

Cours 4 : Frontend – partie 1



Applications web et mobiles

Christophe Gonzales

Création d'un nouveau composant

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

```
File Edit View Navigate Code Refactor Run Tools Git Window Help
xmobile_cours src) app) app.component.html Angular CLI Server Git:
Project
  xmobile_cours /home/apache/xmobile_cours
  .angular
  .vscode
  node_modules library root
  src
    app
      courses
        courses.component.html
        courses.component.scss
        courses.component.spec.ts
        courses.component.ts
      app.component.html
      app.component.scss
      app.component.spec.ts
1 <h1>Ceci est mon composant App</h1>
2
3 <router-outlet></router-outlet>
4
Terminal: Local x +
[shalmanser:/home/apache/xmobile_cours]<1> ng generate component courses
Node.js version v21.4.0 detected.
Odd numbered Node.js versions will not enter LTS status and should not be used for production. F
or more information, please see https://nodejs.org/en/about/previous-releases/.
CREATE src/app/courses/courses.component.scss (0 bytes)
CREATE src/app/courses/courses.component.html (22 bytes)
CREATE src/app/courses/courses.component.spec.ts (603 bytes)
CREATE src/app/courses/courses.component.ts (239 bytes)
[shalmanser:/home/apache/xmobile_cours]<2> 
```

TypeScript du nouveau composant

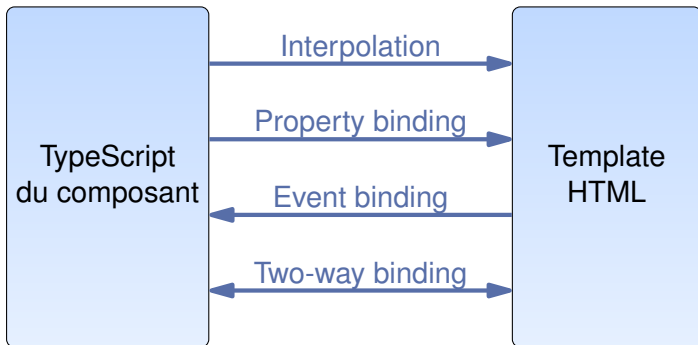
The screenshot shows the Visual Studio Code editor with the following details:

- File Explorer (Left):** Shows the project structure. The file `courses.component.ts` is selected under the path `src/app/courses`.
- Code Editor (Center):** Displays the TypeScript code for `courses.component.ts`. The code includes an Angular component decorator, a class implementing `OnInit`, and a `ngOnInit` method.
- Terminal (Bottom):** Shows the message: "Externally added files can be added to Git // View Files // Always Ad... (today 10:32)".

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

@Component({
  selector: 'app-courses',
  standalone: true,
  imports: [],
  templateUrl: './courses.component.html',
  styleUrls: ['./courses.component.scss']
})
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 attributs en
  // récupérant des données de bases de données (elle
  // n'est pas bloquante).
  ngOnInit():void {
  }
}
```



Interpolation

The image shows a code editor with three files open:

- courses.component.ts**:

```
@Component({
  selector: 'app-courses',
  standalone: true,
  imports: [],
  templateUrl: './courses.component.html',
  styleUrls: ['./courses.component.scss']
})
export class CoursesComponent implements OnInit {
  titre:string = 'liste de cours';

  constructor() {}
}
```
- app.component.html**:

```
<h1>Ceci est mon composant App</h1>
<app-courses></app-courses>
<app-courses></app-courses>
<router-outlet></router-outlet>
```
- courses.component.html**:

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

A browser window titled "XmobileCours" is open, showing the rendered output of the component. The address bar shows the URL `127.0.0.1:4200`. The page content is:

Ceci est mon composant App

Interpolation : liste de cours et 7

Interpolation : liste de cours et 7

Interpolation avec des méthodes

The image shows a development environment with a code editor and a browser window. The code editor displays the following TypeScript code for a component:

```
File Edit View Navigate Code Refactor Run Tools Git Window Help
courses.component.ts > CoursesComponent > getTitle() Angular CLI Server
courses.component.ts
imports: [],
templateUrl: './courses.component.html',
styleUrl: './courses.component.scss'
})
export class CoursesComponent implements OnInit {
  titre = 'liste de cours';

  constructor() {}

  ngOnInit() {}

  getTitle() {
    return this.titre;
  }
}
```

The browser window, titled "XmobileCours", shows the rendered output of the component:

Ceci est mon composant App

Interpolation bis : liste de cours

Interpolation bis : liste de cours

At the bottom of the code editor, the interpolation syntax is shown: `<p> Interpolation bis : {{getTitle()}}</p>`

Property binding

The image shows a development environment with the following components:

- Top Panel:** File Explorer showing the file path `mobile_cours > src > app > courses > courses.component.ts`. The Angular CLI Server is running.
- Editor (Left):** TypeScript code for `CoursesComponent` implementing `OnInit`.

```
export class CoursesComponent implements OnInit {
  titre = 'liste de cours';
  ma_valeur = 'valeur initiale';

  constructor() {}

  ngOnInit() {}

  getTitle() {
    return this.titre;
  }
}
```
- Editor (Bottom):** HTML template for `courses.component.html`.

```
<p> Interpolation bis : {{getTitle()}}</p>
<!-- Property binding : on insère une valeur dans une propriété d'un élément du DOM.
L'insertion se fait via :
[nom_propriété]="valeur provenant du TypeScript" -->
<input type="text" [value]="ma_valeur"/>
```
- Browser Preview (Right):** A window titled `XmobileCours` at `127.0.0.1:4200` displaying the rendered output:

Ceci est mon composant App

Interpolation bis : liste de cours

► **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

The image shows a development environment with two main windows. The top window is a code editor for `courses.component.ts`, containing the following TypeScript code:

```
export class CoursesComponent implements OnInit {  
  titre = 'liste de cours';  
  ma_valeur = 'valeur initiale';  
  
  constructor() {}  
  
  ngOnInit() {}  
  
  getTitle() {  
    return this.titre;  
  }  
  
  modifierTitle() {  
    this.titre = 'nouveau titre'  
  }  
}
```

The bottom window is a browser displaying the rendered application. The browser address bar shows `127.0.0.1:4200`. The page content is as follows:

Ceci est mon composant App

Interpolation évaluée à runtime : nouveau titre

Interpolation évaluée à runtime : liste de cours

The bottom part of the IDE shows the corresponding HTML template for `courses.component.html`:

```
<p> Interpolation évaluée à runtime : {{getTitle()}}</p>  
  
<!-- Event binding: (event)="callback()" -->  
<input type="text" (click)="modifierTitle()"/>
```

Two-way binding (1/2)

The image shows a development environment with the following components:

- File Explorer:** Shows the file path `mobile_cours > src > app > courses > courses.component.ts`.
- Code Editor:** Contains the following TypeScript code for `CoursesComponent`:

```
export class CoursesComponent implements OnInit {
  titre = 'liste de cours';
  ma_valeur = 'valeur initiale';

  constructor() {}

  ngOnInit() {}

  getTitle() {
    return this.titre;
  }

  modifierTitre() {
    this.titre = 'nouveau titre';
  }
}
```
- HTML Editor:** Shows the template for `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"/>
```
- Browser Window:** Displays an error message: `NG8002: Can't bind to 'ngModel' since it isn't a known` with the file path `src/app/courses/courses.component.html:4:19`.
- Terminal/Status Bar:** Shows the Angular CLI server running on `127.0.0.1:4200` and the TypeScript version `TypeScript 5.2.2`.

Two-way binding (2/2)

The image shows a development environment with an IDE on the left and a browser preview on the right. The IDE displays the `courses.component.ts` file with the following code:

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

@Component({
  selector: 'app-courses',
  standalone: true,
  imports: [
    FormsModule
  ],
  templateUrl: './courses.component.html',
  styleUrls: ['./courses.component.scss']
})
export class CoursesComponent implements OnInit {
  titre = 'liste de cours';
  ma_valeur = 'valeur initiale';

  constructor() {}
}
```

The browser preview shows the rendered component with the following content:

XmobileCours

127.0.0.1:4200

Ceci est mon composant App

Interpolation évaluée à runtime : liste XXXXX

Interpolation évaluée à runtime : liste de cours

The IDE also shows the `courses.component.html` file with the following code:

```
<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"/>
```

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

Interfaces et données

The image shows a development environment with two windows. The background window is a code editor displaying the following TypeScript code:

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

// interface imposant les informations que doit contenir un cours
export interface Course {
  nom: string; // on indique le nom des attributs ainsi
  nb_etuds : number; // que leur type
}

export class CoursesComponent implements
  titre = 'liste de cours';
  UE : Course[] = [
    {nom: 'c1', nb_etuds: 3},
    {nom: 'c2', nb_etuds: 5}
  ];
}
```

The foreground window is a browser showing the rendered application:

XmobileCours

127.0.0.1:4200

Ceci est mon composant App

liste de cours

- c1 : 3 étuds
- c2 : 5 étuds

The IDE interface includes a menu bar (File, Edit, View, Navigate, Code, Refactor, Run, Tools, Git, Window, Help), a toolbar with icons for running and debugging, and a sidebar with Project, Commit, Bookmarks, and npm views.

- ▶ 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

Anatomie d'un service

The screenshot shows a Visual Studio Code editor with a project structure on the left and a code editor in the center. The project structure includes folders for `.angular`, `src`, `app`, `courses`, `services`, and `assets`. The `courses` folder contains `courses.component.html`, `courses.component.scss`, `courses.component.spec.ts`, and `courses.component.ts`. The `services` folder contains `courses.service.spec.ts` and `courses.service.ts`. The `assets` folder contains `favicon.ico`, `index.html`, `main.ts`, `styles.scss`, and `editorconfig`.

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

// Comme le service a pour objectif de transmettre les listes de
// cours, il est logique que l'interface Course soit définie ici
// plutôt que dans le composant Courses
export interface Course {
  nom: string;
  nb_etuds : number;
}

@Injectable({ // le service sera créé par dependency injection
  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 ainsi
  // juste à appeler la méthode pour obtenir sa liste d'UE
  getCourses (): Course[] {
    return [{nom: 'c1', nb_etuds: 2},
            {nom: 'c2', nb_etuds: 5}];
  }
}
```

The status bar at the bottom shows: Git, TODO, Problems, Terminal, Services, Externally added files can be added to Git // View Files // Alwa... (16 minutes ago), 20:28 LF UTF-8, TypeScript 5.2.2 2 spaces* main.

Exploitation du service dans le composant Courses

```
File Edit View Navigate Code Refactor Run Tools Git Window Help
courses.component.ts > CoursesComponent > constructor() Angular CLI Server Git:
Project
Commit
Structure
npm
@Component({
  selector: 'app-courses',
  standalone: true,
  imports: [FormsModule],
  templateUrl: './courses.component.html',
  styleUrls: ['./courses.component.scss']
})
export class CoursesComponent implements OnInit {
  titre = 'liste de cours';
  UE! : Course[]; // ici, on n'initialise plus la liste UE

  constructor() {
    // 1ère idée : dans le constructeur, on crée une instance du service et
    // on appelle sa méthode getCourses pour pouvoir remplir notre tableau UE
    const service = new CoursesService();
    this.UE = service.getCourses();
  }

  ngOnInit() {}
  setTitle() { return this.titre; }
}
```

Externally added files can be added to Git // View Files // Alwa... (23 minutes ago) 16:18 LF UTF-8 TypeScript 5.2.2 2 spaces* main

Exploitation du service : bonnes et mauvaises idées

Bonne idée :

- ▶ Faire en sorte que le constructeur connaisse le service

Mauvaises idées :

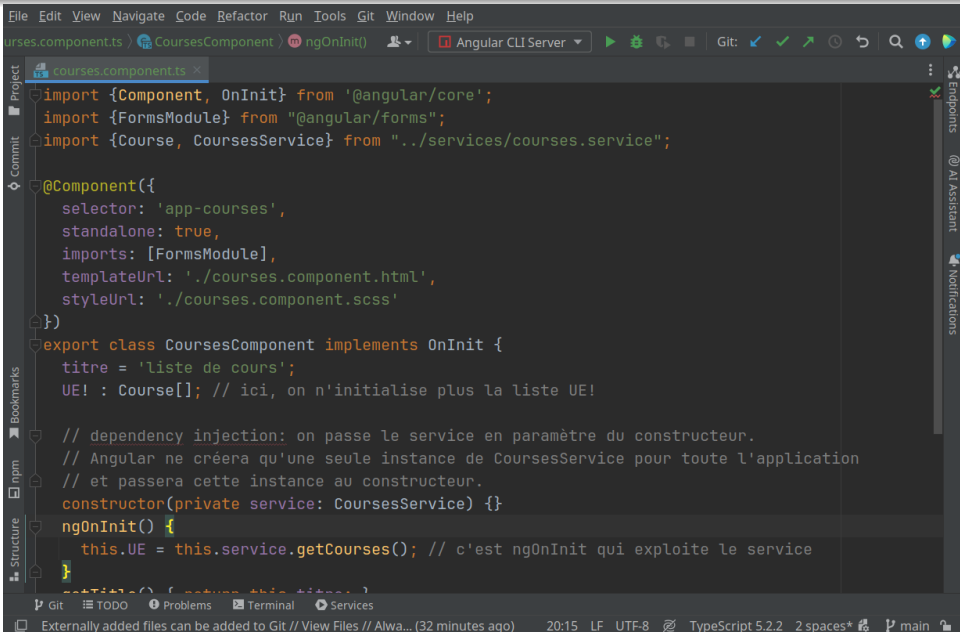
- ▶ Demander au constructeur de créer l'instance
 - ▶ Plusieurs composants \implies plusieurs instances
 - ▶ Modif du constructeur du service \implies modif du composant
- ▶ Utiliser le service dans le constructeur
 \implies délais dans les affichages



Bonne pratique des services :

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

Dependency injection



The screenshot shows an IDE window with the following code:

```
import {Component, OnInit} from '@angular/core';
import {FormsModule} from "@angular/forms";
import {Course, CoursesService} from "../services/courses.service";

@Component({
  selector: 'app-courses',
  standalone: true,
  imports: [FormsModule],
  templateUrl: './courses.component.html',
  styleUrls: ['./courses.component.scss']
})
export class CoursesComponent implements OnInit {
  titre = 'liste de cours';
  UE! : Course[]; // ici, on n'initialise plus la liste UE!

  // dependency injection: on passe le service en paramètre du constructeur.
  // Angular ne créera qu'une seule instance de CoursesService pour toute l'application
  // et passera cette instance au constructeur.
  constructor(private service: CoursesService) {}

  ngOnInit() {
    this.UE = this.service.getCourses(); // c'est ngOnInit qui exploite le service
  }
}
```

The IDE interface includes a menu bar (File, Edit, View, Navigate, Code, Refactor, Run, Tools, Git, Window, Help), a toolbar with icons for running and debugging, and a sidebar with Project, Commit, Bookmarks, and Structure views. The bottom status bar shows: Externally added files can be added to Git // View Files // Alwa... (32 minutes ago) 20:15 LF UTF-8 TypeScript 5.2.2 2 spaces* main

- ▶ Service actuel : synchrone
⇒ `ngOnInit` doit attendre les données
- ▶ Services HTTP : asynchrones
⇒ 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 ⇒ l'observable émet une valeur
⇒ la callback est appelée

Mise en place des observables dans le service

The screenshot shows an IDE window with the following content:

```
File Edit View Navigate Code Refactor Run Tools Git Window Help
courses.service.ts > CoursesService > getCourses() Angular CLI Server
Project
courses.service.ts x
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";

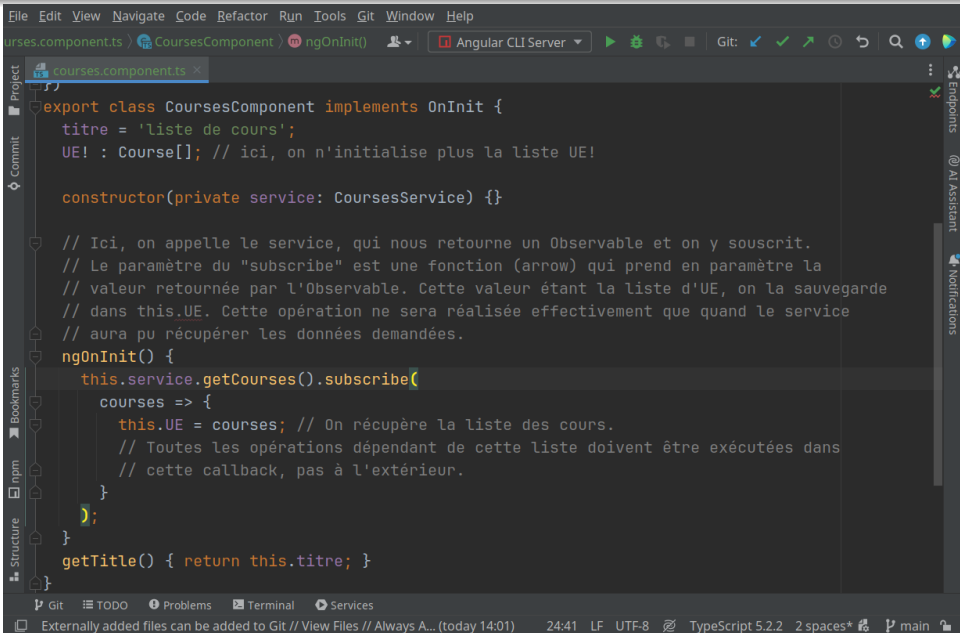
export interface Course {
  nom: string;
  nb_etuds : number;
}

@Injectable({ // le service sera créé par dependency injection
  providedIn: 'root'
})
export class CoursesService {
  constructor() { }

  // Ici, on retourne un Observable sur un tableau de Course
  getCourses (): Observable<Course[]> {
    return of([{nom: 'c1', nb_etuds: 2},
              {nom: 'c2', nb_etuds: 5}]);
  }
}
```

The IDE interface includes a sidebar on the left with 'Project', 'Commit', 'Bookmarks', and 'Structure' views. The bottom status bar shows: 'Externally added files can be added to Git // View Files // Always A... (today 14:01) 18:40 LF UTF-8 TypeScript 5.2.2 2 spaces* main'.

Mise en place des observables dans le composant



The screenshot shows a code editor with the following content:

```
File Edit View Navigate Code Refactor Run Tools Git Window Help
courses.component.ts > CoursesComponent > ngOnInit() Angular CLI Server Git:
Project
Commit
Structure
npm
Bookmarks
Terminal
Services
Endpoints
@AI Assistant
Notifications

export class CoursesComponent implements OnInit {
  titre = 'liste de cours';
  UE! : Course[]; // ici, on n'initialise plus la liste UE!

  constructor(private service: CoursesService) {}

  // Ici, on appelle le service, qui nous retourne un Observable et on y souscrit.
  // Le paramètre du "subscribe" est une fonction (arrow) qui prend en paramètre la
  // valeur retournée par l'Observable. Cette valeur étant la liste d'UE, on la sauvegarde
  // dans this.UE. Cette opération ne sera réalisée effectivement que quand le service
  // aura pu récupérer les données demandées.
  ngOnInit() {
    this.service.getCourses().subscribe(
      courses => {
        this.UE = courses; // On récupère la liste des cours.
        // Toutes les opérations dépendant de cette liste doivent être exécutées dans
        // cette callback, pas à l'extérieur.
      }
    );
  }

  getTitle() { return this.titre; }
}
```

Externally added files can be added to Git // View Files // Always A... (today 14:01) 24:41 LF UTF-8 TypeScript 5.2.2 2 spaces* main

Service HTTP : 1 importer le HttpClientModule

The screenshot shows an IDE window with the following components:

- File Explorer (Left):** Displays the project structure. The file `app.config.ts` is selected and highlighted in blue. Other files include `courses.component.ts`, `courses.service.ts`, `app.component.html`, `app.component.spec.ts`, `app.routes.ts`, `assets`, `favicon.ico`, `index.html`, `main.ts`, `styles.scss`, `.editorconfig`, `.gitignore`, `angular.json`, `getCourses.php`, `package.json`, `package-lock.json`, and `README.md`.
- Code Editor (Center):** Shows the content of `app.config.ts`. The code is as follows:

```
1 import {ApplicationConfig, importProvidersFrom} from '@angular/core';
2 import { provideRouter } from '@angular/router';
3
4 import { routes } from './app.routes';
5 import {provideHttpClient} from "@angular/common/http";
6
7 export const appConfig: ApplicationConfig = {
8   providers: [
9     provideRouter(routes),
10    // ici, il faut déclarer que notre application va utiliser
11    // le module HttpClientModule, qui contient la classe
12    // HttpClient, qui réalisera les échanges avec le backend
13    provideHttpClient()
14  ]
15 };
16
```
- Terminal (Bottom):** Shows the status of the application, including the Angular CLI Server, Git status, and other system information.

Service HTTP : 2 notre service utilise HttpClient

```
File Edit View Navigate Code Refactor Run Tools Git Window Help
courses.service.ts > CoursesService > getCourses() Angular CLI Server
courses.service.ts x
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, of} from "rxjs";
import {HttpClient} from "@angular/common/http";

export interface Course {
  nom: string;
  nb_etuds : number;
}

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

  // Ici, on retourne un Observable sur un tableau de Course
  getCourses (): Observable<Course[]> {
    return this.http.get<Course[]>( // on converse avec le backend via une méthode GET
      'http://127.0.0.1/xmobile_cours/getCourses.php'
    );
  }
}
```

Git TODO Problems Terminal Services

Externally added files can be added to Git // View Files // Always A... (today 14:01) 19:40 LF UTF-8 TypeScript 5.2.2 2 spaces* main

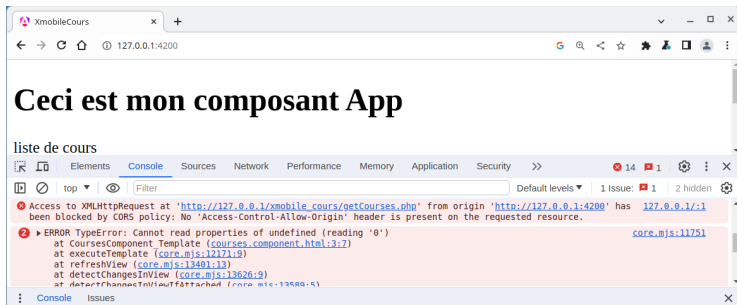
Service HTTP : 3 getCourses.php

```
<?php

// on indique ici que le script va transmettre des données JSON
header('Content-type:application/json;charset=utf8');

echo json_encode ([
    [ 'nom' => 'c1', 'nb_etuds' => 2 ],
    [ 'nom' => 'c2', 'nb_etuds' => 5 ],
    [ 'nom' => 'c3', 'nb_etuds' => 6 ]
]);

?>
```

► Cross-Origin Resource Sharing (CORS) :

- Requête cross-origine : provient de 127.0.0.1:4200
accède à 127.0.0.1:80

⇒ CORS refuse la requête

- **Contre-mesure** : dans `getCourses.php`, rajouter :

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

Service HTTP : 3 le bon getCourses.php

```
File Edit View Navigate Code Refactor Run Tools Git Window Help
xmobile_cours / getCourses.php Angular CLI Server
getCourses.php x
<?php

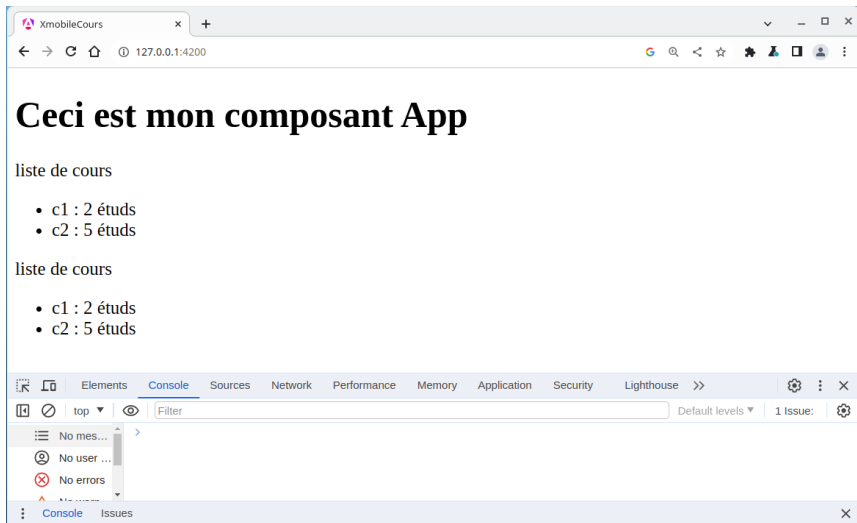
// on indique ici que le script va transmettre des données JSON
header('Content-type:application/json;charset=utf8');

// Ici, on ajoute un header pour contourner le problème de CORS.
// Notez que l'on doit le faire, ici, parce que le frontend Angular est servi sur
// le port 4200 alors que le backend PHP est servi par Apache sur le port 80.
header("Access-Control-Allow-Origin: " . $_SERVER['HTTP_ORIGIN']);

echo json_encode ([
    [ 'nom' => 'c1', 'nb_etuds' => 2 ],
    [ 'nom' => 'c2', 'nb_etuds' => 5 ],
    [ 'nom' => 'c3', 'nb_etuds' => 6 ]
]);

?>
```

Service HTTP : le bon résultat



XmobileCours

127.0.0.1:4200

Ceci est mon composant App

liste de cours

- c1 : 2 étuds
- c2 : 5 étuds

liste de cours

- c1 : 2 étuds
- c2 : 5 étuds

Elements Console Sources Network Performance Memory Application Security Lighthouse >>

top Filter Default levels 1 Issue

No mes...
No user ...
No errors

Console Issues

Service HTTP : 2 HttpClient avec méthode POST

```
File Edit View Navigate Code Refactor Run Tools Git Window Help
courses.service.ts > CoursesService > getCourses() Angular CLI Server
courses.service.ts x
import {Observable, of} from "rxjs";
import {HttpClient} from "@angular/common/http";

export interface Course {
  nom: string;
  nb_etuds : number;
}

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

  // Ici, on retourne un Observable sur un tableau de Course
  getCourses (): Observable<Course[]> {
    return this.http.post<Course[]>( // on converse avec le backend via une méthode GET
      'http://127.0.0.1/xmobile_cours/getCourses.php', // url du backend
      null // les données qu'on envoie par POST
    );
  }
}
```

Service HTTP : 3 méthode POST et session

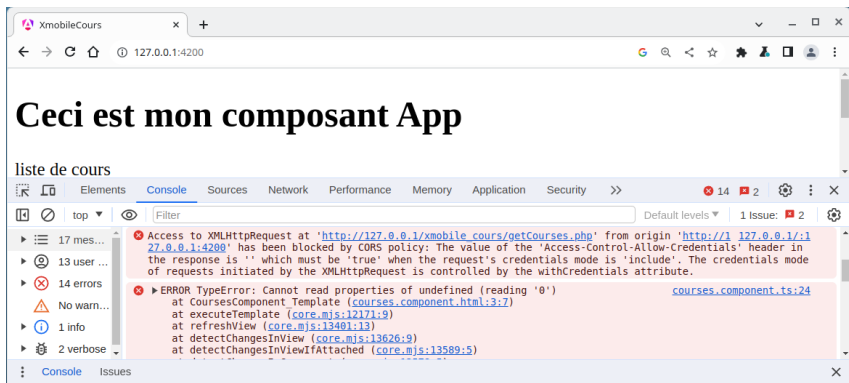
```
File Edit View Navigate Code Refactor Run Tools Git Window Help
courses.service.ts > CoursesService > getCourses() Angular CLI Server
courses.service.ts x
import {Observable, of} from "rxjs";
import {HttpClient} from "@angular/common/http";

export interface Course {
  nom: string;
  nb_etuds : number;
}

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

  // Ici, on retourne un Observable sur un tableau de Course
  getCourses (): Observable<Course[]> {
    return this.http.post<Course[]>( // on converse avec le backend via une méthode GET
      'http://127.0.0.1/xmobile_cours/getCourses.php', // url du backend
      null, // les données qu'on envoie par POST
      {withCredentials: true} // pour capturer les cookies de session
    );
  }
}
```

Service HTTP : 4 problème et solution



The screenshot shows a web browser window with the URL `127.0.0.1:4200`. The page title is "Ceci est mon composant App" and the content is "liste de cours". The browser's developer tools are open to the Console tab, showing two error messages:

- Access to XMLHttpRequest at 'http://127.0.0.1/xmobile_cours/getCourses.php' from origin 'http://127.0.0.1:4200' has been blocked by CORS policy: The value of the 'Access-Control-Allow-Credentials' header in the response is '' which must be 'true' when the request's credentials mode is 'include'. The credentials mode of requests initiated by the XMLHttpRequest is controlled by the withCredentials attribute.
- ERROR TypeError: Cannot read properties of undefined (reading '0')
at CoursesComponent Template (courses.component.html:3:7)
at executeTemplate (core.mjs:12171:9)
at refreshView (core.mjs:13401:13)
at detectChangesInView (core.mjs:13626:9)
at detectChangesInViewIfAttached (core.mjs:13589:5)

► **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.config.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`