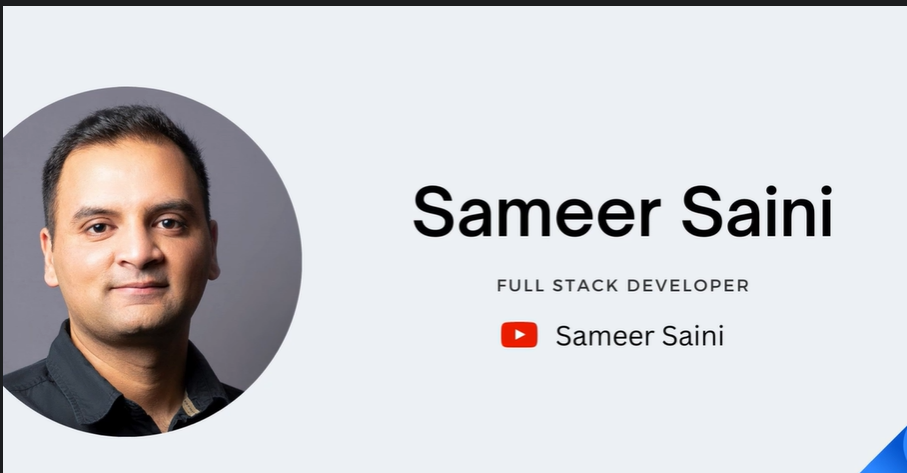
# Cours UdeMy : ANGULAR and ASP.NET Core REST API - Real World Application

https://mern.udemy.com/course/real-world-app-angular-aspnet-core-web-api-and-sql/learn/lecture/38606284#overview



Create Full Stack Web Applications with Angular, ASP.NET Core Web API, Entity Framework Core, C# REST API, JWT Token



# Le code source du cours se trouve dans :

<https://github.com/sameer8saini/CodePulse.API>

<https://github.com/sameer8saini/CodePulse.UI>

# Créer la structure de la solution et des projets :

1. Créer le dossier parent : mkdir CodePulse
2. cd CodePulse
3. Créez la solution : **dotnet new sln --name CodePulse**
4. Créez le sous-dossier et le projet Web API :
5. mkdir CodePulse.API
6. cd CodePulse.API
7. **dotnet new webapi**
8. cd ..
9. Ajoutez le projet API à la solution :

dotnet sln CodePulse.sln add CodePulse.API/CodePulse.API.csproj

1. Créer l’application Angular (NOTE le Prof utilise la v16.0.2 dans le cours):
2. ng new CodePulse.UI --routing --style css
3. cd ..
4. La structure finale ressemblera à ceci :
5. Copy
6. CodePulse/
7. ├── CodePulse.sln
8. ├── CodePulse.API/
9. │ └── CodePulse.API.csproj
10. │ └── [autres fichiers du projet API]
11. └── CodePulse.UI/
12. └── [fichiers du projet Angular]

Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? (y/N)

Pour Angular, je vous conseille de répondre "y" (oui) à cette question.

Le Server-Side Rendering (SSR) et Static Site Generation (SSG) offrent plusieurs avantages :

- Meilleure performance au premier chargement

- Meilleur référencement (SEO)

- Support des réseaux sociaux (aperçus de partage)

- Performance améliorée sur les appareils mobiles

Ces fonctionnalités font partie d'Angular Universal et sont particulièrement utiles si :

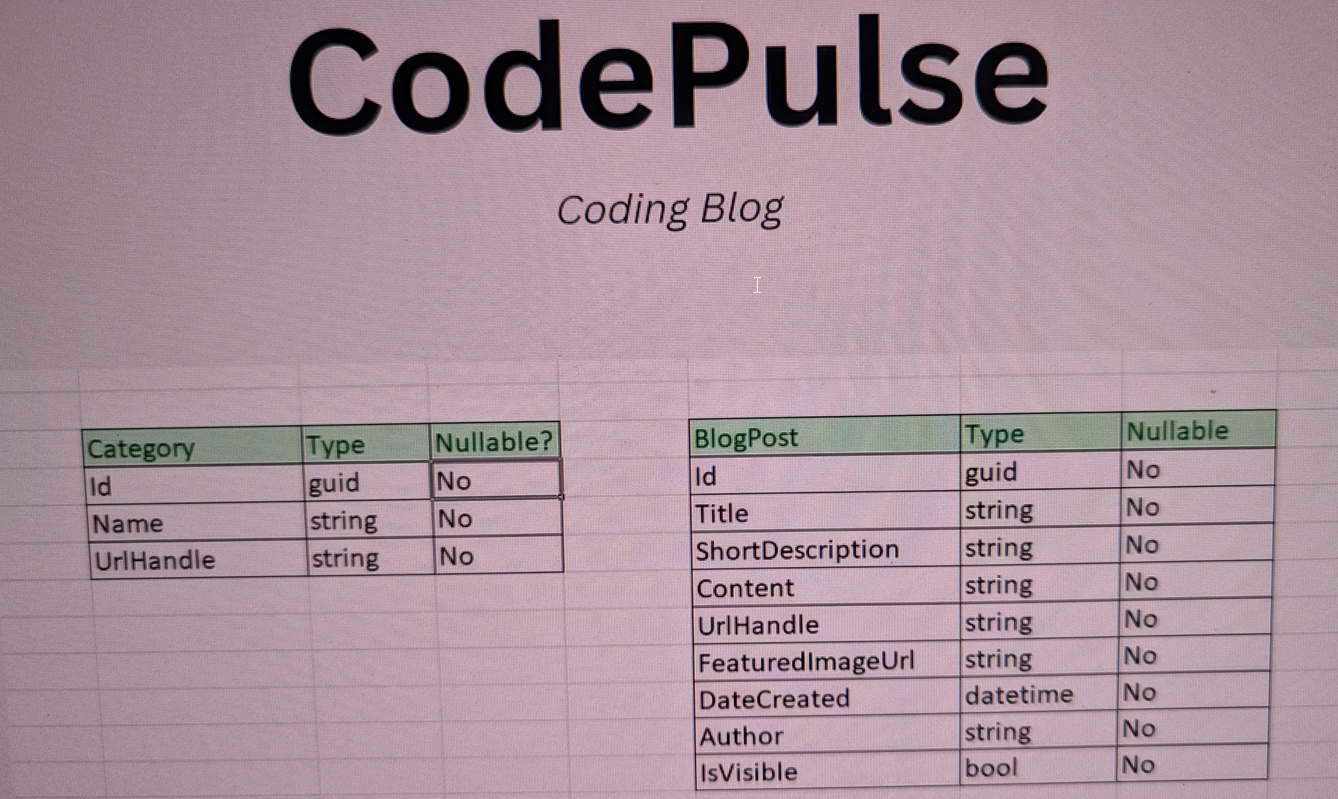
- Votre site doit être bien référencé

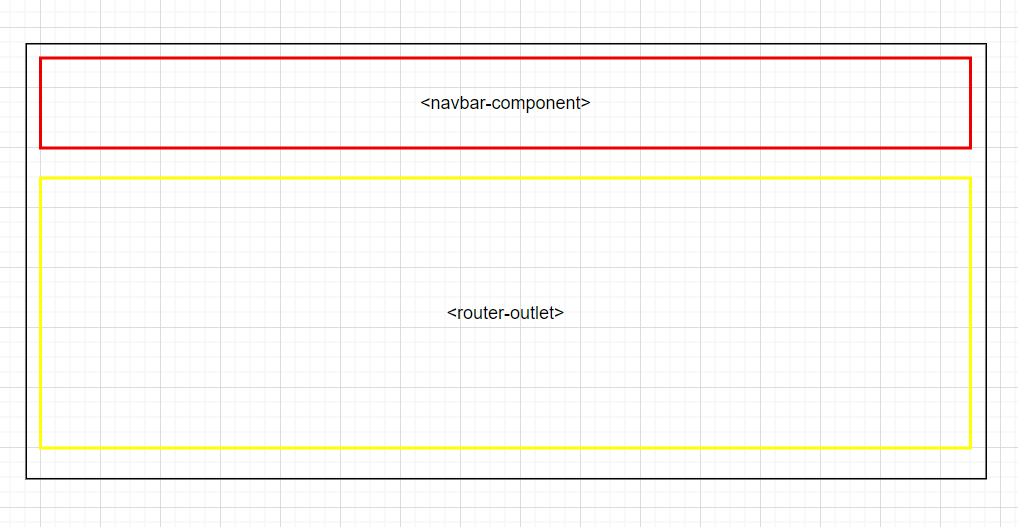
- Vous visez une performance optimale

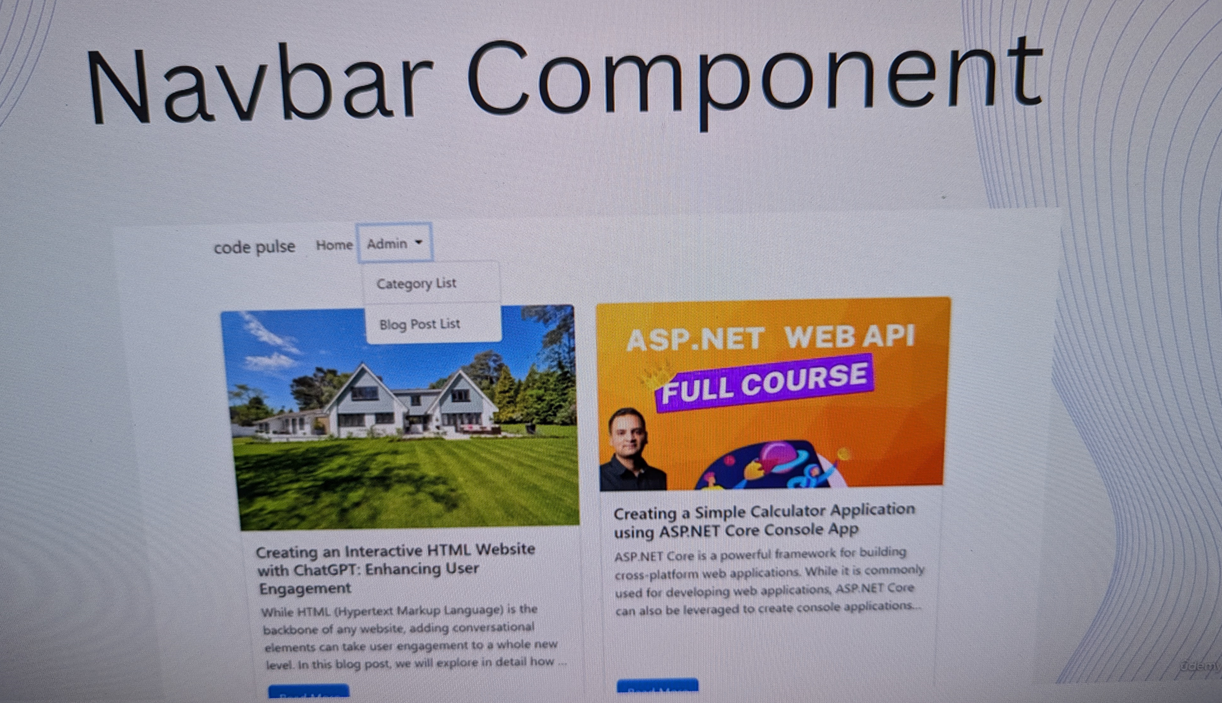
- Vous avez du contenu dynamique qui doit être pré-rendu

Donc pour ce projet, tapez "y" et appuyez sur Entrée.

CodePulse Blog







[**Generate Component**](https://angular.dev/cli/generate)

The ng generate component command in Angular CLI is used to create a new component in an Angular application. This command generates the necessary files and boilerplate code for the component.

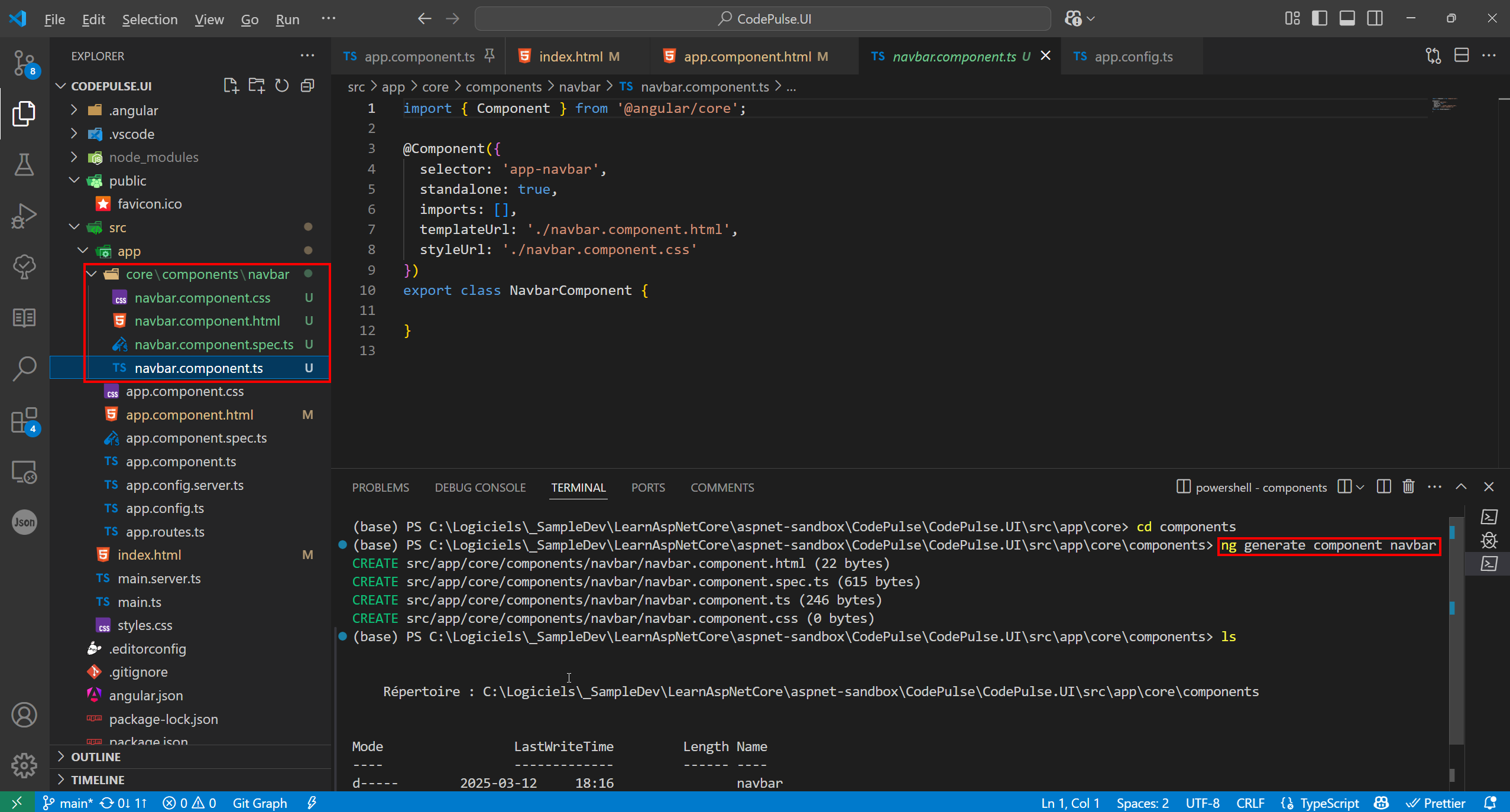
**Example**

ng generate **component** <componentName> ou ng g c <componentName>

i.e ng generate component product

This command will create a new component named product with the following files:

* product.component.html
* product.component.spec.ts
* product.component.ts
* product.component.scss



[Generate Service](https://angular.dev/cli/generate/service)

ng generate service[name][options] ou ng generate s[name][options]

Creates a new service in your project. Services are used to encapsulate reusable logic, such as data access, API calls, or utility functions. This schematic simplifies the process of generating a new service with the necessary files and boilerplate code.

On this page

* [Arguments](https://angular.dev/cli/generate/service#arguments)
* [Options](https://angular.dev/cli/generate/service#options)

arrow\_upward\_altBack to the top

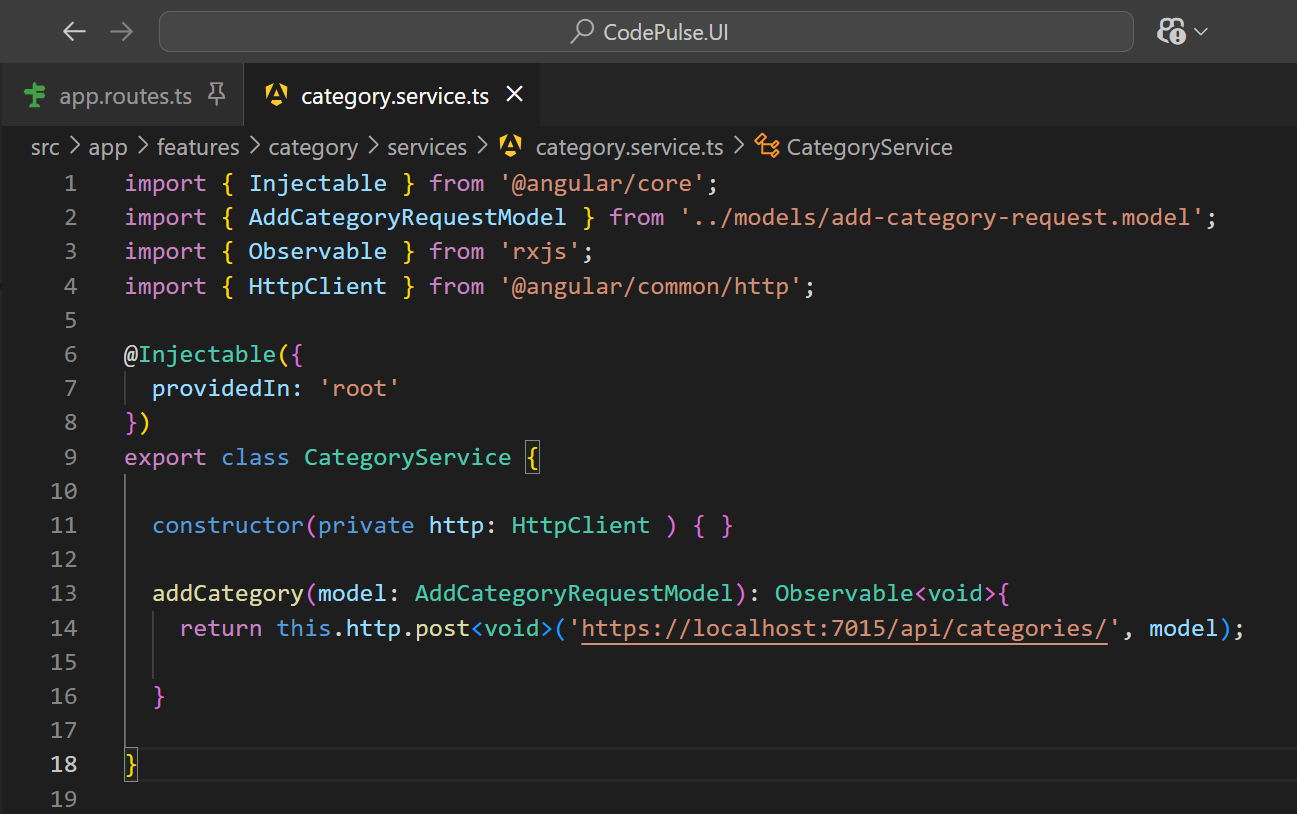
[Arguments](https://angular.dev/cli/generate/service#arguments)

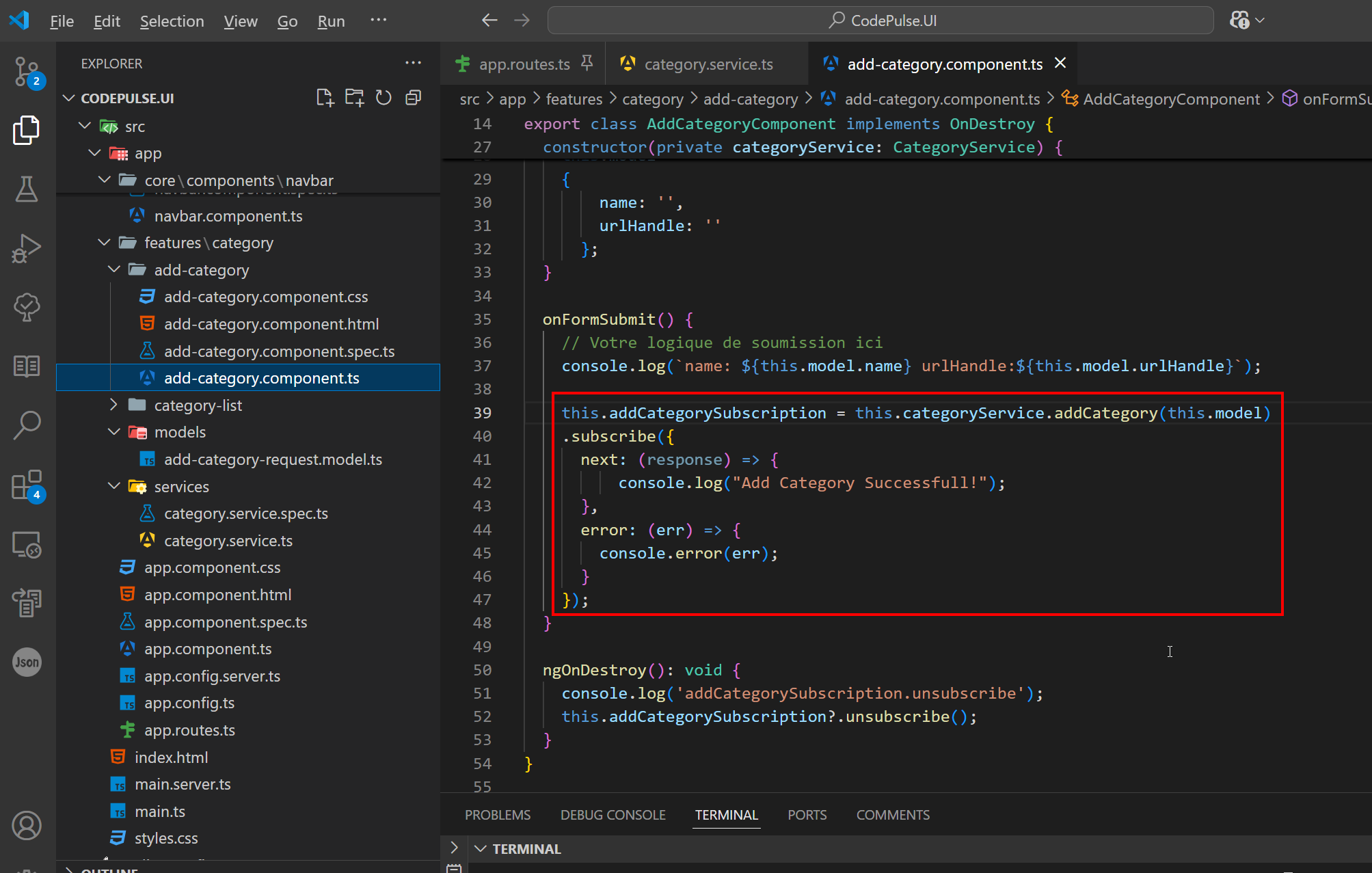
[Making HTTP requests](https://angular.dev/guide/http/making-requests)

HttpClient has methods corresponding to the different HTTP verbs used to make requests, both to load data and to apply mutations on the server. Each method returns an [RxJS Observable](https://rxjs.dev/guide/observable" \t "_blank) which, when subscribed, sends the request and then emits the results when the server responds.

**Note:** Observables created by HttpClient may be subscribed any number of times and will make a new backend request for each subscription.

Through an options object passed to the request method, various properties of the request and the returned response type can be adjusted.





[Create Environment Files](https://angular.dev/tools/cli/environments#configure-environment-specific-defaults)

ng generate environments[options]

@angular-devkit/build-angular:browser supports file replacements, an option for substituting source files before executing a build. Using this in combination with --configuration provides a mechanism for configuring environment-specific data in your application.

Start by [generating environments](https://angular.dev/cli/generate/environments) to create the src/environments/ directory and configure the project to use file replacements.

ng generate environments

check

The project's src/environments/ directory contains the base configuration file, environment.ts, which provides the default configuration for production. You can override default values for additional environments, such as development and staging, in target-specific configuration files.

For example:

my-app/src/environments

├── environment.development.ts

├── environment.staging.ts

└── environment.ts

The base file **environment**.ts, contains the default environment settings.

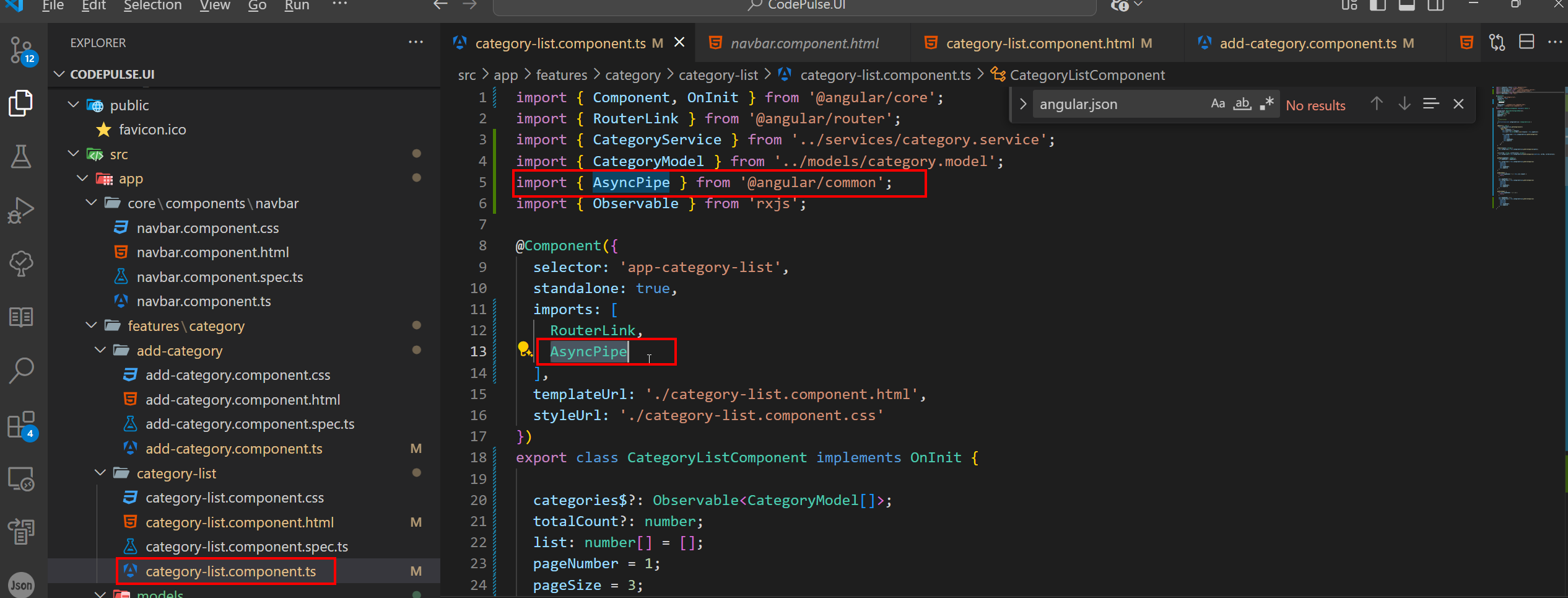
[Async Pipe](https://angular.dev/api/common/AsyncPipe)

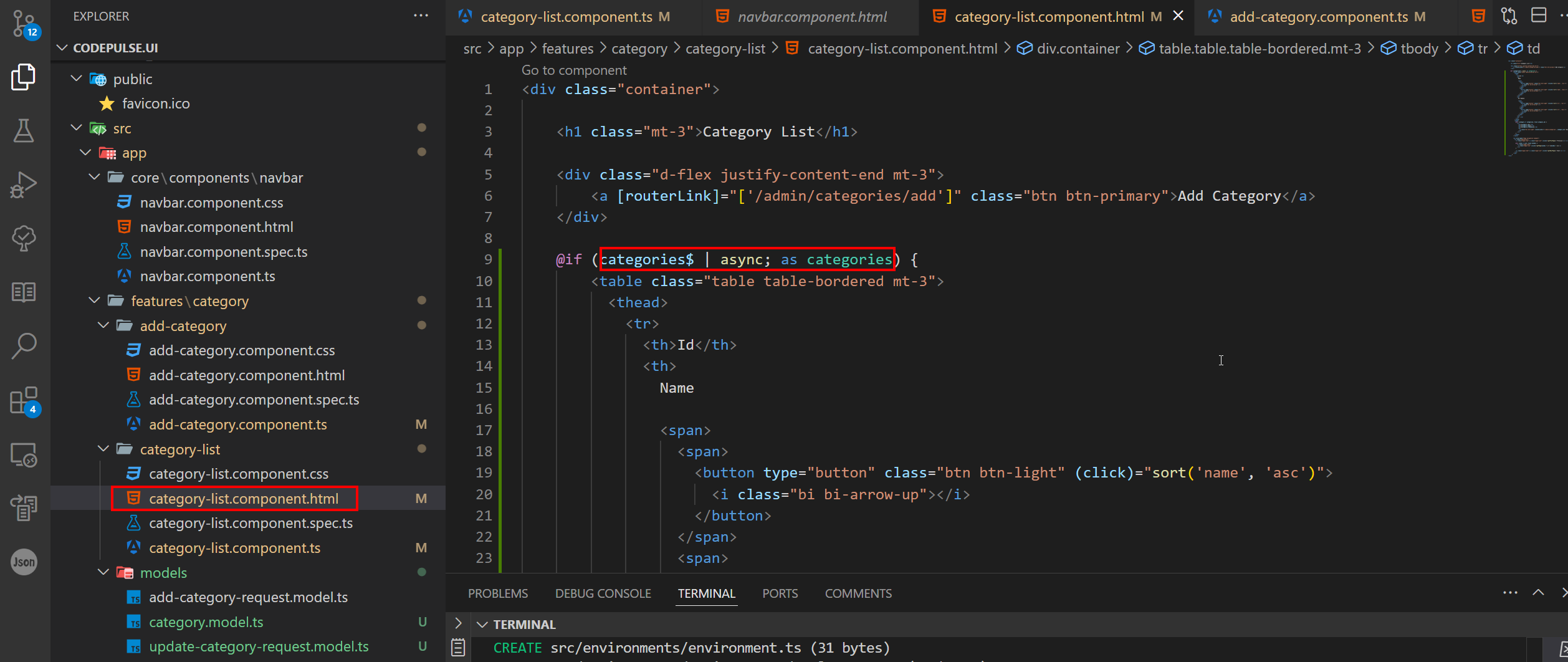
Unwraps a value from an asynchronous primitive.

[Description](https://angular.dev/api/common/AsyncPipe#description)

Unwraps a value from an asynchronous primitive.

The async pipe subscribes to an Observable or Promise and returns the latest value it has emitted. When a new value is emitted, the async pipe marks the component to be checked for changes. When the component gets destroyed, the async pipe unsubscribes automatically to avoid potential memory leaks. When the reference of the expression changes, the async pipe automatically unsubscribes from the old Observable or Promise and subscribes to the new one.





Question à Claude.AI:

**JD**

explique moi la nomenclature pour categories$ comme dans : categories$?: Observable<CategoryModel[]>;

Le symbole dollar ($) à la fin d'un nom de variable comme dans categories$ est une convention de nommage utilisée en programmation, particulièrement populaire dans l'écosystème Angular et RxJS.

Cette convention indique que la variable est un Observable, c'est-à-dire un flux de données asynchrone qui peut émettre des valeurs au fil du temps. C'est une pratique recommandée pour distinguer rapidement les Observables des autres types de variables dans votre code.

Dans votre exemple :

categories**$**?: Observable<CategoryModel[]>;

Cela signifie que :

* categories$ est un Observable
* Le ? indique que cette propriété est optionnelle (peut être undefined)
* Cet Observable va émettre des tableaux de CategoryModel
* Vous pouvez vous abonner à cet Observable ou l'utiliser avec le pipe async dans le template

Cette convention est particulièrement utile quand vous manipulez à la fois des données synchrones et asynchrones. Par exemple :

// Variable contenant des données synchrones (un tableau normal)

categories: CategoryModel[] = [];

// Observable qui émet des tableaux de catégories de façon asynchrone

categories**$**: **Observable**<CategoryModel[]>;

Le suffixe $ permet de distinguer immédiatement ces deux types de variables dans votre code, ce qui améliore la lisibilité et la maintenabilité.

[Routing et angular/router](https://angular.dev/guide/routing)

Angular Routing

Routing helps you change what the user sees in a single-page app.

In a single-page app, you change what the user sees by showing or hiding portions of the display that correspond to particular components, rather than going out to the server to get a new page.

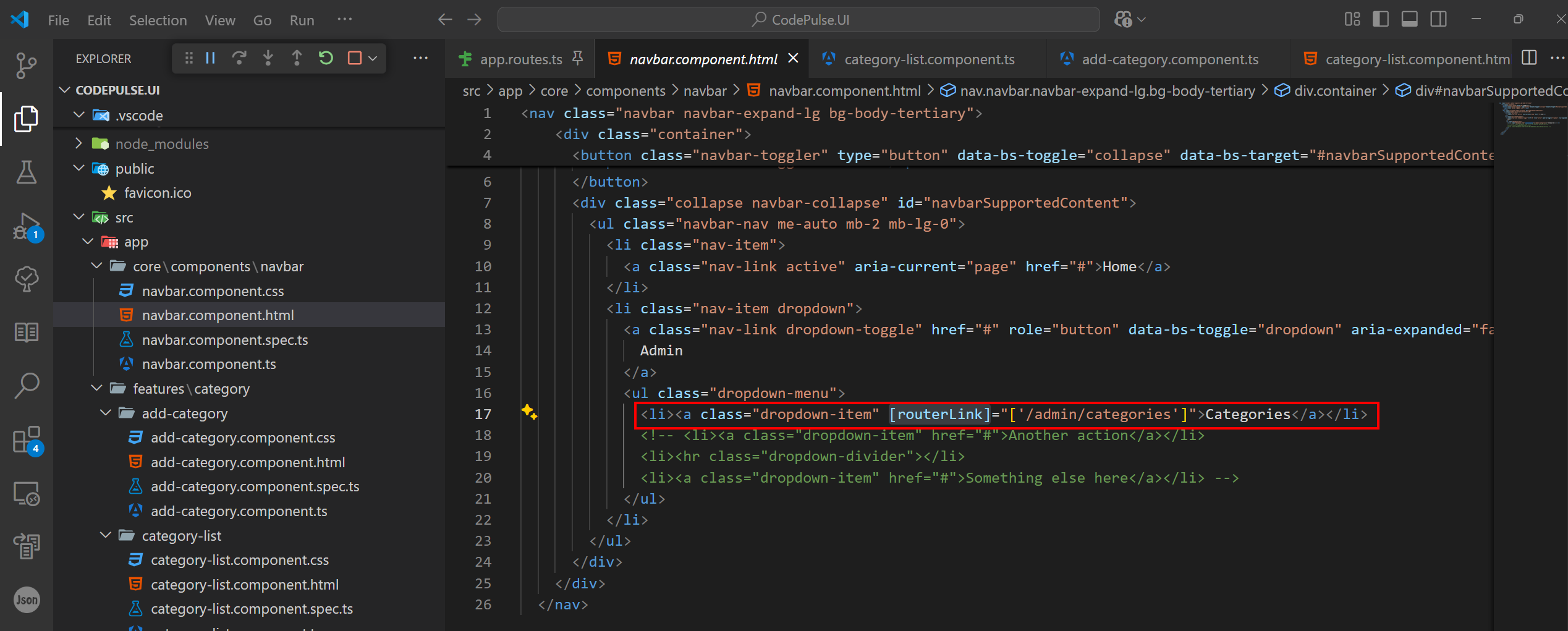
As users perform application tasks, they need to move between the different views that you have defined.

To handle the navigation from one view to the next, you use the Angular Router. The Router enables navigation by interpreting a browser URL as an instruction to change the view.

Définition des routes

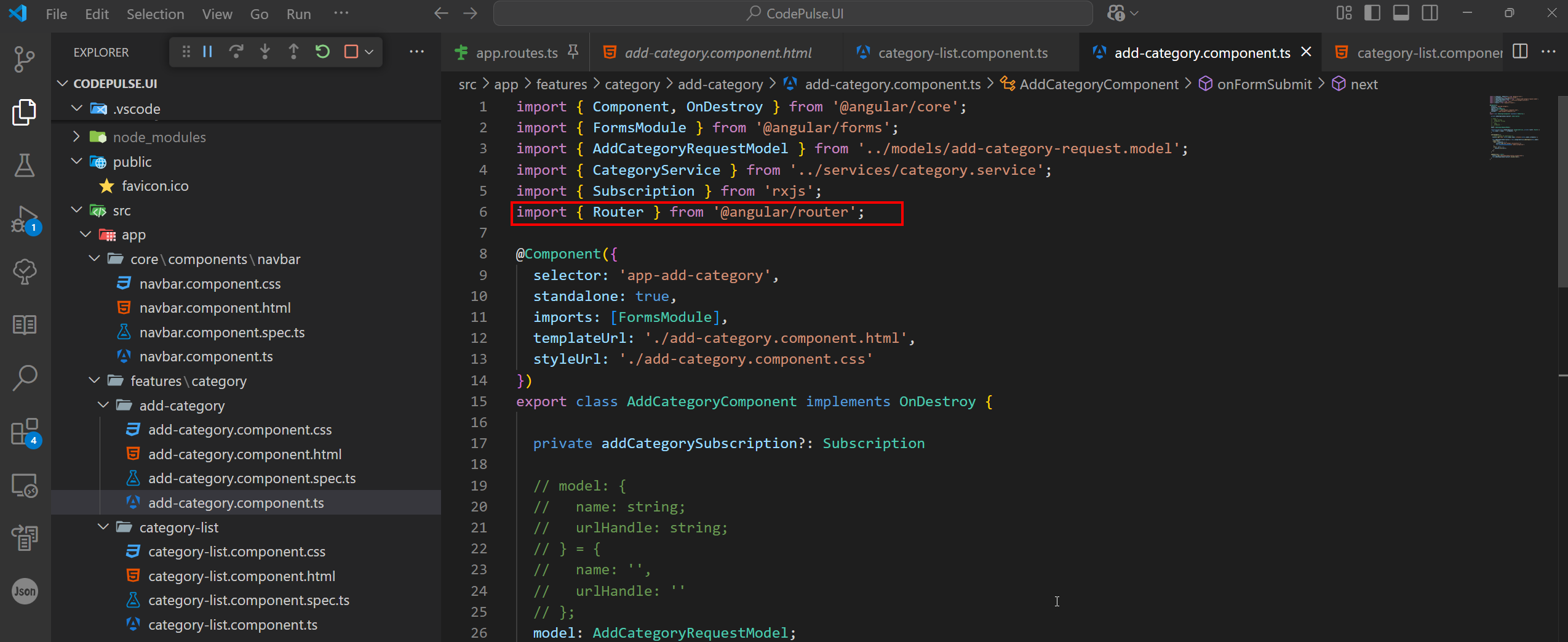


Ce qui permet de naviguer dans les page HTML avec [routerLink]



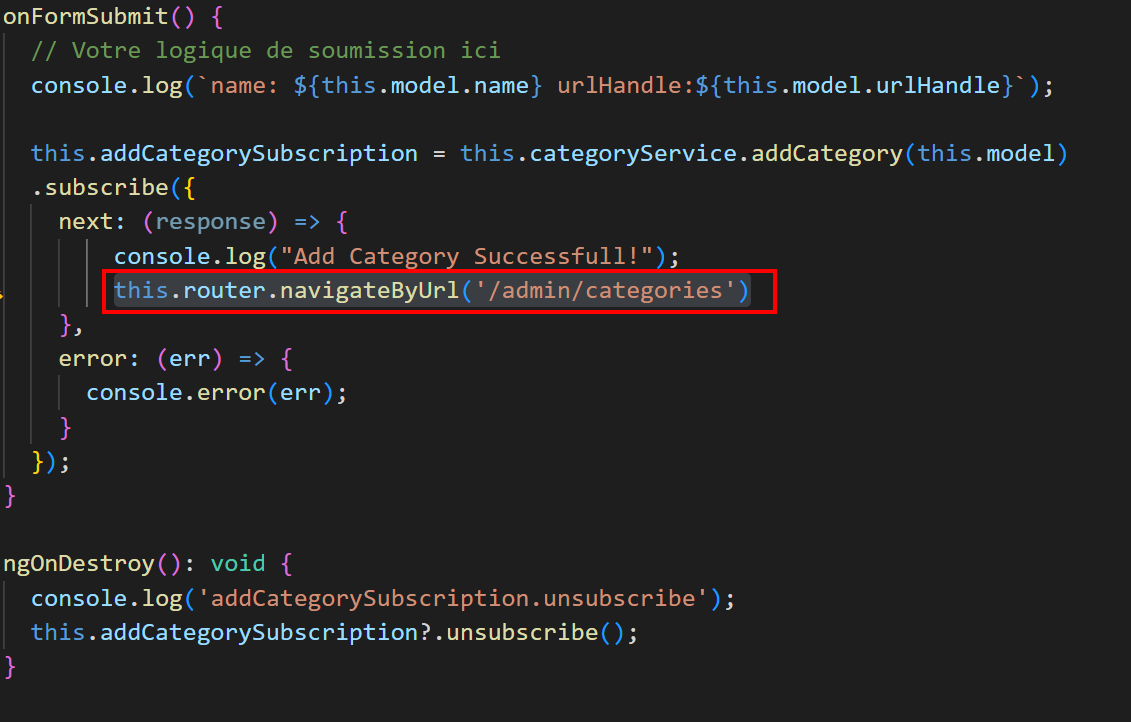
Dans le code TypeScript d’un component, il est possible de naviguer avec router.navigateByUrl

Il faut d’Abord importer Router :

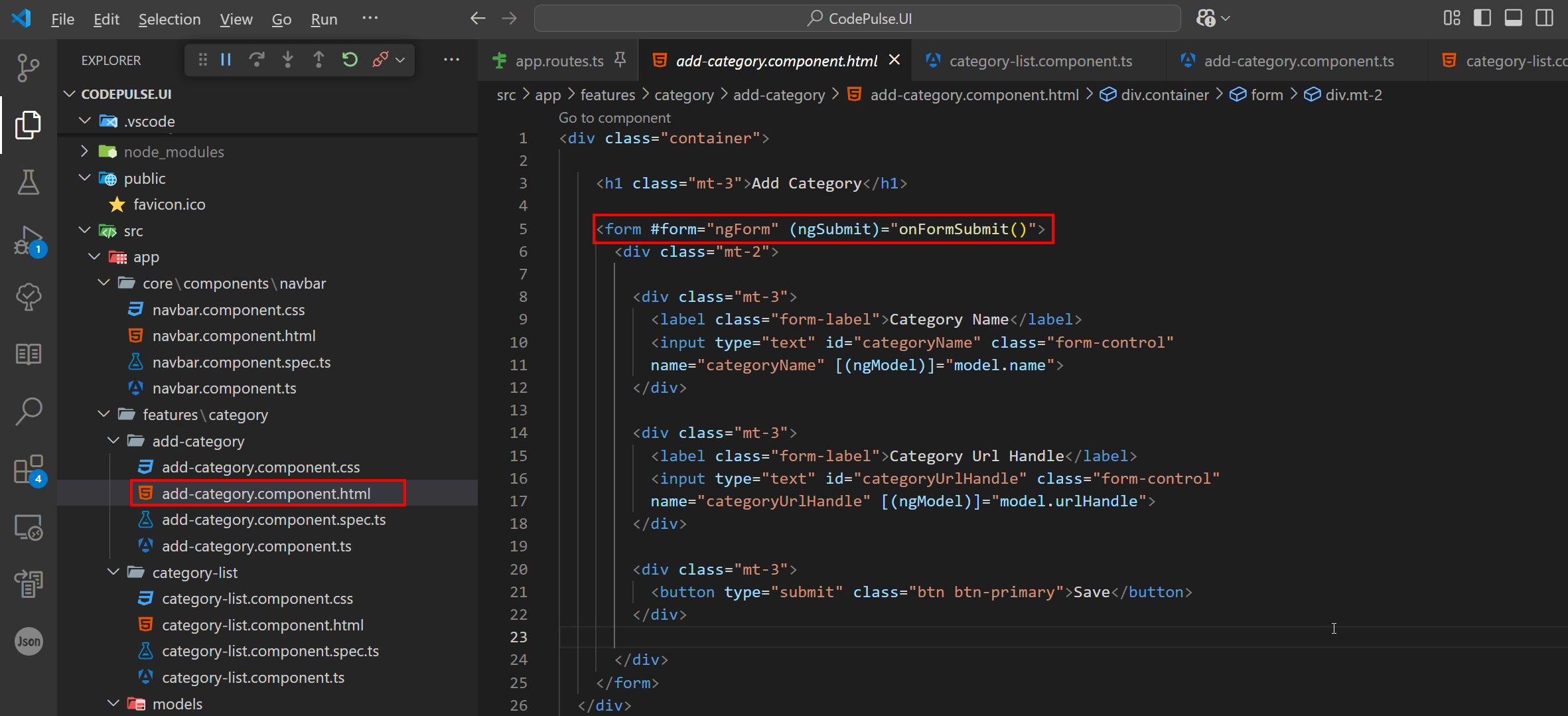


Dans l’exemple suivant, après l’ajout d’une nouvelle Category (categoryService.addCategory

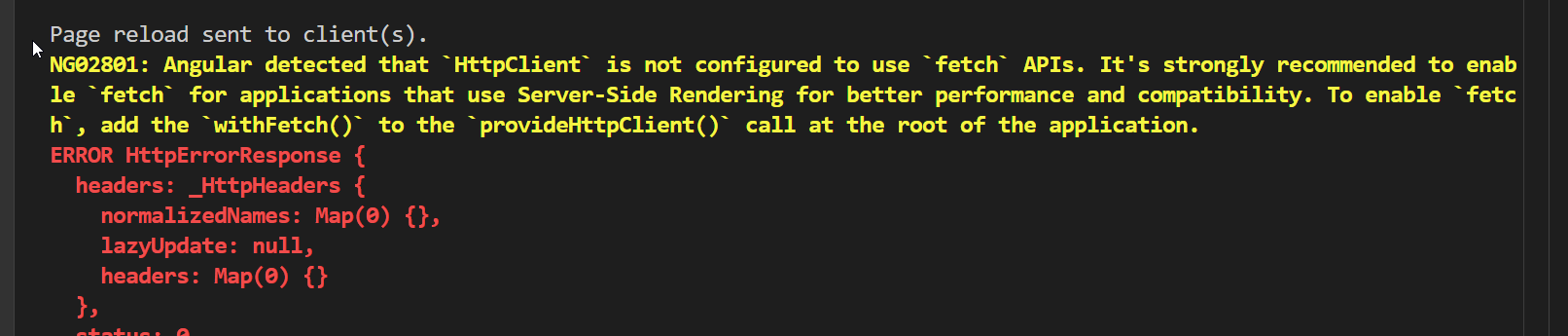
) on va rediriger avec this.router.navigateByUrl('/admin/categories')

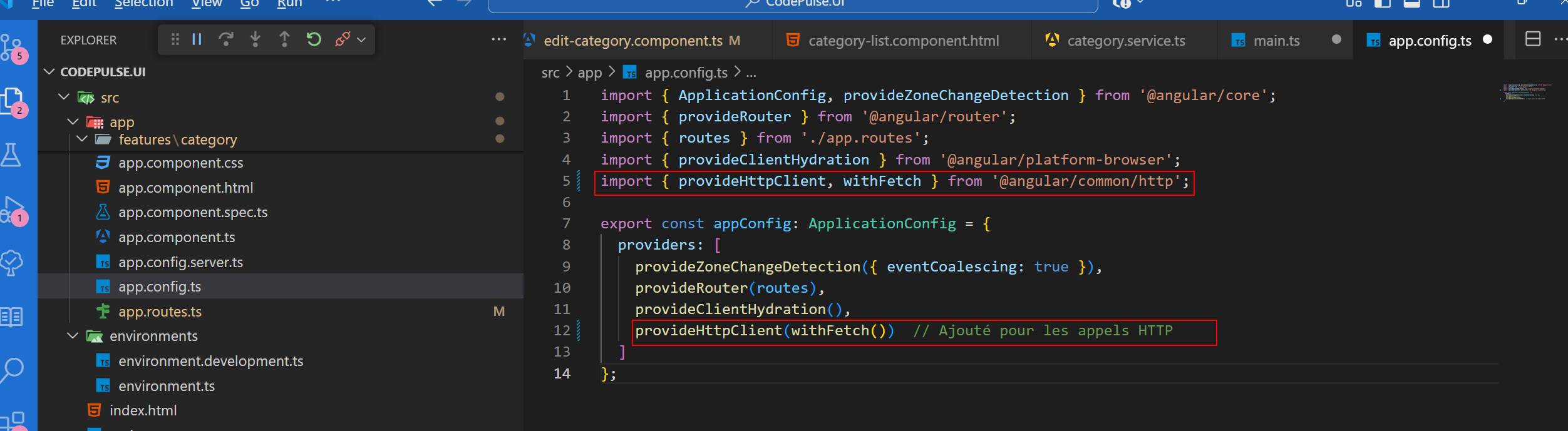


Dans le HTML :



Configuration des Appels avec HttpClient





Dans ton cas, l’application utilise bootstrapApplication(AppComponent, appConfig), donc tu dois modifier **app.config.ts** (ou là où appConfig est défini).

**Ajout de withFetch() dans app.config.ts**

Ouvre **app.config.ts** et modifie la configuration pour inclure provideHttpClient(withFetch()) :



Ensuite, assure-toi que ton **main.ts** reste comme ceci :



**🚀 Étapes après la modification**

1. Sauvegarde les fichiers.
2. Redémarre ton application avec ng serve.
3. Vérifie si l’erreur **NG02801** disparaît.

Dis-moi si tu rencontres d'autres soucis ! 😊

[@if](https://angular.dev/api/core/@if)

The @if block conditionally displays its content when its condition expression is truthy.

**Syntax**

@if (a > b) {

{{a}} is greater than {{b}}

} @else if (b > a) {

{{a}} is less than {{b}}

} @else {

{{a}} is equal to {{b}}

}

DataBinding

[Two-way binding](https://angular.dev/guide/templates/two-way-binding)

**Two way binding** is a shorthand to simultaneously bind a value into an element, while also giving that element the ability to propagate changes back through this binding.

On this page

* [Syntax](https://angular.dev/guide/templates/two-way-binding#syntax)
* [Two-way binding with form controls](https://angular.dev/guide/templates/two-way-binding#two-way-binding-with-form-controls)
* [Two-way binding between components](https://angular.dev/guide/templates/two-way-binding#two-way-binding-between-components)
* [Enabling two-way binding between components](https://angular.dev/guide/templates/two-way-binding#enabling-two-way-binding-between-components)

arrow\_upward\_altBack to the top

[Syntax](https://angular.dev/guide/templates/two-way-binding#syntax)

The syntax for two-way binding is a combination of square brackets and parentheses, [()]. It combines the syntax from property binding, [], and the syntax from event binding, (). The Angular community informally refers to this syntax as "banana-in-a-box".

**Exemple de oneway vs two way binding dans le html**

Copilot

Bien sûr ! Voici un exemple simple pour illustrer la différence entre le **one-way binding** et le **two-way binding** dans Angular.

**One-Way Binding**

Le one-way binding permet de lier les données du modèle à la vue, mais pas l'inverse. Voici un exemple :

<!-- app.component.html -->

<p>{{ message }}</p>

// app.component.ts

export class AppComponent {

message = 'Bonjour, Angular!';

}

Dans cet exemple, la valeur de message dans le modèle est affichée dans la vue, mais toute modification dans la vue ne changera pas le modèle.

**Two-Way Binding**

Le two-way binding permet de lier les données du modèle à la vue et vice versa. Voici un exemple :

<!-- app.component.html -->

<input [(ngModel)]="message" />

<p>{{ message }}</p>

// app.component.ts

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

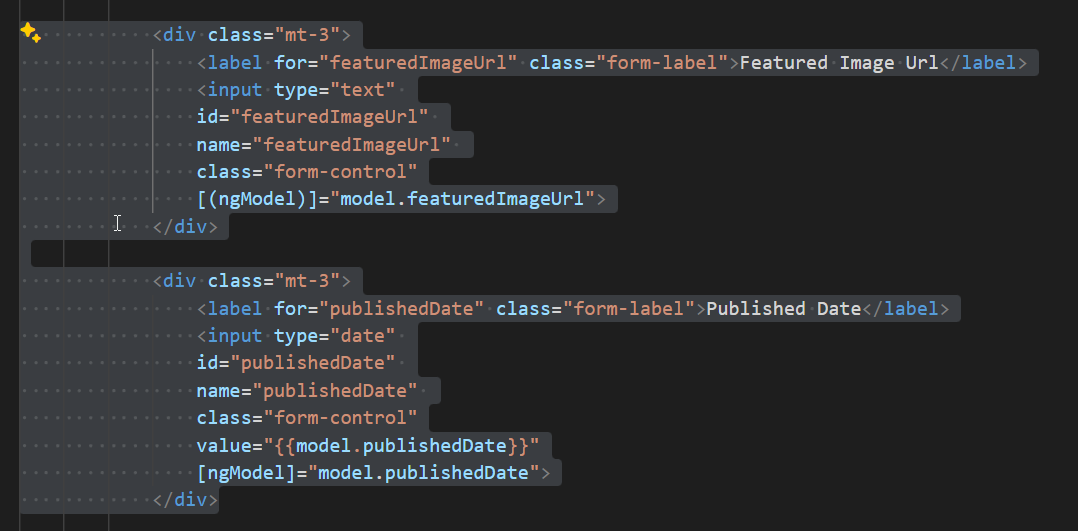
})

export class AppComponent {

message = 'Bonjour, Angular!';

}

Dans cet exemple, la valeur de message est liée à l'élément input. Toute modification dans l'input mettra à jour le modèle, et toute modification dans le modèle mettra à jour l'input.



Bien sûr ! Voici une explication de ce code HTML avec Angular :

**Première section : Featured Image Url**

<div class="mt-3">

<label for="featuredImageUrl" class="form-label">Featured Image Url</label>

<input type="text"

id="featuredImageUrl"

name="featuredImageUrl"

class="form-control"

[(ngModel)]="model.featuredImageUrl">

</div>

* <div class="mt-3"> : Crée une division avec une marge supérieure de 3 unités.
* <label for="featuredImageUrl" class="form-label">Featured Image Url</label> : Crée une étiquette pour l'input avec le texte "Featured Image Url".
* <input type="text" id="featuredImageUrl" name="featuredImageUrl" class="form-control" [(ngModel)]="model.featuredImageUrl"> : Crée un champ de texte avec :
  + id="featuredImageUrl" : Identifiant unique pour l'input.
  + name="featuredImageUrl" : Nom de l'input.
  + class="form-control" : Classe Bootstrap pour le style.
  + [(ngModel)]="model.featuredImageUrl" : Two-way binding avec la propriété featuredImageUrl du modèle. Toute modification dans l'input mettra à jour le modèle et vice versa.

**Deuxième section : Published Date**

<div class="mt-3">

<label for="publishedDate" class="form-label">Published Date</label>

<input type="date"

id="publishedDate"

name="publishedDate"

class="form-control"

value="{{model.publishedDate}}"

[ngModel]="model.publishedDate">

</div>

* <div class="mt-3"> : Crée une division avec une marge supérieure de 3 unités.
* <label for="publishedDate" class="form-label">Published Date</label> : Crée une étiquette pour l'input avec le texte "Published Date".
* <input type="date" id="publishedDate" name="publishedDate" class="form-control" value="{{model.publishedDate}}" [ngModel]="model.publishedDate"> : Crée un champ de date avec :
  + id="publishedDate" : Identifiant unique pour l'input.
  + name="publishedDate" : Nom de l'input.
  + class="form-control" : Classe Bootstrap pour le style.
  + value="{{model.publishedDate}}" : One-way binding pour initialiser la valeur de l'input avec publishedDate du modèle.
  + [ngModel]="model.publishedDate" : One-way binding pour lier la valeur de l'input à publishedDate du modèle.

En résumé, le premier input utilise le two-way binding pour synchroniser les données entre le modèle et la vue, tandis que le second input utilise le one-way binding pour afficher la valeur du modèle dans la vue.

Si vous avez d'autres questions ou besoin de plus de détails, n'hésitez pas à demander ! 😊