

Cours 2 : Frontend A



Web — Applications web et mobile

Christophe Gonzales

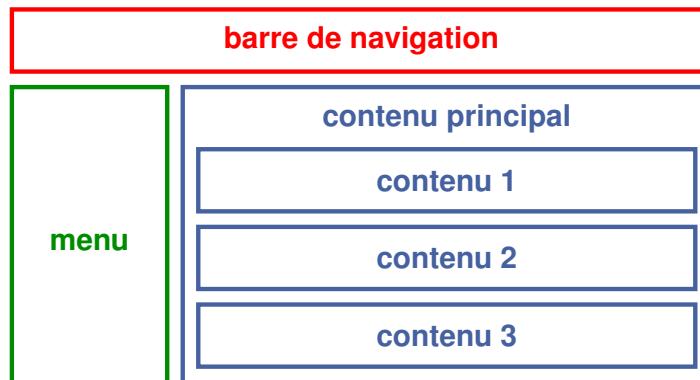
Qu'est-ce qu'Angular

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

2/57

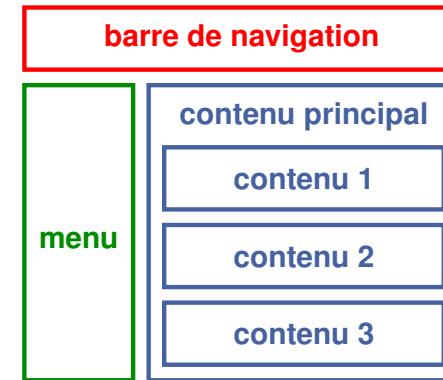
Structure d'une application Angular

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

4/57

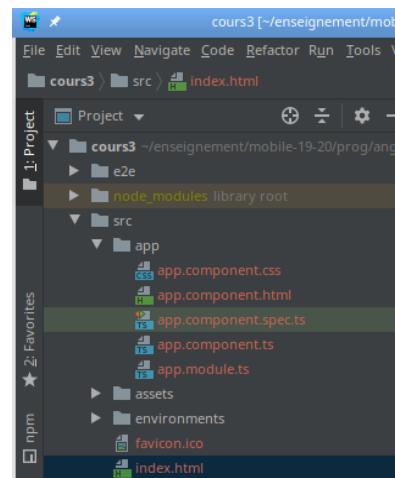
Contenu d'un composant Angular

Un composant Angular contient essentiellement :

- ➊ un fichier TypeScript contenant :
 - ▶ les données du composant
 - ▶ la logique/le comportement du composant
- ➋ un fichier html
 - ▶ contenant le code HTML affiché par le browser
 - ▶ des instructions pour interagir avec le code TypeScript
- ➌ 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

Cours 2 : Frontend Angular

5/57

Index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Cours3</title>
    <base href="/">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />
    <link rel="icon" type="image/x-icon" href="favicon.ico" />
  </head>
  <body>
    <app-root></app-root>
  </body>
</html>
```

Cours 2 : Frontend Angular

7/57

Cours 2 : Frontend Angular

6/57

Fonctionnement d'un projet Angular

- ➊ Créer les composants (et les modules)
- ➋ 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`

Cours 2 : Frontend Angular

8/57

Le composant App et l'appli servie

The screenshot shows the WebStorm IDE interface. The left panel displays the project structure for 'cours3' with files like 'app.component.html', 'app.component.css', and 'app.component.ts'. The right panel shows a browser window titled 'Cours3 - Chromium' displaying the text 'Ceci est mon composant App'. Below the browser is a terminal window showing the command 'ng serve' being run, with output indicating the project is running at <http://localhost:4200>.

Le TypeScript du composant app

The screenshot shows the WebStorm IDE interface. The left panel displays the project structure for 'cours3' with files like 'app.component.html', 'app.component.css', and 'app.component.ts'. The right panel shows the 'app.component.ts' file containing the code for the 'AppComponent' class, which includes annotations for selector, templateUrl, and styleUrls.

Cours 2 : Frontend Angular

9/57

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 interface. The left panel displays the project structure for 'cours3' with a new 'courses' folder added under 'src/app'. The right panel shows a terminal window where the command 'ng generate component courses' is being run, creating files such as 'courses.component.css', 'courses.component.html', 'courses.component.spec.ts', and 'courses.component.ts'.

TypeScript du nouveau composant

The screenshot shows the WebStorm IDE interface. The left panel displays the project structure for 'cours3' with the 'courses' folder and its files. The right panel shows the 'courses.component.ts' file for the 'CoursesComponent' class, which implements the 'OnInit' interface and contains logic for the 'constructor' and 'ngOnInit()' methods.

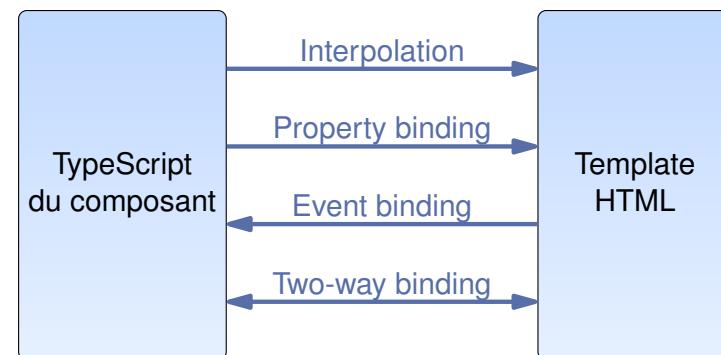
Cours 2 : Frontend Angular

11/57

Cours 2 : Frontend Angular

12/57

Interactions entre le TypeScript et le HTML



Interpolation

Angular CLI Server

```
courses.component.ts
1 @Component({
2   selector: 'app-courses', // balise d'insertion
3   templateUrl: './courses.component.html',
4   styleUrls: ['./courses.component.css']
5 })
6 export class CoursesComponent implements OnInit {
7   titre: string = 'liste des cours';
8
9   constructor() { }
10
11   ngOnInit() {
12     this.getTitle();
13   }
14
15   getTitle() {
16     return this.titre;
17   }
18 }
```

courses.component.html

```
<h1>Ceci est mon composant App</h1>
<app-courses></app-courses>
<app-courses></app-courses>
```

courses.component.html

```
<!-- Interpolation : on insère un champ de la classe
ou n'importe quelle expression javascript en
spécifiant {{ expression }} -->
<p> Interpolation : {{ titre }} et {{ 3 + 4 }}</p>
```

Ceci est mon composant

Interpolation : liste des cours et 7

Interpolation : liste des cours et 7

Version Control Terminal TypeScript 3.5.3 TODO Event Log

TSLint: The project code style and editor settings were updated based... (today 08:15) 12:1 LF UTF-8 EditorConfig Git: master

Cours 2 : Frontend Angular

13/57

Interpolation avec des méthodes

Cours 2 : Frontend Angular

14/57

Property binding

Angular CLI Server

```
courses.component.ts
1 @Component({
2   selector: 'app-courses', // balise d'insertion
3   templateUrl: './courses.component.html',
4   styleUrls: ['./courses.component.css']
5 })
6 export class CoursesComponent implements OnInit {
7   titre: string = 'liste des cours';
8   ma_valeur: string = 'valeur initiale';
9
10  constructor() { }
11
12  ngOnInit() {
13    this.getTitle();
14  }
15
16  getTitle() {
17    return this.titre;
18  }
19 }
```

courses.component.html

```
<!-- Property Binding : on insère une valeur dans une propriété d'un
champ d'un élément du DOM. L'insertion se fait via :
[nom propriété] = "valeur_dans_le_typescript"-->
<p> Interpolation : {{ titre }} et {{ 3 + 4 }}</p>
<input type="text" [value]="ma_valeur" />
```

Ceci est mon composant

Interpolation : liste des cours et 7

valeur initiale

Version Control Terminal TypeScript 3.5.3 TODO Event Log

TSLint: The project code style and editor settings were updated based... (today 08:15) 3:48 LF UTF-8 EditorConfig Git: master

Cours 2 : Frontend Angular

16/57

Cours 2 : Frontend Angular

15/57

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

```
courses3 [-/enseignement/mobile-19-20/prog/angular/cours3] - .../src/app/courses/courses.component.ts - WebStorm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
cours3 > src > app > courses > courses.component.ts
cours3 ~enseignement e2e node_modules librairie
src app courses
courses.component.ts
templateUrl: './courses.component.html',
styleUrls: ['./courses.component.css']
}
export class CoursesComponent implements OnInit {
titre = 'liste des cours';
constructor() { }
ngOnInit() {}

getTitle(): string { return this.titre; }

modifTitle() {
this.titre = 'nouveau titre';
}

CoursesComponent > getTitle()
CoursesComponent > modifTitle()

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

input
Version Control Terminal TypeScript 3.5.3 TODO
TSLint: The project code style and editor settings were updated bas... (today 08:15) 14:36 LF UTF-8 EditorConfig Git: master Event Log
```

Ceci est mon composant

Interpolation évaluée à runtime : nouveau titre

Interpolation évaluée à runtime : liste des cours

Cours 2 : Frontend Angular

17/57

Two-way binding (1/2)

Cours 2 : Frontend Angular

18/57

Two-way binding (2/2)

```
courses3 [-/enseignement/mobile-19-20/prog/angular/cours3] - .../src/app/app.module.ts - WebStorm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
cours3 > src > app > app.module.ts
cours3 ~enseignement e2e node_modules librairie
src app
app.module.ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { CoursesComponent } from './courses/courses.component';

// pour utiliser ngModel, il faut importer FormsModule
import { FormsModule } from '@angular/forms';

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

Ceci est mon composant

Interpolation évaluée à runtime : liste des
listes
Interpolation évaluée à runtime : liste des cours
listes cours
Version Control Terminal TypeScript 3.5.3 TODO
TSLint: The project code style and editor settings were updated bas... (yesterday 08:15) 16:13 LF UTF-8 EditorConfig Git: master Event Log
```

Cours 2 : Frontend Angular

20/57

Cours 2 : Frontend Angular

19/57

Cours 2 : Frontend Angular

19/57

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

Cours 2 : Frontend Angular

21/57

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

The screenshot shows the WebStorm IDE with the courses.component.html file open. The template contains:

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

The browser window shows the output: "Ceci est mon composant App" followed by "Interpolation évaluée à runtime : liste des cours". Below it is a text input field containing "liste des cours" and two lines of text below it saying "course works!".

Cours 2 : Frontend Angular

23/57

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

The screenshot shows the WebStorm IDE with the courses.component.ts file open. The code defines a CoursesComponent that implements OnInit:

```
export interface Course { // interface imposant les informations
  titre_: string; // que doit contenir un cours
  nb_etud_: number;
}

@Component({
  selector: 'app-courses',
  templateUrl: './courses.component.html',
  styleUrls: ['./courses.component.css']
})
export class CoursesComponent implements OnInit {
  titre = 'liste des cours';
  UE_ : Course[] = [ {titre: 'c1', nb_etud: 2}, {titre: 'c2', nb_etud: 5} ];
  constructor() {}
  ngOnInit() {}
  getTitle() { return this.titre; }
}
```

Cours 2 : Frontend Angular

22/57

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

The screenshot shows the WebStorm IDE with the course.component.ts file open. The code defines a CourseComponent that implements OnInit:

```
import {Component, Input, OnInit} from '@angular/core';
import {Course} from './course';

@Component({
  selector: 'app-course',
  templateUrl: './course.component.html',
  styleUrls: ['./course.component.scss']
})
export class CourseComponent implements OnInit {
  // Avec des @Input, on indique que, dans le HTML du parent, on va
  // binder la propriété "contenu" de <app-course>. Cela permettra à
  // la classe CourseComponent de recevoir les informations dont
  // elle a besoin pour s'afficher correctement.
  @Input() contenu!: Course; // Ici, il est important de spécifier
                            // le type de contenu. Notez le ! qui permet, en mode strict,
                            // que contenu ne soit pas initialisé lors de sa création.
  constructor() {}
}

<p>Cours : {{contenu.titre}} -- {{contenu.nb_etud}} étudiants</p>
```

Cours 2 : Frontend Angular

24/57

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

The screenshot shows the WebStorm IDE with the project structure on the left. The current file is `courses.component.html`. The code contains interpolation and property binding. Below the editor is a browser window displaying the application's home page with the title "Ceci est mon composant App". The page lists two courses: "Cours : c1 -- 2 étudiants" and "Cours : c2 -- 5 étudiants".

Interactions enfant → parent : préparation de l'enfant

The screenshot shows the WebStorm IDE with the project structure on the left. The current file is `course.component.html`. The code includes an input field with a name attribute and a change event binding. A multi-line comment explains the purpose of this binding: "on rajoute un event binding sur l'événement 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. -->".

Cours 2 : Frontend Angular

25/57

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

The screenshot shows the WebStorm IDE with the project structure on the left. The current file is `course.component.ts`. The code defines a `CourseComponent` class that implements `OnInit`. It uses `@Input()` and `@Output()` decorators to handle data exchange between parent and child components. The `updateNb()` method is triggered by the input change event.

Cours 2 : Frontend Angular

26/57

Interactions enfant → parent : la réception du parent

The screenshot shows the WebStorm IDE with the project structure on the left. The current file is `courses.component.html`. The code uses interpolation and property binding. The `courses.component.ts` file is also shown, defining the `CoursesComponent` class which receives the new number of students via the `newNb` event from the child component.

Cours 2 : Frontend Angular

28/57

Interactions enfant → parent : le résultat

Ceci est mon composant App

Interpolation évaluée à runtime : liste des cours : 7

liste des cours

Cours : c1

2 étudiants

Cours : c2

5 étudiants

Ceci est mon composant App

Interpolation évaluée à runtime : liste des cours : 60

liste des cours

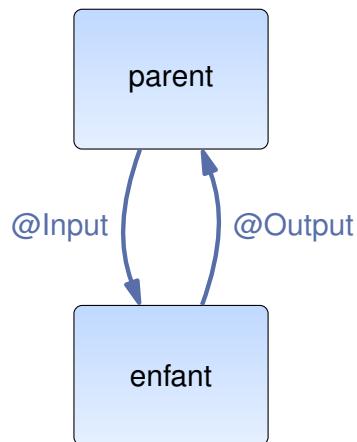
Cours : c1

55 étudiants

Cours : c2

5 étudiants

Résumé des interactions enfant – parent



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

Cours 2 : Frontend Angular

29/57

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`

Cours 2 : Frontend Angular

30/57

La directive `*ngFor`

```
<p> Interpolation évaluée à runtime : {{ getTitle() }} : {{nb_etuds}}</p>
<input type="text" [(ngModel)]="titre" />
<!-- directive *ngFor : on parcourt tous les éléments de UE. Chaque
élément est stocké dans la variable cours. On peut alors utiliser
cette variable, notamment dans les property bindings -->
<app-course *ngFor="let cours of UE"
  [contenu]="cours"
  (newNb)="onNewNb($event)"></app-course>

<!-- attention : on ne peut pas appliquer 2 directives (par exemple ngIf et ngFor
au même composant. Il faut choisir : soit utiliser ngIf, soit ngFor.
Si l'on a besoin des 2, l'astuce est de créer un composant supplémentaire
auquel on appliquera, par exemple, le ngFor, et ce composant contiendra le
composant qui nous intéresse, auquel on appliquera le ngIf -->
```

Cours 2 : Frontend Angular

31/57

31/57

32/57

La directive *ngIf

```
<p> Interpolation évaluée à runtime : {{ getTitle() }} : {{nb_etuds}}</p>
<!-- Ici, la page web ne contiendra qu'un seul des paragraphes ci-dessous.
Si Courses contient moins de 10 étudiants, ce sera le 1er paragraphe,
sinon, ce sera le 2ème. Quand je dis que "la page web contiendra...", 
je parle du DOM : le DOM ne contiendra effectivement qu'un <p>, et
non 2 <p> avec l'un des deux hidden. -->
<p *ngIf="nb_etuds < 10">peu d'étudiants</p>
<p *ngIf="nb_etuds >= 10">beaucoup d'étudiants</p>
</div>
<input type="text" [(ngModel)]="titre" />
<!-- directive *ngFor : on parcourt tous les éléments de UE. Chaque
élément est stocké dans la variable cours. On peut alors utiliser
cette variable, notamment dans les property bindings -->
<app-course *ngFor="let cours of UE"
    [contenu]="cours"
    (newNb)="onNewNb($event)"></app-course>
```

Cours 2 : Frontend Angular 33/57

Les services : providers de données

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



les services récupèrent les données

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

Cours 2 : Frontend Angular 35/57

La directive *ngIf : le résultat

Ceci est mon composant App

Interpolation évaluée à runtime : liste des cours : 12
beaucoup d'étudiants

Cours : c1

2 étudiants

Cours : c2

5 étudiants

Cours : c3

5 étudiants

Ceci est mon composant App

Interpolation évaluée à runtime : liste des cours : 7
peu d'étudiants

Cours : c1

2 étudiants

Cours : c2

0 étudiants

Cours : c3

5 étudiants

Cours 2 : Frontend Angular 34/57

Anatomie d'un service

```
import { Injectable } from '@angular/core';
@Injectable({
  providedIn: 'root'
})
export class CoursesService {
  constructor() { }

  // ici, il suffit de créer une méthode qui renvoie les
  // données dont a besoin le composant Courses. Ce dernier
  // aura juste à ainsi juste à appeler la méthode pour
  // obtenir sa liste d'UE
  getcourses() {
    return [ {titre: 'c1', nb_etud: 2},
            {titre: 'c2', nb_etud: 5},
            {titre: 'c3', nb_etud: 5} ];
  }
}
```

Cours 2 : Frontend Angular 36/57

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 CoursesComponent Angular CLI Server Git: 9 1 3
Project Structure
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.getGames();
20   }
21   ngOnInit() { this.getNbEtuds(); }
22 }
Problems Git Terminal TODO
22:1 LF UTF-8 2 spaces* TypeScript 3.5.3 master Event Log
```

Cours 2 : Frontend Angular

37/57

Dependency injection

```
cours3 - courses.component.ts
File Edit View Navigate Code Refactor Run Tools Git Window Help
cours3> src> courses> courses.component.ts CoursesComponent ngOnInit Angular CLI Server Git: 8 1 3
Project Structure
1 import { Component, OnInit } from '@angular/core';
2 import { Course } from '../course/course';
3 import { CoursesService } from '../services/courses.service';
4
5 @Component({
6   selector: 'app-courses', // balise d'insertion
7   templateUrl: './courses.component.html',
8   styleUrls: ['./courses.component.css']
9 })
10 export class CoursesComponent implements OnInit {
11   titre = 'Liste des cours';
12   UE!: Course[]; // ici, on n'initialise plus la liste d'UE
13   nb_etuds!: number;
14   // dependency injection : on passe le service en paramètre du constructeur.
15   // Angular ne créera qu'une seule instance de CoursesService pour tout l'application
16   // et passera cette instance au constructeur
17   constructor( private service: CoursesService ) {}
18   ngOnInit() {
19     this.UE = this.service.getGames(); // c'est ngOnInit qui exploite le service
20     this.getNbEtuds();
21   }
22   getTitle() { return this.titre; }
23   getNbEtuds() {
24     this.nb_etuds = 0;
25     for (const ue of this.UE) this.nb_etuds += ue.nb_etud;
26   }
27 }
Problems Git Terminal TODO
20:83 LF UTF-8 2 spaces* TypeScript 3.5.3 master Event Log
```

Cours 2 : Frontend Angular

39/57

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 => plusieurs instances
 - Modif du constructeur du service => modif du composant
- Utiliser le service dans le constructeur
 - => délais dans les affichages



Bonne pratique des services :

- Dependency injection
- Utiliser le service dans méthode ngOnInit

Cours 2 : Frontend Angular

38/57

Asynchronie et observables

- Service actuel : synchrone
 - => ngOnInit doit attendre les données
- Services HTTP : asynchrones
 - => évite de bloquer les affichages



Utilisation de services asynchrones :

- ➊ Le service retourne tout de suite un **Observable**
- ➋ ngOnInit appelle le service et récupère l'observable
- ➌ ngOnInit souscrit à l'observable en donnant une callback
- ➍ ngOnInit continue son exécution
- ➎ Les données arrivent => l'observable émet une valeur
 - => la callback est appelée

Cours 2 : Frontend Angular

40/57

Mise en place des observables dans le service

```
import { Injectable } from '@angular/core';
// au lieu de renvoyer un tableau de Course, on va renvoyer un Observable sur
// ce tableau => il faut importer les déclarations des observables
import { Observable, of } from 'rxjs';
import { Course } from '../course/course';

@Injectable({
  providedIn: 'root'
})
export class CoursesService {
  constructor() {}

  // ici, il faut retourner un Observable sur un tableau de Course
  getCourses_(): Observable<Course[]> {
    // of retourne un Observable
    return of([
      {titre: 'c1', nb_etud: 2},
      {titre: 'c2', nb_etud: 5},
      {titre: 'c3', nb_etud: 5}
    ]);
  }
}
```

Mise en place des observables dans le composant

```
export class CoursesComponent implements OnInit {
  titre = 'liste des cours';
  UE!: Course[]; // ici, on n'initialise plus la liste d'UE
  nb_etuds!: number;
  constructor( private service: CoursesService ) {}
  ngOnInit() {
    // on appelle le service, qui nous retourne un observable et on y souscrit.
    // Le paramètre est une fonction (arrow) qui prend en paramètre la valeur
    // renournée par l'observable. Cette valeur étant la liste des UE, on la
    // sauvegarde dans this.UE. Cette opération ne sera réalisée effectivement que
    // quand le service aura pu récupérer les données demandées.
    this.service.getCourses().subscribe(
      next: courses => {
        this.UE = courses; // on récupère la liste des cours
        this.getNbEtuds(); // on effectue le reste des opérations dans cette callback
      }
    );
  }
  getTitle() { return this.titre; }
  getNbEtuds() {
    this.nb_etuds = 0;
    for (const ue of this.UE) this.nb_etuds += ue.nb_etud;
  }
}
```

Cours 2 : Frontend Angular

41/57

Service HTTP : ① importer le HttpClientModule

```
import { CoursesComponent } from './courses/courses.component';
import { FormsModule } from '@angular/forms';
import { CourseComponent } from './course/course.component';
// pour utiliser HttpClient, importer son module
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  declarations: [
    AppComponent,
    CoursesComponent,
    CourseComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    HttpClientModule // importer ici le module HttpClient
  ],
  providers: []
})
```

Cours 2 : Frontend Angular

42/57

Service HTTP : ② notre service utilise HttpClient

```
import { Injectable } from '@angular/core';
// on a toujours besoin des Observables, mais on rajoute HttpClient, qui va
// s'occuper de converser avec le serveur web
import { Observable } from 'rxjs';
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.get<Course[]>(`http://127.0.0.1/forum/getCourses.php`);
  }
}
```

Cours 2 : Frontend Angular

44/57

Cours 2 : Frontend Angular

43/57

Service HTTP : ③ getCourses.php

File Edit View Navigate Code Refactor Run Tools VCS Window Help

home apache forum getCourse.php Angular CLI Server Git: Event Log

```
<?php
header('Content-type:application/json;charset=utf8');
//header("Access-Control-Allow-Origin: *");

echo json_encode([
    ['titre' => 'c1', 'nb_etud' => 2 ],
    ['titre' => 'c2', 'nb_etud' => 5 ],
    ['titre' => 'c3', 'nb_etud' => 6 ]
]);
?>
```

TODO Version Control TypeScript 3.5.3 Terminal Event Log

12:3 LF UTF-8 4 spaces Git: master

Cours 2 : Frontend Angular 45/57

Service HTTP : Résultat

Cours3 - Chromium

Ceci est mon composant App

Interpolation évaluée à runtime : liste des cours :

liste des cours

Elements Console Sources Network Performance Memory Application Security Audits

top Filter Default levels

- 4 messages
- 3 user mess...
- 2 errors

Access to XMLHttpRequest at 'http://127.0.0.1/forum/getCourses.php' from origin 'http://localhost:4200' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.

No warnings

► Cross-Origin Resource Sharing (CORS) :

- Requête cross-origine : provient de localhost :4200 accède à localhost :80
⇒ CORS refuse la requête

► Contre-mesure : dans getCourse.php, rajouter :

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

Cours 2 : Frontend Angular

46/57

Service HTTP : ③ le bon getCourse.php

File Edit View Navigate Code Refactor Run Tools VCS Window Help

home apache forum getCourse.php Angular CLI Server Git: Event Log

```
<?php
// le script doit renvoyer les données au format JSON
header('Content-type:application/json;charset=utf8');

// ici, on rajoute un header pour contourner le problème CORS
// A noter que ce n'est à faire ici que parce qu'on utilise Angular en front-end
// sur le port 4200 et PHP en back-end sur le port 80.
header("Access-Control-Allow-Origin: *");

echo json_encode([
    ['titre' => 'c1', 'nb_etud' => 2 ],
    ['titre' => 'c2', 'nb_etud' => 5 ],
    ['titre' => 'c3', 'nb_etud' => 6 ]
]);
?>
```

TODO Version Control TypeScript 3.5.3 Terminal Event Log

18:1 LF UTF-8 4 spaces Git: master

Cours 2 : Frontend Angular 47/57

Service HTTP : le bon résultat

48/57

Ceci est mon composant App

Interpolation évaluée à runtime : liste des cours : 13

beaucoup d'étudiants

liste des cours

Cours : c1

2 étudiants

Cours : c2

5 étudiants

Cours : c3

6 étudiants

Cours3 - Chromium

Ceci est mon composant App

Interpolation évaluée à runtime : liste des cours : 13

liste des cours

Elements Console Sources Network Performance Memory Application Security Audits

top Filter Default levels

- 2 messages
- 2 user mess...

No errors

Cours 2 : Frontend Angular

48/57

Service HTTP : ② HttpClient avec méthode POST

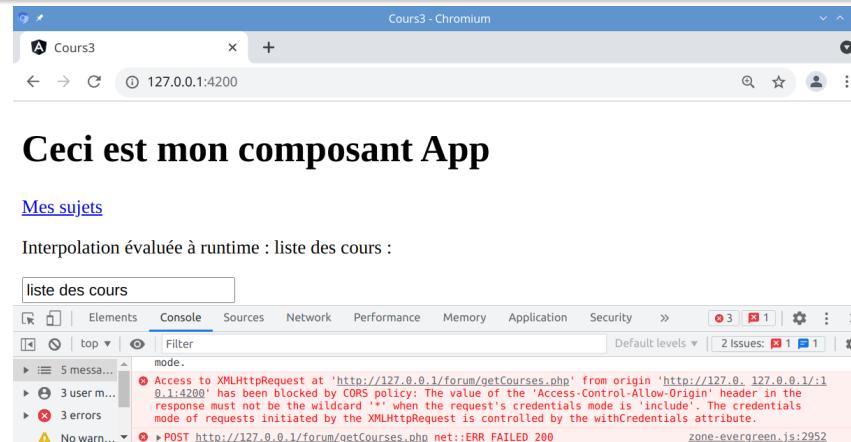
```
import { HttpClient } from '@angular/common/http';
import { Course } from '../course/course';

@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[]>(`http://127.0.0.1/forum/getCourses.php`, // URL du serveur
      { body: null }, // les données qu'on envoie pas POST
    );
  }
}
```

Cours 2 : Frontend Angular 49/57

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");
```

Cours 2 : Frontend Angular 51/57

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[]>(`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
      });
  }
}
```

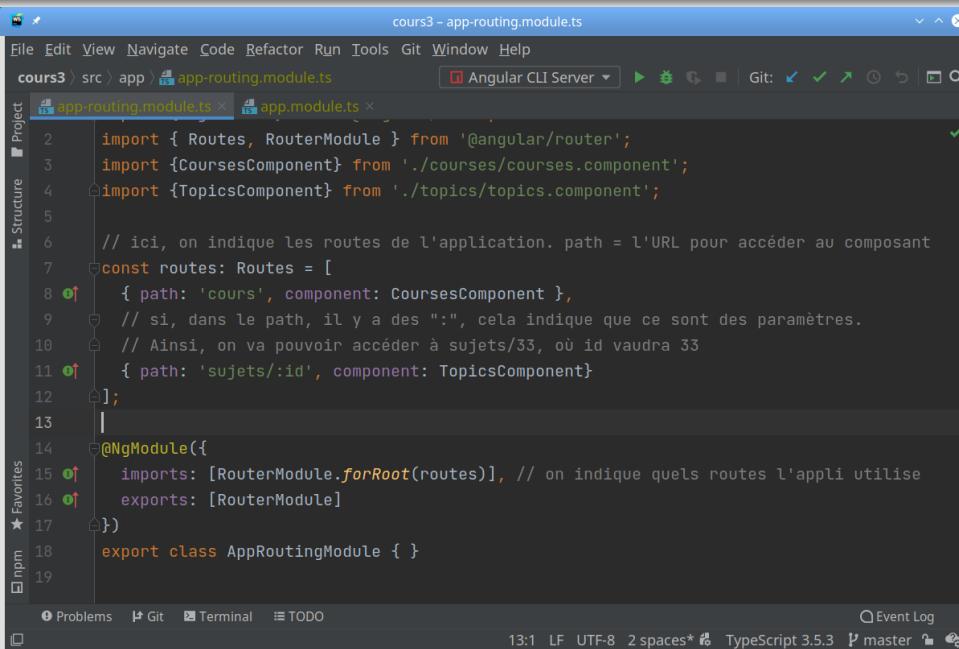
Cours 2 : Frontend Angular 50/57

Service HTTP en résumé

- ➊ Dans app.module.ts : importer le HttpClientModule
- ➋ Faire ng generate service mon_service
- ➌ mon_service importe la classe HttpClient
- ➍ Constructeur de mon_service : dependency injection de HttpClient
- ➎ Méthodes qui appellent get ou post de HttpClient
⇒ renvoient des observables
- ➏ Composant <> client <> importe le service
- ➐ Dans son ngOnInit, on récupère les observables et on y souscrit
Le code dépendant des données est dans la callback en paramètre de subscribe

Cours 2 : Frontend Angular 52/57

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

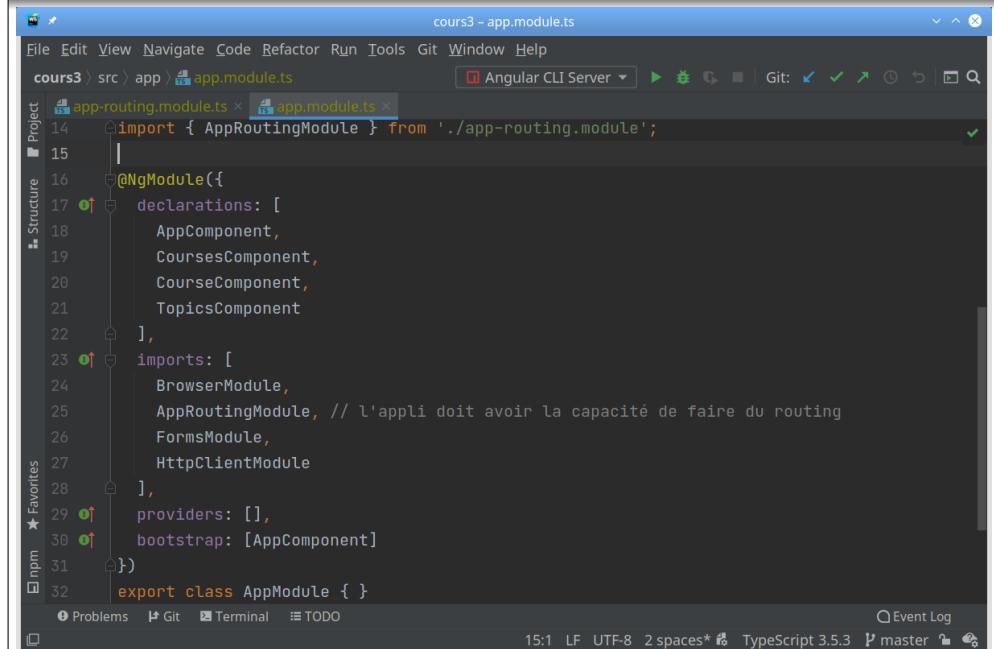


```
File Edit View Navigate Code Refactor Run Tools Git Window Help
cours3 > src > app > app-routing.module.ts
import { Routes, RouterModule } from '@angular/router';
import { CoursesComponent } from './courses/courses.component';
import { TopicsComponent } from './topics/topics.component';

// ici, on indique les routes de l'application. path = l'URL pour accéder au composant
const routes: Routes = [
  { path: 'cours', component: CoursesComponent },
  // si, dans le path, il y a des ":", cela indique que ce sont des paramètres.
  // Ainsi, on va pouvoir accéder à sujets/33, où id vaudra 33
  { path: 'sujets/:id', component: TopicsComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)], // on indique quels routes l'appli utilise
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Navigation : AppRoutingModule dans app.module.ts



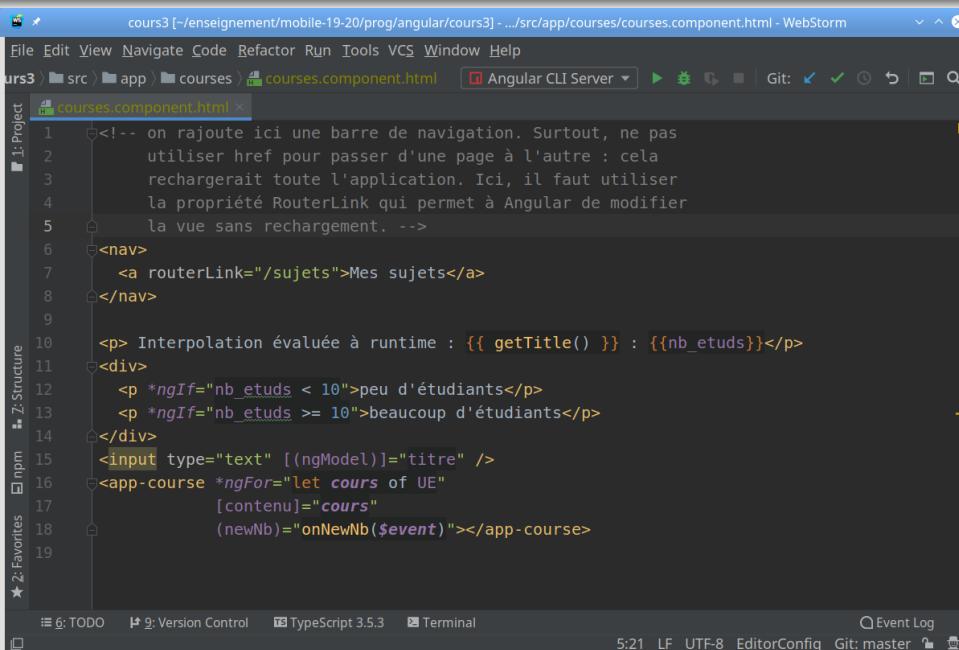
```
File Edit View Navigate Code Refactor Run Tools Git Window Help
cours3 > src > app > app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { CoursesComponent } from './courses/courses.component';
import { CourseComponent } from './course/course.component';
import { TopicsComponent } from './topics/topics.component';

@NgModule({
  declarations: [
    AppComponent,
    CoursesComponent,
    CourseComponent,
    TopicsComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule, // l'appli doit avoir la capacité de faire du routing
    FormsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Cours 2 : Frontend Angular

53/57

RouterLink : le lien de navigation



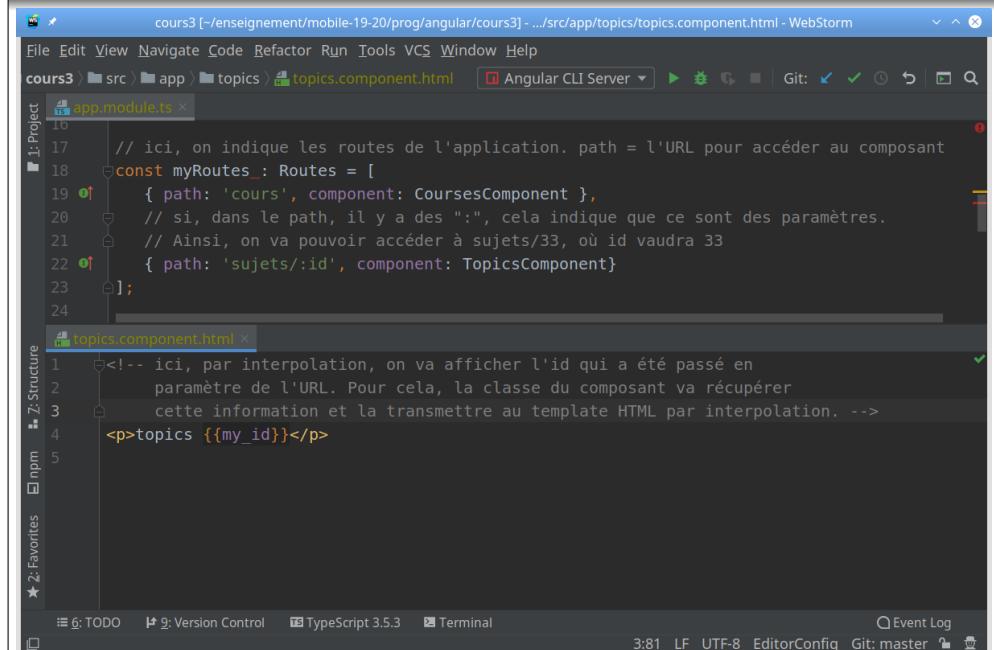
```
File Edit View Navigate Code Refactor Run Tools VCS Window Help
cours3 [<~/enseignement/mobile-19-20/prog/angular/cours3 - .../src/app/courses/courses.component.html - WebStorm]
courses > courses > courses.component.html
<!!-- on rajoute ici une barre de navigation. Surtout, ne pas utiliser href pour passer d'une page à l'autre : cela rechargerait toute l'application. Ici, il faut utiliser la propriété RouterLink qui permet à Angular de modifier la vue sans rechargeement. -->
<nav>
  <a routerLink="/sujets">Mes sujets</a>
</nav>

<p> Interpolation évaluée à runtime : {{ getTitle() }} : {{nb_etuds}}</p>
<div>
  <p *ngIf="nb_etuds < 10">peu d'étudiants</p>
  <p *ngIf="nb_etuds >= 10">beaucoup d'étudiants</p>
</div>
<input type="text" [(ngModel)]="titre" />
<app-course *ngFor="let cours of UE"
  [contenu]="cours"
  (newNb)="onNewNb($event)"></app-course>
```

Cours 2 : Frontend Angular

54/57

Les liens paramétrés (1/2)



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help
cours3 [<~/enseignement/mobile-19-20/prog/angular/cours3 - .../src/app/topics/topics.component.html - WebStorm]
topics > topics > topics.component.html
<!!-- ici, on indique les routes de l'application. path = l'URL pour accéder au composant
const myRoutes: Routes = [
  { path: 'cours', component: CoursesComponent },
  // si, dans le path, il y a des ":", cela indique que ce sont des paramètres.
  // Ainsi, on va pouvoir accéder à sujets/33, où id vaudra 33
  { path: 'sujets/:id', component: TopicsComponent }
];
```

```
<!-- ici, par interpolation, on va afficher l'id qui a été passé en
paramètre de l'URL. Pour cela, la classe du composant va récupérer
cette information et la transmettre au template HTML par interpolation. -->
<p>topics {{my_id}}</p>
```

Cours 2 : Frontend Angular

56/57

Les liens paramétrés (2/2)

The screenshot shows a code editor and a browser window side-by-side.

Code Editor: The file `topics.component.ts` contains the following TypeScript code:

```
File Edit View Navigate Code Refactor Run Tools Git Window Help
cours3 > src > app > topics > topics.component.ts
Angular CLI Server | Git: ✓ ✓ ✓
Project Structure
topics.component.ts
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
```

Browser: The browser window displays the URL `localhost:4200/sujets/25`. The page content is "Ceci est mon composant App" with the word "topic" underlined and circled in red, and the number "25" also circled in red.