

Cours 2 : Frontend



Web — Applications web et mobile

Christophe Gonzales

Qu'est-ce qu'Angular

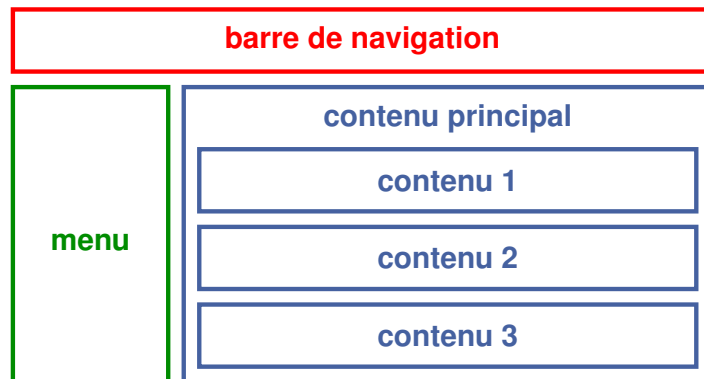
- ▶ Framework pour construire des applications clientes
⇒ front-end
- ▶ Structure l'application
⇒ simplifie programmation/maintenance/débuggage
- ▶ Mise en place de tests simple
- ▶ Utilise TypeScript/Javascript, HTML, CSS

Cours 2 : Frontend Angular

2/57

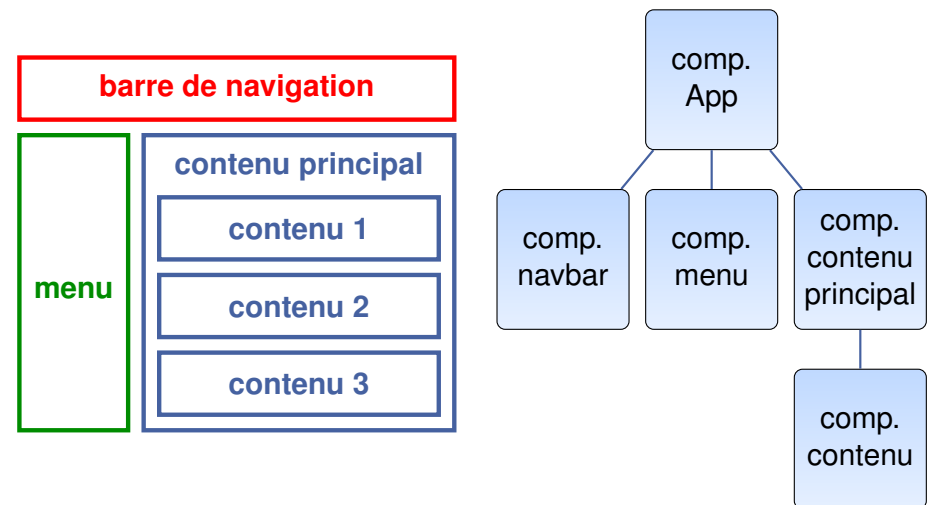
Structure d'une application Angular

Affichage de la page web :



- ▶ Affichage ⇒ structure
- ▶ Chaque rectangle = **composant Angular**
- ▶ Intérêt des composants : réutilisables plusieurs fois
- ▶ Un composant peut en inclure d'autres

Logique de l'application : arbre de composants



⇒ permet de structurer facilement le code !

Cours 2 : Frontend Angular

3/57

Cours 2 : Frontend Angular

4/57

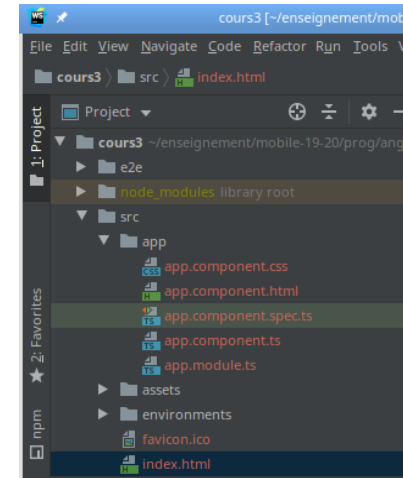
Contenu d'un composant Angular

Un composant Angular contient essentiellement :

- 1 un fichier TypeScript contenant :
 - ▶ les données du composant
 - ▶ la logique/le comportement du composant
 - 2 un fichier html
 - ▶ contenant le code HTML affiché par le browser
 - ▶ des instructions pour interagir avec le code TypeScript
 - 3 un fichier css contenant le style propre au composant
- ▶ Répertoire `src/app` contient les composants
- ▶ 1 composant principal appelé `app` ou `root`

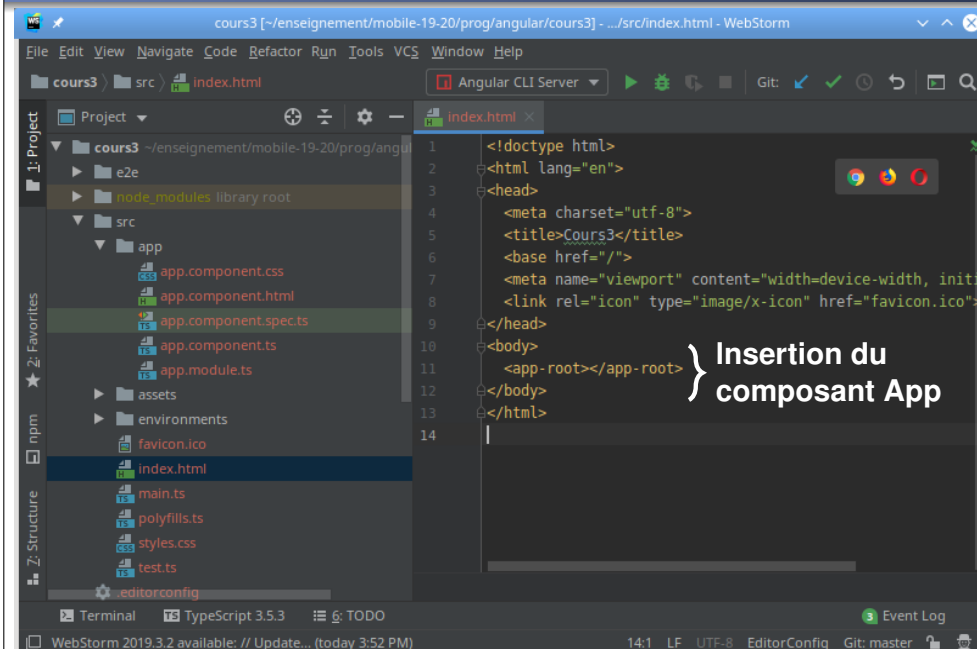
Génération d'un projet Angular

- ▶ `ng new mon-projet`
⇒ crée le composant `app` :



- ▶ Dans `src/app` :
- ▶ `app.component.ts` : code TypeScript
 - ▶ `app.component.spec.ts` : pour faire des tests
 - ▶ `app.component.html` : template HTML
- ▶ Dans `src` :
- ▶ `index.html` : point d'entrée de l'appli

Index.html



Fonctionnement d'un projet Angular

- 1 Créer les composants (et les modules)
- 2 Les insérer dans l'application via des balises dans les fichiers HTML
Exemple : `<app-root></app-root>`

- ▶ Pour compiler et « servir » votre application :
- ```
ng serve
```

# Le composant App et l'appli servie

The screenshot shows the WebStorm IDE with the file `app.component.html` open. The code contains the following HTML snippet:

```
<h1>Ceci est mon composant App</h1>
```

A browser window titled "Cours3 - Chromium" is displayed in the foreground, showing the rendered output of the component: "Ceci est mon composant App". The terminal at the bottom shows the command `ng serve` and the output indicating the application is running at `http://localhost:4200/webpack-dev-server/`.

# Le TypeScript du composant app

The screenshot shows the WebStorm IDE with the file `app.component.ts` open. The code defines the `AppComponent` class:

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

// décorateur qui indique à Angular que la classe TypeScript
// en dessous est le code d'un composant
@Component({
 // indique la balise HTML à utiliser pour insérer le
 // composant dans l'application
 selector: 'app-root',

 // indique où se trouve le code HTML du composant (celui à
 // insérer dans l'appli quand on insère le composant
 templateUrl: './app.component.html',

 // les styles propres au composant
 styleUrls: ['./app.component.css']
})
export class AppComponent {
 // un composant contient une classe TypeScript qui sert
 // à contenir ses données et sa logique
}
```

# Création d'un nouveau composant

- Utiliser la commande `ng generate component courses`
- ⇒ crée un répertoire `courses` et des fichiers spécifiques

The screenshot shows the WebStorm IDE with the `app.component.html` file open. The terminal at the bottom shows the command `ng generate component courses` and the output:

```
gonzales@shalmaneser:~/enseignement/mobile-19-20/prog/angular/cours3$ ng generate component courses
CREATE src/app/courses/courses.component.css (0 bytes)
CREATE src/app/courses/courses.component.html (22 bytes)
CREATE src/app/courses/courses.component.spec.ts (635 bytes)
CREATE src/app/courses/courses.component.ts (273 bytes)
UPDATE src/app/app.module.ts (479 bytes)
```

# TypeScript du nouveau composant

The screenshot shows the WebStorm IDE with the file `courses.component.ts` open. The code defines the `CoursesComponent` class:

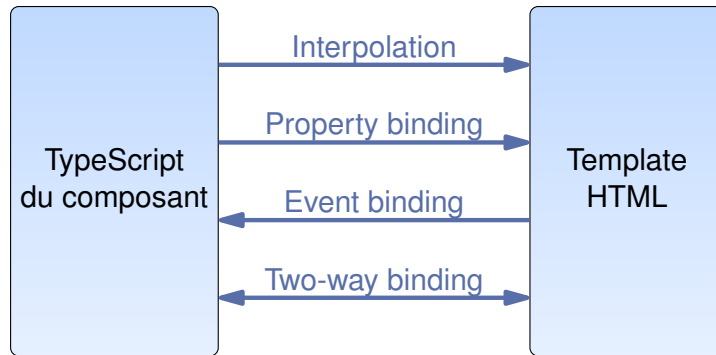
```
import { Component, OnInit } from '@angular/core';

@Component({
 selector: 'app-courses', // balise d'insertion
 templateUrl: './courses.component.html',
 styleUrls: ['./courses.component.css']
})
export class CoursesComponent implements OnInit {

 // dans le constructeur, on ne met que des directives
 // d'affichage. Le constructeur doit s'exécuter RAPIDEMENT
 // donc pas de récupération de données de BD ici
 constructor() { }

 // cette méthode est appelée après le constructeur. Elle
 // sert, notamment, à initialiser des champs en récupérant
 // des données de bases de données
 ngOnInit() {
 }
}
```

# Interactions entre le TypeScript et le HTML



# Interpolation

```
courses.component.ts: @Component({ selector: 'app-courses', templateUrl: './courses.component.html', styleUrls: ['./courses.component.css'] }) export class CoursesComponent implements OnInit { titre_: string = 'liste des cours'; constructor() {} } courses.component.html: <h1>Ceci est mon composan App</h1> <app-courses></app-courses> <app-courses></app-courses>
```

**Ceci est mon composan**

Interpolation : liste des cours et 7  
Interpolation : liste des cours et 7

# Interpolation avec des méthodes

```
courses.component.ts: @Component({ selector: 'app-courses', templateUrl: './courses.component.html', styleUrls: ['./courses.component.css'] }) export class CoursesComponent implements OnInit { titre = 'liste des cours'; constructor() {} } courses.component.html: <p> Interpolation bis : {{ getTitle() }}</p>
```

**Ceci est mon composant App**

Interpolation bis : liste des cours  
Interpolation bis : liste des cours

# Property binding

```
courses.component.ts: @Component({ selector: 'app-courses', templateUrl: './courses.component.html', styleUrls: ['./courses.component.css'] }) export class CoursesComponent implements OnInit { titre_: string = 'liste des cours'; ma_valeur_: string = 'valeur initiale'; constructor() {} } courses.component.html: <p> Interpolation : {{ titre }} et {{ 3 + 4 }}</p> <input type="text" [value]="ma_valeur" />
```

**Ceci est mon composant App**

Interpolation : liste des cours et 7  
valeur initiale

# Différence entre interpolation et property binding

- ▶ **Interpolation `{{}}` :**  
Permet de transférer des données du TypeScript n'importe où dans le template HTML  
Évalué à runtime !
- ▶ **Property binding `[]` :**  
Permet de mettre des valeurs dans les propriétés des éléments du DOM

Exemple intéressant : `[hidden]="valeur"`

# Event binding

```
courses.component.ts
export class CoursesComponent implements OnInit {
 titre = 'liste des cours';
 constructor() {}
 ngOnInit() {}
 getTitle() { return this.titre; }
 modifierTitle() {
 this.titre = 'nouveau titre';
 }
}
```

```
courses.component.html
<p> Interpolation évaluée à runtime : {{ getTitle() }}</p>
<input type="text" (click)="modifierTitle ()" />
```

**Ceci est mon composant**

Interpolation évaluée à runtime : nouveau titre

Interpolation évaluée à runtime : liste des cours

# Two-way binding (1/2)

```
courses.component.html
<p> Interpolation évaluée à runtime : {{ getTitle() }}</p>
<!-- two-way binding : [(ngModel)]
permet les interactions TypeScript - Template HTML
dans les 2 sens -->
<input type="text" [(ngModel)]="titre" />
```

**DevTools - localhost:4200/**

Uncaught Error: Template parse errors: Can't bind to 'ngModel' since it isn't a known property of 'input'. ("<input type="text" [(ngModel)]="titre" />")

# Two-way binding (2/2)

```
app.module.ts
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { CoursesComponent } from './courses/courses.component';
import { FormsModule } from '@angular/forms';

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

**Ceci est mon composant**

Interpolation évaluée à runtime : liste des

Interpolation évaluée à runtime : liste des cours

## Interactions TypeScript-template HTML en résumé

### Uniquement de TypeScript vers le template HTML :

- ▶ Property Binding : [propriété]="valeur"  
affecte des valeurs aux propriétés d'éléments du DOM
- ▶ Interpolation : {{ champ ou méthode }}  
à utiliser quand on ne peut pas faire de property binding

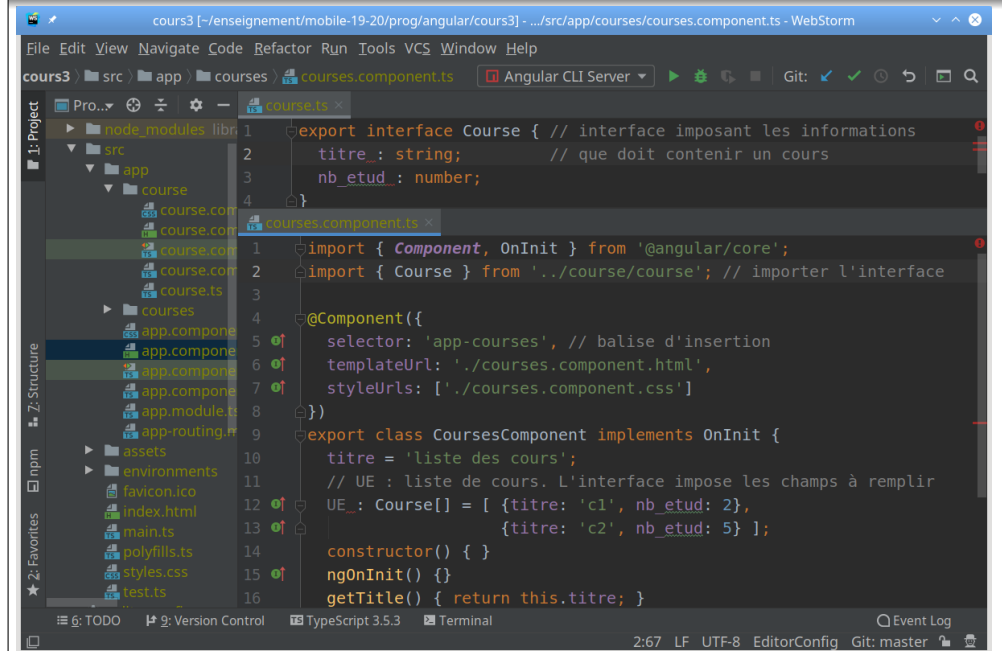
### Uniquement du template HTML vers le TypeScript :

- ▶ Event binding : (event)="méthode()"  
Appelle la méthode quand un événement du DOM arrive

### Dans les deux sens :

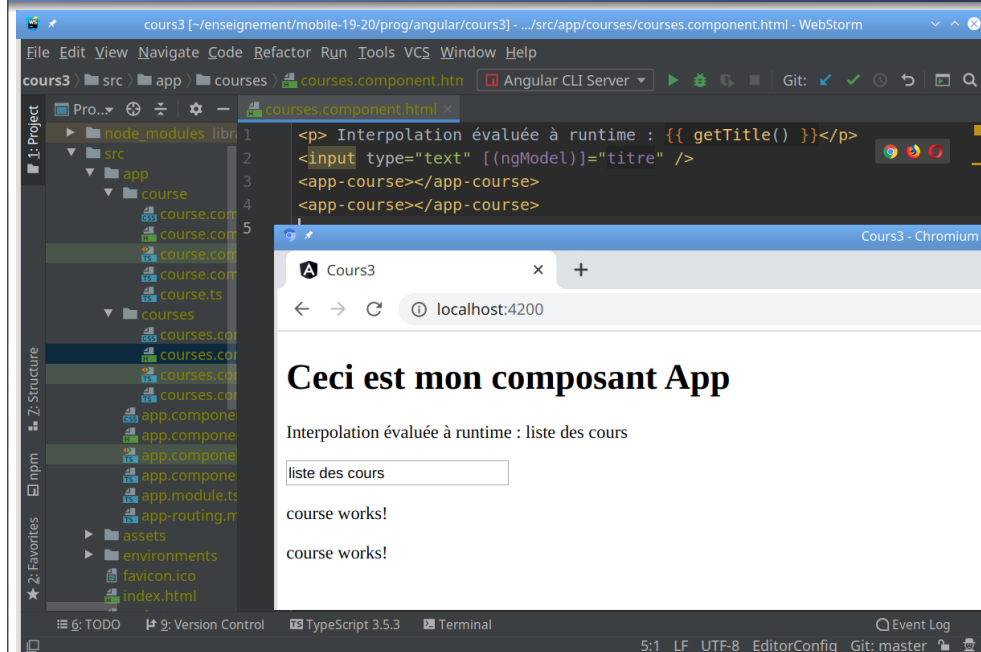
- ▶ Two-way binding : [(ngModel)]="valeur"  
⚠ ne pas oublier d'importer FormsModule dans app.module.ts

## Rajout d'un composant Course enfant de Courses (1/2)



```
1 export interface Course { // interface imposant les informations
2 titre : string; // que doit contenir un cours
3 nb_etud : number;
4 }
5
6 import { Component, OnInit } from '@angular/core';
7 import { Course } from '../course/course'; // importer l'interface
8
9 @Component({
10 selector: 'app-courses', // balise d'insertion
11 templateUrl: './courses.component.html',
12 styleUrls: ['./courses.component.css']
13 })
14 export class CoursesComponent implements OnInit {
15 titre = 'liste des cours';
16 // UE : liste de cours. L'interface impose les champs à remplir
17 UE : Course[] = [{titre: 'c1', nb_etud: 2},
18 {titre: 'c2', nb_etud: 5}];
19
20 constructor() { }
21 ngOnInit() { }
22 getTitle() { return this.titre; }
23 }
```

## Rajout d'un composant Course enfant de Courses (2/2)

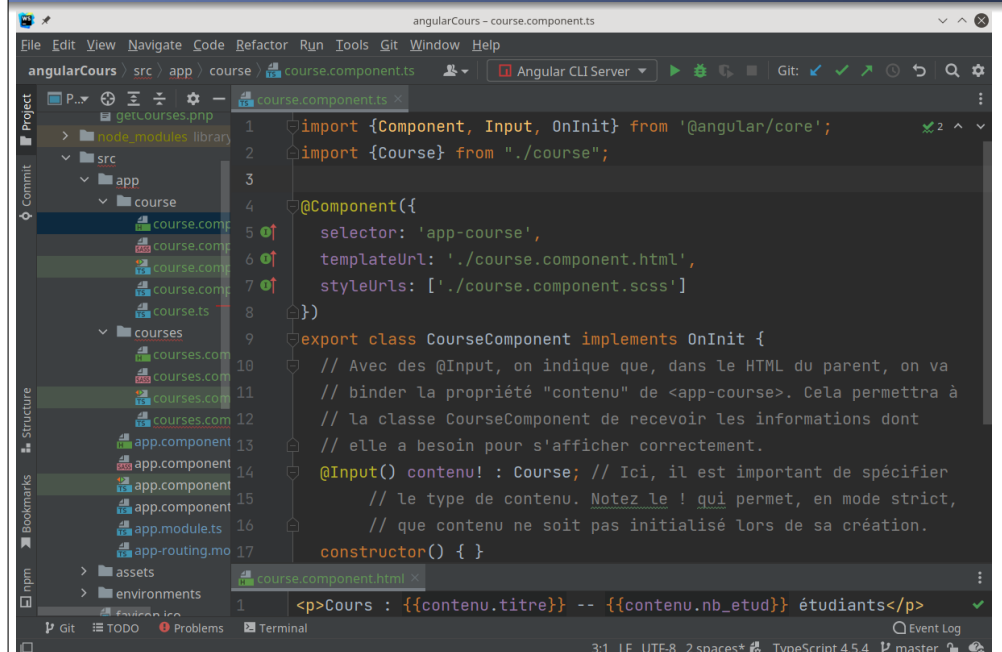


```
1 <p> Interpolation évaluée à runtime : {{ getTitle() }}</p>
2 <input type="text" [(ngModel)]="titre" />
3 <app-course></app-course>
4 <app-course></app-course>
```

Ceci est mon composant App

Interpolation évaluée à runtime : liste des cours

## Passer des paramètres au constructeur de l'enfant (1/2)



```
1 import { Component, Input, OnInit } from '@angular/core';
2 import { Course } from "../course";
3
4 @Component({
5 selector: 'app-course',
6 templateUrl: './course.component.html',
7 styleUrls: ['./course.component.scss']
8 })
9 export class CourseComponent implements OnInit {
10 // Avec des @Input, on indique que, dans le HTML du parent, on va
11 // binder la propriété "contenu" de <app-course>. Cela permettra à
12 // la classe CourseComponent de recevoir les informations dont
13 // elle a besoin pour s'afficher correctement.
14 @Input() contenu! : Course; // Ici, il est important de spécifier
15 // le type de contenu. Notez le ! qui permet, en mode strict,
16 // que contenu ne soit pas initialisé lors de sa création.
17 constructor() { }
18 }
```

## Passer des paramètres au constructeur de l'enfant (2/2)

The screenshot shows the HTML template of a child component in VS Code. The code includes an interpolation for the title and an input field for the title. The rendered output in the browser shows the title 'Ceci est mon composant App' and a list of courses: 'c1 -- 2 étudiants' and 'c2 -- 5 étudiants'.

```
<p> Interpolation évaluée à runtime : {{ getTitle() }}</p>
<input type="text" [(ngModel)]="titre" />
<!-- on passe les @Input par property binding -->
<app-course [contenu]="UE[0]"></app-course>
<app-course [contenu]="UE[1]"></app-course>
```

Cours 2 : Frontend Angular

25/57

## Interactions enfant → parent : préparation de l'enfant

The screenshot shows the HTML template of a parent component. It includes an interpolation for the title and an input field with an event binding for the 'change' event. The event binding calls the 'updateNb' method of the 'CourseComponent' class.

```
<p>Cours : {{ contenu.titre }} </p>
<!-- on rajoute un event binding sur l'évènement change
(en HTML : onchange="...") : chaque fois que l'on modifie
l'input, on va ainsi appeler la méthode updateNb de la classe
CourseComponent. L'idée est que cette méthode va envoyer des
informations sur les changements effectués au parent. -->
<input name="{{contenu.titre}}"
[(ngModel)]="contenu.nb_etud"
(change)="updateNb()" /> étudiants
```

Cours 2 : Frontend Angular

26/57

## Interactions enfant → parent : l'émetteur de l'enfant

The screenshot shows the TypeScript code for the child component. It includes imports for 'Component', 'EventEmitter', 'Input', 'OnInit', and 'Output' from '@angular/core', and 'Course' from './course'. The code defines the '@Component' decorator and the 'CoursesComponent' class, which implements 'OnInit' and has an '@Input()' and '@Output()' decorator.

```
import {Component, EventEmitter, Input, OnInit, Output} from '@angular/core';
import {Course} from './course';
@Component({
 selector: 'app-course',
 templateUrl: './course.component.html',
 styleUrls: ['./course.component.scss']
})
export class CoursesComponent implements OnInit {
 @Input() contenu : Course; // infos parent -> enfant
 @Output() newNb = new EventEmitter<number>(); // infos enfant -> parent
 // ici, newNb va être un événement que l'enfant va déclencher et le parent écouter
 lastNb! : number; // va servir à calculer le nombre que l'on émet vers le parent
 constructor() {}
 ngOnInit(): void { this.lastNb = this.contenu.nb_etud; }
 updateNb(): void { // fonction appelée quand on change l'input
 const nb = this.contenu.nb_etud - this.lastNb;
 this.lastNb = this.contenu.nb_etud;
 this.newNb.emit(nb);
 }
}
```

Cours 2 : Frontend Angular

27/57

## Interactions enfant → parent : la réception du parent

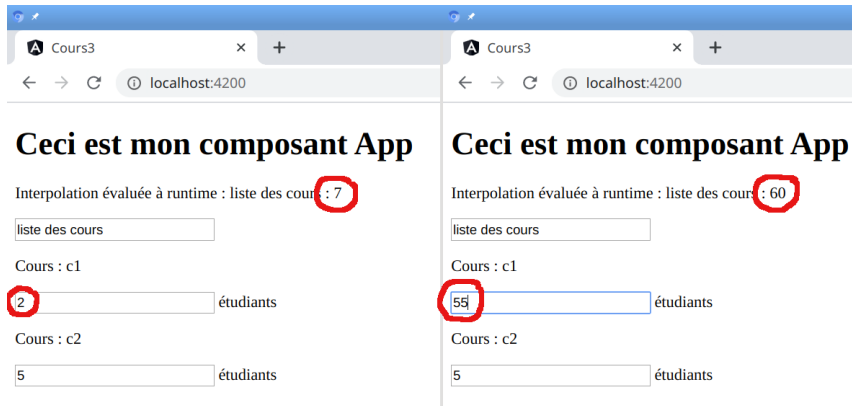
The screenshot shows the TypeScript code for the parent component. It includes the '@Component' decorator and the 'CoursesComponent' class, which implements 'OnInit' and has an '@Input()' and '@Output()' decorator. The code defines the 'CoursesComponent' class and its methods, including 'ngOnInit', 'getNbEtuds', and 'onNewNb'.

```
<p>Interpolation évaluée à runtime : {{getTitle()}} : {{nb_etuds}}</p>
<input type="text" [(ngModel)]="titre" />
<!-- par event binding, on écoute les événements newNb émis par l'enfant.
Le nombre transmis par l'enfant se trouve dans $event -->
<app-course [contenu]="UE[0]" (newNb)="onNewNb($event)"></app-course>
<app-course [contenu]="UE[1]" (newNb)="onNewNb($event)"></app-course>
export class CoursesComponent implements OnInit {
 titre = 'liste des cours';
 UE : Course[] = [{titre: 'c1', nb_etud: 2},
 {titre: 'c2', nb_etud: 5}];
 nb_etuds! : number;
 constructor() {}
 ngOnInit(): void { this.getNbEtuds(); }
 getNbEtuds(): void {
 this.nb_etuds = 0;
 for (const ue of this.UE) this.nb_etuds += ue.nb_etud;
 }
 onNewNb(delta: number) { this.nb_etuds += delta; } // callback quand newNb arrive
 getTitle(): string { return this.titre; }
}
```

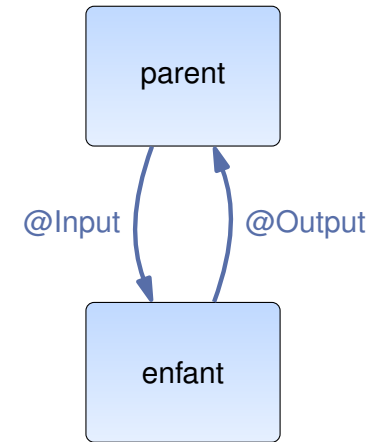
Cours 2 : Frontend Angular

28/57

## Interactions enfant → parent : le résultat



## Résumé des interactions enfant – parent



⚠ Il existe d'autres types d'interactions (@ViewChild, etc.)

## Listes de cours



Insérer une liste de cours dans l'appli :

- 1 Créer un composant **Cours**
- 2 Stocker la liste des cours dans le composant **Courses**
- 3 Dans le template de **Courses**, itérer l'insertion des cours avec la directive **\*ngFor**

⚠ Toutes les directives (**\*ngFor**, **\*ngIf**, etc.) débutent par une \*

⚠ Peut nécessiter l'inclusion de **CommonModule** dans **app.module.ts**

▶ **\*ngIf** : le composant est utilisé si et seulement si **ngIf=true**

## La directive \*ngFor

```
1 <p> Interpolation évaluée à runtime : {{ getTitle() }} : {{nb_etuds}}</p>
2 <input type="text" [(ngModel)]="titre" />
3 <!-- directive *ngFor : on parcourt tous les éléments de UE. Chaque
4 élément est stocké dans la variable cours. On peut alors utiliser
5 cette variable, notamment dans les property bindings -->
6 <app-course *ngFor="let cours of UE"
7 [contenu]="cours"
8 (newNb)="onNewNb($event)"></app-course>
9
10 <!-- attention : on ne peut pas appliquer 2 directives (par exemple ngIf et ngFor
11 au même composant. Il faut choisir : soit utiliser ngIf, soit ngFor.
12 Si l'on a besoin des 2, l'astuce est de créer un composant supplémentaire
13 auquel on appliquera, par exemple, le ngFor, et ce composant contiendra le
14 composant qui nous intéresse, auquel on appliquera le ngIf -->
15
```



## La directive \*ngIf

```
courses3 [-enseignement/mobile-19-20/prog/angular/courses3] - .../src/app/courses/courses.component.html - WebStorm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
courses3 src app courses courses.component.html Angular CLI Server Git:
1 <p> Interpolation évaluée à runtime : {{ getTitle() }} : {{nb_etuds}}</p>
2 <div>
3 <!-- Ici, la page web ne contiendra qu'un seul des paragraphes ci-dessous.
4 Si Courses contient moins de 10 étudiants, ce sera le 1er paragraphe,
5 sinon, ce sera le 2ème. Quand je dis que "la page web contiendra...",
6 je parle du DOM : le DOM ne contiendra effectivement qu'un <p>, et
7 non 2 <p> avec l'un des deux hidden. -->
8 <p *ngIf="nb_etuds < 10">peu d'étudiants</p>
9 <p *ngIf="nb_etuds >= 10">beaucoup d'étudiants</p>
10 </div>
11 <input type="text" [(ngModel)]="titre" />
12 <!-- directive *ngFor : on parcourt tous les éléments de UE. Chaque
13 élément est stocké dans la variable cours. On peut alors utiliser
14 cette variable, notamment dans les property bindings -->
15 <app-course *ngFor="let cours of UE"
16 [contenu]="cours"
17 (newNb)="onNewNb($event)"></app-course>
18
```

## La directive \*ngIf : le résultat

Cours3 localhost:4200 Ceci est mon composant App

Interpolation évaluée à runtime : liste des cours : 12

beaucoup d'étudiants

liste des cours

Cours : c1

2 étudiants

Cours : c2

5 étudiants

Cours : c3

5 étudiants

Cours3 localhost:4200 Ceci est mon composant App

Interpolation évaluée à runtime : liste des cours : 7

peu d'étudiants

liste des cours

Cours : c1

2 étudiants

Cours : c2

0 étudiants

Cours : c3

5 étudiants

## Les services : providers de données

- ▶ Actuellement : UE stockées en « dur » dans composant Courses
- ▶ Vraie application : UE récupérées d'un serveur

**A** : les services récupèrent les données

- ▶ Créer un service : `ng generate service nom_service`
- ▶ Utiliser le service comme n'importe quelle classe

## Anatomie d'un service

```
courses3 [-enseignement/mobile-19-20/prog/angular/courses3] - .../src/app/services/courses.service.ts - WebStorm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
courses3 src app services courses.service.ts Angular CLI Server Git:
1 import { Injectable } from '@angular/core';
2
3 @Injectable({
4 providedIn: 'root'
5 })
6 export class CoursesService {
7 constructor() { }
8
9 // ici, il suffit de créer une méthode qui renvoie les
10 // données dont a besoin le composant Courses. Ce dernier
11 // aura juste à ainsi juste à appeler la méthode pour
12 // obtenir sa liste d'UE
13 getCourses() {
14 return [{titre: 'c1', nb_etud: 2},
15 {titre: 'c2', nb_etud: 5},
16 {titre: 'c3', nb_etud: 5}];
17 }
18 }
19
```

## Exploitation du service dans le composant Courses

```
cours3 - courses.component.ts
File Edit View Navigate Code Refactor Run Tools Git Window Help
cours3 src app courses courses.component.ts CoursesCompon Angular CLI Server Git
1 import { Component, OnInit } from '@angular/core';
2 import { Course } from '../course/course';
3 // ici, on importe la classe du service pour pouvoir l'utiliser plus bas
4 import { CoursesService } from '../services/courses.service';
5
6 @Component({
7 selector: 'app-courses', // balise d'insertion
8 templateUrl: './courses.component.html',
9 styleUrls: ['./courses.component.css']
10 })
11 export class CoursesComponent implements OnInit {
12 titre = 'liste des cours';
13 UE!: Course[]; // ici, on n'initialise plus la liste d'UE
14 nb_etuds!: number;
15 constructor() {
16 // 1ère idée : dans le constructeur, on crée une instance du service et
17 // on appelle sa méthode getCourses pour pouvoir remplir notre tableau UE
18 const service = new CoursesService();
19 this.UE = service.getCourses();
20 }
21 ngOnInit() { this.getNbEtuds(); }
22 }
```

## Exploitation du service : bonnes et mauvaises idées

### Bonne idée :

- ▶ Faire en sorte que le constructeur connaisse le service

### Mauvaises idées :

- ▶ Demander au constructeur de créer l'instance
  - ▶ Plusieurs composants  $\implies$  plusieurs instances
  - ▶ Modif du constructeur du service  $\implies$  modif du composant
- ▶ Utiliser le service dans le constructeur
  - $\implies$  délais dans les affichages



### Bonne pratique des services :

- ▶ Dependency injection
- ▶ Utiliser le service dans méthode `ngOnInit`

## Dependency injection

```
cours3 - courses.component.ts
File Edit View Navigate Code Refactor Run Tools Git Window Help
cours3 src app courses courses.component.ts CoursesCompon ngOnInit() Angular CLI Server Git
6 @Component({
7 selector: 'app-courses', // balise d'insertion
8 templateUrl: './courses.component.html',
9 styleUrls: ['./courses.component.css']
10 })
11 export class CoursesComponent implements OnInit {
12 titre = 'liste des cours';
13 UE!: Course[]; // ici, on n'initialise plus la liste d'UE
14 nb_etuds!: number;
15 // dependency injection : on passe le service en paramètre du constructeur.
16 // Angular ne créera qu'une seule instance de CoursesService pour tout l'application
17 // et passera cette instance au constructeur
18 constructor(private service: CoursesService) {}
19 ngOnInit() {
20 this.UE = this.service.getCourses(); // c'est ngOnInit qui exploite le service
21 this.getNbEtuds();
22 }
23 getTitle() { return this.titre; }
24 getNbEtuds() {
25 this.nb_etuds = 0;
26 for (const ue of this.UE) this.nb_etuds += ue.nb_etud;
27 }
28 }
```

## Asynchronie et observables

- ▶ Service actuel : synchrone
  - $\implies$  `ngOnInit` doit attendre les données
- ▶ Services HTTP : asynchrones
  - $\implies$  évite de bloquer les affichages



### Utilisation de services asynchrones :

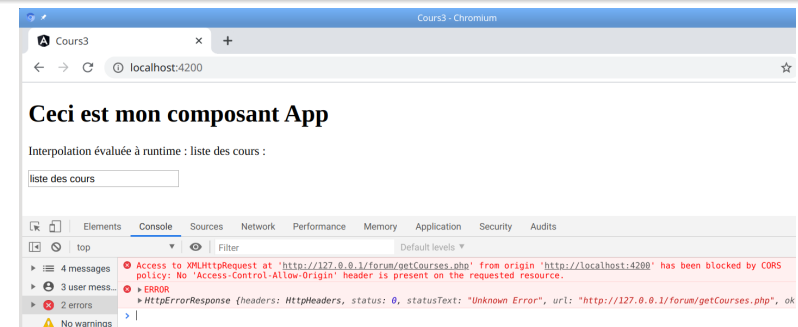
- 1 Le service retourne tout de suite un **Observable**
- 2 `ngOnInit` appelle le service et récupère l'observable
- 3 `ngOnInit` souscrit à l'observable en donnant une callback
- 4 `ngOnInit` continue son exécution
- 5 Les données arrivent  $\implies$  l'observable émet une valeur
  - $\implies$  la callback est appelée



## Service HTTP : 3 getCourses.php

```
1 <?php
2
3 header('Content-type:application/json;charset=utf8');
4 //header("Access-Control-Allow-Origin: *");
5
6 echo json_encode ([
7 ['titre' => 'c1', 'nb_etud' => 2],
8 ['titre' => 'c2', 'nb_etud' => 5],
9 ['titre' => 'c3', 'nb_etud' => 6]
10]);
11
12 ?>
```

## Service HTTP : Résultat



### ► Cross-Origin Resource Sharing (CORS) :

- Requête cross-origine : provient de localhost :4200  
accède à localhost :80

⇒ CORS refuse la requête

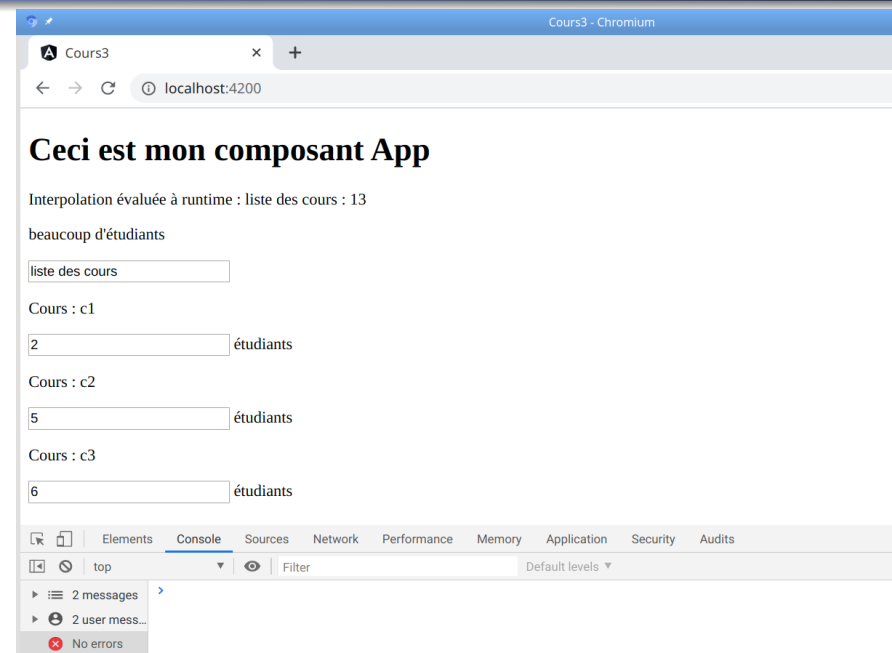
### ► Contre-mesure : dans getCourses.php, rajouter :

```
header("Access-Control-Allow-Origin: *");
```

## Service HTTP : 3 le bon getCourses.php

```
1 <?php
2
3 // le script doit renvoyer les données au format JSON
4 header('Content-type:application/json;charset=utf8');
5
6 // ici, on rajoute un header pour contourner le problème CORS
7 // A noter que ce n'est à faire ici que parce qu'on utilise Angular en front-end
8 // sur le port 4200 et PHP en back-end sur le port 80.
9 header("Access-Control-Allow-Origin: *");
10
11 echo json_encode ([
12 ['titre' => 'c1', 'nb_etud' => 2],
13 ['titre' => 'c2', 'nb_etud' => 5],
14 ['titre' => 'c3', 'nb_etud' => 6]
15]);
16
17
18 ?>
```

## Service HTTP : le bon résultat



## Service HTTP : 2 HttpClient avec méthode POST

```
import { HttpClient } from '@angular/common/http';
import { Course } from '../course/course';

@Injectable({
 providedIn: 'root'
})
export class CoursesService {
 // ici, dependency injection : on va utiliser le HttpClient
 constructor(private http_: HttpClient) { }

 // ici, il faut retourner un Observable sur un tableau de Course
 getCourses(): Observable<Course[]> {
 return this.http.post<Course[]>(// on converse avec le serveur via la méthode POST
 url: 'http://127.0.0.1/forum/getCourses.php', // URL du serveur
 body: null // les données qu'on envoie pas POST
);
 }
}
```

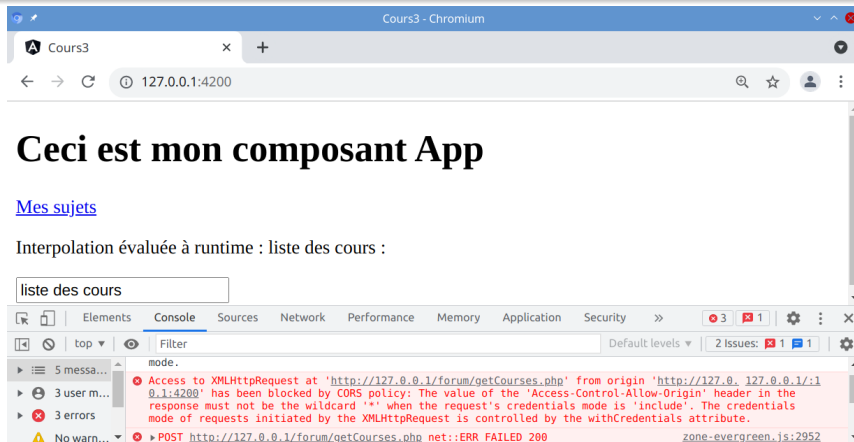
## Service HTTP : 3 méthode POST et session

```
import { HttpClient } from '@angular/common/http';
import { Course } from '../course/course';

@Injectable({
 providedIn: 'root'
})
export class CoursesService {
 // ici, dependency injection : on va utiliser le HttpClient
 constructor(private http: HttpClient) { }

 // ici, il faut retourner un Observable sur un tableau de Course
 getCourses(): Observable<Course[]> {
 return this.http.post<Course[]>(// on converse avec le serveur via la méthode POST
 url: 'http://127.0.0.1/forum/getCourses.php', // URL du serveur
 body: null, // les données qu'on envoie pas POST
 options: {withCredentials: true} // pour capturer les cookies de session
);
 }
}
```

## Service HTTP : 4 problème et solution



▶ **Contre-mesures** : dans `getCourses.php` :

```
header("Access-Control-Allow-Origin: " .
 $_SERVER['HTTP_ORIGIN']);
header("Access-Control-Allow-Credentials: true");
```

## Service HTTP en résumé

- 1 Dans `app.module.ts` : importer le `HttpClientModule`
- 2 Faire `ng generate service mon_service`
- 3 `mon_service` importe la classe `HttpClient`
- 4 Constructeur de `mon_service` : dependency injection de `HttpClient`
- 5 Méthodes qui appellent `get` ou `post` de `HttpClient`  
⇒ renvoient des observables
- 6 Composant « client » importe le service
- 7 Dans son `ngOnInit`, on récupère les observables et on y souscrit  
Le code dépendant des données est dans la callback en paramètre de `subscribe`

## Navigation : les routes dans app-routing.module.ts

```
cours3 - app-routing.module.ts
File Edit View Navigate Code Refactor Run Tools Git Window Help
cours3 | src | app | app-routing.module.ts | Angular CLI Server
Project
2 import { Routes, RouterModule } from '@angular/router';
3 import { CoursesComponent } from './courses/courses.component';
4 import { TopicsComponent } from './topics/topics.component';
5
6 // ici, on indique les routes de l'application. path = l'URL pour accéder au composant
7 const routes: Routes = [
8 { path: 'cours', component: CoursesComponent },
9 // si, dans le path, il y a des ":", cela indique que ce sont des paramètres.
10 // Ainsi, on va pouvoir accéder à sujets/33, où id vaudra 33
11 { path: 'sujets/:id', component: TopicsComponent }
12];
13
14 @NgModule({
15 imports: [RouterModule.forRoot(routes)], // on indique quels routes l'appli utilise
16 exports: [RouterModule]
17 })
18 export class AppRoutingModule { }
19
Problems Git Terminal TODO Event Log
13:1 LF UTF-8 2 spaces* TypeScript 3.5.3 master
```

## Navigation : AppRoutingModuleModule dans app.module.ts

```
cours3 - app.module.ts
File Edit View Navigate Code Refactor Run Tools Git Window Help
cours3 | src | app | app.module.ts | Angular CLI Server
Project
14 import { AppRoutingModule } from './app-routing.module';
15
16 @NgModule({
17 declarations: [
18 AppComponent,
19 CoursesComponent,
20 CourseComponent,
21 TopicsComponent
22],
23 imports: [
24 BrowserModule,
25 AppRoutingModule, // l'appli doit avoir la capacité de faire du routing
26 FormsModule,
27 HttpClientModule
28],
29 providers: [],
30 bootstrap: [AppComponent]
31 })
32 export class AppModule { }
Problems Git Terminal TODO Event Log
15:1 LF UTF-8 2 spaces* TypeScript 3.5.3 master
```

## RouterLink : le lien de navigation

```
cours3 [-enseignement/mobile-19-20/prog/angular/cours3] - .../src/app/courses/courses.component.html - WebStorm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
cours3 | src | app | courses | courses.component.html | Angular CLI Server
Project
1 <!-- on rajoute ici une barre de navigation. Surtout, ne pas
2 utiliser href pour passer d'une page à l'autre : cela
3 rechargerait toute l'application. Ici, il faut utiliser
4 la propriété RouterLink qui permet à Angular de modifier
5 la vue sans rechargement. -->
6 <nav>
7 Mes sujets
8 </nav>
9
10 <p> Interpolation évaluée à runtime : {{ getTitle() }} : {{ nb_etuds }}</p>
11 <div>
12 <p *ngIf="nb_etuds < 10">peu d'étudiants</p>
13 <p *ngIf="nb_etuds >= 10">beaucoup d'étudiants</p>
14 </div>
15 <input type="text" [(ngModel)]="titre" />
16 <app-course *ngFor="let cours of UE"
17 [contenu]="cours"
18 (newNb)="onNewNb($event)"></app-course>
19
Problems Git Terminal TODO Event Log
5:21 LF UTF-8 EditorConfig Git: master
```

## Les liens paramétrés (1/2)

```
cours3 [-enseignement/mobile-19-20/prog/angular/cours3] - .../src/app/topics/topics.component.html - WebStorm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
cours3 | src | app | topics | topics.component.html | Angular CLI Server
Project
17 // ici, on indique les routes de l'application. path = l'URL pour accéder au composant
18 const myRoutes: Routes = [
19 { path: 'cours', component: CoursesComponent },
20 // si, dans le path, il y a des ":", cela indique que ce sont des paramètres.
21 // Ainsi, on va pouvoir accéder à sujets/33, où id vaudra 33
22 { path: 'sujets/:id', component: TopicsComponent }
23];
24
25 <!-- ici, par interpolation, on va afficher l'id qui a été passé en
26 paramètre de l'URL. Pour cela, la classe du composant va récupérer
27 cette information et la transmettre au template HTML par interpolation. -->
28 <p>topics {{my_id}}</p>
29
Problems Git Terminal TODO Event Log
3:81 LF UTF-8 EditorConfig Git: master
```

## Les liens paramétrés (2/2)

```
cours3 - topics.component.ts
File Edit View Navigate Code Refactor Run Tools Git Window Help
cours3 src app topics topics.component.ts Angular CLI Server Git
Project
 topics.component.ts
Structure
 topics.component.ts
Bookmarks
 npm
Problems Git Terminal TODO Event Log
26:1 LF UTF-8 2 spaces* TypeScript 3.5.3 master
```

```
9 export class TopicsComponent implements OnInit {
10 my_id!: number;
11
12 // le constructeur récupère par dependency injection la
13 // route qui a mené à lui => on va pouvoir récupérer les
14 // paramètres passés dans l'URL
15 constructor(private route: ActivatedRoute) { }
16
17 ngOnInit() {
18 // ici, on récupère le paramètre de la route
19 this.my_id = this.route.snapshot.params['id'];
20 }
21 }
22 }
23
24
25
26
27
28
29
```

