en même temps ; on l'utilise, par exemple, pour les formulaires, afin de pouvoir déclarer et récupérer le contenu des champs, entre autres.

Pour pouvoir utiliser le two-way binding, il vous faut importer `FormsModule` depuis @angular/forms dans votre application. Vous pouvez accomplir cela en l'ajoutant à l'array `imports` de votre `AppModule` (sans oublier d'ajouter le statement `import` correspondant en haut du fichier) :

src > app > TS app.module.ts > AppModule

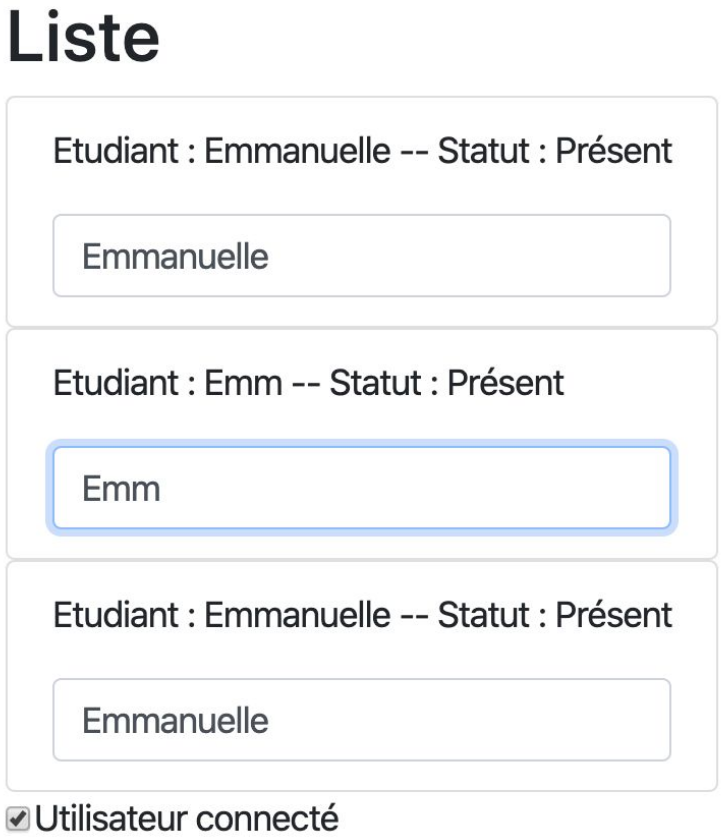
```ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { MonPremierComponentComponent } from './mon-premier-component/mon-premier-component.component';
import { FormsModule } from '@angular/forms';

@NgModule({
  declarations: [
    AppComponent,
    MonPremierComponentComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Le two-way binding emploie le mélange des syntaxes de property binding et d'event binding : des crochets et des parenthèses `[()]` . Pour une première démonstration, ajoutez un `<input>` dans votre template `mon-premier-component.component.html` et liez-le à la variable `persoName` en utilisant la directive `ngModel` :

Dans votre template, vous verrez un `<input>` par case. Le nom de la personne est déjà indiqué dedans, et si vous le modifiez, le contenu du `<p>` est modifié avec. Ainsi vous voyez également que chaque instance du composant `<app-mon-premier-component></app-mon-premier-component>` est entièrement indépendante une fois créée : le fait d'en modifier une ne change rien aux autres.

Ce concept est très important, et il s'agit de l'une des plus grandes utilités d'Angular !

Résultat :


Liste
Etudiant : Emmanuelle -- Statut : Présent
Emmanuelle
Etudiant : Emm -- Statut : Présent
Emm
Etudiant : Emmanuelle -- Statut : Présent
Emmanuelle
Utilisateur connecté

V - Propriétés personnalisées

Il est possible de créer des propriétés personnalisées dans un composant afin de pouvoir lui transmettre des données depuis l'extérieur.

Pour notre application, il serait intéressant de faire en sorte que chaque instance ait un nom différent qu'on puisse régler depuis l'extérieur du code. Pour ce faire, il faut utiliser le décorateur `@Input()` en remplaçant la déclaration de la variable `persoName` :

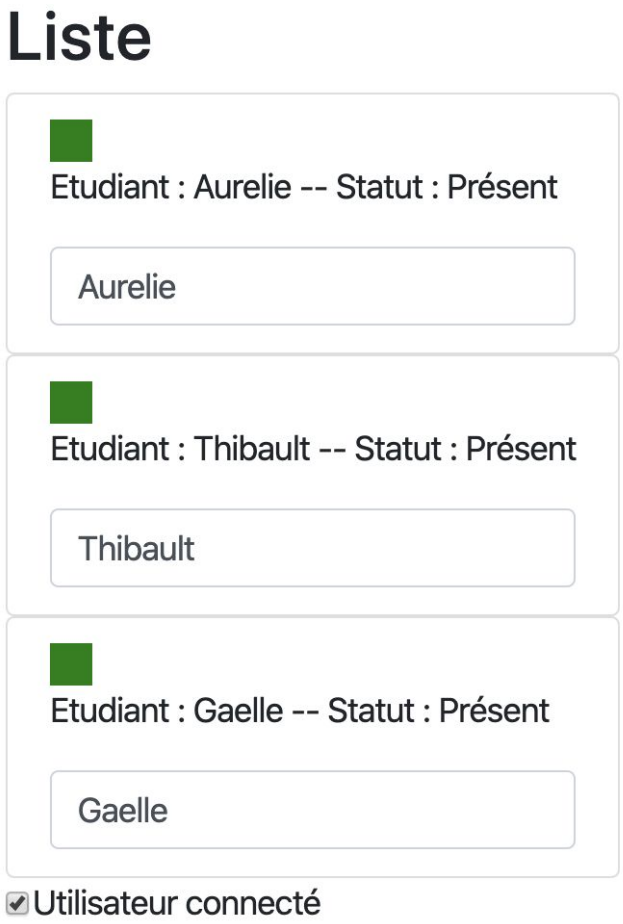
Remarque: Il nous faut aussi penser à importer `Input` depuis `@angular/core` en haut du fichier !

src > app > mon-premier-component > TS mon-premier-component.component.ts > MonPremierComponentComponent

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

@Component({
  selector: 'app-mon-premier-component',
  templateUrl: './mon-premier-component.component.html',
  styleUrls: ['./mon-premier-component.component.css']
})
export class MonPremierComponentComponent implements OnInit {

  @Input() persoName: string;

  persoStatus: string = 'Présent';

  constructor() {

  }

  ngOnInit() {
  }

  getStatus() {
    return this.persoStatus;
  }
}
```

Ce décorateur, en effet, crée une propriété `persoName` qu'on peut fixer depuis la balise `<app-mon-premier-component>` :

Résultat:


Liste
Etudiant : Aurelie -- Statut : Présent
Aurelie
Etudiant : Thibault -- Statut : Présent
Thibault
Etudiant : Gaelle -- Statut : Présent
Gaelle
Utilisateur connecté

AppComponent
=> regroupe les fichiers:
- app.component.html
- app.component.css
- app.component.ts

HTML
```
<app-mon-premier-component persoName="Aurélie"></app-mon-premier-component>
```

Transfert de données du composant parent (*AppComponent*) vers le composant enfant (*MonPremierComponent*)

TypeScript
```
@Input() persoName: string;
```

MonPremierComponent
=> regroupe les fichiers:
- mon-premier-component.component.html
- mon-premier-component..component.css
- mon-premier-component..component.ts

HTML
```
<p>Etudiant : {{ persoName}} </p>
```

Il est à l'intérieur du composant AppComponent

Cette première méthode est intéressante mais ce serait encore plus dynamique de pouvoir passer des variables depuis `AppComponent` pour nommer nos cases (on peut imaginer une autre partie de l'application qui récupérerait ces noms depuis un serveur, par exemple).

src > app > app.component.html > div.container

```html
<div class="container">
  <div class="row">
    <div class="col-xs-12">
      <h2>Liste</h2>
      <ul class="list-group">
        <app-mon-premier-component [persoName] = "persoOne"></app-mon-premier-component>
        <app-mon-premier-component [persoName] = "persoTwo"></app-mon-premier-component>
        <app-mon-premier-component [persoName] = "persoThree"></app-mon-premier-component>

      </ul>
      <input type="checkbox" [checked]="isAuth" (click)="onClique()">Utilisateur connecté
    </div>
  </div>
</div>
```

Résultat:


Liste
Etudiant : Jessica -- Statut :
Jessica
Etudiant : Enzo -- Statut :
Enzo
Etudiant : Jean -- Statut :
Jean
Utilisateur connecté

A présent, nous allons créer une propriété pour régler le status :

src > app > mon-premier-component > mon-premier-component.component.ts > MonPremierComponentComponent > persoStatus

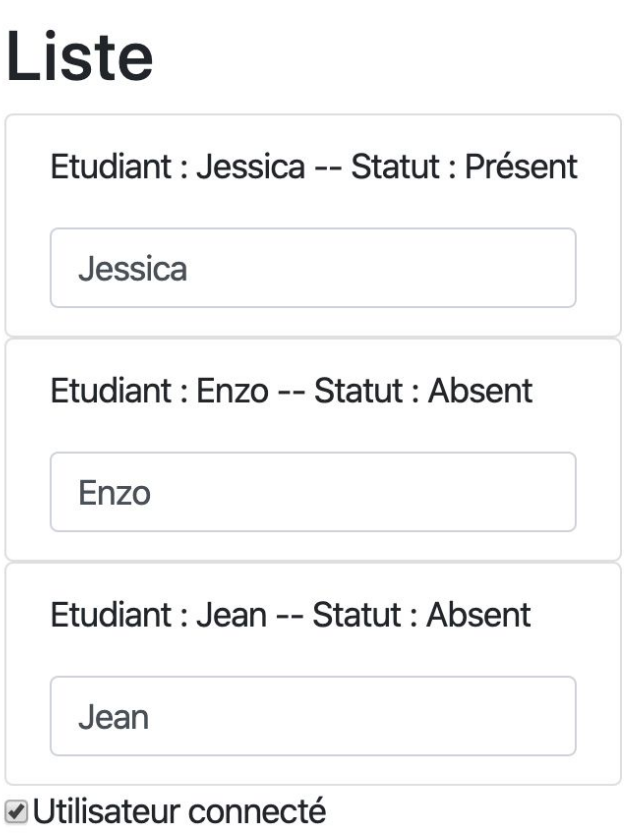
```ts
import { Component, Input, OnInit } from '@angular/core';

@Component({
  selector: 'app-mon-premier-component',
  templateUrl: './mon-premier-component.component.html',
  styleUrls: ['./mon-premier-component.component.css']
})
export class MonPremierComponentComponent implements OnInit {

  @Input() persoName: string;
  @Input() persoStatus: string;

  constructor() {

  }

  ngOnInit() {
  }

  getStatus() {
    return this.persoStatus;
  }
}
```

src > app > app.component.html > div.container > div.row > div.col-xs-12 > ul.list-group > app-mon-premier-component

```
1  div class="container">
2   <div class="row">
3     <div class="col-xs-12">
4       <h2>Liste</h2>
5       <ul class="list-group">
6         <app-mon-premier-component [personName] = "persoOne" [persoStatus] = "'Présent'"></app-mon-premier-component>
7         <app-mon-premier-component [personName] = "persoTwo" [persoStatus] = "'Absent'"></app-mon-premier-component>
8         <app-mon-premier-component [personName] = "persoThree" [persoStatus] = "'Absent'"></app-mon-premier-component>
9
10      </ul>
11      <input type="checkbox" [checked]="isAuth" (click)="onClique()">Utilisateur connecté
12    </div>
13  </div>
14 /div>
```

ATTENTION : Notez bien que si vous employez les crochets pour le property binding et que vous souhaitez y passer un string directement, il faut le mettre entre apostrophes, car entre les guillemets, il doit y avoir un statement de TypeScript valable. Si vous omettez les apostrophes, vous essayez d'y passer une variable nommée `Présent` ou `Absent` et l'application ne compilera pas

Résultat:


Liste
Etudiant : Jessica -- Statut : Présent
Jessica
Etudiant : Enzo -- Statut : Absent
Enzo
Etudiant : Jean -- Statut : Absent
Jean
Utilisateur connecté

V - Les Pipes

Le principe d'un pipe est de prendre en entrée une donnée et de la transformer comme vous le désirez dans le but de l'afficher à l'utilisateur

Syntaxe

```
<div>{{ persoName | lowercase }}</div> // le texte s'affichera en caractère minuscule
```

Il existe de nombreuses pipes : https://angular.io/api?type=pipe

Pour plus de détails sur les pipes voir :
https://angular.io/guide/pipes
https://guide-angular.wishtack.io/angular/pipes