Cours 2 : Frontend

Web — Applications web et mobile

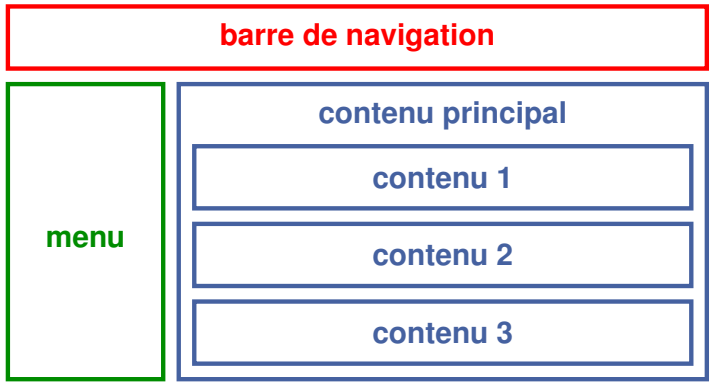
Christophe Gonzales

Qu'est-ce qu'Angular

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

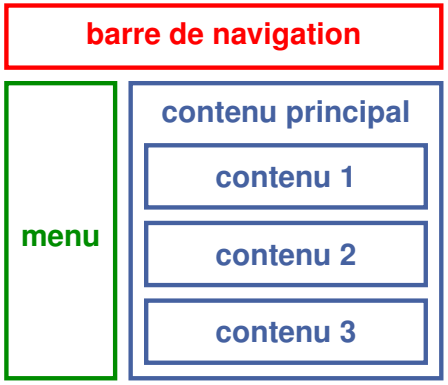
Structure d'une appplication Angular

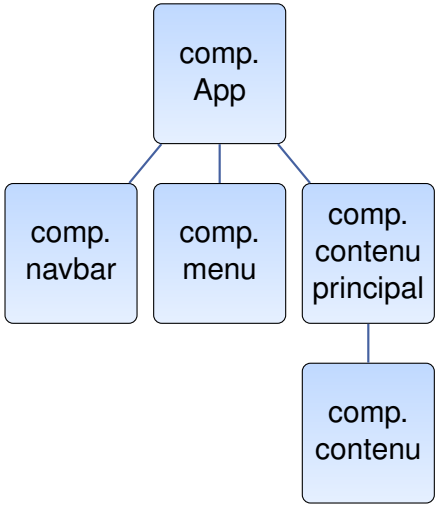
Affichage de la page web :



- Affichage $\Longrightarrow$ 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





$\Longrightarrow$ permet de structurer facilement le code !

Contenu d'un composant Angular

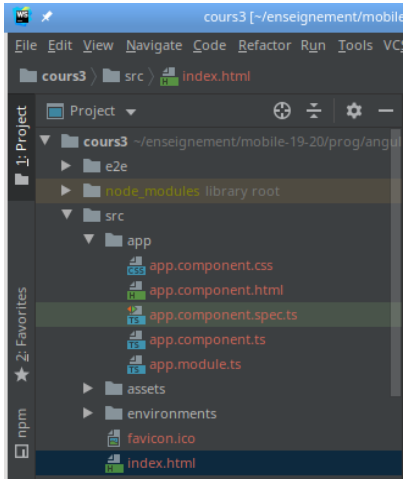
Un composant Angular contient essentiellement :

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

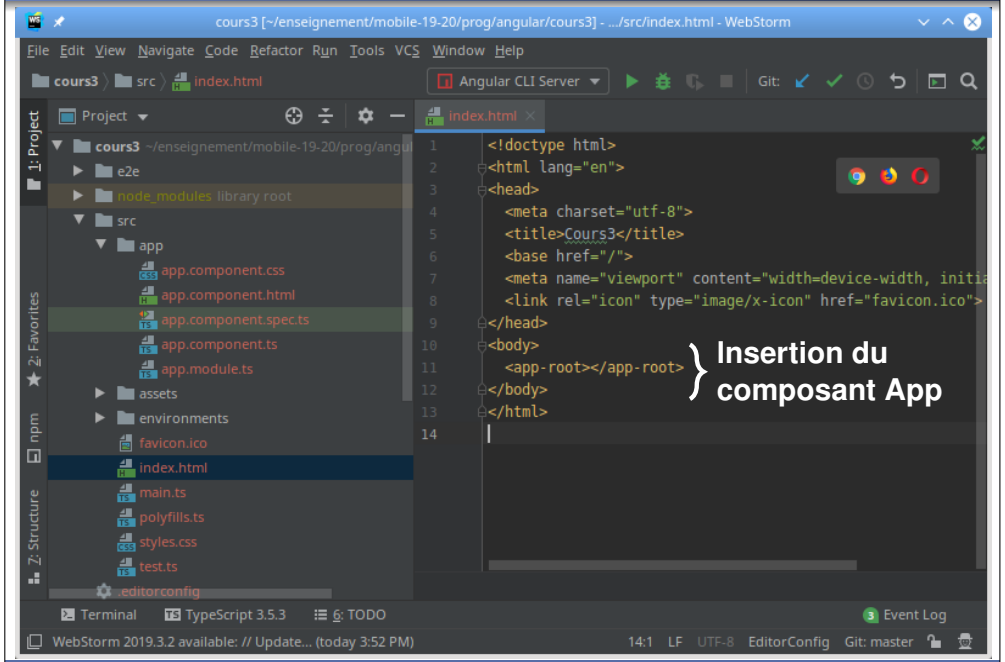
Génération d'un projet Angular

- `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



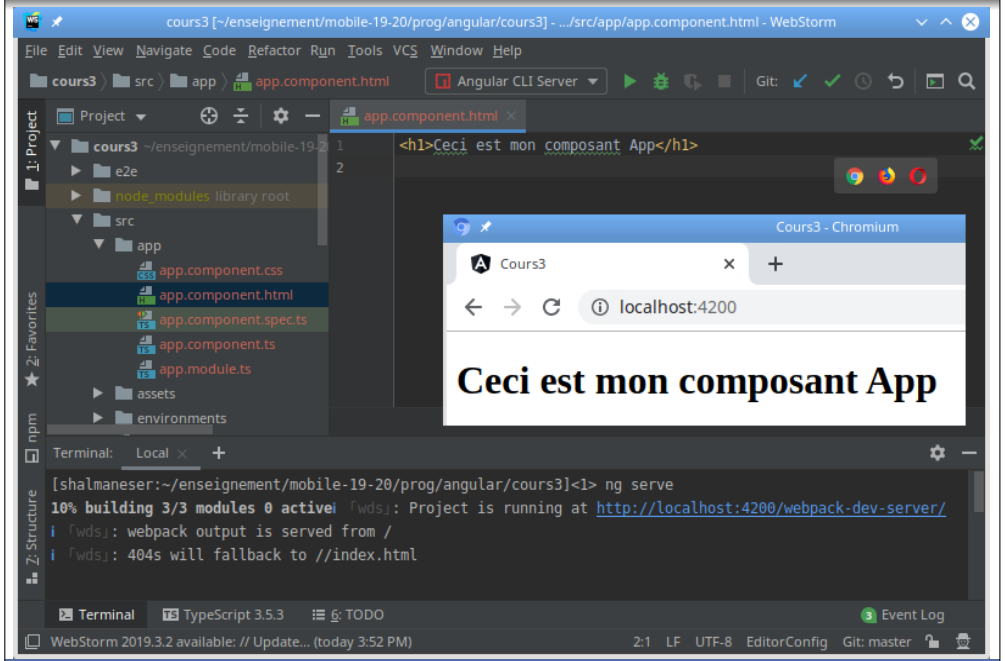
Fonctionnement d'un projet Angular

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>`

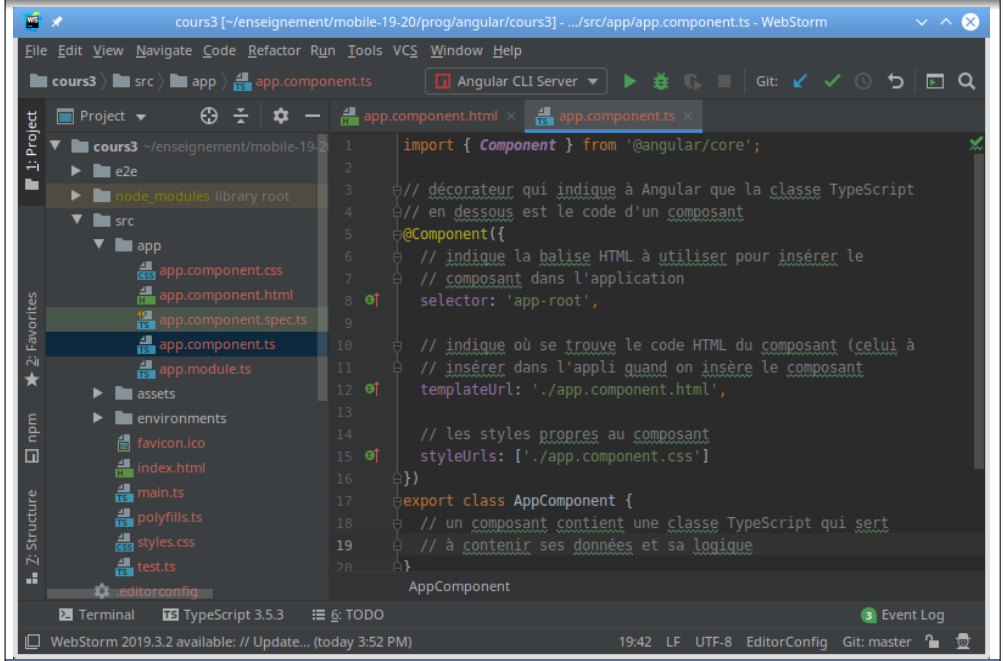
- Pour compiler et « servir » votre application :

```
ng serve
```

Le composant App et l'appli servie



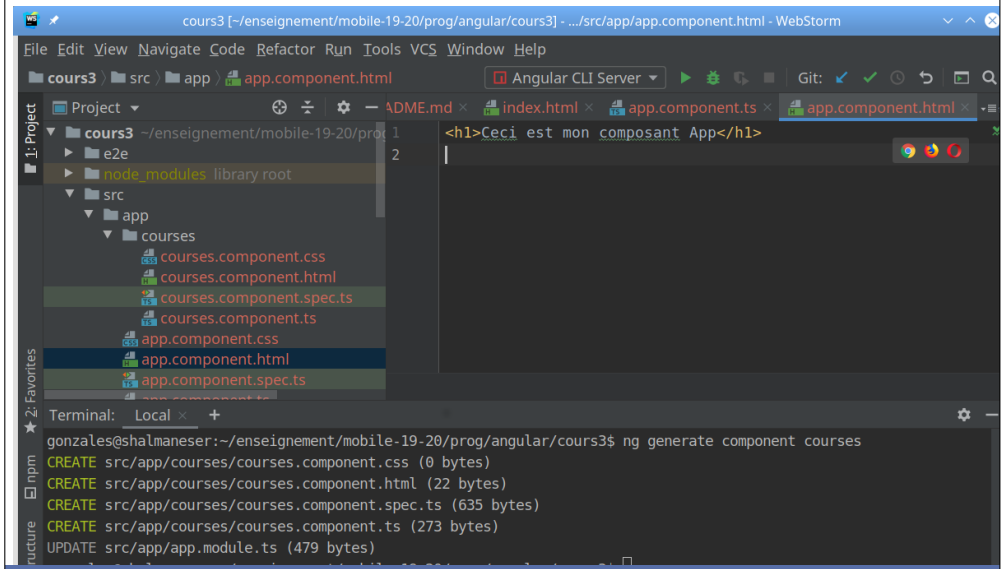
Le TypeScript du composant app



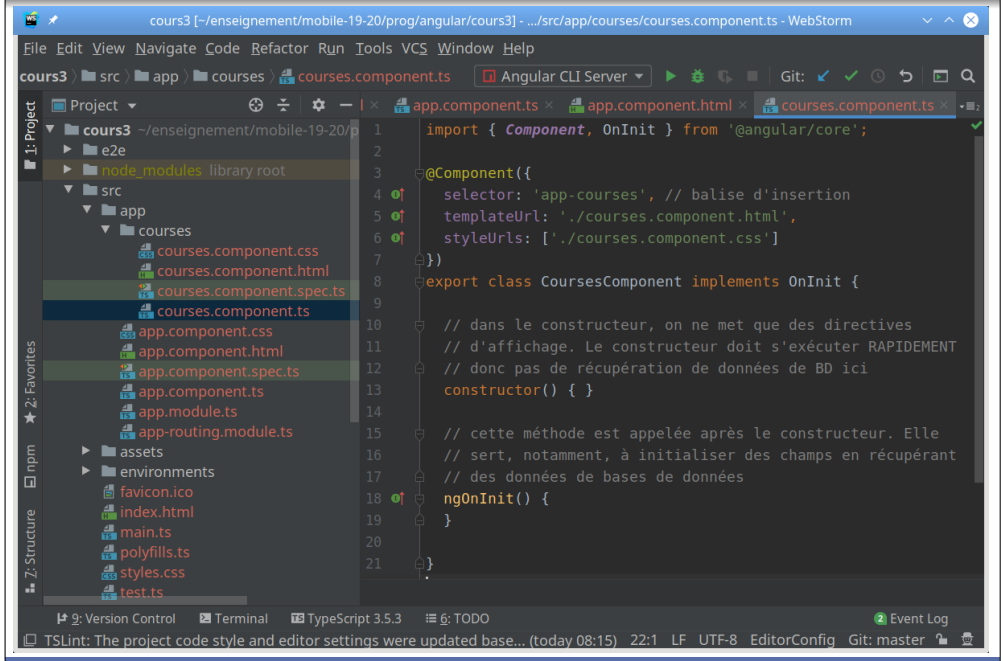
Création d'un nouveau composant

- Utiliser la commande `ng generate component courses`

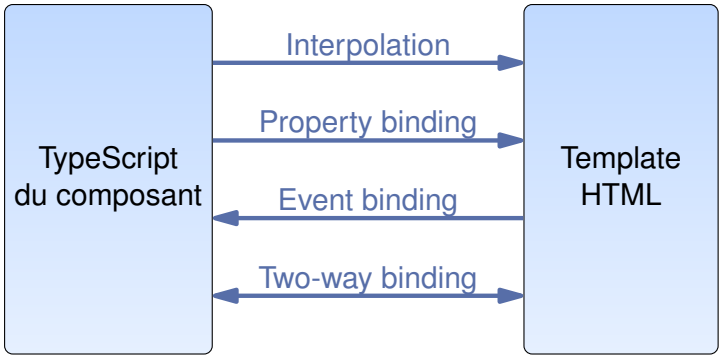
  $\Longrightarrow$ crée un répertoire `courses` et des fichiers spécifiques



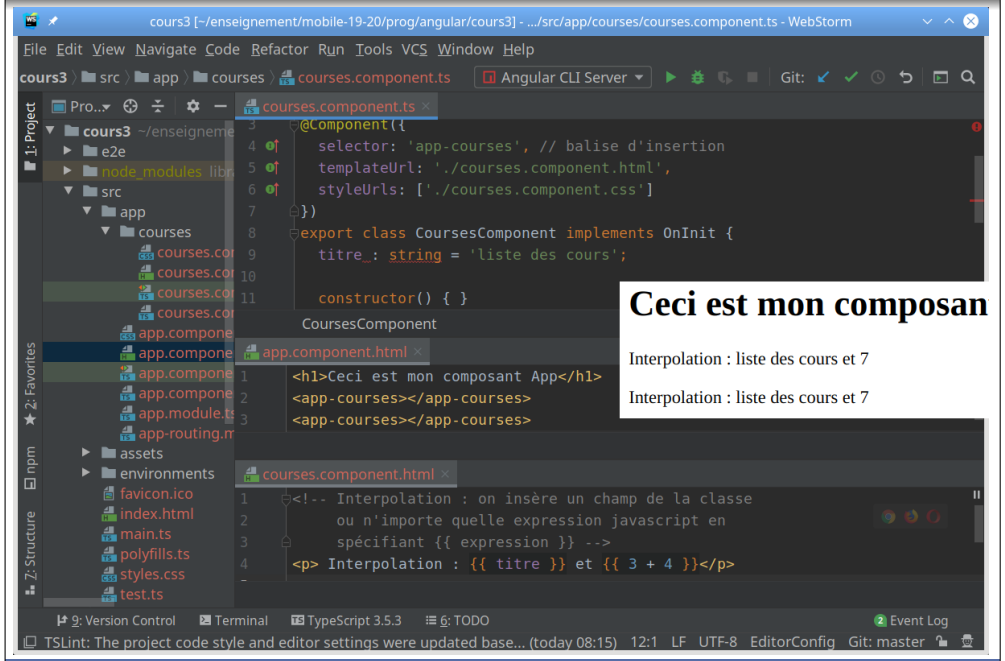
TypeScript du nouveau composant



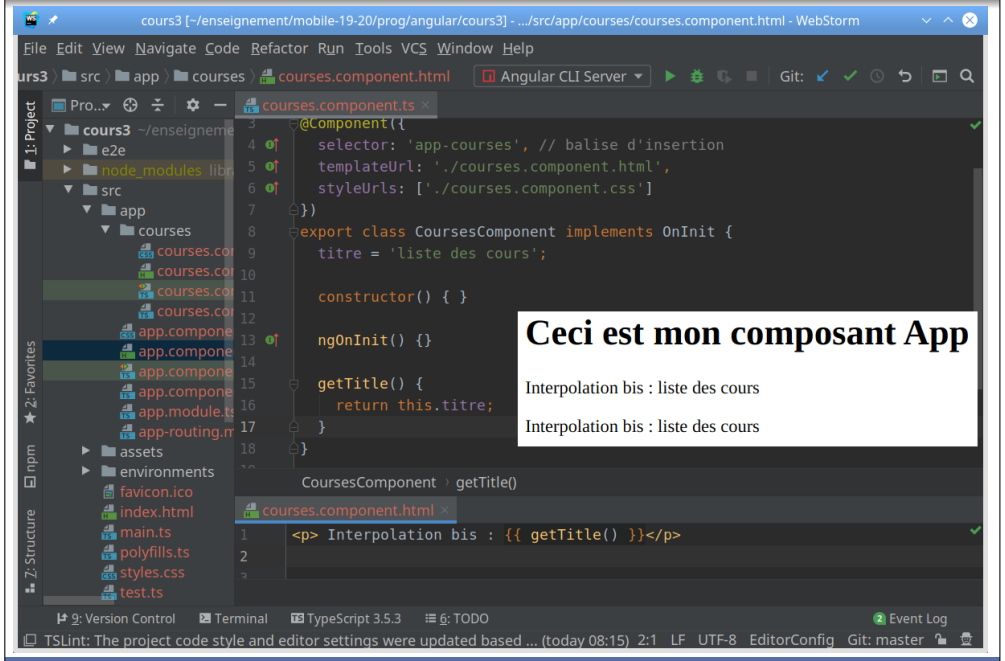
Interactions entre le TypeScript et le HTML



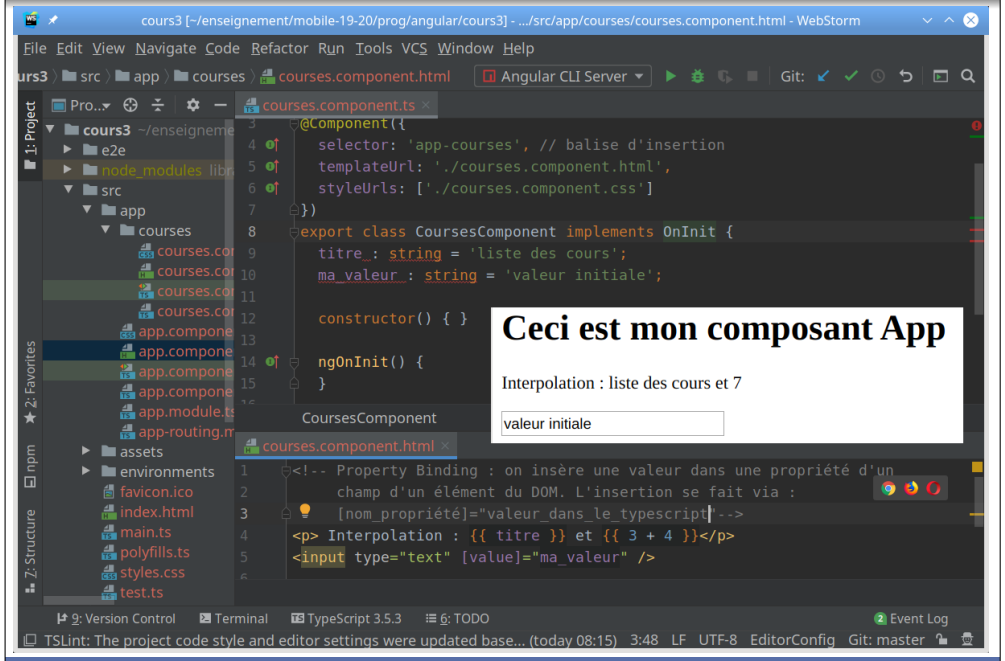
Interpolation



Interpolation avec des méthodes



Property binding



Différence entre interpolation et property binding

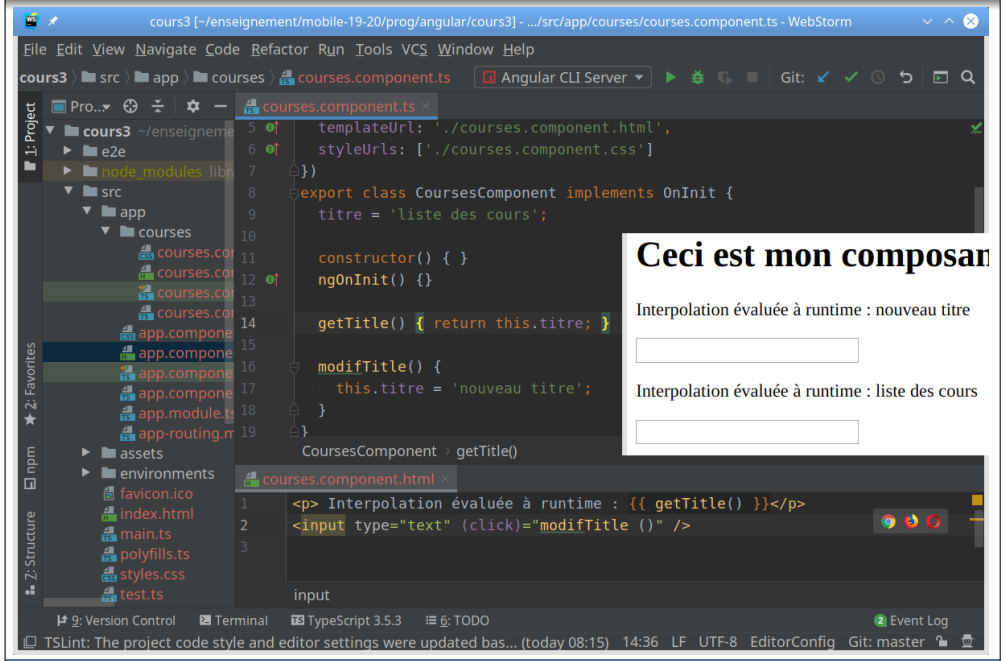
- **Interpolation {{}} :**
  Permet de transférer des données du TypeScript n'importe où dans le template HTML

  Évalué à runtime !

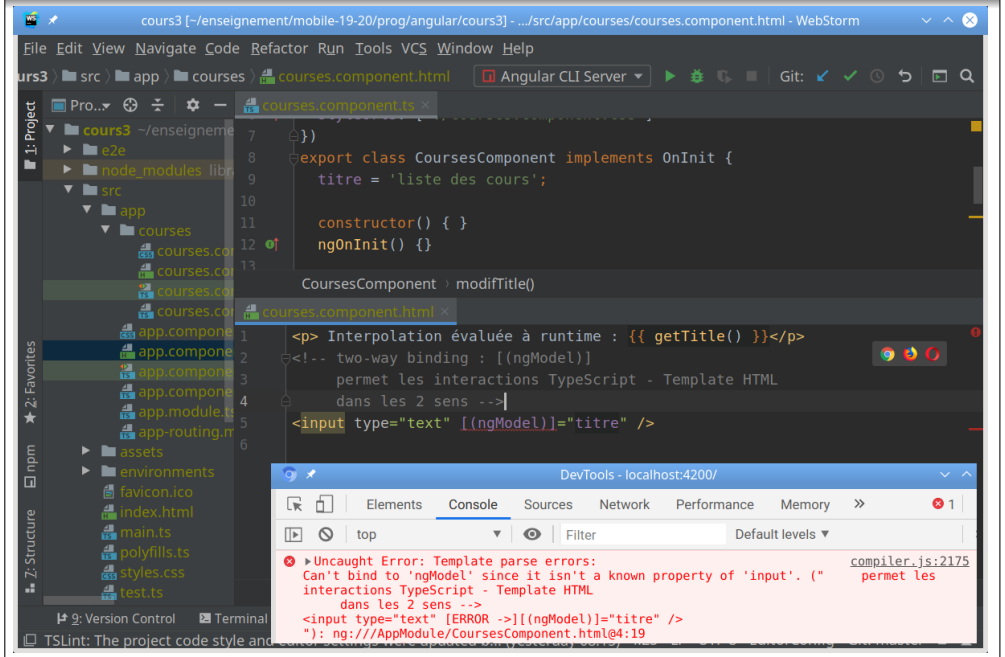
- **Property binding [] :**
  Permet de mettre des valeurs dans les propriétés des éléments du DOM

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

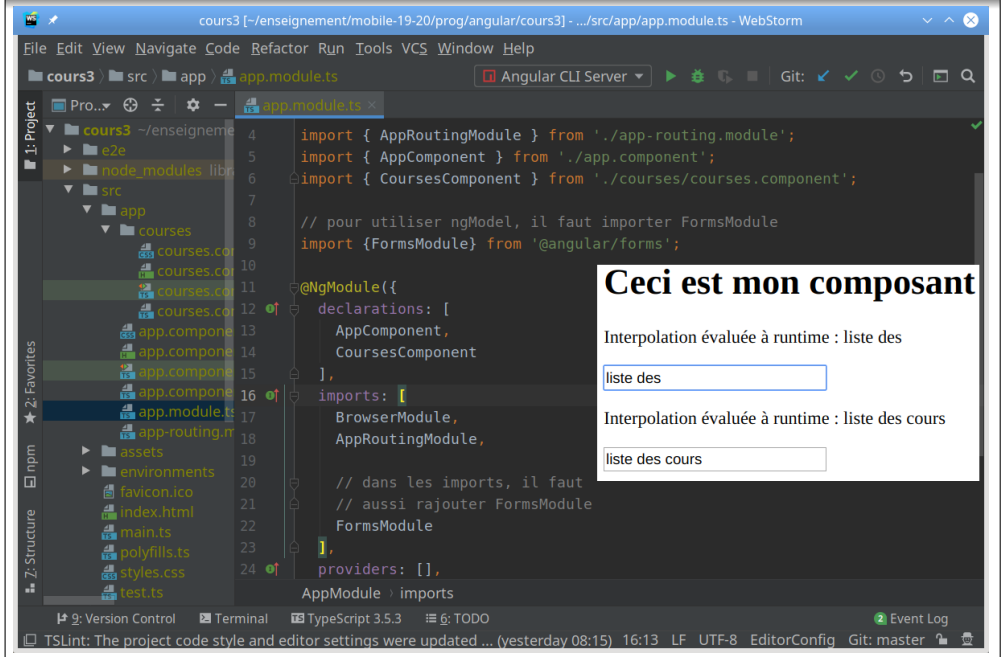
Event binding



Two-way binding (1/2)



Two-way binding (2/2)



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 :

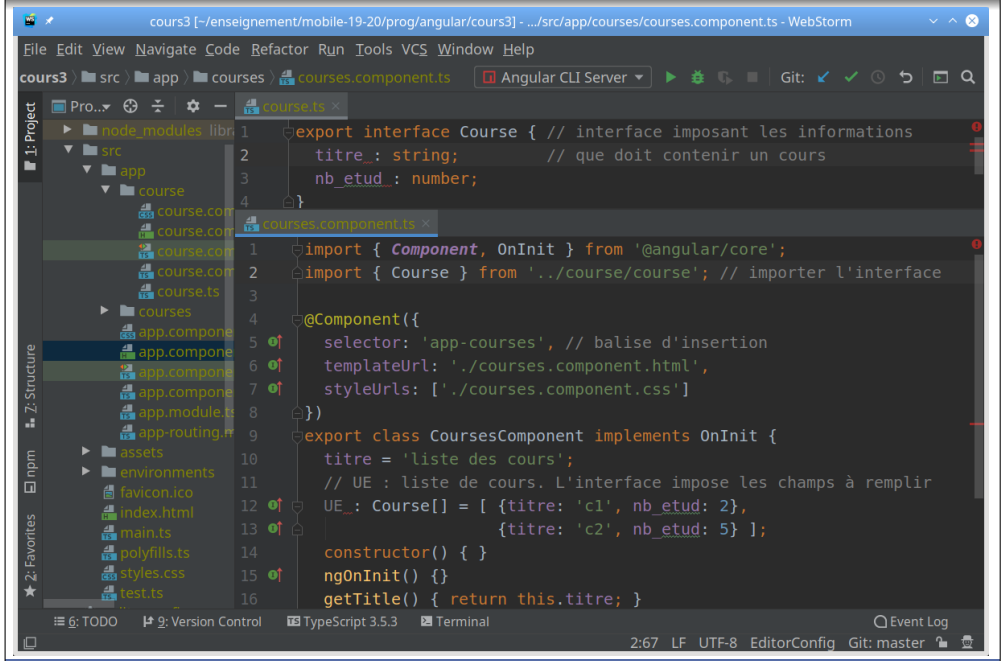
- Event binding : (event)=”méthode()”
  Appelle la méthode quand un événement du DOM arrive

Dans les deux sens :

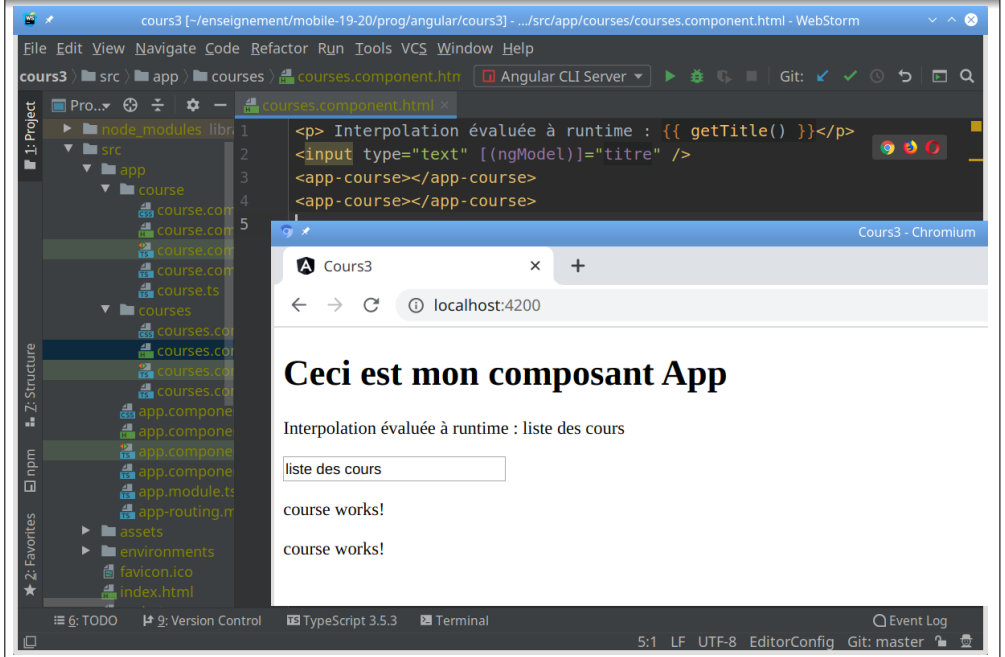
- Two-way binding : [(ngModel)]=”valeur”

⚠ ne pas oublier d'importer FormsModule dans app.module.ts

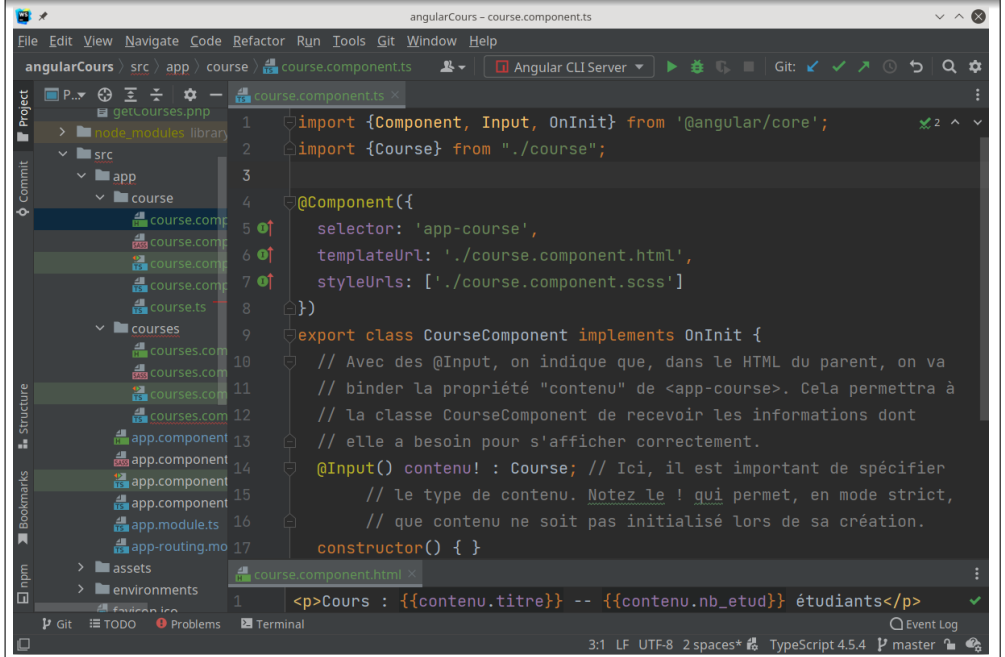
Rajout d'un composant Course enfant de Courses (1/2)



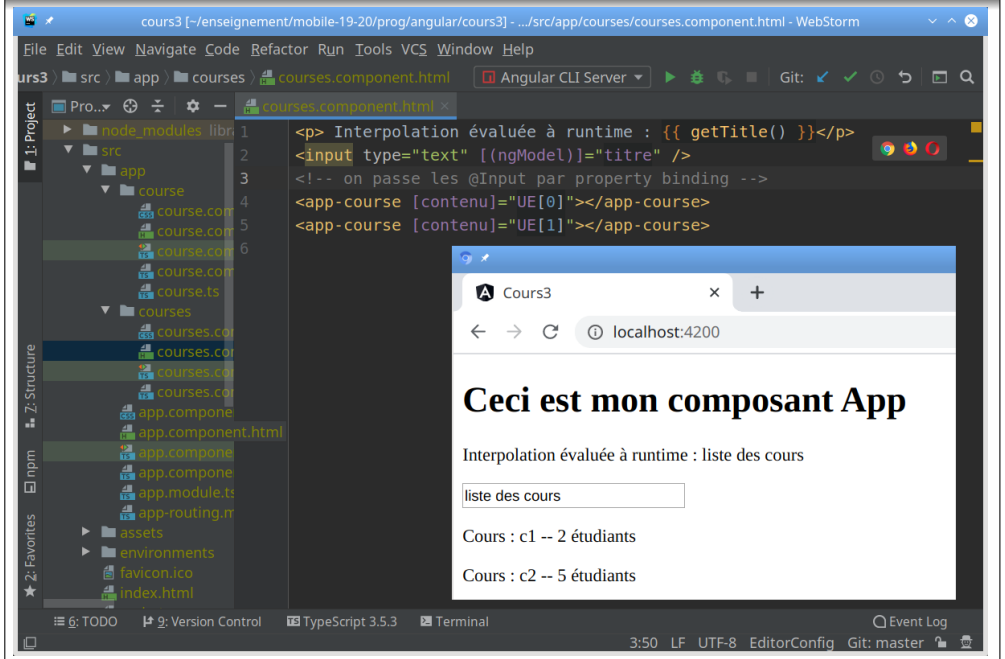
Rajout d'un composant Course enfant de Courses (2/2)



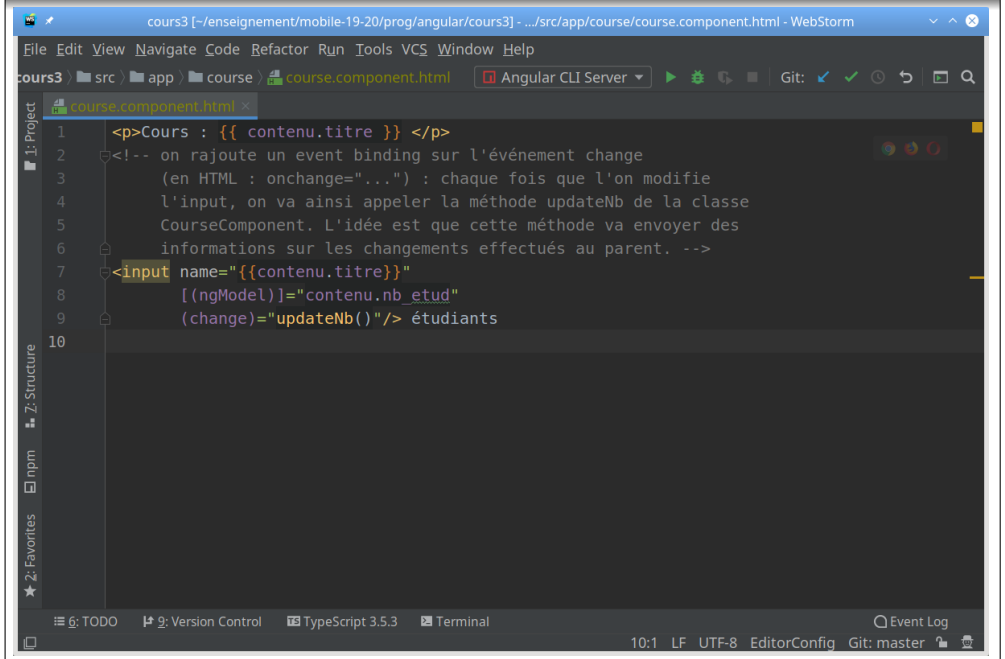
Passer des paramètres au constructeur de l'enfant (1/2)



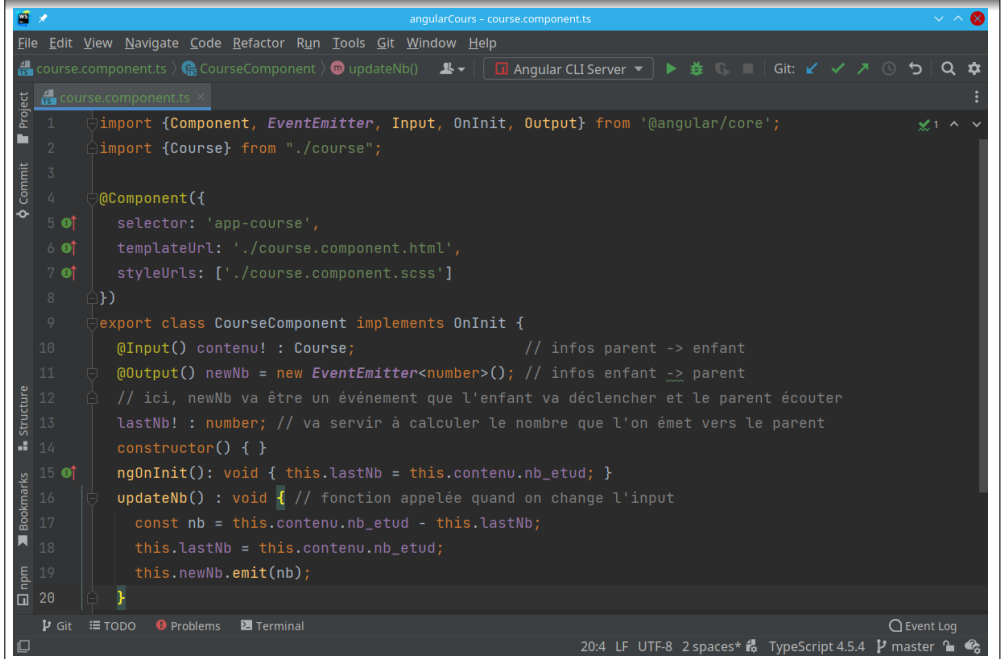
Passer des paramètres au constructeur de l'enfant (2/2)



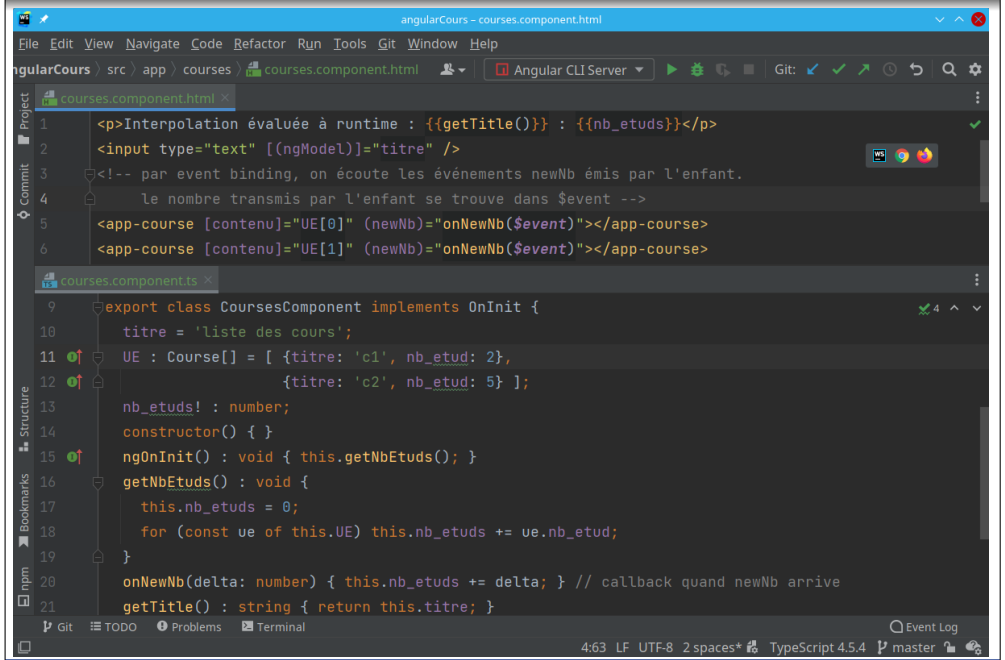
Interactions enfant → parent : préparation de l'enfant



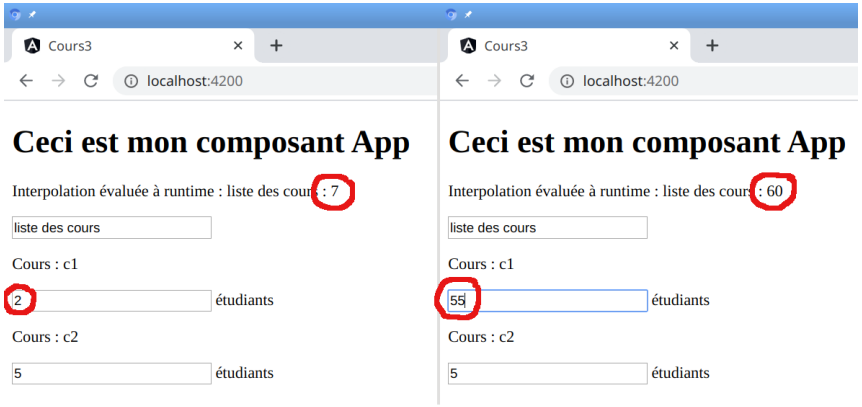
Interactions enfant → parent : l'émetteur de l'enfant



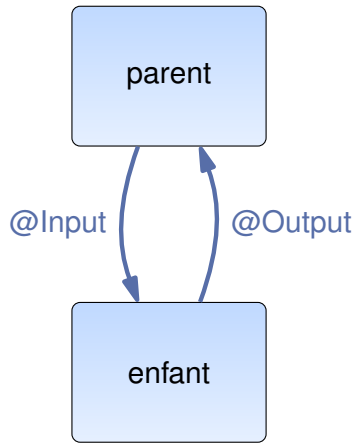
Interactions enfant → parent : la réception du parent



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 :

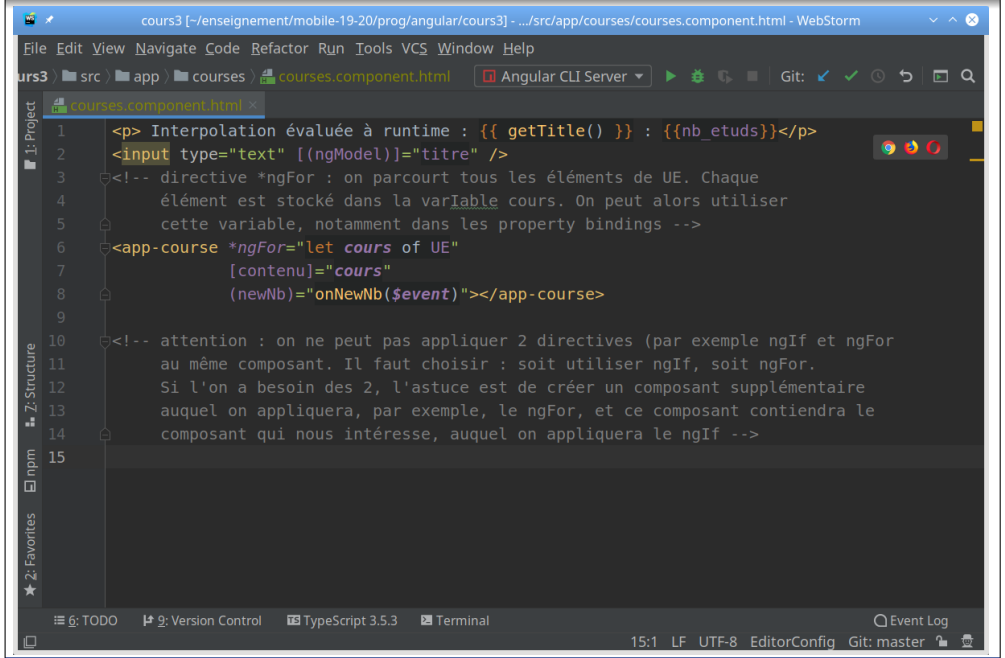
1. Créer un composant Cours
2. Stocker la liste des cours dans le composant Courses
3. Dans le template de Courses, itérer l'insertion des cours avec la directive *ngFor

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

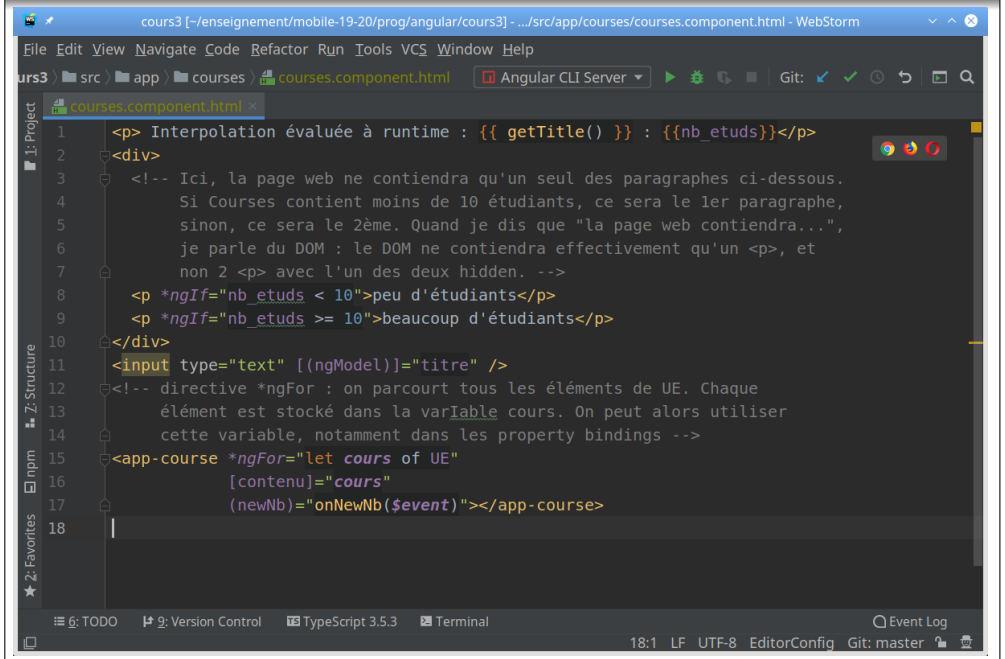
⚠ Peut nécessiter l'inclusion de CommonModule dans app.module.ts

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

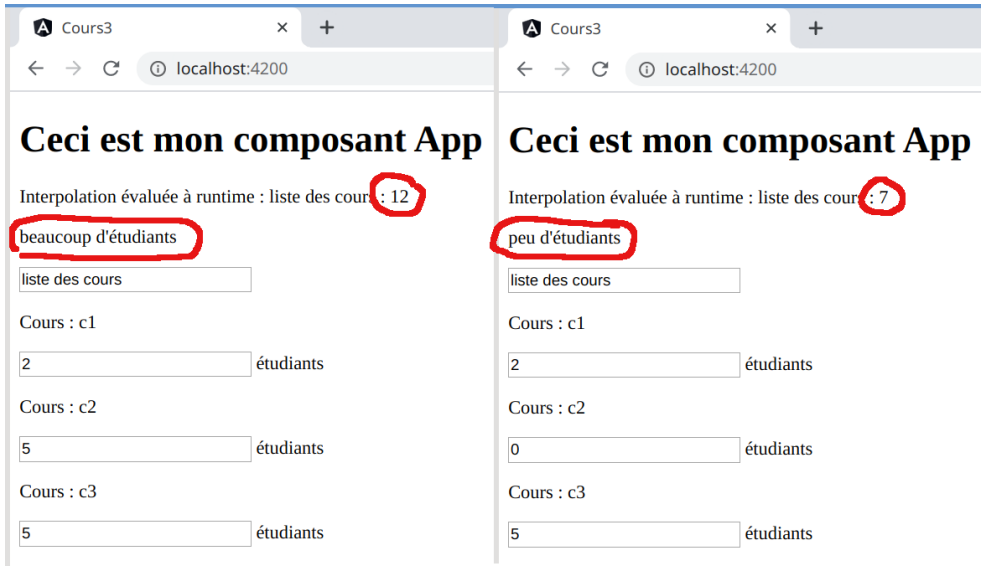
La directive *ngFor



La directive *ngIf



La directive *ngIf : le résultat



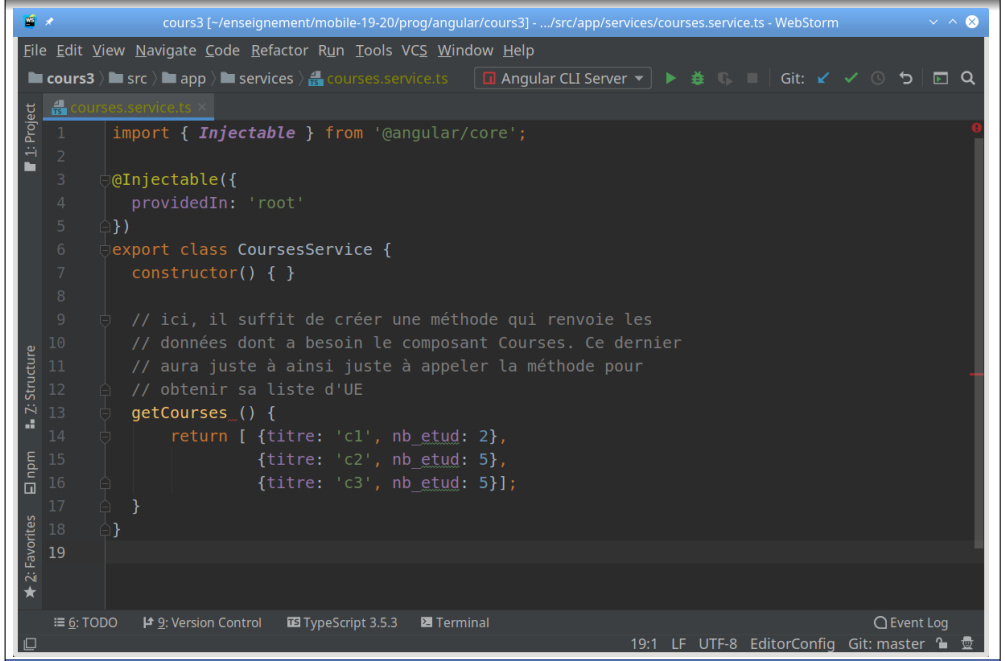
Les services : providers de données

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

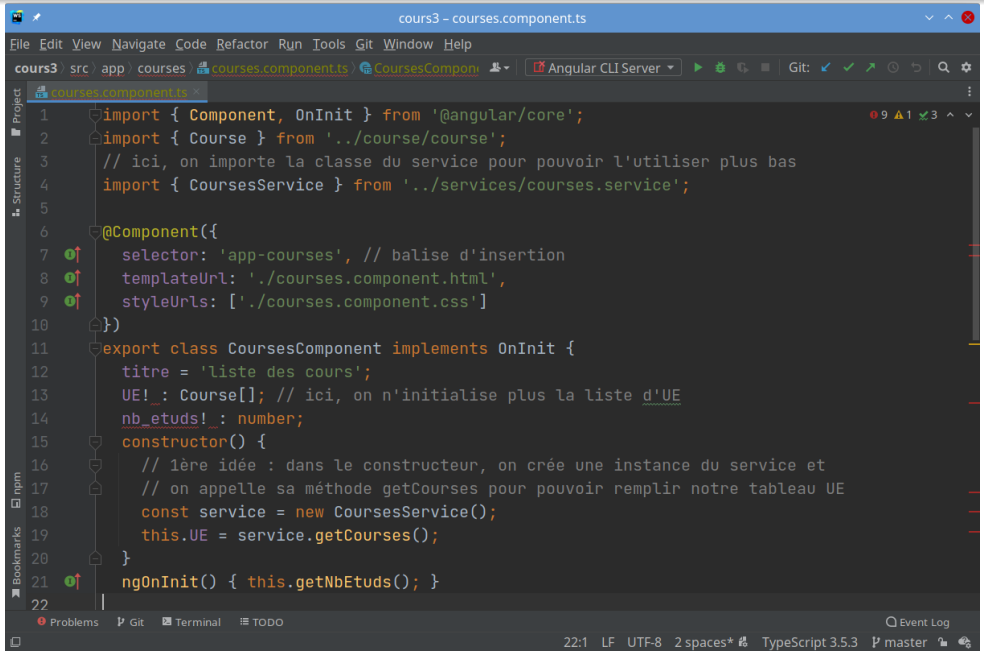
: les services récupèrent les données

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

Anatomie d'un service



Exploitation du service dans le composant Courses



Exploitation du service : bonnes et mauvaises idées

Bonne idée :

- Faire en sorte que le constructeur connaisse le service

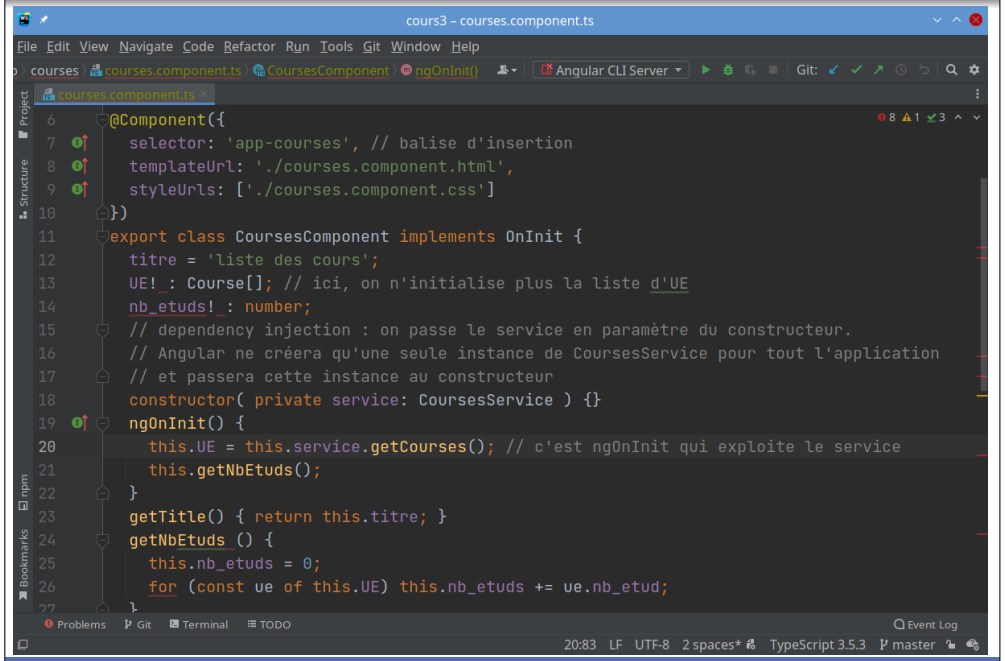
Mauvaises idées :

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

Bonne pratique des services :

- Dependency injection
- Utiliser le service dans méthode ngOnInit

Dependency injection



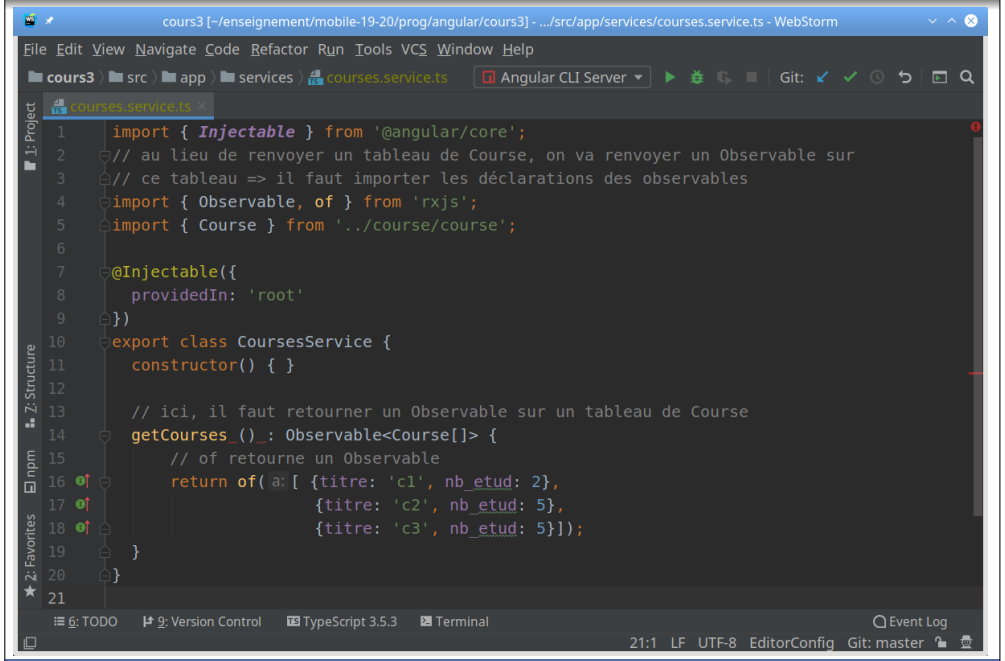
Asynchronie et observables

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

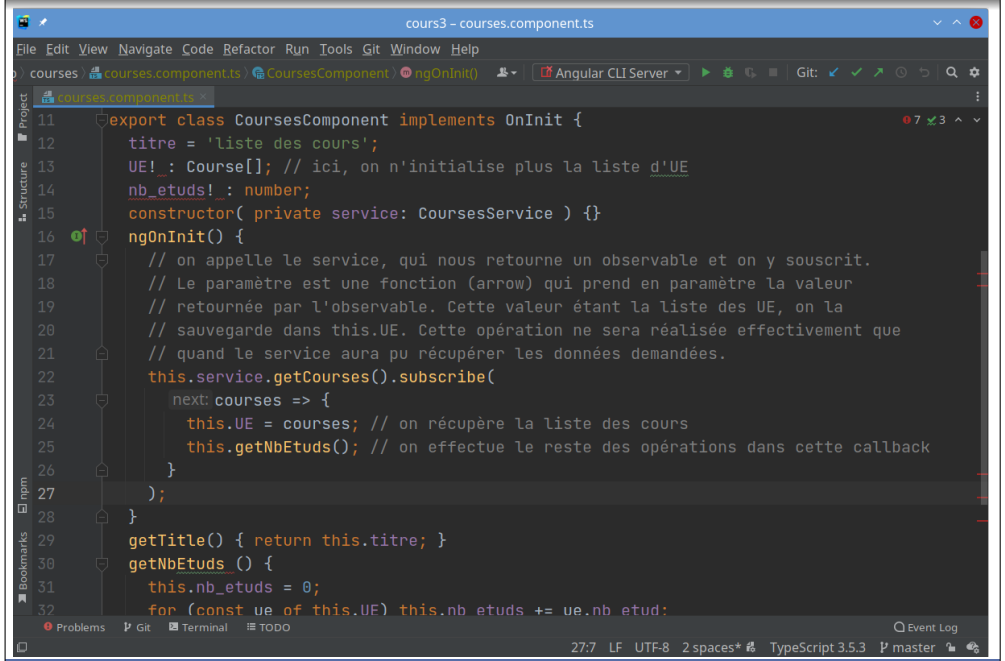
Utilisation de services asynchrones :

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 $\Longrightarrow$ l'observable émet une valeur
   $\Longrightarrow$ la callback est appelée

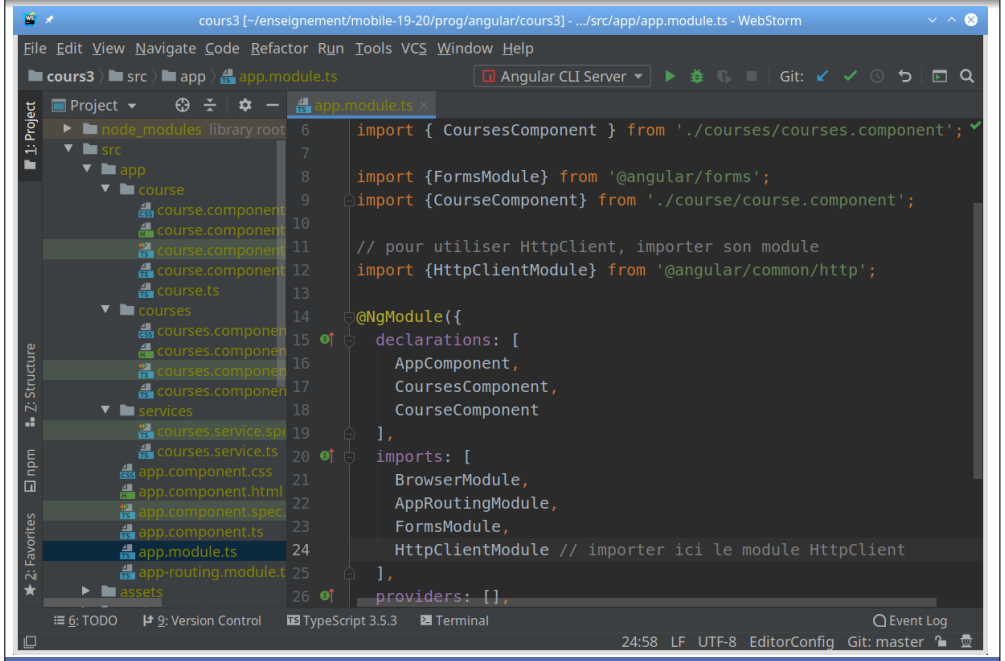
Mise en place des observables dans le service



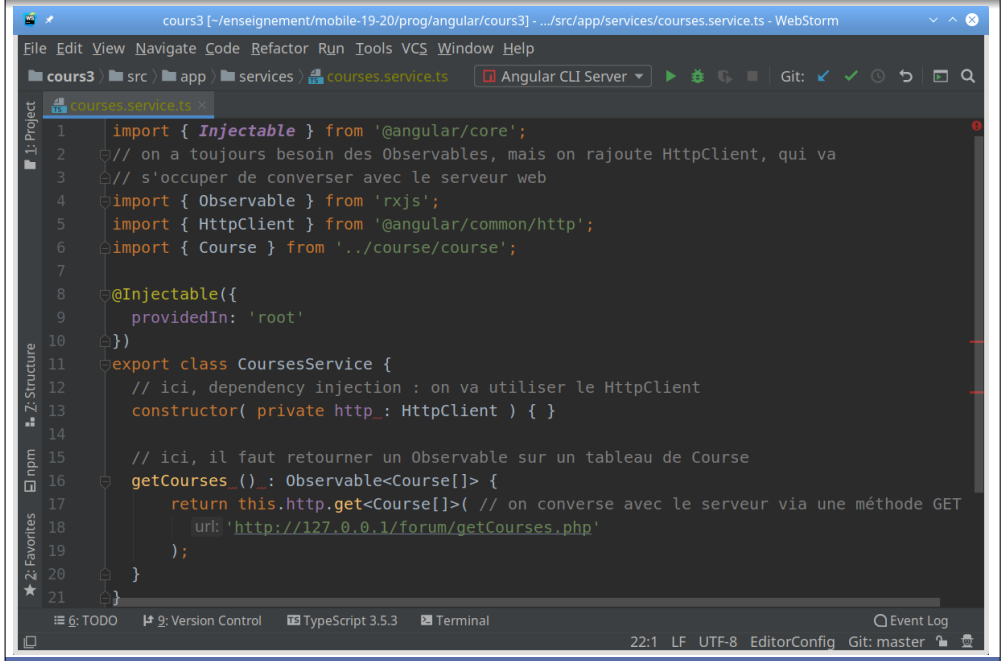
Mise en place des observables dans le composant



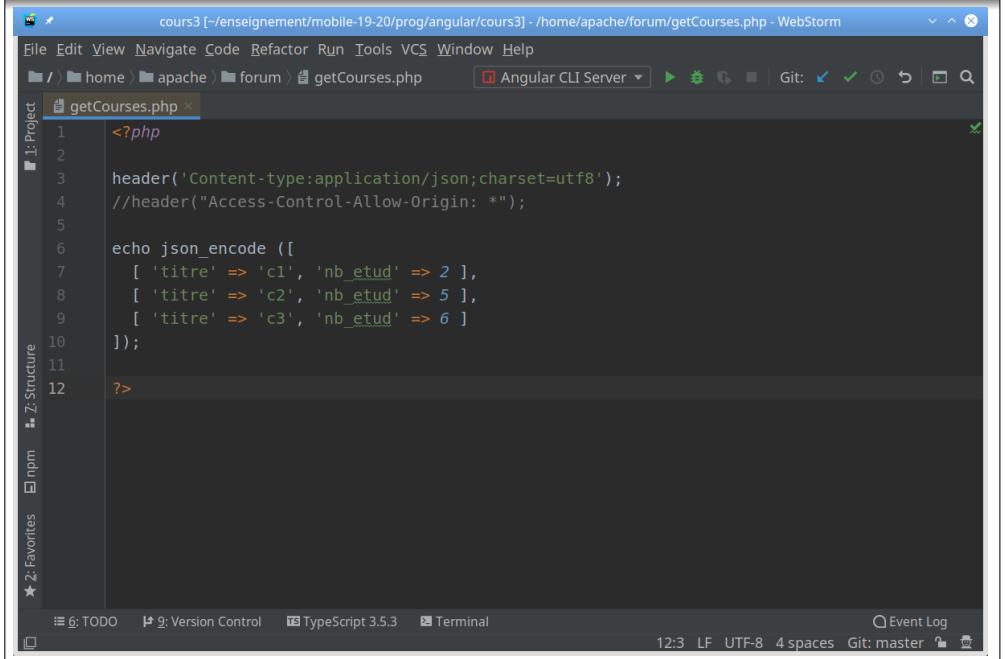
Service HTTP : 1 importer le HttpClientModule



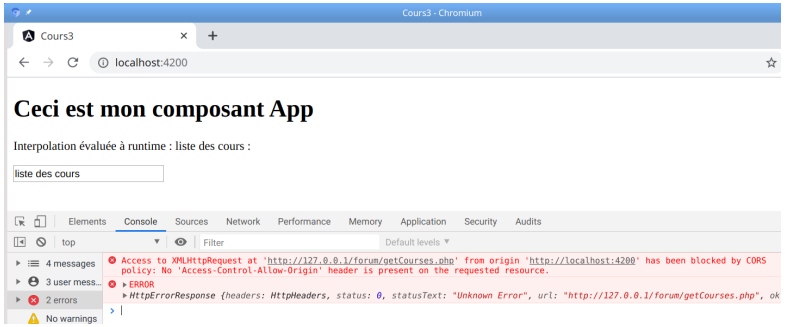
Service HTTP : 2 notre service utilise HttpClient



Service HTTP : 3 getCourses.php



Service HTTP : Résultat

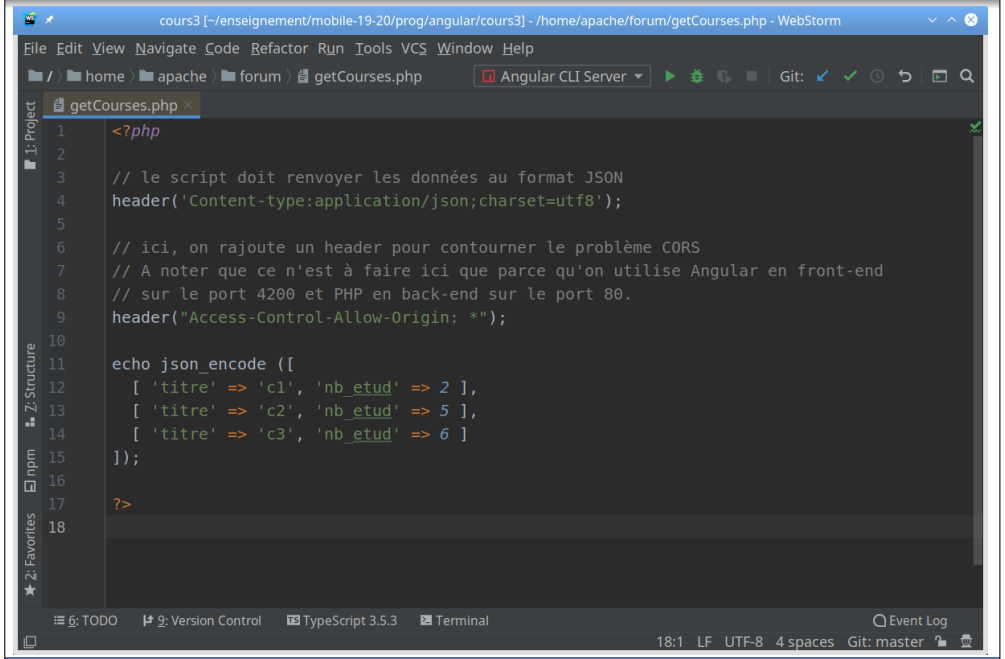


- **Cross-Origin Resource Sharing (CORS) :**
- Requête cross-origine : provient de localhost :4200
  accède à localhost :80

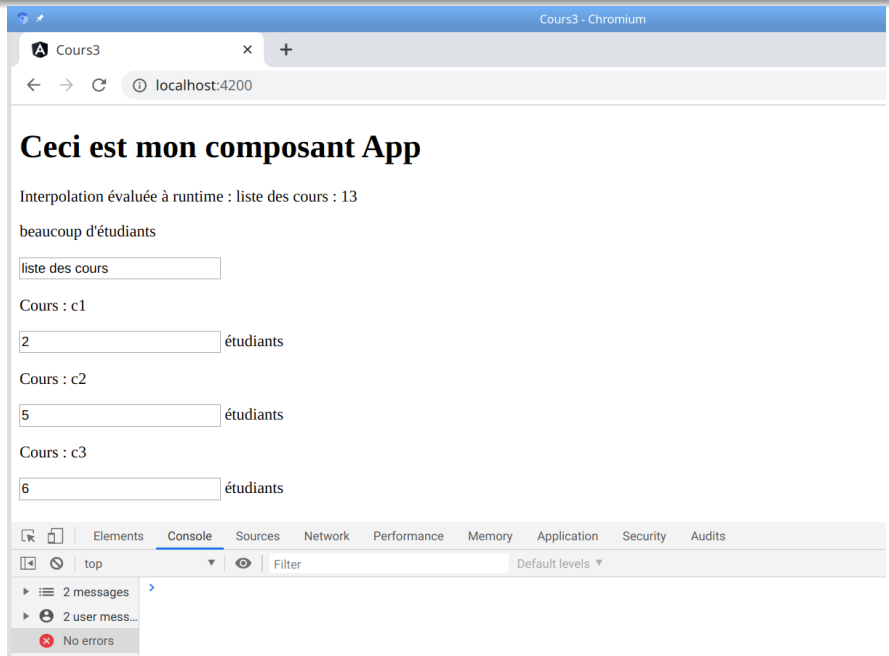
  $\Longrightarrow$ CORS refuse la requête
- **Contre-mesure :** dans getCourses.php, rajouter :

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

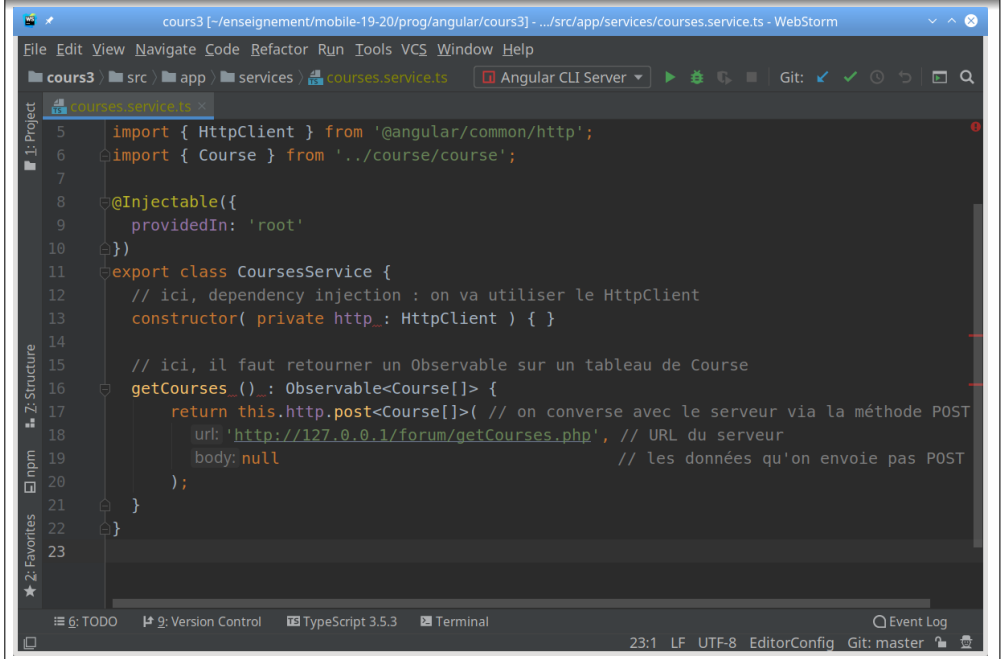

Service HTTP : 3 le bon getCourses.php



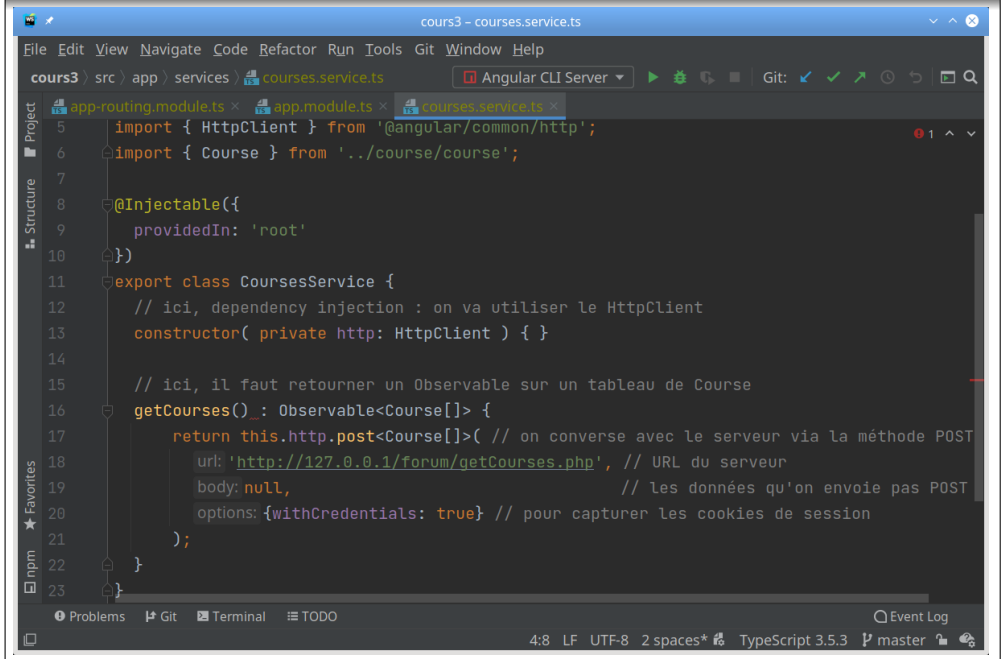
Service HTTP : le bon résultat



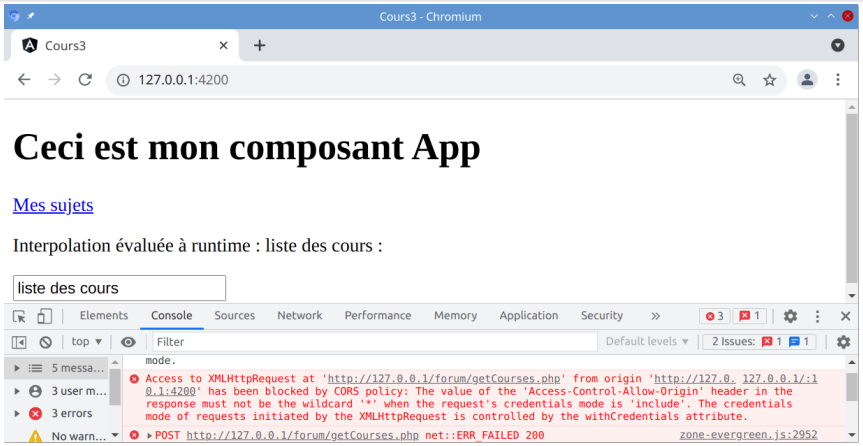
Service HTTP : 2 HttpClient avec méthode POST



Service HTTP : 3 méthode POST et session



Service HTTP : 4 problème et solution



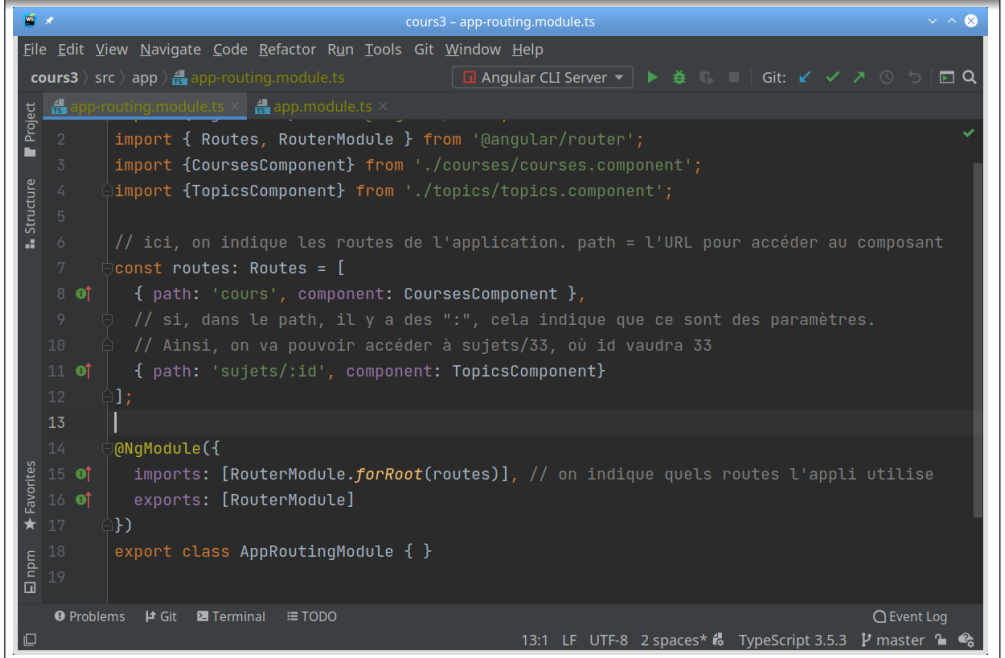
▶ **Contre-mesures :** dans getCourses.php :

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

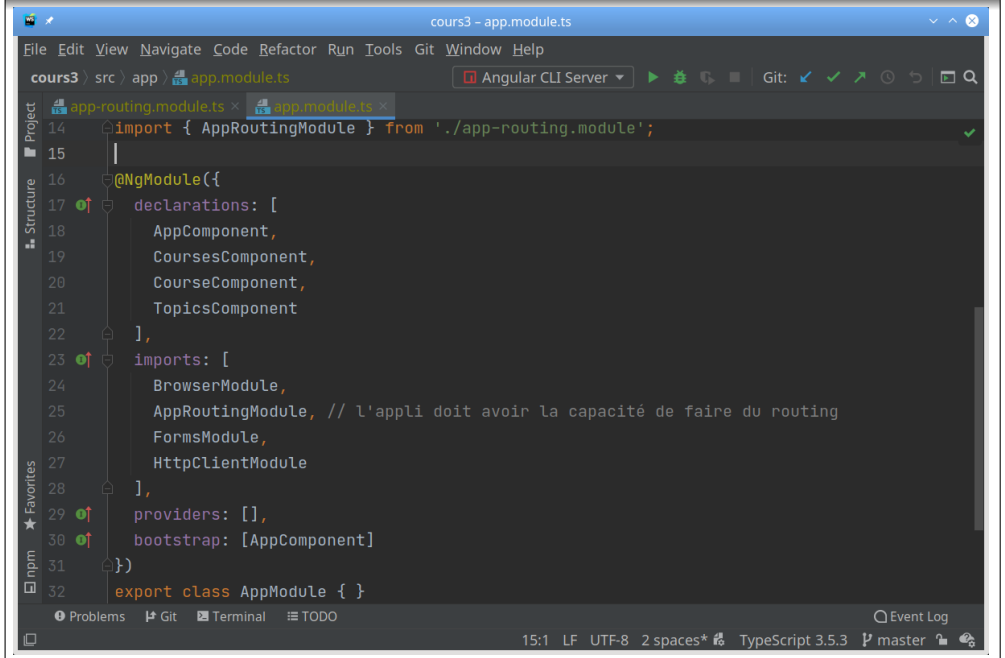

Service HTTP en résumé

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
   $\Longrightarrow$ 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`

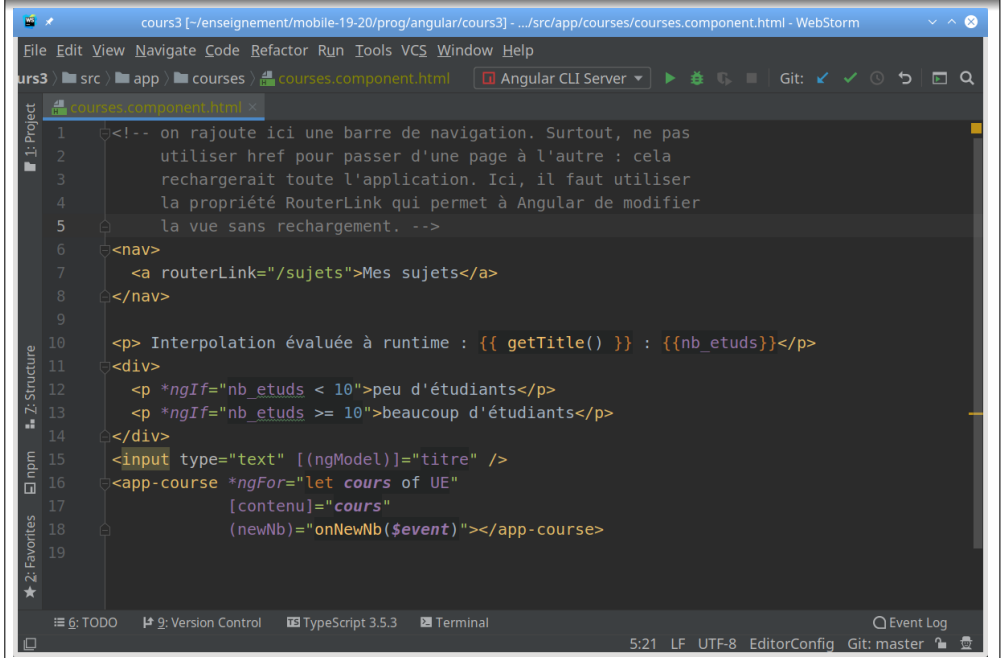
Navigation : les routes dans app-routing.module.ts



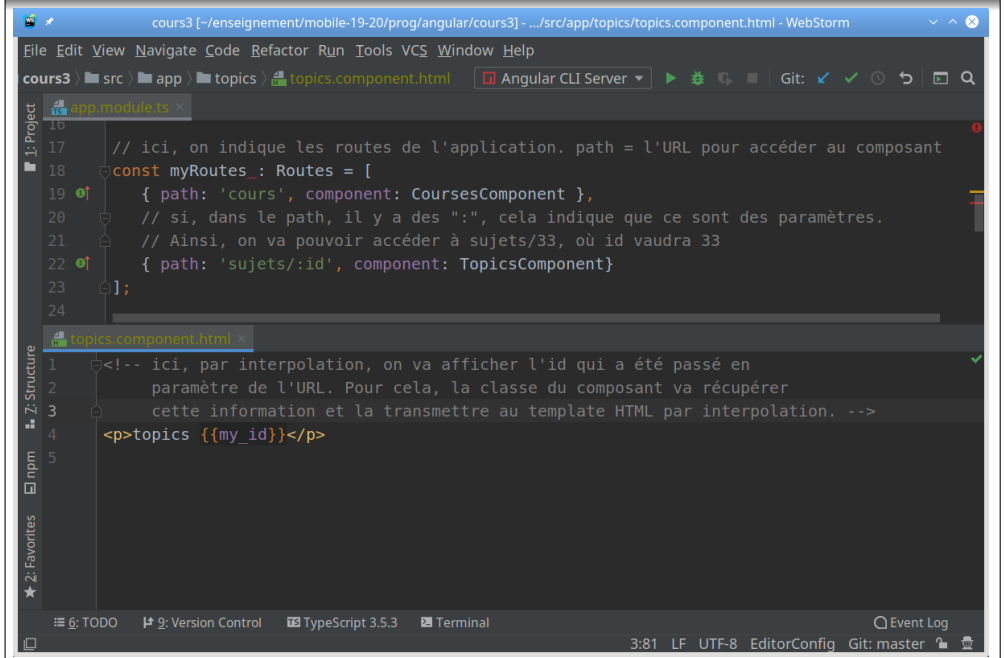
Navigation : AppRoutingModule dans app.module.ts



RouterLink : le lien de navigation



Les liens paramétrés (1/2)



Les liens paramétrés (2/2)